Lezioni di Calcolo Numerico

Alberto Tibaldi

26 luglio 2019

Indice

Qualche parola di introduzione al Calcolo Numerico

Credo che possa essere appropriato iniziare queste note con alcune domande diciamo *ontologiche*.

> «Cos'è il Calcolo Numerico? Perché ci si prende la briga di dedicarvi un corso universitario?»

Per aiutarmi a rispondere a queste domande, vorrei sottoporre all'attenzione del lettore il seguente quesito: come si fa a valutare l'espressione

$$I = \int_4^7 \mathrm{e}^{-x^2}\,\mathrm{d}x \quad ? \tag{1}$$

Uno studente reduce da Analisi Matematica I, pensando al tipico problema di calcolo di integrali definiti, cercherebbe presumibilmente di scrivere la primitiva della funzione in questione, di valutarla agli estremi di integrazione, e calcolare la differenza dei due numeri che troverebbe, sfruttando il teorema fondamentale del calcolo integrale. Dopo qualche tentativo a vuoto, il malcapitato si renderebbe conto che questo non è un problema risolubile percorrendo questa strada. Tuttavia, questo non dipende da ragioni *filosofiche*, *astratte*, legate alle proprietà di regolarità di questa funzione. In effetti, per quanto possa sembrare strano, integrare una funzione è, in linea di principio, molto più *indolore* rispetto a derivarla. Infatti, l'integrale è un'operazione che *aumenta la regolarità* di una funzione, che la rende più *liscia*; al contrario, applicare l'operatore di derivazione produce funzioni *meno regolari* rispetto a quella di partenza. Si pensi per esempio all'integrale e alla derivata della funzione $|x|$: se da un lato l'integrale è liscio, poiché fa sparire il punto angoloso, la derivata presenta un punto di discontinuità, che abbatte la regolarità al punto da ridurre il dominio. Al fine di aiutare a visualizzare questo concetto, la Fig. 1 riporta $f(x) = |x|$ (curva blu), la sua derivata $f'(x)$ (curva rossa) e la sua funzione integrale $F(x)$ (curva verde). Si noti che non ha senso valutare la derivata $f'(x)$ in $x = 0$, come enfatizzato. Questa figura è ottenuta a partire dallo script MATLAB® :

Script usato per disegnare i grafici di Fig. 1.

```
clear
close all
clc

x=linspace(-5,5,100001);
f=abs(x);
fder=sign(x);
fint=sign(x).*x.^2;

figure
grid on
hold on
box on
axis equal
plot(x,f,'-',x,fder,'-',x,fint,'-','LineWidth',2)
axis([-2,2,-2,2])
xlabel('x')
legend('f(x)','f'''(x)','F(x)','Location','Best')
```

nel quale vengono disegnate le espressioni esplicite

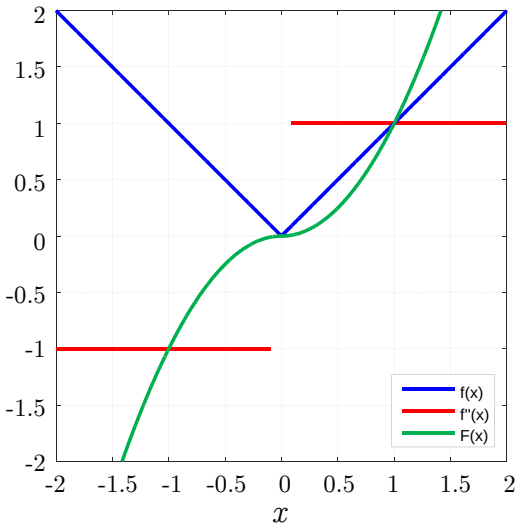

Figura 1: Grafico della funzione $f(x) = |x|$ (curva blu), della sua derivata $f'(x)$ (curva rossa) e del suo integrale $F(x)$ (curva verde), le cui espressioni sono riportate in (2). Si noti come la derivazione esalti il punto angoloso trasformandolo in una discontinuità, mentre l'integrazione d'altro canto tenda a mascherarlo facendolo degenerare in un flesso.

$$\begin{aligned} f(x) &= |x| \\ f'(x) &= \operatorname{sign} x \\ F(x) &= x^2 \operatorname{sign} x. \end{aligned} \tag{2}$$

Si ricordi che nel corso di Analisi Matematica I si parla spessissimo di funzioni derivabili, ma quasi mai di funzioni integrabili. Questo è dovuto al fatto che le funzioni integrabili costituiscono un insieme molto più vasto di quelle derivabili, e quindi la derivabilità è una condizione *meno scontata* da possedere per una funzione; in altre parole, una funzione derivabile è integrabile, ma il viceversa non è necessariamente vero. Il problema dell'esercizio (1) è di natura più, diciamo, *operativa*: non siamo capaci di scrivere, in termini di funzioni elementari (prodotti, frazioni, potenze, seni, coseni, esponenziali, logaritmi e via discorrendo) la primitiva della funzione integranda. Tuttavia, poiché la funzione è assolutamente regolare, da un punto di vista **numerico** questo problema è assolutamente standard.
Ispirati da un *leitmotiv* che riecheggia nelle aule, potremmo chiederci

> «Perché nel 2019 devo imparare a risolvere tutti questi integrali, quando me li può fare un computer?!?»

Questa domanda sarà un po' tracotante, ma può aiutarci a spiegare in maniera un po' folkloristica quale sarà lo scopo di questo corso. L'obiettivo del Calcolo Numerico è far *digerire* un problema matematico a un computer, riducendolo ad algoritmi, a passi elementari. Per quanto un computer possa essere svelto a fare conti, ovvero operazioni aritmetiche elementari, non è certamente uno strumento in grado di *capire* un problema e risolverlo. Per questo motivo, saremo noi a *digerire* alcuni problemi al posto del computer, che svolgerà un ruolo di supporto nei nostri confronti, risolvendo velocemente le operazioni in cui è senz'altro più bravo.

Vorrei approfittare di questa introduzione per augurare agli eventuali lettori di queste note una buona lettura, sperando che possa in qualche modo stuzzicare il loro interesse verso questa materia.

Capitolo 1

Aritmetica, errori

Introduzione

Quando ero studente, non riuscivo a prendere molto sul serio la parte di corso relativa all'aritmetica del calcolatore. Se infatti da un lato si trattava di un argomento un po' noioso, l'ambiente che mi circondava sembrava suggerire che fosse anche piuttosto inutile. Pensando ai vari esami di Fisica o Ingegneria, infatti, il testo chiedeva di riportare risultati *con due-tre cifre dopo la virgola*. Per vari anni, questo fatto non sembrava cambiare: aldilà di esperimenti di interesse puramente accademico (forse al limite della speculazione), nella maggior parte dei casi risultati accurati *alla terza cifra decimale* erano più che soddisfacenti. Alla luce di questo, ho avuto la tentazione di chiedermi

> «Ma che senso ha farsi tanti problemi su quante cifre il computer usi per fare i conti, se tanto a me interessano solo le prime due-tre?!?»

Purtroppo, questa tentazione contiene un gigantesco errore di fondo. Generalmente, un problema fisico/ingegneristico richiede uno *svolgimento* composto da diversi calcoli, del cui *risultato* ci interessano solo alcune cifre. La vera domanda da porsi dunque è:

> «Chi ci dice che, per avere un *risultato* accurato alla terza cifra dopo la virgola, sia sufficiente *svolgere i vari passaggi* con la stessa accuratezza?»

La risposta è: **nessuno, perché non è così.**

Credo che una persona debba sbattere -violentemente- su un problema prima di riuscire a riconoscerlo come tale. Per questo, prima di passare alle questioni *serie*, vorrei raccontare una mia esperienza risalente al luglio 2016. Talvolta, capita di dover utilizzare un diodo a semiconduttore con correnti molto ridotte. Questo per esempio può essere il caso di un fotorivelatore, un dispositivo elettronico atto a *misurare la luce* trasformandola in una corrente; la corrente *in condizioni di buio* del dispositivo deve essere molto piccola, in modo da rendere quella *in presenza di luce* più semplice da percepire per il dispositivo. In questo contesto, il mio obiettivo era sviluppare un software in grado di simulare un dispositivo di questo tipo, in presenza di correnti molto basse. Al termine del lavoro, il risultato fu la curva blu di Fig. 1.1. Dopo aver perso **settimane** cercando un purtroppo inesistente errore di programmazione, mi resi conto che il problema poteva essere legato all'accuratezza con cui il computer *svolgeva i passaggi intermedi* finalizzati a ottenere il risultato. Nella stragrande maggioranza dei casi, la doppia precisione (tipica delle architetture a 64 bit, che corrisponde a circa 16 cifre decimali significative) è più che sufficiente per ogni genere di calcolo; tuttavia, esistono situazioni in cui è necessario lavorare con un'aritmetica ancora più precisa. Capito questo e prese le opportune contromisure, in particolare *aumentando l'accuratezza dei calcoli intermedi*, il problema è stato completamente risolto, permettendomi di ottenere la curva rossa tratteggiata di Fig. 1.1, assolutamente realistica e ragionevole in quanto priva di rumore.

Concludo questo breve racconto di piccole[1] disgrazie, sperando che possa servire da monito al lettore, o quantomeno stimolare un po' di curiosità nei confronti delle note che seguiranno nel resto del capitolo.

[1] almeno, rispetto alle tragedie descritte in http://ta.twi.tudelft.nl/users/vuik/wi211/disasters.html

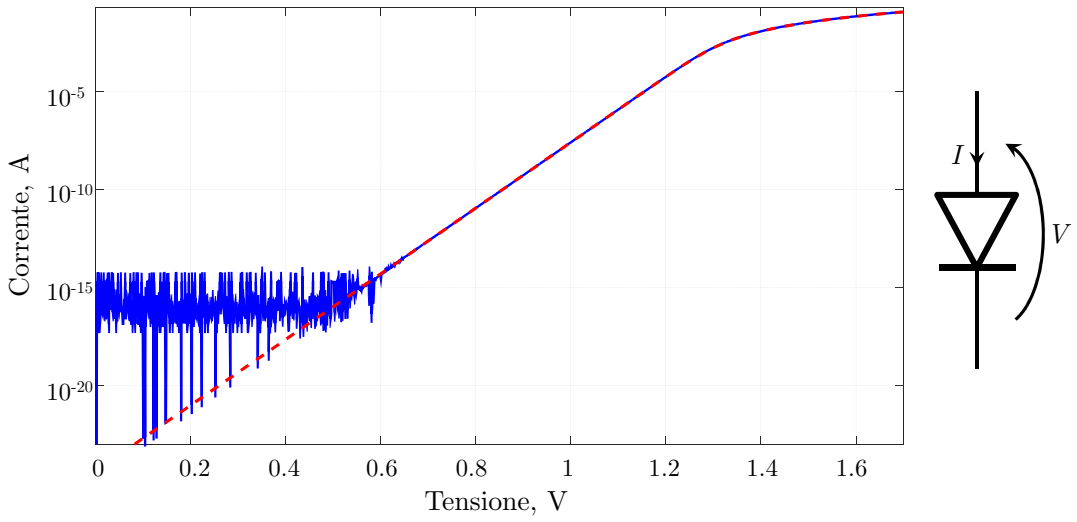

Figura 1.1: Caratteristica corrente-tensione $I(V)$ di un diodo a semiconduttore in scala logaritmica. Le curve blu e rossa tratteggiata sono state ottenute con simulatori programmati per operare a *doppia* e *quadrupla precisione macchina*, rispettivamente.

1.1 Rappresentazione floating-point

Il primo argomento che affronteremo riguarda la **rappresentazione floating-point**, detta anche **rappresentazione a virgola mobile**[2]. Il nome *rappresentazione* deve far pensare a un *modo di scrivere qualcosa*. In particolare, questa rappresentazione è finalizzata ad avvicinarsi al modo di operare di un computer. Un numero reale $a \in \mathbb{R}$ si può scrivere, nella notazione floating-point, come

$$a = (-1)^s \, p \, N^q, \quad N \in \mathbb{N}, \quad p \in \mathbb{R}, \quad p \geq 0, \quad q \in \mathbb{Z}, \quad s \in \{0, 1\} \tag{1.1}$$

ovvero, N è un numero naturale (intero positivo), p è reale positivo, q è intero, e s può assumere i valori 0 o 1. Il numero N è detto **base** del sistema di numerazione. Per questioni anatomiche[3], gli esseri umani sono soliti fare i conti con $N = 10$, ovvero con il cosiddetto **sistema decimale**. Diversamente, i computer effettuano conti utilizzando il **sistema binario**, ovvero $N = 2$, per via della semplicità nel trattare una logica contenente solo due possibili valori; poi, il risultato delle operazioni ci viene presentato in sistema decimale tramite un *cambio di base* effettuato a posteriori del calcolo.

Per come è stata definita in (1.1), la rappresentazione **non è univoca**, ovvero, a parità di sistema di numerazione scelto (e quindi di base), è possibile rappresentare lo stesso numero reale a utilizzando diversi valori di p e q. Per esempio, volendo scrivere il numero 125.54, questo può essere scritto tanto come 1.2554×10^2 ($p = 1.2554$, $q = 2$), quanto come 12554×10^{-2} ($p = 12554$, $q = -2$). **Si noti e si sappia che da adesso in poi, in questo testo, si farà uso della notazione americana, per cui la demarcazione tra interi e decimali viene indicata con un punto**. Questa scelta è legata al fatto che il codice MATLAB®, nonché la stragrande maggior parte dei linguaggi di programmazione, fa uso di questa convenzione.

Al fine di rendere univoca la rappresentazione, ovvero di poter scrivere il numero reale a in un **unico** modo, si deve fissare una convenzione, un vincolo; in particolare, la nostra scelta sarà di avere sempre e comunque

$$N^{-1} \leq p < 1, \tag{1.2}$$

ovvero, la prima cifra di p è sempre diversa da zero, ma allo stesso tempo p deve essere minore di 1. Assieme a (1.1), il vincolo (1.2) permette di definire la **rappresentazione floating-point normalizzata**. Fissata la convenzione (1.2), il numero p viene detto **mantissa**, mentre q viene detto **esponente** o **caratteristica**; infine s è il **segno** del numero (poiché se $s = 0$ il segno del numero è positivo, se $s = 1$

[2] nella letteratura inglese/americana, si utilizza *point* per indicare il simbolo di separazione tra unità e decimali, poiché in questi contesti viene utilizzato un punto, mentre la virgola (facoltativamente) viene utilizzata per indicare le migliaia. Al contrario, nella letteratura italiana, la virgola indica l'inizio dei decimali, e (facoltativamente) il punto indica le migliaia. Per esempio, volendo scrivere *duemilacinquecentoventicinque virgola cinquecentosessantasette* in inglese e italiano, si avrebbero, rispettivamente, 2,525.567 e 2.525,567

[3] le mani, strumento di calcolo di base, hanno dieci dita ;-)

sarà negativo). Disporre di questa rappresentazione significa che il numero reale a è **univocamente** determinato, nella base N, dalla conoscenza dei tre numeri s, p e q. Tuttavia,

> «Per quale motivo stiamo decidendo di complicarci la vita in questo modo? Cioè, invece di dover lavorare con un singolo numero, stiamo scegliendo di lavorare con tre numeri!»

Per rispondere a questa domanda, cerchiamo di ricordare prima di tutto qual è il nostro scopo: **lavorare con un computer**. Considerando un generico numero reale a, per esempio π o $\sqrt{2}$, questo andrebbe scritto, *esattamente*, tenendo conto di infinite cifre. Volendo lavorare con infinite cifre, sarebbe necessario, a un certo punto, memorizzarlo, e per far questo servirebbe *infinita memoria*, ancora prima di iniziare ad agire su di esso: non avrebbe molto senso. In questa direzione, la rappresentazione floating-point nasce da un compromesso tra voler descrivere un numero tenendo conto di una certa accuratezza, ovvero *usando una certa quantità di cifre*, e non volersi precludere la possibilità di trattare numeri molto grandi o molto piccoli. Immaginiamo per esempio di dover fare dei conti su un condensatore. L'unità di misura della capacità di un condensatore è il farad (F) ma, in microelettronica, valori tipici sono dell'ordine dei picofarad, quindi 10^{-12} F. Avrebbe senso scrivere 3.839 pF come 0.000000000003839 F, ovvero, salvare undici zeri dopo la virgola e poi le cifre utili, o salvare *esclusivamente* le cifre *utili* e *quanti zeri ci sono*? Per esempio, questo stesso numero, in notazione floating-point normalizzata, si potrebbe scrivere come

$$0.3839 \times 10^{-11},$$

ovvero, $s = 0$, $p = 0.3839$, $q = -11$. La stessa situazione potrebbe verificarsi in chimica, volendo quantificare il numero di molecole in una mole (6.023×10^{23}): dovremmo scrivere (e memorizzare) venti zeri?! Disporre dell'esponente q, quindi, ci permette di rappresentare numeri molto grandi o molto piccoli con semplicità, evitando di salvare inutili zeri.

1.2 Numeri di macchina, errori

1.2.1 L'insieme dei numeri di macchina

Un computer può lavorare con un numero finito di cifre: sia per quanto riguarda la mantissa, sia per quanto riguarda l'esponente. Tuttavia, è raro trovare problemi matematici o fisici che non coinvolgano numeri reali; si pensi anche soltanto al calcolo dell'area di un cerchio di raggio r: $A = \pi r^2$. Per questo motivo, ha senso lavorare sui cosiddetti **numeri di macchina**, ovvero numeri rappresentabili **esattamente** da un calcolatore. Questo insieme è definito come:

$$\mathcal{F} = \{0\} \bigcup \{ \underbrace{(-1)^s}_{\text{segno}} \times \underbrace{(0.a_1a_2a_3...a_t)}_{\text{mantissa}} \times \underbrace{N^q}_{\text{esponente}} \ , \ 0 \le a_i < N, \ \ a_1 \neq 0, \ \ L \le q \le U \}. \tag{1.3}$$

Descriviamo a parole la definizione (1.3). Qui, rivediamo le definizioni di segno, mantissa ed esponente. Un primo dettaglio molto importante riguarda a_t: come si può vedere, la mantissa è sempre costituita da t cifre a_i (da a_1 a a_t), poiché il computer viene progettato per operare solo con esse. Seguono nella definizione le condizioni di normalizzazione, tali per cui ciascuna cifra della mantissa deve essere maggiore o uguale di 0, e strettamente minore della base[4]; come già detto si richiede, per rendere univoca la rappresentazione, che la prima cifra della mantissa, a_1, sia diversa da zero. Infine, esistono limiti anche per quanto riguarda il numero di cifre dell'**esponente** che si possono memorizzare. In particolare, L e U rappresentano il minimo e il massimo valore che un esponente possa assumere. Detto in altre parole, l'aritmetica con cui si intende lavorare è determinata una volta fissati N, t, L e U.

Una conseguenza di lavorare unicamente con t cifre di mantissa e con un intervallo finito di esponenti è la **finitezza** dell'insieme dei numeri di macchina $\mathcal{F}$. Infatti, né un generico numero razionale né tantomeno un generico numero reale è un numero macchina. In altre parole,

$$\mathcal{F} \subset \mathbb{Q} \subset \mathbb{R}.$$

La nostra incapacità di lavorare con un numero arbitrario, possibilmente infinito, di cifre, ci porta a dover considerare l'insieme dei numeri di macchina definito in (1.3) come l'unica piattaforma ragionevole per i nostri scopi. Di conseguenza, un numero che non appartiene già a questo insieme dovrà essere **approssimato** in qualche modo, e dietro a ogni approssimazione abbiamo degli **errori** che, in determinate

[4] per esempio, per $N = 10$, una cifra di un numero può essere al minimo 0, e al massimo 9

situazioni, possono compromettere gravemente il risultato delle nostre operazioni. Nel seguito del testo cercheremo di capire più precisamente quali siano le cause nascoste dietro a questi errori, quali siano le situazioni che possono degenerare, e come cercare di porre rimedio a questi problemi.

1.2.2 Errori legati alla *grandezza* del numero

Al fine di identificare la natura del primo tipo di errore, poniamoci due domande:

1. Qual è, in una data aritmetica, il numero rappresentabile più piccolo?
2. Qual è, in una data aritmetica, il numero rappresentabile più grande?

Queste domande possono essere poste in modo lievemente diverso: per *numero rappresentabile*, infatti, è corretto intendere *numero di macchina*. In particolare, il più piccolo numero di macchina m è

$$m = 0.1 \times N^L.$$

Infatti, il minimo valore che la mantissa possa avere è $p = 0.1$, poiché la prima cifra dopo la virgola deve essere diversa da 0, ma quindi al minimo può valere 1. Infine, come già scritto, L è il minimo esponente accettabile nell'aritmetica macchina. Per quanto riguarda il numero macchina M più grande che si possa rappresentare, questo avrà tutte le cifre della mantissa pari a $N-1$, ed esponente pari a U. Per esempio, con $t = 5$, $N = 10$,

$$M = 0.\underbrace{99999}_{t=5} \times 10^U.$$

L'interesse per il massimo e il minimo numero rappresentabili in una data aritmetica è legato alla possibilità che il numero a di partenza sia *troppo piccolo* o *troppo grande* per essere rappresentato. In particolare,

- per $a \in (-m, m)$, di fatto si dice che $a = 0$ (regione di **underflow**);
- per $a \in (-\infty, -M) \cup (+M, +\infty)$, si dice che $a = \pm\infty$ (regione di **overflow**).

Non è molto difficile cadere in una di queste situazioni. Consideriamo per esempio di voler calcolare il numero a definito come

$$a = \frac{\overline{M} + M}{2},$$

dove si è implicitamente definito

$$\overline{M} = \frac{M}{2},$$

e M è il massimo numero di macchina. Calcolare mediante un computer il valore di a non è banale: se infatti prima si calcola la somma di $\overline{M}$ e M, il risultato parziale sarà più grande del massimo numero rappresentabile M, e quindi, per il computer, uguale a ∞; a questo punto, $\infty/2 = \infty$. Se invece calcolo l'espressione con il passaggio

$$a = \frac{\overline{M}}{2} + \frac{M}{2},$$

ossia prima dimezzando i singoli addendi e poi sommandoli, l'operazione si può effettuare. Infatti, in questo modo, si avrebbe:

$$a = \frac{\overline{M}}{2} + \frac{M}{2} = \frac{M}{4} + \frac{M}{2} = \frac{3}{4}M,$$

dal momento che nessuno dei *passaggi intermedi*[5] risulta essere affetto da fenomeni di overflow.

[5] quelli di cui parlavo nell'introduzione ;-)

Un esempio un po' subdolo

Per concludere questo argomento, si propone ora un esempio un po' più cattivello; sarà possibile capirlo più a fondo al termine di questa sezione, quando si parlerà esattamente delle potenzialità di MATLAB® .
Segue il testo del problema.
Dati $x_1 = 10^{10}$, $x_2 = 10^{170}$, calcolare mediante MATLAB® la quantità

$$R = \sqrt{x_1^2 + x_2^2}.$$

Viene riportato ora un breve script implementante due soluzioni, R_1 e R_2.

```
clear
close all
clc

x1=1e10; % notazione esponenziale indicante 10^(10)
x2=1e170; % notazione esponenziale indicante 10^(170)

R1 = sqrt(x1^2 + x2^2) % soluzione errata
R2 = abs(x2)*sqrt(1 + (x1/x2)^2) % soluzione corretta
```

Provando questo script su MATLAB® , si può notare che una delle soluzioni produce un overflow (si capisce dal risultato infinito), l'altra no. Infatti, dal momento che x_2 è un numero molto grande, elevandolo al quadrato diviene ancora più grande e, quindi, genera un overflow. Se al contrario si raccoglie x_2^2 e lo si porta fuori dalla radice, usando il trucco

$$R = \sqrt{x_1^2 + x_2^2} = \sqrt{x_2^2\left(\frac{x_1^2}{x_2^2} + 1\right)} = |x_2|\sqrt{1 + \left(\frac{x_1}{x_2}\right)^2}$$

non si verificherà più alcun fenomeno di overflow. Ovviamente, questa soluzione è valida perché $x_2 \gg x_1$; nel caso opposto, sarebbe stato necessario raccogliere x_1^2 dentro al segno di radice, e procedere di conseguenza.

1.2.3 Errori legati al numero di cifre della mantissa; rounding to even

Come già accennato in precedenza, un generico numero reale non è un numero macchina, in quanto servirebbe un computer a precisione t infinita. Tuttavia, se volessimo per esempio introdurre in un calcolo il numero $\sqrt{2}$, come potremmo fare? La soluzione è **approssimare** $\sqrt{2}$ al numero macchina *più vicino*, *più simile*. Per poter effettuare questa approssimazione, occorre quindi stabilire un processo di approssimazione, ovvero un metodo in grado di associare a un numero reale $a \in \mathbb{R}$ un numero macchina $\overline{a} \in \mathcal{F}$. In particolare, a partire dalla rappresentazione floating-point normalizzata di a,

$$a = (-1)^s \, p \, N^q,$$

il numero macchina $\overline{a}$ che lo approssima avrà una rappresentazione floating-point del tipo

$$\overline{a} = (-1)^s \, \overline{p} \, N^q.$$

Si noti che in questa sezione non ci concentriamo sugli esponenti, poiché gli unici problemi che potrebbero riguardarli sono underflow e overflow, già trattati in precedenza.

La Fig. 1.2 mostra che la mantissa p del numero reale a è compresa tra le mantisse di due numeri macchina $\overline{p}_1, \overline{p}_2$, in cui si assume $\overline{p}_1 > \overline{p}_2$. La mantissa $\overline{p}$ sarà scelta tra $\overline{p}_1$ e $\overline{p}_2$.

Dobbiamo a questo punto capire due cose: quale sia la mantissa di macchina più opportuna, e qual è l'errore che commettiamo approssimando p con essa.

Innanzitutto, si osservi che

$$\overline{p}_2 = \overline{p}_1 + N^{-t},$$

ovvero, per passare da una mantissa di macchina alla successiva, è necessario aggiungere N^{-t}. Questo concetto, illustrato in Fig. 1.2, può essere chiarito mediante un esempio. Si consideri un calcolatore con $t = 5$, $N = 10$. Si consideri $\overline{p}_1 = 0.85472$. Per calcolare $\overline{p}_2$, dovremo scrivere $\overline{p}_2 = \overline{p}_1 + 10^{-5}$, ossia

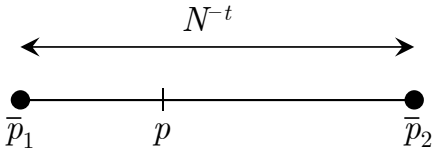

Figura 1.2: Illustrazione della mantissa p associata al numero reale a, e delle mantisse dei due numeri macchina più vicini a esso, $\overline{p}_1$ e $\overline{p}_2$ Nella figura si evidenzia che $\overline{p}_2 - \overline{p}_1 = N^{-t}$.

$$\begin{array}{r} 0.8\,5\,4\,7\,2 \\ +\ 0.0\,0\,0\,0\,1 \\ \hline 0.8\,5\,4\,7\,3. \end{array}$$

In altre parole, si aggiunge un'unità alla cifra meno significativa della mantissa $\overline{p}_1$.

La procedura di approssimazione che si utilizza nei calcolatori è il **rounding to even**, ovvero, **arrotondamento a pari**. Questa tecnica si può riassumere nei seguenti passi.

1. Nel caso la mantissa p sia equidistante alle due più vicine, $\overline{p}_1$ e $\overline{p}_2$, essa va approssimata alla mantissa $\overline{p}$ con ultima cifra pari.
2. Altrimenti,
   (a) si somma a p il termine $\frac{1}{2}N^{-t}$
   (b) si tronca il risultato della precedente operazione alla t-esima cifra.

Al fine di chiarire questo algoritmo, verranno ora proposti alcuni esempi.

Esempio 1

Si consideri un calcolatore operante in sistema decimale con tre cifre. Quindi, sia $p = 0.15814$. Arrotondarlo secondo l'algoritmo rounding to even.

Prima di tutto, $N = 10$, $t = 3$. Di conseguenza, le due mantisse di macchina più vicine sono $\overline{p}_1 = 0.158$ e $\overline{p}_2 = 0.159$. Inoltre, si può capire facilmente che p non sia equidistante da esse, quindi si deve passare alla parte (2) dell'algoritmo. Si consideri quindi

$$p + \frac{1}{2}N^{-t} = 0.15814 + \frac{1}{2}10^{-3},$$

che produce

$$\begin{array}{r} 0.1\,5\,8\,1\,4 \\ +\ 0.0\,0\,0\,5 \\ \hline 0.1\,5\,8\,6\,4. \end{array}$$

A questo punto, si tronchi 0.15864 alle prime $t = 3$ cifre, ottenendo $\overline{p} = 0.158$.

Esempio 2

Si consideri un calcolatore operante in sistema decimale con tre cifre. Quindi, sia $p = 0.1585432$. Arrotondarlo secondo l'algoritmo rounding to even.

Di nuovo, $N = 10$, $t = 3$. Di conseguenza, le due mantisse di macchina più vicine sono $\overline{p}_1 = 0.158$ e $\overline{p}_2 = 0.159$. Inoltre, si può capire facilmente che p non è equidistante da esse, quindi si deve passare alla parte (2) dell'algoritmo. Si consideri quindi

$$p + \frac{1}{2}N^{-t} = 0.1585432 + \frac{1}{2}10^{-3},$$

che produce

$$\begin{array}{r} {\scriptstyle 1} \\ 0.1\,5\,8\,5\,4\,3\,2 \\ +\ 0.0\,0\,0\,5 \\ \hline 0.1\,5\,9\,0\,4\,3\,2. \end{array}$$

A questo punto, si tronchi 0.1590432 alle prime $t = 3$ cifre, ottenendo $\overline{p} = 0.159$.

Esempio 3

Si consideri un calcolatore operante in sistema decimale con tre cifre. Quindi, sia $p = 0.1585$. Arrotondarlo secondo l'algoritmo rounding to even.

In questo caso, p è equidistante da $\overline{p}_1 = 0.158$ e $\overline{p}_2 = 0.159$. Di conseguenza, la mantissa di macchina è quella più vicina e che termina con una cifra pari, ovvero $\overline{p} = 0.158$.

1.2.4 Errore assoluto, errore relativo

Dato un numero $x \in \mathbb{R}$, sia $\widetilde{x}$ il numero associato a x tramite la procedura descritta nella sezione precedente. Una volta memorizzato $\widetilde{x}$ e poi lavorando con esso, commettiamo un errore, ovvero introduciamo una differenza tra ciò che avremmo usando x e ciò che abbiamo usando $\widetilde{x}$. Esistono sostanzialmente due definizioni di errore:

- Si definisce **errore assoluto** e_a la grandezza

$$e_\mathrm{a} = |x - \widetilde{x}|.$$

- Si definisce **errore relativo** e_r la grandezza

$$e_\mathrm{r} = \frac{|x - \widetilde{x}|}{|x|}, \quad x \neq 0.$$

In entrambi i casi, queste definizioni permettono di quantificare la differenza tra *la verità* e ciò che otteniamo dalla nostra approssimazione. L'errore relativo, generalmente, è una quantità più significativa, perché non solo dice *quanto si sbaglia*, ma anche rispetto all'entità del numero. Volendo fare un esempio un po' stupido, si immagini di sbagliare una domanda in un esame contenente trenta domande, e una domanda in un esame contenente tre domande. In entrambi i casi, si sbaglia una sola domanda. Ma il voto finale in un caso sarà 29, nell'altro sarà 18. :-D

È possibile stimare l'errore che si commette nel momento in cui si effettua un arrotondamento. In particolare, quando si parla di *stima*, si intende cercare una *maggiorazione*, ossia un numero che ci permetta di capire *al peggio* quanto possa valere questo errore. Dati un numero reale a e il suo arrotondamento $\overline{a}$, si è detto che

$$\begin{aligned} a &= (-1)^s\, p\, N^q \\ \overline{a} &= (-1)^s\, \overline{p}\, N^q. \end{aligned}$$

Quindi, si può calcolare l'errore assoluto come

$$\begin{aligned} e_\mathrm{a} = |a - \overline{a}| = |(-1)^s p N^q - (-1)^s \overline{p} N^q| = |(-1)^s N^q (p - \overline{p})| = \\ = N^q\, |p - \overline{p}|. \end{aligned}$$

Maggiorare l'errore assoluto sulla mantissa è, in questo caso, molto semplice! Infatti, ricordandoci la Fig. 1.2, sappiamo che, nel caso peggiore, p può essere esattamente al centro tra $\overline{p}_1$ e $\overline{p}_2$; in questo caso, sapendo che $\overline{p}_2 - \overline{p}_1 = N^{-t}$, l'errore *di caso peggiore* è $\frac{1}{2} N^{-t}$. Di conseguenza, abbiamo appena capito che

$$|p - \overline{p}| \leq \frac{1}{2} N^{-t}.$$

Quindi, considerando l'errore assoluto sull'intero numero a,

$$|a - \overline{a}| = N^q |p - \overline{p}| \leq N^q N^{-t} \frac{1}{2} = \frac{1}{2} N^{q-t}.$$

Per quanto riguarda l'errore relativo,

$$e_r = \frac{|a - \overline{a}|}{|a|} = \frac{N^q |p - \overline{p}|}{N^q |p|}.$$

Tuttavia, è possibile affermare, per via della normalizzazione che permette di rendere univoca la rappresentazione floating point (1.2), che

$$|p| \geq N^{-1},$$

e quindi, sostituendo al denominatore $|p|$ il suo minorante N^{-1}, si ottiene un'ulteriore maggiorazione dell'errore relativo, scrivibile in modo compatto:

$$\frac{|p - \overline{p}|}{|p|} \leq \frac{\frac{1}{2} N^{-t}}{N^{-1}} = \frac{1}{2} N^{1-t} \triangleq \varepsilon_m.$$

Il termine ε_m che abbiamo definito nella riga precedente è detto *precisione di macchina*, ed è la stima che stavamo cercando per quanto riguarda l'errore che si commette ogni volta che si effettua un arrotondamento. Infatti, abbiamo appena dimostrato che

$$\frac{|a - \overline{a}|}{|a|} = \frac{|p - \overline{p}|}{|p|} \leq \varepsilon_m.$$

L'importanza di questa formula è assoluta. Infatti, essa ci insegna che, nel momento in cui arrotondiamo un numero, al massimo l'errore che commettiamo è pari a ε_m. In altre parole, abbiamo la garanzia che non sia possibile avere un errore superiore!

1.2.5 Alcune note su MATLAB

È importante a questo punto sapere quali siano questi parametri per quanto riguarda MATLAB®, software di Calcolo Numerico impiegato in queste note. MATLAB® lavora con lo standard IEEE754 binario in doppia precisione. Parlando di sistema binario dunque $N = 2$, si parla di bit. Lo standard prevede:

- 1 bit di segno,
- 52 bit di mantissa,
- 11 bit di esponente.

Per quanto riguarda la precisione di macchina, possiamo calcolare

$$\varepsilon_m = \frac{1}{2} N^{1-t} = \frac{1}{2} 2^{1-52} \simeq 2.204 \times 10^{-16}.$$

In effetti, provando a scrivere il comando `eps` sulla console di MATLAB®, scopriremmo che...

```
>> eps
ans =
   2.2204e-16
>>
```

... tutto sommato quello che stiamo dicendo ha un suo perché. ;-) Volendo ragionare in base 10, ciò che abbiamo appena mostrato ci dice che MATLAB® è in grado di lavorare con circa 16 cifre decimali.

Per quanto riguarda gli esponenti, i numeri massimi e i numeri minimi, si può scoprire[6] che $L = -1021$, $U = +1024$, e quindi che:

$$M = \left(1 - 2^{-52}\right) \times 2^{1024} = \left(2 - 2^{-51}\right) \times 2^{1023} \simeq 1.7977 \times 10^{308}$$
$$m = 2^{-1} \times 2^{-1021} = 2.2251 \times 10^{-308}.$$

In particolare, se proviamo a lanciare i comandi `realmax` e `realmin` sulla console di MATLAB®, scopriamo che la teoria appena sviluppata è effettivamente consistente.

[6] maggiori dettagli possono essere per esempio trovati in [1, pag. 3]

```
>> realmax
ans =
  1.7977e+308
>> realmin
ans =
  2.2251e-308
>>
```

Questo dimostra che MATLAB® tratta come infiniti i numeri più grandi di $M(1+\varepsilon_m)$. Tuttavia, per quanto riguarda il minimo reale rappresentabile m, vengono effettuate delle rinormalizzazioni che permettono di rappresentare numeri ancora minori.

1.3 Operazioni di macchina

Nella precedente sezione abbiamo appreso cosa accade ogni volta che arrotondiamo un numero reale, al fine di associargli un numero macchina che sia trattabile dall'aritmetica del nostro calcolatore. Tuttavia, questa operazione è solo il preludio: il motivo per cui ci interessa memorizzare questi numeri è poter sfruttare il calcolatore per eseguire le operazioni aritmetiche al nostro posto! Diciamo che questo è un po' come quando noi esseri umani, prima di fare una somma o un'addizione, abbiamo bisogno di tenere a mente, o di scrivere su un foglio, gli addendi; allo stesso modo, il computer dovrà salvarli nella propria memoria, prima di poter eseguire le operazioni. Poiché né noi né i computer sappiamo lavorare con infinite cifre, abbiamo capito che dobbiamo in qualche modo arrotondare.

Nonostante abbiamo la garanzia che l'arrotondamento *di per sé* non possa influenzare più di tanto l'accuratezza dei nostri calcoli, dobbiamo ancora capire cosa succede quando eseguiamo delle operazioni aritmetiche a partire dai numeri di macchina ottenuti dall'arrotondamento. In particolare, una prima domanda che potremmo farci è:

> «Eseguire un'operazione aritmetica su due numeri di macchina, ci dà la garanzia che il risultato sia un numero di macchina?»

La risposta a questa domanda, purtroppo, è negativa. Consideriamo, per esempio, $N = 10$, $t = 4$, $\overline{a}_1 = 0.5823$, $\overline{a}_2 = 0.6214$. Il fatto di aver indicato i numeri con una barretta ($\overline{a}$) è legato al fatto che questi sono già numeri arrotondati, e che infatti sono numeri di macchina per l'aritmetica specificata da N e t. La somma di questi numeri è:

$$\begin{array}{r} {\scriptstyle 1\ 1\ \ } \\ 0.5\,8\,2\,3 \\ +\ 0.6\,2\,1\,4 \\ \hline 1.2\,0\,3\,7. \end{array}$$

Il numero 1.2037, nella notazione floating-point normalizzata, è scritto come 0.12037×10^1 ma, quindi, la sua mantissa non sarebbe memorizzabile nella nostra aritmetica (richiederebbe 5 cifre, ma noi abbiamo $t = 4$). Questo semplicissimo esempio, quindi, permette di dimostrare che **il risultato di un'operazione aritmetica tra due numeri di macchina generalmente non fornisce un numero di macchina**. Questo fatto ci costringe ad aggiungere una nuova definizione: quella di **operazione di macchina**.

Un'operazione di macchina $\odot$ associa a due numeri di macchina un terzo numero di macchina, il quale è ottenuto arrotondando il risultato dell'operazione esatta eseguita sui numeri di macchina di partenza. A partire quindi da due numeri $\overline{a}_1$ e $\overline{a}_2$, una generica operazione di macchina $\odot$ si definisce come:

$$\overline{a}_1 \odot \overline{a}_2 = \overline{\overline{a}_1 \cdot \overline{a}_2} = (\overline{a}_1 \cdot \overline{a}_2)(1+\varepsilon_\odot).$$

Per esempio, per la somma, si avrebbe

$$\overline{a}_1 \oplus \overline{a}_2 = \overline{\overline{a}_1 + \overline{a}_2} = (\overline{a}_1 + \overline{a}_2)(1+\varepsilon_\oplus).$$

Queste ultime espressioni si possono leggere come:

> «L'operazione macchina $\odot$ eseguita tra i numeri di macchina $\overline{a}_1$ e $\overline{a}_2$ è uguale all'arrotondamento del risultato di $\overline{a}_1 \cdot \overline{a}_2$. Di conseguenza, l'errore $\varepsilon_\odot$ introdotto dall'arrotondamento del risultato è, proprio come in passato, minore della precisione di macchina ε_m.»

Riprendendo l'esempio di prima, una volta ottenuto $\overline{a}_1 + \overline{a}_2 = 0.12037 \times 10^1$, il risultato di $\overline{a}_1 \oplus \overline{a}_2$ sarebbe quindi 0.1204×10^1, ovvero l'arrotondamento alla quarta cifra.

Una particolarità delle operazioni di macchina, rispetto alle operazioni in aritmetica esatta, è l'assenza di alcune delle *proprietà di base*. In particolare, valgono le proprietà commutative della somma e del prodotto

$$\overline{a}_1 \oplus \overline{a}_2 = \overline{a}_2 \oplus \overline{a}_1$$
$$\overline{a}_1 \otimes \overline{a}_2 = \overline{a}_2 \otimes \overline{a}_1,$$

ma **non** valgono varie altre proprietà, tra cui le proprietà associative

$$\overline{a}_1 \oplus (\overline{a}_2 \oplus \overline{a}_3) \neq (\overline{a}_1 \oplus \overline{a}_2) \oplus \overline{a}_3$$
$$\overline{a}_1 \otimes (\overline{a}_2 \otimes \overline{a}_3) \neq (\overline{a}_1 \otimes \overline{a}_2) \otimes \overline{a}_3,$$

e le proprietà distributive

$$\overline{a}_1 \otimes (\overline{a}_2 \oplus \overline{a}_3) \neq (\overline{a}_1 \otimes \overline{a}_2) \oplus (\overline{a}_1 \otimes \overline{a}_3)$$
$$(\overline{a}_1 \otimes \overline{a}_2) \oslash \overline{a}_3 \neq (\overline{a}_1 \oslash \overline{a}_3) \otimes \overline{a}_2.$$

Inoltre, un'altra relazione molto strana per chi è abituato a pensare in aritmetica esatta è

$$\overline{a}_1 \oplus \overline{a}_2 = \overline{a}_1,$$

che, detto brutalmente, significa che *se un termine è molto più piccolo dell'altro, allora non conta nell'operazione macchina*. Cosa vuol dire *molto* più piccolo? In questo caso, dire che $|\overline{a}_2| \ll |\overline{a}_1|$ si può tradurre come $|\overline{a}_2| < \varepsilon_{\mathrm{m}} |\overline{a}_1|$. Per esempio, si potrebbe provare a introdurre i seguenti comandi sulla command window di MATLAB®:

```
>> format long e
>> 5+1.0e-18
ans =
5
>> 2017.3+1.0e-14
ans =
2.017300000000000e+03
```

Sommando questi due numeri, si ha che 10^{-14} è trascurabile a 2017.3, proprio perché il primo è più piccolo del secondo moltiplicato per ε_{m}.

Più in generale, quando si opera con un'aritmetica di macchina, è possibile che due espressioni siano **equivalenti**, per quanto invece non lo sarebbero in aritmetica esatta. In particolare, due quantità $\overline{a}_1$ e $\overline{a}_2$ si dicono equivalenti quando

$$e_{\mathrm{r}} = \frac{|\overline{a}_1 - \overline{a}_2|}{\overline{a}_1} \quad \text{oppure} \quad e_{\mathrm{r}} = \frac{|\overline{a}_1 - \overline{a}_2|}{\overline{a}_2}$$

è dello stesso ordine di grandezza (o possibilmente minore), della precisione di macchina ε_{m}. Per dare un po' di *feeling numerico* in più sull'argomento, consideriamo per esempio $N = 10$, $t = 5$, e $\overline{a}_1 = 0.99999$, $\overline{a}_2 = 1$. Se calcoliamo

$$e_{\mathrm{r}} = \frac{|0.99999 - 1|}{1} = 0.00001,$$

vediamo che $0.00001 = 0.1 \times 10^{-4}$ è minore di ε_{m}, che invece sarebbe:

$$\varepsilon_{\mathrm{m}} = \frac{1}{2} \times 10^{1-5} = 0.5 \times 10^{-4},$$

e quindi, per l'aritmetica considerata in questo esempio, i numeri 0.99999 e 1 sono del tutto equivalenti. Se vogliamo pensarlo in maniera un po' più intuitiva, possiamo dire che, se il rapporto tra i due numeri a partire dai quali vogliamo effettuare un'operazione di macchina è più piccolo di ε_{m}, allora il numero piccolo è sostanzialmente invisibile dal punto di vista del calcolatore, e non conterà nulla. Questo ovviamente avrà conseguenze sull'ordine con cui facciamo i calcoli, rendendo invalide le proprietà fondamentali quali la distributiva e/o la associativa.

1.4 Cancellazione numerica

Finora non abbiamo trovato punti critici: l'errore introdotto da un'operazione di macchina, applicata quindi a dei numeri già arrotondati, è semplicemente un errore di arrotondamento sul risultato; di conseguenza, questo è sicuramente più piccolo di ε_{m}. È vero, abbiamo notato cose un po' strane per quanto riguarda le proprietà delle operazioni di macchina, o per il fatto che due espressioni che sarebbero diverse in aritmetica esatta risultino esattamente *equivalenti* in aritmetica di macchina, quindi *dal punto di vista del calcolatore*; tutti questi errori, però, sono assolutamente minuscoli: sempre inferiori alla precisione di macchina! Ma a questo punto, è d'obbligo chiedersi:

> «Se né arrotondare un numero né le operazioni di macchina introducono problemi, allora da dove nascono 'sti problemi? Cioè, insomma, cos'è che causa i disastri di cui si parlava nell'introduzione?!»

La risposta alla domanda è nascosta nella domanda stessa. Infatti, se leggiamo con attenzione, possiamo notare che:

- l'operazione di arrotondamento coinvolge solo i numeri reali/razionali che si desidererebbe rappresentare in aritmetica macchina, ma non considera in alcun modo ciò che succederà dopo, ovvero, come *useremo* i numeri arrotondati;
- le operazioni di macchina producono un numero di macchina a partire da operandi, assumendo che essi siano già dei numeri di macchina; tuttavia, quindi, non prende in alcun modo in considerazione l'operazione di arrotondamento che li produce.

Se però pensiamo all'intero procedimento, questo è:

1. partire da due numeri che in generale possono essere reali, a_1 e a_2;
2. arrotondarli, ottenendo due numeri $\overline{a}_1$, $\overline{a}_2$;
3. eseguire l'operazione $\overline{a}_1 \cdot \overline{a}_2$ in aritmetica esatta;
4. arrotondare questo risultato: $\overline{\overline{a}_1 \cdot \overline{a}_2}$, concludendo l'operazione di macchina cominciata nel punto precedente.

Nonostante presi singolarmente arrotondamento e operazioni di macchina non generino grossi errori, **usarli assieme**, come si deve di fatto fare, può portare a disastri. Per capire meglio questo punto, è opportuno concentrarsi su un esempio.

1.4.1 Esempio e definizione di cancellazione numerica

Consideriamo $N = 10$, $t = 5$. In questo caso, si può vedere facilmente con le solite formule che $\varepsilon_{\mathrm{m}} = 0.5 \times 10^{-4}$. Consideriamo quindi i numeri $a_1 = 0.157824831$, $a_2 = 0.157348212$. Il nostro obiettivo sarà calcolare la loro differenza in aritmetica esatta e in aritmetica di macchina, e capire se le cose continuano ad andar bene come finora è successo.

Prima di tutto, proviamo a calcolare quindi $a_1 - a_2$, ovvero la differenza dei due numeri esatti, **non arrotondati**:

$$\begin{array}{r} 0.1\,5\,7\,8\,2\,4\,8\,3\,1 \\ -\ 0.1\,5\,7\,3\,4\,8\,2\,1\,2 \\ \hline 0.0\,0\,0\,4\,7\,6\,6\,1\,9, \end{array}$$

ovvero, $a_1 - a_2 = 0.476619 \times 10^{-3}$. Questo non è un numero di macchina, ma, se immaginiamo di arrotondare questo risultato, non notiamo nulla di devastante; infatti, si otterrebbe

$$\overline{a_1 - a_2} = 0.47662 \times 10^{-3},$$

dove l'errore commesso come al solito sarebbe minore di ε_{m}.

Tuttavia, l'operazione che abbiamo appena descritto non è davvero quella che si esegue quando si lavora con un calcolatore, è in qualche modo *idealizzata*. Infatti, come descritto poco fa nel testo, **prima si arrotonda, poi si esegue l'operazione di macchina a partire dai numeri arrotondati**: non è possibile

eseguire l'operazione di macchina su numeri *esatti*: il calcolatore in questione non è stato progettato per memorizzarli!

Volendo quindi seguire la procedura *corretta*, prima dovremmo arrotondare i due numeri, ottenendo $\overline{a}_1 = 0.15782$, $\overline{a}_2 = 0.15735$. Come al solito, l'errore di arrotondamento è inferiore a ε_m: potremmo calcolarlo, ma non è necessario, dal momento che siamo sicuri della nostra stima legata a ε_m ! Quindi, ora, dovremmo calcolare, a partire dai numeri arrotondati,

$$\begin{array}{r} 0.1\,5\,7\,8\,2 \\ -\ 0.1\,5\,7\,3\,5 \\ \hline 0.0\,0\,0\,4\,7, \end{array}$$

che è già un numero di macchina, e che si può scrivere come

$$\overline{a}_1 \ominus \overline{a}_2 = \overline{\overline{a}_1 - \overline{a}_2} = 0.47 \times 10^{-3}.$$

Vedere solo due cifre nel risultato dovrebbe già far nascere un qualche senso di inquietudine. A questo punto, però, proviamo a calcolare l'errore relativo tra il risultato della prima operazione e quello della versione *corretta*:

$$e_r = \frac{|(a_1 - a_2) - (\overline{a}_1 \ominus \overline{a}_2)|}{|a_1 - a_2|} = \frac{0.47662 - 0.47}{0.47662}$$

se lo calcoliamo con MATLAB®, per esempio,

```
>> abs(0.47662-0.47)/(0.47662)
ans =
    0.0139
```

ossia, $e_r \simeq 0.1 \times 10^{-1}$: **questo è molto, molto maggiore rispetto alla precisione di macchina!!!** Abbiamo provato a calcolare l'arrotondamento solo del risultato, ottenuto a partire dall'operazione eseguita sui numeri esatti e non abbiamo riscontrato problemi; l'arrotondamento di per sé non introduce problemi; ciononostante, **unire le due sotto-operazioni può introdurre un'amplificazione degli errori di arrotondamento!**

Il fatto di aver *perso* cifre significative nel risultato dell'operazione di macchina *corretta*, *non idealizzata*, è la conseguenza del cosiddetto fenomeno di **cancellazione numerica**. Visto che si parla di *sparizione di cifre*, possiamo intuire che l'operazione che permette un fenomeno del genere debba essere la sottrazione: è l'unica che possa far *sparire* qualcosa[7]! Inoltre, guardando con attenzione l'esempio, notiamo che i due numeri sono abbastanza simili tra loro. Questo ci permette di capire che **la cancellazione numerica avviene quando si effettua la sottrazione tra due numeri simili tra loro**.

Si noti che **simili** significa che l'esponente dei numeri di partenza deve essere lo stesso (altrimenti i due numeri non sarebbero confrontabili), così come il segno (altrimenti la differenza diventerebbe una somma), e almeno uno dei due sottraendi deve essere affetto da errori di arrotondamento (altrimenti, la *perdita di cifre* sarebbe in verità il risultato corretto). Consideriamo alcuni esempietti in cui **non** avviene cancellazione numerica.

Esempio 1

Consideriamo $N = 10$, $t = 5$, $a_1 = 0.15782$, $a_2 = 0.15735$. Questi due numeri sono numeri di macchina, quindi $\overline{a}_1 = a_1$, $\overline{a}_2 = a_2$. Se si calcolano $a_1 - a_2$, o $\overline{a}_1 - \overline{a}_2$, si otterranno esattamente gli stessi risultati e, quindi, non si avrà cancellazione numerica.

Esempio 2

Consideriamo $N = 10$, $t = 5$, $a_1 = 0.157824831 \times 10^0$, $a_2 = 0.157348212 \times 10^{-1}$. Questi due numeri non sono numeri di macchina, quindi andranno arrotondati, ottenendo $\overline{a}_1 = 0.15782 \times 10^0$, $\overline{a}_2 = 0.15735 \times 10^{-1}$. Tuttavia, se ora calcoliamo la loro sottrazione esatta $a_1 - a_2$, avremo

$$\begin{array}{r} 0.1\,5\,7\,8\,2\,4\,8\,3\,1\,0 \\ -\ 0.0\,1\,5\,7\,3\,4\,8\,2\,1\,2 \\ \hline 0.1\,4\,2\,0\,9\,0\,0\,0\,9\,8, \end{array}$$

[7] questa intuizione verrà formalizzata meglio nella prossima sezione

e, per quanto riguarda la sottrazione di macchina,

$$\begin{array}{r} 0.1\,5\,7\,8\,2\,0 \\ -\ 0.0\,1\,5\,7\,3\,5 \\ \hline 0.1\,4\,2\,0\,8\,5, \end{array}$$

e, chiaramente, non si hanno errori clamorosi. Questo perché **in questo caso, l'esponente è diverso**.

Un esempio di cancellazione numerica

Abbiamo quindi capito che i problemi devastanti di cui si parlava nascono semplicemente dal sottrarre due numeri simili tra loro.

> «Ma, cioè, ma chi se ne frega! Capita così spesso?!»

Sì. Per esempio, un calcolo con cui i reduci di Analisi Matematica I dovrebbero avere una certa confidenza, è

$$\lim_{h\to 0} \frac{f(x+h)-f(x)}{h},$$

ossia, il calcolo della derivata mediante limite del rapporto incrementale con incremento h. Molto spesso, capita di non conoscere un'espressione *in forma chiusa* della funzione $f(x)$, bensì solo per punti. In questo caso, dovendo calcolare la pendenza in un punto, si può usare questo limite calcolando esplicitamente questo rapporto. Il problema è che

- se mettiamo un h grande, la pendenza che si ottene non è significativa, poiché stiamo semplicemente calcolando la pendenza della retta che interseca i punti x e $x+h$ lontani tra loro;
- se mettiamo un h troppo piccolo, $f(x+h) \simeq f(x)$, che è proprio il caso sfortunato per cui avvengono i fenomeni di cancellazione numerica. :-(

1.4.2 Idee per risolvere la cancellazione numerica

Abbiamo intuito da dove nasce il problema della cancellazione numerica, dunque ora il lettore si aspetta che gli si possa dare una *ricetta* in grado di risolverlo. Purtroppo, sarà allora piuttosto deluso, perché:

- una ricetta *unica* e *indolore* non esiste;
- esistono svariati *trucchetti* che possiamo usare nelle diverse situazioni che ci capitano, e che in effetti in quelle situazioni sono molto efficaci;
- nel caso non si disponga di trucchetti, è sempre possibile **aumentare la precisione di macchina**[8]; questo però è solo un palliativo: il problema della cancellazione continuerà a sussistere, ma apparirà con maggiore difficoltà.

L'obiettivo di questa sezione del testo non vuole essere spingere verso soluzioni *brute-force*, quale per esempio è aumentare la precisione di macchina. Il nostro scopo è imparare almeno alcuni dei trucchi che possono risolvere alcune specifiche situazioni in maniera *pulita*. Purtroppo, ogni situazione è diversa da tutte le altre e, quindi, solo *l'esperienza*, acquisita mediante l'esercizio, può aiutare a capire come districarsi nella giungla della cancellazione numerica.

Esempio 1: razionalizzazione -se possibile-

Si consideri il seguente esempio:

$$f(x) = \sqrt{x+\delta} - \sqrt{x},$$

in cui $x > 0$, $x+\delta > 0$. Evidentemente, abbiamo una sottrazione tra le due radici. Se abbiamo che $|\delta| \ll x$, si avrà che il primo termine sarà molto simile al secondo, e quindi potrebbero verificarsi fenomeni di

[8] per esempio esiste un tool a pagamento per MATLAB®, con informazioni disponibili presso https://www.advanpix.com/; in effetti questa è la soluzione che ho adottato per il mio simulatore di trasporto elettronico D1ANA usato per esempio in [2]

cancellazione numerica. Un trucco che permette di risolvere questo caso è basato sulla **razionalizzazione**, idea usata spesso anche in Analisi Matematica I per risolvere limiti con forme indeterminate. In particolare, si può moltiplicare a numeratore e denominatore per il termine $\sqrt{x+\delta}+\sqrt{x}$, ottenendo:

$$f(x) = \sqrt{x+\delta} - \sqrt{x} = (\sqrt{x+\delta} - \sqrt{x})\frac{\sqrt{x+\delta}+\sqrt{x}}{\sqrt{x+\delta}+\sqrt{x}} = \frac{x+\delta-x}{\sqrt{x+\delta}+\sqrt{x}} =$$
$$= \frac{\delta}{\sqrt{x+\delta}+\sqrt{x}}.$$

Scritta in questo modo, questa espressione non contiene più le sottrazioni *critiche*, e quindi non può essere affetta da problemi di cancellazione numerica per δ piccolo. Inoltre, dal momento che per passare dalla scrittura di partenza a quella di arrivo non sono state sfruttate né approssimazioni di alcun tipo, né proprietà dell'aritmetica di macchina, le due espressioni sono totalmente **equivalenti in aritmetica esatta**.

Tuttavia, questo esempio apparentemente banale contiene altre insidie. Infatti, se $\delta \simeq -x$, la differenza tra le due radici non è più un problema, ma all'interno di $\sqrt{x+\delta}$ si verificherebbe un **altro** problema di cancellazione, che in questo caso non sarebbe risolubile con la razionalizzazione (poiché razionalizzare farebbe apparire la radice in questione al denominatore, come abbiamo visto, e questa continuerebbe a essere affetta dal problema). Dunque, per $\delta \simeq -x$, **non abbiamo soluzioni banali al problema della cancellazione numerica**.

Esempio 2: espansione di Taylor

Si consideri l'espressione

$$f(x) = \frac{-1+e^x}{x}.$$

Quando $x \simeq 0$, si ha che $e^0 \simeq 1$, e quindi si verifica un fenomeno di cancellazione numerica. Per risolvere questo problema, una soluzione può riguardare l'utilizzo di un'espansione di Taylor nell'intorno del punto di interesse. Infatti, come noto da Analisi Matematica I, l'espansione di Maclaurin (Taylor in $x = 0$) è:

$$e^x \simeq 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + ...$$

In particolare, quindi, la funzione diverrebbe

$$f(x) \simeq \frac{1}{x}\left(-1 + 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + ...\right) =$$
$$= 1 + \frac{x}{2!} + \frac{x^2}{3!} + ...$$

che non contiene più differenze, e quindi nemmeno problemi di cancellazione numerica!

Esempio 3: relazioni trigonometriche -quando disponibili-

Sembrerebbe che utilizzare espansioni di Taylor / Maclaurin possa essere la fantastica *ricetta definitiva* che ci piacerebbe avere: più o meno ogni funzione ammette uno sviluppo di Taylor! Purtroppo, però, questa strada ha due controindicazioni.

- Questi sviluppi di Taylor richiedono -in teoria- la somma di infiniti termini e, quindi, valutarli può richiedere un certo sforzo computazionale.
- Considerando per esempio il caso delle funzioni oscillanti come seno o coseno, gli sviluppi di Taylor sono a segni alterni, quindi introdurre Taylor non elimina necessariamente l'operazione di sottrazione dalle espressioni.

Considerando per esempio la funzione

$$f(x) = \frac{1-\cos x}{x^2}$$

per $x \simeq 0$, Taylor non rappresenta la via più indicata. Tuttavia, possiamo ricordarci le formule di bisezione del seno,

$$\sin\left(\frac{x}{2}\right) = \pm\sqrt{\frac{1-\cos x}{2}}.$$

Manipoliamo quindi la nostra $f(x)$ al fine di poter sfruttare questa espressione; per far questo, moltiplichiamo e dividiamo per due, ottenendo

$$\frac{1-\cos x}{x^2} = \frac{2}{x^2}\frac{1-\cos x}{2}.$$

Possiamo quindi sostituire in questa espressione il quadrato dell'espressione appena trovata (facendo così anche sparire l'indeterminazione di segno $\pm$), ottenendo

$$\frac{2}{x^2}\frac{1-\cos x}{2} = \frac{2}{x^2}\sin^2\left(\frac{x}{2}\right) = \frac{1}{2}\frac{4}{x^2}\sin^2\left(\frac{x}{2}\right) = \frac{1}{2}\left[\frac{\sin\left(\frac{x}{2}\right)}{\frac{x}{2}}\right]^2.$$

Questa funzione per $x \to 0$ non presenta alcun problema[9], e quindi la cancellazione numerica è stata risolta.

1.5 Condizionamento di un problema numerico

Nella precedente sezione abbiamo capito, studiando un certo numero di esempi, che esiste la possibilità di avere fenomeni di amplificazione degli errori, ovvero, di crescita imprevista rispetto alle stime che siamo riusciti a ottenere sui semplici arrotondamenti. In effetti, il meccanismo alla base di questa amplificazione è una specie di *combo* di arrotondamento dei dati e arrotondamento delle operazioni di macchina. In aggiunta a tutto ciò, questi esempi ci hanno permesso di capire che alcune situazioni in cui si verifica la cancellazione numerica si possono risolvere mediante dei *trucchi*, ma altre no. Volendo quindi dirlo con parole diverse, alcuni problemi sono **intrinsecamente presenti nel problema**, mentre altri sono **contenuti nell'algoritmo**, ovvero, modificando la procedura di calcolo è possibile rimuoverli. Con riferimento all'esempio nella sezione 1.4.2, il problema per $|\delta| \ll x$ era risolubile modificando la procedura di calcolo della funzione, quindi l'algoritmo, mentre la cancellazione legata a $\delta \simeq -x$ non era trattabile, e quindi **intrinsecamente presente nel problema**. L'obiettivo di questa sezione sarà formalizzare un po' meglio queste definizioni, fino a spiegare, in termini diversi, cosa sia il problema della cancellazione numerica.

1.5.1 Definizione di problema numerico; forma implicita, forma esplicita

Un problema numerico in questo contesto è una relazione f che lega i dati di ingresso (input), ossia le x, ai dati di uscita (output), ovvero le y. A seconda della situazione, un problema numerico può essere fornito in forma **esplicita**, ovvero del tipo

$$y = f(x),$$

o in forma **implicita**, ovvero

$$f(x, y) = 0.$$

La particolarità della forma esplicita è che a partire dal valore dell'input x, è direttamente possibile ottenere gli output. Questo non è possibile nella forma implicita. Per capire questo concetto, si pensi a una retta:

[9] se non una insignificante discontinuità eliminabile

- l'equazione della retta può essere scritta in forma esplicita, ovvero

$$y = mx + q,$$

  e quindi, per vari valori di x, è immediatamente possibile trovare i valori di y;

- l'equazione della retta può anche essere scritta in forma implicita, ovvero

$$ax + by + c = 0;$$

  in questo caso, non è possibile, a partire dai valori di x, trovare immediatamente quelli di y; tutto ciò che si può fare è, data una coppia di valori (x, y), capire se essi soddisfano l'equazione e quindi appartengono alla retta.

Nel caso della retta, con semplici manipolazioni, è possibile ottenere una rappresentazione dall'altra; questo non è il caso di molti altri problemi. Per esempio, si consideri il problema matriciale[10]

$$\underline{\underline{A}}\,\underline{y} - \underline{x} = 0.$$

Questa è un'equazione matriciale: un sistema lineare. Questo è scritto in forma implicita, dal momento che si dovrebbero provare vari vettori $\underline{x}$ e $\underline{y}$ nell'equazione, per verificare se è soddisfatta. Al contrario, è possibile a certe condizioni scrivere questo problema in forma implicita come

$$\underline{y} = \underline{\underline{A}}^{-1}\underline{x},$$

dove $\underline{\underline{A}}^{-1}$ è detta *matrice inversa*.

1.5.2 Definizione di condizionamento di un problema numerico

In un problema numerico, da un ingresso x, si immagini di ottenere un numero $\overline{x}$, dopo averlo perturbato[11]. Quindi, si calcoli

$$\overline{y} = f(\overline{x}).$$

Questo significa che $\overline{y}$ è il risultato dell'operazione f, effettuata in precisione infinita (aritmetica esatta), a partire dai dati perturbati $\overline{x}$. Se l'errore su $\overline{y}$ è dello stesso ordine di grandezza di $\overline{x}$, il problema si dice **ben condizionato**. Volendo scrivere questa proposizione in *matematichese*, avremmo

$$\frac{|y - \overline{y}|}{|y|} \leq \kappa(f, x)\frac{|x - \overline{x}|}{|x|},$$

dove $y = f(x)$ è il risultato ottenuto a partire dal dato non perturbato (e lavorando sempre in aritmetica esatta). Il senso di questa operazione, quindi è semplicemente quello di concentrarsi sul fatto che perturbare un dato di ingresso non causi un'esplosione dell'errore. In altre parole, si desidera trovare un valore per κ che ci permetta di avere la certezza che il membro sinistro non sia più grande del membro destro; se però il κ che troviamo è un numero grande, allora, nel caso peggiore, la perturbazione dell'ingresso sarà fortemente amplificata sull'uscita. Il problema è dunque detto detto **ben condizionato** se è possibile trovare una costante di amplificazione dell'errore κ, la quale dipende dal problema f e dall'input x, che sia **piccola**; la costante κ viene per questo detta **numero di condizionamento**.

1.5.3 Esempio: cancellazione numerica

Al fine di applicare i concetti introdotti nella precedente sezione, ci poniamo l'obiettivo di studiare il condizionamento del problema

$$y = x_1 + x_2, \qquad x_1, x_2 \in \mathbb{R}.$$

[10] in queste note, per chiarezza, le matrici verranno sempre indicate con una doppia sottolineatura, e i vettori con una singola sottolineatura

[11] per esempio arrotondare un numero reale a per ottenere il numero di macchina $\overline{a}$ si può vedere come una perturbazione: una piccola variazione dell'input!

Sfruttando la teoria prima introdotta, l'idea è perturbare i dati di ingresso x_1 e x_2, ottenendo $\overline{x}_1$, $\overline{x}_2$. Definiamo quindi gli errori relativi[12] ε_1, ε_2, come

$$\varepsilon_1 = \frac{\overline{x}_1 - x_1}{x_1} \qquad \varepsilon_2 = \frac{\overline{x}_2 - x_2}{x_2}.$$

Mediante semplici operazioni aritmetiche, è possibile riscrivere i valori perturbati degli ingressi come segue:

$$\overline{x}_1 = \overline{x}_1 + x_1 - x_1 = x_1 + \overline{x}_1 - x_1 = x_1\left(1 + \frac{\overline{x}_1 - x_1}{x_1}\right) = x_1(1+\varepsilon_1)$$
$$\overline{x}_2 = \overline{x}_2 + x_2 - x_2 = x_2 + \overline{x}_2 - x_2 = x_2\left(1 + \frac{\overline{x}_2 - x_2}{x_2}\right) = x_2(1+\varepsilon_2).$$

Dal momento che $y = x_1 + x_2$, e che $\overline{y} = \overline{x}_1 + \overline{x}_2$, è possibile scrivere l'errore relativo su $\overline{y}$ come:

$$\frac{y-\overline{y}}{y} = \frac{x_1 + x_2 - (\overline{x}_1 + \overline{x}_2)}{x_1 + x_2} = \frac{x_1 + x_2 - x_1(1+\varepsilon_1) - x_2(1+\varepsilon_2)}{x_1 + x_2} =$$
$$= \frac{x_1 + x_2 - x_1 - x_1\varepsilon_1 - x_2 - x_2\varepsilon_2}{x_1 + x_2} = -\frac{x_1}{x_1 + x_2}\varepsilon_1 - \frac{x_2}{x_1 + x_2}\varepsilon_2.$$

A questo punto, definiamo

$$\kappa_1 \triangleq \left|\frac{x_1}{x_1 + x_2}\right| \qquad \kappa_2 \triangleq \left|\frac{x_2}{x_1 + x_2}\right|,$$

e così si può stimare l'errore relativo su $\overline{y}$ come

$$\left|\frac{y-\overline{y}}{y}\right| \leq \kappa_1\varepsilon_1 + \kappa_2\varepsilon_2 \leq \kappa(|\varepsilon_1| + |\varepsilon_2|),$$

che è stato maggiorato dall'ultima espressione, dove abbiamo definito κ come il massimo tra κ_1 e κ_2 (e quindi è certamente più grande), e i valori assoluti. Riassumendo, abbiamo quindi ricavato che

$$\left|\frac{y-\overline{y}}{y}\right| \leq \kappa\left(\underbrace{\left|\frac{\overline{x}_1 - x_1}{x_1}\right|}_{|\varepsilon_1|} + \underbrace{\left|\frac{\overline{x}_2 - x_2}{x_2}\right|}_{|\varepsilon_2|}\right),$$

con

$$\kappa = \left|\frac{\max\{|x_1|, |x_2|\}}{x_1 + x_2}\right|.$$

L'espressione di κ è scritta in modo tale da mettere immediatamente in evidenza quale sia la situazione di *mal condizionamento*. Infatti, κ, e dunque il fattore di amplificazione degli errori sugli ingressi, è grande quando $x_1 + x_2 \to 0$, ossia quando $x_1 \to -x_2$. Questo, guarda caso, è esattamente il caso in cui si effettua la sottrazione di due numeri molto vicini tra loro: **la cancellazione numerica**! Tutto questo conto, quindi, ha permesso di formalizzare meglio l'idea di cancellazione numerica.

1.6 Stabilità di un algoritmo

Per concludere l'argomento, dobbiamo introdurre un'idea un po' *duale* a quella di condizionamento. Se vogliamo, il *condizionamento* è legato ai *problemi del problema numerico*: a partire dal problema numerico, lo applichiamo -sempre con precisione infinita di calcolo- prima sui dati esatti, poi sui dati approssimati, e, studiando l'errore tra questi due risultati, definiamo il numero di condizionamento come una sorta di *cifra di merito* della *qualità* del problema, facendo finta che esso agisca con precisione infinita.

[12] al fine di semplificare i conti si omette il valore assoluto; questo non riduce la generalità della dimostrazione

Quando si parla di **stabilità dell'algoritmo**, si fa qualcosa di molto diverso. Si parta da $\overline{x}$, quindi dai dati già approssimati, senza porsi ulteriori domande su cosa accadrebbe se si lavorasse con i dati esatti. Poi, ci chiediamo, focalizzandoci esclusivamente su questi $\overline{x}$, cosa succederebbe calcolando il risultato dell'algoritmo in precisione di macchina e in aritmetica esatta. A tal fine, è opportuno ricordare che **un algoritmo è una sequenza finita di operazioni che permette di calcolare l'output di un problema**. Ovviamente, **sequenza finita** è un sinonimo di **precisione finita**: non è possibile ottenere un risultato in aritmetica esatta usando un numero finito di passaggi (si pensi per esempio a sommare $\sqrt{2}$ e π: sarebbe necessario avere infinite operazioni!). In questo senso, dato $\widetilde{y}$ il risultato dell'algoritmo applicato con precisione finita, e $\overline{y}$ quello dell'algoritmo applicato in precisione infinita, viene detto **stabile** un algoritmo tale per cui

$$\frac{|\widetilde{y} - \overline{y}|}{|\overline{y}|} < \varepsilon_{\mathrm{m}}.$$

Bibliografia

[1] A. Quarteroni e F. Saleri, "Introduzione al calcolo scientifico," 3ª edizione, Springer-Verlag Italia, Milano, 2006.

[2] M. Calciati, A. Tibaldi, F. Bertazzi, M. Goano, and P. Debernardi, "Many-valley electron transport in AlGaAs VCSELs," *Semicond. Sci. Technol.*, vol. 32, no. 5, 055007, Apr. 2017.

Capitolo 2

Approssimazione di dati e funzioni

Introduzione

Nell'introduzione avevamo discusso il calcolo dell'integrale definito

$$I = \int_4^7 \mathrm{e}^{-x^2}\,\mathrm{d}x \tag{2.1}$$

e avevamo capito che non ci sono limitazioni *filosofiche* che ci impediscano di calcolarlo: il problema è unicamente la nostra incapacità di scrivere la primitiva in termini di funzioni elementari. Si tratta dunque di una limitazione legata al tipo di procedura di calcolo: cercare in tutti i modi di applicare il teorema fondamentale del calcolo integrale a una primitiva scritta in forma chiusa. Immaginiamo a questo punto di cambiare tattica: se non è possibile integrare una funzione $f(x)$, potremmo cercare di sostituirla con un'altra funzione $\widetilde{f}$, che in qualche modo *le assomigli* o, per usare il titolo della sezione, *la approssimi*. Se le due funzioni si assomigliano, evidentemente anche i loro integrali si assomiglieranno! Un problema differente, ma che comunque è riconducibile a quanto appena descritto, è l'approssimazione di **dati**, ovvero coppie (x_i, y_i) fornite per esempio da misure sperimentali o osservazioni di un qualche tipo.

Purtroppo, tra il dire e il fare c'è sempre di mezzo il mare: approssimare una funzione o un insieme di dati, in effetti, potrebbe essere più difficile di quanto sembri. Per capire quali possono essere le problematiche, guardiamo l'esempio di Fig. 2.1. La curva blu riporta la funzione di Runge

$$f(x) = \frac{1}{1+x^2}, \tag{2.2}$$

disegnata[1] su una griglia costituita da 1001 punti. Volendo cercare di approssimarla, il primo passo è valutare questa funzione in alcuni valori delle ascisse, che chiameremo *nodi*. In questo esempio si è scelto di usare 14 nodi equispaziati, marcati corrispettivamente ai relativi valori della funzione con dei cerchietti neri. Al fine di ottenere quindi un'approssimazione della funzione f, vogliamo utilizzare il **criterio dell'interpolazione**: definire la nostra funzione approssimata $\widetilde{f}$ in modo tale che **passi per ciascuno dei cerchietti**. Ci sono diversi modi per ottenere un obiettivo del genere. Per esempio, la curva verde mostra il risultato di un'**interpolazione con una funzione lineare a tratti**; in parole povere, unire ciascuna coppia di cerchietti adiacenti mediante un segmento di retta. In alcune regioni dell'intervallo mostrato questo approccio sembra essere vincente, dal momento che non si riesce a distinguere la curva verde da quella blu, originale. Tuttavia, questo approccio si dimostra inefficace in regioni dove la funzione varia rapidamente, per esempio in prossimità di $x = 0$, in cui si ha una forma molto *tondeggiante* che viene approssimata mediante un singolo segmento orizzontale. Volendo cercare di rappresentare meglio queste regioni *tondeggianti*, si può immaginare di sostituire questo approccio con un'**interpolazione polinomiale**, ovvero in cui si utilizza come funzione approssimante un polinomio *progettato* in modo tale da passare per i vari nodi di interpolazione; sfruttando quindi il fatto che i polinomi di grado elevato sono già di per sé funzioni rapidamente variabili (si pensi per esempio a una semplice parabola: ha una certa concavità in prossimità del vertice!). La curva rossa, però, dimostra che anche questa scelta si può rivelare problematica: per quanto infatti il polinomio passi per ciascuno dei nodi di interpolazione,

[1] lo script MATLAB® utilizzato per disegnare questa figura è stato riportato in Appendice A.1

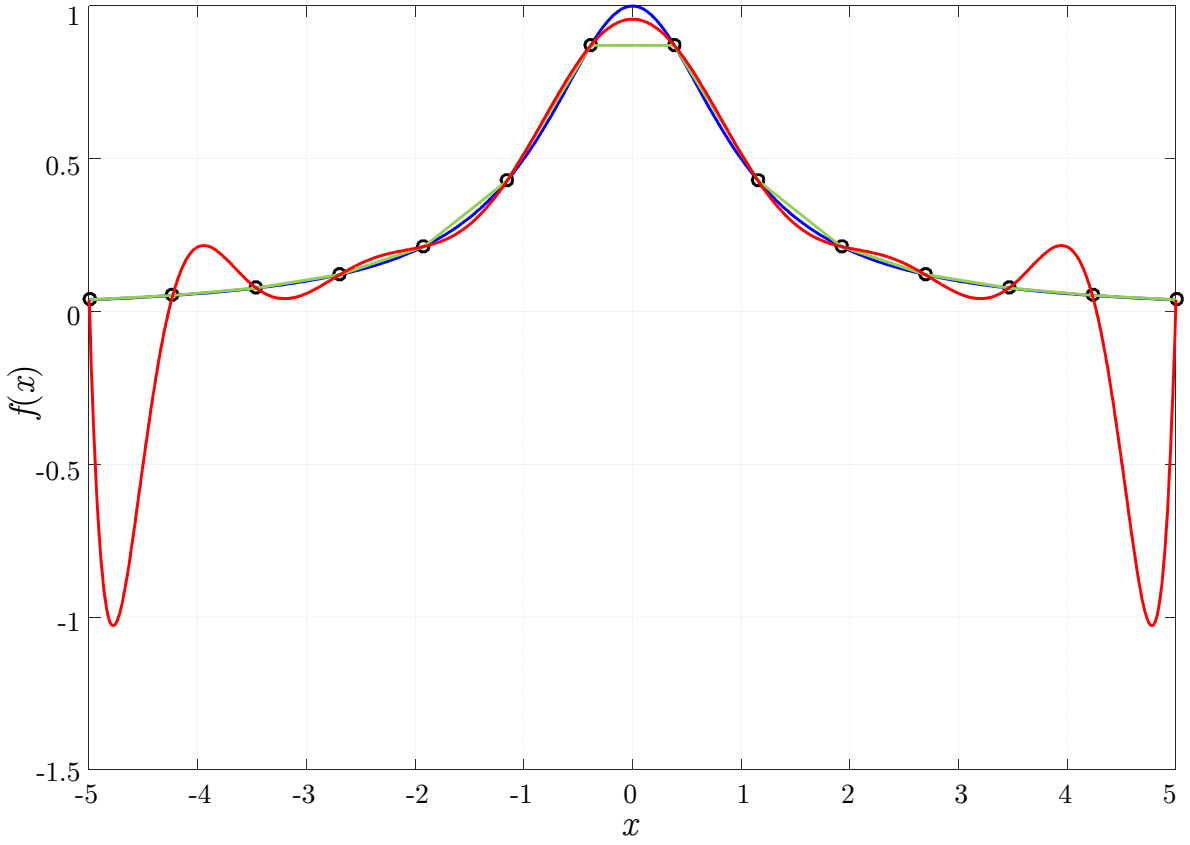

Figura 2.1: Tentativi di interpolazione della funzione di Runge, disegnata con una curva blu. I 14 nodi di interpolazione equidistanti sono stati rappresentati mediante dei marker circolari neri. Il risultato dell'interpolazione con funzione lineare a tratti è rappresentato mediante una curva verde. La curva rossa riporta il polinomio interpolante.

è impossibile dire che *assomigli* alla funzione di partenza! Esso infatti interpola, ma tra un nodo e un altro presenta un comportamento completamente fuori controllo! Nel testo che segue, studieremo due possibili approcci atti a interpolare delle funzioni:

- l'interpolazione polinomiale, ossia, approssimare mediante un **unico polinomio** di grado n la funzione sull'intero intervallo;
- l'interpolazione mediante funzioni polinomiali a tratti (spline), ovvero, usare come interpolanti funzioni di ordine ridotto e raccordate mediante condizioni di regolarità.

2.1 Approssimazione polinomiale

In questa sezione ci concentreremo sul formalizzare quanto anticipato nell'introduzione effettuando approssimazione polinomiale, ossia, cercando di approssimare una funzione f in un intervallo mediante un singolo polinomio; questo non è per esempio il caso della curva verde di Fig. 2.1, dove sono usate segmenti di retta diversi su ciascun intervallo. A tal fine, è opportuno introdurre un po' di notazione. In questo capitolo noi scriveremo un generico polinomio di grado n nella forma

$$c_1 x^n + c_2 x^{n-1} + c_3 x^{n-2} + \ldots + c_n x + c_{n+1}. \tag{2.3}$$

Dire che il polinomio è di grado n implica che esiste un monomio che è elevato a potenza pari a n. Il polinomio è dato da una combinazione lineare di monomi pesati per opportuni coefficienti $\{c_i\}$. In questo testo, si sceglie di nominare c_1 l'indice del polinomio di grado massimo, e c_{n+1} il termine noto del polinomio, ovvero il coefficiente del monomio di grado zero[2]. Il nostro scopo è scegliere un polinomio che sia in grado di approssimare la nostra funzione di partenza f. Questo, in altre parole, significa **scegliere i coefficienti $\{c_i\}$ che permettano di effettuare questa approssimazione; infatti, un polinomio è determinato in modo univoco da questi coefficienti.**

2.1.1 Un esempio di approssimazione polinomiale

Al fine di chiarire di cosa si sta parlando, è opportuno ricordare che lo studente reduce di Analisi Matematica I ha già avuto a che fare, nella propria vita, con una tecnica di approssimazione polinomiale: lo sviluppo di Taylor! In particolare, l'approssimazione di Taylor si prepone come obiettivo trovare una funzione $\widetilde{f}$ che *assomigli* alla f non ovunque, ma solo nelle vicinanze di un certo punto x_0; per questo motivo, il corrispondente polinomio di Taylor si può indicare come Tf_{n,x_0}, specificando il grado n e il punto di sviluppo x_0. Quindi, è lecito scriverlo nella forma

$$f \simeq \widetilde{f} = Tf_{n,x_0} = c_{n+1} + c_n(x-x_0) + c_{n-1}(x-x_0)^2 + \ldots + c_2(x-x_0)^{n-1} + c_1(x-x_0)^n = \sum_{k=0}^{n} c_{n+1-k}(x-x_0)^k. \tag{2.4}$$

Dire che il polinomio di Taylor *deve assomigliare alla funzione solo vicino a un punto*, significa richiedere che f e $\widetilde{f}$, ovvero il polinomio, soddisfino le seguenti condizioni:

$$\begin{aligned} f(x_0) &= \widetilde{f}(x_0) \\ f'(x_0) &= \widetilde{f}'(x_0) \\ f''(x_0) &= \widetilde{f}''(x_0) \\ &\vdots \\ f^{(n)}(x_0) &= \widetilde{f}^{(n)}(x_0), \end{aligned} \tag{2.5}$$

ovvero, *che tutte le derivate siano uguali intorno a $x = x_0$, ignorando cosa capiti lontano da qui.* Imponendo le condizioni (2.5) in (2.4), è possibile ottenere[3]

[2] questa scelta può sembrare del tutto controintuitiva, ma sarà molto più chiara in seguito; volendole dare una qualche giustificazione, si può dire che c_1 è il coefficiente del monomio *più significativo*: quello di grado più elevato, quindi *più importante* da certi punti di vista

[3] la dimostrazione non è qui proposta poiché non strettamente collegata al Calcolo Numerico

$$Tf_{n,x_0} = f(x_0) + \frac{f'(x_0)}{1!}(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \frac{f'''(x_0)}{3!}(x - x_0)^3 + ... + \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n. \quad (2.6)$$

Al fine di chiarire il concetto senza sviluppare troppi conti, si consideri per esempio $x_0 = 0$; in questa situazione, l'espressione (2.6) diventa

$$Tf_{n,0} = \underbrace{f(0)}_{c_{n+1}} + \underbrace{\frac{f'(0)}{1!}}_{c_n} x + \underbrace{\frac{f''(0)}{2!}}_{c_{n-1}} x^2 + \underbrace{\frac{f'''(0)}{3!}}_{c_{n-2}} x^3 + ... + \underbrace{\frac{f^{(n)}(0)}{n!}}_{c_1} x^n, \quad (2.7)$$

in cui è immediato identificare c_{n+1} come il termine noto, c_n come il termine che moltiplica x elevato a potenza 1, c_{n-1} che moltiplica x^2, e così via. Riassumendo, imporre i vincoli (2.5) in (2.4) permette di trovare le espressioni dei coefficienti $\{c_i\}$ elencate in (2.7) e, quindi, di determinare il polinomio approssimante in questione.

2.1.2 Il criterio dell'interpolazione

Come si è già detto, il criterio di Taylor per la determinazione dei coefficienti era: approssimare benissimo la funzione localmente, ossia intorno a un certo punto $x = x_0$, ma fregarsene completamente di cosa capiti lontano da esso. Certamente, una delle conseguenze di approssimare mediante polinomi di Taylor di ordine elevato è anche aumentare l'ampiezza dell'intorno in cui l'approssimazione è valida, ma questo è un *effetto collaterale*, per quanto positivo. Il nostro obiettivo per questa sezione è ottenere, al posto di qualcosa di *mozzafiato* intorno a un singolo punto, un'approssimazione $\widetilde{f}$ della funzione f che sia *decente* in un intervallo abbastanza largo.

A tal fine, ripensiamo un momento alle condizioni (2.5); richiedere qualcosa di *decente* può voler dire per esempio richiedere che solo la prima delle condizioni, ossia l'uguaglianza della funzione approssimata $\widetilde{f}$ e della funzione di partenza f, sia soddisfatta. Però, al fine di garantire che questa condizione venga soddisfatta su tutto un intervallo, possiamo decidere di partizionare il nostro intervallo in $n+1$ punti $\{x_i\}_{i=1,...,n+1}$ e, per ciascuno di essi, valutare la funzione $y_i = f(x_i)$. Quindi, perché questa condizione di *decenza* valga su tutto l'intervallo, richiederemo che

$$\widetilde{f}(x_i) = f(x_i) = y_i, \qquad i = 1, ..., n+1.$$

Questo criterio è detto **criterio dell'interpolazione**: anziché imporre condizioni sulle derivate, si chiede che, una volta partizionato l'intervallo di interesse in $n+1$ nodi, la funzione approssimante $\widetilde{f}$ sia uguale alla funzione di partenza f in ciascuno dei nodi, ossia, che **passi per essi**[4]. Questo per esempio è il caso della curva rossa di Fig. 2.1, che, nonostante tutti i problemi che può esibire, passa per ciascuno dei punti marcati coi cerchietti neri.

2.2 Interpolazione polinomiale

2.2.1 Unicità (ed esistenza?) del polinomio interpolante

Al fine di introdurre l'interpolazione polinomiale, consideriamo il seguente teorema.

Assegnati $n+1$ dati (x_i, y_i), $i = 1, ..., n+1$, esiste uno e un solo polinomio $p_n(x)$ di grado minore o uguale a n interpolante i dati assegnati, ovvero soddisfacente le condizioni di interpolazione

$$p_n(x_i) = y_i, \qquad i = 1, ..., n+1. \quad (2.8)$$

Cercando di parafrasare il testo del teorema, questo ci garantisce che è sempre possibile trovare un polinomio $p_n(x)$ che sia in grado di passare per i punti (x_i, y_i). Il teorema però ci dà anche la garanzia che non solo questo polinomio esiste, ma anche che è unico, ossia, che non possono esistere due polinomi in grado di passare attraverso questi punti.

Questo è un teorema di esistenza e unicità, quindi la dimostrazione si articola in due parti: quella di unicità, e quella di esistenza. Per quanto riguarda l'esistenza, essa si può dimostrare banalmente trovando un polinomio che soddisfi la condizione di interpolazione (2.8). A questo fine, più avanti mostreremo

[4] si noti che ovviamente questo approccio è applicabile, senza alcuna modifica, anche al caso in cui le $n+1$ coppie di valori (x_i, y_i) vengano non da una funzione matematica, ma da un esperimento di laboratorio o generici dati

un modo di costruire polinomi di questo tipo. Per quanto riguarda l'unicità, possiamo procedere per assurdo: cerchiamo di effettuare un ragionamento corretto applicato a un'ipotesi scorretta, e questo dovrebbe prima o poi portarci alla contraddizione di un principio primo. Visto che vogliamo verificare l'unicità, ipotizziamo per assurdo che esistano due polinomi interpolanti diversi tra loro, $p_n(x)$ e $q_n(x)$. Se entrambi sono interpolanti, allora entrambi devono soddisfare, per $i = 1, ..., n+1$, le condizioni di interpolazione

$$p_n(x_i) = y_i$$
$$q_n(x_i) = y_i.$$

A questo punto, costruiamo la differenza di questi polinomi, ovvero

$$p_n(x) - q_n(x);$$

questa differenza:

- è un polinomio, dal momento che la somma/sottrazione di polinomi è ancora un polinomio;
- è di grado al più n, dal momento che la somma/sottrazione di polinomi al più di grado n non può certamente far crescere in grado;
- non è nullo, dal momento che per ipotesi stiamo dicendo che $p_n(x)$ deve essere diverso da $q_n(x)$;
- ha $n+1$ zeri; infatti, se ci ricordiamo le ipotesi sulle condizioni di interpolazione appena scritte,

$$\underbrace{p_n(x_i)}_{y_i} - \underbrace{q_n(x_i)}_{y_i} = 0, \qquad i = 1, ..., n+1.$$

Tuttavia, abbiamo appena trovato un paradosso: un polinomio di grado n ha n radici, come suggerisce il teorema fondamentale dell'algebra; queste possono essere reali o complesse, ma devono essere esattamente pari a n. Dal momento che un ragionamento corretto applicato all'ipotesi iniziale ci ha portato a una contraddizione, il polinomio interpolante deve essere unico.

Interpretazione geometrica del teorema di esistenza e unicità del polinomio interpolante

Il teorema che abbiamo appena enunciato e dimostrato è in realtà la generalizzazione di un concetto molto famoso.

> «Dati due punti distinti su un piano, per essi passa una e una sola retta.»

In effetti, quando abbiamo due punti, $n+1 = 2$, e quindi $n = 1$. Come ben noto, una retta si può scrivere come un polinomio di primo grado nella forma

$$f(x) = mx + q.$$

Questo, e il fatto che per due punti passa una e una sola retta, sono dimostrati dal teorema che stiamo studiando. Ma c'è dell'altro!

> «Dati tre punti distinti e non collineari su un piano, per essi passa una e una sola parabola.»

Di nuovo, stiamo dicendo più o meno la stessa cosa! Se ho $n+1 = 3$, allora $n = 2$, e quindi la forma del generico polinomio di grado 2 è

$$y = ax^2 + bx + c,$$

che è proprio l'equazione della parabola. Il teorema in realtà prevede anche il caso di tre punti collineari; infatti, il polinomio interpolante è unico, ma ha grado **al più** pari a n; se quindi i tre punti sono collineari, il polinomio interpolante sarà ancora una retta!

2.2.2 Rappresentazione monomiale del polinomio di interpolazione

La sezione precedente propone argomentazioni riguardo l'unicità del polinomio interpolante, ma non fornisce alcuna indicazione né riguardo a come *scriverlo*, né tantomeno a come poi si faccia *di fatto* a ottenere; inoltre, si è detto che questo polinomio esiste, ma non si è ancora dimostrato perché.

Prima di tutto, concentriamoci sul problema di *come si fa a scrivere* questo polinomio. Volendo usare una terminologia adeguata, più che di *scrivere* dovremmo parlare di *rappresentare* il polinomio. E, a tal scopo, riprendiamo la **rappresentazione monomiale** già introdotta in (2.3):

$$p_n(x) = c_1 x^n + c_2 x^{n-1} + \ldots + c_{n-1} x^2 + c_n x + c_{n+1}, \tag{2.9}$$

dove i $\{c_k\}$ sarebbero, se questo ipotetico polinomio esistesse, i suoi coefficienti, **che lo determinano completamente**.

A questo punto è possibile proporre una *prima* dimostrazione dell'esistenza del polinomio interpolante: esiste un comando MATLAB® in grado di calcolare i coefficienti di (2.9). Se questo comando calcola questi coefficienti, vuol dire che il polinomio deve necessariamente esistere ;-). Segue un esempio di codice che utilizza questi comandi.

```
clear
close all
clc

x = [0  1 2 1/2];              % definisco i nodi in cui sono definiti i dati da interpolare...
y = [1 -1 1   2];              % ...e i dati da interpolare

% il polinomio di interpolazione DEVE avere grado n, a partire da n+1 nodi.
c = polyfit(x,y,length(x)-1);  % c sono i coefficienti nella rappresentazione monomiale
z = linspace(min(x),max(x));   % z e` la griglia "fine" sulla quale andiamo a vedere l'interpolato...
p = polyval(c,z);              % vado a valutare il polinomio di coefficienti c nella griglia z

plot(x,y,'ro',z,p,'b')         % disegno i nodi di interpolazione, cerchi rossi, e l'interpolante
```

Può essere di interesse investigare un po' meglio come agisce questo codice. In particolare se, dopo averlo eseguito, chiediamo nella command window di visualizzare il contenuto della variabile `c`, otteniamo

```
>> c
c =
    6.6667  -18.0000    9.3333    1.0000
>>
```

Come vediamo, il codice ha 4 nodi/dati di interpolazione, e si richiede che il polinomio in questione abbia grado pari a $4-1=3$ (infatti il comando `length` restituisce la lunghezza di un vettore). Quindi, un polinomio di grado 3 ha 4 coefficienti: c_1, c_2, c_3, c_4. Il vettore `c` appena stampato a schermo contiene proprio questi coefficienti[5]. Il comando `polyval` permette, una volta noti i coefficienti `c` calcolati grazie a `polyfit`, di calcolare il corrispettivo polinomio interpolante in una griglia di punti più fitta, che in questo esempio è chiamata `z`. I coefficienti calcolati con `polyfit` sono calcolati secondo la forma monomiale (2.9); questo è il motivo per cui viene usata questa notazione. In effetti, il comando

```
p = polyval(c,z);
```

si potrebbe rimpiazzare con

```
c1 = c(1);
c2 = c(2);
c3 = c(3);
c4 = c(4);
p = c1.*z.^3 + c2.*z.^2 + c3.*z + c4;
```

Al fine di non appesantire eccessivamente il testo, l'Appendice A.1 riporta alcuni altri esempi di script MATLAB® atti a calcolare il polinomio interpolante per alcune funzioni[6]. Si può notare che, a volte, eseguendo script di questo genere, possa apparire il messaggio

[5] come faccia il comando polyfit a produrre questi coefficienti sarà argomento del prossimo capitolo del corso ;-)

[6] in particolare, il secondo di questi esempi sfrutta alcune sintassi più avanzate che sarebbe opportuno conoscere; viene presentato un codice molto commentato che descrive ogni passaggio.

```
Warning: Polynomial is badly conditioned. Add points with distinct X values,
reduce the degree of the polynomial, or try centering and scaling as described
in HELP POLYFIT.
> In polyfit (line 75)
  In Lez03_EsempioInterpAvanzato (line 35)
```

Ossia, MATLAB® denuncia problemi di **cattivo condizionamento**[7]. Quando impareremo come MATLAB® calcola questi coefficienti, capiremo esattamente da dove nasce questo cattivo condizionamento.

2.2.3 Rappresentazione di Lagrange del polinomio di interpolazione

La precedente sezione ha proposto, come argomentazione dell'esistenza del polinomio interpolante, il fatto che MATLAB® *sa calcolarlo*. Volendo però essere un po' più seri e proporre un'argomentazione più solida e rigorosa, possiamo fare riferimento a uno dei risultati più importanti di Lagrange[8]. Prima di chiamarlo in causa, però, è opportuno che un concetto sia assolutamente chiaro: **la rappresentazione monomiale e la rappresentazione di Lagrange sono due modi diversi di scrivere esattamente lo stesso polinomio**. Come già detto e dimostrato, il polinomio interpolante è unico; nonostante si possa scrivere in due modi apparentemente molto diversi, **per un insieme di dati e nodi, il polinomio interpolante è lo stesso a prescindere da tutti i modi in cui possiamo scriverlo**. Per chiarire ulteriormente il concetto, proponiamo un esempio banale: se uno scrive

$$(x-3)(x+2),$$

o scrive

$$x^2 - x - 6,$$

in verità sta scrivendo la stessa, identica cosa: la prima è una **rappresentazione fattorizzata**, la seconda è una **rappresentazione estesa**, ma il polinomio **è lo stesso**.

Al fine di spiegare la rappresentazione di Lagrange, probabilmente il metodo più facile è basarsi su un caso specifico. Si consideri un insieme di 4 nodi, che potrebbero essere per esempio quelli dello script riportato sopra:

$$x_i = \{x_1, x_2, x_3, x_4\} = \left\{0, 1, 2, \frac{1}{2}\right\}.$$

Si noti che non è necessario né che siano in ordine, né che siano equispaziati. Per ciascuno di questi nodi, è possibile definire un polinomio. In questo caso, il primo di questi polinomi è:

$$\ell_1(x) = \frac{(x-x_2)(x-x_3)(x-x_4)}{(x_1-x_2)(x_1-x_3)(x_1-x_4)}.$$

Cerchiamo di capire come è costruito: sia al numeratore, sia al denominatore, si può vedere il prodotto di 3 fattori (3 sarebbe il numero di nodi meno 1). La variabile x appare solo al numeratore, mentre il denominatore contiene solo i nodi[9]. Il numeratore e il denominatore hanno una forma estremamente simile, eccetto che la variabile x è rimpiazzata con x_i, dove i è l'indice del polinomio. Infine, si nota chiaramente che x_1 al numeratore non appare. Avendo capito come si costruisce questo polinomio, è possibile intuire che i successivi saranno:

$$\ell_2(x) = \frac{(x-x_1)(x-x_3)(x-x_4)}{(x_2-x_1)(x_2-x_3)(x_2-x_4)},$$

$$\ell_3(x) = \frac{(x-x_1)(x-x_2)(x-x_4)}{(x_3-x_1)(x_3-x_2)(x_3-x_4)},$$

$$\ell_4(x) = \frac{(x-x_1)(x-x_2)(x-x_3)}{(x_4-x_1)(x_4-x_2)(x_4-x_3)}.$$

Vale sempre la regola generale: per il polinomio ℓ_i, al numeratore ci sono tutti i prodotti $(x-x_j)$, dove j assume tutti i valori tranne i, così come il denominatore contiene tutti i prodotti (x_i-x_j), tranne $i=j$.

[7] ricorda nulla?

[8] data l'importanza del personaggio, riporto qualche pillola di storia in Appendice A.2.

[9] ovviamente: altrimenti, se x fosse sia al numeratore sia al denominatore, non sarebbe un polinomio ma una funzione razionale!

Concentriamoci ancora su $\ell_1(x)$; possiamo notare che, se valutiamo ℓ_1 in $x = x_1$, esso vale:

$$\ell_1(x_1) = \frac{(x_1 - x_2)(x_1 - x_3)(x_1 - x_4)}{(x_1 - x_2)(x_1 - x_3)(x_1 - x_4)} = 1.$$

Ovviamente, questa è una proprietà generale di questi polinomi: $\ell_i(x_i) = 1$. Se invece provassimo a valutare $\ell_1(x)$ in $x = x_3$, avremmo

$$\ell_1(x_3) = \frac{(x_3 - x_2)(x_3 - x_3)(x_3 - x_4)}{(x_1 - x_2)(x_1 - x_3)(x_1 - x_4)} = 0.$$

Allo stesso modo, $\ell_i(x_j)$, per $i \neq j$, è sempre uguale a 0.

Note tutte queste proprietà, possiamo scrivere

$$p_n(x) = \sum_{j=1}^{n+1} y_j \ell_j(x).$$

Il nome $p_n(x)$ non è casuale, dal momento che questo è esattamente la rappresentazione del polinomio interpolante che stavamo cercando. Infatti, scrivendolo più esplicitamente per il caso in esame, abbiamo che:

$$p_n(x) = y_1\ell_1(x) + y_2\ell_2(x) + y_3\ell_3(x) + y_4\ell_4(x).$$

Partendo da tutto ciò che abbiamo scritto, è molto semplice vedere che questo è un polinomio interpolante. Infatti, se andiamo a verificare le condizioni di interpolazione, notiamo che

$$p_n(x_1) = y_1 \underbrace{\ell_1(x_1)}_{1} + y_2 \underbrace{\ell_2(x_1)}_{0} + y_3 \underbrace{\ell_3(x_1)}_{0} + y_4 \underbrace{\ell_4(x_1)}_{0} = y_1$$

$$p_n(x_2) = y_1 \underbrace{\ell_1(x_2)}_{0} + y_2 \underbrace{\ell_2(x_2)}_{1} + y_3 \underbrace{\ell_3(x_2)}_{0} + y_4 \underbrace{\ell_4(x_2)}_{0} = y_2$$

$$p_n(x_3) = y_1 \underbrace{\ell_1(x_3)}_{0} + y_2 \underbrace{\ell_2(x_3)}_{0} + y_3 \underbrace{\ell_3(x_3)}_{1} + y_4 \underbrace{\ell_4(x_3)}_{0} = y_3$$

$$p_n(x_4) = y_1 \underbrace{\ell_1(x_4)}_{0} + y_2 \underbrace{\ell_2(x_4)}_{0} + y_3 \underbrace{\ell_3(x_4)}_{0} + y_4 \underbrace{\ell_4(x_4)}_{1} = y_4.$$

Inoltre: è evidente che $\ell_i(x)$ è un polinomio di grado 3, quindi p_n è davvero il polinomio interpolante. In particolare, questi $\ell_i(x)$ vengono detti **polinomi fondamentali di Lagrange.**

2.2.4 Convergenza del polinomio di interpolazione

Ricordiamoci sempre che l'obiettivo di questa sezione del testo è cercare di ottenere, a partire da una funzione $y = f(x)$, un unico polinomio in grado di approssimarla in maniera accettabile su un intervallo $[a, b]$. Per fare questo, i coefficienti che definiscono questo polinomio devono essere ricavati imponendo il passaggio per un certo insieme di punti da noi scelti[10]. Nell'introduzione abbiamo però visto una situazione in cui, nonostante il polinomio intersecasse davvero la funzione in corrispondenza dei nodi, questo non assomigliava quasi in alcun modo alla funzione di partenza, in quanto presentava sovraelongazioni assolutamente innaturali rispetto al comportamento atteso. A questo punto il lettore, avendo visto questo e tenendo presente l'obiettivo finale, potrebbe chiedersi:

> «Abbiamo una qualche garanzia che non vengano fuori queste sovraelongazioni, e che quindi il nostro polinomio non sia solo interpolante, ma anche *rassomigliante* alla funzione di partenza?»

[10] ricordiamo sempre che questo problema è equivalente a richiedere il passaggio per coppie (x_i, y_i) che costituiscono un insieme di dati derivanti da un esperimento di un qualche genere

La risposta è no: per ciò che è stato scritto fino a questo momento, non sono state fornite garanzie di alcun tipo. Però, per poterlo dire, sarebbe opportuno cercare di dare un po' di formalità in più a queste affermazioni.

Errore di interpolazione

Fino a questo momento ci siamo solo concentrati sul cercare di soddisfare la condizione di interpolazione, sperando che come conseguenza naturale si avrebbe avuto un polinomio *rassomigliante* alla funzione di partenza. Per capire cosa sta succedendo, però, dobbiamo cercare di esprimere in *matematichese* questo concetto di *rassomigliare*, per poi poter quantificare davvero quanto due funzioni siano effettivamente simili. Partiamo dalla nostra $f(x)$ originale, e immaginiamo di approssimarla mediante un polinomio interpolante $p_n(x)$. Il fatto di approssimare qualcosa a qualcos'altro comporta **sbagliare**, commettere un **errore**; più piccolo è questo errore, più la funzione $f(x)$ e il polinomio $p_n(x)$ saranno somiglianti. Al fine di quantificare questo errore, definiamo la funzione **errore di interpolazione**

$$E_n(x) = f(x) - p_n(x). \tag{2.10}$$

Si può notare che questa funzione, nei nodi di interpolazione $\{x_i\}$, si annulli; infatti, dal momento che il nostro polinomio soddisfa le condizioni di interpolazione

$$p_n(x_i) = f(x_i),$$

allora certamente la funzione varrà zero in tali punti[11]. Un altro dettaglio è: se la funzione $f(x)$ fosse un polinomio di grado minore o uguale a n, grado del polinomio interpolante, allora $E_n(x)$ sarebbe identicamente nulla. Infatti, **il polinomio interpolante di un polinomio è il polinomio stesso**: per il teorema di unicità esposto precedentemente, il polinomio è certamente unico.

Norma infinito e convergenza uniforme

Abbiamo definito l'errore di interpolazione (2.10) come una funzione ma, insomma, non è facilissimo *quantificare* questo errore. In effetti, questa funzione ci fornisce, per ogni punto della griglia *fine* in cui andiamo a valutare il polinomio di interpolazione, il valore dell'errore commesso. Immaginiamo per esempio di avere, per la stessa funzione, due polinomi di interpolazione di grado diverso, e di voler capire quale di questi sia *migliore* dell'altro. Non ci serve a nulla sapere punto per punto quale sia il migliore: ci servirebbe avere un unico indicatore che ci dia un'idea di come sia l'errore sull'intero intervallo!

In questo senso, dobbiamo cercare una maniera per ricavare, a partire dalla funzione $E_n(x)$, un singolo numero che *rappresenti* l'intera funzione. Ogni volta che vogliamo produrre un numero a partire da una funzione o da un vettore, si ricorre al concetto di **norma**[12]. Per questo caso, è utile definire la norma infinito di una funzione g su un intervallo $[a, b]$ come

$$\|g\|_\infty = \max_{x\in[a,b]} |g(x)|.$$

In altre parole, la norma infinito $\|.\|_\infty$ della funzione $g(x)$ è ottenuta cercando il massimo assoluto del modulo della stessa, nell'intervallo $[a, b]$. Volendo per esempio calcolare la norma infinito del vettore

$$\underline{b} = \begin{bmatrix} 1 & 3 & -10 & 9 & 9 \end{bmatrix},$$

questa sarà semplicemente il massimo dei valori assoluti delle componenti e, quindi, 10.

Cos'ha a che vedere tutta questa storia con il fatto che il polinomio interpolante $p_n(x)$ *assomigli* o meno alla nostra funzione? Beh, per capirlo, dovremmo semplicemente applicare questa idea della norma infinito alla nostra funzione errore $E_n(x)$. In effetti, se calcoliamo

$$\|E_n\|_\infty = \max_{x\in[a,b]} |f(x) - p_n(x)|,$$

[11] in effetti, è possibile che in letteratura le procedure di interpolazione vengano anche chiamate *point matching*, ossia, soddisfare esattamente una condizione in un insieme di punti.

[12] il concetto di norma verrà introdotto con più calma e approfondito nelle prossime sezioni; per ora, si immagini semplicemente che esso costituisce una sorta di parametro che quantifica la *lunghezza* o l'*ampiezza* di un vettore/funzione

stiamo valutando il **massimo errore** che commettiamo quando effettuiamo l'interpolazione. Ovviamente, se il **massimo** degli errori sull'intervallo è piccolo, a maggior ragione tutti gli altri saranno ancora più piccoli!!! Per questo motivo, più piccola sarà la norma infinito dell'errore, più simili saranno le funzioni!

Chiarito questo, propongo una domanda: se calcolassimo

$$\lim_{n\to\infty} \|E_n\|_\infty = \lim_{n\to\infty} \max_{x\in[a,b]} |f(x) - p_n(x)|,$$

questo tenderebbe a zero?

Prima di tutto, cosa significa quel limite? Beh, a ogni valore di n corrisponde un diverso polinomio interpolante $p_n(x)$ avente grado n. Per ottenerlo, però, è necessario utilizzare un diverso numero di nodi di interpolazione, dal momento che $p_n(x)$ è ottenuto interpolando esattamente $n+1$ nodi. Quindi, l'idea di calcolare questo limite, è: faccio crescere i nodi di interpolazione, in modo da *campionare* la curva in punti sempre più vicini tra loro, che tendono dunque ad avvicinarsi. Se faccio questo, il polinomio interpolante tende ad assomigliare di più alla curva di partenza, oppure no? E in particolare, se immaginiamo che $n \to \infty$, cioè, se immaginiamo ipoteticamente di poter far crescere fino all'infinito i nodi di interpolazione e il grado del polinomio interpolante, le due funzioni si assomiglieranno sempre di più, o no? **No. La risposta è NO.** Volendone una dimostrazione, l'Appendice (A.3.1) riporta un esempio di codice MATLAB® che, se eseguito con i parametri proposti, permette di vedere chiaramente che al crescere del grado del polinomio di interpolazione, l'errore non solo non tende a zero, ma tende a esplodere! Assurdamente, la funzione studiata in questo codice è $C^{(\infty)}$ e, di conseguenza, estremamente liscia; ciononostante, non si riesce a interpolare!

A rincarare ulteriormente la dose è il seguente teorema: data una **qualunque successione di nodi** distinti, tutti situati in $[a,b]$, esiste sempre una **funzione continua** $f(x)$ in $[a,b]$ che, interpolata su quei nodi, genera una successione di polinomi di interpolazione $\{p_n(x)\}$ **non uniformemente convergente** a $f(x)$ in $[a,b]$.

Di solito si parla di *teoremi costruttivi*, ma questo se vogliamo è **distruttivo**. Questo teorema garantisce che, se prendo un generico insieme di nodi, ci sarà sempre una qualche funzione continua tale per cui i polinomi interpolanti non le potranno mai assomigliare! Cioè, **non abbiamo garanzie!** E non stiamo parlando di funzioni assurde contenenti discontinuità, singolarità, salti: parliamo di funzioni **continue!** Notate inoltre che questo è proprio uno di quei problemi che a me vien da definire *filosofici*: questo non c'entra niente con la procedura con la quale stiamo valutando i coefficienti dei polinomi di interpolazione (che sia basata su `polyfit` o sui polinomi di Lagrange), quindi con ipotetici problemi di condizionamento: **questo è un limite dell'operazione di interpolazione!** È un limite intrinseco del tentare di interpolare, non c'entra nulla con i problemi di aritmetica, cancellazione o condizionamento: **è un problema filosofico!** Ma allora...

> «Ma allora che senso ha studiare tutte queste cose, se poi non abbiamo la minima garanzia che funzionino?!?!?!???!?!»

In effetti, non solo non abbiamo la garanzia che funzionino, ma il teorema ci dice che abbiamo la garanzia che **non funzionino.** Tuttavia, se potessimo in qualche modo garantire delle ipotesi un po' più forti di quelle del teorema, potremmo pensare di ottenere di più. In questo senso, ci conviene agire su due fronti.

- Dobbiamo garantire qualcosina in più sulla regolarità della funzione di partenza. In particolare, se questa funzione fosse solo continua, ma con derivate discontinue, non potremmo fare nulla. Dobbiamo quindi chiedere che f sia almeno derivabile una volta.
- Non possiamo scegliere una **qualunque** successione di nodi: ci conviene scegliere delle **particolari** successioni di nodi. Abbiamo provato sulla nostra pelle che usare nodi equispaziati non sia una scelta particolarmente da pane e volpe.

Partiamo dal secondo punto. Una scelta decisamente più sagace è quella di utilizzare i nodi di Chebyshev-Lobatto

$$z_i = -\cos\left(\frac{(i-1)\pi}{n}\right) \in [-1,1], \qquad i = 1,\dots,n+1, \tag{2.11}$$

o di Chebyshev

$$z_i = -\cos\left(\frac{(2i-1)\pi}{2(n+1)}\right) \in [-1,1], \qquad i = 1, ..., n+1. \tag{2.12}$$

Come sempre, la n utilizzata in queste espressioni è il grado del polinomio interpolante la funzione in $n+1$ nodi. Tuttavia, invece di essere equispaziati, saranno distribuiti in maniera non uniforme.

Detto questo, le noie non sono finite: se andiamo a valutare (2.11) o (2.12), vediamo che i nodi sono definiti nell'intervallo $z \in [-1,1]$: questi sono il risultato di un coseno con un certo argomento! Questo non va bene, perché, in generale, una funzione va studiata su un intervallo $x \in [a,b]$! Per poter usare questi nodi sul nostro intervallo, è necessario *mapparli* su di esso: ci serve definire una funzione $x = m(z)$ che trasformi i nodi nel formato $\{z_i\}$ al formato $\{x_i\}$. Inoltre, abbiamo capito che la spaziatura relativa tra i vari nodi è fondamentale: equispaziati non van bene, ma con questa spaziatura *di Chebyshev* sì. Quindi, la trasformazione per passare da z a x deve essere tale da non perturbare la spaziatura relativa e quindi, l'unica possibilità, è **la retta**: la curva che mantiene uguali i rapporti, e quindi le spaziature relative.

Come mi insegnava la prof. della terza liceo, la retta passante per due punti che permette di trasformare l'intervallo $[z_1, z_2] = [-1,1]$ nell'intervallo $[x_1, x_2] = [a,b]$ soddisfa l'equazione

$$\frac{x-x_1}{x_2-x_1} = \frac{z-z_1}{z_2-z_1} \Longrightarrow \frac{x-a}{b-a} = \frac{z-(-1)}{1-(-1)} \Longrightarrow x-a = \frac{b-a}{2}(z+1).$$

Per dare un'interpretazione alternativa a questa formula, la si può anche leggere nel seguente modo, ripensando alle *proporzioni*:

> «La distanza di un generico punto x rispetto all'estremo sinistro, a, sta alla lunghezza dell'intero intervallo, $b-a$, come la distanza di un generico punto z nell'intervallo di partenza rispetto al suo estremo sinistro, cioè -1, sta alla lunghezza dell'intervallo di partenza, cioè 2.»

La formula appena descritta ci permette di scrivere, finalmente:

$$x = \frac{b-a}{2}z + \frac{b+a}{2}. \tag{2.13}$$

Al fine di mostrare un esempio di programmazione, l'Appendice A.3.2 mostra l'implementazione dello stesso esempio di Appendice A.3.1, usando però nodi di Chebyshev. A questo punto, possiamo enunciare un altro teorema.

Sia $\{p_n(x)\}$ la successione dei polinomi interpolanti $f(x)$ nei nodi di Chebyshev-Lobatto oppure di Chebyshev. Se $f \in C^{(k)}([a,b])$, $k \geq 1$, allora

$$\|f - p_n\|_\infty = \mathcal{O}\left(\frac{\log n}{n^k}\right), \quad n \to \infty.$$

Cosa significa tutto questo? Se si usano i nodi di Chebyshev o di Chebyshev-Lobatto, l'errore in norma infinito tende a 0 e, più la funzione è regolare, ovvero più derivate continue essa ha, e più questo errore decresce rapidamente, al crescere del grado del polinomio interpolante. Quindi, l'uso di questi particolari nodi di interpolazione ha ripristinato l'intuizione per cui più una funzione è *liscia*, regolare, e più facile deve essere interpolarla. Un appunto è però doveroso: questo teorema non è in alcun modo in conflitto con il teorema *distruttivo* enunciato precedentemente. Infatti, le garanzie di convergenza che esso ci fornisce sono contenute nel dettaglio $k \geq 1$, ovvero, nella richiesta di avere a che fare con funzioni almeno derivabili. Se infatti la funzione fosse solamente continua, avremmo $k = 0$ e, quindi, un tasso[13] $\mathcal{O}(\log n)$ che, quindi, all'infinito diverge, esplode: tende a ∞.

2.3 Interpolazione polinomiale a tratti: spline

2.3.1 Introduzione alle spline

Fino a questo momento studiando questo argomento ci siamo concentrati sull'interpolazione polinomiale, ossia sull'idea di approssimare una funzione f definita su un intervallo $[a,b]$ mediante un **unico**

[13] la notazione *ogrande* $\mathcal{O}$ indica il comportamento asintotico, quindi per argomento tendente a infinito, dell'oggetto di cui ci si sta occupando

polinomio di grado n. A questo fine, abbiamo partizionato il nostro intervallo in $n+1$ punti e, mediante procedure analitiche (rappresentazione di Lagrange) o numeriche (il comando `polyfit`) abbiamo ricavato i coefficienti del polinomio.

Tuttavia, esiste un secondo principio che possiamo applicare al fine di approssimare una funzione. Immaginiamo, proprio come prima, di aver partizionato l'intervallo $[a,b]$ in n sotto-intervalli[14]. Al posto di utilizzare un unico polinomio definito su tutto l'intervallo, la scelta che discuteremo in questa sezione sarà **utilizzare un polinomio diverso e di grado basso su ciascun sotto-intervallo**. Questo significa, in altre parole, utilizzare come approssimante una **funzione polinomiale a tratti**: la funzione di partenza è approssimata da un polinomio di grado d **caratterizzato da coefficienti tra loro diversi su ciascun intervallo**; il grado massimo d del polinomio è lo stesso in tutti i sotto-intervalli. A questo punto, lo studente impaziente, leggendo questa definizione, potrebbe pensare

> «C'è qualcosa che mi puzza: la grande idea del giorno è usare tanti polinomi diversi su ciascun sotto-intervallo e che non c'entrano nulla l'uno con l'altro. Bah. Almeno prima avevamo un unico polinomio che almeno sapevamo essere liscio su tutto l'intervallo $[a,b]$. Cioè, ma chi ci dice che ora sui punti di raccordo tra due polinomi, questi non facciano qualcosa di pazzo?!»

In effetti, l'idea che vorremmo presentare è ancora incompleta. Con riferimento ai dubbi del nostro studente impaziente, è sì vero che vogliamo usare su ciascun sotto-intervallo polinomi diversi, con coefficienti diversi, ma non ricordo di aver scritto che **debbano essere indipendenti l'uno dall'altro**. Ciò che in effetti distingue un banale insieme di funzioni polinomiali a tratti dalle funzioni che intendiamo definire è proprio **l'imposizione di condizioni di raccordo tra coppie di polinomi adiacenti**. Queste condizioni di raccordo fanno dunque sì che i polinomi siano diversi ma che, d'altra parte, ciascun polinomio *cominci e termini avendo la consapevolezza* di quello che sta per fare il precedente/successivo. Introdurre questo rapporto tra polinomi adiacenti permette alle funzioni che useremo, dette **spline**, da un lato una certa *libertà di movimento*, poiché ciascun sotto-intervallo ha un polinomio diverso, ma anche una certa *armonia*, poiché ciascuna funzione *sa cosa sta facendo* la funzione successiva.

Il nostro compito adesso è cercare di tradurre in un linguaggio più formale l'idea appena espressa. Volendo una definizione intermedia tra il discorso appena fatto e una formulazione rigorosa, si definisce una spline di ordine d interpolante f nei nodi $\{x_i\}$ una funzione $S_d(x)$ polinomiale a tratti con proprietà di regolarità nei punti di raccordo. Più nel dettaglio, $S_d(x)$ deve soddisfare le condizioni ora riportate.

1. $S_d(x)$ è un polinomio di grado (al più) pari a d su ciascun sotto-intervallo $[x_i, x_i+1]$, per $i = 1, ..., n$.
2. La spline deve essere interpolante, quindi soddisfare le condizioni di interpolazione

$$S_d(x_i) = y_i, \qquad i = 1, ..., n+1.$$

3. La derivata k-esima $S_d^{(k)}(x)$, per $k = 0, 1, ..., d-1$, è una funzione continua nell'intero intervallo $[a,b]$.

2.3.2 Spline di ordine 1

Prima di passare al piatto forte, è opportuno introdurre quello che sicuramente è l'esempio *più semplice* di spline: la spline lineare, ovvero, $d = 1$. Si noti che questa è esattamente la stessa cosa che fa il comando `plot` ogni volta che utilizziamo un numero di nodi molto basso per disegnare una funzione: si ottiene un grafico *squadrato*, che in effetti altro non è che la spline lineare *su pochi punti* della funzione. In altre parole, l'approssimazione consiste nel rappresentare la funzione di partenza unendo mediante segmenti di retta i vari nodi di interpolazione.

La spline lineare $S_1(x)$ soddisfa le condizioni presentate al termine della Sezione 2.3.1: $S_1(x)$ è continua, ma non derivabile, dal momento che $d = 1$ e tutte le derivate fino alla $(d-1)$-esima sono continue (quindi, fino alla derivata k-esima con $k = 0$, che significa continuità e nient'altro). D'altra parte se approssimiamo la funzione su ciascun intervallo mediante una retta, allora da un intervallo a un altro le rette dovranno cambiare pendenza, altrimenti non sarà possibile che seguano il profilo della funzione! Ma quindi, la derivata sarà certamente -e giustamente- discontinua!

[14] dire di partizionare un intervallo in n sotto-intervalli è esattamente coincidente a dire che lo si partiziona in $n+1$ nodi; se per esempio volessimo $n+1 = 5$ nodi, allora l'intervallo sarebbe partizionato in 4 sotto-intervalli, in 4 segmenti

L'espressione matematica della spline lineare è molto semplice, e si può ricavare mediante la solita dimostrazione che ho imparato in terza liceo su come trovare la retta passante per due punti: proprio di questo si parla! In particolare, si ha che

$$\frac{S_1(x) - f(x_i)}{f(x_{i+1}) - f(x_i)} = \frac{x - x_i}{x_{i+1} - x_i}, \qquad x \in [x_i, x_{i+1}]$$

che, dopo alcune semplicissime manipolazioni algebriche, permette di ottenere

$$S_1(x) = f(x_i) + \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}(x - x_i), \qquad x \in [x_i, x_{i+1}]. \tag{2.14}$$

In ogni caso, non è strettamente necessario, su MATLAB® , implementare questa funzione. In effetti, esiste il comando `interp1` che permette esattamente di ottenere il risultato garantito da (2.14). Per concludere, si segnala che l'Appendice A.4 fornisce un codice MATLAB® in grado di generare, per diversi nodi di interpolazione, la spline lineare interpolante.

Studiando il precedente argomento (interpolazione polinomiale), abbiamo notato che possono esserci problemi di convergenza. Per le spline lineari, ci viene incontro un teorema, che ora enunceremo. Data una partizione $a = x_1 < x_2 < ... < x_{n+1} = b$, sia S_1 la spline interpolante una funzione $f \in C^{(2)}([a, b])$ nei nodi della partizione x_i, $i = 1, ..., n + 1$. Dato

$$h = \max_{1 \le i \le n}(x_{i+1} - x_i),$$

allora

$$\|f - S_1\|_\infty = \mathcal{O}(h^2), \quad h \to 0.$$

Detto in altre parole, data una qualsiasi funzione a derivata seconda continua sull'intero intervallo, l'approssimazione mediante una spline lineare converge **sempre** uniformemente al crescere del numero di nodi (o, detto in altra maniera, al **decrescere della massima distanza tra nodi** h), e con un tasso di convergenza pari al quadrato di h.

2.3.3 Spline cubiche

«Siamo contenti?»

Mah. Abbastanza, diciamo. Sicuramente poteva andarci peggio. Dobbiamo riconoscere che le spline lineari ci danno delle garanzie: aumentando il numero di nodi, possiamo approssimare la nostra funzione arbitrariamente bene, ossia bene quanto ci piace. Alla fine, questo è un po' lo stesso principio che usiamo al momento di disegnare una funzione con `plot`: se vediamo la funzione troppo *squadrata* rispetto a quello che ci aspetteremmo, possiamo sempre provare ad aumentare il numero di nodi, e il disegno diventerà più liscio.

Se però la domanda fosse stata

«Siamo entusiasti?»

La risposta sarebbe stata no. Abbiamo la garanzia che il tasso di convergenza sia $\mathcal{O}(h^2)$ indipendentemente dalla regolarità della funzione f (o quasi: deve avere almeno la derivata seconda continua su $[a, b]$!). Tuttavia, a noi tutto sommato farebbe piacere che, se la f fosse bella liscia, con insomma molte derivate continue, allora non sarebbe necessario usare troppi nodi. Avere un h basso significa usare molti nodi, ma ricordiamoci sempre che calcolare queste spline ha comunque un certo costo computazionale, ossia, richiede un certo numero di operazioni.

Tuttavia, con $d = 1$ non è possibile fare di meglio. Ma non ce l'ha mica detto il dottore, che le spline lineari siano l'unica possibilità a disposizione. In effetti, quando si parla di spline, uno pensa sempre alle **spline cubiche**, ovvero $d = 3$. Questo significa che, per ogni j-esimo sotto-intervallo, la nostra spline avrà una forma del tipo

$$S_3(x) = a_j x^3 + b_j x^2 + c_j x + d_j, \qquad x \in [x_j, x_{j+1}]. \tag{2.15}$$

Abbiamo dunque implicitamente definito il j-esimo sotto-intervallo come quello compreso tra x_j e x_{j+1}. Dal momento che ipotizziamo sempre di lavorare con $n + 1$ nodi, avremo ancora n intervalli. In altre

parole, al fine di determinare i 4 coefficienti a_j, b_j, c_j, d_j per ognuno degli n sotto-intervalli, avremo bisogno di $4n$ condizioni che le nostre spline devono rispettare.

Non è che dobbiamo inventarci noi le condizioni che la spline deve rispettare: ne abbiamo un elenco al termine dell'introduzione in Sezione 2.3.1! Volendole scrivere per esteso per quanto riguarda il caso in esame, $d = 3$,

1. Come già espresso in (2.15), $S_3(x)$ è un polinomio cubico su ciascun sotto-intervallo.
2. La spline deve essere interpolante e quindi, per ogni nodo della partizione, deve soddisfare le condizioni di interpolazione

$$S_3(x_i) = y_i, \qquad i = 1, ..., n+1.$$

Questa espressione può essere scritta in maniera più esplicita come

$$a_j x_j^3 + b_j x_j^2 + c_j x_j + d_j = y_j, \qquad j = 1, ..., n,$$

che, tradotto dal matematichese, significa: *fai in modo che l'estremo sinistro di ognuno dei polinomi sia agganciato a* (x_j, y_j). C'è un'ulteriore condizione di interpolazione, che è

$$a_n x_{n+1}^3 + b_n x_{n+1}^2 + c_n x_{n+1} + d_n = y_{n+1}, \tag{2.16}$$

che va imposta separatamente, dal momento che noi abbiamo scelto di far sì che sia *l'estremo sinistro* di ciascun polinomio a soddisfare la condizione di interpolazione, ma, non essendoci un polinomio *a destra dell'ultimo*, l'ultima condizione va imposta a parte[15]

3. Da un lato, dalla precedente scelta, sembrerebbe che ciascun polinomio funzioni solo *a sinistra*, ma *a destra* sia libero. In effetti, però, lo scopo delle condizioni di raccordo è proprio quello di *dare armonia* e far sì che i polinomi *vadano d'accordo* sull'intero intervallo $[a, b]$. La prima di queste condizioni *di armonia* o, più formalmente, di raccordo, è quella di continuità: la spline deve essere continua su ciascun punto di raccordo; quindi,

$$a_j x_{j+1}^3 + b_j x_{j+1}^2 + c_j x_{j+1} + d_j = a_{j+1} x_{j+1}^3 + b_{j+1} x_{j+1}^2 + c_{j+1} x_{j+1} + d_{j+1}, \qquad j = 1, ..., n-1. \tag{2.17}$$

Qui scegliamo che *l'estremo destro* di ciascun j-esimo polinomio sia uguale *all'estremo sinistro* del successivo. Come conseguenza dell'identificazione del j-esimo intervallo come definito tra i nodi $[x_j, x_{j+1}]$, questa condizione va soddisfatta solamente per j che va da 1 a $n-1$, dal momento che in questo modo la condizione di continuità sarà imposta solo per i nodi compresi da x_2 a x_n; non avrebbe senso imporre una condizione di continuità su x_1 o x_{n+1}, dal momento che a sinistra di x_1 e a destra di x_{n+1} non abbiamo altri polinomi.

4. Dal momento che $d = 3$, anche la derivata prima deve essere continua; di conseguenza, l'espressione della derivata sarà, per ciascun sotto-intervallo,

$$\frac{\mathrm{d}S_3(x)}{\mathrm{d}x} = 3a_j x^2 + 2b_j x + c_j, \qquad j = 1, ..., n.$$

Esattamente come per la condizione di continuità sarà dunque necessario imporre la condizione di continuità della derivata. Questo porta a un'espressione analoga alla precedente, che sarà

$$3a_j x_{j+1}^2 + 2b_j x_{j+1} + c_j = 3a_{j+1} x_{j+1}^2 + 2b_{j+1} x_{j+1} + c_{j+1}, \qquad j = 1, ..., n-1. \tag{2.18}$$

5. Di nuovo, essendo $d = 3$, anche la derivata seconda dovrà essere continua sull'intero intervallo $[a, b]$. Quindi, si può scrivere la derivata seconda per ciascun sotto-intervallo come

$$\frac{\mathrm{d}^2 S_3(x)}{\mathrm{d}x^2} = 6a_j x + 2b_j, \qquad j = 1, ..., n.$$

E quindi, proprio come per le condizioni di continuità e derivabilità, dovremo imporre

$$6a_j x_{j+1} + 2b_j = 6a_{j+1} x_{j+1} + 2b_{j+1}, \qquad j = 1, ..., n-1. \tag{2.19}$$

[15] tutta questa è stata una scelta arbitraria, ma che non cambia in alcun modo il risultato finale: avremmo potuto richiedere che fossero gli estremi destri di ciascun polinomio a soddisfare la condizione di interpolazione, e il primo sarebbe stato trattato separatamente, o chissà quale altra combinazione: non cambia nulla.

«Tutto qui?»

Purtroppo, temo di no. Ricordiamoci che abbiamo bisogno di $4n$ condizioni; le abbiamo trovate? Contiamo un po':

$$\underbrace{n+1}_{\text{interp.}} + \underbrace{n-1}_{\text{cont.}} + \underbrace{n-1}_{\text{der.I}} + \underbrace{n-1}_{\text{der.II}} = 4n-2.$$

Al fine di garantire l'unicità della spline (e quindi un metodo che possa permetterci di trovarne una, senza ambiguità), avremmo voluto $4n$ condizioni e invece, seguendo le indicazioni forniteci dalle definizioni di spline, ne abbiamo trovate solo $4n-2$: abbiamo bisogno di altre due condizioni, che dobbiamo decidere noi.

Nella letteratura, vengono proposte, per poter arrivare a $4n$, tre categorie di condizioni aggiuntive, ora presentate.

1. Spline cubiche naturali: si ottengono imponendo, come due condizioni aggiuntive, la derivata seconda uguale a zero ai nodi estremi:

$$\begin{aligned} 6a_1x_1 + b_1 &= 0 \\ 6a_nx_{n+1} + b_n &= 0. \end{aligned}$$

2. Spline cubiche *not-a-knot*: si ottengono imponendo, come condizioni aggiuntive, la continuità della derivata *terza* nei nodi x_2 e x_n. Questo significa che

$$\frac{\mathrm{d}^3 S_3(x)}{\mathrm{d}x^3} = 6a_j, \qquad j = 1, \ldots, n.$$

   In altre parole stiamo chiedendo di imporre le condizioni

$$\begin{aligned} a_1 &= a_2 \\ a_{n-1} &= a_n. \end{aligned}$$

   Questa condizione, che è usata di default in MATLAB® , è molto particolare. Infatti, la condizione si chiama *not-a-knot*, ovvero *non un nodo*, perché di fatto, sui primi due e sugli ultimi due intervalli, la spline cubica interpolante diventerà un polinomio interpolante di grado 3. Infatti, se abbiamo due polinomi di grado 3, imporre la condizione **anche** sulla derivata terza significa imporre **l'eguaglianza di tutti i coefficienti**. Questo si può dimostrare semplicemente rivedendo quali sono le condizioni di continuità imposte tra i polinomi appartenenti al primo e al secondo intervallo:

$$\begin{aligned} a_1x_2^3 + b_1x_2^2 + c_1x_2 + d_1 &= a_2x_2^3 + b_2x_2^2 + c_2x_2 + d_2 \\ 3a_1x_2^2 + 2b_1x_2 + c_1 &= 3a_2x_2^2 + 2b_2x_2 + c_2 \\ 6a_1x_2 + 2b_1 &= 6a_2x_2 + 2b_2 \\ 6a_1 &= 6a_2. \end{aligned}$$

   Sostituendo *a gambero* l'ultima condizione nella penultima, si ottiene $b_1 = b_2$; sostituendo le ultime due condizioni nella seconda, si ottiene $c_1 = c_2$; sostituendo queste tre nella prima, si ottiene infine $d_1 = d_2$. Per l'ultimo intervallo vale una dimostrazione analoga.

3. Spline cubiche vincolate: ammesso di conoscere per qualche motivo il valore delle derivate prime $f'(x)$ agli estremi x_1 e x_{n+1}, si richiede che la derivata prima della spline agli estremi sia uguale a quella della funzione, ossia

$$\begin{aligned} \left.\frac{\mathrm{d}S_3(x)}{\mathrm{d}x}\right|_{x=x_1} &= f'(x_1) \\ \left.\frac{\mathrm{d}S_3(x)}{\mathrm{d}x}\right|_{x=x_{n+1}} &= f'(x_{n+1}). \end{aligned}$$

Queste condizioni si possono scrivere in maniera più esplicita come

$$3a_1x_1^2 + 2b_1x_1 + c_1 = f'(x_1)$$
$$3a_nx_{n+1}^2 + 2b_nx_{n+1} + c_n = f'(x_{n+1}).$$

Di queste tre categorie di condizioni proposte, se ne sceglie solo una, ogni volta che si calcola una spline. Non è che queste condizioni *escano dal cilindro*, cioè non sono pensate *a caso*, ma sono state studiate al fine di avere garanzie sulla convergenza delle spline cubiche. In effetti, se utilizziamo una delle spline cubiche soddisfacenti a una di queste tre condizioni, abbiamo la garanzia che la funzione converga rapidamente quanto la spline lineare.

L'obiettivo dietro a tutta la teoria che abbiamo presentato, però, era cercare di superare il tasso della convergenza delle spline lineari. In effetti, se usiamo le condizioni (2) o (3), abbiamo la garanzia, ammesso di avere una funzione abbastanza regolare ($f \in C^{(4)}([a,b])$), di avere una convergenza rapida in norma infinito non solo della funzione, ma anche delle derivate! Vale infatti il seguente teorema. Sia $h_i = x_{i+1} - x_i$, il passo legato all'intervallo i-esimo. Sia poi

$$h = \max_{1\le i\le n} h_i.$$

Se vigono le condizioni aggiuntive (2) o (3), ovvero spline *not-a-knot* o *vincolata*, se $f \in C^{(4)}([a,b])$, e se

$$\frac{h}{h_i} \le \gamma < \infty,$$

allora

$$\left\| f^{(p)} - S_3^{(p)} \right\|_\infty = \mathcal{O}(h^{4-p}), \quad h \to 0, p = 0,1,2,3.$$

Spendiamo un po' di tempo per commentare questo teorema. Prima di tutto, commentiamo l'ipotesi sul massimo passo h:

$$\frac{h}{h_i} \le \gamma < \infty.$$

Scrivere questo significa scrivere

$$h \le \gamma h_i,$$

dove si richiede che γ sia minore di infinito e, conseguentemente, h_i diverso da zero. Questa scrittura è semplicemente un modo di dire che **la griglia deve essere più o meno uniforme**: non dobbiamo avere dei passi h_i troppo piccoli o tantomeno il massimo passo h non deve essere troppo grosso rispetto agli altri. Se vige questo, abbiamo che la p-esima derivata della spline converge uniformemente alla corrispondente derivata della funzione. Questo è più di quanto avevamo visto finora: fino a questo momento ci eravamo posti domande sulla convergenza della funzione, non della derivata; quando abbiamo un fenomeno come quello appena osservato, possiamo parlare di **simultanea approssimazione**. Da un lato, questa ci permette una convergenza molto rapida della funzione ($p = 0$), che sarà $\mathcal{O}(n^4)$ in virtù di quanto abbiamo appena detto; *simultaneamente*, anche le derivate convergeranno.

Tutto è bene quel che finisce bene, ma a questo punto dobbiamo spezzare una lancia a favore della interpolazione polinomiale. Infatti, qui abbiamo questo $\mathcal{O}(n^{4-p})$, con quel 4 che in qualche modo limita il tasso di convergenza. Infatti, se avessimo per esempio una funzione con 12 derivate continue, quindi $C^{(12)}([a,b])$, il tasso di convergenza sarebbe sempre lo stesso di quello di una funzione $C^{(4)}([a,b])$. Questo è un limite legato alle spline cubiche, e viene detto **fenomeno della saturazione**. In questo senso, i polinomi interpolanti, se usati adeguatamente[16], come avevamo visto, permettono di ottenere una convergenza uniforme tipo

$$\|f - p_n\|_\infty = \mathcal{O}\left(\frac{\log n}{n^k}\right),$$

[16] con gli opportuni nodi di interpolazione

dove k è l'ordine della derivata massima continua su $[a, b]$: qui, non abbiamo nessun fenomeno di saturazione!

Prima di concludere questo argomento, si vuole proporre una doverosa precisazione: **data una funzione continua, le spline cubiche convergono sempre uniformemente a essa**. Fino a questo momento, questa frase non era stata scritta chiaramente; infatti, sono state proposte varie stime dei tassi di convergenza ipotizzando di interpolare funzioni sufficientemente regolari, ma sia chiaro che, se la funzione è continua, si ha la **garanzia**, anche se magari non con un tasso *ottimo*, che la spline cubica, al crescere dei nodi, convergerà uniformemente a essa.

2.3.4 Cenni al metodo dei trapezi

Un argomento che non verrà approfondito in queste note, ma che vale la pena di menzionare in qualità di *applicazione delle spline lineari* è il **metodo dei trapezi** per il calcolo dell'integrale di una funzione. L'idea di questo metodo è molto semplice: approssimare una funzione mediante una spline lineare, e quindi calcolare l'area di ciascun trapezio di base maggiore B, base minore b e altezza h mediante la solita formuletta analitica

$$A = \frac{B+b}{2}h.$$

In MATLAB®, il comando `trapz` permette di realizzare questa operazione senza doverla fare *a mano*. Al fine di mostrarlo, proponiamoci di calcolare un integrale molto semplice, del quale conosciamo la soluzione analitica[17]:

$$\int_2^3 x^2 \, \mathrm{d}x = \frac{x^3}{3}\bigg|_2^3 = \frac{27}{3} - \frac{8}{3} = \frac{19}{3}.$$

Si può per esempio implementare il seguente script, e vedere come va:

```
clear
close all
clc

Nnodi=10; % numero di nodi di interpolazione/integrazione
a=2; b=3; % estremi di interpolazione/integrazione
x = linspace(a,b,Nnodi); % usiamo (non e` obbligatorio) nodi equispaziati
f = x.^2;
Integrale = trapz(x,f);
Integrale_esatto = 19/3;
errrel = abs(Integrale-Integrale_esatto)/Integrale_esatto
```

L'errore relativo `errrel` dovrebbe essere dell'ordine di 10^{-4} usando 10 nodi; questa non è una tecnica molto raffinata, ma di sicuro può essere un punto di partenza nel caso capitassero problemi di integrazione semplici da risolvere.

[17] per quanto la tentazione di provare da subito la Gaussiana sia forte, eh?

Appendice A

Appendice: Approssimazione di dati e funzioni

A.1 Esempi di implementazione MATLAB del polinomio di interpolazione

Viene ora riportato lo script MATLAB® che genera il grafico di Fig. 2.1.

```
clear       % svuoto la memoria
close all   % chiudo tutte le figure precedentemente aperte
clc         % pulisco la command window

n=13; % definisco il grado del polinomio interpolante
x=linspace(-5,5,n+1); % vettore di nodi di interpolazione, equidistanti
f=1./(1+x.^2); % valutazione della funzione nei nodi di interpolazione

xref=linspace(-5,5,1001); % griglia, fitta, di riferimento per i plot
fref=1./(1+xref.^2); % valutazione della funzione nella griglia fitta

c=polyfit(x,f,n); % calcolo dei coefficienti del polinomio interpolante
pol=polyval(c,xref); % valutazione del polinomio interpolante su griglia fitta

figure % apro una figura
set(gcf,'Position',[368 212 955 638]) % imposto dimensioni e posizione della figura
grid on % ordino di mettere una griglia sullo schermo
hold on % ordino alla funzione di mantenere le tracce precedenti
box on % per gradevolezza, faccio chiudere il plot da un "bordo"
plot(xref,fref,'b') % ordino di disegnare la funzione sulla griglia fitta, in colore blu
plot(x,f,'ko') % ordino di disegnare i nodi di interpolazione e marcarli con dei cerchietti (o) neri (k)
plot(x,f,'g') % ordino di disegnare la funzione, in colore verde, sulla griglia di interpolazione; questo ↩
    coincide
            % con disegnare l'interpolazione lineare dei nodi (e` simile a interp1)
plot(xref,pol,'r') % disegno il polinomio interpolante valutato sulla griglia fitta, in colore rosso
axis([-5,5,-1.5,1]) % imposto gli assi che verranno visualizzati: x va da -5 a 5, y va da -1.5 a 1
xlabel('x') % imposto l'etichetta dell'asse delle ascisse
ylabel('f(x)')  % imposto l'etichetta dell'asse delle ordinate
```

Inoltre, viene riportato un secondo esempio, che contiene esempi di utilizzo di sintassi più avanzate rispetto a quelle viste fino a ora.

```
clear
close all
clc

f = @(x) x.*sin(x); % questa sintassi permette di definire un function
                    % handle; si tratta di un modo di lavorare di MATLAB
                    % che permette di trattare "f" non come una variabile,
                    % ma come una funzione che si puo` valutare con una
                    % sintassi tipo f(x), f(z) o quello che e`, dove x e z
                    % sono dei vettori di punti

z = linspace(0,2*pi); % definisco la griglia "fine" per vedere gli interpolanti

% segue un esempio un po' particolare di utilizzo del ciclo for. Il
% concetto del ciclo for e` lo stesso del C, pero` qui la variabile sulla
```

```
% quale si cicla va definita come un vettore, e MATLAB eseguira` tanti cicli
% quanti sono gli elementi nel vettore n. Nell'esempio in questione, n
% contiene due elementi, quindi avremo due cicli. Per ogni ciclo, n assume
% il valore corrispettivo.
% In questo caso, diversi valori di n corrispondono, come si puo` vedere
% all'interno del codice, a diversi ordini del polinomio interpolante
for n = [5 11]
    % il comando linspace(a,b,N) permette di definire un vettore di
    % elementi equispaziati a partire da a, fino a b, composto da N
    % elementi. Si tratta di un comando un po' diverso dalla sintassi a:b,
    % dal momento che anche in quel caso erano equispaziati, ma in quel
    % caso non si fissava il numero di elementi, quanto piuttosto il passo.
    x = linspace(0,2*pi,n+1); % definisco un vettore di n+1 nodi di interpolazione
                              % poiche' voglio un polinomio di grado n

    y = f(x); % sfrutto il function handle per poter scrivere velocemente che
              % i dati da interpolare, y, sono uguali a f valutata nei nodi
              % x

    c = polyfit(x,y,n); % calcolo coefficienti polinomio interpolante grado n...
    p = polyval(c,z);   % ... e valuto il polinomio sulla griglia fine z

    plot(x,y,'ro',z,f(z),'r',z,p,'b') % i nodi di interpolazione sono disegnati con
                                      % cerchietti rossi, la funzione di
                                      % partenza nella griglia fine e`
                                      % disegnata con una curva rossa, e il
                                      % polinomio interpolante con una
                                      % curva blu

    pause % metto una pausa, per permettere di visualizzare per ogni polinomio la figura
end
```

A.2 Pillole di storia: Joseph-Louis Lagrange

Nonostante il nome non sembrerebbe suggerirlo, Joseph-Louis Lagrange[1] era italiano e, in particolare, torinese; questo è il motivo per il quale gli è stato dedicato il Dipartimento di Scienze Matematiche (DISMA) del Politecnico di Torino, nonché una delle principali vie del centro di Torino. Giuseppe Lodovico Lagrangia nasceva a Torino il 25 gennaio 1736. Volendo usare un termine che va di moda, è stato un *cervello in fuga*, in quanto, dopo Torino, andò prima a Berlino verso il 1760, poi a Parigi intorno al 1787, dove si stabilì più o meno definitivamente. Esistono diverse versioni della sua firma, tutte attribuibili a lui, perché in quegli anni c'era la rivoluzione francese e utilizzare nomi che potevano evocare origini nobili poteva portarti dal boia. Il punto di arrivo fu il cognome *Lagrange* con cui ora lo conosciamo tutti.

Per capire un po' meglio il personaggio, vorrei citare le parole[2] di un suo illustre studente, Jean Baptiste Joseph Fourier, padre della teoria del calore.

> Lagrange, il principale accademico d'Europa, sembra avere tra i 50 e i 60 anni di età, anche se in realtà è più giovane (*n.d.r.: all'epoca in realtà aveva proprio 59 anni...*); ha un forte accento italiano e pronuncia una *s* come se fosse una *z*; si veste con molta discrezione, in nero o marrone; parla informalmente e con qualche difficoltà, con l'esitante semplicità di un bambino. Tutti sanno che è un uomo straordinario, ma bisogna averlo visto per riconoscerlo come tale. Parla solo in discussione, e parte di ciò che dice eccita derisioni. L'altro giorno ha detto:
> "Ci sono molte cose importanti da dire su questo argomento, ma io non le dirò."
> Gli studenti sono incapaci di apprezzare il suo genio, ma i professori lo riconoscono chiaramente.

L'importanza di Lagrange nel progresso della Matematica è conclamata grazie ai suoi contributi in diversi ambiti, che vale la pena di riassumere.

- La fondazione del Calcolo delle Variazioni. I metodi variazionali permettono di attribuire rigore matematico ad alcuni principi fondamentali della Fisica, quali il principio di minima azione, e permettono di fornire una delle formulazioni del metodo degli elementi finiti (FEM) per la soluzione di equazioni alle derivate parziali.

- La formulazione della Meccanica Analitica, anche detta Meccanica Razionale.

[1] altri dettagli su https://it.wikipedia.org/wiki/Joseph-Louis_Lagrange
[2] Fonte: http://www-groups.dcs.st-and.ac.uk/history/Extras/Fourier_teachers.html

- L'ideazione dei moltiplicatori di Lagrange, che permettono di risolvere problemi di ottimizzazione vincolata, ossia, cercare il minimo di una qualche funzione a molte variabili, date alcune condizioni che devono essere soddisfatte.
- Ovviamente, la rappresentazione di Lagrange del polinomio interpolante, anch'essa impiegata nella forma basilare del metodo degli elementi finiti.

A.3 Convergenza uniforme del polinomio di interpolazione: esempi MATLAB

A.3.1 Esempio 1: nodi equispaziati

Viene ora riportato uno script MATLAB® che permette di generare i polinomi di interpolazione di vari gradi utilizzando nodi equispaziati. Questo permette chiaramente di dimostrare che, agendo in maniera ingenua, non si hanno garanzie sulla convergenza uniforme del polinomio interpolante. L'esempio studia sull'intervallo $[-5, 5]$ la funzione di Runge, che ha ha infinite derivate continue.

```
clear
close all
clc

nvet = [1 2 3 4 5 6 7 8 9 10 11 12]; % specificare i gradi dei polinomi di interpolazione che si vogliono ↩
    provare
a = -5; % estremo inferiore intervallo
b = +5; % estremo superiore intervallo
f = @(x)1./(1+x.^2); % definisco la funzione di partenza
z = linspace(a,b,10001); % definisco la griglia fine su cui valutare i polinomi di interpolazione

figure(1)
set(gcf,'Position',[381 175 1208 753])

for indn = 1:length(nvet)
    n = nvet(indn);
    x = linspace(a,b,n+1); % in questo esercizio definisco i nodi di interpolazione tra loro equispaziati
    c = polyfit(x,f(x),n); % valuto i coefficienti del polinomio interpolante
    p = polyval(c,z); % valuto il polinomio interpolante
    %
    figure(1), clf
    hold on
    grid on
    box on
    plot(z,f(z),'b',x,f(x),'ko',z,p,'r') % blu, curva originale; cerchietti neri, nodi di interpolazione; ↩
        rosso, polinomio
    xlabel('x');
    ylabel('f(x) e p_n(x)')
    title(['Funzione di partenza (blu) e polinomio p_{',num2str(n),'}(x)'])
    errrel = norm(f(z)-p,inf)/norm(f(z),inf);
    disp(['n = ',num2str(n),', errore relativo norma infinito ',num2str(errrel)])
    pause

end
```

A.3.2 Esempio 2: nodi di Chebyshev

Viene ora riportato uno script MATLAB® che permette di generare i polinomi di interpolazione di vari gradi, dove però vengono usati nodi di Chebyshev anziché nodi equispaziati. L'esempio studia di nuovo la funzione di Runge sull'intervallo $[-5, 5]$.

```
clear
close all
clc

nvet = [10 12 14 18 22]; % specificare i gradi dei polinomi di interpolazione che si vogliono provare
a = -5; % estremo inferiore intervallo
b = +5; % estremo superiore intervallo
f = @(x)1./(1+x.^2); % definisco la funzione di partenza
z = linspace(a,b,10001); % definisco la griglia fine su cui valutare i polinomi di interpolazione

figure(1)
set(gcf,'Position',[381 175 1208 753])
```

```
for indn = 1:length(nvet)
    n = nvet(indn);
    i = 1:(n+1); % indici dei nodi di Chebyshev
    zi = -cos(((2*i-1)*pi)/(2*(n+1))); % nodi Cheb. su intervallo [-1,1]
    x = (b-a)/2*zi+(b+a)/2; % mappare nodi Cheb. su intervallo [a,b]
    c = polyfit(x,f(x),n); % valuto i coefficienti del polinomio interpolante
    p = polyval(c,z); % valuto il polinomio interpolante
    %
    figure(1), clf
    hold on
    grid on
    box on
    plot(z,f(z),'b',x,f(x),'ko',z,p,'r') % blu, curva originale; cerchietti neri, nodi di interpolazione; ↩
        rosso, polinomio
    xlabel('x');
    ylabel('f(x) e p_n(x)')
    title(['Funzione di partenza (blu) e polinomio p_{',num2str(n),'}(x)'])
    errrel = norm(f(z)-p,inf)/norm(f(z),inf);
    disp(['n = ',num2str(n),', errore relativo norma infinito ',num2str(errrel)])
    pause

end
```

A.4 Esempio di uso di spline lineari

Viene ora riportato uno script MATLAB® che permette di generare, una volta fissato il numero di intervalli, la spline lineare interpolante una funzione nota. Lanciando questo codice per diversi intervalli, è possibile apprezzare la convergenza uniforme di queste funzioni.

```
clear
close all
clc

nvet = [1 2 3 4 5 6 7 8 9 10 11 12]; % specificare i numeri di nodi (-1) della partizione
a = -5; % estremo inferiore intervallo
b = +5; % estremo superiore intervallo
f = @(x)1./(1+x.^2); % definisco la funzione di partenza
z = linspace(a,b,10001); % definisco la griglia fine su cui valutare la spline

figure(1)
set(gcf,'Position',[381 175 1208 753])

for indn = 1:length(nvet)
    n = nvet(indn);
    x = linspace(a,b,n+1); % in questo esercizio definisco i nodi di interpolazione tra loro equispaziati
    S = interp1(x,f(x),z); % valuto la spline lineare interpolante
    %
    figure(1), clf
    hold on
    grid on
    box on
    plot(z,f(z),'b',x,f(x),'ko',z,S,'r') % blu, curva originale; cerchietti neri, nodi di interpolazione; ↩
        rosso, spline
    xlabel('x');
    ylabel('f(x) e p_n(x)')
    title(['Funzione di partenza (blu) e polinomio p_{',num2str(n),'}(x)'])
    errrel = norm(f(z)-S,inf)/norm(f(z),inf);
    disp(['n = ',num2str(n),', errore relativo norma infinito ',num2str(errrel)])
    pause

end
```

A.5 Esempio di uso di spline cubiche

Viene ora riportato uno script MATLAB® che permette di generare, una volta fissato il numero di intervalli, la spline cubica interpolante una funzione nota. Lanciando questo codice per diversi intervalli, è possibile apprezzare la convergenza uniforme di queste funzioni.

```
clear
close all
clc

nvet = [1 2 3 4 5 6 7 8 9 10 11 12]; % specificare i numeri di nodi (-1) della partizione
a = -5; % estremo inferiore intervallo
```

```
b = +5; % estremo superiore intervallo
f = @(x)1./(1+x.^2); % definisco la funzione di partenza
z = linspace(a,b,10001); % definisco la griglia fine su cui valutare la spline

figure(1)
set(gcf,'Position',[381 175 1208 753])

for indn = 1:length(nvet)
    n = nvet(indn);
    x = linspace(a,b,n+1); % in questo esercizio definisco i nodi di interpolazione tra loro equispaziati
    S = spline(x,f(x),z); % valuto la spline cubica interpolante
    %
    figure(1), clf
    hold on
    grid on
    box on
    plot(z,f(z),'b',x,f(x),'ko',z,S,'r') % blu, curva originale; cerchietti neri, nodi di interpolazione; ↩
        rosso, spline
    xlabel('x');
    ylabel('f(x) e p_n(x)')
    title(['Funzione di partenza (blu) e polinomio p_{',num2str(n),'}(x)'])
    errrel = norm(f(z)-S,inf)/norm(f(z),inf);
    disp(['n = ',num2str(n),', errore relativo norma infinito ',num2str(errrel)])
    pause

end
```

A.6 Approfondimenti sulle spline

Questa appendice ha l'obiettivo di presentare alcuni approfondimenti e riflessioni riguardanti le spline, con particolare attenzione verso le spline cubiche. Per questo motivo, è consigliata la lettura esclusivamente in caso di interesse personale verso l'argomento, in quanto si prenderanno in considerazione dettagli che possono essere complicati e la cui comprensione non è fondamentale ai fini del semplice utilizzo delle spline cubiche.

A.6.1 Quando abbiamo pochi nodi: spline, polinomi interpolanti, o *entrambi*?

Al fine di comprendere queste tecniche numeriche, è opportuno applicarle a casi *semplici*, e quindi, per poter capire bene cosa sta succedendo, a griglie di interpolazione con pochi nodi. Il minimo sindacale ovviamente sarebbe usare 2 nodi: non posso ottenere alcuna informazione sull'andamento di una funzione interpolando solo in un punto!

Parlando di spline cubiche interpolanti *not-a-knot*, voglio proporre una frase un po' forte.

> «Quando abbiamo un numero di nodi minore o uguale a 4, non ha senso distinguere polinomi interpolanti e spline cubiche.»

Perché mai? Beh, qualche sezione fa, si era detto che

> «Assegnati $n+1$ dati (x_i, y_i), $i = 1, \dots, n+1$, esiste uno e un solo polinomio $p_n(x)$ di grado minore o uguale a n interpolante i dati assegnati, ovvero soddisfacente le condizioni di interpolazione.»

Ovvero, il famoso teorema di esistenza e unicità. Questo ci dice che il polinomio interpolante questi punti ha grado 3. Da un altro punto di vista, se immaginiamo di usare 4 nodi di interpolazione e di cercare una spline cubica *not-a-knot*, dovremo dividere il nostro intervallo in tre sotto-intervalli, e imporre:

- quattro condizioni di interpolazione;
- due condizioni di continuità della spline cubica;
- due condizioni di continuità della derivata prima della spline cubica;
- due condizioni di continuità della derivata seconda della spline cubica;
- **due condizioni di continuità della derivata terza della spline cubica** (*not-a-knot*).

Come avevamo scritto nel testo, le condizioni *not-a-knot* fanno sì che il primo e il secondo intervallo, nonché il penultimo e l'ultimo, abbiano polinomi uguali tra loro, quindi con coefficienti assolutamente identici. Dire che i coefficienti sono identici, è come dire che abbiamo un unico polinomio definito su tutti e tre i sotto-intervalli.

Questa dimostrazione *astratta* per quanto corretta può essere rafforzata da un esperimento numerico. A tal fine, si propone un codice che interpola una certa funzione su quattro nodi disposti in modo casuale mediante polinomio interpolante e mediante spline cubica; studiando l'errore relativo, si può vedere che è comparabile alla precisione di macchina e che, quindi, i due procedimenti, con 4 nodi, **sono esattamente uguali**. Il seguente codice MATLAB® permette di dimostrare questo fatto mediante un esperimento numerico.

```
clear
close all
clc

x=[0.1  0.5  1  3]; % definizione di alcuni nodi disposti a caso
y=cos(x); % usiamo come funzione da interpolare una funzione a caso

z = linspace(min(x),max(x),1001); % definisco la griglia fine

% Interpolo mediante spline
S = spline(x,y,z);

% Interpolo mediante polinomio interpolante
c = polyfit(x,y,length(x)-1);
p = polyval(c,z);

% Calcolo l'errore in norma infinito tra le due rappresentazioni
errrel = norm(S-p,inf)/norm(p,inf)
```

Possiamo invece dire qualcosa riguardo al caso con 2 e 3 nodi? Sì: sono identici a quello a 4 nodi, e per lo stesso motivo. In effetti, quando abbiamo 2 nodi, dire che usiamo un polinomio interpolante di grado 1 o dire che usiamo una spline lineare è la stessa cosa: le due curve sono coincidenti. Allo stesso modo, il caso con 3 nodi, produrrà, con i comandi `spline` e `polyfit/polyval`, esattamente gli stessi risultati, e per la stessa spiegazione che abbiamo fornito. Quindi, perché le spline siano un qualcosa di effettivamente diverso dal polinomio interpolante, è necessario aver a che fare almeno con 5 nodi di interpolazione.

A.6.2 Ma come fa MATLAB a calcolare le spline?!

Volendo essere **davvero** convinti riguardo a ciò che è stato scritto nel testo, in questa appendice studiamo ora uno schema numerico che permette di calcolare i coefficienti delle spline per un caso molto semplice: una funzione interpolata in 5 nodi (che, come abbiamo imparato nella precedente sottosezione, è il primo caso che abbia davvero senso studiare mediante le spline). In questa sezione, quindi, prima proporremo la formulazione teorica, e poi un'implementazione dell'algoritmo atto a ricavare i coefficienti delle spline su 5 nodi. Da un punto di vista concettuale, questo è esattamente ciò che fa MATLAB® (che però è ottimizzato in vari modi dal punto di vista della programmazione).

Il primo passo è ricordare l'espressione delle condizioni di interpolazione (2.16); nell'esercizio che stiamo svolgendo, abbiamo 5 nodi, e quindi le condizioni si possono scrivere esplicitamente come

$$
\begin{aligned}
a_1 x_1^3 + b_1 x_1^2 + c_1 x_1 + d_1 &= y_1 \\
a_2 x_2^3 + b_2 x_2^2 + c_2 x_2 + d_2 &= y_2 \\
a_3 x_3^3 + b_3 x_3^2 + c_3 x_3 + d_3 &= y_3 \\
a_4 x_4^3 + b_4 x_4^2 + c_4 x_4 + d_4 &= y_4 \\
a_4 x_5^3 + b_4 x_5^2 + c_4 x_5 + d_4 &= y_5.
\end{aligned}
$$

Come spiegheremo in dettaglio più avanti[3], queste espressioni si possono scrivere in termini di un prodotto matriciale riga per colonna come

[3] nell'ambito dello studio dei sistemi lineari

$$\begin{bmatrix} x_1^3 & x_1^2 & x_1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & x_2^3 & x_2^2 & x_2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & x_3^3 & x_3^2 & x_3 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & x_4^3 & x_4^2 & x_4 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & x_5^3 & x_5^2 & x_5 & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ b_1 \\ c_1 \\ d_1 \\ a_2 \\ b_2 \\ c_2 \\ d_2 \\ a_3 \\ b_3 \\ c_3 \\ d_3 \\ a_4 \\ b_4 \\ c_4 \\ d_4 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix}.$$

Infatti, si può vedere *facilmente* che, calcolando i prodotti riga per colonna, si ottengono le cinque equazioni scritte prima.

Proseguiamo a questo punto con le condizioni di continuità delle spline; partendo da (2.17) possiamo scrivere esplicitamente, portando tutte le espressioni al membro sinistro,

$$\begin{aligned} a_1x_2^3 + b_1x_2^2 + c_1x_2 + d_1 - (a_2x_2^3 + b_2x_2^2 + c_2x_2 + d_2) &= 0 \\ a_2x_3^3 + b_2x_3^2 + c_2x_3 + d_2 - (a_3x_3^3 + b_3x_3^2 + c_3x_3 + d_3) &= 0 \\ a_3x_4^3 + b_3x_4^2 + c_3x_4 + d_3 - (a_4x_4^3 + b_4x_4^2 + c_4x_4 + d_4) &= 0, \end{aligned}$$

che si scrivono, in forma matriciale,

$$\begin{bmatrix} x_2^3 & x_2^2 & x_2 & 1 & -x_2^3 & -x_2^2 & -x_2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & x_3^3 & x_3^2 & x_3 & 1 & -x_3^3 & -x_3^2 & -x_3 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & x_4^3 & x_4^2 & x_4 & 1 & -x_4^3 & -x_4^2 & -x_4 & -1 \end{bmatrix} \begin{bmatrix} a_1 \\ b_1 \\ c_1 \\ d_1 \\ a_2 \\ b_2 \\ c_2 \\ d_2 \\ a_3 \\ b_3 \\ c_3 \\ d_3 \\ a_4 \\ b_4 \\ c_4 \\ d_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}.$$

Seguono quindi le condizioni di continuità della derivata prima, che si scrivono per esteso, a partire da (2.18),

$$\begin{aligned} 3a_1x_2^2 + 2b_1x_2 + c_1 - (3a_2x_2^2 + 2b_2x_2 + c_2) &= 0 \\ 3a_2x_3^2 + 2b_2x_3 + c_2 - (3a_3x_3^2 + 2b_3x_3 + c_3) &= 0 \\ 3a_3x_4^2 + 2b_3x_4 + c_3 - (3a_4x_4^2 + 2b_4x_4 + c_4) &= 0 \end{aligned}$$

e quindi, in forma matriciale, come

$$\begin{bmatrix} 3x_2^2 & 2x_2 & 1 & 0 & -3x_2^2 & -2x_2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3x_3^2 & 2x_3 & 1 & 0 & -3x_3^2 & -2x_3 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3x_4^2 & 2x_4 & 1 & 0 & -3x_4^2 & -2x_4 & -1 & 0 \end{bmatrix} \begin{bmatrix} a_1 \\ b_1 \\ c_1 \\ d_1 \\ a_2 \\ b_2 \\ c_2 \\ d_2 \\ a_3 \\ b_3 \\ c_3 \\ d_3 \\ a_4 \\ b_4 \\ c_4 \\ d_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}.$$

Seguendo un procedimento analogo, è possibile scrivere anche le condizioni sulla derivata seconda (2.19):

$$\begin{aligned} 6a_1x_2 + 2b_1 - (6a_2x_2 + 2b_2) &= 0 \\ 6a_2x_3 + 2b_2 - (6a_3x_3 + 2b_3) &= 0 \\ 6a_3x_4 + 2b_3 - (6a_4x_4 + 2b_4) &= 0, \end{aligned}$$

che si possono scrivere in forma matriciale come segue:

$$\begin{bmatrix} 6x_2 & 2 & 0 & 0 & -6x_2 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 6x_3 & 2 & 0 & 0 & -6x_3 & -2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 6x_4 & 2 & 0 & 0 & -6x_4 & -2 & 0 & 0 \end{bmatrix} \begin{bmatrix} a_1 \\ b_1 \\ c_1 \\ d_1 \\ a_2 \\ b_2 \\ c_2 \\ d_2 \\ a_3 \\ b_3 \\ c_3 \\ d_3 \\ a_4 \\ b_4 \\ c_4 \\ d_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}.$$

Infine, per quanto riguarda le condizioni *not-a-knot* si ha, a partire da

$$\begin{aligned} a_1 - a_2 &= 0 \\ a_3 - a_4 &= 0, \end{aligned}$$

la scrittura matriciale

$$\begin{bmatrix} 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} a_1 \\ b_1 \\ c_1 \\ d_1 \\ a_2 \\ b_2 \\ c_2 \\ d_2 \\ a_3 \\ b_3 \\ c_3 \\ d_3 \\ a_4 \\ b_4 \\ c_4 \\ d_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

Impilando tutte queste matrici, è possibile trovare un'equazione matriciale nella forma

$$\begin{bmatrix} x_1^3 & x_1^2 & x_1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & x_2^3 & x_2^2 & x_2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & x_3^3 & x_3^2 & x_3 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & x_4^3 & x_4^2 & x_4 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & x_5^3 & x_5^2 & x_5 & 1 \\ x_2^3 & x_2^2 & x_2 & 1 & -x_2^3 & -x_2^2 & -x_2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & x_3^3 & x_3^2 & x_3 & 1 & -x_3^3 & -x_3^2 & -x_3 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & x_4^3 & x_4^2 & x_4 & 1 & -x_4^3 & -x_4^2 & -x_4 & -1 \\ 3x_2^2 & 2x_2 & 1 & 0 & -3x_2^2 & -2x_2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3x_3^2 & 2x_3 & 1 & 0 & -3x_3^2 & -2x_3 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3x_4^2 & 2x_4 & 1 & 0 & -3x_4^2 & -2x_4 & -1 & 0 \\ 6x_2 & 2 & 0 & 0 & -6x_2 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 6x_3 & 2 & 0 & 0 & -6x_3 & -2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 6x_4 & 2 & 0 & 0 & -6x_4 & -2 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} a_1 \\ b_1 \\ c_1 \\ d_1 \\ a_2 \\ b_2 \\ c_2 \\ d_2 \\ a_3 \\ b_3 \\ c_3 \\ d_3 \\ a_4 \\ b_4 \\ c_4 \\ d_4 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

Segue quindi uno script che implementa tutte le formule appena ricavate al fine di ricavare i coefficienti e valutare le spline.

```
clear
close all
clc

% definiamo alcuni nodi in cui andare a interpolare
x1=0.1;
x2=0.5;
x3=1;
x4=3;
x5=4;

x = [x1 x2 x3 x4 x5];
y=cos(x); % usiamo una funzione a caso

n=length(x)-1;
b = zeros(4*n,1);

% Matrice delle condizioni di interpolazione
AI = [x1^3  x1^2    x1     1     0     0     0     0     0     0     0     0     0     0     0  0;
         0     0     0     0  x2^3  x2^2    x2     1     0     0     0     0     0     0     0  0;
         0     0     0     0     0     0     0     0  x3^3  x3^2    x3     1     0     0     0  0;
         0     0     0     0     0     0     0     0     0     0     0     0  x4^3  x4^2    x4  1;
         0     0     0     0     0     0     0     0     0     0     0     0  x5^3  x5^2    x5  1];
b(1:n+1)=y;

% Matrice delle condizioni di continuita`
AC = [x2^3  x2^2    x2     1 -x2^3 -x2^2   -x2    -1     0     0     0     0     0     0     0  0;
```

```
        0     0     0     0  x3^3  x3^2    x3     1 -x3^3 -x3^2   -x3    -1     0     0     0  0;
        0     0     0     0     0     0     0     0  x4^3  x4^2    x4     1 -x4^3 -x4^2   -x4 -1];

% Matrice delle condizioni di continuita` derivata prima
AD1 = [3*x2^2 2*x2^1  1   0 -3*x2^2 -2*x2^1   -1 0       0       0  0 0       0       0   0   0;
           0     0  0   0  3*x3^2  2*x3^1    1 0 -3*x3^2 -2*x3^1 -1 0       0       0   0   0;
           0     0  0   0       0       0    0 0  3*x4^2  2*x4^1  1 0 -3*x4^2 -2*x4^1  -1   0];

% Matrice delle condizioni di continuita` derivata seconda
AD2 = [6*x2     2  0     0    -6*x2   -2     0  0     0      0     0  0     0     0  0  0;
          0     0  0     0     6*x3    2     0  0 -6*x3    -2     0  0     0     0  0  0;
          0     0  0     0     0       0     0  0  6*x4     2     0  0 -6*x4    -2  0  0];

% Matrice delle condizioni not-a-knot
ANAK = [1     0     0     0    -1     0     0     0  0     0     0     0     0     0     0     0;
        0     0     0     0     0     0     0     0  0     0     0     1     0     0     0    -1];

% Assemblo la matrice finale impilando le varie matrici
A = [AI;
     AC;
     AD1;
     AD2;
     ANAK
    ];

% Calcolo i coefficienti risolvendo il sistema lineare
Coeff = A\b;

% I coefficienti sono ordinati come segue:
% [a1,b1,c1,d1,a2,b2,c2,d2,a3,b3,c3,d3,a4,b4,c4,d4]
Coeff1=Coeff(1:4);
Coeff2=Coeff(4+[1:4]);
Coeff3=Coeff(8+[1:4]);
Coeff4=Coeff(12+[1:4]);

% Per semplificare, definisco 4 vettori di z, uno per ciascun intervallo.
z1 = 0.1:0.01:0.5;
z2 = 0.5:0.01:1;
z3 = 1:0.01:3;
z4 = 3:0.01:4;

% Calcolo le spline usando la formula e i relativi coefficienti
S3_1 = Coeff1(1)*z1.^3 + Coeff1(2)*z1.^2 + Coeff1(3)*z1 + Coeff1(4);
S3_2 = Coeff2(1)*z2.^3 + Coeff2(2)*z2.^2 + Coeff2(3)*z2 + Coeff2(4);
S3_3 = Coeff3(1)*z3.^3 + Coeff3(2)*z3.^2 + Coeff3(3)*z3 + Coeff3(4);
S3_4 = Coeff4(1)*z4.^3 + Coeff4(2)*z4.^2 + Coeff4(3)*z4 + Coeff4(4);

z = [z1 z2 z3 z4];
S3 = [S3_1 S3_2 S3_3 S3_4];

% Calcolo le spline con le funzioni MATLAB, per verificare la correttezza
S=spline(x,y,z);

figure(3289)
grid on
hold on
box on
plot(z,S3,z,S,'--')

errrel=norm(S-S3)/norm(S) % l'errore relativo e` nullo: i procedimenti sono uguali!
```

Questo esempio permette di capire davvero quanto sia complicato calcolare una spline anche per il caso più semplice possibile: per trovare i coefficienti relativi all'interpolazione di una funzione su 5 nodi è necessario risolvere un sistema 16×16! Inoltre, questo script di esempio permette di dimostrare e chiarire ulteriormente alcuni concetti. Se consideriamo i vettori dei vari coefficienti `Coeff1`, `Coeff2`, `Coeff3`, `Coeff4`, ciascuno relativo a uno dei quattro sottointervalli, possiamo farli stampare sulla command window (ciascuno sarà un vettore colonna contenente, su ciascuna riga, a_j, b_j, c_j, d_j, dove j è l'indice dell'intervallo), ottenendo

```
>> [Coeff1 Coeff2 Coeff3 Coeff4]
ans =

    0.1574    0.1574    0.1140    0.1140
   -0.6753   -0.6753   -0.5450   -0.5450
    0.0628    0.0628   -0.0675   -0.0675
    0.9953    0.9953    1.0387    1.0387
>>
```

Questo è molto interessante: i coefficienti della prima e della seconda spline sono uguali tra loro, così come quelli della terza e della quarta. Questa è la conseguenza della condizione *not-a-knot*, come già dimostrato formalmente nel testo!

Capitolo 3

Sistemi lineari

Introduzione

L'argomento *sistemi lineari* è talmente importante da non richiedere tante presentazioni: ogni volta che si devono risolvere equazioni che per qualche motivo sono tra loro *accoppiate*, dobbiamo risolvere un sistema. Una situazione tipica in cui si ha a che fare con sistemi lineari si ha ogni volta che si cerca di *dare in pasto* un'equazione differenziale a un computer: la si approssima in modo da trasformarla in un sistema di equazioni. Per entrare nel merito,

- un **sistema** è un insieme di equazioni tra loro accoppiate e che devono essere risolte **simultaneamente**;
- un **sistema lineare** è un sistema in cui ciascuna delle equazioni è **lineare**, ovvero non contiene potenze, prodotti tra le variabili o funzioni non lineari di vario tipo (logaritmi, esponenziali...). Per esempio, il sistema

$$\begin{cases} x + y + 2 = 0 \\ xy + 3 = 0 \end{cases}$$

è non lineare, dal momento che contiene il prodotto xy. D'altra parte,

$$\begin{cases} x + y - 5 = 0 \\ 4x + 7y + 9 = 0 \end{cases}$$

è un tipico esempio di sistema lineare.

Il concetto di sistema lineare è indissolubilmente legato a quello di matrice: in effetti, al fine di risolvere i sistemi lineari, la procedura è prima di tutto determinare la *matrice associata* al sistema e, poi, manipolarla. Lo scopo di questa parte del testo sarà presentare dei metodi *furbi* per risolvere sistemi lineari in diverse condizioni. Finché non verrà esplicitamente detto diversamente, ci concentreremo su sistemi a n equazioni e n incognite aventi una e una sola soluzione, dunque associati a matrici quadrate con determinante diverso da zero. Verso la fine dell'argomento studieremo la soluzione di sistemi sovradimensionati, ossia con più equazioni che incognite.

Ci concentreremo sui **sistemi lineari densi**, ossia le cui matrici non contengono molti zeri, e con **metodi diretti**, che cioè risolvono il sistema *in un colpo solo* o comunque *con un numero di passi noto a priori*. Per dare un'idea, risolvere un sistema lineare denso mediante un metodo diretto è assolutamente ragionevole con matrici di sistema 1000×1000, cioè con 1000 righe e 1000 colonne, utilizzando un PC di fascia media. Il mio *record* in questo senso è stato, con MATLAB® e una workstation (da 32 GiB di RAM), un sistema 30000×30000, ma già si era al limite. Quando si raggiungono dimensioni così grandi, è necessario utilizzare tecniche di cui non parleremo, cercando se possibile di sfruttare l'eventuale sparsità[1] della matrice o di ricorrere a metodi *iterativi*.

[1] con un sistema *sparso* (di quelli con tanti zeri che non studieremo), un sistema 150000×150000 si risolve abbastanza tranquillamente con un normale computer in circa un secondo

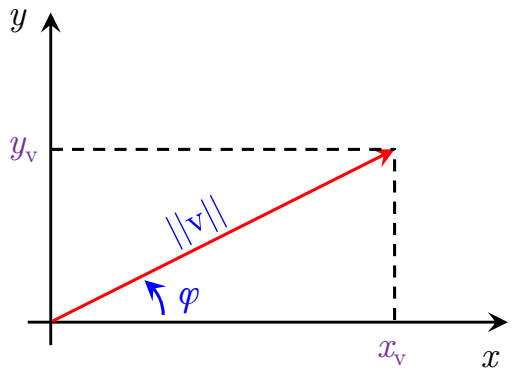

Figura 3.1: Rappresentazione geometrica di un vettore (freccia rossa) in un sistema di assi cartesiano avente componenti x_v, y_v, o equivalentemente rappresentato in coordinate *polari* (parenti di *modulo*, *direzione*, *verso*).

3.1 Concetti preliminari sui sistemi lineari

Per via del fortissimo legame presente tra sistemi lineari e vettori/matrici, in questa sezione riprenderemo e/o introdurremo alcuni concetti su di essi.

3.1.1 Il concetto di norma

La parola *vettore* viene pronunciata molto frequentemente in vari contesti apparentemente disgiunti: algebra lineare, geometria, fisica, informatica. Buone notizie: tutto sommato, in tutti questi contesti si parla sempre della stessa cosa, ma studiata da punti di vista un po' diversi.

- Dal punto di vista dell'algebra lineare e da quello informatico, un vettore viene definito come un insieme di numeri disposti in riga o in colonna.
- Dal punto di vista geometrico e da quello fisico, un vettore viene visto come un segmento al quale si attribuisce un modulo, una direzione e un verso; in particolare, in geometria viene sempre disegnato come una freccia che parte dall'origine degli assi, mentre in fisica si parla di *vettori applicati*[2] con un significato abbastanza simile.

In alcune situazioni, è possibile identificare con grande facilità la connessione tra tutti questi punti di vista. Pensiamo per esempio a $\mathbb{R}^2$, quindi a un vettore $\underline{v}$ con due componenti reali. Ciò che si può fare è attribuire alla prima e alla seconda componente del vettore, x_v e y_v, il significato geometrico di coordinate di un punto sul piano cartesiano. A questo punto, è possibile ottenere la rappresentazione geometrica del vettore congiungendo mediante una freccia l'origine del nostro sistema di riferimento con il punto disegnato sul piano. La rappresentazione per componenti cartesiane $[x_v \;\; y_v]$ non è l'unica ammissibile: in Fisica è comune sentir parlare, come anticipato poco fa, di *modulo*, *direzione* e *verso*. Queste tre quantità sono facilmente ottenibili, mediante un po' di trigonometria, a partire dalle componenti:

- si può identificare il *modulo* del vettore mediante la sua *lunghezza*, calcolabile mediante il teorema di Pitagora:

$$L_v = \sqrt{x_v^2 + y_v^2}; \tag{3.1}$$

- si può identificare la *direzione* per esempio usando l'angolo φ compreso tra il vettore $\underline{v}$ e l'asse x:

$$\varphi = \arctan\left(\frac{y_v}{x_v}\right);$$

- il *verso* è semplicemente il *segno* di questo vettore: positivo se punta verso l'alto, negativo se punta verso il basso.

[2] per esempio, studiando le forze applicate su un punto le frecce partono da esso.

È possibile riferirsi alla Fig. 3.1 al fine di visualizzare meglio i concetti appena introdotti.

Per poter dire di *conoscere completamente* il vettore $\underline{v} \in \mathbb{R}^2$, sono necessari 2 numeri, sia che si parli delle componenti cartesiane, sia che si parli di quelle polari. Tuttavia, può accadere che, per un motivo o per un altro, si decida di sacrificare parte di queste informazioni ed *estrarre*, a partire da $\underline{v}$, un qualche numero che in qualche modo lo rappresenti. Questo potrebbe essere per esempio il caso della *lunghezza*: potremmo dire che del vettore ci interessa solo sapere il modulo, ma non come sia diretto. Al fine di produrre dei numeri che rappresentino le più interessanti proprietà del vettore, è possibile ricorrere a diverse definizioni di **norma**[3]. Si consideri da ora un vettore colonna $\underline{x} \in \mathbb{R}^n$ definito come

$$\underline{x} = \begin{bmatrix} x_1 & x_2 & x_3 & \dots & x_n \end{bmatrix}^{\mathrm{T}}.$$

In questo testo considereremo in particolare tre diverse definizioni di norma per $\underline{x}$.

- La norma 1, definita come la somma dei moduli delle componenti del vettore

$$\|\underline{x}\|_1 = \sum_{i=1}^{n} |x_i|.$$

- La norma 2, definita come la radice della somma dei quadrati delle componenti

$$\|\underline{x}\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2}.$$

  Questa è una delle norme più significative, dal momento che rappresenta un'estensione al caso n-dimensionale del teorema di Pitagora; infatti, per $n = 2$, questa degenera al classico teorema di Pitagora (3.1). In altre parole, questa si può vedere come una generalizzazione del concetto di lunghezza del vettore. Esiste un modo diverso per scrivere la norma 2, basato sul prodotto righe per colonne. Infatti, questa si può anche scrivere come

$$\|\underline{x}\|_2 = \sqrt{\underline{x}^{\mathrm{T}}\underline{x}}.$$

  Dato per esempio un vettore in $\mathbb{R}^3$, si avrebbe:

$$\underline{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}, \quad \underline{x}^{\mathrm{T}} = \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix},$$

  e, quindi,

$$\underline{x}^{\mathrm{T}}\underline{x} = \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = x_1^2 + x_2^2 + x_3^2 = \sum_{i=1}^{3} x_i^2.$$

  Calcolando quindi la radice di questo, si arriva alla definizione appena scritta.

- La già citata norma infinito, definita come il massimo dei moduli delle componenti del vettore:

$$\|\underline{x}\|_\infty = \max_{1 \le i \le n} |x_i|.$$

Il concetto di norma è anche applicabile a una matrice $\underline{\underline{A}} \in \mathbb{R}^{n,n}$. Sia a_{ij} l'elemento della matrice $\underline{\underline{A}}$ sulla riga i e sulla colonna j. Allora,

[3] questo è già stato fatto in precedenza: quando abbiamo avuto bisogno, per questioni di maneggiabilità dei dati, di ricavare un singolo numero a partire da una funzione o da un vettore, abbiamo dovuto fatto ricorso al concetto di norma infinito

- La norma 1 della matrice $\underline{\underline{A}}$ si può scrivere come

$$\left\|\underline{\underline{A}}\right\|_1 = \max_{1\leq j\leq n} \sum_{i=1}^{n} |a_{ij}| .$$

  Spiegato a parole, questo significa, per ogni j-esima colonna, calcolare la somma dei suoi moduli, e trovare il massimo tra queste. Questa si può anche spiegare come: considerare la matrice $\underline{\underline{A}}$ come un impilamento di n vettori colonna; calcolare la norma 1 di ciascuno dei vettori colonna, e ottenere quindi un singolo vettore riga contenente tutte le norme 1 delle varie colonne; di queste, trovare la massima.

- La norma infinito della matrice $\underline{\underline{A}}$ si può scrivere come

$$\left\|\underline{\underline{A}}\right\|_\infty = \max_{1\leq i\leq n} \sum_{j=1}^{n} |a_{ij}| .$$

  Si tratta un po' di un'operazione opposta rispetto alla norma 1 di matrice: considero la matrice come l'impilamento di tanti vettori riga, e di ciascuno di questi calcolo la norma 1, ottenendo quindi un vettore colonna che contiene su ciascuna componente la norma 1 della riga di $\underline{\underline{A}}$ corrispondente; di queste, calcolo poi la massima.

- Esiste un'estensione del concetto di norma 2 anche per quanto riguarda le matrici, e questa viene detta **norma spettrale**; sarà tuttavia discussa più avanti nel testo, nel capitolo relativo ad autovalori e autovettori.

Abbiamo introdotto quindi i concetti di norma di vettori e di matrici; tuttavia, in generale, non è detto che queste siano *compatibili*. Si dice che una norma di vettore e una norma di matrici sono compatibili se vale la relazione

$$\left\|\underline{\underline{A}}\,\underline{x}\right\| \leq \left\|\underline{\underline{A}}\right\| \|\underline{x}\| . \tag{3.2}$$

Si noti che in (3.2) non sono stati specificati i *numeri al pedice*: la relazione deve essere valida per norme di vettori e matrici di tipo corrispondente, per esempio

$$\left\|\underline{\underline{A}}\,\underline{x}\right\|_1 \leq \left\|\underline{\underline{A}}\right\|_1 \|\underline{x}\|_1 .$$

Inoltre, si osservi che, al membro sinistro, si ha $\underline{\underline{A}}\,\underline{x}$, che è un vettore: il prodotto di una matrice per un vettore colonna è ancora un vettore colonna! Dunque, l'unica norma di matrice che entra in gioco nell'espressione (3.2) è quella al membro destro. Inoltre, essendo la norma semplicemente un numero sia per i vettori, sia per le matrici, questa relazione è una semplice diseguaglianza tra scalari. Tutte le norme che considereremo in questa trattazione sono compatibili. Questa è un'ottima notizia, perché ci permette quindi di sfruttare la diseguaglianza (3.2) per poter effettuare maggiorazioni o minorazioni di alcuni termini al momento di studiare equazioni matriciali. In questo senso, è utile tenere presente anche la diseguaglianza

$$\left\|\underline{\underline{A}}\,\underline{\underline{B}}\right\| \leq \left\|\underline{\underline{A}}\right\| \left\|\underline{\underline{B}}\right\| . \tag{3.3}$$

Infine, si sappia che per tutte le norme considerate, data $\underline{\underline{I}}$ la matrice identità, $\left\|\underline{\underline{I}}\right\| = 1$.

3.1.2 Carrellata delle principali categorie di matrici

Per quanto questa trattazione sia focalizzata su matrici dense, è importante sapere che una matrice potrebbe avere particolari configurazioni, per esempio avere un certo numero di elementi a 0; di conseguenza, i metodi che utilizzeremo per risolvere i sistemi lineari associati a esse, saranno diversi. Proponiamo ora una *carrellata* delle più famose matrici; a questo fine, si consideri una matrice $\underline{\underline{A}} \in \mathbb{R}^{5,5}$, che nella più generale forma si può scrivere come

$$\underline{\underline{A}} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \end{bmatrix} .$$

Da qui si intende che, quando un elemento a_{ij} è scritto esplicitamente (al posto di 0), questo possa essere non nullo.

- Una matrice si dice **diagonale** se solo gli elementi della diagonale principale sono diversi da zero:

$$\underline{\underline{A}} = \begin{bmatrix} a_{11} & 0 & 0 & 0 & 0 \\ 0 & a_{22} & 0 & 0 & 0 \\ 0 & 0 & a_{33} & 0 & 0 \\ 0 & 0 & 0 & a_{44} & 0 \\ 0 & 0 & 0 & 0 & a_{55} \end{bmatrix},$$

  ossia, se $i \neq j$, si ha che $a_{ij} = 0$.

- Una matrice si dice **triangolare superiore** se tutti gli elementi sotto la diagonale principale (esclusa) sono uguali a zero:

$$\underline{\underline{A}} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ 0 & a_{22} & a_{23} & a_{24} & a_{25} \\ 0 & 0 & a_{33} & a_{34} & a_{35} \\ 0 & 0 & 0 & a_{44} & a_{45} \\ 0 & 0 & 0 & 0 & a_{55} \end{bmatrix},$$

  ovvero se, per $i > j$, si ha che $a_{ij} = 0$.

- Una matrice si dice **triangolare inferiore** se tutti gli elementi sopra la diagonale principale (esclusa) sono uguali a zero:

$$\underline{\underline{A}} = \begin{bmatrix} a_{11} & 0 & 0 & 0 & 0 \\ a_{21} & a_{22} & 0 & 0 & 0 \\ a_{31} & a_{32} & a_{33} & 0 & 0 \\ a_{41} & a_{42} & a_{43} & a_{44} & 0 \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \end{bmatrix},$$

  ovvero se, per $i < j$, si ha che $a_{ij} = 0$.

- Una matrice si dice **tridiagonale** se solo gli elementi della diagonale principale, della prima codiagonale inferiore e della prima codiagonale superiore sono diversi da zero:

$$\underline{\underline{A}} = \begin{bmatrix} a_{11} & a_{12} & 0 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & 0 & 0 \\ 0 & a_{32} & a_{33} & a_{34} & 0 \\ 0 & 0 & a_{43} & a_{44} & a_{45} \\ 0 & 0 & 0 & a_{54} & a_{55} \end{bmatrix},$$

  ovvero se, per $|i - j| > 1$, $a_{ij} = 0$ (in questo caso, per esempio, se considero a_{53}, ho che $|5 - 3| = 2$, che è maggiore di 1, e quindi l'elemento va a 0.

- Una matrice si dice **Hessenberg superiore** se, oltre a essere come la triangolare superiore, ha anche gli elementi della prima codiagonale inferiore diversi da zero.

$$\underline{\underline{A}} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ 0 & a_{32} & a_{33} & a_{34} & a_{35} \\ 0 & 0 & a_{43} & a_{44} & a_{45} \\ 0 & 0 & 0 & a_{54} & a_{55} \end{bmatrix},$$

  ovvero se, per $i > j + 1$, $a_{ij} = 0$ (per esempio, $i = 5$, $j = 1$, soddisfa $5 > 1 + 1$, e infatti l'elemento va a 0).

3.1.3 Operazioni e altre definizioni sulle matrici

Terminata la carrellata delle principali forme di matrici che può capitare di trovare, è opportuno introdurre alcune altre matrici che si distinguono più per alcune proprietà particolari che per il numero o le posizioni di elementi nulli. Ora come ora può non essere chiaro a cosa servano tutte queste definizioni ma, man mano che andremo avanti, le ritroveremo e le riutilizzeremo!

Trasposta di una matrice e matrici simmetriche

Prima di tutto, a questo fine, è opportuno ricordare la definizione di **trasposta** $\underline{\underline{A}}^{\mathrm{T}}$ di una matrice $\underline{\underline{A}}$, che si ottiene scambiando gli elementi simmetrici rispetto alla diagonale. Per esempio, a partire dalla nostra $\underline{\underline{A}} \in \mathbb{R}^{5,5}$, si ha:

$$\underline{\underline{A}} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \end{bmatrix} \Longrightarrow \underline{\underline{A}}^{\mathrm{T}} = \begin{bmatrix} a_{11} & a_{21} & a_{31} & a_{41} & a_{51} \\ a_{12} & a_{22} & a_{32} & a_{42} & a_{52} \\ a_{13} & a_{23} & a_{33} & a_{43} & a_{53} \\ a_{14} & a_{24} & a_{34} & a_{44} & a_{54} \\ a_{15} & a_{25} & a_{35} & a_{45} & a_{55} \end{bmatrix}.$$

Detto ancora in un altro modo, bisogna scrivere, al posto di ciascun a_{ij}, il a_{ji}: scambiare gli indici di righe e di colonne.

Esistono matrici che non cambiano una volta che viene applicata l'operazione di trasposizione; queste matrici vengono dette **simmetriche**. Questa cosa è possibile quando gli elementi simmetrici rispetto alla diagonale principale sono uguali tra loro. Questo è il caso della seguente matrice $\underline{\underline{A}}$:

$$\underline{\underline{A}} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{12} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{13} & a_{23} & a_{33} & a_{34} & a_{35} \\ a_{14} & a_{24} & a_{34} & a_{44} & a_{45} \\ a_{15} & a_{25} & a_{35} & a_{45} & a_{55} \end{bmatrix},$$

in cui si è cercato di evidenziare con gli stessi colori gli elementi simmetrici[4]. In sintesi, il fatto che applicare la trasposizione non cambia niente, e quindi la **simmetria** della matrice, si può esprimere come

$$\underline{\underline{A}}^{\mathrm{T}} - \underline{\underline{A}} = 0.$$

Inversa di una matrice, matrici ortogonali e matrici di permutazione

Data una matrice $\underline{\underline{A}}$, a patto che il suo determinante $\det\{\underline{\underline{A}}\}$ sia diverso da zero, è possibile definire la matrice inversa $\underline{\underline{A}}^{-1}$ tale per cui

$$\underline{\underline{A}}^{-1}\underline{\underline{A}} = \underline{\underline{A}}\,\underline{\underline{A}}^{-1} = \underline{\underline{I}}.$$

Un'altra definizione che ci tornerà molto utile più avanti è quella di **matrice ortogonale**. Una matrice $\underline{\underline{A}}$ viene detta **ortogonale** se

$$\underline{\underline{A}}^{\mathrm{T}}\underline{\underline{A}} = \underline{\underline{A}}\,\underline{\underline{A}}^{\mathrm{T}} = \underline{\underline{I}}.$$

Si noti che, nel caso di matrici ortogonali, $\underline{\underline{A}}^{\mathrm{T}} = \underline{\underline{A}}^{-1}$.

Come noto dai rudimenti di Geometria, la parola *ortogonale* è un po' un sinonimo di *perpendicolare*. Tuttavia questo concetto, più che alle matrici, sembra essere applicabile ai vettori: per esempio, si dice che due vettori sono ortogonali se da un punto di vista geometrico formano un angolo retto (pensiamo a $\mathbb{R}^2$...), o, generalizzando, se il loro prodotto scalare è nullo. In questo secondo senso, la matrice è ortogonale se ciascuna delle proprie colonne, trasposta e trasformata dunque in una riga, ha prodotto scalare uguale a zero con tutte le altre, tranne che per sé stessa.

Per chiarire un po' meglio questo concetto, è opportuno focalizzarci su una particolare classe di matrici ortogonali: le **matrici di permutazione**. Considerando per esempio $\mathbb{R}^{3,3}$, la matrice

[4] purtroppo si avevano a disposizione pochi colori e alcuni sono stati ripetuti

$$\underline{\underline{P}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

è una matrice di permutazione. Il motivo per cui matrici di questo tipo vengono chiamate in questo modo è legato a come esse agiscono su un vettore. Immaginiamoci di considerare $\underline{x} = [x_1 \quad x_2 \quad x_3]^{\mathrm{T}}$; allora, calcolando il prodotto righe per colonne, otteniamo

$$\underline{\underline{P}}\,\underline{x} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_3 \\ x_2 \end{bmatrix},$$

ovvero, uno scambio tra la seconda e la terza componente di $\underline{x}$! Le matrici di permutazione sono infatti ottenute a partire dalla matrice identità $\underline{\underline{I}}$, scambiando alcune sue righe; in questo modo si ottiene una matrice che, moltiplicata da sinistra per un vettore colonna, produce un vettore con le medesime componenti, ma scambiate di posizione: **permutate**!

Però: cosa c'entrano queste matrici con le matrici ortogonali? Beh, è semplice dimostrare, su questo esempio, che le matrici di permutazione sono matrici ortogonali! Infatti, calcolando, si dimostra che

$$\underline{\underline{P}}\,\underline{\underline{P}}^{\mathrm{T}} = \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}}_{\underline{\underline{P}}} \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}}_{\underline{\underline{P}}^{\mathrm{T}}} = \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\underline{\underline{I}}}!$$

Questa proprietà è, ovviamente, generale per matrici di permutazione $\underline{\underline{P}} \in \mathbb{R}^{n,n}$!

Matrici a diagonale dominante (per righe/colonne)

Data una matrice $\underline{\underline{A}} \in \mathbb{R}^{n,n}$, essa viene definita **a diagonale dominante per righe** se

$$|a_{ii}| > \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}|, \quad i = 1, \ldots, n. \tag{3.4}$$

Cosa significa questa cosa? Per capirla meglio, proviamo a guardare un esempio pratico:

$$\underline{\underline{A}} = \begin{bmatrix} 4 & 1 & -1 \\ 2 & -7 & 1 \\ 3 & -2 & 9 \end{bmatrix}.$$

Prima di tutto, invece di questa matrice, conviene guardare $\left|\underline{\underline{A}}\right|$, ovvero la matrice dei moduli:

$$\left|\underline{\underline{A}}\right| = \begin{bmatrix} 4 & 1 & 1 \\ 2 & 7 & 1 \\ 3 & 2 & 9 \end{bmatrix}.$$

Adesso, seguendo la definizione (3.4), verifichiamo che ciascun elemento della diagonale sia più grande della somma di tutti gli elementi sulle righe corrispondenti. In particolare:

- per la prima riga,

$$4 > 1 + 1$$

- per la seconda riga,

$$7 > 2 + 1$$

- per la terza riga,

$$9 > 3 + 2.$$

Dal momento che tutte e tre queste condizioni sono soddisfatte, la matrice $\underline{\underline{A}}$ in questione è a diagonale dominante per righe.

Allo stesso modo, è possibile proporre una definizione di matrici a diagonale dominante per colonne. Come si può immaginare, seguendo l'esempio appena svolto, invece di fare la somma dei moduli degli elementi fuori-diagonale delle righe, lo faremo con quelli delle colonne. Tuttavia, non è garantito che una matrice a diagonale dominante per righe lo sia anche per colonne; ad esempio, nella matrice $\underline{\underline{A}}$ appena considerata, si ha che, per la prima colonna,

$$4 \not> 2 + 3,$$

quindi $\underline{\underline{A}}$ è a diagonale dominante per righe, ma non per colonne.

Matrici simmetriche definite positive

Consideriamo, almeno preliminarmente (per poi riprenderla in seguito), un'ultima definizione: quella di **matrice simmetrica definita positiva**. Data una matrice $\underline{\underline{A}}$ simmetrica, essa si dice **definita positiva** se, **per ogni** $\underline{x}$ vettore colonna non identicamente nullo,

$$\underline{x}^{\mathrm{T}} \underline{\underline{A}}\, \underline{x} > 0.$$

Discutiamo un momento questa definizione. Prima di tutto, $\underline{\underline{A}}\, \underline{x}$ è un vettore colonna, dal momento che $\underline{x}$ è un vettore colonna e gli si sta applicando da sinistra una matrice. Quindi, moltiplicando il vettore colonna $(\underline{\underline{A}}\, \underline{x})$ al vettore riga $\underline{x}^{\mathrm{T}}$ da destra, il risultato sarà un singolo numero: uno scalare. Per questo motivo, ha senso chiedersi se questo sia maggiore o minore di zero.

Detto questo, questa definizione è abbastanza inutile, scritta in questo modo: è **impossibile da verificare**! Infatti, visto che questa deve essere verificata **per ogni vettore non nullo**, noi dovremmo, per quanto ne sappiamo ora, provare qualsiasi vettore esistente, e vedere se il numero che esce è sempre maggiore di 0. Più avanti studieremo di nuovo questa condizione, cosa implica, e un modo furbo per verificarla.

Per ora, chiudo con un piccolo *spoiler*: dimostreremo che, data una matrice $\underline{\underline{B}} \in \mathbb{R}^{n,n}$, la matrice $\underline{\underline{A}}$ ottenuta come

$$\underline{\underline{A}} = \underline{\underline{B}}^{\mathrm{T}} \underline{\underline{B}},$$

con $\det\{\underline{\underline{B}}\} \neq 0$, è simmetrica definita positiva.

3.1.4 Matrice associata a un sistema lineare

È ora giunto il momento più opportuno per visualizzare lo stretto legame esistente tra sistemi lineari e matrici. In buona sostanza, questo legame è costituito dal concetto di prodotto righe per colonne. Si consideri per esempio un sistema lineare 4×4:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + a_{14}x_4 = b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + a_{24}x_4 = b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + a_{34}x_4 = b_3 \\ a_{41}x_1 + a_{42}x_2 + a_{43}x_3 + a_{44}x_4 = b_4. \end{cases}$$

Di questo, ci si concentri sulla prima equazione:

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + a_{14}x_4 = b_1.$$

Le incognite sono gli x_i, il termine noto è il b_1, e gli altri termini noti sono i coefficienti che moltiplicano le x_i. Sfruttando la definizione di prodotto righe per colonne, è possibile riscrivere questa equazione come

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = b_1.$$

La stessa operazione si può ripetere per le altre righe, e, una volta impilate, ci permette di riscrivere il sistema lineare in forma matriciale come

$$\underbrace{\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}}_{\underline{\underline{A}}} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}}_{\underline{x}} = \underbrace{\begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}}_{\underline{b}},$$

o, in forma compatta,

$$\underline{\underline{A}}\,\underline{x} = \underline{b}.$$

Esempio con una matrice 2×2

Al fine di completare la comprensione di questo argomento, si provi a scrivere in forma matriciale il sistema

$$\begin{cases} 2x + y = 4 \\ 3y + 4x = 5. \end{cases}$$

La prima cosa da fare, è decidere in che ordine mettere le incognite; per esempio, possiamo definire il vettore $\underline{x}$ nella forma

$$\underline{x} = \begin{bmatrix} x \\ y \end{bmatrix}.$$

In questo modo, ha quindi senso, per chiarire, riscrivere il sistema mettendo in ordine le incognite:

$$\begin{cases} 2x + y = 4 \\ 4x + 3y = 5. \end{cases}$$

A questo punto è immediato scrivere il sistema in forma matriciale:

$$\begin{bmatrix} 2 & 1 \\ 4 & 3 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 4 \\ 5 \end{bmatrix}.$$

3.1.5 Condizionamento di un sistema lineare

Precedentemente, era stato introdotto il problema del condizionamento. Questo genere di analisi è applicabile anche ai sistemi lineari nella forma

$$\underline{\underline{A}}\,\underline{x} = \underline{b}.$$

La soluzione del sistema lineare è data dal vettore $\underline{x}$, ottenuto a partire dai dati $\underline{\underline{A}}$ e $\underline{b}$. A questo punto, per studiare il problema del condizionamento, ricordiamo che è necessario perturbare i dati, ottenendo $\overline{\underline{\underline{A}}}$, $\overline{\underline{b}}$, e, facendo tutti i conti in aritmetica esatta su questi dati perturbati, si ottiene una soluzione $\overline{\underline{x}}$. Quindi, ci si pone domande sull'errore relativo ε_x, cercando di stimarlo a partire dagli errori introdotti sui dati in seguito alla loro perturbazione. Si noti che tutti questi errori sono calcolati a partire da vettori e/o matrici; di conseguenza, utilizzare il modulo sarebbe improprio: finiremmo per avere un sacco di numeri, e non capire esattamente a cosa servano. Per questo motivo, la regola generale è che, **quando si ha a che fare con vettori o matrici, il segno di modulo va sostituito con quello di norma**. I conti che stiamo per fare funzionano sia con la norma 1, sia con la norma 2, sia con la norma infinito, quindi non lo specificheremo ulteriormente ai pedici dei segni di norma. Per esempio, per l'errore sulla soluzione, abbiamo

$$\varepsilon_x = \frac{||\underline{x} - \overline{\underline{x}}||}{||\underline{x}||}.$$

Allo stesso modo, per $\underline{\underline{A}}$ e $\underline{b}$, avremo:

$$\varepsilon_A = \frac{\left|\left|\underline{\underline{A}} - \overline{\underline{\underline{A}}}\right|\right|}{\left|\left|\underline{\underline{A}}\right|\right|}, \qquad \varepsilon_b = \frac{||\underline{b} - \overline{\underline{b}}||}{||\underline{b}||}.$$

È possibile dimostrare il seguente teorema: dato per ipotesi

$$\left\|\underline{\underline{A}} - \overline{\underline{\underline{A}}}\right\| < \frac{1}{2\left\|\underline{\underline{A}}^{-1}\right\|},$$

il sistema perturbato

$$\overline{\underline{\underline{A}}}\,\overline{\underline{x}} = \overline{\underline{b}}$$

ammette una e una sola soluzione, e

$$\varepsilon_x \leq 2\kappa(\underline{\underline{A}})(\varepsilon_A + \varepsilon_b),$$

dove

$$\kappa(\underline{\underline{A}}) = \left\|\underline{\underline{A}}\right\| \left\|\underline{\underline{A}}^{-1}\right\|$$

è detto **numero di condizionamento del sistema lineare**. Si ribadisce che questi risultati e i prossimi valgono per le norme 1, 2, ∞.

È possibile spendere qualche parola in più, su questo numero di condizionamento. Infatti, se ricordiamo la proprietà (3.3), che ci permette di scrivere:

$$\left\|\underline{\underline{A}}\right\| \left\|\underline{\underline{A}}^{-1}\right\| \geq \left\|\underline{\underline{A}}\,\underline{\underline{A}}^{-1}\right\| = \left\|\underline{\underline{I}}\right\| = 1.$$

Questo dimostra che il numero di condizionamento di un sistema lineare non può essere più piccolo di 1. In particolare, se esso è prossimo a 1 (il caso migliore), il sistema si dice **ben condizionato**, ovvero, a piccole perturbazioni dei dati corrispondono piccole perturbazioni della soluzione $\overline{x}$. Al contrario, se $\kappa(\underline{\underline{A}}) \gg 1$, piccole perturbazioni dei dati **potrebbero** portare enormi perturbazioni sulla soluzione.

Interpolazione polinomiale: reprise!

È ora giunto il momento di chiudere definitivamente l'argomento *interpolazione polinomiale*, rispondendo a una delle grandi domande ancora aperte dai capitoli precedenti: come funziona il comando `polyfit` di MATLAB®?

Ciò che sappiamo è che MATLAB® lavora sulla forma monomiale, dal momento che `polyfit` permette di ottenere i corrispettivi coefficienti. Ripassiamo dunque questa forma monomiale:

$$p_n(x) = c_1 x^n + x_2 x^{n-1} + \ldots + c_n x + c_{n+1}.$$

Perché il polinomio sia interpolante, esso deve soddisfare le condizioni di interpolazione, che sono

$$p_n(x_i) = y_i, \qquad i = 1, \ldots, n+1.$$

Questo permette di scrivere il seguente sistema lineare:

$$\begin{cases} c_1 x_1^n + c_2 x_1^{n-1} + \ldots + c_n x_1 + c_{n+1} = y_1 \\ c_1 x_2^n + c_2 x_2^{n-1} + \ldots + c_n x_2 + c_{n+1} = y_2 \\ \vdots \\ c_1 x_n^n + c_2 x_n^{n-1} + \ldots + c_n x_n + c_{n+1} = y_n \\ c_1 x_{n+1}^n + c_2 x_{n+1}^{n-1} + \ldots + c_n x_{n+1} + c_{n+1} = y_{n+1}, \end{cases}$$

che, in forma matriciale, si può scrivere come

$$\underbrace{\begin{bmatrix} x_1^n & x_1^{n-1} & \ldots & x_1 & 1 \\ x_2^n & x_2^{n-1} & \ldots & x_2 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_n^n & x_n^{n-1} & \ldots & x_n & 1 \\ x_{n+1}^n & x_{n+1}^{n-1} & \ldots & x_{n+1} & 1 \end{bmatrix}}_{\underline{\underline{V}}} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \\ c_{n+1} \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \\ y_{n+1} \end{bmatrix}.$$

Si noti che il sistema è lineare anche se contiene questi x_i^n, perché le incognite del sistema sono i $\{c_i\}$. In ogni caso, se il sistema non fosse stato lineare, non saremmo mai stati in grado di scriverlo in forma matriciale! Detto questo, a partire da un vettore di punti $\underline{x}$ avente $n+1$ componenti, è possibile definire la cosiddetta matrice di Vandermonde nella forma di $\underline{\underline{V}}$. MATLAB® permette di calcolare a partire da un vettore $\underline{x}$ la matrice di Vandermonde mediante il comando `vander`. Sciaguratamente, è risaputo che i sistemi lineari associati alla matrice di Vandermonde siano mal condizionati, e, per questo, talvolta MATLAB® si lamenta quando si cerca di usare il comando `polyfit`. MATLAB® permette di calcolare, nelle varie norme di nostro interesse (1, 2, infinito) il numero di condizionamento di un sistema mediante il comando `cond`. In particolare, data `A` la variabile che contiene la matrice del sistema lineare, `cond(A)` restituisce il numero di condizionamento in norma 2, mentre `cond(A,1)` e `cond(A,inf)` in norma 1 e infinito.

Un altro esempio di sistema mal condizionato

L'*altro* esempio *famoso* di sistema mal condizionato è quello associato alla matrice di Hilbert. La matrice di Hilbert appare al momento di utilizzare il metodo dei minimi quadrati, che però in questo testo verrà descritto in maniera leggermente diversa. Esiste un comando MATLAB® che permette di generare una matrice di Hilbert con grande facilità: data `n` la dimensione della matrice (numero di righe e numero di colonne: la matrice di Hilbert è quadrata!), `H = hilb(n)` permette di generare la suddetta matrice e memorizzarla nella variabile `H`.

Prima di mostrare un segmento di codice contenente un esempio pratico, è opportuno ricordare qual è il nostro punto di arrivo: questa sezione è finalizzata a introdurre metodi numerici atti a risolvere sistemi lineari. Per questo motivo, un po' come per ogni tipo di problema che si vuole affrontare, è opportuno disporre di sistemi **dei quali conosciamo la soluzione**. In questo modo, potremmo calcolare l'errore relativo rispetto a un riferimento che sappiamo essere esatto, e quindi per esempio quantificare il legame tra condizionamento ed errore sulla soluzione.

Allora, un tipo di esercizio che affronteremo spesso, è: **applicare un metodo per la soluzione di un sistema lineare, del quale sappiamo già che la soluzione è il vettore unitario**. L'idea è: considerando una matrice fornitaci dal testo del problema, ricavare quel vettore di termini noti **tale per cui la soluzione sia uguale al vettore con tutti uni**: $\underline{x} = [1 \;\; 1 \;\; ... \;\; 1]^T$. Considerando per esempio una matrice $\underline{\underline{A}} \in \mathbb{R}^{4,4}$

$$\underline{\underline{A}} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix},$$

qual è il vettore $\underline{b}$ tale per cui il sistema lineare $\underline{\underline{A}}\,\underline{x} = \underline{b}$ abbia soluzione pari al vettore contenente tutti uni? Beh, nulla di più semplice: abbiamo appena scritto che $\underline{\underline{A}}\,\underline{x} = \underline{b}$, quindi, per calcolare questo vettore, dobbiamo semplicemente scrivere

$$\underline{b} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}.$$

Ma quindi, la prima componente del vettore sarà il prodotto riga per colonna della prima riga di $\underline{\underline{A}}$ per il vettore $[1 \;\; 1 \;\; 1 \;\; 1]^T$:

$$b_1 = a_{11} \times 1 + a_{12} \times 1 + a_{13} \times 1 + a_{14} \times 1 = \sum_{j=1}^{4} a_{1j}.$$

Come si può intuire, in generale, si avrà, per una matrice $\mathbb{R}^{n,n}$,

$$b_i = \sum_{j=1}^{n} a_{ij}. \tag{3.5}$$

A questo punto, possiamo proporre un segmento di codice in cui si cerca di mettere in correlazione l'errore relativo sulla soluzione del sistema causato dal condizionamento della matrice. In particolare, questo è il primo segmento di codice che mostrerà come risolvere un sistema lineare mediante MATLAB® . Data la matrice di sistema memorizzata nella variabile `A`, e il termine noto memorizzato nella

variabile `b`, come nella notazione $\underline{\underline{A}}\,\underline{x} = \underline{b}$, la soluzione del sistema matriciale si può ricavare scrivendo `x = A\b`.

Viene ora riportato lo script che risolve il problema appena descritto.

```
clear
close all
clc

ord = [2 3 4 5 6 7 8 9 10 11 12 13 14]; % ordini della matrice da considerare

for n = 1:length(ord) % ciclo da 1 al numero di elementi di ord
    A = hilb(ord(n)); % in questo modo A viene definita come la matrice di
                      % Hilbert n x n
    b = sum(A,2); % il comando sum usato in questo modo produce, a partire
                  % dalla matrice A, un vettore colonna ottenuto sommando
                  % gli elementi di ciascuna riga; questo corrisponde con
                  % la formula (5) del testo.
    x = A\b; % trovo la soluzione x del sistema; in virtu` di quanto
             % abbiamo fatto nella riga precedente, questa dovra` essere un
             % vettore che in qualche modo "somigli" al vettore unitario:
             % se il numero di condizionamento e` basso (e dunque il
             % sistema ben condizionato), questo sara` molto simile al
             % vettore con tutti 1
    u = ones(ord(n),1); % vettore con tutti 1 di dimensione n x 1.
    errrel(n) = norm(u-x,inf)/norm(u,inf); % errore relativo in norma infinito
    Kinf(n) = cond(A,inf);
end

format short e % metto un format che permetta di apprezzare l'errore

[errrel.' Kinf.']

% guardando i risultati, risulta evidente come al crescere del numero di
% condizionamento, l'errore relativo cresca inesorabilmente
```

3.2 Soluzione di sistemi lineari di forma particolare

Incominceremo da adesso a studiare i metodi numerici che permettono di risolvere sistemi lineari. Il nostro scopo sarà studiare diversi metodi, cercando di suggerire quale possa essere il più adatto nelle varie situazioni che studieremo, in particolare a seconda della forma del sistema e quindi della matrice ad esso associata. Per *adatto* intendiamo quale sia quello che ci permetta di ottenere la soluzione minimizzando il numero di operazioni elementari: ottimizzando il costo computazionale. Stimeremo quindi per ciascuna di queste tecniche il numero di operazioni necessarie per portarle a termine, come funzione della dimensione n della matrice (che, a meno di specificare diversamente, sarà sempre quadrata, e quindi $n \times n$).

3.2.1 Soluzione di sistemi diagonali

La prima classe di *sistemi* lineari che studieremo è quella dei sistemi diagonali. Si tratta quindi di sistemi associati a matrici diagonali, per esempio quindi nella forma

$$\begin{cases} a_{11}x_1 = b_1 \\ a_{22}x_2 = b_2 \\ a_{33}x_3 = b_3 \\ \quad\vdots \end{cases} \tag{3.6}$$

Francamente, io non riesco nemmeno a chiamarli *sistemi* con tanta convinzione, non perché io desideri in alcun modo sminuirli, quanto perché l'idea di sistema che ho in mente contiene implicitamente un accoppiamento tra le varie equazioni. Qui, semplicemente, abbiamo a che fare con n equazioni, tra loro indipendenti, e che quindi possono essere risolte ciascuna separatamente dalle altre. In particolare, la componente i-esima del vettore soluzione $\underline{x}$ si può scrivere come

$$x_i = \frac{b_i}{a_{ii}}. \tag{3.7}$$

Questo esempio è molto utile perché, nella sua semplicità, permette di introdurre il concetto di *costo computazionale*, ovvero il numero di operazioni elementari (somme, sottrazioni, prodotti, divisioni) necessarie durante l'esecuzione di un algoritmo.

Nel caso dei sistemi lineari diagonali, dal momento che ogni equazione può essere risolta semplicemente con la divisione (3.7), sono sufficienti n operazioni aritmetiche e quindi il costo computazionale è lineare: questo è il **meglio assoluto** che si possa fare con un *sistema lineare*.

3.2.2 Soluzione numerica di sistemi lineari triangolari (superiori)

La seconda categoria di sistemi lineari che impareremo a risolvere è quella dei **sistemi lineari triangolari**, ossia associati a matrici triangolari. Concentriamoci per esempio sui sistemi triangolari superiori, e, al fine di chiarire, su un esempio di sistema a 3 equazioni e 3 incognite:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1 \\ a_{22}x_2 + a_{23}x_3 = b_2 \\ a_{33}x_3 = b_3. \end{cases}$$

Come si può intuire, la matrice associata al sistema avrà una forma triangolare superiore. Aldilà di questo, si osservi prima di tutto che l'ultima equazione è disaccoppiata da tutte le altre, ovvero si può risolvere utilizzando l'espressione (3.7):

$$x_3 = \frac{b_3}{a_{33}}.$$

A questo punto, si consideri la penultima equazione: essa, è accoppiata solamente all'ultima, che abbiamo appena risolto, ma non alle precedenti! Dunque, a partire da

$$a_{22}x_2 + a_{23}x_3 = b_2,$$

possiamo risolvere rispetto a x_2, dal momento che **tutte le altre quantità sono note**; si avrà, quindi,

$$x_2 = \frac{b_2 - a_{23}x_3}{a_{22}}. \tag{3.8}$$

L'unica equazione che rimane è la prima, la quale è accoppiata con tutte le successive (ma non ne ha prima); tuttavia, abbiamo già risolto tutte le successive equazioni, e quindi, a partire da

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1,$$

è lecito risolvere rispetto a x_1, poiché tutte le altre quantità sono note; quindi, avremo:

$$x_1 = \frac{b_1 - a_{12}x_2 - a_{13}x_3}{a_{11}}. \tag{3.9}$$

Una volta chiarito questo esempio, è possibile proporre una ricetta generale per la soluzione di sistemi triangolari superiori.

1. Il primo passo, che risolvo solo una volta, consiste nel trovare la soluzione dell'ultima equazione, quella disaccoppiata da tutte le altre:

$$x_n = \frac{b_n}{a_{nn}}$$

2. A questo punto, devo procedere a ritroso; dato un indice i che varia da $n-1$ a 1, devo eseguire i seguenti passi:

   (a) calcolo un termine s_i definito come

   $$s_i = \sum_{j=i+1}^{n} a_{ij}x_j$$

   questo rappresenta la generalizzazione dei termini evidenziati in colore rosso nelle equazioni (3.8) e (3.9) per matrici $n \times n$.

   (b) calcolo la componente i-esima del vettore soluzione come:

   $$x_i = \frac{b_i - s_i}{a_{ii}}.$$

Se provassimo a contare il numero di operazioni necessarie per risolvere uno di questi sistemi, scopriremmo che il costo computazionale sarebbe circa pari a $n^2/2$.

Viene ora riportato uno script che implementa l'algoritmo appena descritto.

```
function x = fLez06_SistemaTriSup(A,b)

n = length(b); % dimensione della matrice di sistema
x = zeros(n,1); % inizializzo il vettore soluzione con tutti zeri

x(n) = b(n)/A(n,n); % primo passo: risolvo l'equazione indipendente dalle altre

for i = n-1:-1:1 % ciclo esterno: i va da n-1 a 1, a passi di -1

    % calcolo del s_i : valore di s alla i-esima iterazione
    s = 0;
    for j = i+1:n
        s = s+A(i,j)*x(j);
    end

    % calcolo della componente i-esima del vettore delle soluzioni
    x(i) = (b(i)-s)/A(i,i);

end
```

Si tenga presente che MATLAB® permette di effettuare in maniera più astuta (ottimizzata) il calcolo del termine s_i. In effetti, è possibile rimpiazzare il segmento di codice

```
s = 0;
for j = i+1:n
    s = s+A(i,j)*x(j);
end
```

semplicemente con

```
s = A(i,i+1:n)*x(i+1:n)
```

che effettua le operazioni di somma e moltiplicazione utilizzando il prodotto riga per colonna * di MATLAB® . Si sappia tuttavia che questo è un dettaglio implementativo, che sfrutta il fatto che i prodotti riga per colonna sono implementati *built-in* in MATLAB® , e quindi più veloci per il semplice fatto che sono stati ottimizzati a livello di programmazione; dal punto di vista del costo computazionale, il numero di operazioni effettuato è sempre lo stesso!

Un metodo del tutto analogo può essere applicato anche alle matrici triangolari inferiori, tenendo però conto delle differenze con i sistemi triangolari superiori. Nei sistemi triangolari inferiori, la prima equazione è quella disaccoppiata da tutte le altre, e quindi si dovrà procedere *in avanti*, anziché *all'indietro*, sfruttando il fatto che la i-esima equazione è accoppiata solo alle precedenti, e non alle successive, al contrario di come capita nei sistemi triangolari superiori.

3.3 Metodo delle eliminazioni di Gauss

Se dovessimo rispondere alla domanda

> «Dovendo risolvere un generico sistema lineare, quali algoritmi conosciamo?»

La risposta, probabilmente, sarebbe **la regola di Cramer**[5]. Tuttavia[6] il calcolo del determinante, necessario per effettuare questa procedura, è estremamente pesante dal punto di vista computazionale. L'applicazione della regola di Laplace, infatti, ha un costo fattoriale, ossia circa $n!$ operazioni richieste: è totalmente inaccettabile per un computer! Tanto per capirci, immaginando che il nostro calcolatore sia in grado di lavorare ai PHz[7], la soluzione di un sistema con 24 equazioni in 24 incognite mediante regola di Cramer richiederebbe 50 anni. Questo, ci insegni la seguente regola del pollice: **gli algoritmi a costo esponenziale o peggio ancora fattoriale sono terribili e vanno evitati, sempre.**

Facciamo ora un passo indietro: nella precedente sezione abbiamo dedicato la nostra attenzione alle tecniche di soluzione di sistemi lineari triangolari, arrivando a trovare un costo computazionale circa pari a $n^2/2$.

[5] quella basata sul calcolo dei determinanti, si veda per esempio https://it.wikipedia.org/wiki/Regola_di_Cramer

[6] a meno di furberie che introdurremo tra qualche sezione

[7] PHz sta per petahertz, ossia un milione di GHz, che significherebbe eseguire 10^{15} operazioni al secondo; per dare un'idea, i computer attuali lavorano a 2-3 GHz :-/

«Sì va beh ma cosa c'entra? Cioè, vogliamo dire che ora tutti i sistemi lineari sono triangolari?»

Beh, no: non è che capiti così spesso di aver a che fare con un sistema lineare *nativamente* triangolare. Tuttavia, la tecnica più diffusa per la soluzione di un generico sistema lineare si basa sul **trasformare il sistema iniziale in un sistema triangolare equivalente** per poi applicare le tecniche studiate nella precedente sezione. Se avessimo il nostro fantacalcolatore da 1 PHz, invece di 50 anni avremmo bisogno di 10^{-12} secondi. Un po' meglio, no? ;-)

Abbiamo appena nominato il concetto di **equivalenza** tra due sistemi lineari. In particolare, un sistema lineare è equivalente a un altro se, pur essendo questi diversi tra loro, presentano la stessa soluzione. Dall'Algebra Lineare, in effetti, dovrebbe essere noto che scambiando l'ordine di due righe della matrice e del termine noto, o riscrivendo un'equazione come una combinazione lineare di sé stessa e di un'altra equazione, il sistema lineare di partenza e quello così ottenuto sono **equivalenti**, ovvero, hanno la stessa soluzione. Basandoci su questo principio, proporremo ora un **algoritmo** atto a trasformare, mediante un approccio **sistematico**, un sistema lineare da generico a triangolare.

Poniamoci in particolare l'obiettivo di ottenere un sistema triangolare superiore a partire da un sistema generico. In questo senso il nostro algoritmo dovrà, per ogni colonna della matrice associata al sistema, effettuare un certo numero di operazioni atto a introdurre degli zeri al posto dei suoi elementi. Il metodo richiede un certo numero di passi, che dipende sostanzialmente dalla dimensione della matrice. Verrebbe dunque da pensare che il metodo che stiamo per studiare sia *iterativo*, poiché non si risolve *in un colpo solo*, ma **no**: si sa a priori quanti passi servono, quindi è un metodo diretto o, anzi, il **principe** dei metodi diretti!

Probabilmente, al fine di fissare le idee, la cosa migliore è investire una certa quantità di tempo su un esercizio per poi cercare di riassumere l'algoritmo in maniera più schematica. Il nostro obiettivo sarà dunque risolvere il sistema lineare

$$\underbrace{\begin{bmatrix} 2 & -1 & 1 & -2 \\ 0 & 2 & 0 & -1 \\ 1 & 0 & -2 & 1 \\ 0 & 2 & 1 & 1 \end{bmatrix}}_{\underline{\underline{A}}} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}}_{\underline{x}} = \underbrace{\begin{bmatrix} 0 \\ 1 \\ 0 \\ 4 \end{bmatrix}}_{\underline{b}}.$$

Per questioni di notazione, diciamo che la prima operazione consiste semplicemente nel definire la matrice di sistema **all'inizio del primo passo** $\underline{\underline{A}}^{(1)}$, e il corrispondente termine noto $\underline{b}^{(1)}$, come quelli del sistema originale:

$$\underbrace{\begin{bmatrix} 2 & -1 & 1 & -2 \\ 0 & 2 & 0 & -1 \\ 1 & 0 & -2 & 1 \\ 0 & 2 & 1 & 1 \end{bmatrix}}_{\underline{\underline{A}}^{(1)}} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}}_{\underline{x}} = \underbrace{\begin{bmatrix} 0 \\ 1 \\ 0 \\ 4 \end{bmatrix}}_{\underline{b}^{(1)}}.$$

Dal momento che la matrice è 4×4, avremo bisogno di 3 (ovvero $n-1$) passi per portare il sistema a triangolare. L'obiettivo del k-esimo passo è prendere la k-esima colonna e modificarla in modo tale che le sue componenti, dalla $(k+1)$-esima all'ultima, diventino 0. Per ogni k-esimo passo, toccheremo dunque tutte le righe a partire dalla $(k+1)$-esima fino all'ultima. Prima di tutto, guardiamo la matrice, e notiamo che $a_{kk}^{(k)}$, ossia, l'elemento alla posizione (k,k) della matrice di partenza al k-esimo passo, è diverso da 0; questo ci serve, perché tra poco dovremo calcolare dei coefficienti che contengono $a_{kk}^{(k)}$ al denominatore e quindi questo non può essere nullo. Nel caso questo fosse stato nullo, avremmo dovuto effettuare uno scambio di righe della matrice (e del termine noto) con la prima riga successiva avente $a_{ik}^{(k)}$ diverso da 0. Questa **dovrà** esistere, altrimenti la matrice avrebbe un'intera colonna piena di zeri, quindi avrebbe determinante uguale a 0 e la soluzione del sistema non sarebbe unica.

Detto questo, per ogni i-esima riga della k-esima colonna, calcoliamo un coefficiente m_{ik} definito come

$$m_{ik} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}.$$

In particolare, per $k = 1$, abbiamo:

$$m_{21} = 0; \qquad m_{31} = \frac{1}{2}; \qquad m_{41} = 0.$$

Questi m_{ik} sono i coefficienti della combinazione lineare che useremo per modificare ciascuna delle i-esime righe (sotto la k-esima). Il caso in cui m_{ik} è uguale a 0 è molto gradito, dal momento che significa *non toccare la i-esima riga* (è già a 0, quindi è già a posto). Per $k = 1$ quindi, in questo esempio, solo la terza riga va modificata.

La modifica alla i-esima riga consiste nel prendere ciascuna delle sue componenti e sottrarvi la componente della k-esima riga sulla medesima colonna moltiplicata per m_{ik}. Considerando quindi la terza riga (l'unica legata a un coefficiente non nullo), l'operazione da fare sarà la seguente sostituzione:

$$\begin{bmatrix} 1 & 0 & -2 & 1 \end{bmatrix} \Longrightarrow \begin{bmatrix} 1 & 0 & -2 & 1 \end{bmatrix} - \frac{1}{2}\begin{bmatrix} 2 & -1 & 1 & -2 \end{bmatrix} = \begin{bmatrix} 0 & -\left(\frac{1}{2}\right)(-1) & -2-\frac{1}{2} & 1+1 \end{bmatrix},$$

dove abbiamo ottenuto il nostro obiettivo: abbiamo il primo elemento uguale a 0! Ovviamente, come già detto, anche il termine noto va modificato; in particolare,

$$b_3^{(2)} \Longrightarrow b_3^{(1)} - m_{31} b_1^{(1)},$$

che porta a ottenere

$$b_3^{(2)} = 0,$$

essendo che sia $b_1^{(1)}$ sia $b_3^{(1)}$ sono a 0. La **fortunata** (ma anche **fortuita**) situazione $b_k^{(k)} = 0$ è interessante perché permette di non sostituire **alcuna** componente del vettore, anche per le componenti aventi $m_{ik} \neq 0$. Per riassumere, dopo il passo $k = 1$, il sistema lineare è diventato

$$\underbrace{\begin{bmatrix} 2 & -1 & 1 & -2 \\ 0 & 2 & 0 & -1 \\ 0 & \frac{1}{2} & -\frac{5}{2} & 2 \\ 0 & 2 & 1 & 1 \end{bmatrix}}_{\underline{\underline{A}}^{(2)}} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}}_{\underline{x}} = \underbrace{\begin{bmatrix} 0 \\ 1 \\ 0 \\ 4 \end{bmatrix}}_{\underline{b}^{(2)}},$$

dove gli apici (2) indicano che queste matrici, ottenute dopo aver effettuato il primo passo, **sono il punto di partenza del passo 2**. Come scopriremo tra non molto, è utile tenere traccia dei coefficienti che abbiamo utilizzato; la notazione m_{ik} si presta infatti a inserirli in una matrice; iniziamo quindi a riempire una certa matrice $\underline{\underline{L}}$ contenente ciascuno di questi m_{ik}:

$$\underline{\underline{L}}^{(2)} = \begin{bmatrix} * & * & * & * \\ 0 & * & * & * \\ \frac{1}{2} & * & * & * \\ 0 & * & * & * \end{bmatrix}.$$

Il passo $k = 1$ è ora davvero completato. Possiamo procedere con il passo $k = 2$. A questo fine, scriviamo la matrice $\underline{\underline{A}}^{(2)}$ evidenziando alcuni elementi con diversi colori:

$$\begin{bmatrix} 2 & -1 & 1 & -2 \\ 0 & 2 & 0 & -1 \\ 0 & \frac{1}{2} & -\frac{5}{2} & 2 \\ 0 & 2 & 1 & 1 \end{bmatrix}.$$

La prima colonna è già a posto: contiene tutti zeri! Dunque, dovremo preoccuparci di lavorare solo sulla sottomatrice 3×3 degli elementi evidenziati in blu. In dettaglio, il nostro obiettivo sarà far sì che la seconda colonna, dalla terza componente (inclusa) in poi, contenga solo zeri. Per far questo, procediamo analogamente a prima; prima di tutto, calcoliamo gli m_{ik}:

$$m_{32} = \frac{\frac{1}{2}}{2} = \frac{1}{4}; \qquad m_{42} = \frac{2}{2} = 1.$$

Dovremo sostituire, proprio come prima,

$$\begin{bmatrix} \frac{1}{2} & -\frac{5}{2} & -2 \end{bmatrix} \Longrightarrow \begin{bmatrix} \frac{1}{2} & -\frac{5}{2} & -2 \end{bmatrix} - \frac{1}{4}\begin{bmatrix} 2 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 0 & -\frac{5}{2} - 0 & 2 + \frac{1}{4} \end{bmatrix}$$

e

$$\begin{bmatrix} 2 & 1 & 1 \end{bmatrix} \Longrightarrow \begin{bmatrix} 2 & 1 & 1 \end{bmatrix} - 1\begin{bmatrix} 2 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 2 \end{bmatrix},$$

che si possono includere nella sottomatrice 3×3 trasformando quella prima evidenziata in blu nella seguente:

$$\begin{bmatrix} 2 & 0 & -1 \\ \frac{1}{2} & -\frac{5}{2} & 2 \\ 2 & 1 & 1 \end{bmatrix} \Longrightarrow \begin{bmatrix} 2 & 0 & -1 \\ 0 & -\frac{5}{2} & \frac{9}{4} \\ 0 & 1 & 2 \end{bmatrix}.$$

Per quanto riguarda il termine noto, dovremo effettuare un'operazione simile; non avendo questa volta il fortuito caso $b_k^{(k)} = 0$, dovremo fare un conto analogo a quello appena svolto:

$$\underline{b}^{(3)} = \begin{bmatrix} 0 \\ 1 \\ 0 - \frac{1}{4} \\ 4 - 1 \end{bmatrix}.$$

Sostituendo la sottomatrice 3×3 nella parte di $\underline{\underline{A}}^{(2)}$ evidenziata in blu e usando questo $\underline{b}^{(2)}$, è possibile ottenere il risultato del secondo passo

$$\underbrace{\begin{bmatrix} 2 & -1 & 1 & -2 \\ 0 & 2 & 0 & -1 \\ 0 & 0 & -\frac{5}{2} & \frac{9}{4} \\ 0 & 0 & 1 & 2 \end{bmatrix}}_{\underline{\underline{A}}^{(3)}} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}}_{\underline{x}} = \underbrace{\begin{bmatrix} 0 \\ 1 \\ -\frac{1}{4} \\ 3 \end{bmatrix}}_{\underline{b}^{(3)}},$$

e possiamo anche *mettere da parte* i moltiplicatori di questo secondo passo, aggiornando la matrice $\underline{\underline{L}}^{(2)}$ a $\underline{\underline{L}}^{(3)}$:

$$\underline{\underline{L}}^{(3)} = \begin{bmatrix} * & * & * & * \\ 0 & * & * & * \\ \frac{1}{2} & \frac{1}{4} & * & * \\ 0 & 1 & * & * \end{bmatrix}.$$

Questo decreta la fine del secondo passo. Possiamo ora quindi procedere con l'ultimo passo, $k = 3$. Per questo, rivediamo la matrice $\underline{\underline{A}}^{(3)}$, per capire cosa dobbiamo andare a toccare.

$$\begin{bmatrix} 2 & -1 & 1 & -2 \\ 0 & 2 & 0 & -1 \\ 0 & 0 & -\frac{5}{2} & \frac{9}{4} \\ 0 & 0 & 1 & 2 \end{bmatrix}.$$

La prima e la seconda colonna sono a posto. Rimane quindi da lavorare solamente sulla sottomatrice 2×2 evidenziata in blu. Per questo motivo, l'unico coefficiente da calcolare sarà

$$m_{43} = \frac{1}{-\frac{5}{2}} = -\frac{2}{5}.$$

Procedendo come prima, quindi, sostituiamo all'ultima riga sé stessa, meno la penultima moltiplicata per il coefficiente:

$$\begin{bmatrix} 1 & 2 \end{bmatrix} \Longrightarrow \begin{bmatrix} 1 & 2 \end{bmatrix} - \left(-\frac{2}{5}\right)\begin{bmatrix} -\frac{5}{2} & \frac{9}{4} \end{bmatrix} = \begin{bmatrix} 0 & \frac{29}{10} \end{bmatrix}.$$

Questa si può includere nella sottomatrice 2×2 trasformando quella prima evidenziata in blu nella seguente:

$$\begin{bmatrix} -\frac{5}{2} & \frac{9}{4} \\ 1 & 2 \end{bmatrix} \Longrightarrow \begin{bmatrix} -\frac{5}{2} & \frac{9}{4} \\ 0 & \frac{29}{10} \end{bmatrix}.$$

Anche per il termine noto, si potrà procedere come prima, ottenendo

$$\underline{b}^{(4)} = \begin{bmatrix} 0 \\ 1 \\ -\frac{1}{4} \\ 3 - \left(-\frac{2}{5}\right)\left(-\frac{1}{4}\right) \end{bmatrix}.$$

Sostituendo la sottomatrice 2×2 e il termine noto, è possibile ottenere quindi il seguente sistema lineare:

$$\underbrace{\begin{bmatrix} 2 & -1 & 1 & -2 \\ 0 & 2 & 0 & -1 \\ 0 & 0 & -\frac{5}{2} & \frac{9}{4} \\ 0 & 0 & 0 & \frac{29}{10} \end{bmatrix}}_{\underline{\underline{A}}^{(4)}} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}}_{\underline{x}} = \underbrace{\begin{bmatrix} 0 \\ 1 \\ -\frac{1}{4} \\ \frac{29}{10} \end{bmatrix}}_{\underline{b}^{(4)}},$$

in cui è palese che $\underline{\underline{A}}^{(4)}$, che sarebbe il punto di partenza dell'ipotetico passo $k = 4$, sia una matrice triangolare superiore: abbiamo ottenuto il nostro obiettivo! Il passo $k = 4$ dunque non verrà effettuato, e noi considereremo $\underline{\underline{A}}^{(4)}$ e $\underline{b}^{(4)}$ come il nostro punto di arrivo. Per concludere, però, salviamo nella matrice $\underline{\underline{L}}^{(4)}$ anche l'ultimo coefficientino, ottenendo

$$\underline{\underline{L}}^{(4)} = \begin{bmatrix} * & * & * & * \\ 0 & * & * & * \\ \frac{1}{2} & \frac{1}{4} & * & * \\ 0 & 1 & -\frac{2}{5} & * \end{bmatrix}.$$

A questo punto, il sistema potrebbe essere risolto con la tecnica di sostituzione all'indietro, per scoprire che la soluzione in questo caso è $[1 \ 1 \ 1 \ 1]^{\mathrm{T}}$.

Cerchiamo ora di riassumere l'idea del metodo.

1. Prima di tutto, controlliamo che $a_{kk}^{(k)}$ sia diverso da 0; se questo elemento fosse nullo, scambiamo la k-esima riga con la prima successiva, sia questa la i-esima, avente $a_{ik}^{(k)} \neq 0$.
2. Calcolo m_{ik} per ogni i-esima riga, $i = k + 1, ..., n$

$$m_{ik} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}.$$

3. Rimpiazzo la i-esima riga, con $i = k + 1, ..., n$ con sé stessa, a cui sottraggo m_{ik} che moltiplica la k-esima riga. Faccio lo stesso per ciascuna i-esima componente del termine noto.
4. Proseguo fino a quando la matrice non diventa triangolare superiore; questo richiede $n - 1$ passi, dove n è il numero di righe e/o colonne della matrice $\underline{\underline{A}}$.

Questo algoritmo viene detto **metodo delle eliminazioni di Gauss** e, nel caso più generale ora mostrato, il suo costo computazionale è $n^3/3$. Infatti, un'implementazione di questo algoritmo, come si può intuire studiandolo, richiederebbe tre cicli annidati, e dunque sarebbe necessario un numero di operazioni aritmetiche elementari prossimo al cubo del numero delle righe della matrice.

Per chiarire alcuni punti, diversi testi, si veda per esempio [1, p. 127], riportano che il costo della fattorizzazione LU è circa pari a $2n^3/3$ anziché $n^3/3$. Questo testo si basa su [2, p. 45], che riporta esattamente il numero di operazioni richieste al fine di calcolare la fattorizzazione LU, e al termine di esse, indica $n^3/3$ come costo computazionale. Tutti i costi computazionali riportati su questo testo seguono questo testo come riferimento, e sono tra loro consistenti.

3.3.1 Alcune note su ciò che abbiamo appena fatto

Prima di tutto, osserviamo che, nonostante su di noi incombesse la minaccia di avere $a_{kk}^{(k)} = 0$ e quindi dover operare scambi di righe, questo non è successo. Esistono situazioni in cui si ha la certezza che questa cosa possa capitare; in particolare, questa certezza si ha quando:

- La matrice $\underline{\underline{A}}$ è a diagonale dominante per righe.
- La matrice $\underline{\underline{A}}$ è a diagonale dominante per colonne.
- La matrice $\underline{\underline{A}}$ è simmetrica definita positiva.

Si noti che la $\underline{\underline{A}}$ del nostro esempio non soddisfa alcune di queste condizioni, ma comunque non è stato necessario operare scambi: è stato un caso fortuito. Quando si ha una di queste condizioni si sa *a priori* che non serve operare scambi e quindi si ha la certezza che l'algoritmo è più semplice.

Un'altra osservazione: se la matrice $\underline{\underline{A}}$ è simmetrica, e non si effettuano scambi (dovuti a $a_{kk}^{(k)} = 0$), la sottomatrice sarà ancora simmetrica, **a ogni passo**; questo permette di calcolare meno elementi e, quindi, risparmiare conti: in particolare il costo delle eliminazioni di Gauss si dimezza (e si arriva così a $n^3/6$).

Un'osservazione conclusiva riguarda un metodo lievemente diverso di memorizzare gli elementi delle matrici. In particolare, è possibile, in alcuni codici, trovare come output una matrice nella forma

$$\begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ m_{21} & u_{22} & u_{23} & u_{24} \\ m_{31} & m_{32} & u_{33} & u_{34} \\ m_{41} & m_{42} & m_{43} & u_{44} \end{bmatrix} = \begin{bmatrix} 2 & -1 & 1 & -2 \\ 0 & 2 & 0 & -1 \\ \frac{1}{2} & \frac{1}{4} & -\frac{5}{2} & \frac{9}{4} \\ 0 & 1 & -\frac{2}{5} & \frac{29}{10} \end{bmatrix},$$

dove abbiamo mostrato a membro destro come si presenterebbe nel caso del nostro esempio. Al posto di memorizzare gli zeri, sono stati memorizzati i moltiplicatori, i coefficienti della combinazione lineare; in questo modo, nel caso di sistemi molto grossi, si può evitare di memorizzare due matrici e quindi risparmiare memoria.

3.3.2 Matrici triangolari e come ottenerle

Ci siamo presi la premura di salvare i coefficienti della combinazione lineare m_{ij}, e li abbiamo presi talmente in considerazione da porci il problema di memorizzarli in maniera efficiente. Ma perché? A cosa servono?

Consideriamo le seguenti due matrici.

$$\underline{\underline{L}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \frac{1}{2} & \frac{1}{4} & 1 & 0 \\ 0 & 1 & -\frac{2}{5} & 1 \end{bmatrix} \qquad \underline{\underline{U}} = \begin{bmatrix} 2 & -1 & 1 & -2 \\ 0 & 2 & 0 & -1 \\ 0 & 0 & -\frac{5}{2} & \frac{9}{4} \\ 0 & 0 & 0 & \frac{29}{10} \end{bmatrix}.$$

Da dove nascono queste matrici? Beh, $\underline{\underline{L}}$ è semplicemente la matrice contenente tutti i coefficienti delle combinazioni lineari, con, in più, tutti uni sulla diagonale. $\underline{\underline{U}}$, invece, non è altro che $\underline{\underline{A}}^{(4)}$, ovvero la

matrice triangolare superiore ottenuta dalle eliminazioni di Gauss. In altre parole, tutti gli elementi di queste matrici non sono altro che risultati del metodo delle eliminazioni di Gauss. A questo punto, mi chiedo: quanto vale il loro prodotto, ovvero, $\underline{\underline{L}}\,\underline{\underline{U}}$? Un segmento di codice, e le sue uscite, valgono più di mille parole. A questo scopo, implementiamo, per l'esempio che abbiamo preso in considerazione, un codice che calcoli questo prodotto:

```
>> L = [1     0   0   0;
        0     1   0   0;
       1/2  1/4   1   0;
        0     1 -2/5  1];

>> U = [2  -1   1   -2;
        0    2   0   -1;
        0    0 -5/2  9/4;
        0    0   0   29/10];

>> L*U
ans =
    2.0000   -1.0000    1.0000   -2.0000
         0    2.0000         0   -1.0000
    1.0000         0   -2.0000    1.0000
         0    2.0000    1.0000    1.0000
>>
```

Cioè, abbiamo ritrovato la matrice $\underline{\underline{A}}$ di partenza! Cioè. Abbiamo scritto la matrice $\underline{\underline{A}}$ come il prodotto di due fattori: una matrice triangolare inferiore $\underline{\underline{L}}$, e una matrice triangolare superiore $\underline{\underline{U}}$. Dal momento che $\underline{\underline{L}}$ e $\underline{\underline{U}}$ dunque **fattorizzano** la matrice $\underline{\underline{A}}$, con grande fantasia, questa fattorizzazione è stata chiamata **fattorizzazione LU**. Le lettere L e U identificano *lower* e *upper*, come dire *triangolare inferiore* e *triangolare superiore*. Abbiamo capito, ora, a cosa serviva tenere in memoria i moltiplicatori. ;-)

Dopo questo esempio un po' noioso, inizia la parte davvero interessante dell'argomento, poiché ora potremo provare questa fattorizzazione per risolvere dei veri problemi, e iniziare a studiare altre fattorizzazioni valide in altre situazioni.

3.4 La fattorizzazione LU

Abbiamo concluso la precedente sezione introducendo uno dei concetti più potenti e secondo me interessanti nell'ambito della soluzione dei sistemi lineari. La nostra intenzione iniziale era infatti semplicemente trasformare un generico sistema lineare con matrice associata $\underline{\underline{A}}$ in un sistema triangolare superiore con matrice associata $\underline{\underline{U}}$. Tuttavia, abbiamo scoperto che, salvando i coefficienti m_{ik} (di fatto un sottoprodotto delle eliminazioni di Gauss) in una matrice $\underline{\underline{L}}$, che abbiamo visto essere triangolare inferiore, abbiamo ottenuto la possibilità di fattorizzare una matrice in termini di un prodotto tra una matrice triangolare inferiore e di una triangolare superiore: la fattorizzazione LU. Dopo aver aggiunto alcuni dettagli su una sua variante, procederemo con lo studio delle sue applicazioni in vari contesti.

3.4.1 Pivoting parziale: fattorizzazione PA = LU

Nella precedente sezione abbiamo visto che i coefficienti m_{ik} van calcolati con la formula

$$m_{ik} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}},$$

che, nel caso $a_{kk}^{(k)} = 0$, dà luogo a una divisione per zero. Allora, nell'esercizio che abbiamo risolto passo-passo per comprendere l'algoritmo, non abbiamo trovato il caso $a_{kk}^{(k)} = 0$; abbiamo tuttavia detto che, nel caso questo si fosse verificato, avremmo dovuto scambiare la k-esima riga con la prima riga successiva i tale per cui $a_{ik}^{(k)} \neq 0$ (e abbiamo detto che questa **deve** esistere, altrimenti la matrice avrebbe determinante nullo e il sistema non avrebbe una soluzione unica).

Cercando a questo punto di ragionare più *numericamente*, diciamo che $a_{kk}^{(k)}$ potrebbe essere non proprio zero, ma *piccolo*; in questo caso, potremmo comunque andare incontro a dei problemini, dal momento che *dividere per un numero piccolo è un po' come moltiplicare per un numero grande* e, insomma, potremmo comunque avere una qualche sorta di problema di instabilità numerica. Per questo motivo, anche se $a_{kk}^{(k)} \neq 0$, può essere opportuno effettuare comunque uno scambio atto a migliorare la stabilità numerica. In particolare, potremmo pensare di scambiare la k-esima riga con la r-esima tale per cui

$$\left|a_{rk}^{(k)}\right| = \max_{k \leq i \leq n} \left|a_{ik}^{(k)}\right|,$$

ossia, si sostituisce alla k-esima riga la riga successiva, così denominata r-esima, tale per cui l'elemento è il massimo (in modulo); in questo modo, si sta dividendo per il più grande numero possibile, evitando quindi divisioni per numeri, se non pari a zero, vicini a esso. Volendo considerare un esempio, una matrice tipo

$$\underline{\underline{A}}_1 = \begin{bmatrix} 5 & 10 & 12 \\ 2 & 3 & 5 \\ 1 & 2 & 1 \end{bmatrix}$$

non richiederebbe pivoting parziale al primo passo dell'algoritmo, dal momento che l'elemento $a_{1,1}$ è già il più grande (in modulo) della sua colonna. Se invece avessimo avuto una matrice nella forma

$$\underline{\underline{A}}_2 = \begin{bmatrix} 2 & 3 & 5 \\ 5 & 10 & 12 \\ 1 & 2 & 1 \end{bmatrix},$$

sarebbe stato conveniente scambiare la prima riga con la seconda, in modo da avere il 5 come elemento **pivot**.

Questa strategia viene detta **pivoting parziale**, dove con il termine **pivot** si intende l'elemento $a_{kk}^{(k)}$, ovvero l'elemento su cui fa perno l'algoritmo delle eliminazioni di Gauss per procedere[8].

Si può dimostrare che il pivoting è superfluo in alcune situazioni, in particolare:

- quando la matrice di sistema $\underline{\underline{A}}$ è a diagonale dominante per colonne;
- quando la matrice di sistema $\underline{\underline{A}}$ è simmetrica definita positiva.

E se tutti i *candidati pivot* fossero piccoli rispetto agli altri elementi della matrice? Cioè, se fossero tutti diversi da zero, ma tutti piccoli? Beh, questo significherebbe che il sistema è mal condizionato! Infatti, dire che un'intera colonna ha elementi non proprio pari a 0, ma comunque molto piccoli, significherebbe che il determinante della matrice non sarebbe proprio 0, ma sarebbe molto piccolo. Di conseguenza, l'inversa di questa matrice sarà mal definita e, in generale, potrebbe essere molto grande[9]. Ma quindi il numero di condizionamento, che ricordiamo essere definito come

$$\kappa(\underline{\underline{A}}) = \left\|\underline{\underline{A}}\right\| \left\|\underline{\underline{A}}^{-1}\right\|,$$

avrà la seconda norma molto grande, e quindi di conseguenza anche il numero di condizionamento sarà grande.

Per concludere, proponiamo una rappresentazione sintetica di quanto abbiamo appena descritto, sia che lo scambio di righe derivi dalla situazione $a_{kk}^{(k)} = 0$, sia che derivi dall'applicazione della strategia del pivoting parziale. Un sinonimo di *scambiare*, quando parlo delle righe, è *permutare*. Non è che lo stia dicendo perché io voglia fare in qualche modo concorrenza al prof. Tullio De Mauro, quanto per ricordarvi che ho recentemente parlato del concetto di **matrice di permutazione** $\underline{\underline{P}}$. Ricordo che una matrice di permutazione, moltiplicata da sinistra, permette di scambiare gli elementi di un vettore o -allo stesso modo- le righe di una matrice. Questo significa che possiamo assumere che $\underline{\underline{P}}$ contenga tutte le operazioni di permutazione da effettuare al fine di migliorare il nostro metodo numerico.

Gli elementi della fattorizzazione LU di una matrice $\underline{\underline{A}}$ si possono determinare, con MATLAB®, mediante il comando

[8] La parola **pivot** deriva dalla traduzione in lingua francese della parola **perno**; viene usata molto spesso anche in ambito sportivo, come nella pallamano o nella pallacanestro. Il pivot, nel basket di una volta, era l'elemento chiave della squadra; si pensi per esempio ai tempi di Kareem Abdul-Jabbar, di Hakeem Olajuwon o del più recente Shaquille O'Neal: giocatori alti grossi e forti che tirano da sotto canestro. Credo che questo ruolo inteso in questo modo sia andato un po' perduto anche perché le squadre puntano più sul tiro da tre punti piuttosto che su un attacco da sotto canestro.

[9] si ricordi che nel calcolo della matrice inversa è contenuto il reciproco del determinante

```
[L,U,P] = lu(A);
```

3.4.2 Applicazioni della fattorizzazione PA = LU

Una volta descritto cosa sia questa fattorizzazione, vogliamo spiegare a cosa serve. Vediamo quindi un po' di applicazioni.

Soluzione di un sistema lineare

Si immagini di dover risolvere il sistema lineare

$$\underline{\underline{A}}\,\underline{x} = \underline{b},$$

e di avere, grazie al comando `lu` di MATLAB® , le matrici $\underline{\underline{P}}$, $\underline{\underline{L}}$, $\underline{\underline{U}}$. Ciò che possiamo fare è quindi, partendo da questa espressione del sistema lineare, moltiplicare da sinistra per la matrice di permutazione $\underline{\underline{P}}$, ottenendo

$$\underline{\underline{P}}\,\underline{\underline{A}}\,\underline{x} = \underline{\underline{P}}\,\underline{b}.$$

Dal momento che $\underline{\underline{P}}\,\underline{\underline{A}} = \underline{\underline{L}}\,\underline{\underline{U}}$, è possibile modificare il membro sinistro:

$$\underline{\underline{L}}\,\underline{\underline{U}}\,\underline{x} = \underline{\underline{P}}\,\underline{b}.$$

A questo punto, guardiamo con calma questo sistema. Possiamo dire che $\underline{\underline{U}}\,\underline{x}$ è un vettore colonna; volendogli dare un nome, chiamiamolo $\underline{y}$; allo stesso modo, $\underline{\underline{P}}\,\underline{b}$ è un vettore colonna, che possiamo per esempio chiamare $\underline{c}$:

$$\underline{y} \triangleq \underline{\underline{U}}\,\underline{x} \qquad \underline{c} \triangleq \underline{\underline{P}}\,\underline{b}. \tag{3.10}$$

Quindi, il sistema si può riscrivere come

$$\underline{\underline{L}}\,\underline{y} = \underline{c}. \tag{3.11}$$

Come accennato qualche sezione fa, in MATLAB® il comando \ permette di risolvere un sistema lineare nella forma $\underline{\underline{A}}\,\underline{x} = \underline{b}$, che guarda caso è proprio quella appena scritta. Usando dunque il comando

```
y = L\c;
```

possiamo risolvere il sistema triangolare inferiore (3.11) e trovare il vettore $\underline{y}$. A questo punto, però, possiamo ricordare le definizioni (3.10), che scrivono che

$$\underline{\underline{U}}\,\underline{x} = \underline{y},$$

dove però ora $\underline{y}$ è noto dalla soluzione di (3.11). Quindi, possiamo usare il comando MATLAB®

```
x = U\y;
```

per trovare la soluzione del sistema di partenza.

Riassumendo, una volta calcolata la fattorizzazione PA = LU, che come abbiamo visto ha un costo computazionale circa pari a $n^3/3$, abbiamo a disposizione i due fattori triangolari. Con un po' di manipolazioni abbiamo quindi capito che, **una volta noti questi due fattori**, il sistema lineare si può risolvere **mediante la soluzione di due sistemi triangolari**, ciascuno dei quali ha costo computazionale circa pari a $n^2/2$, ottenendo quindi alla fine solo un costo pari a n^2, che è decisamente meglio di $n^3/3$.

Il comando \, nella sua apparente banalità, è strutturato in maniera molto intelligente, dal momento che prima trova l'algoritmo ottimo per il problema, e poi agisce. Nel caso più generale, applica la decomposizione PA = LU. Nel caso appena proposto, capisce che $\underline{\underline{L}}$ e $\underline{\underline{U}}$ sono matrici triangolari, e quindi utilizzerà il metodo delle sostituzioni in avanti/indietro.

Soluzione di più sistemi lineari con stessa matrice

Un buon osservatore potrebbe obiettare che se non avessimo assunto per ipotesi di conoscere i fattori $\underline{\underline{P}}$, $\underline{\underline{L}}$, $\underline{\underline{U}}$, avremmo potuto risolvere questo esercizio scrivendo direttamente il comando

```
x = A\b;
```

senza farci tutte queste domande e tutti questi conti sulla fattorizzazione, dal momento che MATLAB® li fa già internamente. Esistono però situazioni in cui capita di dover risolvere p sistemi caratterizzati dalla stessa matrice $\underline{\underline{A}}$, che dunque differiscono solo per il termine noto[10]. In questa situazione

- se usassi p volte il comando `x_i = A\b_i` farei calcolare a MATLAB le eliminazioni di Gauss ogni volta; in questo modo, il costo computazionale complessivo sarebbe $pn^3/3$, ossia p volte quello delle eliminazioni di Gauss;
- se invece calcolassi una volta la fattorizzazione LU (mediante eliminazioni di Gauss, ovviamente) e poi la tenessi in memoria e la utilizzassi per risolvere $2p$ sistemi triangolari (per ogni sistema bisogna risolvere 2 sistemi triangolari, come spiegato nella precedente sezione), il costo computazionale complessivo sarebbe $n^3/3 + pn^2$. Questo significa che se p è piccolo, ovvero se non ho molti sistemi lineari (o se hanno dimensione n grossa), è come risolvere solo 1 sistema lineare anziché p :-) .

Calcolo del determinante di una matrice

Come già anticipato, la regola di Laplace non è il metodo più da *pane e volpe* che ci sia per calcolare il determinante di una matrice $\underline{\underline{A}}$. Tuttavia, esiste almeno una situazione in cui calcolare il determinante diventa cosa facile: il caso in cui la matrice è triangolare. Infatti, è possibile dimostrare, applicando la regola di Laplace, che il determinante di una matrice triangolare è semplicemente dato dal prodotto degli elementi della diagonale principale[11]. Come possiamo sfruttare questo fatto?

Il nostro obiettivo finale è calcolare $\det\{\underline{\underline{A}}\}$. Tuttavia, per raggiungerlo, poniamoci l'obiettivo di calcolare, al suo posto, $\det\{\underline{\underline{P}}\,\underline{\underline{A}}\}$, dove $\underline{\underline{P}}$ è la solita matrice di permutazione ottenuta mediante la fattorizzazione LU. Dall'Algebra Lineare, si sa che il determinante del prodotto di due matrici è pari al prodotto dei determinanti:

$$\det\{\underline{\underline{P}}\,\underline{\underline{A}}\} = \det\{\underline{\underline{P}}\}\det\{\underline{\underline{A}}\}. \tag{3.12}$$

In tutto questo, però, il determinante della matrice $\underline{\underline{P}}$ può essere uguale a -1 o a $+1$. Infatti, $\underline{\underline{P}}$ è ottenuta a partire dalla matrice identità $\underline{\underline{I}}$ (la quale ha determinante pari a $+1$), permutando un certo numero di righe. Come noto dall'Algebra Lineare, ogni volta che si permutano le righe di una matrice, il suo determinante cambia segno e, quindi, a seconda del numero di permutazioni, esso può valere ± 1. Se il numero di permutazioni effettuato è pari, allora $\det\{\underline{\underline{P}}\} = +1$; se dispari, $\det\{\underline{\underline{P}}\} = -1$. Considerando il seguente esempio di matrice

$$\underline{\underline{P}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

si può vedere a occhio che questa è stata ottenuta dalla matrice identità permutando la seconda e la terza riga, e quindi il numero di permutazioni s è pari a 1.

A questo punto riprendiamo (3.13) e sostituiamo il membro destro come segue:

$$\det\{\underline{\underline{P}}\}\det\{\underline{\underline{A}}\} = (-1)^s \det\{\underline{\underline{A}}\} = \det\{\underline{\underline{P}}\,\underline{\underline{A}}\} = \det\{\underline{\underline{L}}\,\underline{\underline{U}}\} = \det\{\underline{\underline{L}}\}\det\{\underline{\underline{U}}\},$$

quindi, portando il termine di segno al membro destro,

$$\det\{\underline{\underline{A}}\} = (-1)^s \det\{\underline{\underline{L}}\}\det\{\underline{\underline{U}}\}.$$

A questo punto, sfruttiamo il fatto che le matrici $\underline{\underline{L}}$ e $\underline{\underline{U}}$ sono triangolari, e questo permette di scrivere più esplicitamente il determinante come prodotto dei termini sulla diagonale:

[10] un esempio pratico verrà mostrato nell'ambito della stima numerica degli autovalori di una matrice

[11] la stessa cosa vale *ovviamente* anche per le matrici diagonali

$$\det\{\underline{\underline{A}}\} = (-1)^s \det\{\underline{\underline{L}}\} \det\{\underline{\underline{U}}\} = (-1)^s \left(\prod_{i=1}^{n} l_{ii}\right)\left(\prod_{i=1}^{n} u_{ii}\right).$$

Ricordiamo infine che la matrice $\underline{\underline{L}}$ ha tutti gli elementi della diagonale principale pari a 1 e, quindi, il suo determinante è pari a 1. Questo permette di concludere il nostro calcolo ottenendo

$$\det\{\underline{\underline{A}}\} = (-1)^s \prod_{i=1}^{n} u_{ii}. \tag{3.13}$$

Calcolo dell'inversa di una matrice

Il calcolo dell'inversa di una matrice $\underline{\underline{A}}$ non è certamente più semplice del calcolo del determinante: si pensi anche solo al fatto che il calcolo dell'inversa richiede il calcolo del determinante! ;-) Ammesso che dunque qualcuno dei lettori abbia *davvero* un buon motivo per invertire una matrice[12], la soluzione di questo problema ci è di nuovo fornita dalla fattorizzazione LU. Al fine di determinare $\underline{\underline{A}}^{-1}$, proviamo quindi a partire da

$$\underline{\underline{P}}\,\underline{\underline{A}} = \underline{\underline{L}}\,\underline{\underline{U}},$$

e proviamo a calcolare l'inversa di entrambi i membri, ottenendo

$$\left(\underline{\underline{P}}\,\underline{\underline{A}}\right)^{-1} = \left(\underline{\underline{L}}\,\underline{\underline{U}}\right)^{-1}.$$

A questo punto, dall'Algebra Lineare, dovrebbe essere noto che, quando calcolo l'inversa del prodotto di due matrici $\underline{\underline{A}}$ e $\underline{\underline{B}}$, ottengo

$$\left(\underline{\underline{A}}\,\underline{\underline{B}}\right)^{-1} = \underline{\underline{B}}^{-1}\underline{\underline{A}}^{-1},$$

ossia, oltre a invertire i singoli fattori, devo scambiarli di ordine. Questo ci porta a riscrivere la precedente espressione come

$$\underline{\underline{A}}^{-1}\underline{\underline{P}}^{-1} = \underline{\underline{U}}^{-1}\underline{\underline{L}}^{-1}.$$

A questo punto, moltiplichiamo a destra ambo i membri per $\underline{\underline{P}}$, ottenendo

$$\underline{\underline{A}}^{-1} = \underline{\underline{U}}^{-1}\underline{\underline{L}}^{-1}\underline{\underline{P}},$$

che è il metodo con cui MATLAB® calcola l'inversa di una matrice `A` quando lanciamo il comando `inv(A)`. Questa operazione, per quanto più vantaggiosa rispetto alle tecniche insegnate nell'ambito dell'Algebra Lineare, ha costo computazionale pari a n^3, esattamente come il prodotto righe per colonne di due matrici. Per questi motivi, è sconsigliabile passare per il calcolo della matrice inversa al momento di risolvere un sistema lineare.

3.4.3 Esempio: trasmissione del calore

Al fine di dare un po' di *feeling* in più su queste cose, e capire che non servono solo per fare sterili conticini di Matematica, proviamo a risolvere un esercizietto di Fisica in cui la fattorizzazione LU può essere utile.

Si consideri il muro di un'abitazione, rappresentato graficamente in Fig. 3.2. Note le temperature esterna e interna, nonché le caratteristiche del muro, vogliamo calcolare le temperature alle varie interfacce tra materiali diversi. Problema attualissimo, viste tutte le storie sulle certificazioni energetiche che si fanno così spesso ;-)

Visto che questo non è un testo di Fisica ma di Calcolo Numerico, il problema è un po' semplificato. Vi basti sapere che il flusso di calore q è definito come

$$q = \frac{\Delta T}{R}, \tag{3.14}$$

[12] e questo non è per niente scontato, come ci dice John Cook su https://www.johndcook.com/blog/2010/01/19/dont-invert-that-matrix/

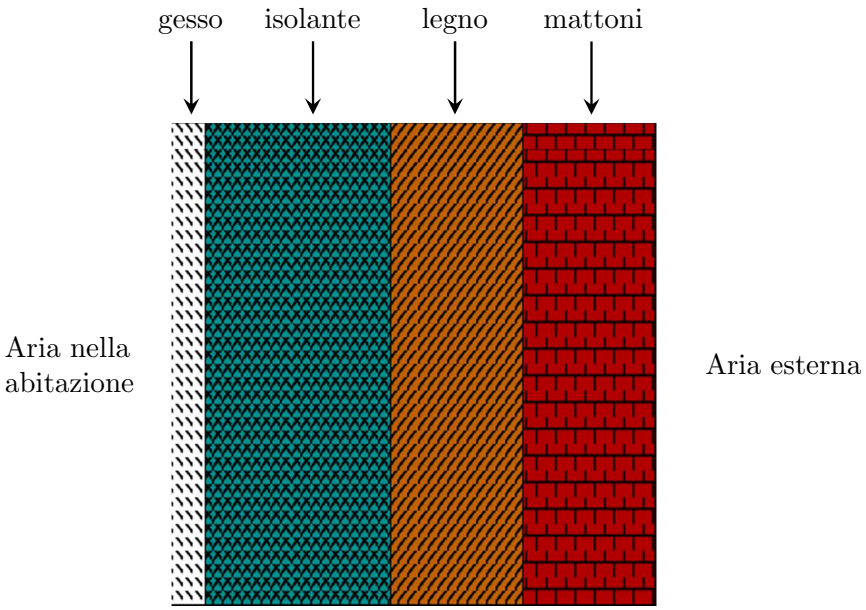

Figura 3.2: Rappresentazione grafica del muro utilizzato come caso di studio.

dove le resistenze termiche sono

$$\underbrace{R_1 = 0.036\,\frac{\mathrm{K}}{\mathrm{W}}}_{\text{gesso}} \quad \underbrace{R_2 = 4.01\,\frac{\mathrm{K}}{\mathrm{W}}}_{\text{isolante}} \quad \underbrace{R_3 = 0.408\,\frac{\mathrm{K}}{\mathrm{W}}}_{\text{legno}} \quad \underbrace{R_4 = 0.038\,\frac{\mathrm{K}}{\mathrm{W}}}_{\text{mattoni}}.$$

L'esercizio si può risolvere ipotizzando che la temperatura interna all'abitazione, T_{int}, e quella esterna, T_{est}, rimangano costanti, e quindi che il flusso termico sia costante[13]. Quindi, imponendo che (3.14) sia costante lungo tutto il muro, otteniamo le relazioni

$$\frac{T_{\text{int}} - T_1}{R_1} = \frac{T_1 - T_2}{R_2} = \frac{T_2 - T_3}{R_3} = \frac{T_3 - T_{\text{est}}}{R_4}.$$

Concentriamoci sulla prima equazione:

$$\frac{T_{\text{int}} - T_1}{R_1} = \frac{T_1 - T_2}{R_2}.$$

Questa si può riscrivere, con un po' di conticini algebrici, come:

$$R_2 T_{\text{int}} - R_2 T_1 = R_1 T_1 - R_1 T_2,$$

in cui l'unico termine completamente noto è quello contenente T_{int}, e quindi, raccogliendo e mettendo in ordine le incognite, possiamo scrivere

$$(R_1 + R_2)T_1 - R_1 T_2 = T_{\text{int}} R_2. \tag{3.15}$$

Procediamo in maniera simile per la seconda eguaglianza:

$$\frac{T_1 - T_2}{R_2} = \frac{T_2 - T_3}{R_3} \Longrightarrow R_3 T_1 - R_3 T_2 = R_2 T_2 - R_2 T_3,$$

che si può riscrivere come

$$R_3 T_1 - (R_2 + R_3)T_2 + R_2 T_3 = 0. \tag{3.16}$$

Infine, per l'ultima eguaglianza, si scrive

[13] per avere più dettagli leggete qualche testo di Fisica Tecnica o Termodinamica

$$\frac{T_2 - T_3}{R_3} = \frac{T_3 - T_{\text{est}}}{R_4} \Longrightarrow R_4 T_2 - R_4 T_3 = R_3 T_3 - R_3 T_{\text{est}},$$

che si può riscrivere come

$$R_4 T_2 - (R_3 + R_4) T_3 = -R_3 T_{\text{est}}. \tag{3.17}$$

A questo punto, le tre equazioni trovate, (3.15), (3.16) e (3.17), costituiscono un sistema lineare:

$$\begin{cases} (R_1 + R_2) T_1 - R_1 T_2 = R_2 T_{\text{int}} \\ R_3 T_1 - (R_2 + R_3) T_2 + R_2 T_3 = 0 \\ R_4 T_2 - (R_3 + R_4) T_3 = -R_3 T_{\text{est}}, \end{cases} \tag{3.18}$$

che si può scrivere anche in forma matriciale:

$$\underbrace{\begin{bmatrix} R_1 + R_2 & -R_1 & 0 \\ R_3 & -(R_2 + R_3) & R_2 \\ 0 & R_4 & -(R_3 + R_4) \end{bmatrix}}_{\underline{\underline{A}}} \underbrace{\begin{bmatrix} T_1 \\ T_2 \\ T_3 \end{bmatrix}}_{\underline{x}} = \underbrace{\begin{bmatrix} R_2 T_{\text{int}} \\ 0 \\ -R_3 T_{\text{est}} \end{bmatrix}}_{\underline{b}}.$$

Questo sistema si può risolvere con facilità mediante MATLAB®. In particolare però diciamo che da bravi certificatori energetici appassionati di Calcolo Numerico decidiamo di simulare l'andamento delle varie temperature del muro per varie stagioni; questo significherebbe risolvere il problema per diversi valori di T_{int} e T_{est}. Dal momento che *il muro è sempre lo stesso*, e quindi le R_i sono considerabili costanti, dal momento che la matrice di sistema $\underline{\underline{A}}$ contiene solamente le R_i, è possibile calcolarne la fattorizzazione LU una volta sola, e quindi risolvere per diversi termini noti, corrispondenti a diverse temperature interne/esterne, dei sistemi non generici, ma sempre triangolari.

Si propone ora uno script MATLAB® implementante la soluzione del problema.

```
clear
close all
clc

Tint = [20  20 20 20]; % vettore delle temperature interne da simulare
Test = [-10 -5  5 10]; % vettore delle temperature esterne da simulare

% Resistenze termiche del muro
R(1) = 0.036; % resistenza termica gesso, K/W
R(2) = 4.01; % resistenza termica isolante, K/W
R(3) = 0.408; % resistenza termica legno, K/W
R(4) = 0.038; % resistenza termica mattoni, K/W

% Matrice di sistema
A = [R(1)+R(2)    -R(1)          0;
     R(3)      -(R(2)+R(3))     R(2);
      0            R(4)    -(R(3)+R(4))];

% Posso calcolare la fattorizzazione LU una tantum
[L,U,P] = lu(A);

for ind=1:length(Tint)
    % Scrivo il termine noto per le temperature di interesse
    b = [Tint(ind)*R(2);0;-Test(ind)*R(3)];
    %
    % Risolvo il sistema esterno, quello che produce il vettore "y":
    c = P*b;
    y = L\c;
    %
    % Risolvo il sistema interno, per ottenere la soluzione
    x = U\y;
    %
    % Salvo per esempio la temperatura alla prima interfaccia
    q(ind) = (Tint(ind)-x(1))/R(1);
end

figure
grid on
hold on
plot(Test,q)
```

Si noti che non è necessario svolgere così tanti passaggi per la soluzione di ciascun sistema; in questo codice, questo è stato fatto solo per questioni pedagogiche. In particolare, il listato

```
c = P*b;
y = L\c;
x = U\y;
```

si può rimpiazzare semplicemente con

```
y = L\(P*b);
x = U\y;
```

o, in modo ancora più compatto, con

```
x = U\(L\(P*b));
```

Queste alternative non sono né meglio né peggio della prima: sono un po' più compatte.

3.5 La fattorizzazione di Choleski

Nella precedente sezione abbiamo discusso ed esplorato in lungo e in largo la fattorizzazione PA = LU. Tra le varie cose che abbiamo visto, abbiamo detto (e non dimostrato) che il numero di operazioni necessarie per il calcolo dei fattori $\underline{\underline{P}}$, $\underline{\underline{L}}$ e $\underline{\underline{U}}$ è $n^3/3$, dove n come al solito è il numero di righe o colonne della matrice di sistema.

Tuttavia, se la matrice $\underline{\underline{A}}$ che intendiamo fattorizzare fosse simmetrica definita positiva, sarebbe possibile fattorizzarla non solo con la PA = LU, ma anche con un metodo diverso, detto **fattorizzazione di Choleski**. Questa fattorizzazione permette di scrivere la matrice $\underline{\underline{A}}$ come

$$\underline{\underline{A}} = \underline{\underline{R}}^{\mathrm{T}} \underline{\underline{R}},$$

ossia in termini del prodotto della trasposta di una certa matrice $\underline{\underline{R}}$, moltiplicata da sinistra a $\underline{\underline{R}}$. In particolare, $\underline{\underline{R}}$ è una matrice triangolare superiore avente elementi positivi sulla diagonale principale. Il motivo per cui questa fattorizzazione è particolarmente interessante è che, oltre a basarsi su un singolo fattore (la matrice $\underline{\underline{R}}$) anziché 3 (le matrici $\underline{\underline{P}}$, $\underline{\underline{L}}$, $\underline{\underline{U}}$), il costo computazionale necessario per calcolarla è $n^3/6$ operazioni anziché $n^3/3$: metà rispetto alla fattorizzazione PA = LU ! MATLAB® permette di calcolare anche questa fattorizzazione per una matrice `A` simmetrica definita positiva, mediante il comando

```
R = chol(A)
```

3.5.1 Applicazioni della fattorizzazione di Choleski

Soluzione di un sistema lineare

Dato un sistema lineare nella solita forma $\underline{\underline{A}}\,\underline{x} = \underline{b}$, dove però $\underline{\underline{A}}$ è una matrice simmetrica definita positiva, è possibile risolverlo utilizzando la fattorizzazione di Choleski. A questo fine, a partire da

$$\underline{\underline{A}}\,\underline{x} = \underline{b},$$

sostituiamo $\underline{\underline{A}}$ con la sua fattorizzazione:

$$\underline{\underline{R}}^{\mathrm{T}} \underbrace{\underline{\underline{R}}\,\underline{x}}_{\underline{y}} = \underline{b},$$

in cui abbiamo implicitamente definito $\underline{y}$ come il prodotto tra la soluzione $\underline{x}$ (che ancora non conosciamo) e $\underline{\underline{R}}$. A questo punto, possiamo riscrivere il sistema come

$$\underline{\underline{R}}^{\mathrm{T}}\,\underline{y} = \underline{b},$$

che si può risolvere mediante il comando \ come

```
y = R'\b; % si noti R', dal momento che bisogna farlo per R trasposta!
```

Questo ci ha permesso di ottenere il vettore $\underline{y}$. Ora, utilizzando la definizione

$$\underline{\underline{R}}\,\underline{x} = \underline{y},$$

si può calcolare la soluzione del sistema finale con il comando

```
x = R\y;
```

Calcolo dell'inversa di una matrice

Dovendo calcolare l'inversa di una matrice $\underline{\underline{A}}$ simmetrica definita positiva, è possibile utilizzare la fattorizzazione di Choleski al posto della LU. Infatti, ricordando che

$$\underline{\underline{A}} = \underline{\underline{R}}^{\mathrm{T}}\underline{\underline{R}},$$

è possibile scrivere

$$\underline{\underline{A}}^{-1} = \left(\underline{\underline{R}}^{\mathrm{T}}\underline{\underline{R}}\right)^{-1} = \underline{\underline{R}}^{-1}(\underline{\underline{R}}^{\mathrm{T}})^{-1} = \underline{\underline{R}}^{-1}\left(\underline{\underline{R}}^{-1}\right)^{\mathrm{T}},$$

in cui abbiamo usato le proprietà delle matrici inverse discusse precedentemente, e scambiato l'operazione di inversione con quella di trasposizione.

Il vantaggio di questo modo di procedere consiste nella possibilità di calcolare semplicemente l'inversa di $\underline{\underline{R}}$ (che è triangolare superiore) invece che dell'intera matrice, trasporre l'inversa, e moltiplicare; in questo modo, il costo del metodo è solo $2n^3/3$, invece di n^3 operazioni.

3.6 La fattorizzazione QR

Fino a questo momento ci siamo concentrati sulla soluzione di sistemi lineari aventi matrici $\underline{\underline{A}}$ quadrate e a determinante non nullo. A questo scopo abbiamo introdotto due fattorizzazioni: la $\underline{\underline{P}}\underline{\underline{A}} = \mathrm{LU}$ e la fattorizzazione di Choleski, avendo sempre in mente il fatto che i sistemi associati hanno soluzione esistente e unica. Da adesso, cambieremo un po' obiettivo: invece di concentrarci su matrici necessariamente quadrate, considereremo la possibilità di dover trattare matrici rettangolari $\underline{\underline{A}} \in \mathbb{R}^{m,n}$, ovvero aventi m righe e n colonne. Ci preoccuperemo prevalentemente del caso $m > n$, ossia matrici con più righe che colonne.

La prima tecnica in grado di fattorizzare una matrice rettangolare che studieremo ha, come proprietà, quella di generare, tra i fattori, una matrice ortogonale. Poiché potrebbe essere opportuno studiare più in dettaglio le matrici ortogonali, Appendice B.1 riporta un po' di spiegazioni e interpretazioni legate a esse.

3.6.1 Introduzione alla fattorizzazione QR e sue proprietà

Il nostro attuale obiettivo è fattorizzare matrici rettangolari. In questo senso ci viene incontro il seguente teorema: ogni matrice $\underline{\underline{A}} \in \mathbb{R}^{m,n}$ è fattorizzabile nella forma

$$\underline{\underline{A}} = \underline{\underline{Q}}\,\underline{\underline{R}}, \tag{3.19}$$

con $\underline{\underline{Q}} \in \mathbb{R}^{m,m}$ ortogonale e $\underline{\underline{R}} \in \mathbb{R}^{m,n}$ avente elementi $r_{ij} = 0$ per $i > j$.

Commentiamo un po' con calma questo teorema. Prima di tutto questo afferma che, data una matrice rettangolare con m righe e n colonne, questa si può **sempre** scrivere come il prodotto di una matrice $\underline{\underline{Q}}$, quadrata, ortogonale, e di una certa matrice $\underline{\underline{R}}$; quest'ultima, non ha nulla a che vedere con la matrice della fattorizzazione di Choleski, anche se vengono sciaguratamente indicate con lo stesso nome. Questa matrice $\underline{\underline{R}}$ è **trapezoidale superiore**, ossia i suoi elementi sono disposti come un trapezio (se la triangolare ha gli elementi disposti come un triangolo, questa è come un trapezio). Per capire di cosa si sta parlando, concentriamoci solo per un momento sul caso *che ci interesserà di meno*, ossia $m < n$, e proviamo a calcolare la fattorizzazione QR di una matrice 3×6 ottenuta prendendo le prime 3 righe della matrice di Hilbert 6×6

```
>> A = hilb(6);
>> A = A(1:3,:)

A =
    1.0000    0.5000    0.3333    0.2500    0.2000    0.1667
    0.5000    0.3333    0.2500    0.2000    0.1667    0.1429
    0.3333    0.2500    0.2000    0.1667    0.1429    0.1250

>> [Q,R] = qr(A)

Q =
   -0.8571    0.5016    0.1170
   -0.4286   -0.5685   -0.7022
   -0.2857   -0.6521    0.7022

R =
   -1.1667   -0.6429   -0.4500   -0.3476   -0.2837   -0.2398
         0   -0.1017   -0.1053   -0.0970   -0.0876   -0.0791
         0         0    0.0039    0.0059    0.0067    0.0070
>>
```

In questo esempietto, viene fuori quello che abbiamo detto finora: data una matrice $\underline{\underline{A}}$ avente $m = 3$ righe, la matrice $\underline{\underline{Q}}$ è proprio 3×3. Potremmo dimostrare, calcolando i prodotti scalari tra le varie colonne, per esempio usando il comando

```
i=1; j=2;
Q(:,i)'*Q(:,j)
```

che questi sono sempre pari a 0 tranne nel caso $i = j$, dimostrando che quindi la matrice $\underline{\underline{Q}}$ è effettivamente ortogonale; oppure, si potrebbe verificare che `Q'*Q` è uguale alla matrice identità. Per quanto riguarda invece la matrice $\underline{\underline{R}}$, ciò che si vede è che sembra un po' una matrice triangolare, dove però abbiamo aggiunto a destra altre colonne, tutte piene; per questo motivo, gli elementi sembrano costituire un trapezio.

Concentriamoci adesso sul caso $m > n$, più interessante per gli scopi di questa sezione, e proviamo a generare una matrice 6×3 ritagliando le prime 3 colonne della matrice di Hilbert 6×6; poi, calcoliamone la fattorizzazione QR:

```
>> A = hilb(6);
>> A = A(:,1:3)

A =
    1.0000    0.5000    0.3333
    0.5000    0.3333    0.2500
    0.3333    0.2500    0.2000
    0.2500    0.2000    0.1667
    0.2000    0.1667    0.1429
    0.1667    0.1429    0.1250
>>
```

Proviamo a calcolare il determinante della sottomatrice 3×3 più alta:

```
>> det(A(1:3,1:3))

ans =
   4.6296e-04
>>
```

e vediamo che questo è diverso da zero; non è grandissimo, ma comunque molto meglio di 10^{-12} o cose del genere! Possiamo quindi dire che $\underline{\underline{A}}$ sia una matrice **di rango massimo**! Tra poco sarà chiaro il motivo di questa verifica; prima, guardiamo in faccia la fattorizzazione QR dell'intera matrice $\underline{\underline{A}}$ ottenuta come appena descritto:

```
>> [Q,R]=qr(A)

Q =
   -0.8189    0.5397   -0.1893    0.0051   -0.0235   -0.0428
   -0.4094   -0.3320    0.7024    0.0204    0.2631    0.3989
   -0.2730   -0.4219    0.1529   -0.4124   -0.5063   -0.5456
   -0.2047   -0.4067   -0.2015    0.8430   -0.1486   -0.1388
   -0.1638   -0.3735   -0.3963   -0.2262    0.7463   -0.2617
   -0.1365   -0.3399   -0.4998   -0.2600   -0.3081    0.6734

R =
   -1.2212   -0.7019   -0.5045
         0   -0.1385   -0.1511
         0         0   -0.0096
         0         0         0
         0         0         0
         0         0         0
>>
```

Sulla matrice $\underline{\underline{Q}}$ non c'è molto da dire che non sia già stato detto: è una matrice $m \times m$ ortogonale. Per quanto riguarda la matrice $\underline{\underline{R}}$, ci sono alcune osservazioni da fare: dalla $(n+1)$-esima riga in poi, tutti gli elementi sono nulli; inoltre, nelle prime n righe, la matrice, più che trapezoidale, sembra essere triangolare.

Ipotizziamo adesso di fare una variante a questa prova: definiamo sempre la matrice $\underline{\underline{A}}$ a partire da quella di Hilbert, ma poi facciamo in modo che le righe dalla terza alla sesta siano imposte uguale alla prima; in questo modo, la matrice $\underline{\underline{A}}$ sarà chiaramente di rango non massimo, poiché solo le prime due righe saranno linearmente indipendenti (e la matrice è 6×3). Calcoliamo anche per questa la fattorizzazione QR:

```
>> A = hilb(6);
>> A = A(:,1:3);
>> A(3,:)=A(1,:);
>> A(4,:)=A(1,:);
>> A(5,:)=A(1,:);
>> A(6,:)=A(1,:);
>> A % faccio scrivere su schermo la matrice A ora definita

A =
    1.0000    0.5000    0.3333
    0.5000    0.3333    0.2500
    1.0000    0.5000    0.3333
    1.0000    0.5000    0.3333
    1.0000    0.5000    0.3333
    1.0000    0.5000    0.3333

>> [Q,R] = qr(A)

Q =
   -0.4364    0.0976    0.8944         0         0         0
   -0.2182   -0.9759    0.0000    0.0000    0.0000    0.0000
   -0.4364    0.0976   -0.2236   -0.5000   -0.5000   -0.5000
   -0.4364    0.0976   -0.2236    0.8333   -0.1667   -0.1667
   -0.4364    0.0976   -0.2236   -0.1667    0.8333   -0.1667
   -0.4364    0.0976   -0.2236   -0.1667   -0.1667    0.8333


R =
   -2.2913   -1.1638   -0.7819
         0   -0.0813   -0.0813
```

```
         0         0    0.0000
         0         0         0
         0         0         0
         0         0         0
>>
```

Vediamo che, in questo caso, la matrice $\underline{\underline{R}}$ è trapezoidale, dal momento che anche la terza riga contiene tutti zeri (l'ultimo elemento ha un po' di sporcizia numerica, ma si potrebbe vedere che è estremamente piccolo rispetto agli altri elementi):

```
>> format short e
>> R

R =
  -2.2913e+00  -1.1638e+00  -7.8195e-01
             0  -8.1325e-02  -8.1325e-02
             0             0   5.7077e-17
             0             0            0
             0             0            0
             0             0            0
>>
```

Prima di trarre le conclusioni, una comunicazione: la fattorizzazione QR di una matrice non è unica; in altre parole, è possibile che esistano più coppie di matrici $\underline{\underline{Q}}$ e $\underline{\underline{R}}$ tali per cui $\underline{\underline{Q}}$ sia ortogonale, $\underline{\underline{R}}$ sia trapezoidale, e il loro prodotto coincida con la matrice $\underline{\underline{A}}$ di partenza. Il motivo per cui abbiamo proposto gli esperimenti numerici di prima, generando una matrice a rango massimo (tale per cui il determinante di almeno una delle sottomatrici 3×3 è diverso da 0) e una a rango non massimo (tale per cui il determinante di ogni sottomatrice 3×3 è uguale a 0), è il seguente teorema. Data $\underline{\underline{A}} \in \mathbb{R}^{m,n}$ **di rango massimo**, avente più righe che colonne (ossia $m > n$), allora il fattore $\underline{\underline{R}}$ della fattorizzazione QR si può scrivere nella forma

$$\underline{\underline{R}} = \begin{bmatrix} \widetilde{\underline{\underline{R}}} \\ \underline{\underline{0}} \end{bmatrix}, \tag{3.20}$$

dove $\widetilde{\underline{\underline{R}}}$ è una matrice triangolare superiore non singolare. Inoltre, in queste condizioni, la matrice $\underline{\underline{A}}$ può essere scritta nella forma

$$\underline{\underline{A}} = \widetilde{\underline{\underline{Q}}}\,\widetilde{\underline{\underline{R}}}, \tag{3.21}$$

dove $\widetilde{\underline{\underline{Q}}} \in \mathbb{R}^{m,n}$ è una matrice con m righe e n colonne avente i vettori colonna tra loro ortogonali. Infine, questo teorema garantisce che, pur non essendo unici nemmeno i fattori $\widetilde{\underline{\underline{Q}}}$ e $\widetilde{\underline{\underline{R}}}$, tra tutti questi, esiste un'unica coppia $\widetilde{\underline{\underline{Q}}}^{(p)}$ e $\widetilde{\underline{\underline{R}}}^{(p)}$ tali per cui $\widetilde{\underline{\underline{R}}}^{(p)}$ abbia gli elementi diagonali positivi[14].

Il comando `qr` di MATLAB® permette di calcolare la fattorizzazione QR nelle due forme appena presentate. In particolare, il comando

```
[Q,R] = qr(A);
```

calcola i fattori `Q` avente dimensione $m \times m$ e `R` avente dimensione $m \times n$. Tuttavia, esiste un metodo alternativo di utilizzare il comando `qr`:

```
[Qtilda,Rtilda] = qr(A,0);
```

Qui, i nomi delle variabili `Qtilda` e `Rtilda` sono stati da me scelti arbitrariamente (richiamando la tilda presente sulle matrici *rimpicciolite* di (3.21)), al fine di evidenziare che si tratta di matrici diverse da quelle immagazzinate in `Q` e `R`. Utilizzando il comando `qr(A,0)`, infatti, si chiede a MATLAB® di calcolare la cosiddetta **fattorizzazione QR economica**, ossia quella indicata in (3.21). Si tenga presente che MATLAB® , anche nel caso in cui il rango della matrice $\underline{\underline{A}}$ è massimo, **non** calcola la *unica fattorizzazione QR economica avente elementi diagonali di* $\widetilde{\underline{\underline{R}}}$ *positivi*: calcola una delle varie possibilità non meglio definite.

[14] io vedo il fatto di avere gli elementi della diagonale di $\widetilde{\underline{\underline{R}}}$ positivi come una sorta di condizione aggiuntiva atta a rendere univoca, nel caso ci servisse per qualche motivo, la fattorizzazione QR di una matrice

3.6.2 Soluzione di sistemi lineari determinati

Facciamo un passo indietro, e parliamo ancora per un momento di sistemi lineari determinati, ossia la cui matrice di sistema $\underline{\underline{A}}$ è quadrata ($\underline{\underline{A}} \in \mathbb{R}^{n,n}$) e a determinante non nullo. Proprio come per le fattorizzazioni PA=LU e di Choleski, anche la fattorizzazione QR può essere utilizzata per risolvere un sistema lineare determinato, ossia quello avente un'unica soluzione. Partiamo dalla ormai familiare equazione matriciale

$$\underline{\underline{A}}\,\underline{x} = \underline{b},$$

dove però possiamo dire che $\underline{\underline{A}} = \underline{\underline{Q}}\,\underline{\underline{R}}$. Sostituendo, si ottiene

$$\underline{\underline{Q}}\,\underline{\underline{R}}\,\underline{x} = \underline{b}.$$

A questo punto, possiamo ricordare che, essendo $\underline{\underline{Q}}$ una matrice ortogonale, si ha

$$\underline{\underline{Q}}^{\mathrm{T}}\,\underline{\underline{Q}} = \underline{\underline{I}},$$

e questo ci suggerisce che, per far *sparire una matrice* e semplificare così i conti, potremmo semplicemente moltiplicare da sinistra entrambi i membri per $\underline{\underline{Q}}^{\mathrm{T}}$, ottenendo:

$$\underline{\underline{Q}}^{\mathrm{T}}\,\underline{\underline{Q}}\,\underline{\underline{R}}\,\underline{x} = \underline{\underline{Q}}^{\mathrm{T}}\,\underline{b},$$

che si semplifica diventando

$$\underline{\underline{R}}\,\underline{x} = \underline{\underline{Q}}^{\mathrm{T}}\,\underline{b}.$$

In effetti, questa procedura permette di non dover risolvere due sistemi lineari triangolari, ma solamente uno. Tuttavia, il costo della fattorizzazione QR è molto più alto rispetto alle fattorizzazioni LU o di Choleski; non lo riportiamo esplicitamente, perché esistono diverse tecniche che permettono di calcolare i fattori $\underline{\underline{Q}}$ e $\underline{\underline{R}}$, e non è scopo di queste note spiegarli. Basti dunque sapere che non conviene ricorrere alla fattorizzazione QR per la soluzione di un sistema lineare determinato, che è un problema che si può risolvere per le altre vie già presentate.

3.6.3 Soluzione di sistemi lineari sovradeterminati

Cos'è un sistema sovradeterminato?

Una delle applicazioni più interessanti in cui la fattorizzazione QR gioca invece un ruolo fondamentale è la soluzione di un sistema lineare **sovradeterminato**, ossia che presenta più equazioni che incognite. Si tratta di sistemi lineari in cui la matrice di sistema è rettangolare e avente più righe che colonne, ovvero il caso $m > n$ che abbiamo prima definito come *nostro prediletto*; ora è chiaro perché! ;-)

Quando si parla di problemi di questo tipo, può capitare che si incontrino i seguenti termini:

- può capitare di sentir parlare di *vincoli*; in questo contesto, *vincoli* è un sinonimo di *equazioni del sistema*: infatti, immaginando per esempio di avere un'equazione del tipo

  $$2x_1 + 3x_2 + 4x_3 = 1,$$

  questa equazione *lega*, ossia *vincola*, x_1, x_2, x_3, a dover soddisfare questa relazione: *la somma del doppio di x_1, del triplo di x_2 e del quadruplo di x_3 è uguale a 1*: è un vincolo che lega le tre variabili!

- in contrapposizione ai vincoli si hanno i *gradi di libertà*, ossia le *incognite* del problema; queste si chiamano *gradi di libertà* perché al momento della soluzione del problema noi siamo liberi di *farle variare* fino a trovare quelle che soddisfano i vincoli.

Questi sinonimi sono molto utili perché ci permettono di poter comprendere intuitivamente alcuni punti del teorema di Rouché-Capelli; ammesso di aver a che fare con matrici di rango massimo, possiamo dire le seguenti frasi.

- In un sistema lineare determinato, si hanno n vincoli e n gradi di libertà; in questo modo, *ciascuno dei gradi di libertà può essere sfruttato per soddisfare un vincolo*.

- In un sistema lineare sottodeterminato si hanno m vincoli e $n > m$ gradi di libertà; sarebbe la situazione in cui si hanno più incognite che equazioni. Sistemi di questo tipo possono avere infinite soluzioni, perché m dei gradi di libertà possono essere usati per soddisfare i corrispondenti m vincoli, ma i restanti $n - m$ non hanno alcun vincolo da soddisfare e, quindi, possono *restare liberi*; per questo motivo, a seconda dei valori che usiamo, abbiamo diverse soluzioni, tutte valide!
- In un sistema sovradeterminato, si hanno m vincoli (assumendo che almeno $n + 1$ non siano ridondanti) e $n < m$ gradi di libertà; in altre parole, abbiamo più equazioni (di cui almeno $n + 1$ linearmente indipendenti) che incognite. Questo, in altre parole, significa che non tutti i vincoli potranno essere pienamente soddisfatti: alcuni lo saranno di più, alcuni lo saranno di meno, ma, poiché le diverse equazioni potranno dare luogo a *vincoli* tra loro incompatibili, non sarà possibile soddisfarli tutti con così pochi gradi di libertà!

Cercando di rendere meglio l'idea, proviamo a cercare di capire cosa comporta avere a che fare con sistemi sovradeterminati su un esempio molto semplice:

$$\begin{cases} 4x + 3y = 7 \\ 2x + 3y = 5 \\ 3x + 4y = 4. \end{cases}$$

Immaginiamo per un attimo di concentrarci solo sulle prime due equazioni, e di non guardare la terza; questo sarebbe uno di quei sistemi in cui il termine noto è costruito con la somma dei coefficienti moltiplicativi dei monomi a membro sinistro, quindi la soluzione sarebbe $x = 1$, $y = 1$. Tuttavia, immaginiamoci a questo punto di inserire questa soluzione nella terza equazione; troveremmo che

$$3x + 4y = 3 \times 1 + 4 \times 1 = 7 \neq 4,$$

quindi, disporre solo di due gradi di libertà, x e y, non permette di trovare una configurazione in grado di soddisfare tutti i tre vincoli. :-(

Il problema dei minimi quadrati

Si consideri un sistema sovradimensionato, sempre nella forma

$$\underline{\underline{A}}\,\underline{x} = \underline{b},$$

dove però, da qui in poi, $\underline{\underline{A}} \in \mathbb{R}^{m,n}$, $\underline{x} \in \mathbb{R}^n$, e $\underline{b} \in \mathbb{R}^m$, e $m > n$. Dall'esempio appena svolto, abbiamo capito che alcuni vincoli, alcune equazioni, potrebbero essere incompatibili con altri, e quindi, per soddisfare un'equazione, un'altra sarà insoddisfatta.

«Va beh ma ci sarà un qualche modo di risolvere sistemi di questo tipo, o no?»

In effetti sì, però dobbiamo cambiare notevolmente il nostro punto di vista: tutto ciò che abbiamo sempre detto sui sistemi determinati e su che senso abbia risolverli, cambierà. In particolare, dobbiamo capire che il vettore delle incognite $\underline{x}$ che *risolve* un sistema sovradimensionato ha un significato un po' diverso rispetto a ciò a cui siamo abituati, con sistemi *determinati*. Per capirlo, può essere opportuno renderci conto di alcune implicazioni. Dire che l'equazione matriciale associata a un sistema lineare è soddisfatta significa, portando a membro sinistro il termine noto $\underline{b}$, che

$$\underline{\underline{A}}\,\underline{x} - \underline{b} = \underline{0},$$

dove $\underline{0}$ è il vettore contenente tutti zeri. A questo punto, proviamo a calcolare la norma di ambo i membri (per ora non specifichiamo se 1, 2, o ∞); avremo:

$$\left|\left|\underline{\underline{A}}\,\underline{x} - \underline{b}\right|\right| = ||\underline{0}|| = 0.$$

Infatti, una delle proprietà di una norma è che essa è nulla **se e solo se** il vettore del quale la stiamo calcolando è uguale al vettore nullo. In altre parole, possiamo dire che la norma $\left|\left|\underline{\underline{A}}\,\underline{x} - \underline{b}\right|\right|$ è nulla se e solo se $\underline{x}$ è soluzione di un sistema nel *senso classico*, ossia se tutti i vincoli sono soddisfatti esattamente. Evidentemente, nel caso di un sistema sovradeterminato, non sarà possibile trovare un $\underline{x}$ tale per cui la norma in questione possa essere nulla. Però -e questo è il momento in cui dobbiamo cercare di cambiare

il nostro punto di vista- possiamo cercare di **minimizzarla**, ossia trovare il vettore $\underline{x}$ tale per cui questa norma è **la più piccola possibile**. Questa procedura si può anche vedere in questo modo: immaginiamo di prendere tanti vettori $\underline{y} \in \mathbb{R}^n$, e provare, per ciascuno di essi, a calcolare $\left\|\underline{\underline{A}}\,\underline{y} - \underline{b}\right\|$. A questo punto, possiamo definire $\underline{x}$ come quel vettore $\underline{y}$ tale per cui

$$\left\|\underline{\underline{A}}\,\underline{x} - \underline{b}\right\| = \min_{\underline{y}\in\mathbb{R}^n} \left\|\underline{\underline{A}}\,\underline{y} - \underline{b}\right\|.$$

Se non possiamo fare sì che questa norma sia zero, possiamo cercare di trovare la minima: **questo è il senso di risolvere un sistema sovradeterminato**!

D'ora in avanti, concentriamoci non su una generica norma, bensì sulla norma 2 o *norma euclidea*. In questa condizione, la ricerca del vettore $\underline{x}$ tale per cui

$$\left\|\underline{\underline{A}}\,\underline{x} - \underline{b}\right\|_2 = \min_{\underline{y}\in\mathbb{R}^n} \left\|\underline{\underline{A}}\,\underline{y} - \underline{b}\right\|_2 \tag{3.22}$$

è detta **problema dei minimi quadrati**, e $\underline{x}$ ne è dunque la soluzione.

Un primo risultato riguardante il problema dei minimi quadrati è espresso nel seguente teorema, che commenteremo punto-punto.

1. Data $\underline{\underline{A}} \in \mathbb{R}^{m,n}$, con $m \geq n$, sia $\mathcal{X}$ l'insieme dei vettori soluzione $\underline{x} \in \mathbb{R}^n$ del problema dei minimi quadrati; allora, $\mathcal{X}$ è non vuoto.

   «Prima di tutto, $\mathcal{X}$ è l'insieme delle soluzioni del problema dei minimi quadrati $\underline{\underline{A}}\,\underline{x} = \underline{b}$, ossia l'insieme dei vettori $\underline{x}$ che minimizzano la norma $\left\|\underline{\underline{A}}\,\underline{x} - \underline{b}\right\|_2$. Parlo di insieme perché per ora nessuno ha detto che ce ne sia solo una, di soluzione. Possiamo quindi dire che questo insieme $\mathcal{X} \subset \mathbb{R}^n$, ossia che sia un sottoinsieme di $\mathbb{R}^n$; ovvio: non è che tutti i vettori a n componenti reali siano soluzioni del problema dei minimi quadrati ;-). Questo pezzo di teorema afferma che $\mathcal{X}$ è non vuoto, ossia, in altre parole, che **esiste sempre soluzione al problema dei minimi quadrati**. Questo è, tutto sommato, ragionevole: dal momento che la soluzione del problema dei minimi quadrati non è una condizione così peculiare, ma genericamente *richiede di minimizzare una certa norma*, allora è ragionevole pensare che esista sempre *almeno* una situazione in cui questa norma è minima!»

2. Si ha che $\underline{x} \in \mathcal{X}$ per il sistema $\underline{\underline{A}}\,\underline{x} = \underline{b}$ se e solo se

   $$\underline{\underline{A}}^{\mathrm{T}}\underline{\underline{A}}\,\underline{x} = \underline{\underline{A}}^{\mathrm{T}}\,\underline{b} \tag{3.23}$$

   è esattamente soddisfatta. Tale sistema viene detto **sistema delle equazioni normali** o **sistema normale**.

   «Cerchiamo un po' di capire una cosa. La matrice $\underline{\underline{A}}$ ha m righe e n colonne, e stiamo ipotizzando $m > n$, ossia che $\underline{\underline{A}}$ è *più alta che larga*. Però, $\underline{\underline{A}}^{\mathrm{T}}$ avrà quindi n righe e m colonne. Se moltiplichiamo da sinistra $\underline{\underline{A}}^{\mathrm{T}}$ a $\underline{\underline{A}}$, il risultato finale avrà n righe e n colonne. Da un certo punto di vista, è come se prendessimo $\underline{\underline{A}}$ e, in qualche senso, la *comprimessimo*, come se ne *buttassimo via alcune righe*. In questo senso, questo pezzo di teorema afferma che se *buttiamo via* un po' di informazioni, $\underline{x}$ può essere interpretata come soluzione *in senso classico* di un qualche sistema lineare: quello con matrice di sistema $\underline{\underline{A}}^{\mathrm{T}}\underline{\underline{A}}$. Almeno, possiamo ricondurci a concetti, a modi di definire la soluzione, che ci sono più familiari ;-)»

3. L'insieme $\mathcal{X}$ si riduce a un solo elemento $\underline{x}^*$ se e solo se la matrice $\underline{\underline{A}}$ ha rango massimo.

   «Nei punti precedenti del teorema abbiamo fornito alcuni risultati di esistenza, e alcune proprietà, delle nostre soluzioni; nessuno ha detto però nulla sul fatto che queste soluzioni possano essere uniche. In generale, la soluzione del problema dei minimi quadrati non è unica. Beh tutto sommato non è così irragionevole: se il nostro obiettivo è fare sì che la norma $\left\|\underline{\underline{A}}\,\underline{y} - \underline{b}\right\|_2$ sia minima, beh, chi ce lo dice che non ci possano essere due

vettori, $\underline{x}_1$ e $\underline{x}_2$, che permettano di ottenere questa condizione? E chi ci dice che non ce ne possano essere anche più di due? Beh, questo teorema ci fornisce un interessante dettaglio: se la matrice $\underline{\underline{A}}$ ha rango massimo, allora l'insieme delle soluzioni, $\mathcal{X}$, si riduce a un solo elemento $\underline{x}^*$; in altre parole, se la matrice ha rango massimo, la soluzione è **unica**!»

4. Esiste uno e un solo vettore $\underline{x} \in \mathcal{X}$ tale che

$$\|\underline{x}^*\|_2 = \min_{\underline{x} \in \mathcal{X}} \|\underline{x}\|_2 ,$$

e questo vettore $\underline{x}^*$ viene detto **soluzione di minima norma**.

«Però non è che siamo sicurissimi che la soluzione sia unica: mettiamo che $\underline{\underline{A}}$ non abbia rango massimo. Tuttavia, una cosa possiamo dirla: di tutte le possibili soluzioni del problema dei minimi quadrati, dunque $\underline{x} \in \mathcal{X}$, ne esiste solo una, $\underline{x}^*$, che ha norma minima. Anche questo ha senso: mettiamo che ci siano tante soluzioni, ossia tanti $\underline{x}$ che permettono di minimizzare la norma $\left\|\underline{\underline{A}}\,\underline{x} - \underline{b}\right\|_2$. Di tutti questi $\underline{x}$, però, non è così impensabile che solo uno possa avere norma minima; non sapendo dunque quale soluzione scegliere, nel caso ne avessimo molte, potremmo decidere di optare per essa: $\underline{x}^*$! Ovviamente, nel caso la soluzione sia invece unica, come al punto precedente, è ovvio che abbia anche norma minima: è l'unica! ;-)»

Soluzione del problema dei minimi quadrati: caso di matrici a rango massimo

Ipotizziamo di dover risolvere un problema dei minimi quadrati

$$\underline{\underline{A}}\,\underline{x} = \underline{b},$$

in cui $\underline{\underline{A}} \in \mathbb{R}^{m,n}$ è una matrice avente rango massimo. Nella precedente sezione abbiamo enunciato un teorema che ci ha insegnato che, in questa situazione, la soluzione del problema dei minimi quadrati è unica: quella che abbiamo indicato con $\underline{x}^*$. Inoltre, poiché abbiamo imparato che una soluzione del problema dei minimi quadrati, per essere davvero tale, deve essere la soluzione del sistema normale

$$\underline{\underline{A}}^{\mathrm{T}}\underline{\underline{A}}\,\underline{x} = \underline{\underline{A}}^{\mathrm{T}}\underline{b},$$

noi potremmo: ricordarci che la matrice $\underline{\underline{A}}^{\mathrm{T}}\underline{\underline{A}}$ è simmetrica definita positiva, applicare la fattorizzazione di Choleski, e trovare la soluzione del problema dei minimi quadrati come soluzione **esatta**, ossia **in senso classico**, del sistema appena scritto. Tuttavia, è risaputo che una procedura del genere sarebbe numericamente molto instabile, dal momento che il sistema delle equazioni normali tende a essere molto mal condizionato.

Di conseguenza, ora studieremo il metodo **corretto** per risolvere il problema dei minimi quadrati, che si bassa sulla fattorizzazione QR. A questo fine, studiamo prima una proprietà delle matrici ortogonali e della norma 2 di un vettore. Come noto dalle sezioni precedenti, il quadrato della norma 2 di un vettore si può scrivere come il prodotto righe per colonne

$$\|\underline{x}\|_2^2 = \underline{x}^{\mathrm{T}}\underline{x}. \tag{3.24}$$

Si consideri a questo punto la norma

$$\left\|\underline{\underline{Q}}^{\mathrm{T}}\underline{z}\right\|_2^2,$$

dove $\underline{z}$ è un vettore colonna e $\underline{\underline{Q}}$ è una matrice ortogonale. Allora, possiamo scrivere questa norma come prodotto righe per colonne dei seguenti vettori:

$$\left\|\underline{\underline{Q}}^{\mathrm{T}}\underline{z}\right\|_2^2 = \left(\underline{\underline{Q}}^{\mathrm{T}}\underline{z}\right)^{\mathrm{T}}\left(\underline{\underline{Q}}^{\mathrm{T}}\underline{z}\right) = \underline{z}^{\mathrm{T}}\underline{\underline{Q}}\,\underline{\underline{Q}}^{\mathrm{T}}\underline{z},$$

tuttavia, come noto, essendo $\underline{\underline{Q}}$ ortogonale, il prodotto $\underline{\underline{Q}}\,\underline{\underline{Q}}^{\mathrm{T}}$ è pari all'identità, dunque

$$\left\|\underline{\underline{Q}}^{\mathrm{T}}\underline{z}\right\|_2^2 = \underline{z}^{\mathrm{T}}\,\underline{\underline{I}}\,\underline{z} = \underline{z}^{\mathrm{T}}\underline{z} = \|\underline{z}\|_2^2. \tag{3.25}$$

Detto a parole, la norma di un vettore moltiplicato per una matrice ortogonale è uguale alla norma del vettore stesso, e questo **sempre**. Potremmo anche dire che le applicazioni lineari associate a matrici ortogonali appartengono alla classe delle *isometrie*; se infatti il concetto di norma viene utilizzato per definire una sorta di *lunghezza del vettore*, se moltiplicare un vettore per una matrice ortogonale non ne cambia la lunghezza, allora è sensato dire che questa sia un'isometria, una applicazione che non cambia le lunghezze. In effetti, l'Appendice B.1 utilizza come esempio di matrice ortogonale quella che rappresenta l'operazione di rotazione su un piano.

Ora che disponiamo di questa fantastica proprietà, cerchiamo di utilizzarla in qualche modo. Come abbiamo già detto in precedenza, il problema dei minimi quadrati è trovare il $\underline{x}$ definito in (3.22). Partiamo quindi dal membro destro, ossia dalla quantità che vogliamo minimizzare, ed eleviamola al quadrato: se troviamo un $\underline{y}$ tale per cui la norma è minima, non vedo perché il quadrato della norma non debba essere più minimo:

$$\left\|\underline{\underline{A}}\,\underline{y} - \underline{b}\right\|_2^2$$

Il motivo dietro a questa elevazione al quadrato è il nostro desiderio di sfruttare la proprietà appena introdotta. Usiamola però *al contrario*, ossia, usiamola per far *apparire* dentro al segno di norma la nostra matrice ortogonale $\underline{\underline{Q}}^{\mathrm{T}}$; come potete immaginare, di tutte le matrici ortogonali, scegliamo proprio la $\underline{\underline{Q}}$ della fattorizzazione QR ;-)

$$\left\|\underline{\underline{A}}\,\underline{y} - \underline{b}\right\|_2^2 = \left\|\underline{\underline{Q}}^{\mathrm{T}}\left(\underline{\underline{A}}\,\underline{y} - \underline{b}\right)\right\|_2^2 = \left\|\underline{\underline{Q}}^{\mathrm{T}}\underline{\underline{A}}\,\underline{y} - \underbrace{\underline{\underline{Q}}^{\mathrm{T}}\underline{b}}_{\underline{c}}\right\|_2^2,$$

dove abbiamo implicitamente definito $\underline{c}$ come

$$\underline{c} = \underline{\underline{Q}}^{\mathrm{T}}\underline{b}.$$

Ricordiamoci a questo punto della definizione di fattorizzazione QR (3.19); se

$$\underline{\underline{A}} = \underline{\underline{Q}}\,\underline{\underline{R}},$$

possiamo moltiplicare da sinistra ambo i membri per $\underline{\underline{Q}}^{\mathrm{T}}$, ottenendo

$$\underline{\underline{Q}}^{\mathrm{T}}\underline{\underline{A}} = \underline{\underline{Q}}^{\mathrm{T}}\underline{\underline{Q}}\,\underline{\underline{R}},$$

ma $\underline{\underline{Q}}$ è ortogonale, e quindi il prodotto a membro destro si riduce all'identità, così che abbiamo appena dimostrato che

$$\underline{\underline{Q}}^{\mathrm{T}}\underline{\underline{A}} = \underline{\underline{R}},$$

e possiamo sostituirlo dentro al nostro segno di norma, ottenendo

$$\left\|\underline{\underline{A}}\,\underline{y} - \underline{b}\right\|_2^2 = \left\|\underline{\underline{R}}\,\underline{y} - \underline{c}\right\|_2^2.$$

Ovviamente, visto che abbiamo scelto come matrice $\underline{\underline{Q}}$ da far apparire dentro la norma *proprio quella della fattorizzazione QR*, vien da sé che anche $\underline{\underline{R}}$ *è proprio quella della fattorizzazione QR*. Questa sezione, tuttavia, si concentra sul caso di matrice $\underline{\underline{A}}$ avente rango massimo e, quindi, valgono le ipotesi che ci permettono di scrivere $\underline{\underline{R}}$ secondo (3.20):

$$\underline{\underline{R}} = \begin{bmatrix} \underline{\underline{\widetilde{R}}} \\ \underline{\underline{\widetilde{0}}} \end{bmatrix},$$

dove $\underline{\underline{\widetilde{R}}} \in \mathbb{R}^{n,n}$ è una matrice triangolare superiore non singolare, e $\underline{\underline{\widetilde{0}}} \in \mathbb{R}^{m-n,n}$ è una matrice rettangolare contenente tutti zeri. Poiché dentro il segno di norma si sta calcolando la differenza di $\underline{\underline{R}}\,\underline{y}$ e $\underline{c}$, dal momento che tratteremo in modo diverso il blocco superiore di $\underline{\underline{R}}$, associato al sistema triangolare con matrice $\underline{\underline{\widetilde{R}}}$, e il blocco inferiore, ha senso considerare il vettore $\underline{c} = \underline{\underline{Q}}^{\mathrm{T}}\underline{b}$ come diviso in due parti analoghe:

$$\underline{c} = \begin{bmatrix} \underline{c}_1 \\ \underline{c}_2 \end{bmatrix},$$

dove $\underline{c}_1 \in \mathbb{R}^n$ è il vettore contenente le prime n componenti (n proprio come la dimensione di $\underline{\underline{\widetilde{R}}}$), mentre $\underline{c}_2 \in \mathbb{R}^{m-n}$ contiene le restanti $m-n$ componenti). Allora, possiamo riscrivere la norma come:

$$\left\| \underline{\underline{R}}\,\underline{y} - \underline{c} \right\|_2^2 = \left\| \begin{bmatrix} \underline{\underline{\widetilde{R}}}\,\underline{y} \\ \underline{0} \end{bmatrix} - \begin{bmatrix} \underline{c}_1 \\ \underline{c}_2 \end{bmatrix} \right\|_2^2,$$

infatti ricordiamo che $\underline{y} \in \mathbb{R}^n$; dal momento che viene moltiplicato per $\underline{\underline{\widetilde{0}}}$, nella parte inferiore esso non conta nulla, e si ottiene un vettore $\underline{0}$ avente $m-n$ componenti, tutte nulle. Infine, possiamo ancora svolgere un piccolo passaggio, ottenendo:

$$\left\| \underline{\underline{A}}\,\underline{y} - \underline{b} \right\|_2^2 = \left\| \underline{\underline{R}}\,\underline{y} - \underline{c} \right\|_2^2 = \left\| \begin{bmatrix} \underline{\underline{\widetilde{R}}}\,\underline{y} - \underline{c}_1 \\ \underline{c}_2 \end{bmatrix} \right\|_2^2. \tag{3.26}$$

Qui avremmo dovuto mettere $-c_2$ al denominatore, ma, essendo sotto il segno di norma ed essendo l'unico termine (sopra essendoci una differenza, dovremmo o cambiare entrambi i segni o tenere così), possiamo buttare via il segno $-$. Questa espressione è il punto di arrivo, che ci permetterà di trarre delle conclusioni. Infatti, il nostro obiettivo iniziale era ottenere $\underline{x}^*$ secondo la seguente espressione:

$$\left\| \underline{\underline{A}}\,\underline{x} - \underline{b} \right\|_2 = \min_{\underline{y} \in \mathbb{R}^n} \left\| \underline{\underline{A}}\,\underline{y} - \underline{b} \right\|_2.$$

Tuttavia, le manipolazioni che abbiamo fatto al membro destro ci permettono di capire con grande semplicità qual è il $\underline{y}$ che stiamo cercando: è semplicemente la soluzione del sistema lineare

$$\underline{\underline{\widetilde{R}}}\,\underline{x}^* = \underline{c}_1.$$

Infatti, $\underline{\underline{\widetilde{R}}}$ è una matrice $n \times n$ triangolare superiore e non singolare, grazie all'ipotesi di $\underline{\underline{A}}$ a rango pieno, e quindi il sistema appena scritto ammette soluzione **esatta**; di conseguenza, scegliendo $\underline{y} = \underline{x}^*$, abbiamo

$$\min_{\underline{y} \in \mathbb{R}^n} \left\| \underline{\underline{A}}\,\underline{y} - \underline{b} \right\|_2 = \left\| \underline{\underline{A}}\,\underline{x}^* - \underline{b} \right\|_2 = \left\| \begin{bmatrix} \underline{0} \\ \underline{c}_2 \end{bmatrix} \right\|_2 = \|\underline{c}_2\|_2$$

e, poiché questo procedimento ci ha permesso di sostituire un $\underline{0}$ nel blocco superiore dell'espressione in (3.26), possiamo sicuramente dire che la soluzione che cercavamo, ovvero la soluzione nel senso dei minimi quadrati, è certamente coincidente con la soluzione del sistema lineare determinato $\underline{\underline{\widetilde{R}}}\,\underline{x}^* = \underline{c}_1$.

Cerchiamo di riassumere cosa abbiamo fatto. Se consideriamo (3.26), ricordando che tutti i nostri gradi di libertà sono in $\underline{y}$, noi possiamo cercare di minimizzare solamente il termine superiore, ma possiamo cambiare $\underline{y}$ quanto vogliamo, e il termine inferiore, quello contenente $\underline{c}_2$, non cambierà di nulla. Questo modo di riscrivere, grazie alla fattorizzazione QR, l'argomento della norma $\left\| \underline{\underline{A}}\,\underline{y} - \underline{b} \right\|_2$, ha permesso di decomporlo in due blocchi: uno *modificabile* dai gradi di libertà, l'altro no. In un sistema lineare determinato, tutti i vincoli sono raggiungibili dai gradi di libertà, ma questo è un sistema sovradeterminato, e quindi la soluzione sarà corretta a meno di un certo residuo. Questo residuo, abbiamo capito, è pari a

$$\min_{\underline{y} \in \mathbb{R}^n} \left\| \underline{\underline{A}}\,\underline{y} - \underline{b} \right\|_2 = \left\| \underline{\underline{A}}\,\underline{x}^* - \underline{b} \right\|_2 = \|\underline{c}_2\|_2.$$

Nel caso $\|c_2\|_2 = 0$, il sistema ammette soluzione nel senso classico.

Soluzione del problema dei minimi quadrati: caso di matrici a rango non massimo

Senza voler fornire un formalismo eccessivo, questa sezione vuole, utilizzando il concetto di nucleo di un'applicazione lineare, accennare quale sia il problema nel caso di matrici a rango non massimo. Immaginando di dover risolvere un sistema sovradeterminato

$$\underline{\underline{A}}\,\underline{x} = \underline{b},$$

nel caso la matrice $\underline{\underline{A}}$ avesse rango non massimo, significherebbe che il suo nucleo conterrebbe dei vettori non banali. Sia per esempio $\underline{z} \in \ker\{\underline{\underline{A}}\}$; allora,

$$\underline{\underline{A}}\,\underline{z} = \underline{0},$$

infatti, gli elementi del nucleo di una matrice sono quelli che, se vi si moltiplica la matrice da sinistra, vengono mappati nel vettore nullo.

Detto questo, immaginiamo che $\underline{x} \in \mathcal{X}$, ossia che $\underline{x}$ sia una soluzione del problema dei minimi quadrati; allora, anche $\underline{x} + \underline{z}$ lo sarà! Infatti, dato $\alpha \neq 0$,

$$\underline{\underline{A}}\,(\underline{x} + \alpha\underline{z}) = \underline{\underline{A}}\,\underline{x} + \alpha\underline{\underline{A}}\,\underline{z} = \underline{\underline{A}}\,\underline{x} = \underline{b}.$$

Quindi, esistono moltissime soluzioni al problema dei minimi quadrati, per colpa di questo $\underline{z}$! Tuttavia, qualche sezione fa abbiamo detto che, tra tutti gli $\underline{x} \in \mathcal{X}$, ne esiste solamente uno, $\underline{x}^*$, avente norma euclidea minima. Se decidessimo che questa, in questi casi, è la soluzione che ci interessa, potremmo trovare un modo di calcolarlo. Questo modo, però, richiede l'utilizzo della decomposizione ai valori singolari, che studieremo più avanti.

Calcolo della retta di regressione

Un esempio di applicazione in cui è fondamentale utilizzare i sistemi sovradimensionati è il calcolo della retta di regressione. Il concetto di retta di regressione in qualche senso è imparentato con l'approssimazione di dati e funzioni, precedente argomento di questo testo. Il principio però è piuttosto diverso: immaginiamo di avere a che fare con molti dati di un esperimento che sembrano suggerire che l'andamento di un fenomeno sia lineare, quindi che il fenomeno si possa descrivere mediante una retta. Nella precedente sezione, però, abbiamo visto che, dati $n+1$ punti, volendo *interpolare*, ossia volendo trovare una funzione che *passi per ciascun punto*, serve usare un polinomio di grado n. Ma qui noi non vogliamo *interpolare*: noi qui vogliamo trovare una retta che sia *vicina* a questi punti, ma che non passi necessariamente per essi. Consideriamo quindi un polinomio nella forma

$$p(x) = c_1 x + c_2,$$

che approssimi *nel senso dei minimi quadrati* il set di dati forniti in Tabella 3.1; qui, possiamo immaginare di usare come ascisse lo sforzo applicato al campione, e come ordinate il risultato ottenuto, ossia la deformazione del campione conseguente all'applicazione dello sforzo. Decidiamo quindi che gli $\{x_i\}$ sono i valori degli sforzi applicati, e gli $\{y_i\}$ i valori delle conseguenti deformazioni; in questo esempio, $i = 1, \ldots, 8$. Trovare i coefficienti c_1 e c_2 richiederà ovviamente un sistema sovradeterminato; infatti, si cercherebbe di soddisfare i seguenti vincoli:

$$\begin{cases} c_1 x_1 + c_2 = y_1 \\ c_1 x_2 + c_2 = y_2 \\ c_1 x_3 + c_2 = y_3 \\ c_1 x_4 + c_2 = y_4 \\ c_1 x_5 + c_2 = y_5 \\ c_1 x_6 + c_2 = y_6 \\ c_1 x_7 + c_2 = y_7 \\ c_1 x_8 + c_2 = y_8, \end{cases}$$

che può essere riscritto in forma matriciale come

$$\underbrace{\begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ x_3 & 1 \\ x_4 & 1 \\ x_5 & 1 \\ x_6 & 1 \\ x_7 & 1 \\ x_8 & 1 \end{bmatrix}}_{\underline{\underline{A}}} \underbrace{\begin{bmatrix} c_1 \\ c_2 \end{bmatrix}}_{\underline{c}^*} = \underbrace{\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \\ y_8 \end{bmatrix}}_{\underline{b}}$$

Questo sistema sovradeterminato può essere risolto, grazie a MATLAB® , mediante il solito, mitico comando \, anche se non si tratta di sistemi quadrati; MATLAB® farà esattamente tutto quello che è stato descritto prima: calcolerà la fattorizzazione QR e troverà la soluzione del sistema $\underline{\underline{\widetilde{R}}}\,\underline{y} = \underline{c}_1$.

Segue uno script che propone un'implementazione esplicita di tutto il metodo prima descritto.

```
% Calcolo della retta di regressione mediante fattorizzazione QR

clear
close all
clc

% Inserisco i dati in MATLAB
x = [0 0.06 0.14 0.25 0.31 0.47 0.60 0.70];
y = [0 0.08 0.14 0.20 0.23 0.25 0.28 0.29];

% La matrice di sistema, A, come appena dimostrato, contiene il vettore
% delle x, e un vettore con tutti 1
A = [x' ones(8,1)];
%
n = size(A,2); % n e` il numero di colonne di A
%
% Il vettore dei termini noti contiene le y
b = y';

[Q,R] = qr(A); % calcolo fattorizzazione QR della matrice
c = Q'*b; % calcolo il vettore c come definito nelle dimostrazioni
c1 = c(1:n); % in questo caso n=2, quindi c1 sara` dato dalle prime 2
             % componenti del vettore c
c2 = c(n+1:end); % questo sara` il vettore da cui prenderemo i residui
Rtilda = R(1:n,1:n);
c_star = Rtilda\c1; % trovo la soluzione come soluzione del sistema
%
% Questo c_star e` imparentato con polyfit/polyval; infatti, si tratta dei
% coefficienti di un polinomio di grado 1; ma allora, posso anche
% utilizzare con polyval questi coefficienti!

% Definisco una griglia fine per disegnare la retta
z = linspace(-0.2,0.9,1001);
p = polyval(c_star,z);

% Calcolo il residuo
res = norm(c2)

% Disegno i dati da approssimare con cerchietti rossi e la retta in blu
figure
hold on
grid on
plot(x,y,'ro',z,p,'b','LineWidth',2)
```

Alternativamente a usare la fattorizzazione QR, è possibile, come già detto, usare il comando \. Qua si propone un'implementazione della cosa.

```
% Calcolo della retta di regressione mediante soluzione di un sistema
% lineare sovradimensionato grazie al comando \

clear
close all
clc

% Inserisco i dati in MATLAB
x = [0 0.06 0.14 0.25 0.31 0.47 0.60 0.70];
y = [0 0.08 0.14 0.20 0.23 0.25 0.28 0.29];

% La matrice di sistema, A, come appena dimostrato, contiene il vettore
% delle x, e un vettore con tutti 1
A = [x' ones(8,1)];

% Il vettore dei termini noti contiene le y
b = y';

% Uso il comando \ per trovare la soluzione
c_star = A\b;

% Definisco una griglia fine per disegnare la retta
z = linspace(-0.2,0.9,1001);
p = polyval(c_star,z);

% Disegno i dati da approssimare con cerchietti rossi e la retta in blu
figure
hold on
grid on
```

Tabella 3.1: Esempio di dati sperimentali: legame tra sforzo applicato su un campione di disco intervertrebrale, e relativa deformazione.

Sforzo σ, MPa	Deformazione ε, cm
0.00	0.00
0.06	0.08
0.14	0.14
0.25	0.20
0.31	0.23
0.47	0.25
0.60	0.28
0.70	0.29

```
plot(x,y,'ro',z,p,'b','LineWidth',2)
```

Il metodo più popolare per il calcolo della retta di regressione, tuttavia, è basato sul già noto comando `polyfit`. Infatti, questo comando presenta tre argomenti: le `x`, le `y`, ma anche il grado del polinomio. Nell'ambito dei polinomi interpolatori, avevamo sempre scelto, come grado del polinomio, il numero di nodi di interpolazione meno 1. Tuttavia, nel caso introducessimo un numero inferiore, come 1, invece di interpolare, otteniamo l'approssimazione nel senso dei minimi quadrati della funzione con un polinomio di grado 1: una retta! Viene ora riportata un'implementazione MATLAB® anche con questo metodo. Si noti che, nel caso si volesse calcolare il residuo, non è necessario passare dal metodo con la fattorizzazione QR e ricavare il vettore $\underline{c}_2$. Infatti, il residuo si puo' calcolare facendo `polyval` sugli stessi dati $\underline{x}$ da cui siamo partiti, e calcolando la differenza tra le `p` così trovate e le `y` di partenza; nel caso del polinomio interpolante, questo sarebbe stato nullo, dal momento che le condizioni di interpolazione sarebbero state soddisfatte esattamente, ma qui non abbiamo abbastanza gradi di libertà e così ci sarà un errore di approssimazione anche per i dati di partenza!

```
% Calcolo della retta di regressione mediante comando polyfit

clear
close all
clc

% Inserisco i dati in MATLAB
x = [0 0.06 0.14 0.25 0.31 0.47 0.60 0.70];
y = [0 0.08 0.14 0.20 0.23 0.25 0.28 0.29];

% Calcolo c_star mediante il comando polyfit, usando come grado 1
c_star = polyfit(x,y,1);

% Definisco una griglia fine per disegnare la retta
z = linspace(-0.2,0.9,1001);

% Uso polyval per valutare il polinomio
p = polyval(c_star,z);

% Disegno i dati da approssimare con cerchietti rossi e la retta in blu
figure
hold on
grid on
plot(x,y,'ro',z,p,'b','LineWidth',2)

% Calcolo il residuo
p = polyval(c_star,x);
res = norm(p-y,2)
```

Bibliografia

[1] A. Quarteroni e F. Saleri, "Introduzione al calcolo scientifico," 3ª edizione, Springer-Verlag Italia, Milano, 2006.

[2] G. Monegato, "Metodi e algoritmi per il calcolo numerico," *CLUT*, Torino, Settembre 2008.

Appendice B

Appendice: Sistemi lineari

B.1 Matrici ortogonali

«Ma che vuol dire che una matrice è ortogonale?!»

Per liquidare questa domanda potremmo semplicemente ricordare una definizione fornita qualche sezione fa; una matrice quadrata $\underline{\underline{A}} \in \mathbb{R}^{m,m}$ si dice **ortogonale** se

$$\underline{\underline{A}}^{\mathrm{T}}\underline{\underline{A}} = \underline{\underline{A}}\,\underline{\underline{A}}^{\mathrm{T}} = \underline{\underline{I}},$$

dove $\underline{\underline{I}}$ è la matrice identità. Detto così, non mi fa capire molto. Volendo però guardare in faccia meglio questa definizione, ripetendo qualcosa di già accennato, ricordiamo che l'inversa di una matrice $\underline{\underline{A}}$, indicata con $\underline{\underline{A}}^{-1}$, si definisce tale per cui

$$\underline{\underline{A}}^{-1}\underline{\underline{A}} = \underline{\underline{A}}\,\underline{\underline{A}}^{-1} = \underline{\underline{I}}.$$

Le due definizioni appena viste si somigliano molto, non fosse che in un caso si ha la trasposta, nell'altro l'inversa. Ma questo inizia già a suonarmi meglio: **le matrici ortogonali sono quelle matrici che hanno l'inversa uguale alla loro trasposta!** Insomma, questo è già più carino, perché insomma a me piace di più calcolare la trasposta di una matrice, che è una banalissima operazione di scambio righe e colonne, piuttosto che un'inversione che è (penso) la più complicata delle operazioni che si possano effettuare su una matrice. Quindi, che queste matrici ortogonali abbiano qualcosa di speciale, è cosa chiara. Però c'è ancora una cosa che da qui sfugge: ma perché queste matrici vengono chiamate proprio **ortogonali**? Cioè, il concetto di ortogonalità è un qualcosa che, più che a una matrice, viene bene legare a dei vettori, a dei segmenti: due segmenti ortogonali sono perpendicolari, formano insomma un angolo retto.

Per cercare di capire cosa c'entrino gli angoli retti con le matrici, concentriamoci su un esempio molto popolare: la cosiddetta **matrice di rotazione**

$$\underline{\underline{A}} = \begin{bmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{bmatrix}.$$

Questa matrice viene chiamata così perché, se viene moltiplicata da sinistra a un vettore colonna, lo ruota di un angolo pari a φ, in senso antiorario[1]; in altre parole, questa è la rappresentazione matriciale dell'applicazione lineare di rotazione di un vettore su un piano. Cercando di essere concreti, fissiamo un certo valore di φ, per esempio $\varphi = 30°$. In questa condizione, la matrice di rotazione $\underline{\underline{A}}$ sarà

$$\underline{\underline{A}} = \begin{bmatrix} \dfrac{\sqrt{3}}{2} & -\dfrac{1}{2} \\ \dfrac{1}{2} & \dfrac{\sqrt{3}}{2} \end{bmatrix}.$$

[1] volendo ottenere la rotazione in senso orario è sufficiente usare un φ negativo, oppure usarne uno positivo e scambiare il segno dei due seni nella matrice

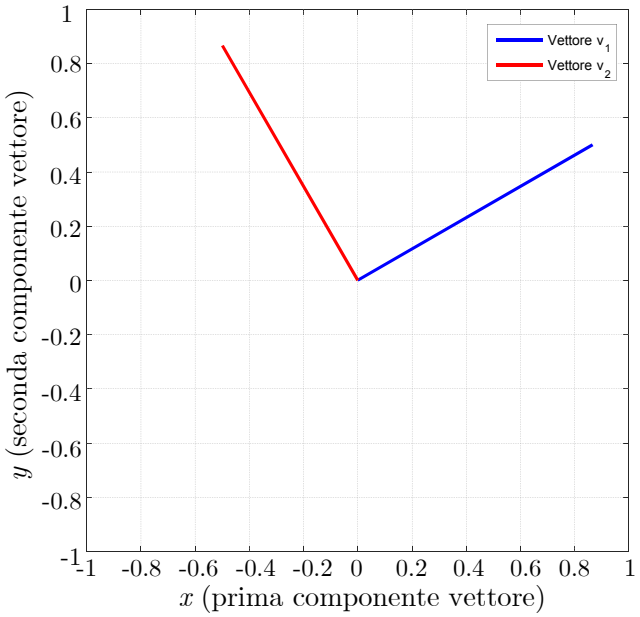

Figura B.1: Rappresentazione geometrica dei vettori $\underline{v}_1$ (blu) e $\underline{v}_2$ (rosso) ottenuti dalle colonne della matrice di rotazione.

Se per esempio prendessimo il vettore $\underline{x} = [1 \;\; 0]^T$ e gli moltiplicassimo la matrice appena presentata da sinistra, otterremmo

$$\begin{bmatrix} \frac{\sqrt{3}}{2} & -\frac{1}{2} \\ \frac{1}{2} & \frac{\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{3}}{2} \\ \frac{1}{2} \end{bmatrix},$$

che è proprio il vettore $\underline{x}$, ruotato di 30°!

Tornando a noi, definiamo $\underline{v}_1$ e $\underline{v}_2$ i due vettori colonna ottenuti rispettivamente dalla prima e dalla seconda colonna di questa matrice:

$$\underline{v}_1 = \begin{bmatrix} \frac{\sqrt{3}}{2} \\ \frac{1}{2} \end{bmatrix}, \qquad \underline{v}_2 = \begin{bmatrix} -\frac{1}{2} \\ \frac{\sqrt{3}}{2} \end{bmatrix}.$$

In Algebra Lineare, ma pure in Fisica, si parla di ortogonalità tra due vettori quando il loro prodotto scalare è nullo. Per vedere cosa succede qui, valutiamo il prodotto scalare tra $\underline{v}_1$ e $\underline{v}_2$; questo si può calcolare per esempio usando il prodotto riga per colonna, trasponendo uno dei due vettori e moltiplicandoli:

$$\underline{v}_1 \cdot \underline{v}_2 = \underline{v}_1^T \underline{v}_2 = \frac{\sqrt{3}}{2} \times \left(-\frac{1}{2}\right) + \frac{1}{2} \times \frac{\sqrt{3}}{2} = 0.$$

Sorpresi? Qui le cose iniziano a farsi chiare sul serio! Cioè, costruiamo tanti vettori colonna, ciascuno con una colonna della matrice; calcoliamo il prodotto scalare tra due di questi vettori, e scopriamo che fa 0. Non solo: proviamo a questo punto a calcolare il prodotto scalare tra $\underline{v}_2$ e sé stesso:

$$\underline{v}_2 \cdot \underline{v}_2 = \underline{v}_2^T \underline{v}_2 = \frac{\sqrt{3}}{2} \times \frac{\sqrt{3}}{2} + \left(-\frac{1}{2}\right) \times \left(-\frac{1}{2}\right) = \frac{3}{4} + \frac{1}{4} = 1.$$

E potremmo scoprire che anche $\underline{v}_1 \cdot \underline{v}_1 = 1$. Ma non è finita qui: tutto questo può essere anche rappresentato graficamente, per $\mathbb{R}^2$, in cui è possibile identificare la prima e la seconda componente come ascisse x e ordinate y di un piano cartesiano. È possibile disegnare, come fatto in Fig. B.1, i due vettori

$\underline{v}_1$ e $\underline{v}_2$ su un piano cartesiano, rispettivamente in colore rosso e blu[2]. Questa rappresentazione grafica consiste semplicemente nel mostrare i segmenti che uniscono l'origine degli assi alle coordinate (x, y), le quali sono identificate con le componenti di $\underline{v}_1 = [x_1 \ y_1]^T$ e $\underline{v}_2 = [x_2 \ y_2]^T$. Dal grafico appare chiaramente che i due vettori sono perpendicolari, ossia che tra loro è presente un angolo pari a 90°. Il seguente script:

```
clear
close all
clc

theta = 30; % gradi

% Scrivo l'espressione della matrice di rotazione; si noti che le funzioni
% sin e cos su MATLAB accettano radianti, quindi bisogna convertire da
% gradi a radianti
A = [cos(theta*pi/180) -sin(theta*pi/180);
     sin(theta*pi/180) cos(theta*pi/180)];

v1 = A(:,1); % definisco v1 come la prima colonna di A
v2 = A(:,2); % definisco v2 come la seconda colonna di A

figure
set(gcf,'Position',[616 226 627 571])
grid on
hold on
box on
% Disegno i due vettori come i segmenti che congiungono l'origine [0,0] con
% il vettore [v1(1),v1(2)], e idem per [v2(1),v2(2)].
plot([0,v1(1)],[0,v1(2)],'b','LineWidth',2) % plot vettore v1
plot([0,v2(1)],[0,v2(2)],'r','LineWidth',2) % plot vettore v2
%
% faccio in modo che MATLAB non riscali gli assi ma attribuisca ad ascisse
% e ordinate la stessa scala: le stesse distanze su x e y hanno lo stesso
% significato
axis equal
% faccio in modo che MATLAB disegni il piano sia per x sia per y
% nell'intervallo [-1,1] x [-1,1]
axis([-1,1,-1,1])
xlabel('x (prima componente vettori)')
ylabel('y (seconda componente vettori)')
legend('Vettore v_1','Vettore v_2','Location','NorthEast')

prodotto_scalare = v1'*v2
```

permette di disegnare i due vettori ottenuti dalla matrice di rotazione per un generico valore di ϑ, che si può fissare all'inizio del codice.

Per riassumere quanto abbiamo appena capito, una matrice ortogonale è una matrice le cui colonne sono vettori tra di loro ortogonali e aventi norma euclidea pari a 1. Dati $\underline{v}_i$, $\underline{v}_j$ il i-esimo e il j-esimo vettore colonna, quindi, dire che questi sono ortogonali corrisponde a dire che:

$$\underline{v}_i \cdot \underline{v}_j = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases},$$

ossia che il loro prodotto scalare è uguale a 1 solo se si considera lo stesso vettore, e 0 nel caso i vettori fossero stati costruiti a partire da colonne diverse. Questa definizione è valida in m dimensioni, ossia per vettori dotati di m componenti, ma, per $m = 2$ (e con un po' di fatica in più $m = 3$) è anche possibile interpretarla geometricamente, in termini della presenza di un angolo retto tra i due vettori.

[2] questo permette di capire ancora meglio cosa significhi *matrice di rotazione*: le righe rossa e blu sono un po' come gli assi x e y, soggetti alla rotazione; è evidente che se moltiplicheremo da sinistra la matrice di rotazione a qualsiasi vettore del nostro spazio, questo sarà ruotato di φ, come in questo esempio

Capitolo 4

Autovalori e valori singolari di matrici

Accenni sulla storia degli autovalori/autovettori

L'obiettivo di questo capitolo sarà, dopo aver *rinfrescato* i principali concetti legati agli autovalori e agli autovettori, introdurre alcune tecniche numeriche atte a calcolarli. Vorrei tuttavia approfittare di questo spazio di introduzione per accennare la storia di queste idee, suggerendo ai lettori interessati di leggere ulteriori dettagli in [1].

Andando molto indietro nel tempo, Cartesio[1] nel 1637 propose di trasformare una generica forma quadratica[2] $ax^2 + 2bxy + cy^2$ nella sua forma normale $\alpha x^2 + \beta y^2$ grazie all'identificazione dei suoi *assi principali*. Passando per i grandi Eulero, Lagrange, Jacobi e Cauchy, si è arrivati a Sylvester e Cayley, che hanno applicato il principio degli *assi principali* al calcolo matriciale, sviluppato in quegli anni (si parla degli anni '50 del 1800). Qui viene un po' fuori il desiderio di sfruttare questa teoria per cercare di *sostituire le matrici con dei numeri*, a patto di saper rappresentare i vettori mediante questi *assi principali*.

Questa idea di sostituire cose complicate con dei numeri era già stata applicata qualche anno prima. Durante la rivoluzione industriale la termodinamica andava di moda, e uno dei matematici più *eroici* in questo contesto era stato Fourier che, con un trucco oggi noto come **trasformata di Fourier**, aveva trovato un modo di *far sparire* le derivate dalle equazioni differenziali, trasformandole in equazioni algebriche. Questa cosa a me ricorda un po' il far sparire una matrice per sostituirle un numero.

Finora non ho nominato *autovalori* o *autovettori*, ricorrendo ai nomi utilizzati dagli antichi; questo, perché questi nomi sono abbastanza recenti. Autovalore e autovettore sono traduzioni non proprio fedelissime di **eigenvalue** e **eigenvector**. Tuttavia, anche chi possiede una certa dimestichezza con l'inglese, potrebbe non avere familiarità col prefisso *eigen-*. In effetti, i termini originali sono stati **Eigenwert** e **Eigenvektor**, e sono tedeschi. Infatti, nella Germania nazista c'era molto interesse verso questi concetti, perché su di essi si basavano sia la teoria dei Campi Elettromagnetici, approfondita al fine di produrre dei RADAR coi quali rilevare la presenza di nemici, sia la Meccanica Quantistica, utile per arrivare in qualche modo alla bomba atomica.

Oggi, gli autovalori e autovettori trovano applicazione in moltissimi campi: dalla Meccanica delle Vibrazioni, all'Automatica, all'Elettronica, all'Elettromagnetismo, alla appunto Meccanica Quantistica, e chi più ne ha più ne metta.

4.1 Concetti fondamentali sugli autovalori

Dopo la parentesi della scorsa sezione sulle matrici rettangolari torniamo a parlare di matrici quadrate $\underline{\underline{A}} \in \mathbb{R}^{n,n}$ e, in questo contesto, introduciamo una classe di vettori un po' *mafiosi*, che insomma si sentono al di sopra delle regole dei normali *cittadini degli spazi vettoriali*. In generale, quando applichiamo una matrice a un vettore colonna moltiplicandola ad esso da sinistra, il vettore viene trasformato in un altro vettore che, volendo dare un'interpretazione geometrica[3], ha direzione e lunghezza diverse da quelle

[1] che dunque non ha solo sfruttato le coordinate geometriche precedentemente inventate da Nicola d'Oresme per legare Algebra e Geometria e capito che *cogito, ergo sum* ;-)

[2] dunque orientata arbitrariamente in un piano nel caso bidimensionale, o nello spazio nel caso tridimensionale

[3] al solito, valida per i casi a due e/o a tre dimensioni

di partenza. Cosa fanno di così strano questi vettori? Beh, proviamolo su un esempio: proviamo a moltiplicare da sinistra la matrice

$$\underline{\underline{A}} = \begin{bmatrix} 2 & 3 \\ 2 & 1 \end{bmatrix}$$

al vettore

$$\underline{x} = \begin{bmatrix} 3 \\ 2 \end{bmatrix}.$$

Vediamo cosa succede calcolando $\underline{\underline{A}}\,\underline{x}$:

$$\underline{\underline{A}}\,\underline{x} = \begin{bmatrix} 2 & 3 \\ 2 & 1 \end{bmatrix}\begin{bmatrix} 3 \\ 2 \end{bmatrix} = \begin{bmatrix} 2\times 3 + 3\times 2 \\ 2\times 3 + 1\times 2 \end{bmatrix} = \begin{bmatrix} 12 \\ 8 \end{bmatrix} = 4 \times \begin{bmatrix} 3 \\ 2 \end{bmatrix}.$$

Ohibò. Moltiplicando $\underline{\underline{A}}$ per $\underline{x}$, viene fuori un vettore che ha la stessa direzione di $\underline{x}$, ma che è moltiplicato per un numero, 4. Quindi, il vettore non ha subito alcuna rotazione nello spazio, ma solo un *allungamento* di 4 volte. A questo punto, potremmo chiederci:

> «Ma siamo sicuri che il *mafioso* in questione sia il vettore, e non la matrice?»

Beh, abbastanza: prendiamo la stessa matrice e moltiplichiamola per un altro vettore $\underline{v}$:

$$\underline{\underline{A}}\,\underline{v} = \begin{bmatrix} 2 & 3 \\ 2 & 1 \end{bmatrix}\begin{bmatrix} 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 2\times 2 + 3\times 3 \\ 2\times 2 + 1\times 3 \end{bmatrix} = \begin{bmatrix} 13 \\ 7 \end{bmatrix},$$

che non è multiplo di $\underline{v}$: possiamo osservare sia un allungamento, sia una rotazione! Quindi, per questa matrice $\underline{\underline{A}}$, $\underline{x}$ soddisfa questa strana relazione, ma $\underline{v}$ non lo fa. Quando vale una relazione nella forma

$$\underline{\underline{A}}\,\underline{x} = \lambda\,\underline{x}, \tag{4.1}$$

dove λ è un numero, allora $\underline{x}$ viene detto **autovettore** di $\underline{\underline{A}}$, e λ viene detto **autovalore relativo a esso**. Questa relazione si può scrivere, portando tutti i termini a sinistra e raccogliendo, come il seguente sistema lineare omogeneo:

$$(\underline{\underline{A}} - \lambda\underline{\underline{I}})\underline{x} = \underline{0}. \tag{4.2}$$

Dunque, $\underline{x}$ si può anche interpretare come una soluzione **non banale** di questo sistema[4]. Per il teorema di Rouché-Capelli, perché esista una soluzione non banale del sistema, si richiede che $\underline{x}$ appartenga allo spazio nullo della matrice $(\underline{\underline{A}} - \lambda\underline{\underline{I}})$. Perché questa abbia uno spazio nullo, il suo determinante deve essere pari a 0. In effetti, in Algebra Lineare, al fine di trovare un autovalore, uno deve imporre la condizione

$$\det\{\underline{\underline{A}} - \lambda\underline{\underline{I}}\} = 0$$

e questo darebbe luogo all'equazione caratteristica, che poi deve essere risolta. L'equazione caratteristica è un'equazione algebrica avente come incognite i λ, ovvero gli autovalori della matrice. Dal momento che si può vedere che i λ sono sulla diagonale della matrice, e nascendo questa equazione da un determinante[5], l'equazione caratteristica avrà grado pari a n, dove n è il numero di righe/colonne della matrice. Il teorema fondamentale dell'Algebra garantisce che l'equazione caratteristica ha n soluzioni, che possono essere reali o complesse; in presenza di un autovalore complesso, anche il suo complesso coniugato sarà un autovalore.

Data una matrice `A`, MATLAB® è in grado di calcolarne autovalori e autovettori, mediante il comando `eig` (da *eigenvalue*, come spiegato prima). In particolare,

- se si utilizza il comando `eig` chiedendo un solo argomento di uscita, quindi nella forma

  ```
  E = eig(A);
  ```

  la variabile `E` sarà un **vettore**, contenente tutti gli autovalori della matrice `A`;

[4] infatti nell'esempio appena visto, $\underline{x}$ non era il vettore contenente solo zeri ;-)

[5] il cui calcolo richiede il prodotto degli elementi della diagonale, arrivando quindi a λ^n

- se si utilizza il comando `eig` chiedendo due argomenti in uscita, quindi nella forma

  ```
  [V,E] = eig(A);
  ```

  le variabili `V` e `E` saranno **due matrici** aventi la stessa dimensione di `A`; in particolare, `V` sarà la matrice avente, come ciascuna colonna, l'autovettore di `A` associato all'autovalore contenuto nella corrispettiva posizione sulla diagonale di `E`; quest'ultima, è una matrice diagonale.

4.1.1 Potenze e esponenziale di una matrice

Dato λ_i il i-esimo autovalore della matrice $\underline{\underline{A}}$, $\underline{x}_i$ sarà l'autovettore a esso associato. Una matrice $\underline{\underline{A}} \in \mathbb{R}^{n,n}$ ha quindi n autovalori (non necessariamente distinti tra loro), $\{\lambda_i\}$, ciascuno dei quali è associato a un autovettore. Volendo, possiamo definire una matrice $\underline{\underline{X}}$, le cui colonne sono i vari autovettori $\{\underline{x}_i\}$:

$$\underline{\underline{X}} = \begin{bmatrix} \underline{x}_1 & \underline{x}_2 & \underline{x}_3 & \dots & \underline{x}_n \end{bmatrix}.$$

Immaginiamo a questo punto di moltiplicare da sinistra la matrice $\underline{\underline{A}}$ per ciascuno di questi vettori; otterremo, in virtù della relazione (4.1) applicata a ciascuno degli autovettori, che

$$\underline{\underline{A}}\,\underline{\underline{X}} = \begin{bmatrix} \underline{\underline{A}}\,\underline{x}_1 & \underline{\underline{A}}\,\underline{x}_2 & \underline{\underline{A}}\,\underline{x}_3 & \dots & \underline{\underline{A}}\,\underline{x}_n \end{bmatrix} = \begin{bmatrix} \lambda_1\,\underline{x}_1 & \lambda_2\,\underline{x}_2 & \lambda_3\,\underline{x}_3 & \dots & \lambda_n\,\underline{x}_n \end{bmatrix}.$$

A questo punto è utile definire la matrice diagonale $\underline{\underline{D}}$

$$\underline{\underline{D}} = \begin{bmatrix} \lambda_1 & 0 & 0 & \cdots & 0 \\ 0 & \lambda_2 & 0 & \cdots & 0 \\ 0 & 0 & \lambda_3 & \cdots & 0 \\ 0 & 0 & 0 & \ddots & 0 \\ \vdots & \vdots & \vdots & \vdots & \lambda_n \end{bmatrix},$$

avente come ciascun elemento della diagonale un autovalore. Questa aiuta a scrivere la precedente espressione nella forma compatta

$$\underline{\underline{A}}\,\underline{\underline{X}} = \underline{\underline{X}}\,\underline{\underline{D}},$$

che altri non è che la proprietà (4.1), dove invece di considerare un autovettore $\underline{x}_i$ per volta, li consideriamo tutti assieme, messi in un'unica matrice $\underline{\underline{X}}$. Quest'ultima proprietà può essere riscritta moltiplicando ambo i membri da destra per $\underline{\underline{X}}^{-1}$, ottenendo

$$\underline{\underline{A}} = \underline{\underline{X}}\,\underline{\underline{D}}\,\underline{\underline{X}}^{-1}. \tag{4.3}$$

Questa proprietà dimostra come sia possibile **diagonalizzare** la matrice $\underline{\underline{A}}$, ovvero scriverla come prodotto di tre matrici, dove al centro di questa specie di *panino* abbiamo una matrice diagonale a fare da *guarnizione*, e $\underline{\underline{X}}$, $\underline{\underline{X}}^{-1}$ a fare da *fette di pane*. Questa rappresentazione è particolarmente interessante perché ci permette di enunciare la prima situazione in cui la decomposizione agli autovalori è fondamentale. Immaginiamo di voler calcolare il quadrato di una matrice, $\underline{\underline{A}}^2$. Anziché calcolare esplicitamente il prodotto, è possibile sostituire (4.3), ottenendo

$$\underline{\underline{A}}^2 = \left(\underline{\underline{X}}\,\underline{\underline{D}}\,\underline{\underline{X}}^{-1}\right)\left(\underline{\underline{X}}\,\underline{\underline{D}}\,\underline{\underline{X}}^{-1}\right) = \underline{\underline{X}}\,\underline{\underline{D}}\,\underline{\underline{D}}\,\underline{\underline{X}}^{-1},$$

dove però calcolare $\underline{\underline{D}}\,\underline{\underline{D}} = \underline{\underline{D}}^2$ è assolutamente banale: è sufficiente calcolare il quadrato di ciascun elemento sulla diagonale, ovvero di ciascun autovalore! Questa cosa, ovviamente, si può generalizzare con grande facilità, permettendoci di ottenere

$$\underline{\underline{A}}^k = \underline{\underline{X}}\,\underline{\underline{D}}^k\,\underline{\underline{X}}^{-1},$$

dove $d_{ii}^k = \lambda_i^k$. Ma questa cosa può valere anche per l'inversa della matrice! Infatti,

$$\underline{\underline{A}}^{-1} = \underline{\underline{X}}\,\underline{\underline{D}}^{-1}\,\underline{\underline{X}}^{-1},$$

dove quindi è sufficiente calcolare ciascuna componente come il reciproco di ciascun autovalore. Questa idea può essere sfruttata al fine di definire e calcolare l'esponenziale di una matrice. Sappiamo infatti che lo sviluppo di Taylor di una funzione esponenziale è:

$$e^x \simeq 1 + x + \frac{x^2}{2} + \frac{x^3}{3!} + ... + \frac{x^n}{n!}.$$

Allora, possiamo definire, analogamente,

$$e^{\underline{\underline{A}}} \simeq \underline{\underline{I}} + \underline{\underline{A}} + \frac{1}{2}\underline{\underline{A}}^2 + \frac{1}{3!}\underline{\underline{A}}^3 + ... + \frac{1}{n!}\underline{\underline{A}}^n,$$

dove ciascuno degli $\underline{\underline{A}}^k$ si può scrivere con la sua forma diagonalizzata, permettendoci di ottenere

$$e^{\underline{\underline{A}}} \simeq \underline{\underline{X}}\left[\underline{\underline{I}} + \underline{\underline{D}} + \frac{1}{2}\underline{\underline{D}}^2 + \frac{1}{3!}\underline{\underline{D}}^3 + ... + \frac{1}{n!}\underline{\underline{D}}^n\right]\underline{\underline{X}}^{-1},$$

ovvero,

$$e^{\underline{\underline{A}}} = \underline{\underline{X}}e^{\underline{\underline{D}}}\underline{\underline{X}}^{-1},$$

dove però è chiaro come si fa a calcolare l'esponenziale della matrice centrale: è sufficiente calcolare tutti gli esponenziali dei vari elementi sulla diagonale!

4.1.2 Diagonalizzazione di matrici simmetriche

Discutiamo a questo punto una particolarità della diagonalizzazione delle matrici simmetriche. Prima di tutto, ricordiamoci che una matrice è simmetrica se è uguale alla sua trasposta:

$$\underline{\underline{A}} = \underline{\underline{A}}^{\mathrm{T}}.$$

Sostituiamo in questa relazione l'espressione del panino (4.3):

$$\underline{\underline{X}}\,\underline{\underline{D}}\,\underline{\underline{X}}^{-1} = \left(\underline{\underline{X}}\,\underline{\underline{D}}\,\underline{\underline{X}}^{-1}\right)^{\mathrm{T}} = \left(\underline{\underline{X}}^{-1}\right)^{\mathrm{T}}\,\underline{\underline{D}}^{\mathrm{T}}\underline{\underline{X}}^{\mathrm{T}}.$$

Questa espressione si può riscrivere, moltiplicando per $\underline{\underline{X}}^{-1}$ da sinistra e per $\underline{\underline{X}}$ da destra, come:

$$\underline{\underline{D}} = \underline{\underline{X}}^{-1}\left(\underline{\underline{X}}^{-1}\right)^{\mathrm{T}}\,\underline{\underline{D}}\,\underline{\underline{X}}^{\mathrm{T}}\underline{\underline{X}},$$

sfruttando il fatto che $\underline{\underline{D}}$ è diagonale. Perché questa eguaglianza sia valida, è necessario che

$$\underline{\underline{X}}^{-1}\left(\underline{\underline{X}}^{-1}\right)^{\mathrm{T}} = \underline{\underline{I}},$$

e, equivalentemente, che

$$\underline{\underline{X}}^{\mathrm{T}}\underline{\underline{X}} = \underline{\underline{I}},$$

entrambe condizioni che dimostrano che la matrice degli autovettori, $\underline{\underline{X}}$, è ortogonale. Dal momento che $\underline{\underline{X}}$ è costruita incolonnando i vari autovettori, possiamo concludere che **gli autovettori di una matrice simmetrica sono tra loro ortogonali**.

4.1.3 Quoziente di Rayleigh

Introduciamo a questo punto il concetto di quoziente di Rayleigh. Partendo da una matrice $\underline{\underline{A}}$, da un suo autovalore λ e dall'autovettore a esso associato, $\underline{x}$, vale (4.1):

$$\underline{\underline{A}}\,\underline{x} = \lambda\underline{x}.$$

A questa espressione, moltiplichiamo ambo i membri da sinistra per il vettore $\underline{x}^{\mathrm{H}}$, detto **Hermitiano** del vettore $\underline{x}$. Questo è semplicemente il vettore trasposto di $\underline{x}$, dove al posto di usare ciascun elemento, se ne prende il **complesso coniugato**; nel caso in cui le componenti di $\underline{x}$ fossero reali, allora questa operazione è semplicemente la trasposizione. Si ottiene:

$$\underline{x}^{\mathrm{H}}\underline{\underline{A}}\,\underline{x} = \lambda\underline{x}^{\mathrm{H}}\underline{x}.$$

Possiamo osservare che sia il membro sinistro, sia il membro destro, sono dei numeri; di conseguenza, è lecito isolare λ ottenendo la seguente espressione:

$$\lambda = \frac{\underline{x}^{\mathrm{H}} \underline{\underline{A}}\, \underline{x}}{\underline{x}^{\mathrm{H}} \underline{x}}.$$

Il rapporto

$$r_{\underline{\underline{A}}} = \frac{\underline{x}^{\mathrm{H}} \underline{\underline{A}}\, \underline{x}}{\underline{x}^{\mathrm{H}} \underline{x}} \tag{4.4}$$

viene detto **quoziente di Rayleigh** e, se $\underline{x}$ è un autovettore della matrice $\underline{\underline{A}}$, questo coincide con l'autovalore λ a esso associato. Questo strumento è molto utile dal momento che ci permette, una volta noto un certo autovettore, di poter ricavare immediatamente l'autovalore a esso associato, con un conticino molto semplice. Spesso, negli esercizi di Algebra Lineare, quello che si fa è infatti il contrario, ossia risolvere l'equazione caratteristica e ricavare l'elenco degli autovalori, e poi, risolvendo per ciascuno di essi il corrispondente sistema lineare omogeneo, trovare gli autovettori. Alcuni dei metodi che studieremo si basano sull'effettuare varie iterazioni di un certo algoritmo, dove il risultato di ciascuna iterazione è una migliore stima dell'autovettore; noto l'autovettore, potremo trovare quindi immediatamente l'autovalore, per procedere con la successiva iterazione.

È giunto il momento di scavare nel nostro passato, e chiudere alcuni dei conti lasciati in sospeso.

«Abbiamo già visto queste espressioni? Numeratore, denominatore... Sembrano familiari!»

Prima di tutto, concentriamoci sul denominatore, che è quello più semplice: è semplicemente la norma 2 del vettore $\underline{x}$!

$$||\underline{x}||_2^2 = \underline{x}^{\mathrm{T}} \underline{x},$$

cosa che vale anche quando si prende l'Hermitiano.

«E il numeratore?»

Questo è un po' più difficile ma, se ci sforziamo, possiamo ricordarci che è il termine che appariva nella definizione di matrice simmetrica definita positiva! E questo, finalmente, ci permette di dare un criterio operativo: **una matrice è simmetrica definita positiva se e solo se tutti i suoi autovalori sono positivi**. Al fine di impostare la nostra dimostrazione, prendiamo un generico vettore $\underline{u}$, che dunque non è necessariamente un autovettore della matrice. Tuttavia, $\underline{u}$ può essere certamente scritto come una combinazione lineare di autovettori; considerando[6] per esempio una situazione tale per cui $\underline{u}$ si può scrivere semplicemente come combinazione lineare di due autovettori, si avrebbe:

$$\underline{u} = \alpha \underline{x}_1 + \beta \underline{x}_2.$$

A questo punto se calcoliamo $\underline{\underline{A}}\, \underline{u}$ otteniamo, sostituendo la riscrittura in termini di combinazione lineare di autovettori,

$$\underline{\underline{A}}\, \underline{u} = \underline{\underline{A}} \left(\alpha \underline{x}_1 + \beta \underline{x}_2 \right) = \alpha \underline{\underline{A}}\, \underline{x}_1 + \beta \underline{\underline{A}}\, \underline{x}_2 = \alpha \lambda_1 \underline{x}_1 + \beta \lambda_2 \underline{x}_2.$$

A questo punto, moltiplichiamo da sinistra il vettore $\underline{u}^{\mathrm{T}}$:

$$\underline{u}^{\mathrm{T}} \underline{\underline{A}}\, \underline{u} = \left(\alpha \underline{x}_1^{\mathrm{T}} + \beta \underline{x}_2^{\mathrm{T}} \right) \left(\alpha \lambda_1 \underline{x}_1 + \beta \lambda_2 \underline{x}_2 \right) = \alpha^2 \lambda_1 \underline{x}_1^{\mathrm{T}} \underline{x}_1 + \alpha \beta \lambda_2 \underline{x}_1^{\mathrm{T}} \underline{x}_2 + \beta \alpha \lambda_1 \underline{x}_2^{\mathrm{T}} \underline{x}_1 + \beta^2 \lambda_2 \underline{x}_2^{\mathrm{T}} \underline{x}_2,$$

dove però i prodotti tra autovettori diversi sono nulli, dal momento che abbiamo appena dimostrato che la matrice degli autovettori è ortogonale. Riconosciamo che gli altri termini contengono i quadrati delle norme 2 degli autovettori; possiamo quindi scrivere che

$$\underline{u}^{\mathrm{T}} \underline{\underline{A}}\, \underline{u} = \alpha^2 ||\underline{x}_1||_2^2 \lambda_1 + \beta^2 ||\underline{x}_2||_2^2 \lambda_2$$

e, se gli autovalori λ_1 e λ_2 sono entrambi positivi, è evidente che dovrà essere positivo anche il membro sinistro[7], dimostrando che una matrice simmetrica avente autovalori positivi è anche definita positiva.

Dimostrare nell'altro senso questa proprietà è molto più semplice: se per ipotesi $\underline{u}^{\mathrm{T}} \underline{\underline{A}}\, \underline{u}$ è maggiore di zero per qualsiasi vettore $\underline{u}$ appartenente allo spazio vettoriale $\mathbb{R}^n$, allora è sufficiente scegliere come

[6] per fare un esempio semplice, ma la prova con n autovettori sarebbe del tutto simile

[7] infatti, tutti gli altri termini contenuti nell'espressione sono al quadrato, e quindi non influenzano il segno dei vari termini

esempi di vettori $\underline{u}$ gli autovettori $\underline{x}_i$, e, poiché al denominatore si ha una norma[8], gli autovalori non potranno che essere maggiori di zero. In ogni caso, il mio obiettivo era, più che dimostrarvi queste proprietà, far vedere un po' di questi calcoletti, che possono sempre tornare utili.

4.1.4 Raggio spettrale e norma spettrale di una matrice

Introduciamo a questo punto un altro concetto: quello di **raggio spettrale**. Data una matrice $\underline{\underline{A}}$, il suo raggio spettrale è **il modulo dell'autovalore di massimo modulo**:

$$\rho(\underline{\underline{A}}) = \max_{i=1,\dots,n} \{|\lambda_i| \ : \ \lambda_i \text{ autovalore di } \underline{\underline{A}}\}. \tag{4.5}$$

Calcolare questo numero mediante MATLAB® non è molto difficile:

```
rho = max(abs(eig(A)));
```

Questa definizione ci permette di chiudere un altro argomento lasciato in sospeso. Infatti, non ho mai detto come si fa a calcolare la norma 2 di una **matrice**! Beh, questa si definisce come

$$\left\|\underline{\underline{A}}\right\|_2 = \sqrt{\rho(\underline{\underline{A}}^{\mathrm{T}}\underline{\underline{A}})}. \tag{4.6}$$

Esistono situazioni in cui questa espressione si semplifica, per esempio in caso di matrici simmetriche. Infatti, se $\underline{\underline{A}}$ è una matrice simmetrica, si ha che

$$\underline{\underline{A}} = \underline{\underline{A}}^{\mathrm{T}},$$

che implica

$$\underline{\underline{A}}^{\mathrm{T}}\underline{\underline{A}} = \underline{\underline{A}}^2.$$

In questa situazione, quindi,

$$\rho(\underline{\underline{A}}^{\mathrm{T}}\underline{\underline{A}}) = \rho(\underline{\underline{A}}^2),$$

ma, poiché gli autovalori di $\underline{\underline{A}}^2$ sono uguali agli autovalori di $\underline{\underline{A}}$, elevati al quadrato[9], ed essendo il raggio spettrale semplicemente il modulo di un autovalore,

$$\rho(\underline{\underline{A}}^2) = \left[\rho(\underline{\underline{A}})\right]^2,$$

e, quindi, se la matrice $\underline{\underline{A}}$ è simmetrica,

$$\left\|\underline{\underline{A}}\right\|_2 = \sqrt{\rho(\underline{\underline{A}}^{\mathrm{T}}\underline{\underline{A}})} = \sqrt{\rho(\underline{\underline{A}}^2)} = \sqrt{\left[\rho(\underline{\underline{A}})\right]^2} = \rho(\underline{\underline{A}}).$$

Tutto questo discorso ci ha permesso di definire la norma 2, o **norma spettrale** di una matrice; tuttavia, essa si può anche calcolare con MATLAB® usando il comando `norm(A,2)`, o anche semplicemente `norm(A)`: quando non si specifica quale norma si desidera, MATLAB® ritorna la norma 2.

4.1.5 Cerchi di Gershgorin

Il nostro obiettivo sarà calcolare, mediante algoritmi, gli autovalori di una matrice. Tuttavia, alcuni di questi algoritmi richiederanno una stima della posizione di questi autovalori, che poi verrà raffinata mediante un procedimento iterativo, quindi un procedimento *a più passi*. Uno strumento matematico che permette di determinare delle stime sono i **cerchi di Gershgorin**. Ne esistono di due tipi: **cerchi riga** e **cerchi colonna**. Partiamo dai primi: si può definire l'area $C_i^{(\mathrm{r})}$ identificata dall'espressione

$$C_i^{(\mathrm{r})} = \left\{ z \in \mathbb{C} \ : \ |z - a_{ii}| \leq \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}| \right\}, \ i = 1, \dots, n. \tag{4.7}$$

[8] la quale è sempre maggiore di zero

[9] lo abbiamo dimostrato usando la proprietà del panino (4.3)

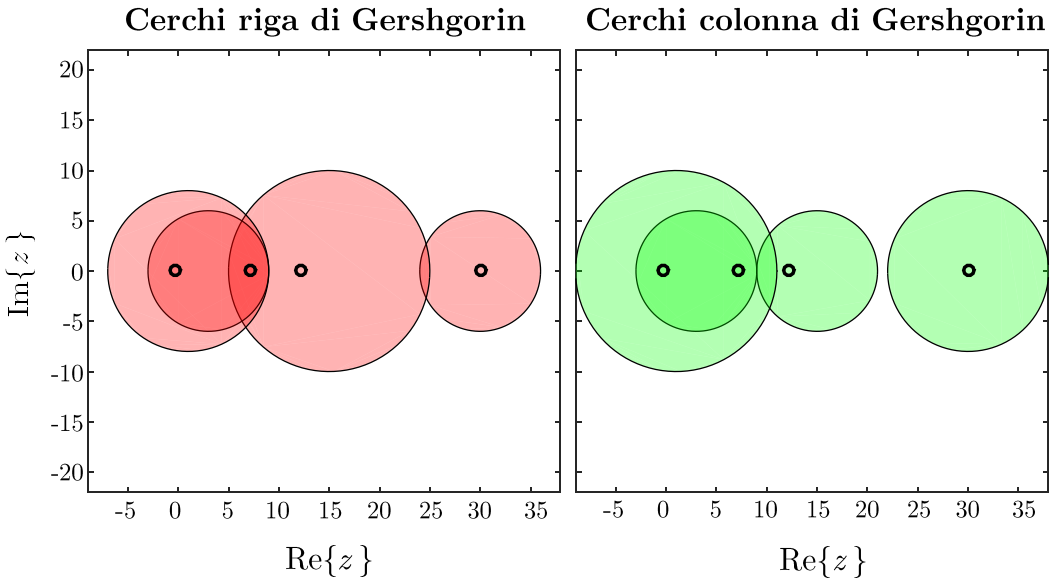

Figura 4.1: Esempio di cerchi riga (sinistra) e colonna (destra) di Gershgorin, ottenuti con il programma riportato in Appendice C.1.

Qua abbiamo bisogno di discutere un po'. Prima di tutto, notiamo che z è una variabile complessa, avente quindi parte immaginaria non necessariamente nulla; quando si parla di autovalori, spesso capita di avere a che fare con dei numeri complessi, dal momento che essi, come già detto, sono soluzioni di un'equazione algebrica le cui radici non sono necessariamente reali. Come dovrebbe essere noto da Analisi Matematica I: se un numero reale si può rappresentare geometricamente come un punto su una retta, allora un numero complesso si può rappresentare geometricamente come un punto su un piano, le cui ascisse e ordinate indicano la parte reale e immaginaria. L'espressione (4.8) contiene un'espressione tipo

$$|z - z_c| \leq r.$$

Questa identifica l'area di un cerchio nel piano complesso centrato in $z_c = a_{ii}$ (i-esimo elemento diagonale della matrice $\underline{\underline{A}}$) e avente raggio r pari alla somma dei moduli degli elementi sulla i-esima riga (escluso l'elemento sulla diagonale),

$$r = \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}|.$$

Si tratta di un cerchio perché è il luogo dei punti z nel piano complesso avente distanza dal punto z_c pari al raggio r; visto che prendiamo il minore-uguale $\leq$ non consideriamo solo la circonferenza, ma tutto l'insieme di punti.

Per i cerchi colonna, non c'è molto di più da dire:

$$C_j^{(c)} = \left\{ z \in \mathbb{C} \ : \ |z - a_{jj}| \leq \sum_{\substack{i=1 \\ i \neq j}}^{n} |a_{ij}| \right\}, \ j = 1, ..., n. \tag{4.8}$$

Ossia, il centro di questi cerchi è sempre l'elemento diagonale a_{jj}, ma questa volta i raggi di ciascuno dei cerchi si calcolano sommando gli elementi lungo le colonne, invece che lungo le righe, escludendo l'elemento sulla diagonale. Un esempio di questi cerchi è riportato in Fig. 4.1

Per una matrice $\underline{\underline{A}} \in \mathbb{R}^{n,n}$, avremo n cerchi riga e n cerchi colonna.

Il motivo per cui ci interessiamo di questi cerchi è il loro legame con gli autovalori della matrice dai quali sono stati ricavati. Definiamo infatti $\mathcal{R}$ e $\mathcal{C}$ le unioni dei cerchi di Gershgorin:

$$\mathcal{R} = \bigcup_{i=1}^{n} C_i^{(\mathrm{r})}, \qquad \mathcal{C} = \bigcup_{j=1}^{n} C_j^{(\mathrm{c})}.$$

Un teorema ci garantisce che **tutti gli autovalori della matrice $\underline{\underline{A}}$ appartengono sia a $\mathcal{R}$, sia a $\mathcal{C}$**. La cosa tutto sommato non dovrebbe sorprenderci troppo: abbiamo detto che questi cerchi sono centrati sugli elementi diagonali delle matrici, e che hanno un raggio pari alla somma dei moduli degli elementi fuori dalla diagonale. Immaginiamoci che gli elementi fuori dalla diagonale siano uguali a zero: questo vorrebbe dire che la matrice sarebbe diagonale, e che quindi i suoi elementi diagonali sarebbero esattamente gli autovalori, e dunque che il raggio sarebbe zero! D'altra parte, se gli elementi fuori dalla diagonale fossero non nulli, ma molto piccoli rispetto a quelli sulla diagonale, significherebbe avere raggi piccoli, e quindi questi cerchi darebbero un'idea abbastanza precisa di dove siano situati gli autovalori.

Non finisce qui: è anche possibile dire che **tutti gli autovalori della matrice $\underline{\underline{A}}$ appartengono all'intersezione delle regioni $\mathcal{C}$ e $\mathcal{R}$**; è dunque possibile delimitare ulteriormente la superficie in cui possiamo trovare questi autovalori! In Fig. 4.1, queste cose si possono vedere *a occhio*, dal momento che sono stati riportati gli autovalori mediante cerchietti neri.

Questi cerchi forniscono altre indicazioni: immaginiamo di non *unire* tutte le regioni, ma solamente k e le restanti $n-k$ in due diversi gruppi, in modo tale che questi due gruppi siano tra loro disgiunti. Nel caso di Fig. 4.1, cerchi colonna, potremmo per esempio usare come regione $\mathcal{R}_1$ i tre cerchi sulla sinistra, le cui intersezioni non sono nulle, e come regione $\mathcal{R}_2$ l'unico cerchio sulla destra. Beh, questa cosa è utile perché un teorema ci garantisce che

- dal momento che la regione $\mathcal{R}_1$ è costituita da 3 cerchi tra loro intersecati, essa conterrà 3 autovalori;
- dal momento che la regione $\mathcal{R}_2$ è costituita da 1 solo cerchio, essa conterrà 1 solo autovalore.

Insomma, il numero di autovalori contenuti in regioni diverse in quanto non sovrapposte, è pari al numero di cerchi costituiti da queste regioni. Questo vale sia per i cerchi riga, sia per i cerchi colonna.

4.1.6 Matrici simili

Una definizione di una relazione tra due matrici, assistita dai concetti di autovalori e autovettori, è quella di **matrici simili**. Due matrici $\underline{\underline{A}}$ e $\underline{\underline{B}}$, entrambe aventi n righe e n colonne, vengono dette simili se esiste una certa matrice $\underline{\underline{S}}$ avente n righe e n colonne, non singolare, tale per cui

$$\underline{\underline{S}}^{-1}\underline{\underline{A}}\,\underline{\underline{S}} = \underline{\underline{B}}.$$

Che questa definizione sia imparentata con autovalori e autovettori è abbastanza evidente, in quanto questa definizione ricorda terribilmente la *relazione panino* (4.3). Inoltre, si può dimostrare che due matrici simili $\underline{\underline{A}}$ e $\underline{\underline{B}}$ hanno gli stessi autovalori. Infatti, se

$$\underline{\underline{B}} = \underline{\underline{S}}^{-1}\underline{\underline{A}}\,\underline{\underline{S}},$$

e

$$\underline{\underline{A}}\,\underline{x} = \lambda\underline{x},$$

è possibile scrivere, moltiplicando da sinistra per $\underline{\underline{S}}^{-1}$,

$$\underline{\underline{S}}^{-1}\underline{\underline{A}}\,\underline{x} = \lambda\underline{\underline{S}}^{-1}\underline{x}.$$

Posso a questo punto applicare un trucchetto: posso far apparire il prodotto $\underline{\underline{S}}\,\underline{\underline{S}}^{-1}$ nell'espressione, cosa lecita (è un po' come moltiplicare e dividere per un numero):

$$\underline{\underline{S}}^{-1}\underline{\underline{A}}\,\underline{x} = \underline{\underline{S}}^{-1}\underline{\underline{A}}\,\underline{\underline{I}}\,\underline{x} = \underline{\underline{S}}^{-1}\underline{\underline{A}}\,\underline{\underline{S}}\,\underline{\underline{S}}^{-1}\underline{x} = \underbrace{\left[\underline{\underline{S}}^{-1}\underline{\underline{A}}\,\underline{\underline{S}}\right]}_{\underline{\underline{B}}}\underline{\underline{S}}^{-1}\underline{x}.$$

Qui, abbiamo potuto identificare la matrice $\underline{\underline{B}}$. Quindi, possiamo riscrivere questa espressione come

$$\underline{\underline{B}}\,\underbrace{\underline{\underline{S}}^{-1}\underline{x}}_{\underline{z}} = \lambda\,\underbrace{\underline{\underline{S}}^{-1}\underline{x}}_{\underline{z}},$$

dove stiamo suggerendo la definizione di un vettore

$$\underline{z} = \underline{\underline{S}}^{-1}\underline{x}.$$

Ma allora, è possibile scrivere l'ultima espressione come

$$\underline{\underline{B}}\,\underline{z} = \lambda\underline{z},$$

che ci dice che λ, autovalore della matrice $\underline{\underline{A}}$ per ipotesi, è anche un autovalore della matrice $\underline{\underline{B}}$, ma con un autovettore diverso: se l'autovettore corrispondente, per $\underline{\underline{A}}$, era $\underline{x}$, ora invece è questo $\underline{z}$.

Data dunque una matrice $\underline{\underline{A}}$, questa si dice **diagonalizzabile** se è simile (ovvero, se condivide gli stessi autovalori) a una matrice diagonale, contenente gli autovalori sulla diagonale:

$$\underline{\underline{S}}^{-1}\,\underline{\underline{A}}\,\underline{\underline{S}} = \underline{\underline{D}}.$$

Che, moltiplicando da sinistra per $\underline{\underline{S}}$, permette di ottenere

$$\underline{\underline{A}}\,\underline{\underline{S}} = \underline{\underline{S}}\,\underline{\underline{D}},$$

che è esattamente la *relazione del panino* (4.3), solo con nomi diversi.

Detto in un altro modo, una matrice $\underline{\underline{A}}$ di ordine n è diagonalizzabile se e solo se essa possiede n autovettori linearmente indipendenti. Infatti, se così non fosse, la matrice $\underline{\underline{S}}$ avrebbe rango non massimo, non sarebbe invertibile, e quindi non sarebbe possibile applicare la *relazione del panino*. Esiste tuttavia un risultato che garantisce che una matrice è diagonalizzabile se i suoi autovalori sono distinti; questa è una relazione solo in un verso (non c'è il *solo se*), nel senso che non è detto che, se gli autovalori non sono distinti[10], la matrice non sia diagonalizzabile.

4.1.7 Condizionamento del calcolo degli autovalori

Come esiste il problema del condizionamento nell'ambito della soluzione dei sistemi lineari, così esiste quello relativo al calcolo degli autovalori. Ricordiamo velocemente di cosa si parla: se gli elementi della matrice di partenza $\underline{\underline{A}}$ sono soggetti a una perturbazione, generata per esempio dagli errori di arrotondamento che commettiamo al momento di memorizzarla, di quanto variano gli autovalori, rispetto al caso che consideriamo *ideale*? La risposta è: possono anche variare un casino. Considerando per esempio la matrice

$$\underline{\underline{A}} = \begin{bmatrix} 101 & -90 \\ 110 & -98 \end{bmatrix},$$

e immaginiamo di perturbare alcuni suoi elementi:

$$\underline{\underline{\widetilde{A}}} = \begin{bmatrix} 101-\varepsilon & -90-\varepsilon \\ 110 & -98 \end{bmatrix}.$$

Vediamo cosa succede nel caso imperturbato e in quello perturbato, con una perturbazione $\varepsilon = 0.001$. Lanciando lo script

```
clear
close all
clc

epsilon=0.001; % perturbazione

% Inserisco la matrice esatta
A = [101  -90;
     110  -98];

% Inserisco la matrice perturbata
Atilde = [101-epsilon -90-epsilon;
                  110          -98];
```

[10] ovvero se alcuni di essi hanno molteplicità maggiore di 1

```
E=eig(A) % autovalori della matrice esatta
Etilde=eig(Atilde) % autovalori della matrice perturbata
norm(A-Atilde,2) % errore di perturbazione sulla matrice
errrel = abs(E-Etilde)./abs(E) % errore relativo sugli autovalori
```

si arriva a dimostrare che, con un errore in norma 2 tra le matrici pari allo 0.1%, si hanno errori del 30% sugli autovalori! Un autovalore, infatti, passa dall'essere pari a 2, all'essere 1.7: terrificante! E questo, con una banalissima matrice 2×2 !!!

È evidente che quindi anche il calcolo degli autovalori è soggetto al problema del condizionamento. Tuttavia, non è ancora chiaro dove questo nasca. A chiarirci le idee ci viene incontro il **teorema di Bauer-Fike**. Questo afferma che, data $\underline{\underline{A}}$ una matrice diagonalizzabile e $\underline{\underline{S}}$ invertibile tale per cui

$$\underline{\underline{S}}^{-1} \underline{\underline{A}}\, \underline{\underline{S}} = \underline{\underline{D}},$$

dove $\underline{\underline{D}}$ è la matrice diagonale avente gli autovalori sulla diagonale principale, allora

$$\min_{1 \le i \le n} \left| \widetilde{\lambda} - \lambda_i \right| \le \kappa(\underline{\underline{S}}) \left\| \underline{\underline{A}} - \widetilde{\underline{\underline{A}}} \right\|, \tag{4.9}$$

dove $\widetilde{\underline{\underline{A}}}$ è il risultato di una perturbazione sulla matrice $\underline{\underline{A}}$, $\widetilde{\lambda}$ è un certo autovalore di $\widetilde{\underline{\underline{A}}}$, e

$$\kappa(\underline{\underline{S}}) = \left\| \underline{\underline{S}} \right\| \left\| \underline{\underline{S}}^{-1} \right\|,$$

con i risultati validi per le norme 1, 2, ∞.

Cosa significa tutto questo? Prima di tutto è importante capire cosa voglia dire il membro sinistro, quel min. A una perturbazione sulla matrice corrisponde una perturbazione amplificata di $\kappa(\underline{\underline{S}})$ sugli autovalori; non è tuttavia scontato riuscire a *riconoscere* quale autovalore è diventato quale altro: se la perturbazione è grossa, potrebbe essere che non si riesca più a effettuare questo riconoscimento[11]. Il caso di prima dove passavamo da 2 a 1.7, per quanto traumatico, non era ingestibile. Visto che non sappiamo bene come un autovalore possa trasformarsi in un altro, tutto ciò che questo teorema può fare è usare questa minima differenza, sperando che l'autovalore trasformato nasca da quello *calcolato idealmente* a esso corrispondente.

Parlando del membro destro, sulla norma della differenza delle matrici, non c'è molto da dire: è una stima della perturbazione. Invece, $\kappa(\underline{\underline{S}})$ è molto interessante, perché assomiglia molto a quello dei sistemi lineari, ma applicato non a $\underline{\underline{A}}$, ma a $\underline{\underline{S}}$, ossia alla **matrice degli autovettori**! Se $\kappa(\underline{\underline{S}})$ è prossimo a 1, non avremo grosse perturbazioni, ma se fosse grande, avremmo un'enorme amplificazione della perturbazione sui dati.

Riassumendo: se la matrice degli autovettori $\underline{\underline{S}}$ è mal condizionata, ossia composta da autovettori che *non sono molto linearmente indipendenti*, allora il calcolo degli autovettori sarà mal condizionato. Voglio però rimarcare che il condizionamento del calcolo degli autovettori e della soluzione di un sistema lineare sono **due problemi del tutto disgiunti**. Un esempio eclatante è dato dalla matrice di Hilbert: come noto, la soluzione di sistemi lineari aventi la matrice di Hilbert come matrice di sistema è un problema mal condizionato. Il calcolo dei suoi autovalori, invece, è assolutamente ben condizionato! Infatti, la matrice di Hilbert è simmetrica; prima, però, abbiamo dimostrato che una matrice simmetrica ha autovettori ortogonali. E ortogonali, significa che sono **molto indipendenti linearmente**!

In generale, per una matrice $\underline{\underline{A}}$ simmetrica, dal momento che la sua matrice degli autovettori $\underline{\underline{S}}$ è ortogonale, si ha che il numero di condizionamento in norma 2 è pari a 1; infatti:

$$\kappa_2(\underline{\underline{S}}) = \left\| \underline{\underline{S}} \right\|_2 \left\| \underline{\underline{S}}^{\mathrm{T}} \right\|_2 = \sqrt{\rho(\underline{\underline{S}}^{\mathrm{T}} \underline{\underline{S}})} \sqrt{\rho(\underline{\underline{S}}\, \underline{\underline{S}}^{\mathrm{T}})} = \sqrt{\rho(\underline{\underline{I}})} \sqrt{\rho(\underline{\underline{I}})} = 1.$$

Oltre a calcolare il numero di condizionamento relativo alla soluzione di sistemi lineari, MATLAB® può calcolare anche il numero di condizionamento relativo alla determinazione dei suoi autovalori, usando il comando `condeig`.

Prima di concludere, può valere la pena di proporre la seguente riflessione sul teorema di Bauer-Fike e in particolare sull'espressione (4.9). Da un punto di vista concettuale, questa è assolutamente diversa da quanto si è visto nell'ambito dei sistemi lineari, in quanto il problema del condizionamento, in quel contesto, era stato applicato sull'errore **relativo** sui dati di ingresso, mentre qui abbiamo a che

[11] un check aggiuntivo si potrebbe ottenere guardando gli autovettori, ma non discutiamo queste cose che non servono a comprendere il discorso...

fare con errori **assoluti**. Tuttavia, al fine di *completare* questo ragionamento e ricercare un'analogia con il condizionamento di sistemi lineari, si tenga presente che vale la seguente relazione:

$$\min_{1\leq i\leq n}\frac{\left|\widetilde{\lambda}-\lambda_i\right|}{|\lambda_i|}\leq \kappa(\underline{\underline{S}})\left\|\underline{\underline{A}}^{-1}(\underline{\underline{A}}-\widetilde{\underline{\underline{A}}})\right\|,$$

dove al membro sinistro calcoliamo il minimo errore relativo dell'autovalore λ_i, mentre a membro destro introduciamo questa $\underline{\underline{A}}^{-1}$, che ci permette di definire, intuitivamente, una sorta di *errore relativo di matrici*.

4.2 Metodi numerici per il calcolo di autovalori e autovettori

I metodi numerici per il calcolo degli autovalori e degli autovettori di una matrice si possono classificare in due grandi famiglie, che si differenziano per il tipo di risultato desiderato. Queste sono:

- i metodi che permettono di calcolare solo uno degli autovalori della matrice;
- i metodi che permettono di calcolare simultaneamente tutti gli autovalori della matrice.

È utile poter disporre di entrambe queste categorie di metodi. Se infatti per esempio avessimo a che fare con matrici molto grandi e, per qualche ragione, ci interessassero solo uno o due autovalori, un algoritmo appartenente alla prima famiglia sarebbe consigliabile in quanto meno dispendioso.

Verranno ora presentati un metodo appartenente alla prima categoria (più una sua variante), e un metodo appartenente alla seconda. In entrambi i casi, i metodi si baseranno su algoritmi iterativi, ossia in cui si arriva a definire delle successioni numeriche convergenti agli autovalori di interesse. Per questo motivo, non sarà possibile sapere *a priori* quanti passi saranno necessari: dovremo costruire dei cicli in cui vengono effettuate le iterazioni e dovremo stabilire criteri di terminazione, ovvero *decidere quando interrompere il ciclo*.

4.2.1 Metodo delle potenze

Sia $\underline{\underline{A}}$ una matrice a n righe e colonne diagonalizzabile, e siano λ_k, per $k = 1, 2, ..., n$ i suoi autovalori. Il metodo che stiamo per descrivere, detto **metodo delle potenze**, permette di calcolare solo un autovalore: **quello avente il massimo modulo**. Nonostante esistano situazioni in cui potrebbe essere importante calcolare proprio l'autovalore di massimo modulo, in generale questa ipotesi è ovviamente limitante. Nella sezione successiva introdurremo però una variante di questo metodo, che ci permetterà di calcolare un generico autovalore a partire da un *guess*, ovvero da una stima iniziale.

Poiché ci interessa l'autovalore di massimo modulo, è conveniente *etichettare* i vari autovalori per ordine decrescente di modulo, ossia definire λ_1, λ_2, e così via, in modo che:

$$|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq ... |\lambda_{n-1}| \geq |\lambda_n|.$$

Detto a parole, il primo autovalore, λ_1, sarà quello di massimo modulo, ossia quello che valuteremo. Notiamo che la prima diseguaglianza è stretta: $|\lambda_2|$ deve essere strettamente minore di $|\lambda_1|$ affinché il metodo funzioni[12]. Data questa ipotesi, λ_1 è certamente reale: se infatti fosse complesso, allora anche il suo complesso coniugato sarebbe un autovalore, e quindi avremmo due autovalori di massimo modulo, andando contro l'ipotesi di partenza; quindi, questo metodo funziona solo con autovalori reali.

Dal momento che per ipotesi la matrice $\underline{\underline{A}}$ è diagonalizzabile, vale quanto ripreso nelle precedenti sezioni: la matrice dei suoi autovettori, $\underline{\underline{S}}$, è non singolare. In altre parole, lo spazio delle colonne di $\underline{\underline{S}}$ è una base per $\mathbb{R}^n$; detto ancora in altre parole, sommando le varie colonne di $\underline{\underline{S}}$ usando opportuni coefficienti moltiplicativi per ciascun vettore (ossia, prendendo una combinazione lineare dello spazio delle colonne), è possibile descrivere un qualsiasi vettore di n componenti, $\underline{z} \in \mathbb{R}^n$. Se invece $\underline{\underline{S}}$ fosse stata singolare, quindi di rango non massimo, non avremmo avuto la garanzia di poter scrivere un generico vettore $\underline{z}$ come combinazione lineare delle sue colonne. Quindi, data $\underline{x}_i$ la i-esima colonna di $\underline{\underline{S}}$, ovvero l'autovettore associato all'i-esimo autovalore λ_i, il generico vettore $\underline{z} \in \mathbb{R}^n$ si può scrivere come:

$$\underline{z} = \sum_{i=1}^{n} \alpha_i \underline{x}_i = \alpha_1 \underline{x}_1 + \alpha_2 \underline{x}_2 + ... + \alpha_n \underline{x}_n,$$

[12] in qualche modo MATLAB® usa degli *sporchi trucchi* che permettono di mitigare questo problema, ma che rischiano di danneggiare le proprietà fisico/matematiche dell'autovalore; al termine della sezione aggiungeremo qualche dettaglio extra su questa cosa

dove i vari α_i sono i coefficienti della combinazione lineare. Data dunque $\underline{\underline{A}}$ la matrice di partenza, è possibile calcolare $\underline{\underline{A}}\,\underline{z}$:

$$\underline{\underline{A}}\,\underline{z} = \underline{\underline{A}}\sum_{i=1}^{n}\alpha_i\underline{x}_i = \underline{\underline{A}}\left(\alpha_1\underline{x}_1 + ... + \alpha_n\underline{x}_n\right) = \alpha_1\underline{\underline{A}}\,\underline{x}_1 + ... + \alpha_n\underline{\underline{A}}\,\underline{x}_n = \sum_{i=1}^{n}\alpha_i\underline{\underline{A}}\,\underline{x}_i.$$

Questa proprietà deriva semplicemente dal fatto che stiamo parlando di applicazioni lineari, e quindi è possibile scambiare il simbolo di sommatoria con la moltiplicazione per la matrice. A questo punto, dal momento che gli $\{\underline{x}_i\}$ sono gli autovettori di $\underline{\underline{A}}$, è possibile usare il truccone di *togliere la matrice e mettere l'autovalore al suo posto*! Quindi, otteniamo

$$\sum_{i=1}^{n}\alpha_i\underline{\underline{A}}\,\underline{x}_i = \sum_{i=1}^{n}\alpha_i\lambda_i\underline{x}_i.$$

A questo punto, proviamo a porci una domanda diversa: immaginiamo di moltiplicare il vettore $\underline{z}$ di partenza invece che per $\underline{\underline{A}}$, per $\underline{\underline{A}}^2$; cosa succede? Vediamo:

$$\underline{\underline{A}}^2\underline{z} = \underline{\underline{A}}\left(\underline{\underline{A}}\,\underline{z}\right) = \underline{\underline{A}}\sum_{i=1}^{n}\alpha_i\lambda_i\underline{x}_i = \sum_{i=1}^{n}\alpha_i\lambda_i\underline{\underline{A}}\,\underline{x}_i = \sum_{i=1}^{n}\alpha_i\lambda_i\left(\lambda_i\underline{x}_i\right) = \sum_{i=1}^{n}\alpha_i\lambda_i^2\underline{x}_i.$$

Come si può immaginare, se invece di avere $\underline{\underline{A}}^2$ avessimo $\underline{\underline{A}}^m$, otterremmo:

$$\underline{\underline{A}}^m\underline{z} = \sum_{i=1}^{n}\alpha_i\lambda_i^m\underline{x}_i = \alpha_1\lambda_1^m\underline{x}_1 + \alpha_2\lambda_2^m\underline{x}_2 + ... + \alpha_n\lambda_n^m\underline{x}_n.$$

Adesso, applichiamo un altro truccone: raccogliamo λ_1^m da questa espressione:

$$\alpha_1\lambda_1^m\underline{x}_1 + \alpha_2\lambda_2^m\underline{x}_2 + ... + \alpha_n\lambda_n^m\underline{x}_n = \lambda_1^m\underbrace{\left[\alpha_1\underline{x}_1 + \left(\frac{\lambda_2}{\lambda_1}\right)^m\underline{x}_2 + ... + \left(\frac{\lambda_n}{\lambda_1}\right)^m\underline{x}_n\right]}_{\underline{y}^{(m)}} =$$

$$= \lambda_1^m\underline{y}^{(m)},$$

dove $\underline{y}^{(m)}$ è stato definito come l'intera parentesi quadra[13].

Siamo a questo punto pronti a trarre le conclusioni: a partire dalla nostra ipotesi $|\lambda_1| > |\lambda_i|$, con $i \geq 2$, i vari termini contenuti nelle parentesi quadre di $\underline{y}^{(m)}$ saranno tutti minori di 1:

$$\frac{\lambda_i}{\lambda_1} < 1.$$

Ma, quindi, al crescere di m, si ha che

$$\lim_{m\to\infty}\left(\frac{\lambda_i}{\lambda_1}\right)^m = 0,$$

e, quindi, si avrà che

$$\lim_{m\to\infty}\underline{\underline{A}}^m\,\underline{z} = \lim_{m\to\infty}\lambda_1^m\underline{y}^{(m)} = \lim_{m\to\infty}\lambda_1^m\left[\alpha_1\underline{x}_1 + \sum_{i=2}^{n}\left(\frac{\lambda_i}{\lambda_1}\right)^m\right] = \lambda_1^m\alpha_1\underline{x}_1.$$

In altre parole, tutti i contributi degli autovettori diversi dal primo tendono a sparire per $m \to \infty$.

Possiamo a questo punto sfruttare tutto ciò che abbiamo introdotto fino a ora per proporre un algoritmo iterativo in grado di calcolare l'autovalore di massimo modulo. Sostanzialmente, ogni iterazione consisterebbe nella moltiplicazione di un certo vettore per la matrice $\underline{\underline{A}}$, e dunque sarebbe un po' come se a ogni iterazione facessimo in modo da far crescere il m del $\underline{\underline{A}}^m$. Partiamo da un $\underline{z}$ generico[14], e a

[13] si noti che si è usato l'indice m tra parentesi tonde per indicare che questo è un m-esimo vettore $\underline{y}$, non un vettore elevato a potenza!

[14] c'è il rischio che questo $\underline{z}$ generico non contenga, tra i vari contributi, l'autovettore $\underline{x}_1$, ossia che, nello scrivere la combinazione lineare, si abbia il coefficiente $\alpha_1 = 0$; questo, è risaputo, non è un vero problema, dal momento che la presenza di errori di arrotondamento e fenomeni del genere, di solito problematici, fanno sì da far apparire una *piccola componente* dell'autovettore $\underline{x}_1$ che, quindi, verrà esaltata dal metodo delle potenze che privilegerà lei e sopprimerà tutte le altre!

forza di moltiplicarlo per $\underline{\underline{A}}$, il *metodo delle potenze* lo *ripulirà* di tutti i contributi degli autovettori che non siano quelli associati all'autovalore di massimo modulo[15]. Possiamo quindi riassumere l'algoritmo nei seguenti passi:

1. Dal momento che un autovettore è definito a meno di una costante moltiplicativa, ma anche che ogni iterazione del metodo delle potenze introduce un fattore moltiplicativo λ_1, al fine di evitare problemi di overflow o underflow è utile **normalizzare** il vettore su cui si lavora; il primo passo della m-esima iterazione dunque è definire il vettore su cui lavoriamo, $\underline{w}^{(m)}$, come

$$\underline{w}^{(m)} = \frac{\underline{z}^{(m)}}{\left\|\underline{z}^{(m)}\right\|_2}. \tag{4.10}$$

2. Dal momento che, al crescere di m, il vettore $\underline{w}^{(m)}$ tende proprio all'autovettore $\underline{x}_1$, ogni iterazione consiste nel definire una nuova stima dell'autovettore *più ripulita*, prendendo quello attuale e moltiplicandolo per la matrice $\underline{\underline{A}}$:

$$\underline{z}^{(m+1)} = \underline{\underline{A}}\,\underline{w}^{(m)}. \tag{4.11}$$

Come detto, questo è un metodo iterativo, quindi a priori non è chiaro quando ci si debba fermare. Si possono introdurre due criteri: prima di tutto, si può stabilire *a priori* di voler eseguire esattamente $m_{\max}$ iterazioni del metodo, sperando che esse siano sufficienti per avere una buona approssimazione dell'autovettore (e quindi dell'autovalore). Alternativamente, è possibile introdurre un **criterio di stop**, per esempio verificare quanto la stima dell'autovalore alla m-esima iterazione si discosti da quella alla $(m-1)$-esima iterazione: se al crescere di m questa differenza rimarrà al di sotto di una certa tolleranza, il metodo potrà essere fermato, in quanto non sta più effettuando *ripuliture*. Tuttavia, come appena descritto, il metodo delle potenze lavora sugli autovettori, dunque calcolare gli autovalori non è necessario, tant'è che né in (4.10) né in (4.11) appare λ_1. Volendo implementare un criterio di stop basato sull'autovalore, è possibile calcolare la sua stima alla m-esima iterazione[16], $\lambda_1^{(m)}$, utilizzando $\underline{w}^{(m)}$ e il quoziente di Rayleigh introdotto nelle precedenti sezioni:

$$\lambda_1^{(m)} = \frac{(\underline{w}^{(m)})^{\mathrm{T}}\underline{\underline{A}}\,\underline{w}^{(m)}}{(\underline{w}^{(m)})^{\mathrm{T}}\underline{w}^{(m)}} = (\underline{w}^{(m)})^{\mathrm{T}}\underbrace{\underline{\underline{A}}\,\underline{w}^{(m)}}_{\underline{z}^{(m+1)}},$$

dove il denominatore è pari a 1 in virtù del fatto che $\underline{w}^{(m)}$ viene normalizzato in (4.10). Questa espressione si può ancora modificare, come suggerito nell'espressione, sostituendo (4.11) e arrivando a trovare

$$\lambda_1^{(m)} = (\underline{w}^{(m)})^{\mathrm{T}}\underline{z}^{(m+1)}. \tag{4.12}$$

Segue un esempio di codice implementante il metodo delle potenze.

```
clear
close all
clc

m_max=100; % fisso un massimo numero di iterazioni
z=[1 2 3]'; % guess iniziale per l'autovettore
toll=1e-10; % tolleranza per il calcolo dell'autovalore; quando lo scarto
            % tra l'autovalore rispetto alla stima al passo precedente e`
            % inferiore alla tolleranza, si puo` bloccare il metodo

A=[1 2 0;1 0 0 ;0 1 0]; % matrice di cui calcolare l'autovalore di massimo
                        % modulo

Deltalambda = inf; % per far partire il ciclo, impongo che Deltalambda sia
                   % infinito; cosi`, faremo almeno 1 iterazione!

lambda = 0; % inizializzo lambda a 0 per la prima iterazione; 0 e` una
            % scelta arbitraria
```

[15] quanto veloce sia questo procedimento di *ripulitura*, è determinato da quanto grande è λ_2/λ_1; se infatti $|\lambda_2| \simeq |\lambda_1|$, allora la successione decade più lentamente!

[16] e ribadisco che la notazione indica che questo è l'autovalore alla m-esima iterazione, **non** che è elevato a potenza m, perché uso le parentesi tonde all'apice per distinguere; è solo una notazione!

```
m = 1; % inizializzo indice iterazioni

while m<=m_max & Deltalambda > toll

    w = z/norm(z,2); % passo 1: normalizzo l'autovettore al passo attuale
    z=A*w; % passo 2: moltiplico A per ottenere la nuova stima dell'autovettore
    lambda(m+1) = w'*z; % quoziente di Rayleigh in forma ridotta

    % Calcolo scarto rispetto a precedente stima
    Deltalambda = abs(lambda(m+1)-lambda(m))./abs(lambda(m+1));

    m = m+1; % incremento indice iterazione

end

lambda(end) % stampo su schermo l'autovalore appena trovato
eig(A) % stampo gli autovalori trovati con il comando MATLAB per verifica
```

Prima di concludere, è opportuno rettificare e/o completare alcune affermazioni.

- Se abbiamo più autovalori uguali tra loro, ossia $\lambda_1 = \lambda_2 = ... = \lambda_k$, non solo in modulo ma anche in segno, si può vedere che il metodo delle potenze riesce, molto faticosamente, a convergere.
- Se invece $\lambda_1 = -\lambda_2$, ossia se esistono due autovalori di uguale modulo massimo ma segno opposto, in generale il metodo delle potenze non converge.
- Se λ_1 e λ_2 sono complessi coniugati, il metodo descritto non converge, anche se è possibile in qualche modo modificarlo per mitigare il problema.
- Si può dimostrare che per una matrice simmetrica $\underline{\underline{A}}$ la convergenza è più rapida, poiché va come $(\lambda_2/\lambda_1)^{2m}$: il quadrato di quella per matrici non simmetriche.

4.2.2 Metodo delle potenze inverse

Il grosso limite del metodo delle potenze è il fatto che esso può calcolare solamente l'autovalore di massimo modulo. Tuttavia, è possibile modificare questo metodo in modo tale da fargli approssimare non l'autovalore di massimo modulo, ma un generico λ tale che sia il più vicino possibile a una certa stima p dataci in qualche modo[17].

Partiamo dalla solita equazione del problema agli autovalori

$$\underline{\underline{A}}\,\underline{x} = \lambda \underline{x}$$

che ci dice che $\underline{x}$ è l'autovettore della matrice $\underline{\underline{A}}$ associato all'autovalore λ. Immaginiamo che p sia una *stima* dell'autovalore λ, ossia un numero abbastanza simile a esso; è lecito sottrarre ad ambo i membri dell'equazione appena scritta $p\underline{x}$, ottenendo

$$\underline{\underline{A}}\,\underline{x} - p\underline{x} = \lambda \underline{x} - p\underline{x},$$

che si può riscrivere, raccogliendo, come

$$\left(\underline{\underline{A}} - p\underline{\underline{I}}\right)\underline{x} = (\lambda - p)\underline{x}.$$

Questa equazione ci dice che il numero $\lambda - p$ è autovalore della matrice $\underline{\underline{A}} - p\underline{\underline{I}}$. Dal momento che gli autovalori dell'inversa di una matrice sono uguali ai reciproci degli autovalori della matrice di partenza, possiamo dire che il numero μ definito come

$$\mu = \frac{1}{\lambda - p}$$

sia autovalore della matrice $(\underline{\underline{A}} - p\underline{\underline{I}})^{-1}$, peraltro sempre con lo stesso autovettore $\underline{x}$:

$$\left(\underline{\underline{A}} - p\underline{\underline{I}}\right)^{-1}\underline{x} = \frac{1}{\lambda - p}\underline{x} \triangleq \mu \underline{x}. \tag{4.13}$$

[17] per esempio dai cerchi di Gershgorin!

Tutte queste bellissime considerazioni sugli autovalori e autovettori non sono un tentativo di ammazzare il tempo, ma hanno uno scopo ben preciso. Se infatti la stima p è abbastanza buona, allora abbiamo che $\lambda - p$ è un numero piccolo, ma quindi

$$|\mu| = \left| \frac{1}{\lambda - p} \right|$$

sarà un numero grande, dal momento che sarà il reciproco di un numero piccolo. Nel dettaglio, se p sarà una stima abbastanza buona, potremo dire che il μ corrispondente, in convergenza, sarà **l'autovalore di massimo modulo della matrice** $(\underline{\underline{A}} - p\underline{\underline{I}})^{-1}$. Ma noi, guarda caso, abbiamo appena imparato un metodo che ci permette di calcolare l'autovalore di massimo modulo di una matrice: **il metodo delle potenze**! Se quindi, invece di applicare il metodo delle potenze alla matrice $\underline{\underline{A}}$, lo applicassimo alla matrice $(\underline{\underline{A}} - p\underline{\underline{I}})^{-1}$, potremmo trovare l'autovalore più vicino alla stima p di partenza. Infatti, a partire da (4.13), potremmo ottenere λ, a partire da μ, come

$$\mu = \frac{1}{\lambda - p} \Longrightarrow \mu(\lambda - p) = 1 \Longrightarrow \mu\lambda - \mu p = 1 \Longrightarrow \mu\lambda = 1 + \mu p,$$

che finalmente ci porta a

$$\lambda = p + \frac{1}{\mu}.$$

Proprio come fatto per il metodo delle potenze, possiamo sfruttare tutte queste considerazioni per scrivere un algoritmo.

1. Proprio come per il metodo delle potenze, per ragioni numeriche è conveniente normalizzare la stima dell'autovettore, lavorando quindi su un $\underline{w}^{(m)}$ normalizzato anziché su $\underline{z}^{(m)}$:

$$\underline{w}^{(m)} = \frac{\underline{z}^{(m)}}{\left\|\underline{z}^{(m)}\right\|_2}. \tag{4.14}$$

   Fin qui, nulla di nuovo.

2. Il metodo delle potenze inverse ha lo scopo di trovare l'autovalore μ di massimo modulo della matrice $(\underline{\underline{A}} - p\underline{\underline{I}})^{-1}$, anziché di trovare il λ_1 di $\underline{\underline{A}}$; di conseguenza, l'iterazione del metodo delle potenze inverse sarà identica alla (4.11), dove però al posto di $\underline{\underline{A}}$ dovremo sostituire $(\underline{\underline{A}} - p\underline{\underline{I}})^{-1}$:

$$\underline{z}^{(m+1)} = \left(\underline{\underline{A}} - p\underline{\underline{I}}\right)^{-1} \underline{w}^{(m)}.$$

   Tuttavia, per quanto questa espressione sia corretta, non è molto furba da un punto di vista computazionale; infatti, questa contiene l'inversa di una matrice :-(. Ciò che invece potremmo fare è moltiplicare da sinistra ambo i membri per $(\underline{\underline{A}} - p\underline{\underline{I}})$, ottenendo

$$\left(\underline{\underline{A}} - p\underline{\underline{I}}\right) \underline{z}^{(m+1)} = \underline{w}^{(m)}. \tag{4.15}$$

   Da un punto di vista teorico, le due espressioni sono esattamente coincidenti. Da un punto di vista pratico, però, questa è molto più interessante, dal momento che è un sistema lineare avente come soluzione la stima dell'autovettore per la successiva iterazione $\underline{z}^{(m+1)}$, avente come matrice di sistema $\left(\underline{\underline{A}} - p\underline{\underline{I}}\right)$ (che ha il vantaggio di non richiedere inversioni per essere valutata), e come termine noto la stima dell'autovettore alla precedente iterazione $\underline{w}^{(m)}$. Quindi, anziché invertire questa matrice, potremo calcolare la fattorizzazione PA = LU **una volta soltanto, come passo preliminare dell'algoritmo**, e **a ogni iterazione del metodo delle potenze inverse, sarà sufficiente risolvere due sistemi triangolari**.

Segue un esempio di codice implementante questo metodo.

```
clear
close all
clc
```

```
m_max=100; % fisso un massimo numero di iterazioni

toll=1e-10; % tolleranza per il calcolo dell'autovalore; quando lo scarto
            % tra l'autovalore rispetto alla stima al passo precedente e`
            % inferiore alla tolleranza, si puo` bloccare il metodo

A=[1 -2 0;0 2 0 ;1 1 3]; % matrice di cui calcolare l'autovalore piu`
                         % vicino alla stima p

z=[1 1 1]'; % guess iniziale per l'autovettore

p=0.5; % guess iniziale per l'autovalore che ci interessa approssimare

lambda=p; % ha senso inizializzare lambda, per la prima iterazione, come il
          % guess su di esso!

Deltalambda = inf; % per far partire il ciclo, impongo che Deltalambda sia
                   % infinito; cosi`, faremo almeno 1 iterazione!

I = eye(size(A)); % matrice identita` di dimensione pari a quella di A

[L,U,P] = lu(A-p*I); % calcolo, una volta soltanto, la fattorizzazione
                     % PA = LU della matrice

m = 1; % inizializzo indice iterazioni

while m<=m_max & Deltalambda > toll

    w = z/norm(z);   % passo 1: normalizzo l'autovettore al passo attuale
    z = U\(L\(P*w)); % passo 2: risolvo il sistema lineare (A-pI)*z = w ;
                     % grazie alla fattorizzazione LU, che ho calcolato
                     % solo una volta esternamente al ciclo, questo compito
                     % si riduce alla soluzione di due sistemi triangolari!

    % calcolo mu, il massimo autovalore di (A-p*eye(n))^(-1)
    mu=w'*z; % quoziente di Rayleigh per calcolo mu
    lambda(m+1) = p+1/mu; % formula di slide 40

    % Calcolo scarto rispetto a precedente stima
    Deltalambda = abs(lambda(m+1)-lambda(m))./abs(lambda(m+1));

    m = m+1; % incremento indice iterazione

end
m

lambda(end) % stampo su schermo l'autovalore appena trovato

eigs(A,1,p) % con il comando eigs posso calcolare solo pochi (o in questo
            % caso solo 1 autovalore) tale per cui esso sia il piu` vicino
            % alla stima iniziale p; uso questo comando per verifica
```

Seguono ora alcune osservazioni conclusive sul metodo delle potenze inverse.

- Se scegliamo come stima iniziale $p = 0$, allora il metodo delle potenze inverse permette di approssimare l'autovalore di minimo modulo della matrice $\underline{\underline{A}}$, nell'ipotesi che esista un solo autovalore reale di minimo modulo.
- Una variante a questo algoritmo potrebbe consistere, dopo un certo numero di iterazioni, nel sostituire la stima p con un'altra stima, più raffinata, in quanto ottenuta dalle prime iterazioni del metodo. Il vantaggio potrebbe essere che, con una stima più raffinata, magari il numero di iterazioni necessarie per convergere potrebbe diminuire. Tuttavia, la controindicazione nella cosa sarebbe la necessità di ricalcolare, una volta sostituita p, la fattorizzazione PA = LU, aumentando il costo computazionale.
- Nel caso un po' sfortunato in cui $p = \lambda$, allora c'è il rischio che il numero $(p - \lambda)^{-1}$ sia infinito, e questo potrebbe causare problemi; c'è pure da dire che, se anche capitasse una cosa del genere, vorrebbe dire che non c'è nessun autovalore da trovare: lo avremmo già!

4.2.3 Metodo QR

Presentiamo ora, in qualità di rappresentante della famiglia dei metodi atti ad approssimare simultaneamente tutti gli autovalori, il **metodo QR**, basato sulla omonima fattorizzazione di matrici. Come i metodi delle potenze, anche questa tecnica è iterativa.

Alla prima iterazione, definiamo

$$\underline{\underline{A}}_1 = \underline{\underline{A}}.$$

Quindi calcoliamo, da $\underline{\underline{A}}_1$, la sua fattorizzazione QR:

$$\underline{\underline{A}}_1 = \underline{\underline{Q}}_1 \underline{\underline{R}}_1,$$

dove ricordiamo che $\underline{\underline{Q}}_1$ è una matrice ortogonale e $\underline{\underline{R}}_1$ una matrice triangolare superiore (assumendo che $\underline{\underline{A}}$ non sia singolare). Il metodo si basa sul definire la matrice di partenza dell'iterazione successiva, $\underline{\underline{A}}_2$, come

$$\underline{\underline{A}}_2 = \underline{\underline{R}}_1 \underline{\underline{Q}}_1,$$

e, in generale,

$$\underline{\underline{A}}_{k+1} = \underline{\underline{R}}_k \underline{\underline{Q}}_k.$$

A questo punto, notiamo un trucchetto furbo; dal momento che $\underline{\underline{Q}}_1$ è ortogonale, abbiamo che

$$\underline{\underline{Q}}_1^{\mathrm{T}} \underline{\underline{Q}}_1 = \underline{\underline{I}}.$$

Questo significa che è concesso *far apparire* dove vogliamo il prodotto $\underline{\underline{Q}}_1^{\mathrm{T}} \underline{\underline{Q}}_1$. Per esempio, nella definizione di $\underline{\underline{A}}_2$;-)

$$\underline{\underline{A}}_2 = \underline{\underline{R}}_1 \underline{\underline{Q}}_1 = \underline{\underline{I}}\, \underline{\underline{R}}_1 \underline{\underline{Q}}_1 = \underline{\underline{Q}}_1^{\mathrm{T}} \underline{\underline{Q}}_1 \underline{\underline{R}}_1 \underline{\underline{Q}}_1.$$

Tuttavia, dal momento che

$$\underline{\underline{Q}}_1 \underline{\underline{R}}_1 = \underline{\underline{A}}_1,$$

questa scrittura ci dice che

$$\underline{\underline{A}}_2 = \underline{\underline{Q}}_1^{\mathrm{T}} \underline{\underline{A}}_1 \underline{\underline{Q}}_1.$$

Questo ci dimostra che $\underline{\underline{A}}_2$ è una matrice **simile** a $\underline{\underline{A}}_1$: infatti, esiste una matrice $\underline{\underline{Q}}_1$ non singolare[18] che permette di passare da $\underline{\underline{A}}_1$ a $\underline{\underline{A}}_2$; questo è quello che di solito viene definito un *cambio di base*. Essendo $\underline{\underline{A}}_2$ simile a $\underline{\underline{A}}_1$, essa ha i suoi stessi autovalori. In generale, ovviamente, $\underline{\underline{A}}_k$ sarà simile a $\underline{\underline{A}}_{k-1}$, e così via, ma quindi anche a $\underline{\underline{A}}_1$, ossia alla matrice di partenza $\underline{\underline{A}}$.

L'utilità di questo algoritmo risiede nel fatto che, iterando questo metodo, si arriva a una matrice $\underline{\underline{A}}_\infty$ che soddisfa le seguenti proprietà:

- se la matrice $\underline{\underline{A}}$ è simmetrica, $\underline{\underline{A}}_\infty$ è diagonale, e, quindi, gli elementi sulla sua diagonale sono gli autovalori della matrice $\underline{\underline{A}}$, grazie alla proprietà di similitudine appena ricordata;
- se la matrice $\underline{\underline{A}}$ non è simmetrica, ma ha autovalori reali, allora $\underline{\underline{A}}_\infty$ è triangolare superiore e, di nuovo, gli elementi sulla sua diagonale sono gli autovalori della matrice $\underline{\underline{A}}$;
- se $\underline{\underline{A}}$ non è simmetrica e ha alcuni autovalori complessi coniugati, $\underline{\underline{A}}_\infty$ è **quasi triangolare superiore**, ossia presenta lungo la diagonale sottomatrici 1×1 contenenti gli autovalori reali di $\underline{\underline{A}}$, e sottomatrici 2×2 i cui autovalori sono autovalori complessi coniugati di $\underline{\underline{A}}$.

Questo è grosso modo l'algoritmo su cui si basa il comando `eig` di MATLAB® ; nella forma qui presentata, la convergenza è garantita se gli autovalori della matrice sono distinti in modulo.

Segue infine uno script in cui viene implementato il metodo QR in questa versione di base.

```
clear
close all
clc

k_max=100; % fisso un massimo numero di iterazioni
toll=1e-14; % tolleranza per il calcolo dell'autovalore; quando lo scarto
            % tra l'autovalore rispetto alla stima al passo precedente e`
```

[18] in questo caso addirittura ortogonale!

```
            % inferiore alla tolleranza, si puo` bloccare il metodo

n=10; % come esempio usiamo una matrice n x n di Hilbert
A=hilb(n); % matrice di Hilbert

% Inizializzo alcune variabili
err=inf; % errore tra gli autovalori al passo precedente e attuale
Eold=inf; % vettore degli autovalori al passo precedente
k=0; % numero di iterazioni del metodo

while(k<k_max & err>toll)

    [Q,R]=qr(A); % calcolo fattorizzazione QR
    A=R*Q; % matrice alla iterazione successiva

    E=diag(A); % la matrice A diventera` diagonale o triangolare; di
               % conseguenza, i suoi autovalori diventeranno gli elementi
               % sulla diagonale

    err=norm(E-Eold,2); % calcolo errore per esempio in norma 2 rispetto
                        % agli autovalori stimati alla iterazione precedente

    Eold=E; % aggiorno il vettore degli autovalori rispetto al passo
            % precedente

    k=k+1; % aggiorno il contatore delle iterazioni

end

k % stampo sul prompt il numero di iterazioni
[Sref,Eref]=eig(A); % calcolo autovalori e autovettori "di riferimento"

figure
hold on
grid on
box on
plot(real(E),imag(E),'bo') % disegno gli autovalori ottenuti in parte reale
                           % e in parte immaginaria
plot(real(diag(Eref)),imag(diag(Eref)),'r*') % autovalori di riferimento
xlabel('Re(\lambda)')
ylabel('Im(\lambda)')
title('Autovalori matrice')
legend('Metodo QR base','Funzione eig')
axis equal
axis square
```

4.3 Decomposizione ai valori singolari

Moltissimi dei concetti introdotti precedentemente hanno senso solamente quando si ha a che fare con matrici quadrate: si pensi per esempio agli autovalori, al determinante o all'inversa di una matrice. Tuttavia, alcuni di questi concetti possono essere in qualche modo estesi anche a matrici rettangolari $\underline{\underline{A}} \in \mathbb{R}^{m,n}$ aventi m righe e n colonne. Al fine di comprendere queste estensioni, cercheremo in qualche misura di appoggiarci al concetto di autovalore e sfruttando le proprietà di particolari matrici simmetriche definite positive. Prima di tutto, quindi arriveremo a *costruire* lo strumento matematico dietro a queste estensioni, ovvero la **decomposizione ai valori singolari**[19]; poi, proporremo una tecnica, per quanto rudimentale, in grado di **calcolarla**. Infine, studieremo le sue **applicazioni**.

Prima di proseguire con questo programma è doveroso citare la fonte che ha ispirato questa trattazione: il magistrale corso *Linear Algebra* tenuto nel 2010 dal prof. Gilbert Strang del Massachussets Institute of Technology[20]. Per dare un'idea di che personaggio sia, il prof. Strang è il primo *MathWorks® Professor of Mathematics*, dove, sì, MathWorks® è proprio **quella** MathWorks®: quella che produce MATLAB® .

4.3.1 Costruzione della decomposizione ai valori singolari

Data una generica matrice $\underline{\underline{A}} \in \mathbb{R}^{m,n}$, da un punto di vista algebrico si può dire che $\underline{\underline{A}}$ sia un'applicazione lineare che trasforma vettori appartenenti al suo spazio delle righe in vettori appartenenti allo spazio delle colonne. Prendiamo per esempio la matrice

[19] per gli amici SVD, acronimo dell'inglese **singular value decomposition**

[20] il materiale, comprensivo di videoregistrazioni, è disponibile all'indirizzo http://ocw.mit.edu/courses/mathematics/18-06-linear-algebra-spring-2010

$$\underline{\underline{A}} = \begin{bmatrix} 1 & 1 & 2 \\ 2 & 1 & 3 \\ 3 & 1 & 4 \\ 4 & 1 & 5 \end{bmatrix}$$

e il vettore $\underline{x} = [0 \quad -1 \quad 1]^{\mathrm{T}}$. Si può vedere chiaramente che questo appartiene allo spazio delle righe, secondo questa definizione; infatti, il loro prodotto

$$\underline{\underline{A}}\,\underline{x} = \begin{bmatrix} 1 & 1 & 2 \\ 2 & 1 & 3 \\ 3 & 1 & 4 \\ 4 & 1 & 5 \end{bmatrix} \begin{bmatrix} 0 \\ -1 \\ 1 \end{bmatrix} = 0 \times \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} - 1 \times \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} + 1 \times \begin{bmatrix} 2 \\ 3 \\ 4 \\ 5 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

appartiene allo spazio delle colonne[21] di $\underline{\underline{A}}$: è un elemento dell'insieme delle possibili combinazioni lineari delle colonne di $\underline{\underline{A}}$. Questa cosa è particolarmente evidente in questo caso, visto che $\underline{\underline{A}}\,\underline{x}$ è non solo una combinazione lineare, ma addirittura una delle colonne della matrice ;-). In quest'ottica, il prodotto righe per colonne si può considerare come la scelta di una particolare combinazione lineare dei vettori colonna, dove i coefficienti sono le componenti del vettore $\underline{x}$. Riassumendo, $\underline{x}$ appartiene allo spazio delle righe[22] perché la combinazione lineare che genera al momento di effettuare il prodotto matriciale non è il vettore nullo.

D'altro canto, tutti quei vettori $\underline{x}$ i cui elementi rappresentano i coefficienti delle combinazioni lineari non banali che portano a ottenere il vettore nullo, saranno elementi appartenenti al nucleo di $\underline{\underline{A}}$. Se non è zuppa, è pan bagnato: un vettore $\underline{x} \in \mathbb{R}^n$, in relazione alle righe di una matrice $\underline{\underline{A}} \in \mathbb{R}^{m,n}$, può appartenere o al suo spazio delle righe, o al suo nucleo. Quindi: se moltiplichiamo per esempio la nostra $\underline{\underline{A}}$ per il vettore $\underline{y} = [1 \quad 1 \quad -1]^{\mathrm{T}}$, otteniamo

$$\underline{\underline{A}}\,\underline{y} = \begin{bmatrix} 1 & 1 & 2 \\ 2 & 1 & 3 \\ 3 & 1 & 4 \\ 4 & 1 & 5 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} = 1 \times \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} + 1 \times \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} - 1 \times \begin{bmatrix} 2 \\ 3 \\ 4 \\ 5 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix},$$

e quindi scopriamo che $\underline{y}$ appartiene al kernel di $\underline{\underline{A}}$, in quanto la moltiplicazione ha mappato $\underline{y}$ nel vettore nullo, nonostante $\underline{y}$ non fosse identicamente nullo.

Riassumendo: una matrice $\underline{\underline{A}} \in \mathbb{R}^{m,n}$ si può vedere come costituita da m vettori riga messi uno sopra l'altro. Le combinazioni lineari di questi vettori riga costituiscono lo spazio delle righe, il quale a sua volta è un sottospazio di $\mathbb{R}^n$, dal momento che ciascuna riga è un vettore dotato di n componenti. Ma quindi, a partire da questi vettori, è certamente lecito applicare l'algoritmo di Gram-Schmidt al fine di ottenere una base ortonormale dello spazio delle righe; questa sarà costituita da $r \leq m$ elementi, che chiamiamo $\{\underline{v}_i\}$, per $i = 1, 2, \ldots, r$. Qui, r è il rango della matrice $\underline{\underline{A}}$: come noto dall'Algebra Lineare. Infatti, il rango coincide con la dimensione dello spazio delle righe, ossia con il minimo numero di vettori linearmente indipendenti le cui combinazioni lineari lo possono generare. Il nucleo di $\underline{\underline{A}}$, anche detto *kernel*, è il complementare a $\mathbb{R}^n$ di questo spazio, e avrà quindi dimensione pari a $n - r$.

Abbiamo parlato tanto delle righe, ma adesso concentriamoci per un momento sulle colonne: se io moltiplico la matrice $\underline{\underline{A}}$ per ciascuno dei vettori della base ortonormale appena introdotta, ottengo, per ciascuno di essi, un vettore $\underline{\underline{A}}\,\underline{v}_i$ che certamente apparterrà allo spazio delle colonne. Tuttavia, da un punto di vista geometrico, moltiplicare la matrice $\underline{\underline{A}}$ per un generico vettore $\underline{v}_i$ provoca una rotazione e/o una dilatazione di quest'ultimo; di conseguenza, nonostante i nostri vettori $\{\underline{v}_i\}$ siano tra loro ortogonali per ipotesi/costruzione, non abbiamo alcuna garanzia in merito al fatto che gli $\{\underline{\underline{A}}\,\underline{v}_i\}$ lo siano: la matrice $\underline{\underline{A}}$ può ruotare e/o dilatare in modo diverso ciascun vettore[23]. Fatta questa introduzione, ora ci poniamo il seguente ambizioso proposito: vogliamo trovare una base dello spazio delle righe $\{\underline{v}_i\}$ tale per cui la applicazione di $\underline{\underline{A}}$ su diversi vettori $\underline{v}_i$ generi vettori colonna $\underline{u}_i$ tra loro ortogonali. Ossia, vogliamo una particolare base ortonormale dello spazio delle righe in cui ciascun elemento si mappa in un elemento di una base ortonormale dello spazio delle colonne! Quindi, in matematichese, vogliamo una cosa del tipo

[21] che si può vedere un po' come l'immagine della applicazione lineare associata alla matrice!

[22] che quindi si può un po' vedere come la controimmagine della applicazione lineare associata alla matrice!

[23] si pensi, per esempio, a un vettore generico, o a un autovettore: un vettore generico subirà una certa rotazione, l'autovettore non ne subirà alcuna!

$$\begin{aligned}\underline{\underline{A}}\begin{bmatrix}\underline{v}_1 & \underline{v}_2 & \underline{v}_3 & \cdots & \underline{v}_r\end{bmatrix} &= \begin{bmatrix}\underline{\underline{A}}\underline{v}_1 & \underline{\underline{A}}\underline{v}_2 & \underline{\underline{A}}\underline{v}_3 & \cdots & \underline{\underline{A}}\underline{v}_r\end{bmatrix} = \\ &= \begin{bmatrix}s_1\underline{u}_1 & s_2\underline{u}_2 & s_3\underline{u}_3 & \cdots & s_r\underline{u}_r\end{bmatrix}.\end{aligned} \tag{4.16}$$

L'equazione matriciale (4.16) si può scrivere più compattamente come

$$\underline{\underline{A}}\,\underline{\underline{V}}^{(r)} = \underline{\underline{U}}^{(r)}\underline{\underline{S}}^{(r)}, \tag{4.17}$$

a patto di definire le tre matrici

$$\underline{\underline{V}}^{(r)} = \begin{bmatrix}\underline{v}_1 & \underline{v}_2 & \underline{v}_3 & \cdots & \underline{v}_r\end{bmatrix},$$

$$\underline{\underline{U}}^{(r)} = \begin{bmatrix}\underline{u}_1 & \underline{u}_2 & \underline{u}_3 & \cdots & \underline{u}_r\end{bmatrix},$$

e $\underline{\underline{S}}^{(r)}$ come una matrice diagonale $r \times r$ avente come elementi gli s_i. L'equazione (4.17) ricorda vagamente le espressioni legate alla decomposizione agli autovalori, anche alla luce del fatto che $\underline{\underline{S}}^{(r)}$ è una matrice diagonale; tuttavia, qui la matrice $\underline{\underline{A}}$ è rettangolare, e le matrici $\underline{\underline{V}}^{(r)}$ e $\underline{\underline{U}}^{(r)}$ sono diverse tra loro, quindi sembra che si stia ottenendo una qualche generalizzazione o estensione del concetto di diagonalizzazione di una matrice. Le matrici $\underline{\underline{V}}^{(r)}$ e $\underline{\underline{U}}^{(r)}$, per costruzione, sono certamente ortogonali: le loro r colonne sono vettori per ipotesi ortogonali! Inoltre, dal momento che ciascuna delle colonne di $\underline{\underline{V}}^{(r)}$ deriva dall'applicazione di Gram-Schmidt sui vettori riga di $\underline{\underline{A}}$, allora, sempre per costruzione, $\underline{\underline{V}}^{(r)}$ ha n righe; dualmente, $\underline{\underline{U}}^{(r)}$ avrà m righe. Quindi, $\underline{\underline{V}}^{(r)} \in \mathbb{R}^{n,r}$, mentre $\underline{\underline{U}} \in \mathbb{R}^{m,r}$.

Possiamo fare di più, ed estendere ulteriormente queste definizioni arrivando a ottenere matrici $\underline{\underline{V}}$ e $\underline{\underline{U}}$ quadrate, di dimensione $n \times n$ e $m \times m$, rispettivamente. Per fare questo, è possibile aggiungere a $\underline{\underline{V}}^{(r)}$ e $\underline{\underline{U}}^{(r)}$ nuove colonne, ortogonali alle precedenti, ma quindi facenti parte dei complementi ortogonali a $\mathbb{R}^n$ e $\mathbb{R}^m$ degli spazi delle righe e delle colonne, rispettivamente. Per capire meglio questa cosa, pensiamo allo spazio delle righe e a $\mathbb{R}^n$: se la matrice $\underline{\underline{A}}$ ha rango r non massimo, significa che lo spazio delle righe è un sottoinsieme strettamente incluso in $\mathbb{R}^n$, avente dimensione r. Quindi, in aggiunta ai r vettori $\underline{v}_i$, $i = 1, ..., r$ definiti come la base ortonormale dello spazio delle righe, è possibile definire altri $n - r$ vettori ortogonali tra di loro e ai precedenti, che però genereranno il complementare dello spazio delle righe: lo spazio nullo della matrice! Si tratterà quindi di vettori aggiuntivi $\underline{v}_i$ tali per cui

$$\underline{\underline{A}}\,\underline{v}_i = \underline{0}, \quad i = r+1, ..., n.$$

Con questa idea applicata su $\underline{\underline{V}}^{(r)}$ e dualmente su $\underline{\underline{U}}^{(r)}$, la (4.16) può essere estesa includendo, come anticipato, anche i vettori appartenenti ai complementi a $\mathbb{R}^n$ e $\mathbb{R}^m$, arrivando a scrivere

$$\underline{\underline{A}}\underbrace{\begin{bmatrix}\underline{v}_1 & \underline{v}_2 & \cdots & \underline{v}_r & \underline{v}_{r+1} & \cdots & \underline{v}_n\end{bmatrix}}_{\underline{\underline{V}}} = \underbrace{\begin{bmatrix}s_1\underline{u}_1 & s_2\underline{u}_2 & \cdots & s_r\underline{u}_r & s_{r+1}\underline{u}_{r+1} & \cdots & s_m\underline{u}_m\end{bmatrix}}_{\underline{\underline{U}}\,\underline{\underline{S}}}. \tag{4.18}$$

Dal momento che

$$\underline{\underline{A}}\,\underline{v}_{r+1} = \underline{\underline{A}}\,\underline{v}_{r+2} = ... = \underline{\underline{A}}\,\underline{v}_n = s_{r+1}\underline{u}_{r+1} + s_{r+2}\underline{u}_{r+2} + ... + s_m\underline{u}_m = \underline{0},$$

e poiché non siamo interessati a soluzioni banali[24], vogliamo che tutti i $\underline{u}_i$ siano diversi dal vettore identicamente nullo, quindi a essere nulli saranno necessariamente i coefficienti s_i:

$$s_i = 0, \quad i > r.$$

Riassumiamo il contenuto di (4.18).

- Indicati in blu troviamo gli elementi della base ortonormale dello spazio delle righe della matrice $\underline{\underline{A}}$.
- Indicati in rosso troviamo gli elementi della base ortonormale del complemento a $\mathbb{R}^n$ dello spazio delle righe della matrice $\underline{\underline{A}}$; di conseguenza, questa sarà una base ortonormale del nucleo della matrice $\underline{\underline{A}}$.

[24] dove per *banali* intendiamo combinazioni lineari che vanno a zero perché costruite con vettori nulli

- Indicati in ciano troviamo gli elementi della base ortonormale dello spazio delle colonne, ovvero dell'immagine, della matrice $\underline{\underline{A}}$.
- Indicati in magenta troviamo gli elementi della base ortonormale del complemento a $\mathbb{R}^m$ dello spazio delle colonne della matrice $\underline{\underline{A}}$.

La (4.18) si può scrivere in modo sintetico come

$$\underline{\underline{A}}\,\underline{\underline{V}} = \underline{\underline{U}}\,\underline{\underline{S}}, \tag{4.19}$$

dove $\underline{\underline{V}} \in \mathbb{R}^{n,n}$, $\underline{\underline{U}} \in \mathbb{R}^{m,m}$, e $\underline{\underline{S}} \in \mathbb{R}^{m,n}$ è una matrice rettangolare che ha elementi diversi da zero solo sulla diagonale della matrice quadrata di dimensione massima che si può ritagliare da essa. Gli elementi di $\underline{\underline{S}}$ vengono chiamati **valori singolari**, le colonne di $\underline{\underline{U}}$ vengono dette **vettori singolari sinistri**, e le colonne di $\underline{\underline{V}}$ vengono dette **vettori singolari destri**. L'espressione (4.19) si può ancora elaborare moltiplicando ambo i membri per $\underline{\underline{V}}^{-1}$, arrivando a una forma *tipo panino*

$$\underline{\underline{A}} = \underline{\underline{U}}\,\underline{\underline{S}}\,\underline{\underline{V}}^{-1}$$

ma, sapendo che $\underline{\underline{V}}$ è una matrice ortogonale per costruzione, $\underline{\underline{V}}^{-1} = \underline{\underline{V}}^{\mathrm{T}}$ e, quindi, si arriva alla forma che si trova usualmente in letteratura:

$$\underline{\underline{A}} = \underline{\underline{U}}\,\underline{\underline{S}}\,\underline{\underline{V}}^{\mathrm{T}}. \tag{4.20}$$

Questa è la cosiddetta **decomposizione ai valori singolari**, o **singular value decomposition** (SVD), della matrice $\underline{\underline{A}}$.

4.3.2 Calcolo della decomposizione ai valori singolari

Fino a questo momento siamo stati molto astratti: abbiamo **costruito** la decomposizione ai valori singolari di una matrice, ovvero, abbiamo *detto come ci piacerebbe che fosse fatta*: abbiamo capito che è una specie di decomposizione dello spazio delle righe e dello spazio delle colonne della matrice, ma non abbiamo alcuna idea di come, operativamente, si possano calcolare le matrici $\underline{\underline{U}}$, $\underline{\underline{S}}$ e $\underline{\underline{V}}$.

Per calcolarle, cercheremo di sfruttare qualcosa che conosciamo già molto bene; infatti, ripartendo da (4.20), proviamo a studiare il prodotto $\underline{\underline{A}}^{\mathrm{T}}\underline{\underline{A}}$:

$$\underline{\underline{A}}^{\mathrm{T}}\underline{\underline{A}} = \left(\underline{\underline{U}}\,\underline{\underline{S}}\,\underline{\underline{V}}^{\mathrm{T}}\right)^{\mathrm{T}}\left(\underline{\underline{U}}\,\underline{\underline{S}}\,\underline{\underline{V}}^{\mathrm{T}}\right) = \underline{\underline{V}}\,\underline{\underline{S}}^{\mathrm{T}}\underline{\underline{U}}^{\mathrm{T}}\underline{\underline{U}}\,\underline{\underline{S}}\,\underline{\underline{V}}^{\mathrm{T}}.$$

Dal momento che la matrice $\underline{\underline{U}}$ è ortogonale, questa espressione si può semplificare; definendo

$$\underline{\underline{S}}^{\mathrm{T}}\underline{\underline{S}} = \underline{\underline{S}}^{2},$$

vediamo che essa sarà diagonale, avrà dimensioni $n \times n$, e i suoi elementi saranno i quadrati dei primi n valori singolari. Dunque, abbiamo

$$\underline{\underline{A}}^{\mathrm{T}}\underline{\underline{A}} = \underline{\underline{V}}\,\underline{\underline{S}}^{2}\underline{\underline{V}}^{\mathrm{T}}.$$

Dal momento che la matrice $\underline{\underline{V}}$ è per ipotesi ortogonale, questa espressione si può riscrivere ulteriormente come

$$\underline{\underline{A}}^{\mathrm{T}}\underline{\underline{A}} = \underline{\underline{V}}\,\underline{\underline{S}}^{2}\underline{\underline{V}}^{-1},$$

che è **esattamente** la decomposizione agli autovalori della matrice $\underline{\underline{A}}^{\mathrm{T}}\underline{\underline{A}}$. Riassumendo, siamo partiti da una matrice $\underline{\underline{A}}$ che può essere rettangolare, poi considerando $\underline{\underline{A}}^{\mathrm{T}}\underline{\underline{A}}$, ne abbiamo ottenuta una quadrata e, mettendo in relazione quest'ultima con la decomposizione ai valori singolari della matrice $\underline{\underline{A}}$ (4.20), abbiamo scoperto che possiamo operativamente calcolare i valori singolari come le radici quadrate degli autovalori di $\underline{\underline{A}}^{\mathrm{T}}\underline{\underline{A}}$, e la matrice $\underline{\underline{V}}$ come la corrispondente matrice degli autovettori.

Come si può immaginare, se calcoliamo $\underline{\underline{A}}\,\underline{\underline{A}}^{\mathrm{T}}$, otteniamo

$$\underline{\underline{A}}\,\underline{\underline{A}}^{\mathrm{T}} = \left(\underline{\underline{U}}\,\underline{\underline{S}}\,\underline{\underline{V}}^{\mathrm{T}}\right)\left(\underline{\underline{U}}\,\underline{\underline{S}}\,\underline{\underline{V}}^{\mathrm{T}}\right)^{\mathrm{T}} = \underline{\underline{U}}\,\underline{\underline{S}}\,\underline{\underline{V}}^{\mathrm{T}}\underline{\underline{V}}\,\underline{\underline{S}}^{\mathrm{T}}\underline{\underline{U}}^{\mathrm{T}} = \underline{\underline{U}}\underline{\underline{S}}^{2}\underline{\underline{U}}^{\mathrm{T}},$$

dove in questo caso $\underline{\underline{S}}^2$ ha dimensioni $m \times m$. Nel caso avessimo $\underline{\underline{A}} \in \mathbb{R}^{m,n}$ con $m > n$ (più righe che colonne), gli ultimi $m - n$ elementi diagonali di $\underline{\underline{S}}^2$ ottenuta da $\underline{\underline{A}}\,\underline{\underline{A}}^{\mathrm{T}}$ sarebbero uguali a zero, e viceversa per il caso $n > m$ e la $\underline{\underline{S}}^2$ ottenuta da $\underline{\underline{A}}^{\mathrm{T}}\underline{\underline{A}}$. Le colonne di $\underline{\underline{U}}$ sono dunque gli autovettori della matrice $\underline{\underline{A}}\,\underline{\underline{A}}^{\mathrm{T}}$, e, proprio come prima, i valori singolari sono le radici quadrate dei suoi autovalori.

Esempio di calcolo della decomposizione ai valori singolari

Una volta trovato un collegamento tra la SVD di una matrice e qualcosa che sappiamo gestire, ovvero gli autovalori, può essere utile svolgere un esercizietto che ci permetta di fissare ulteriormente i concetti. Dunque, calcoliamo la decomposizione ai valori singolari della matrice

$$\underline{\underline{A}} = \begin{bmatrix} 4 & 3 \\ 8 & 6 \end{bmatrix},$$

e discutiamo i risultati.

Prima di tutto, partiamo dal calcolo della matrice $\underline{\underline{V}}$; per ottenerla, usiamo la procedura appena mostrata:

$$\underline{\underline{A}}^{\mathrm{T}}\underline{\underline{A}} \triangleq \underline{\underline{B}} = \begin{bmatrix} 4 & 8 \\ 3 & 6 \end{bmatrix}\begin{bmatrix} 4 & 3 \\ 8 & 6 \end{bmatrix} = \begin{bmatrix} 4\times 4 + 8\times 8 & 4\times 3 + 8\times 6 \\ 3\times 4 + 6\times 8 & 3\times 3 + 6\times 6 \end{bmatrix} = \begin{bmatrix} 80 & 60 \\ 60 & 45 \end{bmatrix}.$$

«Oh! Ma che sorpresa! È simmetrica! È un caso?»

Ma no: è **sempre** vero che $\underline{\underline{A}}^{\mathrm{T}}\underline{\underline{A}}$ è simmetrica (e definita positiva)!

A parte questo: abbiamo visto che i valori singolari sono uguali alle radici quadrate degli autovalori di questa matrice; per una matrice 2×2, l'equazione caratteristica si può scrivere nella forma traccia-determinante:

$$s^4 - \mathrm{Tr}\{\underline{\underline{B}}\}s^2 + \det\{\underline{\underline{B}}\} = 0,$$

che è biquadratica perché andiamo a cercare direttamente i valori singolari s, piuttosto che gli autovalori λ di $\underline{\underline{B}}$; allora, facendo i conticini,

$$\mathrm{Tr}\{\underline{\underline{B}}\} = 80 + 45 = 125,$$

e

$$\det\{\underline{\underline{B}}\} = 80 \times 45 - 60 \times 60 = 0.$$

L'equazione caratteristica quindi è

$$s^4 - 125s^2 = 0,$$

che ha come radici

$$s_1 = \sqrt{125}$$
$$s_2 = 0.$$

È usanza e convenzione diffusa **e da tener presente** ordinare i valori singolari dal più grande al più piccolo. Abbiamo un valore singolare nullo; siamo sorpresi? Beh, no: se avessimo calcolato il determinante della matrice, ci saremmo accorti subito che $\underline{\underline{A}}$ non è di rango massimo; in effetti, la seconda riga è un multiplo della prima ;-).

Calcoliamo ora, con un po' di furbizia, i vettori singolari destri $\underline{v}_i$. A questo fine, partiamo da $\underline{v}_2$, ossia dal vettore singolare associato al valore $s_2 = 0$; questo richiede di risolvere il sistema omogeneo

$$\begin{bmatrix} 80 & 60 \\ 60 & 45 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

Poiché una riga o l'altra sono la stessa cosa, possiamo vedere, dalla prima, che

$$80x_1 + 60x_2 = 0,$$

che ci dice che

$$x_1 = -\frac{6}{8}x_2.$$

Quindi, prendiamo per esempio il vettore

$$\underline{v}_2 = \begin{bmatrix} -\frac{6}{8} \\ 1 \end{bmatrix},$$

e normalizziamolo[25], ottenendo:

$$\|\underline{v}_2\|_2 = \sqrt{\frac{36}{64} + 1} = \sqrt{\frac{100}{64}} = \frac{10}{8},$$

e, infine,

$$\frac{\underline{v}_2}{\|\underline{v}_2\|_2} = \begin{bmatrix} -\frac{6}{10} \\ \frac{8}{10} \end{bmatrix} \triangleq \underline{v}_2$$

(ridefiniamo $\underline{v}_2$ come quello normalizzato). **Dal momento che $\underline{v}_2$ è associato al valore singolare $s_2 = 0$, questo vettore singolare destro costituisce una base dello spazio nullo di $\underline{\underline{A}}$.** Infatti, se noi calcolassimo

$$\underline{\underline{A}} \sum_{i=r+1}^{n} \alpha_i \underline{v}_i = \sum_{i=r+1}^{n} \alpha_i \underline{\underline{A}}\,\underline{v}_i = \sum_{i=r+1}^{m} \alpha_i s_i \underline{u}_i,$$

dove r è il rango della matrice, otterremmo sempre 0! Si noti che gli estremi delle sommatorie sono stati scambiati da n a m, perché i valori singolari dopo il r-esimo, come si è detto prima, sono uguali a zero ma le matrici possono avere dimensione diversa.

Questa è una delle applicazioni più interessanti della SVD: determinare la base dello spazio nullo di una matrice! Questa cosa dimostra semplicemente che la nostra costruzione della precedente sezione è andata a buon fine! ;-)

Dal momento che stiamo ragionando su $\underline{\underline{B}} = \underline{\underline{A}}^{\mathrm{T}}\underline{\underline{A}}$, che è simmetrica, i suoi autovettori saranno ortogonali; quindi, invece di rifare tutti questi brutti conti per trovare $\underline{v}_1$, sarà sufficiente dire che $\underline{v}_1$ è l'unico vettore ortogonale a $\underline{v}_2$ di norma pari a 1 e, quindi, per ispezione,

$$\underline{v}_1 = \begin{bmatrix} \frac{8}{10} \\ \frac{6}{10} \end{bmatrix}.$$

Cerchiamo a questo punto di calcolare i vettori singolari sinistri di $\underline{\underline{A}}$. Partiamo quindi dalla matrice

$$\underline{\underline{A}}\,\underline{\underline{A}}^{\mathrm{T}} \triangleq \underline{\underline{C}} = \begin{bmatrix} 4 & 3 \\ 8 & 6 \end{bmatrix}\begin{bmatrix} 4 & 8 \\ 3 & 6 \end{bmatrix} = \begin{bmatrix} 25 & 50 \\ 50 & 100 \end{bmatrix}.$$

A questo punto, se procedessimo beceramente, dovremmo calcolarci gli autovalori. Ma visto che noi siamo giovani astuti e svelti, possiamo ricordarci che gli autovalori sono gli stessi che abbiamo calcolato prima: $s_1^2 = 125$, $s_2^2 = 0$; purtroppo, gli autovettori invece saranno diversi. Interessiamoci per ora solo all'autovettore associato a s_1, e calcoliamolo impostando il sistema lineare omogeneo associato:

$$\begin{bmatrix} 25 - s_1^2 & 50 \\ 50 & 100 - s_1^2 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} -100 & 50 \\ 50 & -25 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

Prendendo per esempio la seconda equazione,

[25] si noti che MATLAB® normalizza sia gli autovettori sia i vettori singolari rispetto alla norma 2!

$$50x_1 - 25x_2 = 0 \Longrightarrow x_2 = 2x_1,$$

e, quindi, possiamo definire il vettore singolare sinistro $\underline{u}_1$ come

$$\underline{u}_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}.$$

Normalizziamolo:

$$\frac{\underline{u}_1}{\|\underline{u}_1\|_2} = \frac{1}{\sqrt{5}} \begin{bmatrix} 1 \\ 2 \end{bmatrix} \triangleq \underline{u}_1,$$

dove quindi abbiamo ridefinito $\underline{u}_1$ come quello normalizzato.

Al fine di comprendere il significato di un vettore singolare sinistro, che ricordiamo essere, per come abbiamo costruito la SVD, un elemento della base dello spazio delle colonne, proviamo a calcolare, per un certo numero di vettori $\underline{x}$ presi a caso,

$$\underline{y} = \underline{\underline{A}}\,\underline{x}.$$

Possiamo per esempio prendere $\underline{x} = [1 \;\; 2]^{\mathrm{T}}$:

$$\begin{bmatrix} 4 & 3 \\ 8 & 6 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 10 \\ 20 \end{bmatrix}.$$

Oppure, possiamo provare con $\underline{x} = [4 \;\; 1]^{\mathrm{T}}$:

$$\begin{bmatrix} 4 & 3 \\ 8 & 6 \end{bmatrix} \begin{bmatrix} 4 \\ 1 \end{bmatrix} = \begin{bmatrix} 19 \\ 38 \end{bmatrix}.$$

«Ma, ma, porca miseria! È sempre un multiplo di $\underline{u}_1$! Ma perché?»

Beh, perché la matrice è 2×2 e lo spazio delle colonne, ovvero l'immagine, ha dimensione 1 (perché solo un valore singolare è non nullo) e, quindi, il generico vettore $\underline{x}$ si può scrivere come

$$\underline{x} = \alpha_1 \underline{v}_1 + \alpha_2 \underline{v}_2,$$

ma, dal momento che $\underline{v}_2 \in \ker\{\underline{\underline{A}}\}$ per quanto abbiamo detto precedentemente, si ha che

$$\underline{\underline{A}}\,\underline{x} = \underline{\underline{A}}\,(\alpha_1 \underline{v}_1 + \alpha_2 \underline{v}_2) = \alpha_1 \underline{\underline{A}}\,\underline{v}_1 + \alpha_2 \underline{\underline{A}}\,\underline{v}_2 = \alpha_1 \underline{\underline{A}}\,\underline{v}_1 = \alpha_1 s_1 \underline{u}_1,$$

che guarda caso è quello che ci viene numericamente. Questo ci dimostra che, proprio come volevamo dalla nostra costruzione, i $\{\underline{u}_i\}$, per $i = 1, \ldots, r$, costituiscono una base dell'immagine: moltiplicando la matrice $\underline{\underline{A}}$ per un generico vettore $\underline{x}$, il risultato sarà sempre una combinazione lineare dei vettori singolari sinistri associati a valori singolari non nulli!

Questo, è la SVD: una decomposizione in grado di fornire da un lato una base dell'immagine di un'applicazione lineare, ossia del suo spazio delle colonne, e dall'altro una base del suo nucleo. Per di più, grazie alla nostra costruzione, queste basi sono pure ortonormali, che è il meglio che si possa avere.

Prima di concludere, una domanda di carattere *numerico*.

«Il calcolo dei valori singolari è affetto da problemi di condizionamento?»

E, signore e signori, la risposta è **no**! Un'intuizione dietro questa buonissima notizia deriva dall'interpretazione dei valori e vettori singolari come autovalori e autovettori di matrici simmetriche che, certamente, sono molto ben condizionati!

Si noti che $\underline{\underline{U}}$ e $\underline{\underline{V}}$ non sono definite univocamente, dal momento che dipendono dalle scelte che effettuiamo come basi degli spazi delle righe e delle colonne; tuttavia, questo non introduce particolari problemi, da un punto di vista dell'accuratezza.

MATLAB® permette di calcolare la decomposizione ai valori singolari mediante il comando `svd`, usato come segue:

```
[U,S,V] = svd(A);
```

il quale ritorna le matrici $\underline{\underline{U}}$, $\underline{\underline{S}}$, e $\underline{\underline{V}}$.

Queste note non volevano spiegare come MATLAB® calcoli la SVD, ma esclusivamente *introdurre per via costruttiva le sue proprietà*. Non è detto che MATLAB® calcoli la SVD risolvendo due problemi agli autovalori; quel metodo, tuttavia, è fondamentale al fine di fornire interpretazioni forti dei vari attori coinvolti.

Autovalori e valori singolari di matrici simmetriche

Per concludere questa sezione introduttiva alla decomposizione ai valori singolari, si vuole presentare una situazione particolare, in cui autovalori e valori singolari sono fortemente imparentati. In effetti, se abbiamo a che fare con una matrice $\underline{\underline{A}} \in \mathbb{R}^{n,n}$ (dunque quadrata), simmetrica, allora **i suoi valori singolari sono uguali ai moduli dei suoi autovalori**.

Volendo per questa volta evitare una dimostrazione rigorosa, un approccio alternativo per verificare questa cosa potrebbe essere lavorare su una qualche matrice simmetrica e verificarlo numericamente, con MATLAB® . Prima di tutto, costruiamoci una matrice simmetrica

```
>> A = triu(magic(5)); % prima definisco la matrice A come la triangolare superiore
                       % di una matrice piu' o meno a caso (uno puo' provare con rand())
A = A + A'             % e poi calcolo la A come se' stessa piu' la trasposta: questa
                       % certamente sara' simmetrica ;-)
A =
    34    24     1     8    15
    24    10     7    14    16
     1     7    26    20    22
     8    14    20    42     3
    15    16    22     3    18
>>
```

A questo punto, possiamo calcolare gli autovalori e i valori singolari della matrice:

```
>> eig(A)' % metto il trasposto per visualizzarli agevolmente

ans =
  -10.1227   -4.1531   26.1668   39.5516   78.5573

>> svd(A)' % anche qui

ans =
   78.5573   39.5516   26.1668   10.1227    4.1531
```

Mirabilmente, è evidente che a ogni autovalore (perlomeno, al suo valore assoluto), corrisponde un valore singolare: la nostra proprietà è verificata!

Esiste un corollario a questa proprietà: se la matrice, oltre a essere simmetrica, è definita positiva, allora gli autovalori coincidono con i valori singolari, senza bisogno di considerarne i moduli. Alla luce della proprietà precedente, questo corollario dovrebbe essere molto ragionevole, se ricordiamo che gli autovalori di una matrice simmetrica definita positiva sono tutti positivi ;-)

4.3.3 Applicazioni della decomposizione ai valori singolari

Dopo aver introdotto la decomposizione ai valori singolari, in modo spero non troppo traumatico, è finalmente ora di vedere in che contesti essa potrà essere applicata.

Approssimazione di matrici

La prima, interessantissima applicazione della decomposizione ai valori singolari è la **approssimazione di matrici**. Infatti, sfruttando il fatto che, data una matrice di rango r, i primi[26] r vettori singolari sinistri $\{\underline{u}_i\}$ e destri $\{\underline{v}_i\}$ costituiscono rispettivamente una base dell'immagine e dello spazio delle righe, è

[26] ovvero associati ai valori singolari più grandi

possibile costruire una matrice che in qualche senso *approssimi* quella di partenza. Sostanzialmente, invece di prendere le basi nella loro interezza, ne prendiamo solo alcuni elementi. Per portare a termine questo compito possiamo appoggiarci al seguente teorema: si considerino i valori singolari **in ordine decrescente**:

$$s_1 \geq s_2 \geq \ldots \geq s_r \geq s_{r+1} \geq \ldots$$

e consideriamo al più i primi r: **quelli diversi da zero**. Se definiamo la matrice $\underline{\underline{A}}_k$ come

$$\underline{\underline{A}}_k = \sum_{i=1}^{k} s_i \underline{u}_i \underline{v}_i^{\mathrm{T}} \tag{4.21}$$

dove k è un numero intero minore del rango, allora, dato

$$\mathcal{B}_k = \left\{ \underline{\underline{B}} \ : \ \text{rango di } \underline{\underline{B}} \text{ uguale a } k \right\}$$

che, a parole, è l'insieme di tutte le possibili matrici $\underline{\underline{B}}$ aventi rango pari a k, si ha che $\underline{\underline{A}}_k$ è la matrice tale per cui

$$\min_{\underline{\underline{B}} \in \mathcal{B}_k} \left\| \underline{\underline{A}} - \underline{\underline{B}} \right\|_2 = \left\| \underline{\underline{A}} - \underline{\underline{A}}_k \right\|_2 = s_{k+1}.$$

Traduciamo dal matematichese: se definiamo la *distanza di due matrici* come la norma della loro differenza[27], la matrice $\underline{\underline{A}}_k$ è, di tutte le matrici $\underline{\underline{B}} \in \mathcal{B}_k$, quindi aventi rango k, quella meno distante da $\underline{\underline{A}}$, ossia quella che la approssima meglio. Inoltre, è possibile stimare la *distanza* tra $\underline{\underline{A}}$ e $\underline{\underline{A}}_k$, come il valore singolare $(k+1)$-esimo della matrice $\underline{\underline{A}}$. Di conseguenza, la $\underline{\underline{A}}_k$ è la miglior approssimazione in norma 2 di $\underline{\underline{A}}$ con una matrice di rango k.

Operativamente, implementare la definizione (4.21) è semplicissimo: è sufficiente *ritagliare* k colonne dalle matrici $\underline{\underline{U}}$, $\underline{\underline{S}}$, $\underline{\underline{V}}$, e quindi moltiplicare queste matrici; questo si può per esempio effettuare con il seguente codice MATLAB® :

```
[U,S,V] = svd(A); % calcolo decomposizione valori singolari
Uk = U(:,1:k); % ritaglio k colonne matrice dei vettori singolari sinistri
Vk = V(:,1:k); % ritaglio k colonne matrice dei vettori singolari destri
Sk = S(1:k,1:k); % ritaglio il blocco k x k della matrice dei valori singolari
Ak = Uk*Sk*Vk'; % ri-moltiplico le matrici
distanza = S(k+1,k+1) % calcolo distanza come valore singolare (k+1)-esimo
```

Questa idea è alla base di strepitose applicazioni, tra cui per esempio la compressione di immagini, come mostrato nel programmino presentato in Appendice C.2.

Calcolo del rango di una matrice

Il rango di una matrice $\underline{\underline{A}}$ coincide con il numero dei suoi valori singolari non nulli. Questo dovrebbe essere chiaro dalle precedenti sezioni, in cui abbiamo *costruito* la decomposizione ai valori singolari in modo tale che $s_i = 0$ per $i > r$. Tuttavia, potrebbe sorgere la seguente domanda:

> «Ma cosa vuol dire *nulli*? Cioè, proprio zero zero, o qualcosa di diverso?»

In effetti, la cosa più sana è definire il cosiddetto *rango numerico*: una volta fissata una certa tolleranza `toll`, è possibile definire il rango numerico come il numero di valori singolari più grandi di essa. Per esempio, se per qualche motivo fissassimo una tolleranza pari a 10^{-10}, avremmo

$$\{s_i\} = \{\underbrace{1, 0.5, 0.1, 10^{-6}}_{>10^{-10}}, \underbrace{10^{-11}, 0, 0}_{<10^{-10}}\},$$

ossia 4 valori singolari maggiori di 10^{-10}. Quindi, il rango numerico della matrice sarebbe 4. Si può calcolare il rango di una matrice `A` mediante il comando

```
r = rank(A);
```

o, volendo il rango numerico, data tolleranza `toll`,

```
r = rank(A,toll);
```

[27] in questo caso ci interessiamo della norma spettrale

Calcolo del condizionamento spettrale di una matrice

Data $\underline{\underline{A}}$ una matrice di ordine n e rango n, quindi massimo, ricordiamo che il numero di condizionamento spettrale, ovvero il numero di condizionamento in norma 2, è

$$\kappa_2(\underline{\underline{A}}) = \left\|\underline{\underline{A}}\right\|_2 \left\|\underline{\underline{A}}^{-1}\right\|_2.$$

Ragionando sul primo termine, abbiamo visto che la norma spettrale della matrice è imparentata al raggio spettrale:

$$\left\|\underline{\underline{A}}\right\|_2 = \sqrt{\rho(\underline{\underline{A}}^{\mathrm{T}}\underline{\underline{A}})}.$$

Possiamo a questo punto sostituire in questa espressione la decomposizione ai valori singolari della matrice $\underline{\underline{A}}$, ovvero

$$\underline{\underline{A}} = \underline{\underline{U}}\,\underline{\underline{S}}\,\underline{\underline{V}}^{\mathrm{T}},$$

ottenendo

$$\left\|\underline{\underline{A}}\right\|_2 = \sqrt{\rho((\underline{\underline{U}}\,\underline{\underline{S}}\,\underline{\underline{V}}^{\mathrm{T}})^{\mathrm{T}}(\underline{\underline{U}}\,\underline{\underline{S}}\,\underline{\underline{V}}^{\mathrm{T}}))} = \sqrt{\rho(\underline{\underline{V}}\,\underline{\underline{S}}^{\mathrm{T}}\underline{\underline{U}}^{\mathrm{T}}\underline{\underline{U}}\,\underline{\underline{S}}\,\underline{\underline{V}}^{\mathrm{T}})} = \sqrt{\rho(\underline{\underline{V}}\,\underline{\underline{S}}^2\underline{\underline{V}}^{\mathrm{T}})}.$$

Tuttavia, $\rho(\underline{\underline{V}}\,\underline{\underline{S}}^2\underline{\underline{V}}^{\mathrm{T}})$ è semplicemente il massimo autovalore della matrice $\underline{\underline{V}}\,\underline{\underline{S}}^2\underline{\underline{V}}^{\mathrm{T}}$ che, quindi, interpretando questo prodotto come una decomposizione agli autovalori, è

$$\left\|\underline{\underline{A}}\right\|_2 = \sqrt{\rho(\underline{\underline{S}}^2)} = \rho(\underline{\underline{S}}) = s_1,$$

ossia il tutto si riduce a essere s_1: il più grande dei valori singolari!

Con una dimostrazione molto simile si può arrivare a dimostrare che

$$\left\|\underline{\underline{A}}^{-1}\right\|_2 = \frac{1}{s_n},$$

dove s_n è il n-esimo valore singolare, dato n il numero di righe e colonne della matrice in questione. Quindi, riassumendo,

$$\kappa_2(\underline{\underline{A}}) = \frac{s_1}{s_n}.$$

Risoluzione di un sistema lineare determinato

Come quasi tutte le fattorizzazioni che abbiamo studiato finora, anche la SVD permette di risolvere un sistema lineare determinato nella forma

$$\underline{\underline{A}}\,\underline{x} = \underline{b},$$

dove dunque $\underline{\underline{A}}$ ha dimensione $n \times n$ e non è singolare. Applicando la decomposizione ai valori singolari, l'espressione appena riportata diventa

$$\underline{\underline{U}}\,\underline{\underline{S}}\,\underline{\underline{V}}^{\mathrm{T}}\underline{x} = \underline{b}.$$

Ricordando che $\underline{\underline{U}}$ è ortogonale, si ha che $\underline{\underline{U}}^{-1} = \underline{\underline{U}}^{\mathrm{T}}$, e quindi il sistema diventa

$$\underline{\underline{S}}\,\underline{\underline{V}}^{\mathrm{T}}\underline{x} = \underline{\underline{U}}^{\mathrm{T}}\underline{b}.$$

Definiamo a questo punto un vettore $\underline{y}$ come

$$\underline{y} = \underline{\underline{V}}^{\mathrm{T}}\underline{x},$$

che ci permette di riscrivere il sistema come

$$\underline{\underline{S}}\,\underline{y} = \underline{\underline{U}}^{\mathrm{T}}\underline{b}.$$

Notare che il sistema appena scritto è **diagonale**, e quindi la sua soluzione è assolutamente elementare! Risolto quindi il sistema, e trovato $\underline{y}$ dividendo ciascuna componente del vettore $\underline{\underline{U}}^{\mathrm{T}}\underline{b}$ per il corrispettivo elemento diagonale di $\underline{\underline{S}}$, si deve risolvere il sistema

$$\underline{\underline{V}}^{\mathrm{T}}\underline{x} = \underline{y}$$

dove, sfruttando l'ortogonalità di $\underline{\underline{V}}$, è sufficiente portarla a secondo membro trasponendola: $\underline{\underline{V}}^{\mathrm{T}} = \underline{\underline{V}}^{-1}$:

$$\underline{x} = \underline{\underline{V}}\,\underline{y}.$$

Questo metodo quindi non richiede la soluzione di alcun sistema triangolare, e da questo punto di vista sembrerebbe conveniente. Tuttavia, il costo necessario per produrre la SVD è talmente alto che rende tutto questo assolutamente sconsigliabile: si arriva a $32n^3/3$ operazioni richieste!

Per completare il discorso, bisognerebbe però spezzare una lancia a favore dell'uso della SVD per la soluzione di alcuni sistemi lineari determinati, perché esiste una situazione in cui è decisamente consigliata: quando il sistema è mal condizionato e prossimo a essere singolare, la SVD è nettamente più efficace al fine di risolverlo e, quindi, può valere la pena di pagare con l'elevatissimo numero di operazioni una maggiore stabilità numerica. Infatti, è possibile usare la SVD per identificare i valori singolari più piccoli di una prefissata tolleranza, approssimare la matrice $\underline{\underline{A}}$ con una di rango inferiore e, infine, risolvere il risultante problema nel senso dei minimi quadrati seguendo le indicazioni che verranno spiegate nella prossima sezione.

Risoluzione di un sistema lineare sovradeterminato

Non è la prima volta che trattiamo sistemi lineari sovradeterminati, ovvero aventi più equazioni che incognite. Tuttavia, quando lo abbiamo fatto usando la fattorizzazione QR, avevamo risolto il problema per matrici $\underline{\underline{A}}$ aventi rango massimo.

Senza voler ripassare troppo, vorrei ricordare che la soluzione $\underline{x}^*$ di un sistema lineare sovradimensionato non va intesa *in senso classico*, *esatto*, ma nel senso dei minimi quadrati, ovvero

$$\left\|\underline{\underline{A}}\,\underline{x}^* - \underline{b}\right\|_2 = \min_{\underline{y}\in\mathbb{R}^n} \left\|\underline{\underline{A}}\,\underline{y} - \underline{b}\right\|_2,$$

dove $\underline{x}^*$ è la soluzione che otteniamo dopo aver provato tutti i $\underline{y}$ possibili e avendo trovato quello che minimizza la norma del residuo. Similmente a quanto fatto in precedenza, cerchiamo di riscrivere la norma in modo conveniente. Ricordiamoci in particolare la proprietà

$$||\underline{x}||_2^2 = \left\|\underline{\underline{U}}^{\mathrm{T}}\underline{x}\right\|_2^2,$$

ovvero il fatto che, in una norma 2, è possibile far apparire o sparire liberamente una matrice ortogonale $\underline{\underline{U}}^{\mathrm{T}}$. Allora, facciamolo:

$$\left\|\underline{\underline{A}}\,\underline{y} - \underline{b}\right\|_2^2 = \left\|\underline{\underline{U}}^{\mathrm{T}}(\underline{\underline{A}}\,\underline{y} - \underline{b})\right\|_2^2 = \left\|\underline{\underline{U}}^{\mathrm{T}}\underline{\underline{A}}\,\underline{y} - \underline{\underline{U}}^{\mathrm{T}}\underline{b}\right\|_2^2,$$

dove è ovvio che questa $\underline{\underline{U}}^{\mathrm{T}}$ è proprio la trasposta della matrice $\underline{\underline{U}}$ della fattorizzazione SVD di $\underline{\underline{A}}$. A questo punto usiamo l'altro dei nostri trucchi preferiti, ovvero introdurre dove ci fa comodo una matrice identità $\underline{\underline{I}}$, definita come $\underline{\underline{I}} = \underline{\underline{V}}\,\underline{\underline{V}}^{\mathrm{T}}$; questo permette all'espressione di diventare

$$\left\|\underline{\underline{U}}^{\mathrm{T}}\underline{\underline{A}}\,\underline{y} - \underline{\underline{U}}^{\mathrm{T}}\underline{b}\right\|_2^2 = \left\|\underline{\underline{U}}^{\mathrm{T}}\underline{\underline{A}}\,\underline{\underline{V}}\,\underline{\underline{V}}^{\mathrm{T}}\,\underline{y} - \underline{\underline{U}}^{\mathrm{T}}\underline{b}\right\|_2^2,$$

dove guarda caso anche $\underline{\underline{V}}$ è proprio quella della SVD ;-). Dalla ormai ben nota

$$\underline{\underline{A}} = \underline{\underline{U}}\,\underline{\underline{S}}\,\underline{\underline{V}}^{\mathrm{T}},$$

è possibile moltiplicare da destra per $\underline{\underline{V}}$, da sinistra per $\underline{\underline{U}}^{\mathrm{T}}$, sfruttare l'ortogonalità e ottenere

$$\underline{\underline{S}} = \underline{\underline{U}}^{\mathrm{T}}\underline{\underline{A}}\,\underline{\underline{V}},$$

che, guarda di nuovo caso, è proprio quello che è apparso nella norma! Quindi, possiamo riscrivere l'ultimo termine come

$$\left\| \underline{\underline{U}}^{\mathrm{T}} \underline{\underline{A}}\, \underline{\underline{V}}\, \underline{\underline{V}}^{\mathrm{T}} \underline{y} - \underline{\underline{U}}^{\mathrm{T}} \underline{b} \right\|_2^2 = \left\| \underline{\underline{S}}\, \underline{\underline{V}}^{\mathrm{T}} \underline{y} - \underline{\underline{U}}^{\mathrm{T}} \underline{b} \right\|_2^2.$$

Se a questo punto definiamo un vettore

$$\underline{z} = \underline{\underline{V}}^{\mathrm{T}} \underline{y},$$

e

$$\underline{c} = \underline{\underline{U}}^{\mathrm{T}} \underline{b}, \tag{4.22}$$

arriviamo a ottenere

$$\left\| \underline{\underline{A}}\, \underline{y} - \underline{b} \right\|_2^2 = \left\| \underline{\underline{S}}\, \underline{z} - \underline{c} \right\|_2^2,$$

un po' come eravamo riusciti a fare con la fattorizzazione QR.

Continuando a imitare le mosse fatte allora, partiamo dal decomporre i vettori $\underline{z}$ e $\underline{c}$ in due parti:

$$\underline{z} = \begin{bmatrix} \underline{z}_1 \\ \underline{z}_2 \end{bmatrix}, \qquad \underline{c} = \begin{bmatrix} \underline{c}_1 \\ \underline{c}_2 \end{bmatrix}.$$

dove $\underline{c}_1 \in \mathbb{R}^r$ è il vettore contenente le prime r componenti, dato r il rango della matrice $\underline{\underline{A}}$, del vettore $\underline{c}$, mentre $\underline{c}_2 \in \mathbb{R}^{m-r}$ contiene le restanti, ultime, $m - r$ componenti. Lo stesso discorso si può fare per il vettore $\underline{z}$: $\underline{z}_1 \in \mathbb{R}^r$ è il vettore contenente le prime r componenti del vettore $\underline{z}$, mentre $\underline{z}_2 \in \mathbb{R}^{n-r}$ contiene le rimanenti $n - r$. Infine, se definiamo la matrice $\widetilde{\underline{\underline{S}}} \in \mathbb{R}^{r,r}$ come il *ritaglio* delle prime r righe e r colonne della matrice $\underline{\underline{S}}$, possiamo arrivare a scrivere

$$\left\| \underline{\underline{A}}\, \underline{y} - \underline{b} \right\|_2^2 = \left\| \underline{\underline{S}}\, \underline{z} - \underline{c} \right\|_2^2 = \left\| \begin{bmatrix} \widetilde{\underline{\underline{S}}}\, \underline{z}_1 - \underline{c}_1 \\ \underline{c}_2 \end{bmatrix} \right\|_2^2,$$

che ha senso poiché la matrice $\underline{\underline{S}}$ in totale ha una forma del tipo

$$\underline{\underline{S}} = \begin{bmatrix} \widetilde{\underline{\underline{S}}} & \underline{\underline{0}} \\ \underline{\underline{0}} & \underline{\underline{0}} \end{bmatrix},$$

dove tutti gli elementi all'infuori di quelli in $\widetilde{\underline{\underline{S}}}$ sono nulli, dal momento che i valori singolari in posizioni superiori a quella del rango sono nulli per costruzione della SVD.

Proprio come con la fattorizzazione QR, siamo riusciti a decomporre il problema in due sottoproblemi: le prime r equazioni sono raggiungibili dai gradi di libertà a nostra disposizione, e quindi questo ci permette di risolvere il problema dei minimi quadrati mediante la soluzione del sistema lineare

$$\widetilde{\underline{\underline{S}}}\, \underline{z}_1 = \underline{c}_1,$$

ma $\underline{c}_2$ non è *raggiungibile* dai nostri gradi di libertà e, quindi, proprio come prima, non possiamo farci nulla. Il sistema lineare appena scritto è ancora più semplice del solito: la matrice $\widetilde{\underline{\underline{S}}}$ infatti è diagonale e, quindi, è possibile scrivere la soluzione come

$$z_i = \frac{c_i}{s_i}, \qquad i = 1, 2, \ldots, r.$$

dove z_i e c_i sono le i-esime componenti dei vettori $\underline{z}$ e $\underline{c}_1$, rispettivamente, e s_i è l'i-esimo valore singolare della matrice $\underline{\underline{A}}$.

Si noti che però quanto appena scritto vale solo per le prime r componenti del vettore $\underline{z}$ ma non ci dice nulla sulle restanti $n - r$ componenti, che sono state di fatto escluse da tutti questi conti in quanto ininfluenti. Poiché tuttavia ciò che ci piacerebbe è non avere una soluzione a caso, ma **la soluzione di minima norma**, dal momento che qualsiasi cosa noi mettiamo per le ultime $n - r$ componenti del vettore $\underline{z}$, queste non cambieranno in alcun modo la norma che stiamo cercando di minimizzare[28], possiamo definire la soluzione di minima norma $\underline{z}^*$ come avente le componenti

[28] dal momento che corrispondono a elementi nulli della matrice $\underline{\underline{S}}$ in quanto in posizioni maggiori del rango della matrice

$$z_i^* = \begin{cases} \frac{c_i}{s_i}, & i = 1, ..., r \\ 0, & i = r+1, ..., n. \end{cases} \tag{4.23}$$

Dalla $\underline{z}^*$ di minima norma è possibile trovare la $\underline{x}^*$ di minima norma; infatti, dal momento che

$$\underline{x}^* = \underline{\underline{V}}\,\underline{z}^*, \tag{4.24}$$

essendo $\underline{\underline{V}}$ una matrice ortogonale, abbiamo che

$$||\underline{x}^*||_2^2 = \left|\left|\underline{\underline{V}}\,\underline{z}^*\right|\right|_2^2 = ||\underline{z}^*||_2^2,$$

e quindi la norma è preservata: essendo certamente $\underline{z}^*$ la soluzione di minima norma, ed essendo $\underline{z}^*$ e $\underline{x}^*$ legate da una matrice ortogonale, se $\underline{z}^*$ è quella avente minima norma, allora lo sarà anche $\underline{x}^*$.

Matrice pseudoinversa di Moore-Penrose

Se in qualche modo siamo riusciti a estendere il concetto di autovalore, e di conseguenza quello di determinante[29] a matrici rettangolari, non abbiamo ancora avuto modo di estendere il concetto di matrice inversa. Al fine di andare in questa direzione, ricordiamoci che, a partire da un sistema lineare

$$\underline{\underline{A}}\,\underline{x} = \underline{b},$$

in linea di principio[30], è possibile ottenere la soluzione $\underline{x}$ moltiplicando ambo i membri per la matrice inversa di $\underline{\underline{A}}$:

$$\underline{x} = \underline{\underline{A}}^{-1}\underline{b}. \tag{4.25}$$

Questa visione ci permetterà, per analogia, di estendere il concetto di matrice inversa. A partire dalla (4.23), possiamo definire una matrice "diagonale" $\underline{\underline{S}}^+$ avente componenti

$$(\underline{\underline{S}}^+)_{ii} = \begin{cases} \frac{1}{s_i}, & i = 1, ..., r \\ 0, & i = r+1, ..., m. \end{cases}$$

Questa definizione ci permette di scrivere compattamente l'espressione (4.23) come segue:

$$\underline{z}^* = \underline{\underline{S}}^+\underline{c}.$$

Questa espressione si può elaborare sostituendo (4.24) a membro sinistro e (4.22) a membro destro, arrivando a ottenere

$$\underline{\underline{V}}^{\mathrm{T}}\underline{x}^* = \underline{\underline{S}}^+\underline{\underline{U}}^{\mathrm{T}}\underline{b},$$

e, sfruttando l'ortogonalità della matrice $\underline{\underline{V}}$, portarla a membro destro è molto semplice, permettendoci di scrivere

$$\underline{x}^* = \underline{\underline{V}}\,\underline{\underline{S}}^+\underline{\underline{U}}^{\mathrm{T}}\underline{b}. \tag{4.26}$$

È evidente che questa espressione ricordi molto (4.25): si tratta di una relazione diretta tra il termine noto $\underline{b}$ e la soluzione $\underline{x}^*$; per questo motivo la matrice $\underline{\underline{A}}^+$, definita come

$$\underline{\underline{A}}^+ = \underline{\underline{V}}\,\underline{\underline{S}}^+\underline{\underline{U}}^{\mathrm{T}} \tag{4.27}$$

viene detta **matrice pseudoinversa di Moore-Penrose**, e, poiché permette di ottenere una relazione del tipo

$$\underline{x}^* = \underline{\underline{A}}^+\underline{b},$$

essa rappresenta una generalizzazione del concetto di matrice inversa. Non solo: è possibile dimostrare che, se $\underline{\underline{A}}$ è una matrice quadrata e a rango pieno,

[29] interpretato come prodotto degli autovalori

[30] ma la cosa è totalmente sconsigliata, come abbiamo discusso in passato

$$\underline{\underline{A}}^{+} = \underline{\underline{A}}^{-1},$$

a ulteriore testimonianza di quanto appena detto. MATLAB® permette di calcolare questa matrice, mediante il comando `pinv`.

Attenti a quale scorciatoia prendete!

Abbiamo appena completato l'argomento *sistemi lineari sovradeterminati*; tuttavia, vale la pena di proporre una piccola precisazione. Abbiamo visto, dopo aver introdotto la fattorizzazione QR, che l'implementazione può essere semplificata utilizzando il comando \, che riconosce automaticamente il fatto che il sistema è sovradeterminato e *internamente* effettua tutti i passaggi descritti. Tuttavia, **questo è vero solamente quando la matrice associata al sistema ha rango massimo**.

In verità, il comando \ permette di trovare **una** soluzione anche nel caso in cui il rango non sia massimo; tuttavia, **questa non sarà la soluzione avente norma minima**. Al fine di dimostrarlo, si propone il seguente script.

```
clear
close all
clc

A = [1   2   3;
     4   5   6;
     7   8   9;
     10 11 12]; % esempio di sistema sovradeterminato la cui matrice non ha
                % rango massimo

 b = [1:4]'; % esempio di termine noto

 r = rank(A) % dimostriamo, mediante il comando rank, che il rango
             % effettivamente e` inferiore a 3 (numero di colonne di A) !

 xBS = A\b % calcolo soluzione mediante il comando \
 xPINV = pinv(A)*b % calcolo soluzione mediante psuedo-inversa
```

Possiamo vedere che questo script visualizzerà:

```
xBS =
   -0.0000
         0
    0.3333

xPINV =
   -0.0556
    0.1111
    0.2778

>>
```

che sono soluzioni chiaramente diverse! Prima di tutto: è vero che entrambe sono davvero soluzioni del sistema lineare? Per farlo, proviamo a eseguire i seguenti comandi:

```
>> norm(A*xBS-b)
ans =
   9.1551e-16

>> norm(A*xPINV-b)
ans =
   3.0445e-15

>>
```

Entrambe le soluzioni sono valide: la norma 2 del residuo è sostanzialmente pari a zero. Tuttavia proviamo a guardare non solo la norma del residuo, ma anche la norma delle soluzioni; otteniamo:

```
>> norm(xBS)
ans =
    0.3333

>> norm(xPINV)
ans =
    0.3043

>>
```

Questo significa, come abbiamo già anticipato, che il comando \ trova sì una soluzione, **ma non quella di norma minima**! Si sia dunque consapevoli di questo, al momento di cercare delle *scorciatoie* per la risoluzione di problemi e/o esercizi!

Risoluzione di un sistema lineare sottodeterminato

La decomposizione ai valori singolari permette non solo di risolvere un sistema sovradeterminato, ma anche uno sottodeterminato, ammesso che la sua matrice abbia rango massimo! Considerando quindi un sistema nella forma

$$\underline{\underline{A}}\,\underline{x} = \underline{b},$$

dove $\underline{\underline{A}} \in \mathbb{R}^{m,n}$ ma, in questo caso, $m < n$, è possibile applicare la seguente procedura. Sostituendo $\underline{\underline{A}}$ con la sua SVD, si ha

$$\underline{\underline{U}}\,\underline{\underline{S}}\,\underline{\underline{V}}^{\mathrm{T}}\underline{x} = \underline{b},$$

che diventa, come già mostrato,

$$\underline{\underline{S}}\,\underline{\underline{V}}^{\mathrm{T}}\underline{x} = \underline{\underline{U}}^{\mathrm{T}}\underline{b}.$$

Come nel caso dei sistemi lineari determinati, si può quindi definire $\underline{y} = \underline{\underline{V}}^{\mathrm{T}}\underline{x}$, $\underline{c} = \underline{\underline{U}}^{\mathrm{T}}\underline{b}$, e risolvere i due sistemi lineari

$$\begin{cases} \underline{\underline{S}}\,\underline{y} = \underline{c} \\ \underline{\underline{V}}^{\mathrm{T}}\underline{x} = \underline{y}. \end{cases}$$

Proprio come fatto per quanto riguarda i sistemi sovradeterminati, è possibile accorgersi che la matrice $\underline{\underline{S}}$ è decomponibile nei blocchi

$$\underline{\underline{S}} = \begin{bmatrix} \widetilde{\underline{\underline{S}}} & \underline{\underline{0}} \end{bmatrix},$$

dove $\widetilde{\underline{\underline{S}}}$ è il ritaglio $r \times r$ di $\underline{\underline{S}}$, dunque è diagonale con elementi non nulli. Quindi, si può trovare la soluzione $\underline{y}^*$ avente minima norma del sistema

$$\widetilde{\underline{\underline{S}}}\,\underline{y}^* = \underline{c}, \tag{4.28}$$

usando la relazione

$$y_i^* = \begin{cases} \dfrac{c_i}{s_i}, & i = 1, \ldots, r \\ 0, & i = r+1, \ldots, n. \end{cases}$$

Infine, trovato $\underline{y}^*$ in questo modo, si ottiene

$$\underline{x}^* = \underline{\underline{V}}\,\underline{y}^*.$$

L'ipotesi sulla matrice $\underline{\underline{A}}$ di essere di rango massimo è molto importante, nonostante per ora apparentemente ingiustificata. In effetti, se la matrice fosse di rango non massimo, la matrice $\underline{\underline{S}}$ non sarebbe più decomponibile come appena mostrato, ma avrebbe una forma simile a quella dei sistemi sovradeterminati:

$$\underline{\underline{S}} = \begin{bmatrix} \underline{\underline{\widetilde{S}}} & \underline{\underline{0}} \\ \underline{\underline{0}} & \underline{\underline{0}} \end{bmatrix}.$$

Questa cosa sarebbe molto problematica, dal momento che non ci permetterebbe più di scrivere il sistema sottodeterminato nella forma (4.28), per via dei valori singolari nulli, generati dalle dipendenze lineari tra le righe di $\underline{\underline{A}}$. In particolare, eliminando l'ipotesi di rango massimo, perderemmo la garanzia di avere un sistema lineare dotato di soluzione. Infatti, volendo scrivere esplicitamente il sistema, avremmo più di r equazioni e, quindi, una forma del tipo[31]

$$\begin{cases} s_1 y_1^* = c_1 \\ s_2 y_2^* = c_2 \\ \vdots \\ s_r y_r = c_r \\ s_{r+1} y_{r+1} = c_{r+1} \\ \vdots \\ s_m y_m = c_m. \end{cases}$$

Fino alla r-esima equazione, dove r come al solito indica il rango della matrice, nessun problema! Tuttavia, $s_{r+1} = s_{r+2} = ... = s_m = 0$ e, quindi, le ultime $m - r$ equazioni sarebbero

$$\begin{cases} 0 \times y_{r+1} = c_{r+1} \\ \vdots \\ 0 \times y_m = c_m, \end{cases}$$

dove, a meno che per qualche fortuito caso i c_i siano uguali a zero, saremmo dinnanzi a un assurdo: il sistema non avrebbe soluzione! Per questo, l'ipotesi di rango massimo, ovvero $r = m$, è molto importante!

[31] dove le equazioni sono così semplici grazie al fatto che sono costruite a partire dalla matrice $\underline{\underline{S}}$, che è diagonale -o quasi-!

Bibliografia

[1] L. A. Steen, "Highlights in the history of spectral theory," *The American Mathematical Monthly*, vol. 80, no. 4, pp. 359-381, Apr. 1973.

Appendice C

Appendice: Autovalori e valori singolari di matrici

C.1 Un programmino per disegnare i cerchi di Gershgorin

Vengono ora riportati lo script per la generazione di Fig. 4.1, con la function utilizzata.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Questo programma puo` essere usato per produrre i cerchi di Gershgorin
% per un esempio di matrice e per visualizzare i suoi autovalori, in modo
% da cercare di chiarire questi concetti.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear
close all
clc

% Esempio di matrice
A=[30 1   2  3;
    4 15 -4 -2;
   -1  0  3  5;
   -3  5  0  1];

fLez10_Gershgorin(A)

% Calcolo gli autovalori della matrice
E=eig(A);

% Disegno gli autovalori della matrice, in parte reale e immaginaria,
% mediante dei cerchietti neri
subplot(1,2,1)
for ind=1:length(E)
    plot(real(E(ind)),imag(E(ind)),'ko','LineWidth',2)
end
ylim([-22,22])

% Disegno gli autovalori della matrice, in parte reale e immaginaria,
% mediante dei cerchietti neri
subplot(1,2,2)
for ind=1:length(E)
    plot(real(E(ind)),imag(E(ind)),'ko','LineWidth',2)
end
ylim([-22,22])
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Questa funzione e` stata tratta dalle soluzioni all'esercitazione 4, e
% viene commentata e modificata lievemente al fine di abbellire i grafici.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function fLez10_Gershgorin(A)

n = size(A,1); % trovo la dimensione della matrice

% Disegno dei cerchi riga di Gershgorin
figure(1)
set(gcf,'Position',[277 440 1091 420])
subplot(1,2,1)
box on
```

```
centro = diag(A); % centri dei cerchi di Gershgorin
raggio_riga = sum(abs(A),2)-diag(abs(A));
theta = linspace(0,2*pi); % angolo per il disegno dei cerchi
for i=1:n % per ciascuna delle righe della matrice disegno un cerchio
    % Senza entrare in dettaglio, si sfrutta la parametrizzazione della
    % circonferenza mediante le funzioni seno e coseno.
    x = centro(i)+raggio_riga(i)*cos(theta);
    y = raggio_riga(i)*sin(theta);
    patch(x,y,'r') % il comando patch permette di disegnare i cerchi
    axis equal
    hold on
end
alpha(0.3) % questo comando cambia la trasparenza dei cerchi

% Disegno dei cerchi colonna di Gershgorin
subplot(1,2,2)
box on
centro = diag(A);
raggio_colonna = sum(abs(A),1)'-diag(abs(A));
theta = linspace(0,2*pi);
for i=1:n
    x = centro(i)+raggio_colonna(i)*cos(theta);
    y = raggio_colonna(i)*sin(theta);
    patch(x,y,'g')
    axis equal
    hold on
end
alpha(0.3)
```

C.2 Applicazione della SVD alla compressione di immagini

Una delle proprietà della decomposizione ai valori singolari riguarda la possibilità di trovare la migliore approssimante $\underline{\underline{A}}_k$ avente rango massimo k di una matrice $\underline{\underline{A}}$. Un'applicazione di questa fantastica idea è la compressione di immagini. Infatti, dal punto di vista di un computer, un'immagine è descrivibile mediante il linguaggio matriciale. In particolare, negli schermi, ogni colore viene ottenuto mescolando additivamente il rosso, il verde e il blu, portandoci al sistema RGB (red, green, blue)[1]. Ciò che possiamo fare è dunque associare una matrice alle componenti di rosso, una ai verdi e una ai blu, e calcolarne la SVD; quindi, utilizzando il teorema (4.21) è possibile ricostruirne una versione di rango inferiore, arrivando a comprimere l'immagine.

Come possiamo quantificare l'efficacia della compressione? Compressione significa rapporto e, quindi, consideriamo come *dimensione su disco* della figura la dimensione della matrice, memorizzata in doppia precisione, su MATLAB®. Questo non è davvero il *peso* del file `.jpg`, in quanto i JPEG sono già un formato compresso, e non è detto che i numeri siano memorizzati in doppia precisione: per applicazioni di questo tipo probabilmente gli interi a 8 bit, ottenibili a partire dai `double` mediante il comando `uint8`, sono più che sufficienti. Tuttavia, visto che ci interessiamo solo al rapporto tra il *peso* dell'immagine di partenza e di quella compressa, il tipo di memorizzazione non è troppo rilevante, a patto di essere coerenti tra numeratore e denominatore.

Volendo prendere questa strada, ricordiamo che un numero in doppia precisione richiede 64 bit di memoria[2], ovvero 8 byte. Memorizzare una figura in una matrice di numeri a doppia precisione $\underline{\underline{A}}$ richiede $24P_xP_y$ byte, dove P_x e P_y sono i numeri di pixel lungo ascisse e ordinate, e che quindi saranno il numero di colonne e il numero di righe della nostra matrice, rispettivamente: un numero per ogni pixel; il 24 nasce come 8×3: 8 byte, per 3 colori.

Al momento di calcolare la decomposizione ai valori singolari di questa matrice $\underline{\underline{A}}$, otteniamo:

$$\underline{\underline{A}} = \underline{\underline{U}}\,\underline{\underline{S}}\,\underline{\underline{V}}^{\mathrm{T}},$$

dove $\underline{\underline{U}}$ sarà una matrice quadrata di dimensioni $P_y \times P_y$, $\underline{\underline{V}}$ sarà ancora quadrata e di dimensioni $P_x \times P_x$, e $\underline{\underline{S}}$ sarà rettangolare ma avente elementi non nulli solo sulla diagonale principale. Sfruttando il teorema (4.21), invece di considerare tutte le colonne, possiamo decidere di prenderne solo k, e ricostruire quindi la matrice di rango k più simile rispetto alla norma spettrale. Dover salvare solo le matrici $\underline{\underline{U}}^{(k)}$, $\underline{\underline{S}}^{(k)}$ e $\underline{\underline{V}}^{(k)}$ al posto di $\underline{\underline{A}}$ richiede, dal punto di vista della memoria,

$$24(kP_x + kP_y + k),$$

[1] in cartellonistica per esempio si utilizza il sistema sottrattivo ciano-magenta-giallo CMY (cyan, magenta, yellow)

[2] non per nulla negli ultimi anni si utilizzano architetture a 64 bit!

sfruttando il fatto che ritagliamo solo k colonne di $\underline{\underline{U}}$ (ciascuna delle quali ha P_y componenti), k colonne di $\underline{\underline{V}}$ (ciascuna delle quali ha P_x componenti), più k valori singolari; questo, per 8 byte e 3 colori.

Segue un codice che effettua tutti questi passaggi e permette di visualizzare le immagini compresse.

```
clear
close all
clc

vr=[1 2 5 10 20 30 150]; % vettore con ranghi per approssimazione

% A = imread('demo1.jpg'); % carico la figura nella matrice A
A = imread('demo2.jpg'); % carico la figura nella matrice A

% A e` una matrice di matrici, ovvero una matrice a 3 indici; da questa si
% possono ricavare tre matrici "a due indici", corrispondenti alle
% componenti R, G, B dei colori.
% Il comando "squeeze" serve a ottenere una matrice a 2 indici (vedi help)
A_R=squeeze(A(:,:,1)); % blackness matrix color R
A_G=squeeze(A(:,:,2)); % blackness matrix color G
A_B=squeeze(A(:,:,3)); % blackness matrix color B

% La SVD va applicata a numeri "reali", ovvero "double", quindi bisogna
% fare un cast a double prima di calcolarla
A_R=double(A_R);
A_G=double(A_G);
A_B=double(A_B);

% Calcolo la SVD per ciascuna delle matrici di colori
[U_R,S_R,V_R] = svd(A_R);
[U_G,S_G,V_G] = svd(A_G);
[U_B,S_B,V_B] = svd(A_B);

% Calcolo la dimensione totale della figura non compressa
[Py,Px] = size(A_R); % le tre matrici hanno lo stesso numero di pixel e
                     % sono tutte memorizzate in doppia precisione quindi
                     % posso calcolare Px, Py solo da una di esse!

DimTotale = 8*3*Px*Py; % numero totale di pixel per 24 byte

% Applichiamo a questo punto l'algoritmo di approssimazione della matrice;
% proviamo per matrici approssimanti avente diverso rango a verificare la
% qualita` della ricostruzione
for indn=1:length(vr)

    r=vr(indn);
    A_R_r=U_R(:,1:r)*S_R(1:r,1:r)*V_R(:,1:r)';
    A_G_r=U_G(:,1:r)*S_G(1:r,1:r)*V_G(:,1:r)';
    A_B_r=U_B(:,1:r)*S_B(1:r,1:r)*V_B(:,1:r)';

    % Per la visualizzazione, bisogna riconvertire "all'indietro" in numeri
    % interi, e salvare in quella specie di matrice di matrici!
    Ar(:,:,1)=uint8(A_R_r);
    Ar(:,:,2)=uint8(A_G_r);
    Ar(:,:,3)=uint8(A_B_r);

    DimApp = (Px*r + Py*r + r)*8*3;

    figure(1)
    clf % cancello contenuto figura precedente
    imshow(Ar) % mostro la figura approssimata in Ar
    title(['Approssimazione a rango ',num2str(r), ', compressione al ',num2str(DimApp/DimTotale*100),' %'↩
        ])
    pause

end
```

Qualche commento conclusivo. Il comando

```
A = imread('nomefile.jpg');
```

ordina a MATLAB® di leggere la figura e salvarla nella matrice `A`; questa è una specie di *matrice di matrici*, cioè una *matrice a tre indici*. In verità non è nulla di troppo complicato: si tratta di tre matrici, ciascuna delle quali contiene le informazioni su un colore. Volendo, il programma si può semplificare e portare in bianco e nero, usando il comando

```
B = rgb2gray(A);
```

che converte le tre matrici in un'unica matrice, che verrà interpretata come scala di grigi. Le matrici generate con queste funzioni sono in un formato particolare, quindi è necessario effettuare dei cast a `double` al momento di elaborarle, o a `uint8` al momento di visualizzarle, per esempio con il comando `imshow`.