Maria Grazia Gasparo
e
Rossana Morandi

ELEMENTI DI CALCOLO NUMERICO: METODI E ALGORITMI

Nello infinito accadono cose
che non accadono nello finito
Galileo Galilei

Solo perché alcuni di noi sanno leggere e
scrivere e far di conto, questo non vuol dire
che meritiamo di conquistare l'Universo.
Kurt Vonnegut, Jr.

Indice

Prefazione

Il Calcolo numerico, o Analisi numerica, è la disciplina che si occupa della cosiddetta risoluzione numerica di problemi di natura matematica, ovvero della loro risoluzione mediante il calcolatore elettronico.

La necessità di ricorrere al calcolatore può nascere per due motivi: perché non esistono formule o procedimenti espliciti per risolvere il problema o perché i dati e le variabili in gioco sono troppi per poter essere facilmente gestiti "a mano". Un esempio della prima situazione è rappresentato dalle equazioni non lineari: tutti sanno risolvere l'equazione $3x^2 + 5x - 2 = 0$, ma nessuno conosce una formula per calcolare le soluzioni di $e^x cos(x) - sin(x) = 3$. Per quanto riguarda la seconda situazione, basta pensare ad un sistema lineare: se si tratta di 2 o 3 equazioni in altrettante incognite non è difficile fare i conti con carta e penna, ma se le equazioni sono 10, 100 o 1000 bisogna necessariamente usare uno strumento più veloce.

Una volta che un problema è stato correttamente formulato, si deve individuare un metodo per risolverlo numericamente, ovvero un metodo che, realizzato sul calcolatore, fornisce la soluzione del problema o una sua approssimazione. Questi metodi sono detti metodi numerici. Affinché un metodo numerico possa essere usato in pratica, esso deve essere descritto tramite un algoritmo, che è una sequenza di istruzioni non ambigue che si conclude sicuramente entro un tempo finito. Lo studio dei metodi numerici e la formulazione dei corrispondenti algoritmi rientrano fra le prerogative del Calcolo numerico. Spesso si possono individuare diversi metodi numerici per risolvere uno stesso problema. È ancora compito del Calcolo numerico confrontarli, fondamentalmente in base a due criteri: il costo, ovvero la quantità di risorse di memoria e di tempo che ciascuno di essi richiede, e la sensibilità a tutti i possibili errori che possono influenzare il risultato. Mentre il primo aspetto è fortemente dipendente dal tipo di calcolatore che si ha a disposizione, il secondo aspetto è in qualche modo oggettivo e particolarmente importante dal momento che il calcolatore stesso, qualunque esso sia, è una fonte certa di errore: a causa delle sue caratteristiche fisiche, esso genera automaticamente errori sia nella memorizzazione dei numeri che nell'esecuzione delle operazioni aritmetiche, tanto che l'aritmetica del calcolatore ha un nome specifico (aritmetica floating point) per distinguerla dall'aritmetica che si usa abitualmente in matematica, in cui virtualmente si possono eseguire algoritmi anche complessi senza introdurre errori.

I primi due capitoli del libro sono dedicati rispettivamente agli algoritmi e all'aritmetica floating point. Nei capitoli successivi si trattano metodi numerici per la risoluzione di alcune classi di problemi: equazioni non lineari, sistemi lineari e approssimazione di dati e funzioni. Le caratteristiche più importanti dei metodi numerici presentati sono illustrate da numerosi esempi, con figure e tabelle. I risultati numerici che riportiamo nelle tabelle sono stati ottenuti realizzando gli algoritmi del testo in linguaggio FORTRAN e/o in linguaggio MATLAB.

Le note storiche che accompagnano il testo sono tratte spesso da Internet e, quando possibile, verificate in articoli e libri. Citiamo a questo proposito i due libri molto interessanti di J.L. Chabert [8] e H.H. Goldstine [19] di storia degli algoritmi e dell'Analisi numerica.

Il testo è rivolto principalmente a studenti dei corsi di base di Calcolo numerico delle Facoltà di Ingegneria e Scienze matematiche, fisiche e naturali. Gli argomenti sono trattati in modo da richiedere soltanto prerequisiti elementari di Analisi matematica e Geometria. Alcune nozioni di Algebra lineare sono comunque raccolte in Appendice. Abbiamo incluso anche alcuni argomenti che richiedono un po' più di maturità matematica e possono essere eventualmente omessi in un contesto di introduzione al Calcolo numerico. Questi argomenti sono contrassegnati nell'indice da un asterisco (*).

Nonostante tutta l'attenzione che abbiamo messo nella stesura del libro e le innumerevoli riletture, siamo sicure che alcuni errori, speriamo solo di stampa, sono rimasti. Ce ne assumiamo tutta la responsabilità e siamo grate fin da ora a chiunque ci aiuterà ad individuarli.

Desideriamo ringraziare molte persone: Aldo e Ferruccio che tanti anni fa ci hanno fatto appassionare al Calcolo numerico; tutti gli studenti che hanno seguito i nostri corsi e ci hanno fatto capire cosa insegnare e come; i nostri consorti che hanno dimostrato un'infinita pazienza accettando di buon grado che lavorassimo perfino durante le vacanze; un grazie speciale a Linda e Lorenzo che, avendo studiato Calcolo numerico senza un libro di testo di riferimento, hanno spinto perché questo libro fosse finalmente scritto. Infine, vogliamo ringraziare la nostra pluridecennale amicizia in virtù della quale il lavoro non è stato solo faticoso ma anche divertente. Come diceva Aristotele, si decide in fretta di essere amici, ma l'amicizia è un frutto che matura lentamente.

Maria Grazia Gasparo
e Rossana Morandi

Firenze, gennaio 2008

1

ALGORITMI

1.1 Introduzione

Lo scopo del Calcolo numerico è la risoluzione mediante il calcolatore elettronico di problemi di natura matematica. Vale la pena di ricordare che per risolvere un problema occorre individuare i dati, i risultati che si vogliono ottenere e un procedimento che permetta di ottenere i risultati a partire dai dati in un tempo finito. In generale si usa il termine problema per identificare una famiglia di problemi. Ad esempio, un problema è il calcolo delle soluzioni reali di un'equazione di secondo grado

$$ax^2 + bx + c = 0,$$

dove a, b e c sono tre numeri reali, detti coefficienti, e x è l'incognita. In questo caso i dati sono a, b, c e i risultati sono il numero n di soluzioni reali distinte (0, 1 o 2) e, nel caso che n sia diverso da 0, le soluzioni (x_1 ed eventualmente x_2). Una volta fissato un particolare insieme di dati si ottengono i corrispondenti risultati. Così, se i dati sono

$$a = 1, b = 4, c = -5,$$

il risultato è

$$n = 2, x_1 = 1, x_2 = -5;$$

se i dati sono

$$a = 1, b = 2, c = 1$$

il risultato è

$$n = 1, x_1 = 1;$$

se infine

$$a = 1, b = 0, c = 8$$

il risultato è

$$n = 0.$$

Fatta questa premessa, un procedimento risolutivo per un problema è una sequenza finita di istruzioni che, applicata a qualunque insieme di dati, produce il risultato corrispondente in un tempo finito.

Quando si descrive un procedimento per la risoluzione di un problema, occorre tenere presenti le capacità della persona o della macchina a cui sarà affidata l'esecuzione. Per fare un classico esempio non matematico, nel descrivere una ricetta di cucina possiamo scrivere l'istruzione "aggiungere sale quanto basta" soltanto se ci rivolgiamo ad una persona abituata a cucinare; in caso contrario l'istruzione suddetta risulterà ambigua e di fatto renderà impossibile l'esecuzione della ricetta. Nel Calcolo numerico, e in generale nelle applicazioni informatiche, l'esecutore è il calcolatore che è sicuramente caratterizzato dal fatto di non poter risolvere ambiguità mediante il "buon senso" o conoscenze culturali pregresse che caratterizzano invece l'essere umano. Pertanto un procedimento atto a risolvere un problema mediante il calcolatore non deve presentare ambiguità.

Riassumendo, siamo interessati a procedimenti risolutivi per un determinato problema che siano descritti in modo non ambiguo e consentano di passare da un insieme di dati al corrispondente risultato in un tempo finito. Procedimenti con queste caratteristiche sono chiamati algoritmi [1].

1.2 Istruzioni fondamentali

Ogni calcolatore è in grado di interpretare ed eseguire istruzioni scritte in un particolare linguaggio, detto linguaggio macchina, che è strettamente legato alle caratteristiche dell'hardware e ha regole sintattiche non semplici dal momento che può utilizzare soltanto due simboli, "0" e "1". Fortunatamente noi non siamo costretti a descrivere gli algoritmi direttamente in linguaggio macchina, ma possiamo usare i linguaggi di programmazione (ad alto livello), che hanno una sintassi molto più semplice. Un algoritmo scritto in un linguaggio di programmazione si chiama programma e viene tradotto in linguaggio macchina mediante software opportuno. In questo testo non faremo riferimento nello scrivere gli algoritmi a nessun linguaggio di programmazione in particolare, ma useremo una forma "discorsiva", in lingua italiana, molto generale, che potrà comunque essere immediatamente tradotta in uno dei linguaggi di programmazione più comunemente usati nel contesto del Calcolo numerico, quali il FORTRAN, il MATLAB e il C. Nell'ottica precedentemente chiarita, ogni algoritmo può essere descritto usando istruzioni elementari, ossia istruzioni che descrivono azioni elementari (quali ad esempio: leggere un dato, scrivere un risultato, calcolare e memorizzare il valore di un'espressione) e istruzioni più complesse (o co-

[1] Il termine deriva dal nome del matematico persiano Abu Ja'far Mohammed ibn Mâsâ al-Khowârizmî (circa 825 d.C.), che utilizzò questo concetto in un trattato che venne tradotto in latino durante il medioevo con il titolo "Liber algarismi".

strutti) che intervengono sull'ordine sequenziale di esecuzione, ad esempio imponendo una scelta fra due o più percorsi differenti in base al verificarsi o meno di una determinata condizione, oppure imponendo che alcune azioni vengano ripetute più volte. Tutte le istruzioni sono caratterizzate da una parola chiave che esprime l'azione descritta dall'istruzione e agiscono su grandezze identificate mediante nomi simbolici dette abitualmente variabili. Le parole chiave sono stabilite dal linguaggio usato per scrivere l'algoritmo; i nomi delle variabili sono scelti da chi scrive l'algoritmo e di solito sono nomi mnemonici che ricordano il significato della variabile stessa.

1.2.1 Istruzioni elementari

Le istruzioni elementari sono fondamentalmente di due tipi:

Istruzioni di ingresso/uscita
Queste sono le istruzioni che permettono l'acquisizione dei dati e la restituzione dei risultati; esse sono caratterizzate dalle parole chiave "Leggi" e "Scrivi', seguite dalla lista delle variabili di cui si vuole acquisire il valore o delle espressioni di cui si vuole stampare il valore.

Esempi:
1) Leggi: a, b, c
2) Scrivi: n, x_1, x_2
3) Leggi: $n, a_1, ..., a_n$
4) Scrivi: $n + 1, m, k$
5) Scrivi: $'valore\ \ di\ \ n :', n$
I primi quattro esempi non hanno bisogno di spiegazione. Nell'ultimo esempio si scrive la stringa valore di n: seguita dal valore della variabile n.

Istruzione di assegnazione
Queste sono le istruzioni mediante le quali si calcola il valore di un'espressione, nella quale compaiono in generale nomi di variabili e costanti, e successivamente si assegna questo valore ad una variabile v. In questo libro le istruzioni di assegnazione avranno la forma semplice:

$$v \quad = \quad espressione$$

Ci preme sottolineare che il simbolo "=" viene usato in questo contesto non per affermare che i due termini a destra e a sinistra di esso sono uguali, ma per identificare l'operazione di assegnazione sopra descritta.

Esempi:
1) $area = lato^2$
2) $delta = b^2 - 4ac$
3) $a = b + c$

4) $media = somma/m$
5) $x = 0$

1.2.2 Istruzioni di scelta

Queste istruzioni, chiamate anche costrutti decisionali, si utilizzano ogni volta che l'algoritmo prevede una scelta fra due o più strade diverse in base al verificarsi di una o più condizioni. Poiché queste istruzioni iniziano sempre con la parola chiave Se e la lingua dominante nei linguaggi di programmazione è l'inglese, esse sono dette familiarmente istruzioni IF.

Esempio 1:
Se $a \neq b$, allora:
$c = (a + b)/(a - b)$
Fine scelta

Il significato di questa istruzione è: se il valore di a è diverso da quello di b, allora calcola il valore di $(a+b)/(a-b)$ e assegna tale valore alla variabile c; altrimenti salta questa assegnazione e procedi in sequenza con l'istruzione successiva.

Esempio 2:
Se $a > b$, allora:
$max = a$
altrimenti:
$max = b$
Fine scelta

In questo caso, si assegna un valore diverso alla variabile max a seconda della relazione che intercorre fra i valori di a e di b. In ogni caso dopo la scelta l'algoritmo riprende in sequenza con l'istruzione successiva.

Esempio 3:
Se $delta > 0$, allora:
$n = 2$
altrimenti se $delta = 0$, allora:
$n = 1$
altrimenti:
$n = 0$
Fine scelta

Questo è un esempio di costrutto decisionale più complesso dei precedenti: si tratta di scegliere fra tre possibili strade in base al verificarsi di due diverse condizioni che vengono controllate in successione, una dopo l'altra. Nei linguaggi di programmazione un costrutto come questo prende il nome di IF *concatenati*.

1.2.3 Istruzioni di ripetizione

Queste istruzioni, dette anche semplicemente cicli, si utilizzano tutte le volte che un gruppo di istruzioni (eventualmente una sola) deve essere ripetuto più volte. Più precisamente, si tratta di descrivere azioni del tipo: "Ripeti 15 volte le tali istruzioni", oppure " Ripeti il tale gruppo di istruzioni fintanto che vale una certa condizione". Nel primo caso si parla di cicli finiti perché si sa a priori il numero di ripetizioni, mentre nel secondo caso si parla di cicli condizionali o cicli WHILE.

Esempio 1:
Per $k = 1, 2, \ldots, K$
    $a = a + b$
    Scrivi a
Fine ciclo su k

In questo esempio di ciclo finito la variabile k agisce come contatore delle ripetizioni, il cui numero complessivo è fissato pari al valore di K. Per ogni valore di k da 1 a K, il valore della variabile a viene modificato sommandogli il valore della variabile b; ogni volta il nuovo valore di a viene stampato. Puntualizziamo il significato dell'istruzione di assegnazione $a = a + b$: al valore attuale di a si somma il valore di b e si assegna il nuovo valore così ottenuto ad a stessa.

Esempio 2:
Fintanto che $a > b$
    $a = a - b$
    Scrivi a
Fine ciclo

In questo esempio, al valore di a viene sottratto ripetutamente il valore di b; le sottrazioni continuano fintanto che il valore di a resta maggiore di quello di b; non appena a diventa minore o uguale a b, il ciclo si interrompe. Se nel momento in cui inizia l'esecuzione del ciclo il valore di a non è maggiore di quello di b, le istruzioni del ciclo non vengono eseguite neppure una volta. Osserviamo che questo ciclo ha senso se e solo se b è positivo; in caso contrario il ciclo non si interrompe mai. Il rischio di "loop", ovvero di descrivere un'azione che non finirà mai, è un rischio insito nei cicli condizionali, che quindi vanno usati con molta attenzione.

1.3 Esempi

Diamo qui di seguito alcuni esempi di algoritmi per la risoluzione di semplici problemi matematici. In questi algoritmi le istruzioni sono numerate in ordine progressivo per evidenziare la sequenzialità dell'esecuzione. Inoltre non indichiamo esplicitamente le istruzioni di ingresso/uscita, ma preferia-

mo inserire in testa all'algoritmo la descrizione dei dati che l'algoritmo deve acquisire e dei risultati che esso fornisce. Questa scelta evidenzia il fatto che l'acquisizione dei dati non necessariamente avviene attraverso la lettura e, allo stesso modo, la restituzione dei risultati non necessariamente avviene attraverso la scrittura. Per esemplificare, chiamiamo Algoritmo 1 l'algoritmo che stiamo scrivendo e supponiamo che esso sia parte di un algoritmo più lungo, detto Algoritmo 2. In questa situazione, i dati per Algoritmo 1 potrebbero essere calcolati nelle istruzioni di Algoritmo 2 che lo precedono, nel qual caso non è prevista nessuna istruzione di lettura dall'esterno per acquisirne i valori. Allo stesso modo, i risultati forniti da Algoritmo 1 potrebbero servire al proseguimento di Algoritmo 2, nel qual caso non ci sarebbe alcun motivo per visualizzarli tramite un'istruzione di stampa. Per chi conosce un linguaggio di programmazione, basta pensare al caso in cui un algoritmo venga realizzato mediante un sottoprogramma: in questo caso la comunicazione dei dati e dei risultati non avviene direttamente con l'esterno, bensì con l'unità di programma che utilizza il sottoprogramma.

Algoritmo 1.3.1. Algoritmo per il calcolo della somma di n numeri $a_1, \ldots, a_n$.

Dati: n, $a_1, \ldots, a_n$
Risultato: $s = \sum_{i=1}^{n} a_i$
1. $s = 0$
2. Per $i = 1, 2, \ldots, n$
    1. $s = s + a_i$

   Fine ciclo su i
3. Fine

Il risultato s viene costruito attraverso una sequenza di passaggi. Inizialmente (istruzione 1) alla variabile s viene assegnato il valore 0; questa operazione viene detta inizializzazione di s. Successivamente, attraverso un ciclo finito (istruzione 2), si sommano a s uno per volta gli addendi $a_1, a_2, \ldots, a_n$.

Algoritmo 1.3.2. Algoritmo per il calcolo del prodotto di n numeri $a_1, \ldots, a_n$.

Dati: n, $a_1, \ldots, a_n$
Risultato: $p = \prod_{i=1}^{n} a_i$
1. $p = 1$
2. Per $i = 1, 2, \ldots, n$
    1. $p = p \times a_i$

   Fine ciclo su i
3. Fine

Questo algoritmo è analogo al precedente. Dovendo calcolare un prodotto, la variabile p viene inizializzata con il valore 1 (elemento neutro per la

moltiplicazione) e successivamente modificata moltiplicando il suo valore con un fattore a_i per volta.

Algoritmo 1.3.3. Algoritmo per il calcolo del massimo fra p numeri $x_1, \ldots, x_p$.

Dati: p, $x_1, \ldots, x_p$
Risultato: $M = \max\{x_1, \ldots, x_p\}$
1. $M = x_1$
2. Per $i = 2, \ldots, p$
    1. se $x_i > M$, allora:
        $M = x_i$
    Fine scelta
  Fine ciclo su i
3. Fine

L'algoritmo prevede l'inizializzazione della variable M con il valore del primo dato x_1 (istruzione 1). Questo valore viene poi aggiornato mediante successivi confronti con i dati $x_2, \ldots, x_p$: ogni volta che viene trovato un dato maggiore di M, il suo valore viene acquisito e memorizzato come nuovo valore di M. In questo modo, ad ogni passo M rappresenta il più grande fra i dati esaminati fino a quel momento. Al termine del ciclo M è il risultato voluto.

Algoritmo 1.3.4. Algoritmo per determinare la posizione del termine più grande in valore assoluto in un insieme $\{x_1, \ldots, x_p\}$. Se il valore assoluto più grande corrisponde a più dati, l'algoritmo deve restituire l'indice del primo.

Dati: p, $x_1, \ldots, x_p$
Risultato: s tale che $|x_s| = \max\{|x_1|, \ldots, |x_p|\}$
1. $M = |x_1|$ e $s = 1$
2. Per $i = 2, \ldots, p$
    1. se $|x_i| > M$, allora:
        $M = |x_i|$ e $s = i$
    Fine scelta
  Fine ciclo su i
3. Fine

La struttura di questo algoritmo è simile a quella dell'algoritmo precedente. In questo caso l'inizializzazione riguarda M, a cui viene dato il valore $|x_1|$, e s a cui per coerenza viene dato il valore 1. Nel successivo ciclo, se il valore di M viene aggiornato, anche s viene modificato di conseguenza.

Algoritmo 1.3.5. Algoritmo per calcolare quoziente e resto della divisione fra due numeri interi positivi n ed m (versione 1).

Dati: n, m
Risultati: q (quoziente) e r (resto)
1. $r = n$
2. $q = 0$
3. Fintanto che $r \geq m$,
    1. $r = r - m$
    2. $q = q + 1$
  Fine ciclo
4. Fine

Questo algoritmo, spesso citato come "divisione euclidea", si basa su un'idea molto elementare. Se $n < m$, allora il ciclo non viene attivato perché il risultato ($q = 0$ e $r = n$) è già disponibile. Altrimenti si sottrae m da n tante volte quante sono necessarie perché il numero che rimane diventi più piccolo di m. Il quoziente q è il numero di sottrazioni effettuate e il resto r è il numero rimasto dopo l'ultima sottrazione. Poiché non è noto a priori il numero di sottrazioni che vanno fatte, il procedimento ripetitivo si configura come un ciclo condizionale.

L' algoritmo successivo realizza lo stesso procedimento usando istruzioni diverse. Il ciclo WHILE non è descritto usando l'istruzione "Fintanto che" bensì usando in modo opportuno l'istruzione IF e l'istruzione di salto "Vai a 4." che significa "Riprendi l'esecuzione dall'istruzione 4".

Algoritmo 1.3.6. Algoritmo per calcolare quoziente e resto della divisione fra due numeri interi positivi n ed m (versione 2).

Dati: n, m
Risultati: q (quoziente) e r (resto)
1. $r = n$
2. $q = 0$
3. Se $n < m$, allora:
      Fermati
  Fine scelta
4. $r = r - m$
5. $q = q + 1$
6. Se $r \geq m$, allora:
      Vai a 4.
  Fine scelta
7. Fine

Il ciclo si svolge fra le istruzioni 4 e 6. Per fermare l'esecuzione prima di iniziare il ciclo nel caso che n sia minore di m abbiamo dovuto inserire un

controllo esplicito all'istruzione 3.

Algoritmo 1.3.7. Algoritmo per calcolare il valore assunto in $x = \hat{x}$ da un polinomio di grado n della forma

$$P(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n. \tag{1.1}$$

L'algoritmo segue immediatamente dalla forma del polinomio:

Dati: n, $a_0, a_1, \ldots, a_n$, $\hat{x}$
Risultato: $y = P(\hat{x})$
1. $p = 1$
2. $s = a_0$
3. Per $i = 1, 2, \ldots, n$
    1. $p = \hat{x} \times p$
    2. $s = s + a_i \times p$

   Fine ciclo su i
4. $y = s$
5. Fine

Questo algoritmo utilizza due variabili di appoggio, p ed s. I valori assunti da p sono le potenze di $\hat{x}$: infatti p viene inizialmente posto uguale ad 1 e successivamente moltiplicato per $\hat{x}$ ad ogni passo del ciclo (cfr. algoritmo 1.3.2) così che p vale $\hat{x}^i$ al variare di i. Nella variabile s viene accumulata la sommatoria (cfr. algoritmo 1.3.1). L' algoritmo richiede in tutto $3n$ operazioni aritmetiche, $2n$ moltiplicazioni e n addizioni, e non viene quasi mai usato nella pratica perché è possibile ottenere lo stesso risultato facendo meno operazioni.

L'algoritmo utilizzato comunemente per calcolare i valori assunti da un polinomio è noto come algoritmo di Hörner, o delle moltiplicazioni annidate. Esso parte dalla riscrittura del polinomio in una forma diversa da (1.1), apparentemente più complicata ma più conveniente dal punto di vista dei calcoli; la forma in questione è la seguente:

$$\begin{aligned} P(x) &= a_n x^n + a_{n-1} x^{n-1} + \ldots + a_2 x^2 + a_1 x + a_0 \\ &= (a_n x^{n-1} + a_{n-1} x^{n-2} + \ldots + a_2 x + a_1) x + a_0 \\ &= ((a_n x^{n-2} + a_{n-1} x^{n-3} + \ldots + a_2) x + a_1) x + a_0 \\ &\vdots \\ &= (\ldots((a_n x + a_{n-1}) x + a_{n-2}) x + \ldots + a_1) x + a_0 \end{aligned} \tag{1.2}$$

Sulla base della (1.2) si può scrivere il seguente algoritmo:

Algoritmo 1.3.8. Algoritmo di Hörner per calcolare il valore assunto in $x = \hat{x}$ da un polinomio di grado n.

Dati: n, $a_0, a_1, \ldots, a_n$, $\hat{x}$
Risultato: $y = P(\hat{x})$
1. $s = a_n$
2. Per $i = n-1, n-2, \ldots, 0$
   $s = s \times \hat{x} + a_i$
   Fine ciclo su i
3. $y = s$
4. Fine

L'algoritmo calcola per $x = \hat{x}$ e memorizza in s i contenuti delle parentesi tonde in (1.2), partendo dalla più interna e procedendo verso l'esterno. I coefficienti del polinomio intervengono in ordine di indice decrescente, da a_n ad a_0; per questo motivo il ciclo è impostato in modo che i valori di i vadano a diminuire, da $i = n-1$ a $i = 0$ (ciclo all'indietro). Si può facilmente vedere che questo algoritmo richiede soltanto $2n$ operazioni.

1.4 Confronto fra algoritmi

Spesso si hanno a disposizione diversi algoritmi per risolvere uno stesso problema (cfr. i due algoritmi 1.3.7 e 1.3.8). Si pone allora la questione: in base a quali considerazioni si può o si deve preferire un algoritmo rispetto ad un altro? I termini di confronto sono fondamentalmente tre:

1) un algoritmo è da preferirsi ad un altro se è più stabile, ovvero se si può dimostrare (teoricamente e/o sperimentalmente) che fornisce risultati più accurati. Il concetto di stabilità sarà introdotto e analizzato nel prossimo capitolo;
2) un algoritmo è da preferirsi ad un altro se è più robusto, ovvero se è dimostrato sperimentalmente che riesce a risolvere un maggior numero di problemi. Esempi di algoritmi più o meno robusti di altri verranno fatti nel Capitolo 3;
3) infine, un algoritmo è da preferirsi ad un altro se è più efficiente, ovvero se richiede l'utilizzo di meno risorse. In generale, il costo di un algoritmo si può misurare in termini di memoria occupata (per i dati, per i risultati intermedi e per quelli finali) e/o in termini di tempo di esecuzione. D'altra parte, il tempo è ritenuto spesso una risorsa più preziosa perché non riutilizzabile e quindi in generale l'efficienza di un algoritmo numerico viene misurata tramite il suo costo in termini di tempo di esecuzione.

1.4.1 Costo computazionale

Il costo computazionale di un algoritmo è una misura teorica dell'efficienza, volta a dare un'idea del tempo di calcolo necessario alla sua esecuzione, senza di fatto eseguirlo. In generale il costo computazionale è espresso in termini di numero di operazioni coinvolte, il quale a sua volta è spesso

espresso in termini della "dimensione del problema". Abbiamo ad esempio visto nel paragrafo precedente due algoritmi per il calcolo del valore di un polinomio: il confronto fra i due è stato fatto in base al numero di operazioni aritmetiche elementari, il quale a sua volta dipendeva dal grado del polinomio, ovvero in ultima analisi dal numero dei coefficienti.

Il modo di misurare il costo computazionale degli algoritmi varia fortemente da settore a settore del Calcolo numerico. Ad esempio nel Capitolo 3 ci occuperemo di metodi per risolvere equazioni non lineari del tipo $f(x) = 0$. In questo caso il costo computazionale dei diversi algoritmi si misura in termini del numero di valutazioni della funzione f (ed eventualmente della sua derivata) necessarie per risolvere il problema. Vedremo invece nel Capitolo 4 che il costo degli algoritmi che risolvono problemi di algebra lineare è stimato in termini delle dimensioni delle matrici e dei vettori coinvolti e spesso è un costo asintotico, teso a misurare l'efficienza per problemi di grandi dimensioni: se un algoritmo richiede per esempio $3n^3 + 2n^2 + 8n$ operazioni aritmetiche, si dirà che il suo costo è dell'ordine di n^3, ovvero $\mathcal{O}(n^3)$.

Ci teniamo a sottolineare che le stime del costo computazionale di un algoritmo sono sempre stime teoriche, che non tengono conto di fattori estremamente importanti dal punto di vista pratico, quali ad esempio: il linguaggio di programmazione usato per realizzare l'algoritmo, la versione specifica del linguaggio utilizzata, le istruzioni usate per scrivere il programma in presenza di diverse alternative, l'architettura della macchina su cui il programma viene eseguito. Tutti questi fattori influenzano fortemente il tempo effettivo di esecuzione, rendendo le stime del costo computazionale spesso puramente teoriche. Da questo punto di vista il modo migliore di confrontare algoritmi diversi è in generale il metodo sperimentale.

2

ARITMETICA DEL CALCOLATORE ELETTRONICO

2.1 Memorizzazione dei numeri

L'elemento di base della memoria del calcolatore elettronico è un dispositivo che può assumere soltanto due stati fisici (acceso/spento, carico/scarico, on/off..); i due stati fisici sono convenzionalmente rappresentati con le cifre 0 e 1 e il dispositivo stesso è chiamato *bit*, da binary digit, ovvero cifra binaria. Da questo consegue che tutte le informazioni devono essere memorizzate come sequenze di 0 e 1. L'unità minima di memorizzazione è il *byte*, raggruppamento di 8 bits consecutivi. Un byte può assumere 256 diverse configurazioni e questo numero è sufficiente per codificare, ad esempio, tutti i caratteri che sono presenti su una tastiera; così, in un file di puro testo, ogni carattere (lettera dell'alfabeto minuscola o maiuscola, segno di punteggiatura, cifra,...) viene rappresentato in un byte mediante un sistema di codifica noto come ASCII (American Standard Code for Information Interchange) che ad ogni carattere associa una diversa configurazione del byte. Le informazioni più complesse, come le istruzioni di un file eseguibile o i numeri, vengono memorizzate usando raggruppamenti di più bytes. A noi interessa vedere come sono codificati i numeri.

Tipicamente i numeri vengono memorizzati usando il sistema binario, scelta ovvia dal momento che gli unici simboli utilizzabili sono "0" e "1". Questo presuppone l'utilizzo di algoritmi di conversione dalla rappresentazione decimale di un numero a quella binaria e viceversa, utilizzati dal calcolatore in fase di lettura dei dati e scrittura dei risultati rispettivamente. I dettagli di questi algoritmi, e più in generale i dettagli dell'aritmetica binaria, esulano dagli scopi di questo libro. Nondimeno ricordiamo che, esattamente come nella rappresentazione decimale, anche in quella binaria si utilizza la notazione posizionale, nella quale ogni cifra assume un valore diverso a seconda della posizione che occupa nella stringa di cifre che identificano il numero. Ad esempio, con ovvio significato dei simboli,

$$
\begin{array}{lcl}
(1011)_2 & = & 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \\
(100010)_2 & = & 1 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 \\
(0.11)_2 & = & 0 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2}.
\end{array}
$$

Esplicitando la notazione posizionale dei numeri binari otteniamo anche il loro equivalente decimale. Così $(1011)_2 = (11)_{10}$, $(100010)_2 = (34)_{10}$ e $(0.11)_2 = (0.75)_{10}$.

Il fatto principale da tenere in mente è che nel calcolatore ad ogni numero viene riservato un gruppo di bytes di lunghezza prefissata che noi chiameremo locazione di memoria. Da questo fatto discendono notevoli conseguenze che esamineremo nei prossimi paragrafi. Le regole che sovrintendono alla memorizzazione dei numeri nel calcolatore sono fissate attraverso uno standard pubblicato nel 1985, noto come IEEE binary floating point standard [1], nel quale si stabiliscono anche le regole per eseguire le operazioni aritmetiche. In questo capitolo esporremo i concetti principali che sono indispensabili per capire le problematiche tipiche del Calcolo numerico, che nascono quando si usa il calcolatore per risolvere problemi matematici. I dettagli che noi trascuriamo possono essere trovati in [29].

2.1.1 Numeri interi

Solitamente i numeri interi sono memorizzati in locazioni composte da 4 bytes, ovvero 32 bits. L'idea più naturale per usare queste locazioni sarebbe quella di memorizzare i numeri come abitualmente facciamo noi quando li scriviamo: nel primo bit si codifica il segno (ad esempio 0 per i numeri positivi e 1 per i negativi) e nei successivi il valore assoluto del numero, inserendo tanti 0 in testa quanti ce ne vogliono per riempire 31 bits. Con questa regola il numero $(1011)_2$ verrebbe memorizzato come

$$00000000000000000000000000001011 \tag{2.1}$$

e il numero $-(1011)_2$ sarebbe invece

$$10000000000000000000000000001011.$$

In realtà questa modalità viene usata soltanto per i numeri positivi, per i quali possiamo quindi già trarre la seguente conclusione:

Il numero intero positivo più grande che può essere memorizzato in un calcolatore è il numero $(11\ldots11)_2$ composto da 31 cifre tutte uguali a 1, a cui corrisponde la rappresentazione

$$01111111111111111111111111111111.$$

[1] IEEE è un acronimo per Institute for Electrical and Electronics Engineers; il significato del termine floating point sarà chiarito più avanti.

Il valore di questo numero, espresso in base 10, è

$$1 \times 2^{30} + 1 \times 2^{29} + \ldots + 1 \times 2^1 + 1 \times 2^0 = 2^{31} - 1 = 2147483647.$$

La memorizzazione dei numeri negativi segue una regola diversa che permette di eseguire più velocemente le operazioni aritmetiche [29]. Essi vengono memorizzati sotto forma di complemento a 2: al posto del numero $-n$, con $n > 0$, viene memorizzato il complementare di n rispetto a 2^{32}, che è dato da $2^{32} - n$.

L'esempio seguente mostra che il complementare rispetto a 2^{32} di un numero positivo può essere facilmente ottenuto senza bisogno di memorizzare 2^{32} stesso, numero troppo grande per essere memorizzato su 32 bits. Per facilitare la comprensione dell'esempio ricordiamo una semplice regola sulla somma binaria: $(1)_2 + (1)_2 = (10)_2 = (2)_{10}$.

Esempio 2.1.1. Consideriamo il numero decimale 11 la cui rappresentazione in macchina è la (2.1). Per ottenere la rappresentazione di -11 si deve calcolare la differenza $2^{32} - 11$. Osserviamo che $2^{32} = 2^{32} - 1 + 1$ e che la rappresentazione binaria di $2^{32} - 1$ è data da 32 cifre tutte uguali a 1. Il calcolo da fare può allora essere riscritto come

$$(11111111111111111111111111111111)_2 + (1)_2 - (1011)_2.$$

Questa trasformazione è utile perché la sottrazione

$$(11111111111111111111111111111111)_2 - (1011)_2$$

è facilmente realizzabile in macchina, dove

$$\begin{array}{l}(11111111111111111111111111111111)_2 - \\ (00000000000000000000000000001011)_2 = \\ (11111111111111111111111111110100)_2.\end{array}$$

In pratica il risultato si ottiene invertendo logicamente le cifre del sottraendo, ossia portando a 0 tutti i bits uguali a 1 e a 1 tutti quelli uguali a 0. A questo punto si deve sommare il numero $(1)_2$ e si ottiene:

$$\begin{array}{l}(11111111111111111111111111110100)_2 + \\ (00000000000000000000000000000001)_2 = \\ (11111111111111111111111111110101)_2.\end{array}$$

Quindi la rappresentazione in memoria di $= -11$ è

$$11111111111111111111111111110101.$$

■

Non è difficile rendersi conto di un'importante conseguenza di questo modo di memorizzare i numeri interi negativi: poichè il primo bit per i numeri positivi è uguale a 0, il primo bit dei numeri negativi è uguale a 1. La seconda conseguenza è che

il numero intero negativo più piccolo che può essere memorizzato in un calcolatore è -2^{31}, a cui corrisponde la rappresentazione

$$10000000000000000000000000000000.$$

Riassumendo, in una locazione da 32 bits può essere memorizzato ogni numero intero n tale che

$$-2^{31} \leq n \leq 2^{31} - 1. \tag{2.2}$$

Viceversa, qualunque numero intero che non soddisfa questi vincoli non può essere memorizzato. Un qualunque tentativo di memorizzare un numero intero che esce dai limiti (2.2) dà luogo a una situazione di errore chiamata overflow, dal verbo inglese "to overflow" che significa "straboccare". Lo standard IEEE di cui abbiamo parlato all'inizio del capitolo non stabilisce regole specifiche per il trattamento dell'overflow in aritmetica intera e quindi la reazione può variare a seconda del linguaggio di programmazione che si sta usando e talvolta anche a seconda della particolare realizzazione del linguaggio. La reazione più gradita sarebbe ovviamente l'invio di un messaggio che segnali il problema seguito eventualmente dall'interruzione dell'esecuzione del programma. Una reazione meno gradita che spesso si incontra è invece descritta qui di seguito.

Supponiamo che durante l'esecuzione di un programma ci si trovi a calcolare la somma $(2^{31} - 1) + 1$ il cui risultato, 2^{31}, è troppo grande per poter essere memorizzato. Seguendo le regole dell'addizione in aritmetica binaria si ottiene

$$\begin{array}{l} 01111111111111111111111111111111+ \\ 00000000000000000000000000000001 = \\ 10000000000000000000000000000000. \end{array}$$

Il risultato è corretto, ma viene frainteso se non sono previsti particolari controlli sulle situazioni di overflow: poiché il primo bit è uguale a 1, il risultato viene interpretato come numero negativo, per l'esattezza come -2^{31}, con la conseguenza paradossale che "$(2^{31} - 1) + 1$ fa -2^{31}".

Questo rende l'aritmetica intera non affidabile, tanto che in alcuni linguaggi di programmazione, come ad esempio in MATLAB, essa non è utilizzata e si opera esclusivamente con i numeri reali (di cui evidentemente gli interi sono un sottinsieme) che sono memorizzati con una modalità completamente diversa.

2.1.2 Numeri reali

Un numero reale può essere memorizzato in una locazione composta da 4 bytes (32 bits) o da 8 bytes (64 bits). Nel primo caso si dice che il numero è memorizzato in precisione semplice o singola; nel secondo caso si parla invece di memorizzazione in precisione doppia. Il senso di queste espressioni e in particolare l'utilizzo del termine "precisione" saranno chiariti strada facendo. Tutti i linguaggi di programmazione comunemente usati per il calcolo scientifico prevedono la possibilità di scegliere l'una o l'altra forma di memorizzazione per i valori che una grandezza reale può assumere. Alcuni linguaggi, come il MATLAB, privilegiano la memorizzazione in precisione doppia; altri, come il FORTRAN, privilegiano invece la semplice precisione. Con il termine "privilegiano" intendiamo dire che questa è la scelta operata in assenza di esplicite indicazioni da parte del programmatore.

Qualunque sia il numero di bits usati, la memorizzazione dei numeri reali si basa sulla cosiddetta notazione esponenziale (o scientifica) normalizzata, nella quale un qualunque numero x viene espresso in base b nella forma

$$x = \pm c_0.c_1c_2\ldots \times b^q$$

dove $c_0, c_1, c_2, \ldots$ sono cifre del sistema di numerazione in base b con $c_0 \neq 0$ (in questo consiste la normalizzazione) e q è un intero espresso in base b. Per esemplificare, la rappresentazione scientifica normalizzata di 35.48 è 3.548×10^1 e quella di 0.0187 è 1.87×10^{-2}.

La parte $c_0.c_1c_2\ldots$ è detta mantissa [2] di x e può essere formata da un numero qualunque di cifre, anche da infinite cifre; q è detto esponente o caratteristica. Nel calcolatore, poiché la base b è 2, le cifre della mantissa sono 0 oppure 1 e q è un intero binario.

Quando un numero reale deve essere memorizzato nel calcolatore, la locazione ad esso destinata viene suddivisa in campi di lunghezza prefissata nel modo seguente:

Precisione semplice:
- 1 bit per il segno (0 per i numeri positivi e 1 per i numeri negativi);
- 8 bits per la caratteristica;
- 23 bits per la mantissa.

Precisione doppia:
- 1 bit per il segno (0 per i numeri positivi e 1 per i numeri negativi);
- 11 bits per la caratteristica;
- 52 bits per la mantissa.

[2] Lo standard IEEE sconsiglia l'uso del termine "mantissa" in questo contesto perché non è coerente con l'uso dello stesso termine per indicare la parte frazionaria di un logaritmo, uso che risale al 18^o secolo. È stato allora coniato in inglese il neologismo "significand", da usarsi al posto di "mantissa". Nonostante questo, molti informatici e analisti numerici continuano ad usare il termine "mantissa".

La caratteristica
Per motivi tecnici la cui comprensione esula dai nostri scopi, la caratteristica q viene memorizzata nel campo ad essa riservato secondo modalità diverse da quelle viste nel paragrafo precedente per i numeri interi. Consideriamo il caso dei numeri in precisione semplice, nel qual caso alla caratteristica sono riservati 8 bits. Un gruppo di 8 bits può avere $2^8 = 256$ configurazioni diverse, fra cui le configurazioni "estreme" sono 00000000 e 11111111, interpretabili come rappresentazione binaria dei numeri interi 0 e 255. Tutte le configurazioni intermedie sono interpretabili a loro volta come numeri interi fra 1 e 254. Tenendo conto di questo, la regola per la memorizzazione della caratteristica dei numeri reali è la seguente. Le configurazioni estreme sono riservate per codificare lo 0 e altri numeri speciali, come ∞ e NaN [3]. Tutte le altre configurazioni sono usate per memorizzare la caratteristica traslata $\hat{q} = q + 127$. Dovendo essere $1 \leq \hat{q} \leq 254$, un numero x non può essere memorizzato in precisione semplice se la sua caratteristica binaria q non è compresa fra -126 e 127. Ne consegue che un numero x può essere memorizzato soltanto se appartiene a un range prestabilito:

$$\mathbb{U} \leq |x| \leq \mathbb{O} \tag{2.3}$$

dove $\mathbb{U}$ e $\mathbb{O}$ sono dette rispettivamente soglia di underflow e soglia di overflow. $\mathbb{U}$ e $\mathbb{O}$ dipendono dalle limitazioni sulla caratteristica viste prima, da cui segue che

$$\mathbb{U} \simeq 2^{-126} \simeq 10^{-38}$$

e

$$\mathbb{O} \simeq 2^{127} \simeq 10^{38}.$$

Qualunque tentativo di memorizzare numeri reali non appartenenti al range (2.3) dà luogo ad una situazione di errore, di underflow o di overflow a seconda di quale delle due limitazioni è violata. Tipiche reazioni all'overflow sui numeri reali sono l'arresto del programma oppure la codifica speciale come $\pm\infty$ del risultato; l'underflow viene spesso risolto sostituendo 0 al numero non memorizzabile.

Quanto detto finora vale qualitativamente anche per la memorizzazione dei numeri reali in precisione doppia. Ricordiamo che in questo caso i bits dedicati alla caratteristica sono 11. Essendo $2^{11} = 2048$, le configurazioni intermedie di 11 bits corrispondono alla rappresentazione binaria dei numeri interi compresi fra 1 e 2046 e quindi la caratteristica traslata che viene memorizzata è $\hat{q} = q + 1023$. Pertanto un numero può essere memorizzato in doppia precisione se $-1022 \leq q \leq 1023$. Le soglie di underflow e di overflow diventano rispettivamente

$$\mathbb{U} \simeq 10^{-308} \text{ e } \mathbb{O} \simeq 10^{308}.$$

[3] NaN è un acronimo per Not a Number, che si ottiene ad esempio quando si tenta di calcolare una forma indeterminata come $0/0$.

D'ora in avanti supporremo sempre per semplicità di avere a che fare con numeri reali di ordine di grandezza accettabile per la macchina.

La mantissa
Osserviamo come prima cosa che nella rappresentazione scientifica normalizzata di un numero reale binario diverso da 0, la prima cifra c_0 è sicuramente uguale a 1. Quindi essa non viene mai memorizzata e i bits destinati alla mantissa sono usati per la memorizzazione delle cifre successive da c_1 in poi. Talvolta si dice che le macchine usano un bit nascosto per la memorizzazione di c_0. D'ora in avanti indicheremo con m il numero di bits dedicati alla mantissa, compreso il bit nascosto; allora $m = 24$ per la precisione semplice e $m = 53$ per la precisione doppia.

Fissato m, sono sicuramente memorizzabili senza problemi tutti i numeri con una mantissa binaria nella quale $c_m = c_{m+1} = \ldots = 0$; questi numeri sono chiamati numeri di macchina o numeri floating point [4].

Esempi particolarmente importanti di numeri floating point, qualunque sia m, sono le potenze del 2. Infatti, qualunque esponente E si consideri, la mantissa binaria normalizzata di 2^E è composta da $c_0 = 1, c_1 = c_2 = \ldots = 0$.

Esempi altrettanto significativi di numeri non floating point sono invece le potenze negative del numero (decimale) 10. Infatti il numero 10^{-1} diventa periodico quando lo si trasforma in binario:

$$(0.1)_{10} = (0.0001100110011\ldots)_2;$$

la mantissa $1.100110011\ldots$ non può ovviamente essere memorizzata qualunque sia m. Per poter memorizzare questo numero, così come qualunque altro numero x la cui rappresentazione binaria ha una mantissa con più di m cifre, si deve procedere ad una approssimazione tagliando la mantissa per ridurla a m cifre. Questa operazione può avvenire con due diverse modalità:

il troncamento (in inglese "round towards zero") che consiste nel memorizzare soltanto le prime m cifre qualunque sia l'entità della parte trascurata $c_m c_{m+1}..$;

l'arrotondamento (in inglese "round to nearest') che consiste nell'approssimare il numero x con il numero di macchina più vicino.

Lo standard IEEE consiglia l'uso dell'arrotondamento che, nei dettagli, funziona nel modo seguente. Indichiamo con x^- e x^+ i due numeri di

[4] Questo secondo nome deriva dal fatto che la rappresentazione dei numeri basata sulla notazione scientifica normalizzata si chiama in inglese rappresentazione floating point (in italiano virgola mobile) con riferimento al punto decimale che viene spostato fino ad ottenere la forma normalizzata del numero.

macchina più vicini ad x, con $x^- < x < x^+$. Se la prima cifra che non può essere memorizzata, c_m, è uguale a 0, allora x viene approssimato da x^- (come nel troncamento); se $c_m = 1$ e almeno una delle cifre successive è uguale a 1, allora x viene approssimato da x^+; se infine $c_m = 1$ e tutte le cifre successive sono uguali a 0, allora x risulta equidistante da x^- e x^+ e viene approssimato con quello fra i due numeri x^- e x^+ la cui cifra finale è 0. Questa regola è stata scelta per evitare di creare una "direzione preferenziale" nell'operazione di arrotondamento, come spiegato in [18].

Esempio 2.1.2. Consideriamo $x = (1.00001)_2 = 2 + 2^{-5}$, che non può essere memorizzato su una macchina con $m = 5$ bits di mantissa. I due numeri di macchina più vicini sono $x^- = (1.0000)_2 = 2$ e $x^+ = (1.0001)_2 = 2 + 2^{-4}$ ed x è equidistante da questi due numeri, essendo $x - x^- = x^+ - x = 2^{-5}$. La regola precedentemente citata ci dice che x verrà approssimato in macchina da x^-. ■

In conseguenza del fatto che m è un numero finito e prefissato, l'insieme dei numeri di macchina è un insieme discreto. La distanza fra due numeri di macchina successivi aumenta con l'aumentare della distanza da 0. Consideriamo ad esempio $x = 1 = (1.00\ldots00)_2$; il numero di macchina successivo è $(1.00\ldots01)_2 = 1 + 2^{-(m-1)}$, con una distanza pari a $2^{-(m-1)}$. Se invece consideriamo $x = 2^3 = (1.00\ldots00)_2 \times 2^2$, il numero di macchina successivo è $(1.00\ldots01)_2 \times 2^2 = [2^3 + 2^{-(m-1)}] \times 2^2$, con un salto di $2^{-(m-1)} \times 2^2$.

Il numero $2^{-(m-1)}$, cioè la distanza fra il numero 1 e il successivo numero di macchina è spesso indicato come l'epsilon di macchina; esso dà un'idea della "taratura" dello strumento che stiamo usando e sarà indicato nel seguito con il simbolo ϵ_m.

2.1.3 Errori di arrotondamento e precisione di macchina

Abbiamo visto che in generale la memorizzazione di un numero reale x richiede un'approssimazione della mantissa e quindi il numero effettivamente memorizzato è diverso da x. Questo numero viene indicato con il simbolo $round(x)$ o, più tradizionalmente, $fl(x)$: "round" sta per "arrotondamento", "fl" sta per "floating". L'errore $|x - fl(x)|$ viene chiamato errore di arrotondamento (in inglese round-off error) qualunque sia la modalità usata per ottenere $fl(x)$.

Il fatto che i numeri reali nel calcolatore vengono approssimati ha conseguenze molto importanti. La prima osservazione che possiamo fare è la seguente: poiché in generale un algoritmo in macchina non opera su dati esatti, non potremo né dovremo mai aspettarci che fornisca risultati esatti. La seconda osservazione, che svilupperemo nei paragrafi successivi, è che in generale dovremo aspettarci la nascita di errori di arrotondamento anche nell'esecuzione delle operazioni aritmetiche, dal momento che il risultato di

un'operazione fra numeri di macchina non è necessariamente un numero di macchina.

A questo punto è importante trovare una misura dell'accuratezza della macchina, se non altro per poter correttamente interpretare e valutare i risultati di un qualunque algoritmo eseguito sul calcolatore. Un risultato sarà da considerarsi accettabile se l'errore da cui è affetto rientra nei limiti di questa accuratezza.

Iniziamo con il cercare una limitazione superiore degli errori di arrotondamento che la macchina commette nel memorizzare i numeri. Poiché la caratteristica dei numeri viene memorizzata in modo esatto (salvo problemi di overflow o underflow) e l'approssimazione riguarda soltanto la mantissa, dobbiamo trovare una limitazione che non coinvolga la caratteristica. A questo scopo osserviamo che, dato un qualunque numero reale $x \neq 0$, possiamo considerare l'errore di arrotondamento assoluto

$$e_a(x) = |x - fl(x)|$$

e l'errore di arrotondamento relativo

$$e_r(x) = |x - fl(x)|/|x| = e_a(x)/|x|.$$

Chiediamoci quale di questi due modi di misurare l'errore è più adatto ai nostri fini. Per rispondere a questa domanda facciamo un esempio.

Esempio 2.1.3. Supponiamo di lavorare con un calcolatore che usa la base 10 con $m = 5$ e opera per troncamento. In queste ipotesi $\epsilon_m = 10^{-4}$.

Sia $x = 3.45987 \times 10^2$, a cui corrisponde $fl(x) = 3.4598 \times 10^2$; allora gli errori di arrotondamento assoluto e relativo sono

$$e_a(x) = 0.00007 \times 10^2 \ \text{ e } \ e_r(x) = \frac{0.00007 \times 10^2}{3.45987 \times 10^2} = \frac{7 \times 10^{-5}}{3.45987}.$$

Ora consideriamo un altro numero z con la stessa mantissa di x ma caratteristica diversa, ad esempio $z = 3.45987 \times 10^{-3}$; essendo $fl(z) = 3.4598 \times 10^{-3}$, si ha

$$e_a(z) = 0.00007 \times 10^{-3} \ \text{ e } \ e_r(z) = \frac{0.00007 \times 10^{-3}}{3.45987 \times 10^{-3}} = \frac{7 \times 10^{-5}}{3.45987}.$$

Possiamo osservare che $e_a(x) \neq e_a(z)$, mentre $e_r(x) = e_r(z)$. Questo non ci sorprende perché l'errore assoluto risente della caratteristica dei numeri, mentre l'errore relativo è legato esclusivamente all'approssimazione della mantissa e non dipende in alcun modo dalla caratteristica. In particolare, l'errore relativo è minore di 10^{-4}. Si può facilmente vedere che, mantenendo le stesse ipotesi di lavoro, avremmo ottenuto $e_r(x) < 10^{-4}$ per qualunque $x \neq 0$; inoltre, saremmo arrivati molto vicini a questa limitazione considerando per esempio $x = 1.00009$, che porta a $e_r(x) = \frac{9\times10^{-5}}{1.00009}$. Se avessimo supposto di operare per arrotondamento avremmo trovato una limitazione leggermente più piccola degli errori relativi, ovvero 0.5×10^{-4}, a cui si sarebbe arrivati molto vicini con $x = 1.00005$. ■

Questo esempio ci induce a cercare di quantificare l'accuratezza della macchina attraverso una limitazione degli errori di arrotondamento relativi. Tornando agli elaboratori reali e procedendo come nell'esempio 2.1.3, si può dimostrare il seguente risultato generale:

Teorema 2.1.1. Sia x un qualunque numero reale diverso da zero e m il numero di cifre di mantissa usate dal calcolatore. Allora

$$e_r(x) < \epsilon_m,$$

dove $\epsilon_m = 2^{-(m-1)}$ è l'epsilon di macchina. Inoltre, se il calcolatore opera per arrotondamento si ha

$$e_r(x) < \frac{1}{2}\epsilon_m.$$

In virtù di questo risultato ϵ_m è detto anche precisione di macchina. In precisione semplice ($m = 24$) si ha

$$\epsilon_m = 2^{-23} \simeq 1.19 \times 10^{-7}, \tag{2.4}$$

mentre in precisione doppia ($m = 53$) si ha

$$\epsilon_m = 2^{-52} \simeq 2.22 \times 10^{-16}. \tag{2.5}$$

È utile e interessante fare la seguente osservazione. Dato un numero x, il rapporto fra il numero di cifre di mantissa della sua rappresentazione decimale e quello della sua rappresentazione binaria è mediamente 1/3: ad una cifra decimale corrispondono mediamente tre cifre binarie. Allora possiamo dire che lavorare in precisione semplice ($m = 24$) corrisponde in un certo senso a lavorare in base 10 con circa $24/3 = 8$ cifre di mantissa. Questo punto di vista è coerente con il fatto che in queste ipotesi di lavoro si avrebbe una precisione di macchina uguale a $10^{-(8-1)} = 10^{-7}$, circa lo stesso valore della (2.4). Analogo discorso possiamo fare nel caso della precisione doppia: poiché $m = 53$, è come se lavorassimo in base 10 con 17 cifre di mantissa, da cui $\epsilon_m \simeq 10^{-16}$.

La relazione che intercorre fra un numero reale x e la sua rappresentazione $fl(x)$ viene spesso scritta nel modo seguente:

$$fl(x) = x(1 + \epsilon), \tag{2.6}$$

dove ϵ coincide a meno del segno con $e_r(x)$. Questa relazione discende direttamente dalla definizione di errore relativo di arrotondamento. Tenendo conto della (2.6), il teorema 2.1.1 può essere enunciato in una forma leggermente diversa:

Teorema 2.1.2. Sia x un qualunque numero reale diverso da zero e m il numero di cifre di mantissa usate dal calcolatore. Allora $fl(x) = x(1+\epsilon)$, dove

$$|\epsilon| < \epsilon_m.$$

Inoltre, se il calcolatore opera per arrotondamento si ha

$$|\epsilon| < \frac{1}{2}\epsilon_m.$$

2.2 Operazioni di macchina

Quando si esegue un'operazione aritmetica fra due numeri di macchina, può succedere che il risultato non sia un numero di macchina. Ad esempio, supponendo di lavorare in base 10 con $m = 5$, i due numeri $x = 3.4128 \times 10^2$ e $y = 8.2599 \times 10^2$ sono numeri di macchina, ma la loro somma 11.6727×10^2 o il loro prodotto 28.18938727×10^4 non lo sono perché hanno una mantissa composta da piú di 5 cifre. Questo significa che in generale le operazioni aritmetiche eseguite dal calcolatore introducono un errore dovuto alla necessità di approssimare il risultato. Lo standard IEEE impone che questo errore sia compatibile con la precisione di macchina, ovvero che valga la relazione

$$x \odot y = fl(x \cdot y), \tag{2.7}$$

dove il simbolo $\cdot$ indica una qualunque operazione elementare $(+, -, \times, /)$ e $\odot$ indica la corrispondente operazione di macchina. La (2.7) equivale a

$$x \odot y = (x \cdot y)(1+\epsilon)$$

dove ϵ soddisfa la relazione $|\epsilon| < \epsilon_m$ e rappresenta l'errore relativo introdotto dall'operazione di macchina. Per poter rispettare il vincolo (2.7) vengono utilizzati dei registri, che sono delle locazioni di memoria nelle quali il campo destinato alla mantissa è più lungo di quello delle locazioni usuali atte alla memorizzazione dei numeri reali. Senza entrare nei dettagli, illustriamo quanto detto nel caso dell'addizione. Salvo avviso contrario, negli esempi che seguono supporremo di lavorare in base 10 con $m = 5$, da cui segue che la precisione di macchina è $\epsilon_m = 10^{-4}$. Supporremo inoltre di operare per arrotondamento.

Esempio 2.2.1. Vogliamo sommare $x = 3.4128 \times 10^2$ e $y = 8.2599 \times 10^2$. Supponendo di disporre di un registro con almeno 6 cifre di mantissa nel quale memorizzare il risultato 11.6727×10^2 non normalizzato, l'addizione in macchina avviene nel modo seguente:

$$\begin{aligned} x \oplus y &= fl(x+y) = fl(3.4128 \times 10^2 + 8.2599 \times 10^2) \\ &= fl(11.6727 \times 10^2) = 1.1673 \times 10^3. \end{aligned}$$

L'errore relativo nel risultato è

$$\frac{|11.6727 \times 10^2 - 1.1673 \times 10^3|}{|11.6727 \times 10^2|} \simeq 2.6 \times 10^{-5} < \epsilon_m.$$

■

Esempio 2.2.2. Calcoliamo $x \oplus y$, con $x = 1.5728 \times 10^2$ e $y = 2.4100 \times 10^{-2}$. In questo caso x e y non hanno la stessa caratteristica e per eseguire l'addizione occorre modificare uno dei due in modo da ricondursi a sommare due numeri con uguale esponente. A questo scopo si riscrive l'addendo più piccolo, y, riconducendolo ad una forma non normalizzata in cui la caratteristica è uguale a quella di x. Per poter eseguire questo spostamento supponiamo di disporre di un registro con campo di mantissa di almeno 9 cifre.

$$\begin{aligned} x \oplus y &= fl(x+y) = fl(1.5728 \times 10^2 + 2.4100 \times 10^{-2}) \\ &= fl(1.5728 \times 10^2 + 0.00024100 \times 10^2) = fl(1.57304100 \times 10^2) \\ &= 1.5730 \times 10^2. \end{aligned}$$

L'errore relativo nel risultato è

$$\frac{|1.57304100 \times 10^2 - 1.5730 \times 10^2|}{|1.57304100 \times 10^2|} \simeq 2.6 \times 10^{-5} < \epsilon_m.$$

■

Esempio 2.2.3. Calcoliamo $x \oplus y$, con $x = 8.2342 \times 10^2$ e $y = 3.0000 \times 10^{-3}$. Procedendo come nell'esempio precedente e ipotizzando di disporre di un registro con almeno 10 cifre di mantissa, si ottiene:

$$\begin{aligned} x \oplus y &= fl(x+y) = fl(8.2342 \times 10^2 + 3.0000 \times 10^{-3}) \\ &= fl(8.2342 \times 10^2 + 0.000030000 \times 10^2) = fl(8.234230000 \times 10^2) \\ &= 8.2342 \times 10^2. \end{aligned}$$

L'errore relativo nel risultato è ancora minore di ϵ_m. La particolarità di questo esempio consiste nel fatto che sommando y ad x si ottiene come risultato x, situazione che, in assenza di approssimazioni, varrebbe soltanto per $y = 0$. Come si è soliti dire, y è talmente piccolo rispetto a x che la macchina "non lo sente" sommandolo a x. ■

Gli esempi ci dicono che l'aritmetica usata all'interno del calcolatore è diversa da quella usata "fuori dal calcolatore", dove, in linea di principio, è possibile portare avanti dei calcoli senza fare approssimazioni (quante

volte capita a ciascuno di noi nel risolvere un problema o nel calcolare un'espressione di lasciare indicato π, o $2/3$, o $\sqrt{2}$ senza effettivamente sostituire il valore?). Indicheremo d'ora in avanti l'aritmetica del calcolatore con l'espressione aritmetica floating point o aritmetica in precisione finita. Per contro, l'aritmetica esatta sarà indicata talvolta come aritmetica in precisione infinita.

L'esempio 2.2.3 ci ha mostrato che in aritmetica floating point va cambiato il concetto di elemento neutro rispetto alla somma, perché può accadere che la somma fra due numeri sia uguale a uno dei due anche se l'altro non è uguale a 0. Gli esempi successivi ci mostreranno che in aritmetica floating point non vale la proprietà commutativa dell'addizione.

Esempio 2.2.4. Vogliamo calcolare la somma di tre numeri a, b e c usando l'algoritmo 1.3.1 per il calcolo delle sommatorie in un'aritmetica con base 10, $m = 4$ (da cui segue $\epsilon_m = 10^{-3}$) e troncamento. Se $a = 2000$, $b = 2.5$ e $c = 7.8$, il risultato esatto è 2010.3. Avendo presente che in aritmetica esatta vale la proprietà commutativa, possiamo pensare di eseguire il calcolo in diversi modi, ad esempio: $a + b + c$ e $b + c + a$. Nella nostra aritmetica floating point, otteniamo i seguenti risultati.

$$\begin{aligned} a \oplus b \oplus c &= fl(2.000 \times 10^3 + 2.500 \times 10^0) \oplus 7.800 \times 10^0 \\ &= fl(2.000 \times 10^3 + 0.002500 \times 10^3) \oplus 7.800 \times 10^0 \\ &= 2.002 \times 10^3 \oplus 7.800 \times 10^0 \\ &= fl(2.002 \times 10^3 + 0.007800 \times 10^3) \\ &= 2.009 \times 10^3. \end{aligned}$$

$$\begin{aligned} b \oplus c \oplus a &= fl(2.500 \times 10^0 + 7.800 \times 10^0) \oplus 2.000 \times 10^3 \\ &= 1.030 \times 10^1 \oplus 2.000 \times 10^3 \\ &= fl(0.01030 \times 10^3 + 2.00 \times 10^3+) \\ &= 2.010 \times 10^3. \end{aligned}$$

La prima cosa che vediamo è che i due risultati sono diversi. Il primo, chiamiamolo R_1, è meno accurato del secondo, R_2: infatti R_1 è affetto da un errore relativo $|2010.3-2009|/2010.3 \simeq 6.5\times10^{-4}$ più grande di $|2010.3-2010|/2010.3 \simeq 1.5 \times 10^{-4}$. Non ci deve meravigliare il fatto che $R_1 \neq R_2$ perché questi due risultati sono frutto di due diverse sequenze di operazioni che danno luogo ad un diverso accumulo di errori di arrotondamento. In particolare nel calcolo di R_1 ci siamo trovati due volte a dover "incolonnare" numeri con caratteristiche diverse e troncare la mantissa del risultato con conseguente perdita di accuratezza. Nel calcolo di R_2 questa situazione si verifica una sola volta. L'aspetto positivo in questo esempio è che comunque entrambi i risultati sono da considerarsi accettabili perché affetti da un errore relativo minore di ϵ_m. ■

Il seguente esempio non riguarda più soltanto l'addizione, ma mostra un'altra situazione in cui due formule matematicamente equivalenti possono non esserlo quando si opera in aritmetica floating point.

Esempio 2.2.5. Vogliamo calcolare il punto di mezzo c dell'intervallo $[a, b]$ con $a = 6.510$ e $b = 6.512$, supponendo di lavorare in base 10 con $m = 4$ e troncamento. Sappiamo che il risultato esatto è $c = 6.511$.

Usando la formula classica $c = (a + b)/2$, dobbiamo prima di tutto calcolare la somma fra a e b ottenendo:

$$\begin{aligned} a \oplus b &= fl(6.510 \times 10^0 + 6.512 \times 10^0) \\ &= fl(13.022 \times 10^0) = 1.302 \times 10^1. \end{aligned}$$

Dividendo per 2 si ottiene $c = 6.510$; l'errore è accettabile, dal momento che $|6.510 - 6.511|/6.511 \simeq 1.5 \times 10^{-4} < \epsilon_m$; la cosa che appare strana, se così possiamo dire, è che il punto di mezzo calcolato risulta uguale al primo estremo dell'intervallo.

Passiamo ora ad usare un'altra formula: $c = a + (b - a)/2$. In questo caso dobbiamo fare prima di tutto la sottrazione:

$$\begin{aligned} b \ominus a &= fl(6.512 \times 10^0 - 6.510 \times 10^0) \\ &= fl(0.002 \times 10^0) = 2.000 \times 10^{-3}; \end{aligned}$$

dividendo per 2 questo risultato otteniamo 1.000×10^{-3} e l'ultima operazione da effettuare diventa

$$\begin{aligned} a \oplus 1.000 \times 10^-3 &= fl(6.510 \times 10^0 + 1.000 \times 10^{-3}) \\ &= fl(6.510 \times 10^0 + 0.001000 \times 10^0) \\ &= fl(6.511000 \times 10^0) = 6.511 \times 10^0. \end{aligned}$$

Il risultato in questo caso è esatto. ■

2.2.1 Un'altra definizione della precisione di macchina

Ponendoci dal punto di vista delle operazioni aritmetiche, si può introdurre una definizione della precisione di macchina diversa da quella data nel paragrafo 2.1.3, ma ad essa equivalente. Consideriamo a questo scopo il seguente esempio:

Esempio 2.2.6. Supponiamo di lavorare in aritmetica binaria con $m = 5$ cifre di mantissa, da cui segue $\epsilon_m = 2^{-(m-1)} = 2^{-4}$. Se eseguiamo la somma $1 + \epsilon_m$ otteniamo il seguente risultato:

$$\begin{aligned} 1 \oplus 2^{-4} &= fl(1.0000 \times 2^0 + 1.0000 \times 2^{-4}) \\ &= fl(1.0000 \times 2^0 + 0.0001 \times 2^0) \\ &= fl(1.0001 \times 2^0) \\ &= 1.0001 \times 2^0. \end{aligned}$$

Osserviamo che non abbiamo avuto bisogno di utilizzare un campo di mantissa più lungo per il risultato intermedio e che il contributo dell'addendo più piccolo si fa sentire proprio sull'ultimo bit.
Se ora eseguiamo la somma $1 + \frac{1}{2}\epsilon_m = 1 + 2^{-m} = 1 + 2^{-5}$, otteniamo

$$\begin{aligned} 1 \oplus 2^{-5} &= fl(1.0000 \times 2^0 + 1.0000 \times 2^{-5} \\ &= fl(1.0000 \times 2^0 + 0.00001 \times 2^0 \\ &= fl(1.00001 \times 2^0) \end{aligned}$$

Questa volta dobbiamo disporre di un registro con almeno 6 bits di mantissa per poter conservare il risultato intermedio 1.00001×2^0. Se supponiamo che la macchina operi per troncamento, vediamo subito che l'addendo più piccolo 2^{-m} non viene "sentito" e il risultato finale è 1.0000×2^0. D'altra parte, tenendo conto delle regole di arrotondamento stabilite dallo standard IEEE, si otterrebbe come risultato il numero 1 anche operando per arrotondamento (cfr. esempio 2.1.2). ■

L'esempio suggerisce la seguente definizione alternativa della precisione di macchina:

La precisione di macchina ϵ_m è il più piccolo numero floating point che possiamo sommare a 1 ottenendo un risultato maggiore di 1.

In formule, detto x un qualunque numero di macchina:

$$\begin{aligned} 1 \oplus x = fl(1 + x) > 1 \quad &\text{per ogni } \ x \geq \epsilon_m \\ 1 \oplus x = fl(1 + x) = 1 \quad &\text{per ogni } \ x < \epsilon_m. \end{aligned}$$

L'importanza di questa nuova definizione consiste nel fatto che essa porta in modo naturale ad un algoritmo per il calcolo di ϵ_m:

Algoritmo 2.2.1. Algoritmo per il calcolo della precisione di macchina.

Dati: nessuno
Risultati: ϵ_m
1. $x = 1$
2. $x = 0.5 \times x$
3. $y = 1 + x$
4. Se $y > 1$, allora:
   vai a 2.
   Fine scelta
5. $\epsilon_m = 2 \times x$
6. Fine

È facile vedere che dalla definizione ora data consegue che il numero floating point più piccolo che la macchina "sente" sommandolo a un dato numero $a > 0$ è $a \times \epsilon_m$. Essendo infatti $a + x = a \times (1 + x/a)$, il più piccolo valore di x/a "sentito" è ϵ_m.

2.3 Propagazione degli errori. Condizionamento e stabilità

Poiché i numeri reali in generale non sono esattamente rappresentabili in macchina e vengono approssimati, gli algoritmi realizzati sul calcolatore non operano di solito su dati esatti, ma su dati perturbati. Abbiamo anche visto che le operazioni aritmetiche in macchina sono una fonte di errori, sia pure minori di ϵ_m, e si può pensare che errori relativi di questa entità vengano generati anche nel calcolo di funzioni elementari quali ad esempio $e^x, log_{10}x, logx, \sqrt{x}, cosx, senx$, ecc. Nei linguaggi di programmazione queste funzioni vengono infatti realizzate mediante procedure che garantiscono la massima accuratezza del risultato, compatibilmente con la precisione di macchina; in altri termini, se x è un numero di macchina, il valore della funzione viene calcolato con un errore relativo minore di ϵ_m.

Le domande che sorgono spontanee sono le seguenti. Una volta che un errore è stato generato, per via dei dati o per via di un'operazione, come si propaga questo errore attraverso le operazioni successive previste da un algoritmo: viene trasmesso inalterato, viene amplificato, o viene ridotto? E come si combinano fra loro i diversi errori: si sommano, si annullano a vicenda, o una via intermedia fra queste? Rispondere a queste domande è fondamentale ai fini di capire quanto può essere accurato il risultato prodotto da un algoritmo e quanta accuratezza sia lecito attendersi.

Nel seguente esempio mostriamo un caso limite: due algoritmi diversi per il calcolo di una stessa quantità producono risultati completamente diversi, in conseguenza del fatto che in uno vengono generati degli errori di arrotondamento che si sommano fra loro, mentre nell'altro non esistono errori di arrotondamento.

Esempio 2.3.1. Supponiamo di lavorare in precisione semplice, quindi con 24 bits di mantissa. Dobbiamo calcolare $\sum_{i=1}^{n} a_i$, con $n = 2^{24} + 1$ e

$$a_1 = 1 \ , \ \ a_2 = a_3 = \ldots = a_n = 2^{-24},$$

o equivalentemente $\sum_{i=1}^{n} b_i$ con lo stesso n e

$$b_1 = b_2 = \ldots = b_{n-1} = 2^{-24} \ , \ \ b_n = 1.$$

In entrambi i casi i dati sono tutti numeri di macchina e il risultato esatto è 2, anch'esso un numero di macchina.

Lasciamo per esercizio la verifica del fatto che usando l'algoritmo 1.3.1 per il calcolo delle sommatorie si ottiene nel primo caso un risultato completamente sbagliato ed inaccettabile, 1, mentre nel secondo caso si ottiene il risultato esatto. Analizzando i calcoli si vede che nel primo caso ci si trova per 2^{24} volte a eseguire l'operazione $1 + 2^{-24}$, il cui risultato è 1 perché 2^{-24} è minore di $\epsilon_m = 2^{-23}$. L'accumulo di questi piccoli ed accettabili errori risulta fatale. Nel secondo caso la situazione precedente non si verifica

mai e tutte le operazioni avvengono fra numeri di macchina e producono risultati intermedi esattamente rappresentabili in macchina. ■

Lo studio della propagazione degli errori relativi attraverso gli algoritmi è un capitolo fondamentale del Calcolo numerico e in generale è uno studio molto complesso che va portato avanti algoritmo per algoritmo. In questo paragrafo ci limiteremo ad introdurre alcuni concetti fondamentali al riguardo.

Sia R il risultato esatto di un algoritmo e $\hat{R}$ un risultato approssimato; senza perdere in generalità, possiamo supporre che R e $\hat{R}$ abbiano lo stesso ordine di grandezza. Chiediamoci come possiamo valutare l'accuratezza di $\hat{R}$. Come già abbiamo notato parlando di errore di rappresentazione dei numeri reali, l'errore assoluto $e_a(\hat{R}) = |R - \hat{R}|$ porta in sé l'ordine di grandezza di R e $\hat{R}$ mentre l'errore relativo $e_r(\hat{R}) = |R - \hat{R}|/|R|$ dipende esclusivamente dalla differenza fra le mantisse di R e $\hat{R}$. Di conseguenza, l'accuratezza di $\hat{R}$ rispetto a R è espressa da $e_r(\hat{R})$. Per essere più precisi, l'ordine di grandezza di $e_r(\hat{R})$, misurato tramite la quantità $-log_{10}(e_r(\hat{R}))$, ci dice quante cifre uguali hanno approssimativamente R e $\hat{R}$ (stiamo ovviamente parlando delle cifre più significative, ovvero delle prime cifre della mantissa).

Ad esempio, dati $R = 3.141592 \times 10^2$ e $\hat{R} = 3.141589 \times 10^2$, si ha $-log_{10}(e_r(\hat{R})) \simeq 6.02$ e possiamo dire che i due numeri hanno approssimativamente le prime 6 cifre di mantissa uguali; abbiamo usato l'avverbio "approssimativamente" perché R e $\hat{R}$ non hanno esattamente 6 cifre uguali, ma questo succede arrotondando entrambi a 6 cifre. D'ora in avanti quando diremo che un numero ne approssima un altro a p cifre intenderemo dire che i due numeri hanno approssimativamente le prime p cifre di mantissa uguali. Per chiarire ulteriormente il concetto, consideriamo $R = 2.141457 \times 10^{-5}$ e $\hat{R} = 2.141480 \times 10^{-5}$; in questo caso $\hat{R}$ approssima R a 5 cifre, perché se li arrotondiamo entrambi a 5 cifre diventano uguali a 2.1415; a riprova abbiamo $-log_{10}(e_r(\hat{R})) \simeq 4.97$.

Esaminiamo ora le operazioni aritmetiche elementari, in particolare la moltiplicazione e l'addizione, per capire come si ripercuotono sul risultato eventuali errori sui dati. Per semplicità consideriamo le operazioni esatte invece che le operazioni di macchina.

Siano a e b due numeri approssimati rispettivamente da $\hat{a} = a(1 + \epsilon_a)$ e $\hat{b} = b(1 + \epsilon_b)$. In questa notazione, ϵ_a e ϵ_b rappresentano a meno del segno gli errori relativi: se $\hat{a}$ e $\hat{b}$ coincidono con $fl(a)$ e $fl(b)$, allora $|\epsilon_a|$ e $|\epsilon_b|$ sono minori di ϵ_m; se invece l'errore deriva da altre cause, i due errori relativi possono essere maggiori della precisione di macchina. È comunque ragionevole supporre che $\hat{a}$ e $\hat{b}$ siano approssimazioni con almeno una cifra corretta, così che $|\epsilon_a|$ e $|\epsilon_b|$ saranno sicuramente piccoli, tanto da poter

trascurare nei calcoli che seguono espressioni di grado ≥ 2, del tipo di $\epsilon_a^2, \epsilon_b^2$ o $\epsilon_a \times \epsilon_b$.

Supponiamo che $\hat{a}$ e $\hat{b}$ siano diversi da zero. Eseguendo la moltiplicazione si ottiene

$$\hat{a}\hat{b} = a(1+\epsilon_a)b(1+\epsilon_b) \simeq ab(1+\epsilon_a+\epsilon_b),$$

con un errore relativo dato da

$$\frac{\hat{a}\hat{b} - ab}{ab} \simeq \epsilon_a + \epsilon_b. \tag{2.8}$$

Questa relazione ci dice che la moltiplicazione non esalta gli errori sui dati e il prodotto ha praticamente la stessa accuratezza dei dati.

Passiamo ora all'addizione, supponendo $a + b \neq 0$. In questo caso si ottiene

$$\hat{a} + \hat{b} = a(1+\epsilon_a) + b(1+\epsilon_b) = a + b + (a\epsilon_a + b\epsilon_b),$$

e l'errore relativo è pertanto

$$\frac{(\hat{a}+\hat{b}) - (a+b)}{a+b} = \frac{a}{a+b}\epsilon_a + \frac{b}{a+b}\epsilon_b. \tag{2.9}$$

Osservando questa relazione si nota che quando $|a+b| < |a|$ e/o $|a+b| < |b|$ gli errori presenti sui dati vengono amplificati, portando così ad una perdita di accuratezza nel risultato. Evidentemente, condizione necessaria perché questo avvenga è che a e b abbiano segni opposti. A titolo di esempio, supponiamo che a sia un numero positivo con $\epsilon_a = 0$ e b sia un numero negativo; allora l'errore ϵ_b verrà amplificato se $|a + b| < |b|$, ovvero se $b < -a/2$; questa perdita di accuratezza sarà tanto più grave quanto più $a + b$ si avvicina a zero.

Esempio 2.3.2. Consideriamo la successione definita da

$$a_k = 1 + (1 - x)a_{k-1}$$

per $k = 2, 3, \ldots$, con x numero reale assegnato e a_1 dato. È facile verificare che se $a_1 = 1/x$, allora tutti i termini della successione sono uguali a $1/x$. In tabella 2.1 sono mostrati dei risultati ottenuti in semplice precisione [5] : per $x = 2, 5, 11$ si riportano alcuni elementi della successione calcolata $\{\hat{a}_k\}$ con i corrispondenti errori relativi $e_k = |\hat{a}_k - a_k|/|a_k|$. Questi errori sono valutati rispetto al valore $1/x$ calcolato in doppia precisione, che assumiamo come valore esatto. La tabella evidenzia situazioni diverse.

[5] Nelle tabelle scriveremo spesso i numeri reali nella notazione esponenziale tipica dei linguaggi di programmazione, secondo la quale il simbolo XeN sta per $X \times 10^N$.

x	k	$\hat{a}_k$	e_k
2	1	0.50000000	
	2	0.50000000	0.
	4	0.50000000	0.
	6	0.50000000	0.
	8	0.50000000	0.
	10	0.50000000	0.
5	1	0.20000000	
	2	0.19999999	6.0e−08
	4	0.19999981	9.5e−07
	6	0.19999695	1.5e−05
	8	0.19995117	2.4e−04
	10	0.19921875	3.9e−03
11	1	0.09090909	
	2	0.09090906	3.0e−07
	4	0.09090638	3.0e−05
	6	0.09063816	3.0e−03
	8	0.06381607	3.0e−01
	10	-2.61839294	3.0e+00

Tabella 2.1 Risultati relativi all'esempio 2.3.2

Per $x = 2$ gli errori sono tutti nulli: questo si spiega con il fatto che in questo caso l'algoritmo opera esclusivamente con numeri di macchina e non vengono introdotti errori ad alcun livello.

Per $x = 5$ e $x = 11$ si perde accuratezza all'aumentare di k e questa perdita di accuratezza è maggiore per $x = 11$ che per $x = 5$. Possiamo spiegare questo fenomeno utilizzando (2.8) e (2.9). Osserviamo che ad ogni passo si calcola la somma fra 1 e $(1 - x)\hat{a}_{k-1}$, ovvero la somma fra un numero esatto e un numero affetto da un errore relativo che, a causa di (2.8), è uguale a e_{k-1}. Da (2.9) segue allora che l'errore e_k è pari a e_{k-1} moltiplicato per $\frac{(1-x)a_{k-1}}{1+(1-x)a_{k-1}} = 1 - x$. In questa analisi non abbiamo tenuto conto degli errori di arrotondamento introdotti dalle operazioni, che sono comunque minori di ϵ_m. ■

2.3.1 Cancellazione numerica

Consideriamo ora una situazione particolarmente pericolosa dal punto di vista dell'amplificazione degli errori sui dati: la sottrazione fra due numeri molto vicini fra loro.

Per capire dove sta il pericolo, torniamo alla relazione (2.9): se a e b hanno segno opposto e $|a + b|$ è molto piccolo, ci aspettiamo una forte perdita di accuratezza nel risultato. Per esempio, siano $a = 2.1415898 \times 10^0$ e $b = 2.1415761 \times 10^0$ due numeri approssimati da $\hat{a} = 2.141589 \times 10^0$ e $\hat{b} = 2.141576 \times 10^0$ rispettivamente, con $\epsilon_a \simeq 3.7 \times 10^{-7}$ e $\epsilon_b \simeq 4.7 \times 10^{-8}$.

La (2.9) prevede che l'errore relativo di $\hat{a} - \hat{b}$ rispetto a $a - b$ sarà circa 5.1×10^{-2} con una perdita di accuratezza di 5-6 cifre rispetto ai dati.

È molto importante aver chiaro che questa perdita di accuratezza non dipende da errori introdotti dall'operazione di sottrazione, ma soltanto dalla presenza di errori sui dati: se due numeri esatti molto vicini fra di loro vengono sottratti, il risultato della sottrazione è esatto.

Il fenomeno dell'amplificazione dell'errore nella sottrazione fra numeri molto vicini viene detto cancellazione numerica. In taluni libri si aggiunge l'aggettivo catastrofica, per evidenziare il fatto che in questo caso la grossa perdita di accuratezza nasce in una sola operazione e non dall'accumulo di piccoli errori in una sequenza di più operazioni.

Se un fenomeno di cancellazione si produce a un qualche stadio di un algoritmo, l'errore sul risultato della sottrazione influenzerà tutti i calcoli successivi. Di conseguenza, salvo rare eccezioni, il risultato finale dell'algoritmo sarà affetto da un grosso errore. Questo significa che è buona abitudine, quando possibile, sostituire una formula "pericolosa" con un'altra che, non prevedendo sottrazioni, appare più sicura. Ad esempio, se si devono calcolare valori della funzione $f(x) = 1 - cos(x)$ per valori di x vicini a zero, quando $cos(x) \simeq 1$, è preferibile usare la forma alternativa $f(x) = sin^2(x)/(1 + cos(x))$ che può essere valutata abbastanza accuratamente per x vicino a zero. Le due forme della funzione sono equivalenti dal punto di vista matematico e darebbero gli stessi risultati se fosse possibile eseguire i calcoli con precisione infinita; ma esse non sono equivalenti dal punto di vista numerico perché prevedono sequenze diverse di operazioni.

I due esempi successivi mostrano altre situazioni in cui si può evitare il fenomeno della cancellazione usando formule diverse da quelle che la geometria o l'algebra elementare ci suggeriscono per risolvere un problema.

Esempio 2.3.3. Supponiamo di voler calcolare le soluzioni di un'equazione di secondo grado $ax^2 + bx + c = 0$, con coefficienti $a, b, c \neq 0$ tali che $b^2 - 4ac > 0$. La nota formula risolutiva è:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}. \tag{2.10}$$

In questa formula vediamo due possibili fonti di cancellazione numerica.

La prima è nel calcolo del discriminante $b^2 - 4ac$: se le due soluzioni, pur non coincidenti, sono molto vicine fra loro, allora $b^2 - 4ac$ è vicino a zero (ovvero b^2 è vicino a $4ac$) e il calcolo del discriminante provocherà perdita di accuratezza. Purtroppo non esistono formule alternative che evitino questo problema.

Consideriamo ora l'altra possibile fonte di cancellazione. Indichiamo con x_1 la soluzione con valore assoluto più grande e con x_2 l'altra. Se $b > 0$, x_1 si ottiene da (2.10) scegliendo il segno "$-$" e x_2 scegliendo il segno "$+$"; se $b < 0$, la situazione è rovesciata. In ogni caso il calcolo di

x_1 comporta l'addizione fra numeri dello stesso segno e, in base a quanto visto in precedenza, non crea problemi in presenza di errori sugli operandi. Viceversa, il calcolo di x_2 richiede una sottrazione e quindi ci aspettiamo l'insorgere di cancellazione numerica nel caso che sia $\sqrt{b^2 - 4ac} \simeq |b|$, ovvero $b^2 >> 4ac$.

Per evitare l'eventuale cancellazione si può ricorrere ad una formula alternativa in cui non ci siano da fare sottrazioni. A questo scopo, ricordiamo che il prodotto fra le soluzioni di un'equazione di secondo grado è uguale a c/a. Allora, una volta calcolata la soluzione x_1, si ottiene x_2 da:

$$x_2 = \frac{c}{ax_1}. \tag{2.11}$$

La tabella 2.2 riporta i risultati ottenuti in doppia precisione per l'equazione

$$x^2 + 10^k x + 1 = 0,$$

con $k = 1, 2, \ldots, 8$. Per valori grandi di k le due soluzioni tendono rispettivamente a -10^k e -10^{-k}. D'altra parte al crescere di k il numero $\sqrt{b^2 - 4ac} = \sqrt{10^{2k} - 4}$ diventa sempre più vicino a $b = 10^k$: ci aspettiamo quindi perdita di accuratezza nel calcolo di x_2 se usiamo la formula (2.10). Nella tabella si riportano per ogni k:
n : numero di cifre uguali in $\sqrt{b^2 - 4ac}$ e b,
x_2 : soluzione più piccola in valore assoluto calcolata con (2.11),
$\tilde{x}_2$: soluzione più piccola in valore assoluto calcolata con (2.10).

k	n	x_2	$\tilde{x}_2$
1	1	−1.01020514433644e−01	−1.01020514433644e−01
2	4	−1.00010002000500e−02	−1.00010002000488e−02
3	6	−1.00000100000200e−03	−1.00000100002262e−03
4	8	−1.00000001000000e−04	−1.00000001111766e−04
5	10	−1.00000000010000e−05	−1.00000033853576e−05
6	12	−1.00000000000100e−06	−1.00000761449337e−06
7	14	−1.00000000000001e−07	−9.96515154838562e−08
8	16	−1.00000000000000e−08	−7.45058059692383e−09

Tabella 2.2 Risultati relativi all'esempio 2.3.3

Come ci aspettavamo, la formula (2.11) dà risultati corretti per tutti i valori di k, mentre la (2.10) dà risultati affetti da un errore sempre più grande al crescere di k. Quanto più n è grande, tanto più visibile è l'effetto dell'errore di cancellazione. ■

Esempio 2.3.4. L'approssimazione con un elevato numero di cifre del numero irrazionale π, rapporto tra la lunghezza di una circonferenza e il suo diametro, può essere basata sulla seguente osservazione [6].

Costruiamo due poligoni regolari con un numero fissato di lati, uno inscritto nella circonferenza di raggio unitario e l'altro circoscritto alla stessa circonferenza, e consideriamo i perimetri dei due poligoni. Evidentemente la lunghezza della circonferenza è compresa fra i due perimetri e di conseguenza il numero π è compreso fra i due semiperimetri. Al crescere del numero di lati si ottengono due successioni di semiperimetri (dei poligoni inscritti e dei poligoni circoscritti) che risultano l'una monotona crescente e l'altra monotona decrescente; entrambe convergono ad uno stesso limite, che è π. Sulla base di queste considerazioni possiamo costruire un algoritmo per calcolare un'approssimazione di π con la massima accuratezza possibile, compatibilmente con la precisione di macchina.

Indichiamo con l_n la lunghezza del lato del poligono regolare di 2^n lati inscritto nella circonferenza di raggio unitario e con L_n quella del lato del poligono circoscritto con uguale numero di lati. Considerazioni geometriche elementari portano alla seguente espressione di L_n in funzione di l_n:

$$L_n = \frac{2l_n}{\sqrt{4 - l_n^2}}.$$

Poiché

$$l_2 = \sqrt{2},$$

e

$$l_{n+1} = \sqrt{2 - \sqrt{4 - l_n^2}}, \tag{2.12}$$

siamo in grado di calcolare i semiperimetri s_n e S_n dei poligoni inscritti e circoscritti per $n = 1, 2, \ldots$.

In tabella 2.3 riportiamo alcuni risultati ottenuti in doppia precisione. Per ogni n riportiamo anche gli errori $e_n = |s_n - \pi|/\pi$ e $E_n = |S_n - \pi|/\pi$ calcolati usando l'approssimazione di π a 16 cifre $\pi \simeq 3.14159265358979$. Osserviamo che gli errori diminuiscono fino a un certo punto e poi cominciano a risalire: le due successioni sembrano convergere entrambe ad uno stesso limite, che però è diverso da π. Per capire il motivo di questo comportamento dobbiamo tornare alla formula (2.12) sulla quale si basa tutto il procedimento. Poiché $l_n \to 0$ per $n \to \infty$, il valore di $\sqrt{4 - l_n^2}$ tende a 2 e per n sufficientemente grande il calcolo dell'espressione $2 - \sqrt{4 - l_n^2}$ produce un errore di cancellazione catastrofica che si propaga velocemente con

[6] Sulla base di questa osservazione, il matematico cinese Tsu Chung-chi nel quinto secolo d.C. riuscì a calcolare un'approssimazione a 7 cifre $\pi \simeq 3.141593$, rimasta la migliore approssimazione calcolata di questo numero fino al quindicesimo secolo, quando fu scoperta l'espansione in serie e si riuscirono a calcolare più cifre.

n	s_n	e_n	S_n	E_n
2	2.82842712474619	1.0e−01	4.00000000000000	2.7e−01
3	3.06146745892072	2.6e−02	3.31370849898476	5.5e−02
5	3.13654849054594	1.6e−03	3.15172490742926	3.2e−03
7	3.14127725093276	1.0e−04	3.14222362994244	2.0e−04
9	3.14157294036788	6.3e−06	3.14163208070397	1.3e−05
11	3.14159142150464	3.9e−07	3.14159511774302	7.8e−07
13	3.14159257654500	2.5e−08	3.14159280755978	4.9e−08
15	3.14159265480759	3.9e−10	3.14159266924601	5.0e−09
17	3.14159260737572	1.5e−08	3.14159260827812	1.4e−08
19	3.14159412519519	4.7e−07	3.14159412525159	4.7e−07
21	3.14159655370482	1.2e−06	3.14159655370834	1.2e−06
23	3.14182968188920	7.5e−05	3.14182968188942	7.5e−05
25	3.14245127249413	2.7e−04	3.14245127249415	2.7e−04

Tabella 2.3 Risultati relativi all'esempio 2.3.4

gli effetti che vediamo nella tabella. Anche in questo caso si può ovviare usando la formula alternativa

$$l_{n+1} = \frac{l_n}{\sqrt{2 + \sqrt{4 - l_n^2}}}. \tag{2.13}$$

I risultati riportati in tabella 2.4 mostrano che questa formula funziona

n	s_n	e_n	S_n	E_n
2	2.82842712474619	1.0e−01	4.00000000000000	2.7e−01
3	3.06146745892072	2.6e−02	3.31370849898476	5.5e−02
5	3.13654849054594	1.6e−03	3.15172490742926	3.2e−03
7	3.14127725093277	1.0e−04	3.14222362994246	2.0e−04
9	3.14157294036709	6.3e−06	3.14163208070318	1.3e−05
11	3.14159142151120	3.9e−07	3.14159511774959	7.8e−07
13	3.14159257658487	2.5e−08	3.14159280759964	4.9e−08
15	3.14159264877699	1.5e−09	3.14159266321541	3.1e−09
17	3.14159265328899	9.6e−11	3.14159265419140	1.9e−10
19	3.14159265357099	6.0e−12	3.14159265362739	1.2e−11
21	3.14159265358862	3.7e−13	3.14159265359214	7.5e−13
23	3.14159265358972	2.3e−14	3.14159265358994	4.7e−14
25	3.14159265358979	1.0e−15	3.14159265358980	3.4e−15

Tabella 2.4 Risultati relativi all'esempio 2.3.4

bene perché non si verificano errori di cancellazione al tendere di l_n a zero. ■

Gli esempi visti finora ci hanno mostrato che la propagazione degli errori di arrotondamento può essere disastrosa, ma può anche non esserlo. In

particolare, in ciascuno degli esempi 2.3.3 e 2.3.4 abbiamo confrontato due diversi algoritmi: in uno la presenza di sottrazioni fra numeri molto vicini provoca l'esaltazione di piccoli errori introdotti in precedenza, portando in definitiva ad un risultato molto sbagliato; nell'altro i piccoli errori di arrotondamento non vengono amplificati e il risultato è più che accettabile.

D'altra parte, nel caso dell'esempio 2.3.2 la crescita dell'errore all'aumentare di k, non legata a fenomeni di cancellazione, sembra inevitabile e non abbiamo trovato algoritmi alternativi.

In generale, la propagazione degli errori attraverso un algoritmo può venire descritta e studiata attraverso due concetti distinti e fondamentali: il condizionamento del problema e la stabilità dell'algoritmo.

2.3.2 Condizionamento di un problema

Il termine "condizionamento" è usato per descrivere la sensibilità di un problema a cambiamenti nei dati.

Un problema è detto ben condizionato se a piccoli cambiamenti nei dati corrispondono piccoli cambiamenti nei risultati; viceversa, un problema è detto mal condizionato se a piccoli cambiamenti nei dati corrispondono grandi cambiamenti nei risultati [7].

È evidente l'importanza di questo concetto per il Calcolo numerico. Se i dati di un problema non sono numeri di macchina, essi saranno necessariamente sostituiti dalla loro rappresentazione floating point: tutto quello che possiamo sperare a questo punto è di riuscire a calcolare accuratamente il risultato corrispondente ai nuovi dati. Se il problema è ben condizionato questo risultato è vicino a quello del problema originale e possiamo ritenerci soddisfatti. Se invece il problema è mal condizionato, allora il risultato migliore che noi possiamo calcolare è comunque lontano da quello che avremmo voluto trovare. Così la misura del condizionamento di un problema pone una barriera "naturale" all'accuratezza con cui possiamo risolvere numericamente il problema stesso.

Quando all'inizio di questo paragrafo abbiamo esaminato la propagazione degli errori attraverso le operazioni elementari, abbiamo di fatto analizzato il condizionamento di questi due problemi: calcolare la somma (o differenza) fra due numeri e calcolare il prodotto fra due numeri. Da quanto abbiamo visto, il primo problema è mal condizionato quando si tratta di sottrarre due numeri molto vicini fra loro: in questo caso infatti si produce

[7] Alcuni autori usano termini diversi per indicare lo stesso concetto: ad esempio usano "stabile" o "ben posto" invece di "ben condizionato", e "instabile" o "mal posto" invece di "ben condizionato", tutti termini che hanno una lunga tradizione nel campo della Matematica teorica. Noi preferiamo usare i termini "ben condizionato" o "mal condizionato" che sono tradizionalmente usati nel campo della Matematica computazionale.

cancellazione numerica e gli errori sui dati si propagano in maniera disastrosa sul risultato. Per contro, possiamo dire che il calcolo del prodotto fra due numeri è un problema ben condizionato: gli errori relativi sui dati si sommano e quindi il risultato è accurato quanto lo era il meno accurato dei dati. D'altra parte il fatto che l'errore relativo sul prodotto fra due numeri sia la somma degli errori relativi sugli operandi implica che il calcolo della potenza x^n è un problema mal condizionato per n grande: l'errore relativo sul risultato è infatti pari a n volte l'errore relativo sul dato x.

Un altro esempio di problema mal condizionato è il calcolo della successione $\{a_k\}$ nell'esempio 2.3.2 per $|x| > 1$. Infatti l'analisi che abbiamo svolto nell'esempio dimostra che un cambiamento relativo sul dato $a_1 = 1/x$ viene amplificato di un fattore $(1-x)$ ad ogni passo, così che la successione corrispondente al nuovo dato è una successione completamente diversa da quella relativa al dato originale.

Esempio 2.3.5. Consideriamo il polinomio

$$p(x) = 24.5x^3 - 491x^2 + 19.81x - 0.2,$$

le cui radici approssimate a sette cifre sono

$$x_1 = 2.000041 \times 10^1, \quad x_2 = 2.038618 \times 10^{-2} \quad x_3 = 2.002116 \times 10^{-2}.$$

Se il coefficiente di x^2 viene cambiato in 490.99 le radici diventano:

$$\tilde{x}_1 = 2.000000 \times 10^1, \quad \tilde{x}_2 = 2.040816 \times 10^{-2} \quad \tilde{x}_3 = 2.000000 \times 10^{-2},$$

con

$$\frac{|\tilde{x}_1 - x_1|}{|x_1|} \simeq 2.0 \times 10^{-5}, \quad \frac{|\tilde{x}_2 - x_2|}{|x_2|} \simeq 1.1 \times 10^{-3}, \quad \frac{|\tilde{x}_3 - x_3|}{|x_3|} \simeq 1.1 \times 10^{-3}.$$

Poiché a fronte di un cambiamento relativo $\simeq 10^{-5}$ in un dato, i risultati subiscono un cambiamento fino a 10^{-3}, possiamo affermare che questo polinomio è mal condizionato rispetto al calcolo delle radici.

Il condizionamento del calcolo delle radici dei polinomi è stato studiato molti anni fa da J. H. Wilkinson [8] in [37]. Un trattamento abbastanza approfondito della questione si può trovare in [5], da cui questo esempio è tratto. ■

[8] J. H. Wilkinson (1919-1986) è stato uno dei pionieri del Calcolo numerico, individuando fra l'altro l'importanza degli errori nella risoluzione numerica dei problemi matematici, soprattutto di problemi di algebra lineare. Sono dovuti a Wilkinson i primi studi approfonditi sull'influenza del condizionamento dei problemi e sulla stabilità degli algoritmi.

Esempio 2.3.6. Consideriamo il seguente sistema lineare

$$\begin{cases} 3x + 5y &= 10 \\ 3.01x + 5.01y &= 1 \end{cases}$$

la cui soluzione è $x = -2255, y = 1355$. Se cambiamo il coefficiente di x nella seconda equazione, la soluzione cambia. Ad esempio se il coefficiente diventa 3.02 la soluzione diventa $x = -644.3, y = 388.6$; se il coefficiente diventa 3.1 la soluzione diventa $x = -95.96, y = 59.57$. Evidentemente, questo è un problema mal condizionato, dal momento che il cambiamento nella soluzione è molto più grande del cambiamento nel dato.

Se interpretiamo geometricamente il sistema, riusciamo a capire meglio la natura di questo cattivo condizionamento. Sappiamo infatti che risolvere il sistema equivale a trovare le coordinate del punto di intersezione fra le rette descritte dalle due equazioni. Trattandosi di due rette "quasi parallele" (i coefficienti angolari sono $3/5 = -0.6$ e $3.1/5.1 \simeq -0.6008$ rispettivamente), un piccolo spostamento di una delle due provoca un grosso spostamento del punto di intersezione.

A riprova di questo ragionamento, consideriamo un nuovo sistema lineare in cui le due rette sono ben sghembe:

$$\begin{cases} 5x + 10y &= 15 \\ 4x + 2y &= 6. \end{cases}$$

In questo caso la soluzione è $x = 1, y = 1$. Esaminiamo l'effetto di perturbazioni su un coefficiente, ad esempio il coefficiente di y nella prima equazione: se il coefficiente diventa 10.1 la soluzione diventa $x = 1.007, y = 0.9868$; se il coefficiente diventa 9.9 la soluzione diventa $x = 0.9932, y = 1.013$. Poiché la perturbazione sul risultato è praticamente dello stesso ordine della perturbazione sul dato, possiamo pensare che questo sia un problema ben condizionato. ■

2.3.3 Stabilità di un algoritmo

Sappiamo da quanto detto finora che se un algoritmo viene realizzato in precisione finita non possiamo aspettarci in generale di calcolare il risultato esatto. Lo scopo diventa calcolare un'approssimazione del risultato "esatta" nei limiti della precisione di macchina. Alcuni esempi ci hanno mostrato che questo scopo non sempre viene raggiunto. Anche se ogni singola operazione viene eseguita correttamente (ovvero rispettando la (2.7)), l'accumulo degli errori di arrotondamento può portare ad un risultato inaccettabile.

La mancanza di accuratezza nel risultato può essere talvolta imputabile al cattivo condizionamento del problema e in questo caso l'ostacolo non può essere aggirato, neanche cambiando algoritmo. Quando invece il problema è ben condizionato e nonostante questo il risultato non è accurato quanto

sarebbe lecito attendersi, allora l'algoritmo utilizzato è scadente e occorre trovarne uno diverso.

Un algoritmo che, realizzato in aritmetica floating point, dà risultati corretti nei limiti della precisione di macchina e del condizionamento del problema è detto stabile. Viceversa, un algoritmo che dà risultati immotivatamente poco accurati è detto instabile.

Nell'esempio 2.3.3 abbiamo visto a confronto due diversi algoritmi per calcolare la soluzione più vicina a zero di un'equazione di secondo grado. Uno dei due algoritmi è potenzialmente instabile perché prevede una sottrazione. D'altra parte il problema non è malcondizionato, tanto è vero che con il secondo algoritmo si riesce a risolverlo con la massima accuratezza. Analogamente nell'esempio 2.3.4 l'algoritmo basato sulla formula (2.12) è instabile mentre quello basato sulla formula (2.13) è stabile. Anche in questo caso, l'instabilità è causata dall'insorgere di cancellazione numerica da un certo stadio in poi.

Non sempre l'instabilità di un algoritmo è imputabile alla cancellazione numerica. Vedremo nei capitoli successivi esempi "illustri" di algoritmi instabili dove non si verificano necessariamente fenomeni di cancellazione. D'altra parte già l'esempio 2.3.1 ci fornisce lumi al riguardo. In quel caso l'algoritmo usato per calcolare la somma fallisce e dà un risultato completamente sbagliato (pur essendo tutti i dati numeri positivi e, fra l'altro, perfettamente rappresentabili in macchina) quando l'addendo dominante, 1, viene considerato al primo posto. In quel caso la cura è consistita nel cambiare l'ordine dei dati, spostando l'addendo uguale a 1 in fondo. Consideriamo ora un esempio analogo.

Esempio 2.3.7. Un noto risultato di matematica dice che

$$\sum_{k=1}^{\infty} \frac{1}{k^2} = \frac{\pi^2}{6} = 1.64493406684822...$$

Supponiamo di voler verificare questa uguaglianza. L'algoritmo più ovvio consiste nel calcolare la somma dei termini $\frac{1}{k^2}$ per k crescente, fermandosi quando il termine non porta più nessun contributo alla somma (sappiamo che in aritmetica floating point questo sicuramente avverrà prima o poi). Ebbene, eseguendo i calcoli in doppia precisione l'algoritmo si ferma per $k = 94906266$ con $\frac{1}{k^2} \simeq 1.11 \times 10^{-16}$ e la somma calcolata è 1.644934057834575, approssimazione a 9 cifre del risultato corretto. La spiegazione di questa scarsa accuratezza rispetto alle 15 cifre che potremmo aspettarci in doppia precisione sta nel fatto che sommiamo i numeri dal più grande al più piccolo: via via che la somma calcolata aumenta, gli ulteriori addendi, sempre più piccoli, non riescono a far sentire il loro contributo in modo significativo. Ovviamente il rimedio anche in questo caso sarebbe rovesciare l'ordine degli addendi, sommandoli dal più piccolo al più grande; purtroppo però questa

cosa non è fattibile perché non sappiamo a priori quanti termini dobbiamo sommare. Il lettore interessato a scoprire la cura adatta in questa situazione può consultare il libro [22], dove un intero capitolo è dedicato agli algoritmi per calcolare sommatorie. ■

L'analisi della stabilità dei metodi numerici, ovvero del modo in cui gli errori si propagano attraverso un algoritmo, è un settore molto importante del Calcolo numerico. Spesso questa analisi è molto complicata e nei capitoli successivi ci limiteremo per lo più a citare i risultati fondamentali rinviando alla letteratura specializzata chi fosse interessato a conoscere i dettagli.

3

EQUAZIONI NON LINEARI

3.1 Introduzione

Sia data una funzione $f : \mathbb{R} \to \mathbb{R}$ non lineare e supponiamo di voler calcolare una soluzione x^* dell'equazione

$$f(x) = 0, \tag{3.1}$$

ovvero uno zero, o radice, della funzione [1]. Soltanto in rari casi, fondamentalmente quelli che vengono presi in considerazione nei corsi di matematica delle scuole secondarie, sono disponibili formule o procedimenti espliciti per calcolare x^*. In generale l'unica cosa che possiamo fare è procedere per approssimazioni successive, il che ovviamente ci mette nella condizione di dover modificare il nostro obiettivo: non cercheremo più di determinare x^* quanto una sua (possibilmente buona) approssimazione. I metodi numerici per la risoluzione di equazioni non lineari si basano sulla costruzione di una successione $\{x_k\}$ di approssimazioni di x^*, a partire da una stima iniziale x_0. Possiamo sempre supporre di procurarci in qualche modo x_0, ad esempio grazie alla conoscenza del problema fisico di cui $f(x)$ rappresenta il modello matematico, oppure tramite uno studio preliminare a grosse linee dell'andamento della funzione (per via numerica o per via grafica). I metodi di questa natura sono detti metodi iterativi: ogni termine della successione $\{x_k\}$ è un'iterata e il calcolo di ogni iterata è un'iterazione.

A modi diversi di costruire la successione corrispondono metodi diversi, con differenti caratteristiche: uno potrebbe portarci velocemente nelle vicinanze di x^*, un altro potrebbe essere più lento; uno potrebbe richiedere una scelta molto attenta di x_0, per un altro qualunque x_0 potrebbe andar bene; in uno il calcolo di x_k per un dato k potrebbe coinvolgere soltanto

[1] Ricordiamo che una funzione f è lineare rispetto a x quando ha la forma $f(x) = ax+b$, dove i coefficienti a e b sono numeri reali. Tutte le funzioni che non hanno questa forma sono dette non lineari. Per esempio, sono funzioni non lineari tutte le funzioni algebriche, ovvero i polinomi di grado superiore al primo e le funzioni trascendenti come $f(x) = x + 4\sin(x)$, $f(x) = e^x - x^3 - cos^2x - 1$ o $f(x) = \log(x) - \sqrt{x} + 1$.

x_{k-1}, in un altro anche x_{k-2} (nel qual caso non basterebbe più una sola approssimazione iniziale, ma ne occorrerebbero due). Quello che conta perché un metodo sia degno di essere preso in considerazione è che, almeno sotto opportune ipotesi, sia garantita la convergenza della successione $\{x_k\}$ a x^*, cioè

$$\lim_{k\to\infty} x_k = x^*. \tag{3.2}$$

Un metodo per cui questo sia vero si dice convergente. Come vedremo, per alcuni metodi è possibile dimostrare la convergenza soltanto supponendo x_0 sufficientemente vicino a x^*. Tali metodi sono detti localmente convergenti. Invece, metodi per i quali si può dimostrare la convergenza in qualunque modo si parta sono detti globalmente convergenti.

Dato un metodo iterativo convergente, abbiamo la certezza di arrivare a x^* al limite per k tendente all'infinito. Ovviamente nella pratica dobbiamo essere in grado di arrestare il procedimento ad un k finito, ovvero di stabilire quando l'iterata x_k è per noi un'approssimazione accettabile di x^*, fissata l'accuratezza con cui si vuole risolvere il problema. Una volta stabiliti uno o più criteri di arresto, potremo scrivere un algoritmo basato sul metodo iterativo prescelto. Vale la pena di sottolineare fin da ora che le richieste di accuratezza espresse attraverso i criteri di arresto non potranno prescindere dalla precisione di macchina con cui si lavora.

La scelta di un metodo piuttosto che un altro dipende da vari fattori. Ad esempio, è importante sapere se la funzione f è data in forma analitica oppure no e se il calcolo di valori della f comporta poco sforzo computazionale o molto (come può succedere se il calcolo di uno di questi valori richiede l'esecuzione di un algoritmo numerico o la simulazione di un processo fisico). Un'altra questione fondamentale è se si dispone di una buona stima della soluzione per innescare il processo iterativo, oppure soltanto di un'approssimazione grossolana. Per valutare i metodi iterativi per equazioni non lineari e confrontarli fra loro si usano fondamentalmente due parametri:
1) il costo computazionale, inteso come costo di una singola iterazione, misurato attraverso il numero di valutazioni di f (ed eventualmente della derivata f') necessarie per il calcolo di x_k;
2) la velocità di convergenza che è legata al numero di iterazioni che mediamente ci vogliono per ottenere una soluzione approssimata con una certa accuratezza, partendo con un dato errore iniziale.

In questo capitolo presentiamo i principali metodi, che sono il metodo di bisezione, il metodo di Newton e il metodo delle secanti, e rinviamo a [11] o [15] per una panoramica più completa. Il primo metodo che viene introdotto è il metodo di bisezione che è il più semplice da capire e da realizzare: ad ogni iterazione esso richiede soltanto il calcolo di f in un punto ed ha l'ottima proprietà di essere globalmente convergente. D'altra parte questo metodo ha una convergenza molto lenta e quindi talvolta si preferisce usare un metodo diverso, caratterizzato da una maggiore velocità di convergenza. Da questo punto di vista è da preferirsi il metodo di Newton che usa più

informazioni sulla funzione f (in particolare richiede il calcolo non solo di f ma anche della derivata f' o di una sua approssimazione) ed ha quindi un maggior costo computazionale. Tutto questo viene ripagato dalla maggiore velocità che permette in genere di raggiungere l'accuratezza richiesta con un numero molto minore di iterazioni rispetto al metodo di bisezione. Nelle situazioni in cui usare il metodo di Newton è svantaggioso o impossibile, ad esempio quando il calcolo della derivata (esatta o approssimata) richiede troppe operazioni, può essere conveniente l'uso del metodo delle secanti, che sacrifica un po' della velocità di convergenza del metodo di Newton, ma non richiede l'uso della derivata.

Il metodo di Newton e il metodo delle secanti non sono globalmente convergenti e per questo motivo sono spesso usati in accoppiata con il metodo di bisezione che invece lo è: si parte con il metodo di bisezione per avvicinarsi a x^* e poi si fa partire il metodo di Newton o il metodo delle secanti per raggiungere velocemente l'accuratezza fissata. In questo modo si creano dei procedimenti "ibridi" che convergono globalmente e, quanto meno nella fase finale, velocemente.

Nell'ultimo paragrafo del capitolo facciamo un cenno alla generalizzazione del metodo di Newton per il trattamento di sistemi di equazioni non lineari.

3.2 Metodo di bisezione

Il metodo di bisezione si basa sul noto Teorema degli zeri [2]: se la funzione f è continua in un intervallo dato $[a, b]$ e $f(a)f(b) < 0$, allora esiste almeno un punto $x^* \in (a, b)$ tale che $f(x^*) = 0$. Osserviamo che questo risultato riguarda soluzioni dell'equazione $f(x) = 0$ in corrispondenza delle quali il grafico di f attraversa l'asse delle ascisse e non soluzioni in corrispondenza delle quali il grafico di f risulta tangente all'asse stesso.

In virtù di questo risultato, dato l'intervallo $[a, b]$, lo si può dividere a metà e determinare in quale dei due sottointervalli così ottenuti si trova la soluzione. Infatti, se c è il punto medio dell'intervallo, cioè $c = \frac{1}{2}(a + b)$ e consideriamo il prodotto $f(a)f(c)$, ci sono tre possibilità:

1. $f(a)f(c) < 0$
2. $f(a)f(c) = 0$
3. $f(a)f(c) > 0$

Nel caso 2 si conclude che $f(c) = 0$ e quindi abbiamo trovato la soluzione $x^* = c$; negli altri due casi deduciamo che la soluzione sta in uno dei due semintervalli: in quello di sinistra $[a, c]$ nel caso 1 e in quello di destra $[c, b]$ nel caso 3.

L'idea del metodo di bisezione è quella di procedere per dimezzamenti successivi dell'intervallo dato, ogni volta individuando da quale parte sta

la soluzione. Per entrare più in dettaglio introduciamo gli indici di iterazione: partiamo dall'intervallo $[a_0, b_0] = [a, b]$ e poniamo $x_0 = \frac{1}{2}(a_0 + b_0)$; confrontando il segno di $f(x_0)$ con quello di $f(a_0)$ si individua in quale dei due sottointervalli $[a_0, x_0]$ o $[x_0, b_0]$ cade x^*. Questo nuovo intervallo viene chiamato $[a_1, b_1]$ e il suo punto di mezzo x_1; confrontando i segni di $f(a_1)$ e $f(x_1)$ si decide se la soluzione sta in $[a_1, x_1]$ o $[x_1, b_1]$. E così via. Dato l'intervallo $[a_k, b_k]$, l'iterazione segue il seguente schema:

1. Poni $x_k = \frac{1}{2}(a_k + b_k)$
2. Se $f(x_k) = 0$, allora:
   $x^* = x_k$ e fermati
   altrimenti se $f(x_k)f(a_k) < 0$, allora:
   $a_{k+1} = a_k$ e $b_{k+1} = x_k$
   altrimenti:
   $a_{k+1} = x_k$ e $b_{k+1} = b_k$
   Fine scelta
3. Fine iterazione

Una rappresentazione grafica del metodo di bisezione è mostrata nella figura 3.1.

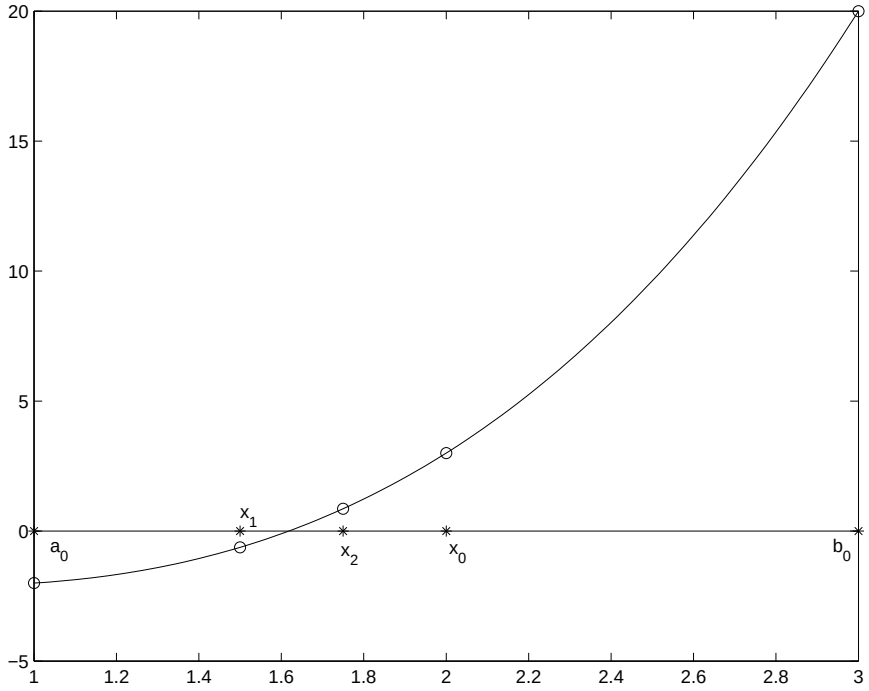

Figura 3.1 Metodo di bisezione

3.2.1 Convergenza del metodo di bisezione

Vogliamo ora dimostrare che il metodo di bisezione è un metodo globalmente convergente.

Teorema 3.2.1. Se f è continua nell'intervallo $[a_0, b_0]$ e $f(a_0)f(b_0) < 0$, allora la successione $\{x_k\}$ generata dal metodo di bisezione converge a un punto x^* interno a $[a_0, b_0]$ tale che $f(x^*) = 0$.

Dimostrazione. Abbiamo già visto che le ipotesi fatte implicano l'esistenza di una soluzione $x^* \in (a_0, b_0)$ dell'equazione $f(x) = 0$. Osserviamo ora che il metodo di bisezione genera due successioni $\{a_k\}$ e $\{b_k\}$ tali che

$$(b_k - a_k) = \frac{1}{2}(b_{k-1} - a_{k-1}) = \left(\frac{1}{2}\right)^2 (b_{k-2} - a_{k-2}) = \ldots = \left(\frac{1}{2}\right)^k (b_0 - a_0)$$

per ogni k; non solo, ma poiché x_k è il punto di mezzo di $[a_k, b_k]$ e ad ogni iterazione $x^* \in [a_k, b_k]$, si ha

$$|e_k| \leq \frac{1}{2}(b_k - a_k), \tag{3.3}$$

dove con e_k abbiamo indicato l'errore $e_k = x_k - x^*$. Allora

$$0 \leq |e_k| \leq \left(\frac{1}{2}\right)^{k+1} (b_0 - a_0), \tag{3.4}$$

da cui deduciamo facilmente per il Teorema dei "carabinieri" che

$$\lim_{k\to\infty} |e_k| = 0,$$

ovvero vale la (3.2). Possiamo allora affermare che il metodo di bisezione è un metodo convergente; poiché inoltre la convergenza è garantita qualunque sia l'ampiezza dell'intervallo iniziale, si può dire che il metodo converge globalmente. □

La (3.4) è una relazione molto utile dal punto di vista pratico: sostanzialmente essa fornisce una maggiorazione della distanza di x_k dalla soluzione e quindi dell'errore che commetteremmo interrompendo il procedimento alla k-esima iterazione e accettando x_k come soluzione approssimata. Dalla (3.4) è anche possibile ricavare a priori una stima del numero k_ϵ di iterazioni necessarie per approssimare x^* con un errore non superiore a una soglia fissata ϵ. Tenendo conto della (3.4) possiamo individuare k_ϵ imponendo che sia

$$\frac{1}{2^{k_\epsilon+1}}(b_0 - a_0) \simeq \epsilon,$$

da cui si ricava

$$k_\epsilon + 1 \simeq \log_2 \left(\frac{b_0 - a_0}{\epsilon} \right) = \frac{log_{10} \left(\frac{b_0 - a_0}{\epsilon} \right)}{log_{10} 2} \simeq 3.3 log_{10} \left(\frac{b_0 - a_0}{\epsilon} \right). \quad (3.5)$$

Ad esempio, se $b_0 - a_0 = 2$ e $\epsilon = 10^{-2}$, si ottiene $k_\epsilon \simeq 6.6$, il che significa che $|x_k - x^*|$ sarà minore o uguale di 10^{-2} al più tardi per $k = 7$. Analogamente, se l'intervallo iniziale fosse di ampiezza uguale a 100, saremmo certi di arrivare a distanza da x^* minore o uguale di $\epsilon = 10^{-8}$ entro le prime 33 iterazioni, poiché dalla (3.5) si ricava $k_\epsilon \simeq 32.2$.

Esempio 3.2.1. Consideriamo la funzione

$$f(x) = arctg(x),$$

che si annulla in $x^* = 0$ e applichiamo il metodo di bisezione a partire da $[a_0, b_0] = [-0.5, 3.1]$. La (3.5) ci dice che riusciremo ad arrivare a distanza

k	x_k	$\lvert f(x_k)\rvert$
0	1.300000000000000e+00	9.2e−01
1	4.000000000000000e−01	3.8e−01
2	−4.999999999999999e−02	5.0e−02
3	1.750000000000000e−01	1.7e−01
4	6.250000000000001e−02	6.2e−02
5	6.250000000000013e−03	6.3e−03
6	−2.187499999999999e−02	2.2e−02
7	−7.812499999999988e−03	7.8e−03
10	9.765625000000124e−04	9.8e−04
11	9.765625000001234e−05	9.8e−05
12	−3.417968749999877e−04	3.4e−04
20	−1.907348632689142e−07	1.9e−07
21	6.675720214967108e−07	6.7e−07
22	2.384185791138983e−07	2.4e−07
23	2.384185792249206e−08	2.4e−08
24	−8.344650267321107e−08	8.3e−08
25	−2.980232237535950e−08	3.0e−08
26	−2.980232226433720e−09	3.0e−09
27	1.043081284802917e−08	1.0e−08
28	3.725290310797726e−09	3.7e−09
29	3.725290421820028e−10	3.7e−10

Tabella 3.1 Esempio 3.2.1: metodo di bisezione per $f(x) = arctg(x)$

da x^* inferiore a 10^{-9} entro 31 iterazioni. In effetti dalla tabella 3.1, dove sono riportati alcuni risultati ottenuti in doppia precisione, si vede che le iterate si avvicinano a x^* e l'errore (in questo caso e_k coincide con x_k) diventa minore di 10^{-9} alla 29-esima iterazione. È anche interessante osservare che gli errori non diminuiscono in modo monotono, come si potrebbe

erroneamente aspettarsi in virtù del fatto che l'ampiezza dell'intervallo si dimezza ad ogni iterazione. Vediamo che in alcune iterazioni x_k è più vicina ad x^* di quanto non lo fosse x_{k-1} (e quindi $|e_k| < |e_{k-1}|$); in altre iterazioni invece x_k si riallontana da x^* e quindi l'errore aumenta. Questo è un comportamento tipico del metodo di bisezione.

Nella tabella vediamo che i valori della funzione $f(x_k)$ tendono a zero al crescere di k, come è ovvio che succeda dal momento che f è una funzione continua. Il fatto che gli errori e i valori di f si avvicinino a zero con la stessa velocità è un caso, dovuto al particolare comportamento della funzione intorno a x^* (ricordiamo che $\lim_{x\to 0} \frac{arctg(x)}{x} = 1$). ■

Esempio 3.2.2. Consideriamo ora la funzione

$$f(x) = Csin(x)cos(x) - x^3,$$

dove C è una costante qualsiasi. Per ogni scelta di C, la funzione si annulla in $x^* = 0$, ma la pendenza in x^* cambia con C perché

$$f'(x) = C(cos^2(x) - sin^2(x)) - 3x^2 \Rightarrow f'(x^*) = C.$$

Nella tabella 3.2 riportiamo i risultati ottenuti in doppia precisione partendo da $[a_0, b_0] = [-0.2, 0.1]$, per $C = -10^{-7}$ e $C = 10^7$. La successione $\{x_k\}$ è la stessa per entrambi i C poiché il metodo di bisezione non usa i valori della f per determinare le iterate, ma soltanto i segni di questi valori. Come ci aspettiamo, al crescere di k la successione $\{f(x_k)\}$ tende a zero

k	x_k	$\|f(x_k)\|$ $C = -10^{-7}$	$\|f(x_k)\|$ $C = 10^7$
0	−5.000000000000000e−02	1.3e−04	5.0e+05
1	2.500000000000000e−02	1.6e−05	2.5e+05
2	−1.250000000000000e−02	1.9e−06	1.3e+05
5	1.562500000000000e−03	4.0e−09	1.6e+04
6	−7.812500000000000e−04	5.6e−10	7.8e+03
10	−4.882812500000000e−05	5.0e−12	4.9e+02
11	2.441406250000000e−05	2.5e−12	2.4e+02
15	1.525878906250000e−06	1.5e−13	1.5e+01
16	−7.629394531250000e−07	7.6e−14	7.6e+00
24	−2.980232238769531e−09	3.0e−16	3.0e−02
35	1.455191522836685e−12		1.5e−05
36	−7.275957614183426e−13		7.3e−06
45	1.421085471520201e−15		1.4e−08
46	−7.105427357601002e−16		7.1e−09

Tabella 3.2 Esempio 3.2.2: metodo di bisezione per $f(c) = Csin(x)cos(x) - x^3$

per entrambi i valori di C, ma la diversa pendenza della funzione intorno a x^* fa sì che, a parità di x_k, i valori della funzione abbiano ordini di grandezza molto diversi: per $C = -10^{-7}$ abbiamo arrestato il procedimento

alla 24-esima iterazione perché $|f(x_k)|$ era ormai alle soglie della precisione di macchina $\epsilon_m \simeq 2.2 \times 10^{-16}$; alla stessa iterazione per $C = 10^7$ si ha ancora un valore abbastanza grosso di $f(x_k)$, circa uguale a 10^{-2}. Questa osservazione verrà ripresa nel paragrafo 3.5, dove discuteremo di criteri per arrestare il procedimento iterativo. ■

Cerchiamo a questo punto di tirare le somme e valutare i pro e i contro del metodo di bisezione. Gli aspetti positivi sono tre:

1) il metodo converge globalmente e quindi la scelta di $[a_0, b_0]$ (ovvero di x_0) non è problematica;

2) il metodo fornisce una stima dell'errore ad ogni iterazione (cfr. (3.4));

3) il costo computazionale è molto basso, in quanto ad ogni iterazione si deve calcolare soltanto un valore di f.

L'aspetto negativo è la lentezza con cui ci si avvicina alla soluzione, ossia la bassa velocità di convergenza.

3.3 Metodo di Newton

Per il metodo di Newton, che consideriamo in questo paragrafo, la situazione di pro e contro si rovescia completamente. La convergenza diventa locale e quindi la scelta di x_0 diventa critica; non si hanno stime dell'errore se non da un punto di vista teorico; il costo computazionale è doppio di quello del metodo di bisezione. D'altra parte, quando il metodo converge, presenta una velocità parecchio più elevata del metodo di bisezione.

È intuitivo che si può riuscire ad ottenere un metodo iterativo più veloce del metodo di bisezione solo utilizzando più informazioni su f. In effetti il metodo di bisezione calcola il valore di f nelle iterate, ma in pratica ne sfrutta soltanto il segno. Per andare oltre, possiamo rileggere un'iterazione del metodo di bisezione in modo geometrico: il punto $x_k = \frac{1}{2}(a_k + b_k)$ può essere visto come intersezione con l'asse delle ascisse della retta che passa per i punti

$$(a_k, \mathrm{sgn}(f(a_k))), \quad (b_k, \mathrm{sgn}(f(b_k))),$$

dove la funzione *sgn* è la funzione "segno", che vale $+1$ per argomenti positivi e -1 per argomenti negativi. In altri termini, ad ogni iterazione si sostituisce alla f una funzione lineare e si sceglie x_k uguale alla radice di questa funzione. Possiamo pensare di generalizzare tale procedimento e costruire un metodo che ad ogni iterazione calcoli x_k come intersezione di una particolare retta con l'asse delle ascisse. Non è difficile intuire che più questa retta è "vicina" al grafico di f (nell'intorno del punto in cui ci troviamo) e più ci possiamo aspettare che le intersezioni con l'asse delle ascisse della retta e del grafico siano vicine. Evidentemente per definire rette "vicine" al grafico di f non basta utilizzare i segni, ma occorrono anche

i valori della funzione ed eventualmente i valori della derivata. Quando la funzione f è non solo continua, ma anche derivabile, si possono utilizzare le rette tangenti che sono in assoluto quelle più vicine al grafico. Questo dà origine al metodo di Newton, noto anche come metodo delle tangenti o metodo di Newton-Raphson [2]. Alla prima iterazione del metodo di Newton, data un'approssimazione iniziale x_0 di x^*, si considera la retta tangente al grafico di f nel punto $(x_0, f(x_0))$

$$y = f(x_0) + f'(x_0)(x - x_0)$$

e, supponendo $f'(x_0) \neq 0$, si calcola il punto di intersezione di questa retta con l'asse delle ascisse chiamandolo x_1, cioè

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}.$$

Si procede poi costruendo la retta tangente al grafico di f nel punto $(x_1, f(x_1))$ e calcolando la sua intersezione x_2 con l'asse delle ascisse, ovviamente supponendo $f'(x_1) \neq 0$ (cfr figura 3.2). Iterativamente, nell'ipotesi che $f'(x_k) \neq 0$ per ogni k, si calcola x_{k+1} con la formula

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}. \tag{3.6}$$

L'esempio successivo mostra che il metodo di Newton può essere molto più veloce del metodo di bisezione, ma anche che la scelta di x_0 non è indolore.

Esempio 3.3.1. Consideriamo la funzione $f(x) = arctg(x)$ come nell'esempio 3.2.1. Eseguendo alcune iterazioni a partire da $x_0 = 1.3$ si ottengono i risultati riportati nella tabella 3.3. Come si vede, le iterate si avvicinano a x^*, molto più rapidamente di quanto succede con il metodo di bisezione a parità di x_0 (cfr. tabella 3.1). Se ora prendiamo come punto di partenza $x_0 = 1.4$, scopriamo che in questo caso il metodo di Newton non converge. Come mostra la tabella 3.4, la successione delle iterate oscilla fra valori negativi e valori positivi allontanandosi da x^*; contemporaneamente i valori della funzione oscillano anch'essi fra valori positivi e negativi addensandosi intorno a $-\frac{\pi}{2}$ e $\frac{\pi}{2}$ (ricordiamo che la funzione f ha due asintoti orizzontali: $\lim_{x\to\infty} f(x) = \frac{\pi}{2}$ e $\lim_{x\to-\infty} f(x) = -\frac{\pi}{2}$). ■

[2] Il metodo fu descritto per la prima volta da Isaac Newton (1642-1727) che lo applicò ad una equazione algebrica. La forma era completamente diversa da quella oggi nota e ricalcava forme antiche del procedimento, già descritte nei lavori del matematico persiano Sharaf al-Din al-Muzaffar al-Tusi (1135-1213 circa) e di Erone di Alessandria (10 a.C.-75 d.C. circa). Il metodo fu descritto in forma più vicina a quella moderna da Joseph Raphson nel 1690, sempre con riferimento a equazioni algebriche; infine, fu successivamente generalizzato a equazioni non lineari qualunque da Thomas Simpson nel 1740.

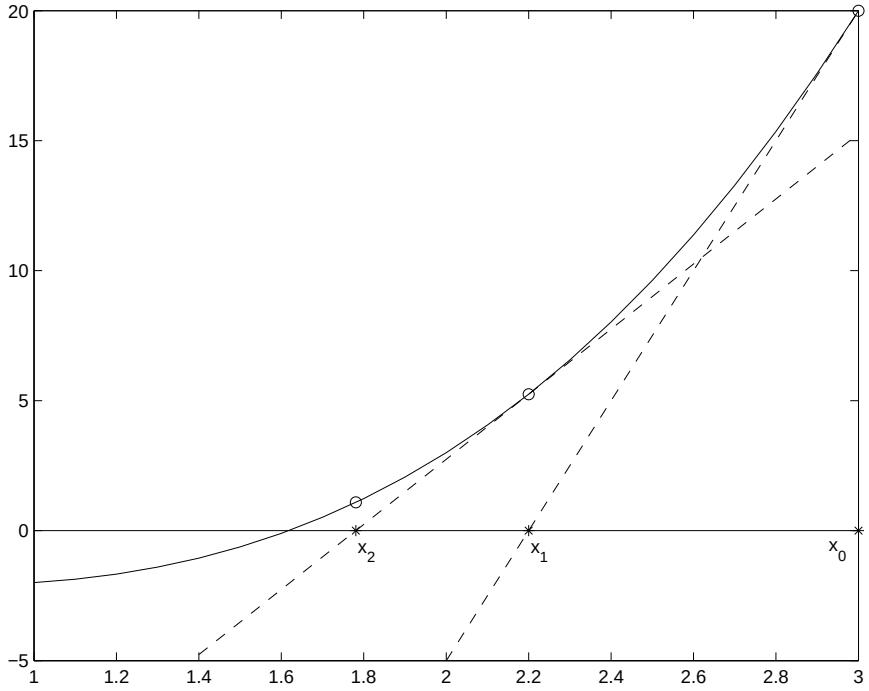

Figura 3.2 Metodo di Newton

k	x_k	$\lvert f(x_k)\rvert$
0	1.300000000000000e+00	9.2e−01
1	−1.161620884488540e+00	8.6e−01
2	8.588963926230877e−01	7.1e−01
3	−3.742406717585654e−01	3.6e−01
4	3.401887344648524e−02	3.4e−02
5	−2.624025442319461e−05	2.6e−05
6	1.204517104005758e−14	1.2e−14

Tabella 3.3 Esempio 3.3.1: metodo di Newton per $f(x) = arctg(x)$, $x_0 = 1.3$

k	x_k	$\lvert f(x_k)\rvert$
0	1.400000000000000e+00	9.5e−01
1	−1.413618648803742e+00	9.6e−01
2	1.450129314628337e+00	9.7e−01
3	−1.550625975637753e+00	1.0e+00
4	1.847054084150190e+00	1.1e+00
5	−2.893562393142410e+00	1.2e+00
6	8.710325846983183e+00	1.5e+00
7	−1.032497737719101e+02	1.6e+00
8	1.654056382723860e+04	1.6e+00
9	−4.297214828964087e+08	1.6e+00

Tabella 3.4 Esempio 3.3.1: metodo di Newton per $f(x) = arctg(x)$, $x_0 = 1.4$

3.3.1 Convergenza del metodo di Newton

Se la derivata prima e la derivata seconda di f non cambiano mai segno il metodo di Newton converge globalmente, come si può intuire osservando la figura 3.2. Ma in generale, quando la funzione presenta massimi, minimi e/o cambi di concavità, la convergenza del metodo di Newton è soltanto locale e la scelta di x_0 è spesso abbastanza difficile. La convergenza locale del metodo di Newton è dimostrata nel seguente teorema nell'ipotesi che $f'(x^*) \neq 0$, ovvero che x^* sia una radice semplice; rimandiamo a [15] per risultati relativi a radici non semplici.

Teorema 3.3.1. Sia x^* una soluzione dell'equazione $f(x) = 0$. Supponiamo che f sia derivabile due volte e f'' sia continua in un intorno di x^*. Se $f'(x^*) \neq 0$, allora esiste un $\delta > 0$ tale che per ogni $x_0 \in [x^* - \delta, x^* + \delta]$ la successione $\{x_k\}$ generata dal metodo di Newton converge a x^*.

Dimostrazione. La dimostrazione si articola in due passi: prima si ricava un'espressione dell'errore e_k in termini di e_{k-1} e poi si usa l'espressione trovata per dimostrare che $\lim_{k\to\infty} e_k = 0$.
Consideriamo la funzione

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

Poiché $f'(x^*) \neq 0$ e f' è continua in un intorno di x^*, la funzione ϕ è definita e continua per x in tale intorno. Inoltre ϕ è derivabile nel solito intorno con

$$\phi'(x) = \frac{f(x)f''(x)}{f'(x)^2}$$

e, essendo continue f, f' e f'', anche ϕ' è continua. La (3.6) implica che

$$\phi(x_{k-1}) = x_k$$

per ogni $k \geq 1$; inoltre, siccome per ipotesi $f(x^*) = 0$ e $f'(x^*) \neq 0$,

$$\phi(x^*) = x^* \ \text{ e } \ \phi'(x^*) = 0.$$

Poiché ϕ' si annulla in x^* ed è continua intorno a x^*, si deduce che comunque si fissi una costante $C > 0$, esiste un δ dipendente da C tale che $|\phi'(x)| \leq C$ per ogni $x \in I_\delta = [x^* - \delta, x^* + \delta]$. Fissiamo un valore di C minore di 1. Usando le proprietà della funzione ϕ e il Teorema del valor medio possiamo scrivere per ogni $k \geq 1$

$$e_k = x_k - x^* = \phi(x_{k-1}) - \phi(x^*) = \phi'(\xi_{k-1})(x_{k-1} - x^*) = \phi'(\xi_{k-1})e_{k-1}, \tag{3.7}$$

con ξ_{k-1} che sta tra x_{k-1} ed x^*.

Consideriamo la prima iterazione ($k = 1$) e supponiamo $x_0 \in I_\delta$. Allora anche $\xi_0 \in I_\delta$ e dalla (3.7) otteniamo

$$|e_1| = |\varphi'(\xi_0)||e_0| \leq C|e_0| \leq C\delta < \delta \tag{3.8}$$

e quindi $x_1 \in I_\delta$. Per lo stesso ragionamento anche $x_2 \in I_\delta$ ed inoltre vale

$$|e_2| \leq C|e_1| \leq C^2|e_0|.$$

Andando avanti così, vediamo che $x_k \in I_\delta$ per ogni k e

$$|e_k| \leq C|e_{k-1}| \leq C^k|e_0|. \tag{3.9}$$

Siccome $C < 1$, la successione $\{C^k\}$ tende a zero per k tendente a ∞ e per il Teorema dei "carabinieri" segue che $\lim_{k\to\infty} e_k = 0$, cioè la successione $\{x_k\}$ è convergente ad x^*. È importante ricordarsi che la convergenza non è garantita per ogni scelta di x_0: per arrivare alla (3.9) abbiamo dovuto supporre $x_0 \in I_\delta$. □

Ora vogliamo stabilire con che velocità il metodo converge. A tale scopo supponiamo che in un intorno di x^* la funzione f ammetta anche derivata terza continua; da questo segue che ϕ ammette derivata seconda continua

$$\phi''(x) = \frac{[f'(x)f''(x) + f(x)f'''(x)]f'(x) - 2f(x)f''(x)^2}{f'(x)^3}$$

con

$$\phi''(x^*) = \frac{f''(x^*)}{f'(x^*)}.$$

Possiamo allora usare lo sviluppo in serie di Taylor, che ci dà

$$e_k = \phi(x_{k-1}) - \phi(x^*) = \phi'(x^*)(x_{k-1} - x^*) + \frac{1}{2}\phi''(\eta_{k-1})(x_{k-1} - x^*)^2$$

con η_{k-1} che sta tra x_{k-1} e x^*. Ricordando che $\phi'(x^*) = 0$, otteniamo

$$e_k = \frac{1}{2}\phi''(\eta_{k-1})e_{k-1}^2,$$

da cui segue

$$\lim_{k\to\infty} \frac{e_k}{e_{k-1}^2} = L \tag{3.10}$$

dove abbiamo posto

$$L = \frac{1}{2}\lim_{k\to\infty} \phi''(\eta_{k-1}) = \frac{1}{2}\phi''(x^*).$$

Dalla (3.10) segue che $e_k \simeq Le_{k-1}^2$ per k sufficientemente grande, ovvero che da un certo punto in poi l'errore ad una data iterazione è uguale, a meno di un fattore L, al quadrato dell'errore all'iterazione precedente. Quando un metodo iterativo è tale per cui la successione degli errori soddisfa la relazione (3.10) per qualche costante $L \neq 0$, si dice che il metodo converge quadraticamente o che ha convergenza quadratica [3]. Il significato pratico della convergenza quadratica è illustrato dal seguente esempio.

Esempio 3.3.2. Utilizziamo il metodo di Newton per calcolare $\sqrt{2}$, ovvero per risolvere l'equazione

$$x^2 - 2 = 0.$$

Partendo da $x_0 = 10$ si ottengono i risultati della tabella 3.5, nella quale riportiamo per ogni k l' errore assoluto $|e_k|$: poichè x^* è vicino a 1, l'ordine di grandezza dell'errore assoluto è uguale a quello dell'errore relativo e dice quante cifre di mantissa esatte ci sono in x_k. Dalla tabella si vede che dopo le prime tre iterazioni la successione degli errori procede secondo una legge quadratica e, di fatto, ad ogni iterazione si raddoppia il numero di cifre esatte. Questa è l'essenza della convergenza quadratica. ■

Il numero di iterazioni della fase di avvicinamento alla soluzione che precede la fase di convergenza quadratica può essere anche molto grande: come mostrato nell'esempio successivo [35] questo dipende da x_0.

Esempio 3.3.3. Per calcolare il reciproco di un numero positivo a si può risolvere l'equazione

$$\frac{1}{x} - a = 0.$$

[3] Talvolta, per esempio se $f(x) = sin(x)$ e x^* è una radice qualunque di f, può succedere che sia $L = 0$. In queste situazioni si possono ripetere i passaggi fatti usando lo sviluppo di Taylor fino a termini di grado più alto del secondo e dimostrare che la convergenza è più veloce nel senso che $\lim_{k\to\infty} \frac{e_k}{e_{k-1}^p} = L$ per qualche $p > 2$ e $L \neq 0$.

k	x_k	$\lvert f(x_k)\rvert$	$\lvert e_k\rvert$
0	1.000000000000000e+01	9.8e+01	8.6e+00
1	5.100000000000000e+00	2.4e+01	3.7e+00
2	2.746078431372549e+00	5.5e+00	1.3e+00
3	1.737194874379598e+00	1.0e+00	3.2e−01
4	1.444238094866232e+00	8.6e−01	3.0e−02
5	1.414525655148738e+00	8.8e−04	3.1e−04
6	1.414213596802269e+00	9.7e−08	3.4e−08
7	1.414213562373095e+00	8.9e−16	2.2e−16

Tabella 3.5 Esempio 3.3.2: metodo di Newton per $f(x) = x^2 - 2$

È facile vedere che il metodo di Newton converge a partire da un qualunque $x_0 \in (0, x^*) = (0, \frac{1}{a})$. In particolare, se $a < 1$ si può scegliere $x_0 = a$, ma questa scelta può non essere particolarmente efficiente quando a è molto piccolo e, di conseguenza, x^* è molto grande. Osserviamo che la formula del metodo di Newton per il nostro problema diventa

$$x_{k+1} = 2x_k - ax_k^2$$

e quindi

$$e_{k+1} = x_{k+1} - \frac{1}{a} = 2x_k - ax_k^2 - \frac{1}{a} = -a(x_k^2 - \frac{2x_k}{a} + \frac{1}{a^2}) = -a(x_k - \frac{1}{a})^2 = -ae_k^2,$$

ovvero

$$e_{k+1} = ae_k^2.$$

Esaminando questa relazione si deduce che $|e_{k+1}| < |e_k|$ se $|e_k| < \frac{1}{a}$. Consideriamo ora un valore di a molto piccolo e scegliamo $x_0 = a$. Allora si ha $|e_0| = |a - \frac{1}{a}|$ da cui si vede che $|e_0|$ è minore di $\frac{1}{a}$ e molto vicino a questo valore. Per quanto detto prima avremo $|e_1| < |e_0|$, ma $|e_1|$ ancora grande, vicino a $\frac{1}{a}$. Lo stesso ragionamento può essere fatto per $|e_2|$, $|e_3|$ e così via, con la conclusione che la diminuzione dell'errore sarà molto lenta nella fase iniziale. Per poter sentire i vantaggi della convergenza quadratica occorre che $|e_k|$ diventi minore di $\frac{1}{\sqrt{a}}$, così che $|e_{k+1}|$ diventa minore di 1. La fase iniziale può richiedere molte iterazioni, tante più quanto più a è piccolo. A titolo di esempio riportiamo nella tabella 3.6 i risultati ottenuti per $a = 1.5367 \times 10^{-10}$ e $x_0 = a$.
Fortunatamente è semplice trovare un buon valore di partenza per questo problema osservando che conosciamo a priori l'ordine di grandezza della soluzione: siccome $a = 1.5367 \times 10^{-10}$, $\frac{1}{a}$ è circa 10^{10}. Possiamo allora usare $x_0 = 10^{10}$. I risultati riportati in tabella 3.7 dimostrano la bontà di questa scelta.

■

k	x_k	$\lvert f(x_k)\rvert$
0	1.53670000000000e−10	6.5e+09
1	3.07340000000000e−10	3.3e+09
2	6.14680000000000e−10	1.6e+09
3	1.22936000000000e−09	8.1e+08
10	1.57358080000000e−07	6.4e+06
20	1.61134673919998e−04	6.2e+03
30	1.65001906091988e−01	6.1e+00
40	1.68961949646842e+02	5.9e−03
50	1.73014738657181e+05	5.8e−06
60	1.77179411981661e+08	5.6e−09
65	3.78446875605145e+09	1.1e−10
66	5.36804445945396e+09	3.3e−11
67	6.30794936326723e+09	4.9e−12
68	6.50133482473072e+09	1.4e−13
69	6.50744528294603e+09	1.4e−16

Tabella 3.6 Esempio 3.3.3: metodo di Newton per $f(x) = \frac{1}{x} - 1.5367 \times 10^{-10}$

k	x_k	$\lvert f(x_k)\rvert$
0	1.00000000000000e+10	5.4e−11
1	4.63300000000000e+09	6.2e−11
2	5.96752124137000e+09	1.4e−11
3	6.46265251096772e+09	1.1e−12
4	6.50714262996534e+09	7.3e−15

Tabella 3.7 Esempio 3.3.3: metodo di Newton per $f(x) = \frac{1}{x} - 1.5637 \times 10^{-10}$

3.3.2 Varianti del metodo di Newton

Un inconveniente del metodo di Newton è la necessità di calcolare ad ogni iterazione la derivata di f in x_k. Il calcolo di un valore della derivata può essere molto più costoso del calcolo di un valore della funzione e quindi il costo computazionale del metodo di Newton può essere più del doppio di quello del metodo di bisezione. Un modo per evitare questo costo aggiuntivo è "congelare" la derivata, usando ad ogni iterazione $f'(x_0)$ invece di $f'(x_k)$. In questo modo si ottiene il metodo di Newton stazionario, noto anche come metodo delle corde, che è descritto dalla formula:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_0)}. \tag{3.11}$$

Geometricamente, in questo metodo si approssima ad ogni iterazione il grafico della funzione con la retta che passa per il punto $(x_k, f(x_k))$ e ha pendenza uguale a quella della tangente nel punto iniziale. Il metodo è ovviamente poco costoso, ma purtroppo ha una velocità di convergenza bassa, simile a quella del metodo di bisezione [24].

Un'alternativa più valida, utile anche in quelle applicazioni nelle quali f non è data in forma analitica e quindi la derivata non è disponibile, consiste nel sostituire a $f'(x_k)$ un rapporto incrementale

$$r_k = \frac{f(x_k + h_k) - f(x_k)}{h_k}$$

e quindi usare la formula iterativa

$$x_{k+1} = x_k - \frac{f(x_k)}{r_k}. \tag{3.12}$$

Questa scelta corrisponde ad usare la retta che passa per i punti $(x_k, f(x_k))$ e $(x_k + h_k, f(x_k + h_k))$ invece della retta tangente in $(x_k, f(x_k))$ e porta al cosiddetto metodo di Newton alle differenze. Si intuisce che se h_k è sufficientemente piccolo il metodo funzionerà in pratica come il metodo di Newton originale.

Per scegliere h_k dobbiamo aver presente che l'aritmetica floating point ci impedisce di usare numeri "piccoli a piacere". Come prima cosa h_k deve rispettare il vincolo

$$|h_k| \geq \epsilon_m x_k,$$

altrimenti avremmo $fl(x_k + h_k) = fl(x_k)$ e quindi $r_k = 0$. Ma questo non basta: se infatti i due valori $f(x_k + h_k)$ e $f(x_k)$ sono molto vicini fra loro, il calcolo della loro differenza provoca un errore di cancellazione numerica, che deve essere assolutamente evitato. Volendo trovare un modo per definire r_k indipendente dalla particolare funzione f, l'unica cosa che possiamo fare è cercare di mantenere abbastanza diversi i due valori $x_k + h_k$ e x_k, sperando poi che la funzione f non abbatta troppo questa differenza. In pratica, la scelta consigliata è la seguente:

$$h_k = \begin{cases} \sqrt{\epsilon_m} x_k & \text{se} \quad x_k \neq 0 \\ \\ \sqrt{\epsilon_m} & \text{se} \quad x_k = 0 \end{cases} \tag{3.13}$$

che assicura che le mantisse di x_k e $x_k + h_k$ abbiano le prime cifre (circa la metà) uguali e le restanti diverse così che l'errore di cancellazione viene in generale evitato.

Esempio 3.3.4. A titolo di esempio consideriamo l'equazione

$$f(x) = x^3 - 100x^2 - x + 100$$

che ammette $x^* = 100$ come soluzione. Nelle tavole 3.8 e 3.9 si riportano i risultati relativi all'approssimazione di x^* a partire da $x_0 = 80$ con i metodi di Newton e di Newton alle differenze. Come si vede, il comportamento è sostanzialmente identico: ad ogni iterazione la differenza fra le iterate prodotte dai due metodi cade oltre la settima-ottava cifra di mantissa.

■

k	x_k	$\lvert f(x_k)\rvert$
0	8.000000000000000e+01	1.3e+05
1	1.200062519537355e+02	2.9e+05
2	1.050028005654527e+02	5.5e+04
3	1.004352740461875e+02	4.4e+03
4	1.000037407942445e+02	3.7e+01
5	1.000000002798674e+02	2.8e−03
6	9.999999999999999e+01	1.2e−10
7	1.000000000000000e+02	0

Tabella 3.8 Esempio 3.3.4: metodo di Newton per $f(x) = x^3 - 100x^2 - x + 100$

k	x_k	$\lvert f(x_k)\rvert$
0	8.000000000000000e+01	1.3e+05
1	1.200062507324600e+02	2.9e+05
2	1.050028003616229e+02	5.5e+04
3	1.004352741015949e+02	4.4e+03
4	1.000037408088659e+02	3.7e+01
5	1.000000002800324e+02	2.8e−03
6	1.000000000000000e+02	1.2e−10
7	1.000000000000000e+02	0

Tabella 3.9 Esempio 3.3.4: metodo di Newton alle differenze per $f(x) = x^3 - 100x^2 - x + 100$

3.4 Metodo delle secanti

Ad ogni iterazione del metodo di Newton alle differenze si devono calcolare due valori di f: $f(x_k)$ e $f(x_k + h_k)$. Per evitare la valutazione di f nel punto $x_k + h_k$ si può prendere $h_k = x_{k-1} - x_k$, da cui segue

$$r_k = \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}.$$

Si ottiene allora la formula iterativa:

$$x_{k+1} = x_k - f(x_k)\frac{(x_k - x_{k-1})}{f(x_k) - f(x_{k-1})} \tag{3.14}$$

che definisce il metodo delle secanti, così chiamato perché r_k è il coefficiente angolare della retta secante il grafico nei punti $(x_k, f(x_k))$ e $(x_{k-1}, f(x_{k-1}))$ (cfr. figura 3.3).
Dalla formula (3.14) appare chiaro che questo metodo ha bisogno di due punti di partenza che chiameremo x_{-1} e x_0: essi vengono scelti di solito come estremi di un intervallo, sufficientemente piccolo, in cui cade una radice di f, ovvero tali che $f(x_{-1})f(x_0) < 0$. Si nota anche dalla (3.14)

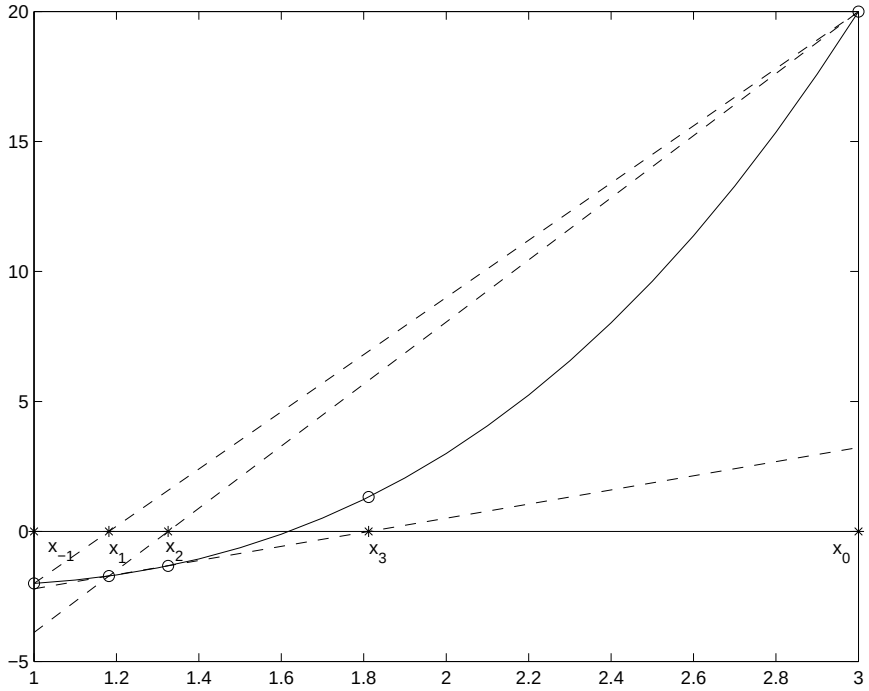

Figura 3.3 Metodo delle secanti

che il metodo si blocca se, a qualche iterazione, $f(x_k) = f(x_{k-1})$, ovvero se la retta secante è parallela all'asse delle ascisse. Un possibile modo per ovviare a questa "impasse" è provare a fare un'iterazione del metodo di Newton alle differenze, per poi riprendere il metodo delle secanti all'iterazione successiva.

Il costo di un'iterazione del metodo delle secanti è uguale a quello del metodo di bisezione: si deve calcolare soltanto $f(x_k)$, avendo già calcolato $f(x_{k-1})$ alla precedente iterazione. Nonostante questo, la sua velocità di convergenza è maggiore, collocandosi in un certo senso a metà fra quella del metodo di bisezione e quella del metodo di Newton.

Esempio 3.4.1. Utilizziamo il metodo delle secanti per calcolare $\sqrt{2}$, come nell'esempio 3.3.2. Operando in doppia precisione con $x_{-1} = 0$ e $x_0 = 10$, otteniamo i risultati riportati in tabella 3.10. Si osserva una diminuzione dell'errore a partire da $k = 4$ abbastanza rapida, anche se non come quella che si riscontra usando il metodo di Newton (cfr. tabella 3.5). ■

Lo studio delle proprietà di convergenza del metodo delle secanti è abbastanza difficile e rinviamo per i dettagli a [11]. In breve, si può dimostrare che esso converge localmente sotto le stesse ipotesi che garantiscono

k	x_k	$\lvert f(x_k)\rvert$	$\lvert e_k\rvert$
−1	0	2.0e+00	1.4e+00
0	10	9.8e+01	8.6e+00
1	1.999999999999993e−01	2.0e+00	1.2e+00
2	3.921568627450974e−01	1.9e+00	1.0e+00
3	3.509933774834446e+00	1.0e+01	2.1e+00
4	8.652911813643911e−01	1.3e+00	5.5e−01
5	1.151281315352864e+00	6.8e−01	2.6e−01
6	1.485785199551161e+00	2.1e−01	7.3e−02
7	1.407077416477727e+00	2.0e−02	7.1e−03
8	1.414037008657048e+00	5.0e−04	1.8e−04
9	1.414214008974291e+00	1.3e−06	4.5e−07
10	1.414213562345216e+00	7.9e−11	2.8e−11
11	1.414213562373095e+00	4.4e−16	0

Tabella 3.10 Esempio 3.4.1: metodo delle secanti per $f(x) = x^2 - 2$

la convergenza del metodo di Newton (cfr. Teorema 3.3.1) e che

$$\lim_{k\to\infty} \frac{|e_k|}{|e_{k-1}|^p} = L, \tag{3.15}$$

dove $p = \frac{1}{2}(1+\sqrt{2}) \simeq 1.618$ e $L = \left(\frac{|f''(x^*)|}{2|f'(x^*)|}\right)^{\frac{1}{p}}$. Essendo $p > 1$, si dice che il metodo delle secanti ha convergenza superlineare [4]. La (3.15) implica che per k sufficientemente grande $|e_k| \simeq L|e_{k-1}|^p$, il che prefigura una diminuzione degli errori più veloce di quella del metodo di bisezione e un po' più lenta di quella del metodo di Newton, ma ad essa molto vicina.

In conclusione, abbiamo studiato tre metodi con diverse caratteristiche. Il metodo di bisezione è l'unico che converge globalmente; esso ha un basso costo computazionale, ma può richiedere molte iterazioni per raggiungere un'accuratezza "ragionevole", a causa della sua lentezza. Il metodo delle secanti converge solo localmente, però ha lo stesso costo computazionale del metodo di bisezione ed è molto più veloce. Infine, il metodo di Newton è ancora localmente convergente e molto veloce, più del metodo delle secanti, però ha un costo computazionale doppio. Le caratteristiche di velocità dei tre metodi sono riassunte e messe a confronto nella tabella 3.11, che si riferisce alla risoluzione dell'equazione [5]

$$x^3 - 2x - 5 = 0.$$

[4] Come già abbiamo notato per il metodo di Newton, la convergenza può essere più veloce quando $L = 0$.

[5] Citiamo da [23], pag. 250: This example is a polynomial which could be treated by more specialized algorithms, but we have used here because of its historical interest. The following excerpt from a letter written by de Morgan to Whewell in 1861 is contained in the book by Whittaker and Robinson (The calculus of observations, 1924): "The

Nella tabella riportiamo le successioni $\{x_k\}$ generate dal metodo di bise-

k	Bisezione	Secanti	Newton
0	2.500000000000000	3.000000000000000	3.000000000000000
1	2.250000000000000	2.058823529411765	2.360000000000000
2	2.125000000000000	2.081263659845023	2.127196780158816
3	2.062500000000000	2.094824146094052	2.095136036933634
4	2.093750000000000	2.094549431035247	2.094551673824268
5	2.109375000000000	2.094551481227599	2.094551481542347
6	2.101562500000000	2.094551481542327	2.094551481542327
7	2.097656250000000		
8	2.095703125000000		
9	2.094726562500000		
10	2.094238281250000		
20	2.094551563262940		
25	2.094551488757134		
30	2.094551481772214		
35	2.094551481553935		
40	2.094551481542567		
45	2.094551481542325		
48	2.094551481542327		

Tabella 3.11 Metodi di bisezione, secanti e Newton per $f(x) = x^3 - 2x - 5$

zione con $[a_0, b_0] = [2, 3]$, dal metodo delle secanti con $x_{-1} = 2$ e $x_0 = 3$ e dal metodo di Newton con $x_0 = 3$. Nelle ultime iterate relative ad ogni metodo sono evidenziate in grassetto le cifre buone, in modo da mostrare l'andamento dell'errore. Questi risultati sono un'ulteriore conferma di quanto emerso da tutti i precedenti esempi.

Chiudiamo il paragrafo con alcune considerazioni relative al metodo delle secanti e a quello di Newton. Se il costo di una valutazione di funzione non è eccessivamente alto e si dispone di un buon punto di partenza, sicuramente il metodo da preferire è il metodo di Newton (nella variante alle differenze) perché è quello che può fornirci il risultato con il minor numero di iterazioni. D'altra parte, se una valutazione della funzione è molto costosa, allora conviene utilizzare il metodo delle secanti, perché è quello che meglio coniuga l'esigenza di fare poche iterazioni con quella di spendere il minor tempo possibile in ogni iterazione.

A questo proposito citiamo il fatto che uno degli algoritmi più robusti ed efficienti che si conoscano per la risoluzione di equazioni non lineari è un algoritmo ibrido, che combina il metodo di bisezione ed il metodo

reason I call $x^3 - 2x - 5 = 0$ a celebrated equation is because it was the one on which Wallis chanced to exhibit Newton's method when he first published it, in consequence of which every numerical solver has felt bound in duty to make it of his examples. Invent a numerical method, neglect to show how it works on this equation, and you are a pilgrim who does not come in at the little wicket."

delle secanti, pubblicato nel 1969 [13]. Come il metodo di bisezione, a partire da un intervallo $[a_0, b_0]$ in cui si trova x^*, l'algoritmo genera una successione di intervalli di ampiezza tendente a zero, tutti contenenti x^*. Il metodo delle secanti interviene nel modo seguente. Ad ogni iterazione, dato $[a_k, b_k]$, l'algoritmo calcola il punto x_k con il metodo delle secanti usando $x_{k-2} = a_k$ e $x_{k-1} = b_k$. Se x_k cade dentro all'intervallo e non è "troppo vicino" ad uno degli estremi, allora il nuovo intervallo $[a_{k+1}, b_{k+1}]$ viene preso uguale a $[a_k, x_k]$ o $[x_k, b_k]$ (a seconda dei segni di f nei tre punti coinvolti); se invece x_k non soddisfa i requisiti suddetti, viene eseguita un'iterazione del metodo di bisezione. Il risultato è che in generale, ad una prima fase di avvicinamento in cui predomina il metodo di bisezione, segue una fase di convergenza veloce in cui, predominando il metodo delle secanti, l'ampiezza degli intervalli viene ridotta ad ogni iterazione di un fattore molto più grande di $\frac{1}{2}$. Chi fosse interessato a maggiori dettagli può vedere la realizzazione dell'algoritmo in MATLAB o quella in linguaggio FORTRAN (reperibile sul sito *www.netlib.org*): entrambi i codici hanno il nome fzero.

3.5 Criteri di arresto

In vista della costruzione di algoritmi relativi ai metodi visti nei paragrafi precedenti, occorre stabilire dei criteri di arresto che permettano di fermare il procedimento ad una certa iterazione k avendo la (ragionevole) certezza di aver risolto il problema con l'accuratezza desiderata.

Nel corso di questo paragrafo verranno analizzati alcuni possibili criteri di arresto. Prima di ogni altra considerazione vogliamo però anticipare che non è possibile stabilire un criterio che sia adatto ad ogni situazione: la scelta di quale o quali criteri adottare dipende sia dal metodo scelto che dalle conoscenze che si hanno sul particolare problema da risolvere. Tutti i criteri prevedono l'introduzione di alcuni valori di soglia, che noi chiameremo tolleranze, che quantificano le richieste di accuratezza. Al momento dell'esecuzione degli algoritmi, i valori assunti dalle tolleranze non potranno prescindere dalla precisione della macchina su cui si lavora. Più precisamente, le tolleranze dovranno essere sempre maggiori della precisione di macchina ϵ_m; in caso contrario, staremmo pretendendo di calcolare il risultato con una precisione maggiore di quella che la macchina utilizza per memorizzare i dati! In generale, a meno di non aver necessità della massima accuratezza possibile, i valori delle tolleranze vengono presi vicini a $\sqrt{\epsilon_m}$, ad esempio 10^{-3} o 10^{-4} se si lavora in precisione semplice, 10^{-8} o 10^{-10} se si lavora in doppia precisione.

3.5.1 Un criterio di salvaguardia

Qualunque sia il metodo che stiamo usando e qualunque sia il criterio di arresto utilizzato, è consigliabile prevedere sempre un numero massimo di

iterazioni K_{max} per fermare il procedimento iterativo quando

$$k > K_{max}. \tag{3.16}$$

Questo criterio sembra poco significativo perché non è assolutamente legato al problema che stiamo risolvendo e non assicura in nessun senso che x_k sia una buona approssimazione della soluzione. D'altra parte esso rappresenta una salvaguardia di fondamentale importanza quando si ha a che fare con metodi che convergono solo localmente, come il metodo di Newton o il metodo delle secanti, per i quali la successione $\{x_k\}$ può anche divergere e nessun criterio di arresto ragionevole sarà mai soddisfatto. Per questi metodi si possono anche definire dei criteri che permettono di arrestare l'algoritmo in situazioni di non convergenza: rimandiamo chi fosse interessato all'articolo [30]. Il criterio (3.16) può essere utile anche in connessione con metodi globalmente convergenti come il metodo di bisezione, per assicurare che l'algoritmo si fermi comunque nel caso che le richieste di accuratezza non siano adeguate alla macchina con cui si lavora.

3.5.2 Un criterio di arresto generale

Prendiamo in considerazione il primo criterio di arresto a cui si può pensare, che è anche il più comunemente usato. Dato che stiamo cercando un valore x^* per cui $f(x^*) = 0$, si può pensare di arrestare il procedimento se, ad una certa iterazione k, il valore di $f(x_k)$ è abbastanza vicino a zero, ossia se

$$|f(x_k)| < \eta, \tag{3.17}$$

dove $\eta > \epsilon_m$ è la tolleranza.

Il criterio (3.17) è perfettamente adeguato alle situazioni in cui ci interessa non tanto trovare un valore di x vicino ad x^*, quanto un valore di x in corrispondenza del quale f assume un valore vicino a zero. Questa precisazione si rende necessaria perché il verificarsi di (3.17) non implica in generale che x_k sia vicino a x^*, né tantomeno che la distanza fra x_k e x^* sia minore di η. Infatti il rapporto che intercorre fra $|x_k - x^*|$ e $|f(x_k)|$ dipende dal comportamento di f intorno a x^*, in particolare da quanto vale $f'(x^*)$, come gli esempi dei paragrafi precedenti ci dimostrano. Negli esempi 3.2.1 e 3.3.2 l'errore $|x_k - x^*|$ e $|f(x_k)|$ hanno lo stesso ordine di grandezza per k sufficientemente grande perché $f'(x^*)$ è uguale o vicino a 1. Ma nell'esempio 3.2.2 vediamo due situazioni estreme nelle quali $f'(x^*) = C$ è molto grande o molto piccola in valore assoluto: se arrestassimo il procedimento iterativo mediante il criterio (3.17) con $\eta = 10^{-9}$, per $C = -10^{-7}$ ci si fermerebbe alla sesta iterazione, quando l'errore $|x_6 - x^*|$ è molto più grande di η (circa 10^{-3}), e per $C = 10^7$ dovremmo procedere oltre la 46-esima iterazione, quando l'ampiezza dell'intervallo $[a, b]$ è pericolosamente vicina alla precisione di macchina.

Una variante del criterio (3.17) che tiene conto di queste considerazioni può assumere la forma

$$|f(x_k)| < \eta_r |\widetilde{f}(x_k)| + \eta_a \tag{3.18}$$

dove η_r e η_a sono due tolleranze e $\widetilde{f}(x_k)$ coincide con $f'(x_k)$ o una sua approssimazione. Questo criterio è eventualmente da utilizzare se si lavora con il metodo di Newton o Newton alle differenze, quando non c'è un costo aggiuntivo nella valutazione di $\widetilde{f}(x_k)$. Per spiegare il significato del criterio (3.18) supponiamo per semplicità $\eta_r = \eta_a$. Se $|\widetilde{f}(x_k)|$ è piccolo il termine η_a predomina sul termine $\eta_r|\widetilde{f}(x_k)|$ e il criterio diventa di fatto equivalente al criterio assoluto (3.17); se invece $|\widetilde{f}(x_k)|$ è grande il termine η_a diventa trascurabile e il criterio diventa un criterio relativo:

$$|f(x_k)| \leq \eta_r |\widetilde{f}(x_k)|.$$

In altri termini, il criterio (3.18) è un criterio misto, che da solo si adatta alle diverse situazioni.

Il rischio insito in questo criterio è il seguente: poiché (per ovvi motivi) in esso non interviene $f'(x^*)$, ma $f'(x_k)$, si rischia che il procedimento venga fermato in una zona del dominio in cui la funzione ha una forte pendenza anche se la pendenza intorno alla soluzione non è elevata. Per questo motivo consigliamo di usarlo soltanto quando siamo sicuri di essere vicino alla soluzione, ad esempio quando si sa che x_0 è una buona approssimazione di x^*.

3.5.3 Un criterio di arresto per il metodo di bisezione

Dato che il metodo di bisezione genera una successione di intervalli di ampiezza tendente a zero, ognuno dei quali contiene la soluzione, un criterio di arresto naturale consiste nel fermarsi quando l'intervallo è sufficientemente piccolo, ovvero:

$$(b_k - a_k) < \sigma, \tag{3.19}$$

dove σ è la tolleranza scelta. Questo criterio garantisce, in virtù della (3.3), che

$$|x_k - x^*| < \frac{\sigma}{2}.$$

A causa delle limitazioni imposte dalla rappresentazione floating point dei numeri reali, questo criterio può risultare pericoloso quando x^* è un numero grande in valore assoluto. In questo caso infatti $|a_k|$ e $|b_k|$ tendono al crescere di k a divenire grandi, dello stesso ordine di grandezza di x^*, e la richiesta (3.19) può divenire "irragionevole" anche per valori di σ "ragionevolmente" maggiori di ϵ_m. Per fare un esempio supponiamo di usare $\sigma = 10^{-4}$, valore accettabile sia in precisione semplice che in precisione

doppia. Allora il criterio (3.19) arresterà l'algoritmo quando a_k e b_k diventano uguali fino ai decimillesimi. Se $x^* \simeq 10^6$, anche a_k e b_k tendono al crescere di k a valori di questo ordine di grandezza e il criterio (3.19) può essere soddisfatto soltanto se a_k e b_k hanno in comune tutte le 6-7 cifre della parte intera e 4 cifre della parte frazionaria. Questo equivale a richiedere che a_k e b_k abbiano 10-11 cifre di mantissa uguali. Se stiamo lavorando in precisione doppia questa è una richiesta accettabile, ma non lo è sicuramente se stiamo lavorando in precisione semplice (ricordiamo che lavorare in precisione semplice o doppia equivale a lavorare con circa 8 o 16 cifre di mantissa decimale rispettivamente).

Per evitare situazioni come quella descritta conviene sostituire (3.19) con un criterio misto, con la stessa logica di (3.18):

$$(b_k - a_k) \leq \sigma_r |a_k| + \sigma_a. \tag{3.20}$$

3.5.4 Un criterio di arresto per metodi a convergenza almeno superlineare

Quando si lavora con un metodo a convergenza superlineare o quadratica, come il metodo delle secanti e il metodo di Newton, si può utilizzare anche un criterio di arresto basato sulla vicinanza di due iterate successive. Consideriamo la distanza $|x_k - x_{k-1}|$. Utilizzando la disuguaglianza triangolare si ottiene

$$|x_k - x_{k-1}| = |x_k - x^* + x^* - x_{k-1}| = |e_k - e_{k-1}| \leq |e_k| + |e_{k-1}|; \tag{3.21}$$

d'altra parte, usando l'altra nota disuguaglianza $|y - z| \geq ||y| - |z||$ e supponendo $|e_{k-1}| > |e_k|$, si ottiene

$$|x_k - x_{k-1}| \geq ||e_k| - |e_{k-1}|| = |e_{k-1}| - |e_k|. \tag{3.22}$$

Unendo (3.21) e (3.22) si deduce la seguente catena di disuguaglianze

$$1 - \frac{|e_k|}{|e_{k-1}|} \leq \frac{|x_k - x_{k-1}|}{|e_{k-1}|} \leq 1 + \frac{|e_k|}{|e_{k-1}|}. \tag{3.23}$$

Poiché per metodi a convergenza superlineare e quadratica valgono rispettivamente le relazioni (3.10) e (3.15) che implicano

$$\lim_{k \to \infty} \frac{|e_k|}{|e_{k-1}|} = 0,$$

da (3.23) segue che

$$\lim_{k \to \infty} \frac{|x_k - x_{k-1}|}{|e_{k-1}|} = 1. \tag{3.24}$$

Questa relazione dice che $|x_k - x_{k-1}| \simeq |e_{k-1}|$ quando k è sufficientemente grande e giustifica quindi l'uso del criterio di arresto

$$|x_k - x_{k-1}| \leq \sigma \tag{3.25}$$

o meglio

$$|x_k - x_{k-1}| \leq \sigma_r |x_k| + \sigma_a. \tag{3.26}$$

Sottolineiamo che questo criterio si basa sulla relazione asintotica (3.24) e va quindi utilizzato solo quando si è sicuri che la successione delle iterate stia veramente andando verso x^* con velocità almeno superlineare (ad esempio quando siamo sicuri che x_0 sia una buona approssimazione di x^*). Ripensiamo ad esempio alla tabella 3.6, che esemplifica la possibilità di un andamento molto lento del metodo di Newton nelle prime iterazioni. Se usassimo in quel caso il criterio (3.26) con $\sigma_r = \sigma_a = 10^{-8}$ ci fermeremmo alla prima iterazione!

3.6 Algoritmi

Prima di descrivere gli algoritmi che realizzano i metodi di bisezione, di Newton e delle secanti, occorre fare alcune osservazioni di carattere generale, utili per la loro comprensione.

Prima di tutto qualche parola a proposito dei criteri di arresto. Poiché non è possibile stabilire un criterio che vada sempre bene, per tutti i metodi e per tutti i problemi, abbiamo previsto negli algoritmi soltanto il criterio (3.17) e il criterio di salvaguardia (3.16). Per il metodo di bisezione abbiamo aggiunto il criterio (3.20).

Non abbiamo previsto negli algoritmi un controllo sulla correttezza dei dati (ad esempio nell'algoritmo relativo al metodo di bisezione per verificare che a e b soddisfino il requisito $f(a)f(b) < 0$), lasciando questo compito come esercizio per i lettori. Dal punto di vista tecnico la mancanza di questo controllo non è un problema, dato che la presenza di K_{max} farebbe comunque terminare l'algoritmo in un tempo finito, anche se a e b non fossero corretti.

Gli algoritmi sono scritti per una generica funzione f. Al momento dell'utilizzo si dovrà affiancare all'algoritmo prescelto un altro algoritmo che, dato un valore di x, calcoli $f(x)$.

Tutti gli algoritmi possono interrompersi in più di un modo. Ad esempio può succedere che l'algoritmo termini ad una certa iterazione perché un criterio di arresto è soddisfatto, oppure che termini perché dopo aver eseguito K_{max} iterazioni non risulta ancora soddisfatto nessun criterio di arresto (questo può accadere perché la successione delle iterate non sta convergendo oppure perché le richieste di accuratezza sono eccessive). Per identificare le possibili diverse situazioni abbiamo introdotto fra i risultati di tutti gli algoritmi la variabile *esito* il cui valore è diverso a seconda del motivo per cui l'algoritmo si è arrestato. I valori sono stati scelti in modo

arbitrario: in generale abbiamo associato il valore 0 ad una terminazione con successo e valori diversi da zero agli altri possibili casi.

Negli algoritmi non usiamo la notazione con indici, ad esempio non usiamo la notazione x_k per le iterate. Infatti usare questa notazione significa implicitamente conservare tutte le iterate come valori distinti, disponibili tutte in qualsiasi momento della procedura. Dal punto di vista della realizzazione degli algoritmi in un qualsiasi linguaggio di programmazione, questo ha un costo perché implica che si occupi spazio in memoria per conservare tutti questi numeri. Così, poiché in generale siamo interessati soltanto all'ultima iterata, negli algoritmi usiamo il simbolo x che al variare di k rappresenta l'iterata corrente x_k.

3.6.1 Algoritmo di bisezione

Presentiamo prima di tutto un algoritmo relativo al metodo di bisezione. A parte i semplici esempi introduttivi del Capitolo 1, questo è il primo algoritmo che scriviamo; per questo motivo, i commenti e le spiegazioni che lo seguono sono molto dettagliati, con un livello di dettaglio che non manterremo per gli algoritmi successivi.

Algoritmo 3.6.1. Algoritmo di bisezione.
Dati: $a, b, K_{max}, \eta, \sigma_a, \sigma_r$
Risultati:

	esito	indicatore del risultato, con i seguenti valori:
		$esito = 0$ se il procedimento è terminato con successo
		$esito = 1$ se sono state fatte K_{max} iterazioni senza successo
	a, b	estremi dell'ultimo intervallo calcolato
	x	ultima iterata calcolata
	y	valore di f in x
	k	numero di iterazioni effettuate

1. $f_a = f(a)$
2. Per $k = 1, 2, \ldots, K_{max}$
   1. $x = \frac{a+b}{2}$
   2. $y = f(x)$
   3. Se $|y| < \eta$, allora:
      $esito = 0$ e fermati
      Fine scelta
   4. Se $(b - a) \leq \sigma_r|a| + \sigma_a$, allora:
      $esito = 0$ e fermati
      Fine scelta
   5. Se $sgn(f_a) \times y > 0$, allora:
      $a = x$ e $f_a = y$
      altrimenti:
      $b = x$
      Fine scelta

   Fine ciclo su k
3. $esito = 1$
4. Fine

Commenti

a) Le variabili a e b hanno un doppio ruolo. Esse compaiono fra i dati in quanto estremi del primo intervallo $[a_0, b_0]$; successivamente, poiché l'algoritmo ne modifica il valore, esse compaiono anche fra i risultati come estremi dell'ultimo intervallo determinato dall'algoritmo.

b) Nel paragrafo precedente abbiamo detto che le tolleranze usate nei criteri di arresto devono sempre essere maggiori di ϵ_m. A questo proposito occorre una precisazione: poiché stiamo usando due criteri di arresto possibili, si può prevedere di inibirne uno dando valore 0 alla corrispondente tolleranza. Se ad esempio η è uguale a 0, il criterio al passo 3 del ciclo non sarà mai soddisfatto (a meno di situazioni molto particolari) e l'algoritmo si fermerà eventualmente in virtù dell'altro criterio al passo 4 del ciclo. Quello che conta è che almeno una delle tre tolleranze η, σ_r e σ_a sia maggiore di 0.

c) Il valore che l'indicatore *esito* assume in uscita serve per interpretare il significato del risultato x: se *esito* vale 0, allora x rappresenta la soluzione approssimata a meno della accuratezza richiesta, altrimenti è semplicemente l'ultima iterata prodotta dall'algoritmo.

d) Con le istruzioni di assegnazione $f_a = f(a)$ e $y = f(x)$ si calcola il valore che la f assume in a o in x mediante l'algoritmo a questo preposto e si memorizza questo valore con il nome f_a o y. Lo scopo è quello di poter successivamente usare quei valori senza doverli ricalcolare, per non aumentare inutilmente il costo computazionale dell'algoritmo.

e) Al passo 5 del ciclo abbiamo sostituito il controllo sul segno del prodotto $f_a \times y$ con un controllo sul segno di $sgn(f_a) \times y$. Lo scopo è quello di evitare possibili fenomeni di underflow nei quali si potrebbe incorrere qualora f_a

e y fossero in valore assoluto molto piccoli (ad esempio, lavorando in precisione semplice, $|y| \simeq 10^{-20}$ e $|f_a| \simeq 10^{-30}$), fenomeno che potrebbe verificarsi per particolari combinazioni di funzioni e tolleranze.

3.6.2 Algoritmo di Newton

Nel paragrafo 3.3 abbiamo visto due forme del metodo di Newton, quella "classica" che usa la derivata di f e quella alle differenze, che approssima la derivata tramite un rapporto incrementale. Nell'algoritmo che segue abbiamo previsto la possibilità di optare per l'una forma o l'altra tramite un dato in ingresso, denominato *ind*: se *ind* vale 0 si usa il metodo classico, altrimenti si usa quello alle differenze. Nel primo caso l'algoritmo dovrà essere accompagnato da un altro algoritmo che, dato x, calcola $f'(x)$. Nel secondo caso invece, avendo necessità di calcolare la precisione di macchina, facciamo riferimento all'algoritmo 2.2.1 del Capitolo 2.

Algoritmo 3.6.2. Algoritmo di Newton.
Dati: ind, x, K_{max}, η
Risultati: *esito* indicatore del risultato, con i seguenti valori:
$esito = 0$ se il procedimento è terminato con successo
$esito = 1$ se sono state fatte K_{max} iterazioni senza successo
$esito = 2$ se la derivata si annulla a qualche iterazione
x ultima iterata calcolata
y valore di f in x
k numero di iterazioni effettuate

1. Se $ind \neq 0$, allora:
   calcola ϵ_m usando l'algoritmo 2.2.1
   Fine scelta
2. Per $k = 1, 2, \ldots, K_{max}$
   1. $y = f(x)$
   2. Se $|y| < \eta$, allora:
      $esito = 0$ e fermati
      Fine scelta
   3. Se $ind = 0$, allora:
      $d = f'(x)$
      altrimenti:
      $h = \sqrt{\epsilon_m}$
      se $x \neq 0$, allora:
      $h = h \times x$
      Fine scelta
      $d = \frac{f(x+h)-y}{h}$
      Fine scelta
   4. Se $d = 0$, allora:
      $esito = 2$ e fermati
      Fine scelta
   5. $x = x - \frac{y}{d}$

   Fine ciclo su k
3. $esito = 1$
4. Fine

La variabile x in questo algoritmo ha doppio ruolo: essa figura fra i dati e rappresenta il punto inziale x_0. Successivamente viene modificata dall'algoritmo e in uscita rappresenta l'ultima iterata calcolata.
Osserviamo il controllo al passo 4 del ciclo: se ad una qualche iterazione la derivata (esatta o approssimata che sia) si annulla, il procedimento deve essere interrotto; il verificarsi di questa situazione è segnalato da un particolare valore dell'indicatore *esito*. Si potrebbe sostituire questo controllo finalizzato a prevenire una divisione per zero con un controllo che prevenga anche eventuali situazioni di overflow. Questo controllo dovrebbe avere la forma

"Se $|d| < \gamma|y|$, allora : $esito = 2$ e fermati"

dove γ rappresenta un valore piccolo, vicino al reciproco della soglia di overflow.

3.6.3 Algoritmo delle secanti

L'algoritmo delle secanti usa tre variabili, x, x_0 e x_{-1}. Per ogni k esse rappresentano rispettivamente la nuova iterata x_k, quella precedente x_{k-1} e quella di due iterazioni prima x_{k-2}.

Algoritmo 3.6.3. Algoritmo delle secanti.
Dati: $x_{-1}, x_0, K_{max}, \eta$
Risultati: *esito* indicatore del risultato, con i seguenti valori:
esito $= 0$ se il procedimento è terminato con successo
esito $= 1$ se sono state fatte K_{max} iterazioni senza successo
esito $= 2$ se $\frac{f(x_k)-f(x_{k-1})}{x_k-x_{k-1}} = 0$ a qualche iterazione
x ultima iterata calcolata
y valore di f in x
k numero di iterazioni effettuate

1. $f_{-1} = f(x_{-1})$
2. $f_0 = f(x_0)$
3. Per $k = 1, 2, \ldots, K_{max}$
   1. $d = \frac{f_0-f_{-1}}{x_0-x_{-1}}$
   2. Se $d = 0$, allora:
      esito $= 2$ e fermati
      Fine scelta
   3. $x = x_0 - \frac{f_0}{d}$
   4. $y = f(x)$
   5. Se $|y| < \eta$, allora:
      esito $= 0$ e fermati
      Fine scelta
   6. $x_{-1} = x_0$ e $f_{-1} = f_0$
   7. $x_0 = x$ e $f_0 = y$

   Fine ciclo su k
4. *esito* $= 1$
5. Fine

Come abbiamo detto a proposito dell'algoritmo di Newton, anche in questo caso il controllo sull'annullarsi di d potrebbe essere sostituito da un controllo teso ad evitare situazioni di overflow. Un commento a parte meritano i passi 6 e 7 del ciclo. Con queste istruzioni si aggiornano i valori di x_{-1} e x_0 e i corrispondenti valori della funzione per prepararsi all'iterazione successiva: il valore attuale di x_{-1} va perso perché non servirà più; a x_{-1} viene attribuito il valore di x_0 e a x_0 il valore dell'iterata appena calcolata x.

3.7 Cenno alla risoluzione di sistemi non lineari (*)

Fra i tre metodi che abbiamo visto nei paragrafi precedenti, l'unico che ha un'immediata estensione al caso in cui si debba risolvere un sistema di equazioni non lineari è il metodo di Newton. Consideriamo inizialmente un sistema di 2 equazioni in 2 incognite, che scriveremo nella forma

$$\begin{cases} f(x,y) &= 0 \\ g(x,y) &= 0 \end{cases} \tag{3.27}$$

dove f e g sono funzioni delle incognite x e y a valori in $\mathbb{R}$. Ad esempio, nel sistema

$$\begin{cases} x^2+y^2 &= 4 \\ x^2y^2-3x &= -27 \end{cases}$$

dove $f(x,y) = x^2+y^2-4$ e $g(x,y) = x^2y^2-3x+27$.

La costruzione che ha portato alla definizione del metodo di Newton per risolvere una singola equazione può essere facilmente estesa alla risoluzione di (3.27). Dato un punto (x_0, y_0) che assumiamo come approssimazione iniziale della soluzione (x^*, y^*), costruiamo il piano tangente alla superficie $z = f(x,y)$ nel punto $(x_0, y_0, f(x_0, y_0))$ la cui equazione è:

$$z = f(x_0,y_0) + \frac{\partial f}{\partial x}(x_0,y_0)(x-x_0) + \frac{\partial f}{\partial y}(x_0,y_0)(y-y_0),$$

dove $\frac{\partial f}{\partial x}$ e $\frac{\partial f}{\partial y}$ sono le derivate parziali di f rispetto a x e y. Analogamente il piano tangente alla superficie $z = g(x,y)$ nel punto $(x_0, y_0, g(x_0, y_0))$ ha equazione

$$z = g(x_0,y_0) + \frac{\partial g}{\partial x}(x_0,y_0)(x-x_0) + \frac{\partial g}{\partial y}(x_0,y_0)(y-y_0).$$

Ora supponiamo che i due piani tangenti e il piano $z = 0$ si incontrino in un unico punto (x_1, y_1) e calcoliamo detto punto; questo significa risolvere il sistema lineare

$$\begin{cases} \frac{\partial f}{\partial x}(x_0,y_0)(x-x_0) + \frac{\partial f}{\partial y}(x_0,y_0)(y-y_0) &= -f(x_0,y_0) \\ \frac{\partial g}{\partial x}(x_0,y_0)(x-x_0) + \frac{\partial g}{\partial y}(x_0,y_0)(y-y_0) &= -g(x_0,y_0) \end{cases}$$

nelle incognite x e y e chiamare (x_1, y_1) la soluzione. A partire da (x_1, y_1) si ripete il procedimento: si costruiscono i piani tangenti alle due superfici nei punti $(x_1, y_1, f(x_1, y_1))$ e $(x_1, y_1, g(x_1, y_1))$ e si calcola il punto (x_2, y_2) risolvendo il sistema lineare

$$\begin{cases} \frac{\partial f}{\partial x}(x_1,y_1)(x-x_1) + \frac{\partial f}{\partial y}(x_1,y_1)(y-y_1) &= -f(x_1,y_1) \\ \frac{\partial g}{\partial x}(x_1,y_1)(x-x_1) + \frac{\partial g}{\partial y}(x_1,y_1)(y-y_1) &= -g(x_1,y_1) \end{cases}$$

E via dicendo. Alla generica iterazione k, data l'iterata corrente (x_k, y_k), definiamo la nuova iterata (x_{k+1}, y_{k+1}) come soluzione del sistema lineare

$$\begin{cases} \frac{\partial f}{\partial x}(x_k, y_k)(x - x_k) + \frac{\partial f}{\partial y}(x_k, y_k)(y - y_k) &= -f(x_k, y_k) \\ \frac{\partial g}{\partial x}(x_k, y_k)(x - x_k) + \frac{\partial g}{\partial y}(x_k, y_k)(y - y_k) &= -g(x_k, y_k). \end{cases}$$

Questo sistema può essere agilmente risolto facendo un semplice cambiamento di variabili. Ponendo $\Delta x = x - x_k$ e $\Delta y = y - y_k$ riscriviamo il sistema nella forma

$$\begin{cases} \frac{\partial f}{\partial x}(x_k, y_k)\Delta x + \frac{\partial f}{\partial y}(x_k, y_k)\Delta y &= -f(x_k, y_k) \\ \frac{\partial g}{\partial x}(x_k, y_k)\Delta x + \frac{\partial g}{\partial y}(x_k, y_k)\Delta y &= -g(x_k, y_k). \end{cases} \tag{3.28}$$

La nuova iterata (x_{k+1}, y_{k+1}) è data allora da:

$$x_{k+1} = x_k + \Delta x \ , \ y_{k+1} = y_k + \Delta y. \tag{3.29}$$

Le formule (3.28)-(3.29) definiscono il metodo di Newton per la risoluzione del sistema (3.27).

Passiamo ora al caso generale, in cui si abbia da risolvere un sistema non lineare di n equazioni in n incognite della forma

$$\begin{cases} f_1(x_1, x_2, \ldots, x_n) &= 0 \\ f_2(x_1, x_2, \ldots, x_n) &= 0 \\ \vdots & \vdots \\ f_n(x_1, x_2, \ldots, x_n) &= 0 \end{cases} \tag{3.30}$$

dove $x_1, x_2, \ldots, x_n$ sono le incognite e $f_1, f_2, \ldots, f_n$ sono funzioni a valori reali. Il sistema può essere espresso in forma compatta introducendo la notazione vettoriale. Chiamiamo x il vettore di componenti $x_1, x_2, \ldots, x_n$ e f la funzione da $\mathbb{R}^n$ in $\mathbb{R}^n$ definita da $f(x) = (f_1(x), f_2(x), \ldots, f_n(x))^T$. Allora il sistema può essere scritto semplicemente come

$$f(x) = 0.$$

La soluzione viene ancora indicata come x^*, che in questo caso generale è un vettore di $\mathbb{R}^n$ di componenti $x_1^*, x_2^*, \ldots, x_n^*$. A partire da una stima iniziale $x^{(0)} = (x_1^{(0)}, x_2^{(0)}, \ldots, x_n^{(0)})^T$ della soluzione, si costruisce una successione $\{x^{(k)}\}$ nel modo seguente [6]. Data la k-esima iterata $x^{(k)}$, si calcolano le derivate parziali delle funzioni $f_1, \ldots, f_n$ rispetto a $x_1, \ldots, x_n$ in $x^{(k)}$;

[6] Poiché in questo contesto utilizziamo gli indici in basso per distinguere le componenti di un singolo vettore, per evitare confusione indichiamo l'iterazione con l'indice in alto (k).

per semplicità indichiamo con f_{ij} la derivata parziale $\frac{\partial f_i}{\partial x_j}$ della i-esima funzione f_i rispetto alla j-esima incognita x_j. Si calcolano anche i valori $f_1(x^{(k)}), \ldots, f_n(x^{(k)})$ e si forma il sistema lineare di n equazioni nelle n incognite $\Delta_1, \Delta_2, \ldots, \Delta_n$:

$$\begin{cases} f_{11}(x^{(k)})\Delta_1 + f_{12}(x^{(k)})\Delta_2 + + \ldots + f_{1n}(x^{(k)})\Delta_n = -f_1(x^{(k)}) \\ f_{21}(x^{(k)})\Delta_1 + f_{22}(x^{(k)})\Delta_2 + + \ldots + f_{2n}(x^{(k)})\Delta_n = -f_2(x^{(k)}) \\ \vdots \\ f_{i1}(x^{(k)})\Delta_1 + f_{i2}(x^{(k)})\Delta_2 + + \ldots + f_{in}(x^{(k)})\Delta_n = -f_i(x^{(k)}) \\ \vdots \\ f_{n1}(x^{(k)})\Delta_1 + f_{n2}(x^{(k)})\Delta_2 + + \ldots + f_{nn}(x^{(k)})\Delta_n = -f_n(x^{(k)}). \end{cases} \tag{3.31}$$

Indicando con Δ il vettore di componenti $\Delta_1, \Delta_2, \ldots, \Delta_n$ e con $J(x^{(k)})$ la matrice dei coefficienti, il sistema può essere scritto in modo compatto nella forma

$$J(x^{(k)})\Delta = -f(x^{(k)}).$$

Se il sistema ha un'unica soluzione $\Delta^{(k)}$, la calcoliamo e otteniamo la nuova iterata con la formula

$$x^{(k+1)} = x^{(k)} + \Delta^{(k)}.$$

Le proprietà di convergenza del metodo rimangono quelle che abbiamo visto nel paragrafo 3.3: se la funzione f è sufficientemente regolare e $x^{(0)}$ sufficientemente vicino a x^*, il metodo converge quadraticamente. Per ovviare al fatto che la convergenza è soltanto locale e rendere quindi meno problematica la scelta del punto iniziale, non si può ricorrere al metodo di bisezione come nel caso unidimensionale, perché questo metodo non è generalizzabile al caso n-dimensionale. Esistono però altre strategie, che vanno sotto i nomi di line search e trust region, che possono essere utilizzate. Per dettagli su queste tecniche di "globalizzazione" e altri aspetti legati alla realizzazione pratica del metodo di Newton in $\mathbb{R}^n$ rimandiamo alle monografie [14] e [24]. Qui ci limitiamo ad osservare che ad ogni iterazione si deve calcolare la matrice delle derivate parziali (detta matrice jacobiana) $J(x^{(k)})$ e risolvere un sistema lineare. Per quanto riguarda il calcolo di $J(x^{(k)})$, si può ricorrere ad approssimazioni alle differenze costruite con lo stesso spirito della formula usata nel caso unidimensionale; per la risoluzione del sistema si può usare uno dei metodi che vedremo nel prossimo capitolo.

4

SISTEMI LINEARI

4.1 Introduzione

Siamo interessati a metodi numerici per risolvere sistemi lineari di n equazioni in n incognite:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \ldots + a_{2n}x_n &= b_2 \\ \vdots & \\ a_{i1}x_1 + a_{i2}x_2 + a_{i3}x_3 + \ldots + a_{in}x_n &= b_i \\ \vdots & \\ a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \ldots + a_{nn}x_n &= b_n \end{cases}$$

o, in forma compatta, $Ax = b$.

Supponiamo $\det(A) \neq 0$, in modo da avere la certezza che il sistema ha una e una sola soluzione per la quale la matematica ci mette a disposizione due formule alternative. La prima, nota come regola di Cramer, è

$$x_i = \frac{\det(A_i)}{\det(A)}, \quad \text{per} \quad i = 1, \ldots n, \tag{4.1}$$

dove A_i è la matrice ottenuta da A sostituendo la colonna i-esima con il vettore dei termini noti b. La seconda, più compatta, è

$$x = A^{-1}b. \tag{4.2}$$

Entrambe le formule non sono utilizzabili nella pratica perché assolutamente inefficienti. La (4.1) richiede il calcolo di $n + 1$ determinanti di matrici di ordine n, per ognuno dei quali sono necessarie circa $n!$ operazioni aritmetiche [1]. D'altra parte il calcolo della matrice inversa con le formule note

[1] Ricordiamo che $n! = n(n-1)(n-2)\ldots 1$ cresce molto rapidamente al crescere di n. Per esempio, $10! \simeq 3.6 \times 10^6$ e $30! \simeq 2.6 \times 10^{32}$.

dell'algebra lineare richiede il calcolo di $\det(A)$ e di altri n^2 determinanti di matrici $(n-1)\times(n-1)$. I metodi numerici per la risoluzione di $Ax = b$ non utilizzano pertanto né (4.1) né (4.2) e si basano su principi completamente diversi. Essi si dividono fondamentalmente in due classi: metodi diretti e metodi iterativi.

I metodi diretti sono caratterizzati dal fatto che in un numero finito predeterminato (in funzione di n) di operazioni calcolano la soluzione esatta, a meno di errori dovuti all'aritmetica floating point. Il principio su cui si basano questi metodi è quello di trasformare il sistema in un altro ad esso equivalente di "facile" risoluzione, nel senso che sono disponibili semplici algoritmi per risolverlo.

I metodi iterativi sono al contrario procedimenti che non modificano il sistema. A partire da una stima iniziale della soluzione, essi costruiscono una successione di vettori che, sotto opportune ipotesi, converge alla soluzione cercata. Ovviamente questi metodi pongono tutte le problematiche esaminate nel capitolo precedente: sotto quali ipotesi è garantita la convergenza? Si tratta di metodi localmente o globalmente convergenti? Con quale velocità convergono? E infine, quali criteri di arresto si possono adottare?

In molte applicazioni occorre risolvere sistemi lineari di grandi dimensioni, nei quali la maggior parte dei coefficienti è uguale a zero (si dice in questo caso che la matrice A è sparsa). In questa situazione è ragionevole pensare di ridurre le richieste di memoria e di tempo, memorizzando e utilizzando nei calcoli soltanto i coefficienti diversi da zero. I metodi diretti non possono trarre vantaggio dalla sparsità della matrice perché nella trasformazione del sistema i coefficienti uguali a zero vengono in generale sostituiti da numeri diversi da zero. I metodi iterativi invece non presentano questo problema e quindi vengono usati per risolvere sistemi grandi e sparsi.

In questo capitolo presenteremo i principali metodi diretti e alcuni fra i più elementari metodi iterativi. Rimandiamo per un panorama più completo e dettagliato, soprattutto per quanto riguarda la seconda classe di metodi, ai molti libri di testo e monografie pubblicati negli ultimi venti anni, fra i quali citiamo [6], [17], [20], [24] e [33].

4.2 Condizionamento dei sistemi lineari

Ricordiamo che con l'espressione "condizionamento di un problema" ci riferiamo alla sua sensibilità alle perturbazioni dei dati, ovvero all'entità del cambiamento del risultato in funzione del cambiamento dei dati. Per quanto riguarda i sistemi lineari, cominciamo con il considerare il caso più semplice in cui si introducano errori soltanto nel vettore b. Sia $b + \Delta b$ il nuovo vettore dei termini noti e indichiamo con $x + \Delta x$ la soluzione del sistema perturbato. Fissata una qualsiasi norma vettoriale, vogliamo trovare una limitazione superiore per la perturbazione relativa sulla soluzione

$\frac{\|\Delta x\|}{\|x\|}$ in funzione della perturbazione relativa sui dati $\frac{\|\Delta b\|}{\|b\|}$. Da $Ax = b$ e $A(x + \Delta x) = b + \Delta b$ deduciamo $A\Delta x = \Delta b$, ovvero

$$\Delta x = A^{-1}\Delta b.$$

Ora passiamo alle norme, utilizzando per le matrici la norma indotta da quella vettoriale prescelta. Sfruttando le proprietà delle norme dall'ultima relazione si ottiene

$$\|\Delta x\| \leq \|A^{-1}\|\|\Delta b\| \tag{4.3}$$

e da $Ax = b$

$$\|x\| \geq \frac{\|b\|}{\|A\|}. \tag{4.4}$$

Da (4.3) e (4.4) segue

$$\frac{\|\Delta x\|}{\|x\|} \leq \frac{\|A^{-1}\|\|\Delta b\|}{\|x\|} \leq k(A)\frac{\|\Delta b\|}{\|b\|}, \tag{4.5}$$

dove abbiamo posto

$$k(A) = \|A\|\|A^{-1}\|.$$

La (4.5) indica che nel caso peggiore (quando vale il segno di "=") la perturbazione relativa sulla soluzione è pari a $k(A)$ volte quella su b. Poiché

$$k(A) = \|A\|\|A^{-1}\| \geq \|AA^{-1}\| = \|I\| = 1,$$

possiamo dire che $k(A)$ funge da fattore di amplificazione degli errori presenti sui dati.

Abbiamo trovato questo risultato ipotizzando la situazione poco realistica in cui le perturbazioni interessino soltanto i termini noti. In realtà un risultato analogo vale in generale, quando anche gli elementi di A sono perturbati. Infatti, consideriamo il sistema perturbato della forma

$$\widetilde{A}x = \tilde{b},$$

dove

$$\widetilde{A} = A + \epsilon\Delta A, \quad \text{e} \quad \tilde{b} = b + \epsilon\Delta b \tag{4.6}$$

e $\det(\widetilde{A}) \neq 0$. In queste formule ΔA è una matrice $n \times n$, Δb è un vettore di $\mathbb{R}^n$ e ϵ è uno scalare positivo che supponiamo sufficientemente piccolo da poter trascurare in quanto segue termini di secondo grado in ϵ. La forma (4.6) è assolutamente generale, qualunque sia la fonte degli errori nei dati. Supponiamo ad esempio che gli errori derivino dalla rappresentazione floating point dei dati. In questo caso ogni coefficiente a_{ij} viene sostituito da $\tilde{a}_{ij} = fl(a_{ij}) = a_{ij}(1 + \epsilon_{ij})$ con $|\epsilon_{ij}| \leq \epsilon_m$ e ogni termine noto b_i da $\tilde{b}_i = fl(b_i) = b_i(1 + \eta_i)$ con $|\eta_i| \leq \epsilon_m$. Allora, $\widetilde{A}$ e $\tilde{b}$ hanno la forma

$$\widetilde{A} = A + \epsilon_m\Delta A \ , \ \tilde{b} = b + \epsilon_m\Delta b \tag{4.7}$$

dove gli elementi di ΔA e Δb sono dati da $\frac{\epsilon_{ij}}{\epsilon_m}a_{ij}$ e $\frac{\eta_i}{\epsilon_m}b_i$ rispettivamente.

Indichiamo la soluzione del sistema perturbato con $x(\epsilon)$; per $\epsilon = 0$ si ottiene il sistema originale, la cui soluzione è quindi $x(0)$. Nel teorema che segue si dà una maggiorazione dell'errore relativo sulla soluzione $\frac{\|x(\epsilon)-x(0)\|}{\|x(0)\|}$ in funzione degli errori relativi sui dati $\epsilon\frac{\|\Delta A\|}{\|A\|}$ e $\epsilon\frac{\|\Delta b\|}{\|b\|}$; in questa maggiorazione interviene $k(A)$.

Teorema 4.2.1. Se $\det(A) \neq 0$ e $\det(\tilde{A}) \neq 0$, allora

$$\frac{\|x(\epsilon) - x(0)\|}{\|x(0)\|} \leq k(A)\left(\epsilon\frac{\|\Delta A\|}{\|A\|} + \epsilon\frac{\|\Delta b\|}{\|b\|}\right). \tag{4.8}$$

Dimostrazione. Consideriamo il sistema perturbato

$$(A + \epsilon\Delta A)x(\epsilon) = b + \epsilon\Delta b$$

e deriviamo entrambi i membri rispetto a ϵ. Si ottiene [2]

$$\Delta A x(\epsilon) + (A + \epsilon\Delta A)\dot{x}(\epsilon) = \Delta b,$$

che, per $\epsilon = 0$, dà $\Delta A x(0) + A\dot{x}(0) = \Delta b$, ovvero:

$$\dot{x}(0) = A^{-1}[\Delta b - \Delta A x(0)]. \tag{4.9}$$

Ora consideriamo lo sviluppo al primo ordine di $x(\epsilon)$ in funzione di ϵ:

$$x(\epsilon) = x(0) + \epsilon\dot{x}(0) + \mathcal{O}(\epsilon^2),$$

che, per la (4.9), diventa

$$x(\epsilon) = x(0) + \epsilon A^{-1}[\Delta b - \Delta A x(0)] + \mathcal{O}(\epsilon^2).$$

Passando alle norme si ottiene

$$\begin{aligned} \|x(\epsilon) - x(0)\| &\leq \epsilon\|A^{-1}\|\|\Delta b - \Delta A x(0)\| + \mathcal{O}(\epsilon^2) \\ &\leq \epsilon\|A^{-1}\|(\|\Delta b\| + \|\Delta A\|\|x(0)\|) + \mathcal{O}(\epsilon^2) \end{aligned}$$

ovvero

$$\frac{\|x(\epsilon) - x(0)\|}{\|x(0)\|} \leq \epsilon\|A^{-1}\|\left(\frac{\|\Delta b\|}{\|x(0)\|} + \|\Delta A\|\right) + \mathcal{O}(\epsilon^2). \tag{4.10}$$

[2] Qui abbiamo a che fare con funzioni che vanno da $\mathbb{R}$ a $\mathbb{R}^n$ e $\mathbb{R}^{n\times n}$. In questo caso l'operazione di derivazione si applica elemento per elemento e continuano a valere le usuali regole di derivazione della somma e del prodotto di funzioni. Anche lo sviluppo di Taylor che useremo più avanti è da intendersi elemento per elemento (cfr. per esempio [3]).

Ora dobbiamo intervenire su (4.10) per far emergere a destra del "$\leq$" gli errori relativi sui dati. A questo scopo moltiplichiamo e dividiamo il membro destro della disuguaglianza per $\|A\|$:

$$\frac{\|x(\epsilon) - x(0)\|}{\|x(0)\|} \leq \epsilon k(A) \left(\frac{\|\Delta b\|}{\|A\|\|x(0)\|} + \frac{\|\Delta A\|}{\|A\|} \right) + \mathcal{O}(\epsilon^2)$$

da cui si arriva alla disuguaglianza desiderata (4.8) ricordando che $Ax(0) = b$ implica $\|A\|\|x(0)\| \geq \|b\|$.

□

Da questo teorema segue che il (buon o cattivo) condizionamento di un sistema lineare è strettamente legato al numero $k(A)$, che viene quindi detto numero di condizionamento del sistema [3]. Quando $k(A)$ è grande il sistema $Ax = b$ risulta mal condizionato; viceversa, il sistema è ben condizionato quando $k(A)$ è piccolo (ricordiamo che $k(A) \geq 1$).

Il significato dei termini "grande" e "piccolo" usati nella frase precedente non sono da intendersi in senso assoluto, ma relativo quanto meno all'ambiente di calcolo in cui ci poniamo. Per esempio la relazione (4.8), insieme con la (4.7), mostra che può essere impossibile risolvere un sistema lineare con numero di condizionamento di ordine $\geq 10^7$ lavorando in precisione semplice ($\epsilon_m \simeq 10^{-7}$), mentre lo stesso sistema può essere risolto in precisione doppia purché ci si accontenti eventualmente di non più di 9 cifre di accuratezza ($\epsilon_m \simeq 10^{-16}$).

È possibile dimostrare [20] che $k(A)$ è legato alla distanza di A dall'insieme delle matrici $n \times n$ singolari. Più precisamente

$$\frac{1}{k(A)} = \min \left\{ \frac{\|B - A\|}{\|A\|}, \;\; B \in \mathbb{R}^{n \times n}, \;\; \det(B) = 0 \right\}. \tag{4.11}$$

Questa caratterizzazione potrebbe far pensare ad una sorta di proporzionalità inversa fra $k(A)$ e $\det(A)$: $k(A)$ è grande se e soltanto se $\det(A)$ è piccolo. È facile vedere che questo legame di fatto non sussiste: basta per questo considerare la matrice diagonale $D = diag(10^{-10}, 10^{-10}, \ldots, 10^{-10}) \in \mathbb{R}^{n \times n}$, che ha determinante 10^{-10n} ma è ottimamente condizionata perché $k(D) = 1$ in qualunque norma naturale, essendo $\|D\| = 10^{-10}$ e $\|D^{-1}\| = 10^{10}$. Osserviamo a questo proposito che, per una data matrice A, il valore di $k(A)$ dipende in generale dalla norma scelta, tanto che talvolta si usano simboli diversi per evidenziare quale norma si è usata: $k_1(A)$ quando si usa la norma-1, $k_2(A)$ quando si usa la norma euclidea e $k_\infty(A)$ quando si usa la norma-∞. Nondimeno, a causa dell'equivalenza fra

[3] $k(A)$ viene anche chiamato numero di condizionamento di A rispetto al problema dell'inversione (ricordiamo che, tramite la relazione (4.2), risolvere un sistema lineare equivale ad invertire la matrice dei coefficienti).

le norme, se una matrice è mal condizionata o ben condizionata secondo una data norma lo è rispetto a tutte le norme naturali.

L'informazione contenuta nel numero di condizionamento è molto importante ai fini della valutazione dei risultati che un qualunque risolutore di sistemi lineari dà in aritmetica floating point. Il calcolo di $k(A)$ non è banale, ma fortunatamente esistono dei metodi numerici che forniscono delle buone approssimazioni (cfr. [22] per una panoramica approfondita ed esaustiva di questi metodi). I valori riportati in questo capitolo sono stati ottenuti in MATLAB usando la funzione cond per $k_2(\cdot)$ e la funzione condest per $k_1(\cdot)$.

Esempio 4.2.1. Le matrici di Vandermonde intervengono nei problemi di interpolazione, di cui parleremo nel Capitolo 5. Dati n numeri $\alpha_1, \alpha_2, \ldots, \alpha_n$, gli elementi della matrice di Vandermonde sono definiti da $a_{ij} = \alpha_i^{j-1}$; se per esempio $n = 4$ e $\alpha_i = i$ per $i = 1, 2, 3, 4$, la matrice è:

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 \\ 1 & 3 & 9 & 27 \\ 1 & 4 & 16 & 64 \end{pmatrix}.$$

Abbiamo calcolato $k_2(A)$ e $k_1(A)$ per le matrici di Vandermonde costruite con $\alpha_i = i$ per $i = 1, \ldots, n$ per diversi valori di n. Come mostra la tabella 4.1, $k_2(A)$ e $k_1(A)$ sono diversi fra loro per ogni n, ma entrambi testimoniano che il condizionamento peggiora al crescere di n. Nella tabella riportiamo anche i valori del determinante di A, che per matrici di Vandermonde ha un'espressione nota:

n	$k_2(A)$	$k_1(A)$	$\det(A)$
2	6.9e+00	9.0e+00	1.0e+00
3	7.1e+01	1.1e+02	2.0e+00
4	1.2e+03	2.0e+03	1.2e+01
5	2.6e+04	4.4e+04	2.9e+02
6	7.3e+05	1.3e+06	3.5e+04
7	2.4e+07	4.1e+07	2.5e+07
8	9.5e+08	1.7e+09	1.3e+11
9	4.2e+10	7.1e+10	5.1e+15
10	2.1e+12	3.6e+12	1.8e+21

Tabella 4.1 Esempio 4.2.1: condizionamento delle matrici di Vandermonde

$$det(A) = \prod_{i=1}^{n-1} \prod_{j=i+1}^{n} (\alpha_j - \alpha_i). \tag{4.12}$$

Questi valori mostrano che al peggiorare del condizionamento di A non corrisponde un avvicinarsi a zero del determinante.

A parità di dimensione n, il condizionamento della matrice di Vandermonde dipende dai valori di $\alpha_1, \ldots, \alpha_n$. Ad esempio, se scegliamo $\alpha_i = cos\left(\frac{2i-1}{2n}\pi\right)$ il numero di condizionamento di A cresce con n, ma molto più lentamente che nel caso della tabella 4.1: per n=10 si ha $k_2(A) \simeq 1.47 \times 10^3$ e per arrivare a un valore maggiore di 10^{12} si deve prendere n circa uguale a 40. Incontreremo di nuovo questi speciali valori α_i nel Capitolo 5. ■

Esempio 4.2.2. Le matrici di Hilbert sono definite da $a_{ij} = \frac{1}{i+j-1}$ per $i, j = 1, \ldots, n$; per esempio la matrice di Hilbert di dimensione 3 è:

$$A = \begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \end{pmatrix}.$$

Esaminando la tabella 4.2 possiamo fare le stesse considerazioni dell'esempio 4.2.1, con la sola differenza che in questo caso il determinante si avvicina

n	$k_2(A)$	$k_1(A)$	$\det(A)$
2	1.9e+01	2.7e+01	8.3e−02
3	5.2e+02	7.5e+02	4.6e−04
4	1.6e+04	2.8e+04	1.7e−07
5	4.8e+05	9.4e+05	3.7e−12
6	1.5e+07	2.9e+07	5.4e−18
7	4.8e+08	9.9e+08	4.8e−25
8	1.5e+10	3.4e+10	2.7e−33
9	4.9e+11	1.1e+12	9.7e−43
10	1.6e+13	3.5e+13	2.2e−53

Tabella 4.2 Esempio 4.2.2: condizionamento delle matrici di Hilbert

sempre più a zero al crescere di n. Precisiamo che, non disponendo di una formula esplicita per il determinante delle matrici di Hilbert, abbiamo calcolato i determinanti tramite la funzione det di MATLAB. I risultati di questa funzione possono non essere completamente accurati quando la matrice è molto mal condizionata; tuttavia, possiamo fidarci di essi per vedere se il determinante è vicino a zero o no. ■

4.2.1 Analisi a posteriori

Il numero di condizionamento riveste un ruolo molto importante anche nella cosiddetta analisi a posteriori della bontà di una soluzione approssimata di

un sistema lineare. Il problema è il seguente: sostituendo un qualunque vettore $\tilde{x}$ in un sistema lineare, si ottiene il vettore residuo

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

che coincide con il vettore nullo se e soltanto se $\tilde{x}$ è soluzione del sistema. Possiamo chiederci allora se, e entro quali limiti, la grandezza del vettore residuo può fornire un'idea della grandezza dell'errore $e = \tilde{x} - x$, dove con x indichiamo come al solito la soluzione esatta. A questo scopo osserviamo che

$$r = A\tilde{x} - b = A\tilde{x} - Ax = A(\tilde{x} - x) = Ae,$$

da cui deduciamo $e = A^{-1}r$. Allora, usando una qualsiasi norma vettoriale, si ottiene

$$\frac{\|r\|}{\|A\|} \leq \|e\| \leq \|A^{-1}\|\|r\|,$$

ovvero

$$\frac{\|r\|}{\|A\|\|x\|} \leq \frac{\|e\|}{\|x\|} \leq \frac{\|A^{-1}\|\|r\|}{\|x\|}.$$

A questo punto possiamo eliminare $\|x\|$ dal primo e dall'ultimo termine di questa catena di disuguaglianze sfruttando il fatto che $\frac{\|b\|}{\|A\|} \leq \|x\| \leq \|A^{-1}\|\|b\|$. Si arriva così alla seguente relazione:

$$\frac{\|r\|}{\|A\|\|A^{-1}\|\|b\|} \leq \frac{\|e\|}{\|x\|} \leq \frac{\|A^{-1}\|\|A\|\|r\|}{\|b\|},$$

che possiamo riscrivere come

$$\frac{1}{k(A)}\frac{\|r\|}{\|b\|} \leq \frac{\|e\|}{\|x\|} \leq k(A)\frac{\|r\|}{\|b\|}. \tag{4.13}$$

La relazione che abbiamo ricavato mostra che se A è ben condizionata ($k(A) \simeq 1$) l'errore relativo $\frac{\|e\|}{\|x\|}$ e il residuo relativo $\frac{\|r\|}{\|b\|}$ sono dello stesso ordine di grandezza; d'altra parte quanto più $k(A)$ è grande tanto meno il residuo relativo fornisce una misura attendibile dell'errore relativo.

4.3 Risoluzione di sistemi triangolari

Prima di introdurre i principali metodi diretti per la risoluzione di sistemi lineari, consideriamo il caso dei sistemi triangolari (ovvero dei sistemi in cui A è triangolare, superiore o inferiore), per i quali disponiamo di semplici algoritmi basati sul metodo di sostituzione. Ovviamente supporremo che A sia invertibile, il che significa che tutti gli elementi diagonali sono diversi da zero.

Consideriamo dapprima il caso in cui A è triangolare superiore:

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ 0 & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & a_{nn} \end{pmatrix}.$$

In questo caso l'algoritmo più naturale è l'algoritmo di sostituzione all'indietro: si ricava il valore di x_n dall'ultima equazione; usando questo valore si ricava x_{n-1} dalla penultima e così via fino a ricavare x_1 dalla prima equazione.

Algoritmo 4.3.1. Algoritmo di sostituzione all'indietro.

Dati: n, A, b
Risultato: x
1. $x_n = \frac{b_n}{a_{nn}}$
2. Per $i = n-1, n-2, \dots, 1$
    1. $s = \sum_{j=i+1}^{n} a_{ij} x_j$
    2. $x_i = \frac{b_i - s}{a_{ii}}$

   Fine ciclo su i
3. Fine

Se la matrice dei coefficienti è triangolare inferiore

$$A = \begin{pmatrix} a_{11} & 0 & \dots & 0 \\ a_{21} & a_{22} & \dots & 0 \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix}$$

si può ancora procedere per sostituzione, stavolta in avanti: si ricava il valore di x_1 dalla prima equazione; usando questo valore si ricava x_2 dalla seconda e così via fino a ricavare x_n dall'ultima equazione.

Algoritmo 4.3.2. Algoritmo di sostituzione in avanti (versione 1).

Dati: n, A, b
Risultato: x
1. $x_1 = \frac{b_1}{a_{11}}$
2. Per $i = 2, \dots, n$
    1. $s = \sum_{j=1}^{i-1} a_{ij} x_j$
    2. $x_i = \frac{b_i - s}{a_{ii}}$

   Fine ciclo su i
3. Fine

Il costo computazionale di questi algoritmi, inteso come numero di operazioni aritmetiche effettuate per arrivare al risultato, può essere facilmente

calcolato in funzione di n. Consideriamo ad esempio il metodo di sostituzione in avanti (il risultato è lo stesso per il metodo di sostituzione all'indietro) e supponiamo di usare l'algoritmo 1.3.1 del Capitolo 1 per calcolare la sommatoria al passo 1 del ciclo. Allora il calcolo di x_1 richiede 1 operazione aritmetica mentre il calcolo di x_i con $i = 2, \ldots, n$ ne richiede $2i$, per l'esattezza $i-1$ moltiplicazioni, $i-1$ addizioni, 1 sottrazione e 1 divisione. Il totale è quindi $1 + \sum_{i=2}^{n} 2i = n(n+1) - 1 = n^2 + n - 1$. Per effettuare questo calcolo abbiamo tenuto conto del fatto che $1+2+\ldots+n = \frac{n(n+1)}{2}$. Il costo dell'algoritmo può essere ridotto se lo scriviamo in modo leggermente diverso, sostanzialmente esplicitando il calcolo della sommatoria:

Algoritmo 4.3.3. Algoritmo di sostituzione in avanti (versione 2).

Dati: n, A, b
Risultato: x
1. $x_1 = \frac{b_1}{a_{11}}$
2. Per $i = 2, \ldots, n$
    1. $x_i = b_i$
    2. Per $j = 1, \ldots, i-1$
        1. $x_i = x_i - a_{ij} x_j$
    Fine ciclo su j
    3. $x_i = \frac{x_i}{a_{ii}}$
  Fine ciclo su i
3. Fine

In questo algoritmo il calcolo di x_i richiede $2i-1$ operazioni ($i-1$ sottrazioni, $i-1$ moltiplicazioni e 1 divisione) e il totale diventa $1 + \sum_{i=2}^{n}(2i-1) = n^2$, con un risparmio di $n-1$ operazioni rispetto alla versione precedente. Notiamo d'altra parte che in ogni caso il costo risulta asintoticamente (per n grande) lo stesso perché il termine di primo grado $n-1$ è trascurabile rispetto al termine di secondo grado n^2. Si esprime questo concetto dicendo che il costo degli algoritmi di sostituzione per sistemi triangolari è $\mathcal{O}(n^2)$.

Uno studio dettagliato della stabilità degli algoritmi di sostituzione in avanti e all'indietro può essere trovata in [22]. La pratica dice che questi algoritmi sono estremamente stabili, anche più di quanto la teoria riesca a prevedere. Citando da J.H. Wilkinson [38]: In practice one almost invariably finds that if L is ill-conditioned, so that $\|L\|\|L^{-1}\| >> 1$, then the computed solution of $Lx = b$ (or the computed inverse) is far more accurate than [standard norm bounds] would suggest.

4.4 Metodo di eliminazione di Gauss

Il metodo di eliminazione di Gauss [4] ha lo scopo di trasformare il sistema

$$Ax = b$$

in un altro sistema

$$Ux = d$$

ad esso equivalente, con matrice dei coefficienti U triangolare superiore. Il metodo procede per eliminazioni successive delle incognite. Al primo passo si ricava x_1 dalla prima equazione e la si elimina dalle altre equazioni per sostituzione; al secondo passo si ricava x_2 dalla seconda equazione e la si sostituisce nelle successive; e così via fino all'ultimo passo che consiste nel ricavare x_{n-1} dalla penultima equazione e sostituirla nell'ultima. A questo punto il sistema è diventato triangolare superiore e può essere risolto con l'algoritmo di sostituzione all'indietro. Vediamo in dettaglio come si svolge il procedimento.

Primo passo
Supponiamo che sia $a_{11} \neq 0$. Allora si può ricavare

$$x_1 = \frac{b_1 - (a_{12}x_2 + a_{13}x_3 + \ldots + a_{1n}x_n)}{a_{11}} = \frac{b_1 - \sum_{j=2}^{n} a_{1j}x_j}{a_{11}};$$

sostituendo nella seconda equazione e raccogliendo i termini relativi a ciascuna incognita si ottiene:

$$(a_{22} - \frac{a_{21}}{a_{11}}a_{12})x_2 + \ldots + (a_{2n} - \frac{a_{21}}{a_{11}}a_{1n})x_n = b_2 - \frac{a_{21}}{a_{11}}b_1.$$

Analogamente, sostituendo x_1 in una qualunque delle equazioni successive, diciamo nella i-esima, si ottiene:

$$(a_{i2} - \frac{a_{i1}}{a_{11}}a_{12})x_2 + \ldots + (a_{in} - \frac{a_{i1}}{a_{11}}a_{1n})x_n = b_i - \frac{a_{i1}}{a_{11}}b_1.$$

Una volta sostituita x_1 in tutte le equazioni il sistema diventa:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1n}x_n &= b_1 \\ a_{22}^{(1)}x_2 + a_{23}^{(1)}x_3 + \ldots + a_{2n}^{(1)}x_n &= b_2^{(1)} \\ \vdots & \vdots \\ a_{i2}^{(1)}x_2 + a_{i3}^{(1)}x_3 + \ldots + a_{in}^{(1)}x_n &= b_i^{(1)} \\ \vdots & \vdots \\ a_{n2}^{(1)}x_2 + a_{n3}^{(1)}x_3 + \ldots + a_{nn}^{(1)}x_n &= b_n^{(1)} \end{cases}$$

[4] Carl Friedrich Gauss (1777-1855) pubblicò il metodo che porta il suo nome nel trattato Theoria Motus del 1809.

dove

$$a_{ij}^{(1)} = a_{ij} - \frac{a_{i1}}{a_{11}} a_{1j}, \quad \text{per } \; i,j = 2,\ldots,n$$

e

$$b_i^{(1)} = b_i - \frac{a_{i1}}{a_{11}} b_1, \quad \text{per } \; i = 2,\ldots,n.$$

La matrice dei coefficienti del sistema è diventata

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & \ldots & a_{1n} \\ 0 & a_{22}^{(1)} & a_{23}^{(1)} & a_{24}^{(1)} & \ldots & a_{2n}^{(1)} \\ 0 & a_{32}^{(1)} & a_{33}^{(1)} & a_{34}^{(1)} & \ldots & a_{3n}^{(1)} \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ 0 & a_{n2}^{(1)} & a_{n3}^{(1)} & a_{n4}^{(1)} & \ldots & a_{nn}^{(1)} \end{pmatrix}.$$

Secondo passo
Supponiamo che sia $a_{22}^{(1)} \neq 0$. Allora si può ricavare dalla seconda equazione

$$x_2 = \frac{b_2^{(1)} - \sum_{j=3}^{n} a_{2j}^{(1)} x_j}{a_{22}^{(1)}};$$

Sostituendo nella terza equazione si ottiene:

$$(a_{33}^{(1)} - \frac{a_{32}^{(1)}}{a_{22}^{(1)}} a_{23}^{(1)})x_3 + \ldots + (a_{3n}^{(1)} - \frac{a_{32}^{(1)}}{a_{22}^{(1)}} a_{2n}^{(1)})x_n = b_3^{(1)} - \frac{a_{32}^{(1)}}{a_{22}^{(1)}} b_2^{(1)}$$

e in generale, sostituendo nella i-esima equazione:

$$(a_{i3}^{(1)} - \frac{a_{i2}^{(1)}}{a_{22}^{(1)}} a_{i3}^{(1)})x_3 + \ldots + (a_{in}^{(1)} - \frac{a_{i2}^{(1)}}{a_{22}^{(1)}} a_{2n}^{(1)})x_n = b_i^{(1)} - \frac{a_{i2}^{(1)}}{a_{22}^{(1)}} b_2^{(1)}.$$

Dopo che x_2 è stata sostituita in tutte le equazioni il sistema diventa:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1n}x_n &= b_1 \\ a_{22}^{(1)}x_2 + a_{23}^{(1)}x_3 + \ldots + a_{2n}^{(1)}x_n &= b_2^{(1)} \\ a_{33}^{(2)}x_3 + \ldots + a_{3n}^{(2)}x_n &= b_3^{(2)} \\ \vdots & \vdots \\ a_{i3}^{(2)}x_3 + \ldots + a_{in}^{(2)}x_n &= b_i^{(2)} \\ \vdots & \vdots \\ a_{n3}^{(2)}x_3 + \ldots + a_{nn}^{(2)}x_n &= b_n^{(2)} \end{cases}$$

dove

$$a_{ij}^{(2)} = a_{ij}^{(1)} - \frac{a_{i2}^{(1)}}{a_{22}^{(1)}} a_{2j}^{(1)} \quad \text{per } \; i,j = 3,\ldots,n$$

e

$$b_i^{(2)} = b_i^{(1)} - \frac{a_{i2}^{(1)}}{a_{22}^{(1)}} b_2^{(1)}, \quad \text{per } \; i = 3,\ldots,n.$$

Al termine di questo passo, la matrice dei coefficienti del sistema trasformato è

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & \ldots & a_{1n} \\ 0 & a_{22}^{(1)} & a_{23}^{(1)} & a_{24}^{(1)} & \ldots & a_{2n}^{(1)} \\ 0 & 0 & a_{33}^{(2)} & a_{34}^{(2)} & \ldots & a_{3n}^{(2)} \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & a_{n3}^{(2)} & a_{n4}^{(2)} & \ldots & a_{nn}^{(2)} \end{pmatrix}.$$

Proseguendo in questo modo, al k-esimo passo eliminiamo l'incognita x_k da tutte le equazioni successive alla k-esima, supponendo $a_{kk}^{(k-1)} \neq 0$; i coefficienti e i termini noti subiscono un cambiamento in base a formule analoghe a quelle viste sopra per $k = 1$ e $k = 2$. Per trasformare il sistema dato nel sistema triangolare superiore $Ux = d$ occorrono $n - 1$ passi.

D'ora in avanti per semplificare le notazioni non useremo più, a meno che non sia strettamente necessario, gli indici in alto per indicare lo stadio del processo di eliminazione a cui siamo arrivati. In questa ottica possiamo dire che al k-esimo passo del metodo di Gauss la matrice A e il vettore b vengono modificati come segue: le righe da 1 a k restano invariate, così come le colonne da 1 a $k-1$ di A; gli elementi $a_{k+1,k},\ldots,a_{nk}$ della colonna k-esima diventano uguali a zero; i restanti elementi vengono trasformati con le seguenti formule:

$$\begin{array}{rcll} a_{ij} & = & a_{ij} - m_{ik}a_{kj} & \text{per } \; i,j = k+1,\ldots,n \\ b_i & = & b_i - m_{ik}b_k & \text{per } \; i = k+1,\ldots,n \end{array} \tag{4.14}$$

dove abbiamo posto

$$m_{ik} = \frac{a_{ik}}{a_{kk}}.$$

I numeri m_{ik} sono detti moltiplicatori e vedremo successivamente che rivestono un ruolo molto importante ai fini della stabilità del metodo. I coefficienti a_{kk} che il procedimento utilizza via via per creare i moltiplicatori sono detti pivots. Sottolineiamo che, di questi, soltanto a_{11} coincide con l'elemento originale di A, mentre i successivi derivano dalle trasformazioni effettuate.

Presupposto essenziale perché il metodo di eliminazione di Gauss possa arrivare alla naturale conclusione è che tutti i pivots siano diversi da

zero. Se al k-esimo passo si trova $a_{kk} = 0$, questo significa che l'incognita x_k non è presente nella k-esima equazione. Poiché comunque noi dobbiamo eliminare x_k da tutte le equazioni che seguono la k-esima, il modo più semplice di procedere consiste nel cercare fra queste un'equazione in cui x_k compaia (almeno una deve esserci se $\det(A) \neq 0$), scambiarla con la k-esima, e poi procedere normalmente. Uno scambio nell'ordine delle equazioni è un'operazione perfettamente lecita perché non modifica la soluzione del sistema.

Possiamo quindi affermare che il metodo di eliminazione di Gauss con eventuali scambi di righe per ovviare alla comparsa di pivots uguali a zero è un procedimento matematicamente ben definito che, in aritmetica esatta, porta a calcolare la soluzione attraverso un numero finito predeterminato di operazioni aritmetiche. Il costo computazionale del metodo è $\mathcal{O}(n^3)$ (per la precisione $\simeq \frac{2}{3}n^3$ operazioni) per la fase di eliminazione [20]. A questo va sommato il costo della risoluzione del sistema triangolare che, come abbiamo visto nel paragrafo 4.3, è $\mathcal{O}(n^2)$.

4.4.1 Strategia del pivoting parziale

Finora ci siamo occupati di descrivere il metodo di eliminazione di Gauss prescindendo dal fatto che in pratica esso dovrà essere realizzato in aritmetica floating point. I due esempi successivi mostrano che in questo contesto il metodo può risultare inaffidabile.

Esempio 4.4.1. Questo è un esempio tratto da [17], nel quale si suppone di lavorare in base 10. Il sistema da risolvere è $Ax = b$, con

$$A = \begin{pmatrix} 0.0001 & 0.5 \\ 0.4 & -0.3 \end{pmatrix} \text{ e } b = \begin{pmatrix} 0.5 \\ 0.1 \end{pmatrix}.$$

La soluzione esatta (a 4 cifre) è $(0.9999, 0.9998)^T$.

Il primo e unico moltiplicatore è $m_{21} = fl(a_{21}/a_{11}) = 4000$, che risulta un numero grande in valore assoluto, in virtù del fatto che a_{11} è piccolo rispetto a a_{21}. Usando m_{21}, modifichiamo a_{22} secondo la regola (4.14) e otteniamo:

$$a_{22} = fl(a_{22} - m_{21}a_{12}) = fl(-0.3 - 4000 \times 0.5) = fl(-2000.3);$$

se operiamo con 4 cifre di mantissa, il nuovo valore di a_{22} diventa -2000 e si perde il contributo del valore iniziale (-0.3) di a_{22}, troppo piccolo per essere sentito nella somma con -2000. Passando alla trasformazione di b_2 e operando per arrotondamento, si ottiene:

$$b_2 = fl(b_2 - m_{21}b_1) = fl(0.1 - 4000 \times 0.5) = fl(-1999.9) = -2000.$$

Anche il contributo del valore originale di b_2, 0.1, non viene sentito e il sistema viene quindi trasformato in un sistema triangolare con

$$A = \begin{pmatrix} 0.0001 & 0.5 \\ 0 & -2000 \end{pmatrix} \text{ e } b = \begin{pmatrix} 0.5 \\ -2000 \end{pmatrix}.$$

La soluzione di questo sistema può essere calcolata senza introdurre errori di arrotondamento ed è $\tilde{x} = (0, 1)^T$, completamente diversa dalla soluzione che cercavamo. Questa totale perdita di accuratezza è dovuta esclusivamente agli errori commessi nel modificare a_{22} e b_2, i quali a loro volta sono dovuti alla presenza di un moltiplicatore "grande". È interessante osservare che saremmo giunti alla stessa soluzione se fossimo partiti da

$$A = \begin{pmatrix} 0.0001 & 0.5 \\ 0.4 & 0 \end{pmatrix} \text{ e } b = \begin{pmatrix} 0.5 \\ 0 \end{pmatrix},$$

ma in questo caso avremmo trovato la soluzione esatta! ■

Esempio 4.4.2. In questo esempio la matrice A ha elementi

$$a_{ij} = cos((j-1)\theta), \text{ con } \theta = \frac{(2i-1)\pi}{2n}, \text{ per } i, j = 1, \ldots, n$$

e il vettore b è scelto in modo tale che la soluzione sia $x = (1, 1, \ldots, 1)^T$. La matrice è ben condizionata, con $k_2(A) \simeq 1.4$ per ogni n. Risolvendo

$n = 10$	$n = 15$	n=10	n=15
.9999917	1.005077	.999999999999944	.999999999993959
1.000023	.9899237	1.00000000000007	1.00000000001044
.9999625	1.009260	1.00000000000002	.999999999994602
1.000050	.9933710	.999999999999929	.999999999997942
.9999472	1.002154	1.00000000000007	1.00000000000847
1.000050	1.002377	.999999999999961	.999999999990272
.9999528	.9953206	1.00000000000000	1.00000000000453
1.000043	1.004338	1.00000000000003	1.00000000000490
.9999686	.9966600	.999999999999952	.999999999984748
1.000016	1.003904	1.00000000000004	1.00000000002471
	.9943701		.999999999967255
	1.006998		1.00000000003807
	.9933585		.999999999961888
	1.004807		1.00000000003101
	.9975928		.999999999982622

Tabella 4.3 Esempio 4.4.2: metodo di Gauss

questo sistema in precisione semplice per $n = 10$ e $n = 15$ si ottengono le soluzioni approssimate $\tilde{x}$ mostrate nelle prime due colonne della tabella 4.3, con un errore relativo in norma-2 $\frac{\|x-\tilde{x}\|}{\|x\|}$ pari rispettivamente a 3.9×10^{-5} e 5.7×10^{-3}. Questa scarsa accuratezza è dovuta al fatto che nel corso del procedimento si incontrano dei moltiplicatori grandi, dell'ordine di 10^2 per $n = 10$ e di 10^3 per $n = 15$. Usando la doppia precisione si riesce ad ottenere una soluzione con un numero maggiore di cifre esatte (vedi terza e quarta colonna della tabella). Nonostante questo, una certa perdita di accuratezza

rispetto alla precisione di macchina si manifesta ancora: l'errore relativo diventa è 4.9×10^{-14} per $n = 10$ e 2.1×10^{-11} per $n = 15$. ■

$n = 10$	$n = 15$	n=10	n=15
1.000000	1.000000	1.00000000000000	1.00000000000000
.9999998	1.000000	1.00000000000000	1.00000000000000
1.000000	.9999998	1.00000000000000	1.00000000000000
1.000000	1.000000	1.00000000000000	1.00000000000000
1.000000	1.000000	1.00000000000000	1.00000000000000
.9999999	1.000000	1.00000000000000	1.00000000000000
.9999999	.9999999	1.00000000000000	1.00000000000000
.9999998	.9999999	1.00000000000000	1.00000000000000
1.000000	1.000000	1.00000000000000	1.00000000000000
1.000000	.9999996	1.00000000000000	1.00000000000000
	.9999999		1.00000000000000
	1.000000		1.00000000000000
	.9999999		1.00000000000000
	1.000000		1.00000000000000
	1.000000		1.00000000000000

Tabella 4.4 Esempio 4.4.2: metodo di Gauss con pivoting parziale

In pratica quindi il metodo di eliminazione di Gauss è potenzialmente instabile perché può dare risultati poco accurati rispetto alla precisione di macchina e al condizionamento del problema. Poiché l'instabilità è legata alla comparsa di moltiplicatori "grandi", qualunque strategia per rendere più stabile il metodo deve tendere ad eliminare questa possibilità. La strategia universalmente adottata, nota come strategia del pivoting parziale, consiste nell'individuare al passo k-esimo un indice $r \geq k$ tale che

$$|a_{rk}| = \max_{k \leq i \leq n} |a_{ik}| \tag{4.15}$$

e, nel caso che sia $r > k$, scambiare la riga r-esima con la riga k-esima. In questo modo si ha la certezza che tutti i moltiplicatori sono minori o uguali di 1 in valore assoluto. Se più indici soddisfano la (4.15), di solito si sceglie il più piccolo. Dopo l'eventuale scambio di righe il pivot a_{kk} è sicuramente diverso da zero; se così non fosse, vorrebbe dire che A è singolare.

Se usiamo questa strategia nella risoluzione dei sistemi lineari dell'esempio 4.4.2 otteniamo i risultati, decisamente migliori di prima, mostrati nella tabella 4.4. Questo non è un esempio isolato. Come mostreremo nel paragrafo 4.4.3, la strategia del pivoting parziale consente di stabilizzare il metodo di eliminazione di Gauss nella stragrande maggioranza dei casi, tanto che in letteratura l'espressione "metodo di Gauss" è sinonimo di "metodo di Gauss con pivoting parziale". Come nota storica, ricordiamo che l'aggettivo "parziale" è tradizionalmente usato per distinguere questa

strategia da quella cosiddetta del pivoting totale nella quale si individua una coppia di indici $r, s \geq k$ tali che

$$|a_{rs}| = \max_{k \leq i,j \leq n} |a_{ij}|;$$

se $r > k$ si scambia la riga r-esima con la k-esima e se $s > k$ si scambia la colonna s-esima con la k-esima. Questa strategia è decisamente più onerosa e fra l'altro richiede che si tenga memoria degli scambi di colonna effettuati perché essi corrispondono a scambi di posto fra le incognite. Questo maggior costo non è in generale compensato da una maggior efficacia.

4.4.2 Algoritmi

Il metodo di Gauss con pivoting parziale viene realizzato abitualmente in due fasi. In un primo tempo si applica il processo di eliminazione alla matrice A per ottenere la matrice triangolare superiore U. Se A è non singolare questa fase arriva sicuramente a buon fine. D'altra parte questa ipotesi non è verificabile a priori e può quindi succedere che il procedimento si blocchi a un certo passo k perché tutti gli elementi a_{kk}, $a_{k+1,k}$, ..., a_{nk} sono uguali a zero; in questo caso si ha la certezza che A è singolare. La stessa conclusione può essere tratta se il procedimento arriva in fondo ma l'ultimo elemento diagonale a_{nn} di U è zero. L'algoritmo 4.4.1 che realizza la prima fase del metodo di Gauss prevede fra i risultati la variabile *esito*, il cui scopo è indicare se questa fase ha avuto buon fine e gli elementi diagonali sono tutti diversi da zero.

Se possibile, una volta ridotta in forma triangolare la matrice A, si effettua la trasformazione del vettore dei termini noti b portandolo a d e si risolve il sistema $Ux = d$.

Questo modo di procedere in due fasi ha diversi vantaggi. Uno dei principali consiste nel fatto che se si devono risolvere più sistemi lineari con la stessa matrice dei coefficienti e termini noti diversi, si può effettuare una volta per tutte la riduzione a forma triangolare della matrice che è la fase più costosa. Per esempio, supponiamo di dover calcolare la matrice inversa di A. Possiamo osservare che in virtù della relazione $AA^{-1} = I$, l'i-esima colonna di A^{-1} è soluzione del sistema lineare

$$Ax = e^{(i)},$$

dove $e^{(i)}$ è l'i-esimo vettore della base canonica. Allora il calcolo di A^{-1} è di fatto equivalente alla risoluzione di n sistemi lineari con matrice dei coefficienti A e termini noti $e^{(1)}, e^{(2)}, \ldots, e^{(n)}$.

Affinchè la divisione in due fasi possa avvenire, occorre che l'algoritmo che realizza la prima fase conservi tutte le informazioni utili per la seconda, ovvero i moltiplicatori e l'indicazione degli scambi di righe eventualmente effettuati. Nell'algoritmo 4.4.1 seguiamo le convenzioni degli algoritmi presenti nelle librerie di software di algebra lineare: la matrice triangolare

U viene sovrascritta alla parte triangolare superiore di A e i moltiplicatori vengono conservati nella parte triangolare strettamente inferiore della matrice A, per intendersi la parte di A che alla fine dovrebbe contenere tutti elementi uguali a zero. Per quanto riguarda le informazioni sugli scambi di righe, esse vengono conservate usando un vettore s, di dimensione $n-1$, con elementi interi: $s_k = i$ indica che al k-esimo passo si è usato come pivot l'elemento a_{ii}; pertanto se $i = k$ si deduce che a quel passo non c'è stato nessuno scambio di riga, mentre lo scambio è avvenuto con la riga i-esima se $i > k$.

Algoritmo 4.4.1. Algoritmo di eliminazione di Gauss con pivoting parziale per ridurre la matrice A in forma triangolare.

Dati: n, A

Risultati: *esito* indicatore del risultato:
esito $= 0$ il procedimento è terminato con successo
esito $= k$ al passo k il pivot è uguale a zero
esito $= n$ il valore finale di a_{nn} è uguale a zero
s vettore che ricorda gli eventuali scambi di righe
A matrice modificata rispetto a quella originale:
nel triangolo superiore si trova U
nel triangolo strettamente inferiore si trovano i moltiplicatori

1. Per $k = 1, \ldots, n-1$
   1. Trova un indice r tale che $|a_{rk}| = \max_{k \le i \le n} |a_{ik}|$ usando l'algoritmo 1.3.4
   2. $s_k = r$
   3. Se $a_{rk} = 0$, allora:
      esito $= k$ e fermati
      Fine scelta
   4. Se $r \neq k$, allora:
      Per $j = k, \ldots, n$
      scambia a_{rj} con a_{kj}
      Fine ciclo su j
      Fine scelta
   5. Per $i = k+1, \ldots, n$
      1. $a_{ik} = a_{ik}/a_{kk}$
      2. Per $j = k+1, \ldots, n$
         $a_{ij} = a_{ij} - a_{ik}a_{kj}$
         Fine ciclo su j
      Fine ciclo su i

   Fine ciclo su k
2. Se $a_{nn} = 0$, allora:
   esito $= n$
   altrimenti:
   esito $= 0$

   Fine scelta
3. Fine

L'algoritmo che segue utilizza i risultati dell'algoritmo 4.4.1 per eseguire la

seconda fase del processo di trasformazione di $Ax = b$ in $Ux = d$ e calcolare la soluzione x. Nello stesso spirito di prima, l'algoritmo sovrascrive d a b.

Algoritmo 4.4.2. Algoritmo che applica il metodo di Gauss per trasformare il vettore b e risolve il sistema.
Dati: n, b
A e s come usciti dall'algoritmo 4.4.1
Risultati: x
1. Per $k = 1, \ldots, n-1$
   1. Scambia b_{s_k} con b_k
   2. Per $i = k+1, \ldots, n$
      $b_i = b_i - a_{ik} b_k$
      Fine ciclo su i
   Fine ciclo su k
2. Risolvi $Ax = b$ usando l'algoritmo 4.3.1
3. Fine

Gli scambi previsti nei due algoritmi possono essere facilmente realizzati. Infatti per scambiare i valori di due variabili x e y si può usare una variabile d'appoggio z e eseguire in sequenza le tre assegnazioni

$$\begin{aligned} z &= x \\ x &= y \\ y &= z. \end{aligned}$$

Il metodo di eliminazione di Gauss senza scambi di righe può essere visto come un metodo di fattorizzazione della matrice A nel senso che, indicata con L la matrice triangolare inferiore con elementi diagonali tutti uguali a 1 e elementi sottodiagonali uguali ai moltiplicatori, A risulta uguale al prodotto dei due fattori L ed U, cioè

$$A = LU. \tag{4.16}$$

Quando intervengono scambi di righe, ad esempio perché si usa la strategia del pivoting parziale, le cose si complicano leggermente; è comunque possibile far vedere (cfr. [6] o [15]) che il metodo di Gauss produce una fattorizzazione del tipo

$$PA = LU \tag{4.17}$$

dove P è una matrice che tiene conto degli scambi di righe [5] e L è ancora una matrice triangolare inferiore con elementi diagonali uguali ad 1 e con i moltiplicatori nella parte sottodiagonale; la differenza rispetto alla matrice

[5] Si tratta di una matrice di permutazione, ottenuta dalla matrice identità scambiando le righe coerentemente con gli scambi effettuati su A.

L che compare in (4.16) è che in questo caso i moltiplicatori hanno subito tutti gli scambi di righe che sono stati effettuati su A.

La relazione (4.17) mette in evidenza che il metodo di Gauss può essere utilizzato anche per calcolare il determinante di A. Infatti per il Teorema di Binet si ha

$$\det(P)\det(A) = \det(L)\det(U).$$

Dato che $\det(L) = 1$ e $\det(P) = (-1)^q$, dove q è il numero di scambi di righe effettuati (poiché il determinante della matrice identità è 1 e ogni scambio di righe provoca un cambiamento di segno del determinante), possiamo concludere che

$$\det(A) = (-1)^q \det(U).$$

Non è difficile modificare l'algoritmo 4.4.1 per ottenere come risultato anche $\det(A)$. La funzione *det* di MATLAB usa proprio questa procedura.

Date P, L e U, il sistema $Ax = b$ viene riscritto come $LUx = Pb$ e la sua risoluzione può avvenire attraverso la risoluzione in sequenza di due sistemi triangolari; il primo è

$$Ld = Pb \tag{4.18}$$

e il secondo è

$$Ux = d. \tag{4.19}$$

Negli algoritmi che abbiamo scritto non abbiamo esplicitamente costruito L, ma abbiamo implicitamente risolto (4.18) nella prima parte dell'algoritmo 4.4.2.

4.4.3 Errore nel metodo di Gauss con pivoting parziale

La relazione (4.17) è il punto di partenza per l'analisi della propagazione degli errori di arrotondamento dovuti alle operazioni aritmetiche nell'eliminazione di Gauss con pivoting parziale. Per escludere da questa analisi la propagazione degli errori eventualmente presenti sui dati, si fa l'ipotesi (di lavoro) che i coefficienti e i termini noti siano numeri di macchina. Lo scopo dell'analisi è quello di fornire delle stime dell'errore, ovvero della distanza fra la soluzione vera del sistema x e la soluzione calcolata $\tilde{x}$. I primi risultati in questo senso sono dovuti a J.H. Wilkinson e molti altri ricercatori hanno poi proseguito e approfondito lo studio. Il risultato fondamentale di Wilkinson dice che la soluzione calcolata $\tilde{x}$ è soluzione esatta di un sistema perturbato

$$(A + E)x = b,$$

con

$$\frac{\|E\|}{\|A\|} = \mathcal{O}(\epsilon_m).$$

In letteratura si possono trovare molte stime dell'errore dedotte dal risultato di Wilkinson assumendo ipotesi tecniche diverse; queste stime hanno forme diverse, ma sono tutte concordi nella sostanza, che noi riassumiamo, come in [23], nella seguente disuguaglianza:

$$\frac{\|x - \tilde{x}\|}{\|\tilde{x}\|} \leq G_n k(A) \epsilon_m, \tag{4.20}$$

dove G_n è una costante che dipende da n. Il valore di G_n risulta grande quando il cosiddetto fattore di crescita

$$\rho_n = \frac{\max_{i,j,k} |a_{ij}^{(k)}|}{\max_{i,j} |a_{ij}|}$$

è grande, ovvero quando ad un qualche stadio del processo di trasformazione si manifesta una grossa crescita in qualche elemento della matrice. Teoricamente si dimostra che ρ_n può assumere valori fino a 2^{n-1}, ma nella pratica il fattore di crescita non è quasi mai grande e G_n risulta eventualmente crescere, ma con lentezza, all'aumentare di n. Tenendo conto di queste considerazioni, la relazione (4.20) esprime il fatto che il metodo di Gauss con pivoting parziale mantiene gli errori nei limiti della precisione di macchina e del condizionamento del problema. Questo significa che, come avevamo preannunciato, questo metodo è in generale stabile.

Alla luce della relazione (4.20) possiamo riconsiderare i risultati della tabella 4.4. Poiché il sistema è ottimamente condizionato per entrambi i valori di n considerati, se trascuriamo il fattore G_n ci aspettiamo errori dell'ordine di grandezza della precisione di macchina. In effetti, gli errori relativi in precisione semplice sono rispettivamente uguali a 1.2×10^{-7} per $n = 10$ e 1.9×10^{-7} per $n = 15$. In doppia precisione essi diventano 2.2×10^{-16} e 5.2×10^{-16} rispettivamente.

Esempio 4.4.3. Utilizziamo il metodo di Gauss con pivoting parziale per risolvere i sistemi lineari di dimensione $n = 4$ e $n = 8$ dove la matrice dei coefficienti è una matrice di Hilbert (cfr. esempio 4.2.2) e il vettore b è scelto in modo che la soluzione sia $x = (1, 1, \ldots, 1)^T$. I risultati sono riportati in tabella 4.5, sia per la semplice che per la doppia precisione; oltre alla soluzione calcolata riportiamo anche l'errore relativo

$$e_r = \frac{\|x - \tilde{x}\|}{\|x\|}$$

e il residuo relativo

$$r_r = \frac{\|Ax - b\|}{\|b\|}$$

misurati in norma-2. Poiché il numero di condizionamento di A è circa 10^4 per $n = 4$ e 10^{10} per $n = 8$, i risultati rispecchiano perfettamente

	$n = 4$	$n = 8$	n=4	n=8
	1.000020	.9997690	.999999999999938	1.00000000002308
	.9997907	1.012122	1.00000000000067	.999999988037014
	1.000489	.8770694	.999999999998406	1.00000001521925
	.9996872	1.444963	1.00000000000103	.999999919308401
		.3684078		1.00000021369769
		1.168472		.999999701667583
		1.338944		1.00000020994476
		.7898017		.999999941322467
e_r	4.9e−04	6.3e−01	1.6e−12	3.0e−07
r_r	1.9e−08	1.0e−08	1.8e−17	1.2e−17

Tabella 4.5 Esempio 4.4.3: metodo di Gauss con pivoting parziale

la stima (4.20). Guardando i valori dei residui relativi osserviamo che questi valori sono molto piccoli in tutti i casi, anche se e_r è grande: più precisamente l'errore relativo è in quasi tutti i casi circa uguale al residuo relativo moltiplicato per $k(A)$; questo corrisponde al caso peggiore previsto dalla relazione (4.13). ■

Concludiamo con un esempio dovuto proprio a Wilkinson nel quale il metodo di Gauss si rivela instabile perché il fattore di crescita risulta grande e quindi si verifica il caso peggiore previsto dalla relazione (4.20).

Esempio 4.4.4. Si può facilmente verificare che applicando l'eliminazione di Gauss alla matrice

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & \dots & 0 & 1 \\ -1 & 1 & 0 & 0 & \dots & 0 & 1 \\ -1 & -1 & 1 & 0 & \dots & 0 & 1 \\ \vdots & & & & & & 1 \\ -1 & -1 & -1 & -1 & \dots & -1 & 1 \end{pmatrix}$$

di dimensione n, si ottiene una matrice triangolare superiore U che contiene elementi di grandezza pari a 2^{n-1} e quindi si ha un fattore di crescita $\rho_n = 2^{n-1}$. Abbiamo risolto in doppia precisione il sistema $Ax = b$ con b scelto in modo che la soluzione sia $x = (0.1, 0.1, \dots, 0.1)^T$ per diversi valori di n. Nella tabella 4.6 riportiamo per ogni n l'errore relativo e_r. A titolo informativo riportiamo anche il fattore di crescita ρ_n, il numero di condizionamento $k_2(A)$ e il residuo relativo r_r.
Dalla tabella si vede che all'aumentare di n la soluzione calcolata risulta sempre meno accurata. Poiché il problema è ottimamente condizionato per ogni valore di n, la perdita di accuratezza è da attribuirsi al fattore di crescita che si ripercuote sempre più negativamente sul risultato. Osserviamo che, coerentemente con la relazione (4.13), anche il residuo relativo aumenta al crescere di n. ■

n	$k_2(A)$	ρ_n	e_r	r_r
10	4.4e+00	5.1e+02	4.8e−15	1.8e−15
15	6.6e+00	1.6e+04	8.6e−14	1.9e−14
20	8.8e+00	5.2e+05	3.5e−12	5.1e−13
25	1.1e+01	1.6e+07	6.8e−11	8.2e−12
30	1.3e+01	5.4e+08	2.9e−09	2.8e−10
35	1.6e+01	1.7e+10	5.9e−08	6.9e−09
40	1.8e+01	5.5e+11	2.6e−06	1.8e−07
45	2.0e+01	1.8e+13	5.3e−05	3.4e−06
50	2.2e+01	5.6e+14	2.4e−03	1.3e−04
55	2.5e+01	1.8e+16	4.9e−02	1.5e−03
60	2.7e+01	5.7e+17	2.9e−01	2.3e−02

Tabella 4.6 Esempio 4.4.4: metodo di Gauss con pivoting parziale

4.5 Metodo di Householder (*)

Presentiamo ora un metodo alternativo al metodo di Gauss per ridurre alla forma triangolare le matrici e di conseguenza risolvere i sistemi lineari. Il vantaggio di questo metodo è la stabilità, garantita senza necessità di particolari strategie di pivoting; anzi, il nuovo metodo non soffre neppure della potenziale (sia pur statisticamente irrilevante) instabilità del metodo di Gauss con pivoting parziale legata al fenomeno della crescita. A fronte di queste ottime caratteristiche, lo svantaggio di questo nuovo metodo sta nel costo computazionale: esso infatti richiede circa il doppio di operazioni aritmetiche del metodo di Gauss.

Il metodo che andiamo a descrivere si basa sull'utilizzo di matrici ortogonali. Ricordiamo che una matrice $Q \in \mathbb{R}^{n\times n}$ è detta ortogonale se $QQ^T = Q^TQ = I$, ovvero se la sua inversa coincide con la trasposta. La relazione $Q^TQ = I$ implica che le colonne di Q, viste come vettori in $\mathbb{R}^n$, hanno tutte norma euclidea uguale a 1 e sono ortogonali l'una all'altra; da qui deriva l'aggettivo "ortogonali". Il motivo per cui nell'algebra lineare numerica le matrici ortogonali sono "sinonimo di stabilità" è che queste matrici non alterano la lunghezza euclidea dei vettori. Infatti, per un qualunque vettore $x \in \mathbb{R}^n$ si ha

$$\|Qx\|_2 = (Qx)^T(Qx) = x^TQ^TQx = x^TIx = x^Tx = \|x\|_2.$$

Da questo segue che gli elementi del vettore Qx non possono essere significativamente più grandi di quelli di x.

4.5.1 Matrici di Householder

Le matrici di Householder [6] hanno la forma

$$Q = I - 2\frac{uu^T}{u^T u},$$

dove u è un vettore qualunque in $\mathbb{R}^n$ diverso da zero. Poiché $u^T u = \|u\|_2^2$, la matrice Q dipende esclusivamente dalla direzione di u e non dalla sua lunghezza. Talvolta scriveremo

$$Q = I - \frac{1}{\beta}uu^T, \text{ con } \beta = \frac{u^T u}{2}.$$

Qualunque sia u, la matrice Q è simmetrica in quanto somma di due matrici simmetriche, la matrice identità e la matrice $B = \frac{1}{\beta}uu^T$ che ha elementi $b_{ij} = \frac{u_i u_j}{\beta}$. È inoltre facile vedere che Q è ortogonale; infatti

$$Q^T Q = QQ^T = QQ = I - \frac{2}{\beta}uu^T + \frac{1}{\beta^2}uu^T uu^T = I - \frac{2}{\beta}uu^T + 2\frac{\beta}{\beta^2}uu^T = I.$$

Una proprietà che rende le matrici di Householder molto comode da usare nella pratica è la seguente: per calcolare il prodotto fra la matrice Q e un vettore x non occorre costruire esplicitamente Q, ma basta conoscere u, dal momento che

$$Qx = \left(I - \frac{1}{\beta}uu^T\right)x = x - \frac{1}{\beta}uu^T x = x - \frac{u^T x}{\beta}u. \tag{4.21}$$

Le matrici di Householder sono dette anche riflettori elementari, a causa del loro significato geometrico, che è illustrato in figura 4.1. Supponiamo per semplicità di essere in $\mathbb{R}^2$ e sia u un vettore fissato. Consideriamo un vettore x e scomponiamolo nelle sue componenti lungo la direzione di u e quella perpendicolare, ovvero scriviamo $x = \alpha u + w$, con α scalare e $w \in \mathbb{R}^2$ tale che $w^T u = 0$. Applichiamo Q ad x e otteniamo

$$Qx = (I - \frac{1}{\beta}uu^T)(\alpha u + w) = \alpha(I - \frac{u^T u}{\beta})u + w - \frac{1}{\beta}uu^T w;$$

poiché $\beta = \frac{u^T u}{2}$ e $u^T w = 0$, si ha $Qx = -\alpha u + w$. Vediamo quindi che il vettore Qx ha la stessa componente di x lungo la direzione perpendicolare

[6] Queste speciali matrici vennero usate per la prima volta in un contesto diverso da quello dei sistemi lineari in un libro di H.W. Turnbull e A.C. Aitken nel 1932. Esse devono però la loro notorietà al fatto che Alston Scott Householder (1904-1993) le utilizzò come strumento per trasformare le matrici in forma triangolare in un lavoro del 1958. Per questo motivo oggi esse sono note come "matrici di Householder".

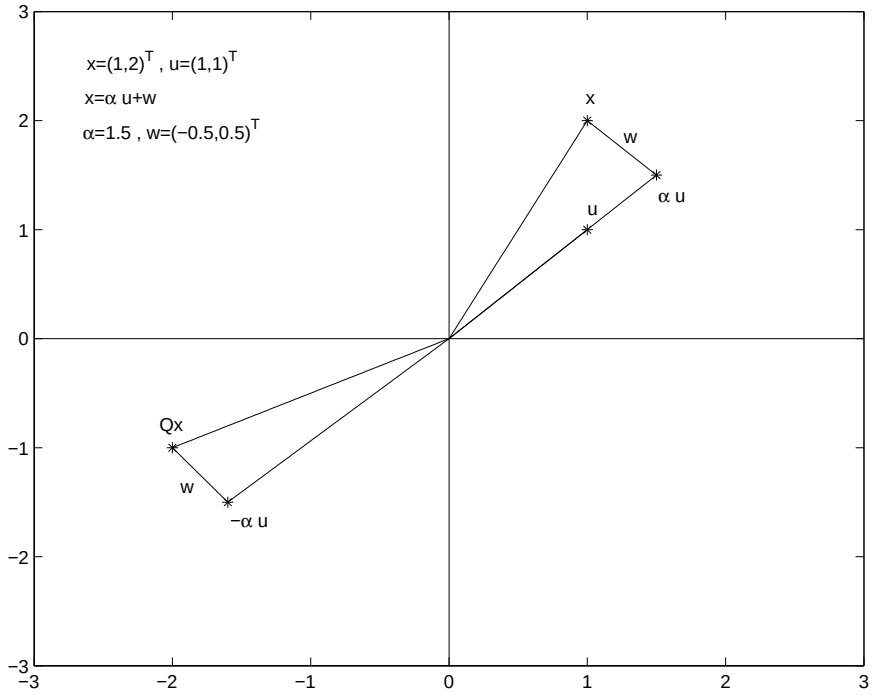

Figura 4.1 Significato geometrico delle matrici di Householder

a u e la componente lungo la direzione di u risulta uguale ma di segno opposto: in pratica Q ha riflesso x rispetto alla direzione di w.

Basandosi su questa proprietà geometrica si intuisce un fatto di cui omettiamo per semplicità la dimostrazione: dati due vettori $x, y \in \mathbb{R}^n$ di uguale lunghezza ($\|x\|_2 = \|y\|_2$) si può sempre scegliere u in modo tale che sia $Qx = y$. Gli esempi successivi mostrano alcune situazioni di questo tipo.

Esempio 4.5.1. Supponiamo di voler determinare una matrice di Householder Q che porti un dato vettore x sul primo asse coordinato. In altri termini, vogliamo Q tale che $Qx = y = (y_1, 0, \ldots, 0)^T$; poichè sappiamo che y deve avere la stessa lunghezza di x, dovrà necessariamente essere $y_1 = \pm\|x\|_2$. A questo scopo possiamo usare $u = x \pm \|x\|_2 e^{(1)}$, dove $e^{(1)}$ è il primo vettore della base canonica di $\mathbb{R}^n$, cioè

$$u = \begin{pmatrix} x_1 \pm \|x\|_2 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}. \tag{4.22}$$

Per verificare che con questa scelta raggiungiamo il nostro scopo, facciamo riferimento alla (4.21) e calcoliamo separatamente $u^T x$ e β. Si ottiene

$$u^T x = (x \pm \|x\|_2 e^{(1)})^T x = x^T x \pm \|x\|_2 x_1 = \|x\|_2^2 \pm \|x\|_2 x_1 = \|x\|_2(\|x\|_2 \pm x_1);$$

$$\beta = \frac{(x \pm \|x\|_2 e^{(1)})^T (x \pm \|x\|_2 e^{(1)})}{2} = \frac{x^T x + \|x\|_2^2 \pm 2\|x\|_2 x_1}{2} = \|x\|_2(\|x\|_2 \pm x_1).$$

Essendo $u^T x = \beta$, la (4.21) dà

$$y = x - \frac{u^T x}{\beta} u = x - u = \mp \|x\|_2 e^{(1)} = \begin{pmatrix} \mp\|x\|_2 \\ 0 \\ \vdots \\ 0 \end{pmatrix}.$$

La scelta del segno + o − nella (4.22) viene fatta generalmente in base al segno di x_1: se $x_1 < 0$ si sceglie il segno −, altrimenti si sceglie il segno +; in questo modo si evita una sottrazione, che potrebbe causare errori di cancellazione nel caso che x avesse la prima componente x_1 molto più grande delle altre. ■

Esempio 4.5.2. Cerchiamo una matrice di Householder che trasformi x in un vettore della forma $y = (y_1, y_2, 0, \ldots, 0)^T$ con $\|y\|_2 = \|x\|_2$. Un modo per procedere è il seguente: posto $x' = (x_2, x_3, \ldots, x_n)^T \in \mathbb{R}^{n-1}$, consideriamo il vettore

$$u = \begin{pmatrix} 0 \\ x_2 \pm \|x'\|_2 \\ x_3 \\ \vdots \\ x_n \end{pmatrix},$$

dove riconosciamo dalla seconda componente in poi un vettore di dimensione $n-1$ scelto in modo da annullare tutte le componenti di x' eccetto la prima. Lasciamo per esercizio al lettore la verifica del fatto che questa scelta di u porta ad una matrice di Householder che fa al caso nostro. Come suggerimento, osserviamo che, avendo posto la prima componente di u uguale a zero, la matrice Q corrispondente ha la forma

$$Q = \begin{pmatrix} 1 & 0 & 0 & \ldots & 0 \\ 0 & * & * & \ldots & * \\ 0 & * & * & \ldots & * \\ \cdot & \cdot & \cdot & \ldots & \cdot \\ 0 & * & * & \ldots & * \end{pmatrix} \tag{4.23}$$

e non altera la prima componente di x. Inoltre la sottomatrice di dimensione $n-1$ che si ottiene trascurando prima riga e prima colonna di Q altro non

è che la matrice di Householder che trasforma x' in un vettore con tutte le componenti uguali a zero eccetto la prima. Con questa scelta si ottiene

$$y = Qx = \begin{pmatrix} x_1 \\ \mp\|x'\|_2 \\ 0 \\ \vdots \\ 0 \end{pmatrix}.$$

■

La costruzione dell'esempio precedente può essere generalizzata. Per trasformare mediante una matrice di Householder un dato vettore x in un vettore del tipo $y = (x_1, x_2, \ldots, x_k, y_{k+1}, 0, \ldots, 0)^T$ con $k \leq n-2$ fissato dobbiamo usare

$$u = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ x_{k+1} \pm \|x'\|_2 \\ x_{k+2} \\ \vdots \\ x_n \end{pmatrix}, \quad \text{dove} \quad x' = \begin{pmatrix} x_{k+1} \\ x_{k+2} \\ \vdots \\ x_n \end{pmatrix}.$$

Infatti, in virtù del fatto che i primi k elementi di u sono uguali a zero, la matrice di Householder corrispondente Q ha la forma

$$Q = \begin{pmatrix} I_k & 0 & 0 & \ldots & 0 \\ 0 & * & * & \ldots & * \\ 0 & * & * & \ldots & * \\ \cdot & \cdot & \cdot & \ldots & \cdot \\ 0 & * & * & \ldots & * \end{pmatrix},$$

dove I_k indica la matrice identità di dimensione k e i simboli 0 nella prima riga (colonna) sono da intendersi come vettori colonna (riga) di k componenti tutte uguali a zero. Questo implica che Q lascia invariate le prime k componenti di x. Le successive componenti di u sono scelte in modo da annullare tutte le componenti di x' eccetto la prima.

4.5.2 Riduzione di una matrice in forma triangolare mediante matrici di Householder

Il processo di riduzione di una matrice quadrata in forma triangolare superiore può essere visto nel modo seguente: la prima colonna deve essere

trasformata in un vettore con tutte le componenti uguali a zero eccetto la prima, la seconda colonna in un vettore con tutte le componenti uguali a zero eccetto le prime due, e così via fino alla penultima colonna che deve essere trasformata in un vettore con l'ultimo elemento uguale a zero. Ognuna di queste trasformazioni può essere realizzata con un'opportuna matrice di Householder.

Data la matrice A di dimensione n, applichiamo la trasformazione Q_1 che annulla tutti gli elementi della prima colonna eccetto il primo. Otterremo allora una matrice trasformata con la struttura

$$A^{(1)} = Q_1 A = \begin{pmatrix} * & * & * & \cdots & * \\ 0 & * & * & \cdots & * \\ 0 & * & * & \cdots & * \\ \cdot & \cdot & \cdot & \cdots & \cdot \\ 0 & * & * & \cdots & * \end{pmatrix},$$

nella quale tutti gli elementi di A risultano cambiati. Ora applichiamo ad $A^{(1)}$ la matrice di Householder Q_2 che annulla tutti gli elementi della sua seconda colonna eccetto i primi due e otteniamo

$$A^{(2)} = Q_2 A^{(1)} = Q_2 Q_1 A = \begin{pmatrix} * & * & * & \cdots & * \\ 0 & * & * & \cdots & * \\ 0 & 0 & * & \cdots & * \\ \cdot & \cdot & \cdot & \cdots & \cdot \\ 0 & 0 & * & \cdots & * \end{pmatrix}.$$

Poiché Q_2 ha la forma (4.23), essa non produce cambiamenti nella prima riga e nella prima colonna di $A^{(1)}$, ma ne cambia tutti gli altri elementi. Proseguendo in questo modo si arriva a determinare la matrice di Householder Q_{n-1} che annulla l'elemento di indici $(n, n-1)$ senza modificare le righe e colonne da 1 a $n-2$.

Riassumendo, possiamo ridurre la matrice A in forma triangolare superiore usando una sequenza finita $Q_1, Q_2, \ldots, Q_{n-1}$ di matrici ortogonali tali che

$$A^{(n-1)} = Q_{n-1} \ldots Q_2 Q_1 A = R, \tag{4.24}$$

con R triangolare superiore. Il metodo di Householder per la risoluzione di un sistema lineare $Ax = b$ si basa proprio su questo processo di triangolarizzazione.
Osserviamo che la relazione (4.24) può essere riscritta nella forma

$$A = (Q_{n-1} \ldots Q_2 Q_1)^{-1} R = Q_1^{-1} Q_2^{-1} \ldots Q_{n-1}^{-1} R.$$

Poiché tutte le matrici $Q_1, Q_2, \ldots, Q_{n-1}$ sono ortogonali, deduciamo che

$$A = Q_1^T Q_2^T \ldots Q_{n-1}^T R = (Q_{n-1} \ldots Q_2 Q_1)^T R.$$

Se poniamo $Q = (Q_{n-1} \dots Q_2 Q_1)^T$, possiamo riscrivere la (4.24) come una fattorizzazione

$$A = QR.$$

Ai fini della risoluzione di $Ax = b$, questo equivale a dover risolvere in cascata i due sistemi

$$Qd = b$$

e

$$Rx = d$$

il primo dei quali è di immediata risoluzione: $d = Q^T b$.

4.5.3 Algoritmi

Per scrivere degli algoritmi che realizzino il metodo di Householder abbiamo due possibilità, come per il metodo di Gauss: calcolare contemporaneamente R e d applicando le trasformazioni $Q_1, Q_2, \dots, Q_{n-1}$ ad A e b , oppure calcolare R conservando le informazioni essenziali per creare d in un secondo tempo. Seguiamo la seconda strada, costruendo due algoritmi distinti per le due fasi. Le informazioni che il primo algoritmo deve conservare sono i vettori necessari a calcolare le matrici di Householder $Q_1, Q_2, \dots, Q_{n-1}$. Consideriamo il k-esimo vettore u: poiché le sue prime $k-1$ componenti sono nulle, è sufficiente memorizzare le sue ultime componenti, da u_k a u_n. Poiché inoltre la matrice Q_k dipende soltanto dalla direzione di u, possiamo normalizzare il vettore in modo che la sua k-esima componente (ovvero la prima di quelle che dobbiamo conservare) sia uguale a 1. Con questa convenzione, per memorizzare le informazioni essenziali su u e poter successivamente ricostruire Q_k bastano $n-k$ locazioni di memoria. Come nell'algoritmo di Gauss scegliamo di usare a questo scopo le $n-k$ locazioni degli elementi $a_{k+1,k}, \dots, a_{nk}$ che la trasformazione Q_k rende uguali a zero.

L'algoritmo 4.5.1 realizza la prima fase del metodo di Householder. Fra i risultati dell'algoritmo è prevista la variabile *esito*, il cui scopo è indicare se la riduzione a forma triangolare ha avuto buon fine e gli elementi diagonali sono tutti diversi da zero. Se ad un passo della riduzione in forma triangolare gli elementi $a_{kk}, \dots, a_{nk}$ sono tutti uguali a zero, questo significa che la matrice è singolare e la triangolarizzazione si arresta. Nell' algoritmo abbiamo utilizzato per semplicità alcune istruzioni di assegnazione "vettoriali", nelle quali a sinistra e a destra del segno di "=" non compaiono grandezze scalari, bensì vettoriali. Queste istruzioni sono da intendersi come assegnazioni elemento per elemento e sono di fatto equivalenti a una sequenza di assegnazioni; ad esempio, l'istruzione $\bar{u} = (a_{kk}, \dots, a_{nk})^T$ equivale alla sequenza di assegnazioni $\bar{u}_1 = a_{kk}, \bar{u}_2 = a_{k+1,k}, \dots, \bar{u}_{n-(k-1)} = a_{nk}$. Infine, il calcolo della norma al passo 1.1 e del prodotto scalare al passo 1.9.2 possono essere realizzati usando l'algoritmo 1.3.1 per calcolare le sommatorie.

Algoritmo 4.5.1. Algoritmo di Householder per ridurre la matrice A in forma triangolare.

Dati: n, A
Risultati: $esito$ indicatore del risultato:
$esito = 0$ il procedimento è terminato con successo
$esito = k$ al passo k si ha $a_{kk} = \ldots = a_{nk} = 0$
A matrice modificata rispetto a quella originale:
nel triangolo superiore si trova R
nel triangolo strettamente inferiore si trovano le informazioni necessarie per ricostruire i vettori u di Householder

1. Per $k = 1, \ldots, n-1$
   1. $\bar{u} = (a_{kk}, \ldots, a_{nk})^T$
   2. $\sigma = \|\bar{u}\|_2$
   3. Se $\sigma = 0$, allora:
      $esito = k$ e fermati
      Fine scelta
   4. Se $a_{kk} < 0$, allora:
      $\sigma = -\sigma$
      Fine scelta
   5. $\bar{u}_1 = \bar{u}_1 + \sigma$
   6. $\beta = |\sigma|(|a_{kk}| + |\sigma|)$
   7. $a_{kk} = -\sigma$
   8. Per $i = k+1, \ldots, n$
      1. $a_{ik} = \frac{\bar{u}_{i-(k-1)}}{|\bar{u}_1|}$
      Fine ciclo su i
   9. Per $j = k+1, \ldots, n$
      1. $w = (a_{kj}, \ldots, a_{nj})^T$
      2. $\gamma = \frac{\bar{u}^T w}{\beta}$
      3. Per $i = k, \ldots, n$
         1. $a_{ij} = a_{ij} - \gamma \bar{u}_{i-(k-1)}$
      Fine ciclo su i
   Fine ciclo su j
   Fine ciclo su k
2. Se $a_{nn} = 0$, allora: $esito = n$
   altrimenti: $esito = 0$
   Fine scelta
3. Fine

L'algoritmo che segue utilizza i risultati dell'algoritmo 4.5.1 per trasformare il termine noto b in $d = Q^T b$ e calcolare la soluzione x. Nello stesso spirito dell'algoritmo 4.4.2, d viene sovrascritto a b.

Algoritmo 4.5.2. Algoritmo che applica le trasformazioni di Householder al vettore b e risolve il sistema triangolare $Ax = b$.

Dati: n, b e A come uscita dall'algoritmo 4.5.1
Risultati: b modificato , x
1.Per $k = 1, \ldots, n-1$
1. $\bar{u} = (1, a_{k+1,k}, \ldots, a_{nk})^T$
2. $\beta = \frac{\bar{u}^T \bar{u}}{2}$
3. $w = (b_k, \ldots, b_n)^T$
4. $\gamma = \frac{\bar{u}^T w}{\beta}$
5. Per $i = k, \ldots, n$
    1. $b_i = b_i - \gamma \bar{u}_{i-(k-1)k}$

   Fine ciclo su i

Fine ciclo su k
2. Risolvi $Ax = b$ usando l'algoritmo 4.3.1
3. Fine

4.5.4 Errore nel metodo di Householder

Esempio 4.5.3. Utilizziamo il metodo di Householder in doppia precisione per risolvere i sistemi lineari di dimensione $n = 4$ e $n = 8$ dove la matrice dei coefficienti è una matrice di Hilbert e il vettore b è scelto in modo che la soluzione sia $x = (1, 1, \ldots, 1)^T$ (cfr. esempio 4.4.3). La tabella 4.7 mostra

	n=4	n=8
	.999999999999960	.999999999960079
	1.00000000000046	1.00000000198989
	.999999999998872	.999999973481358
	1.00000000000074	1.00000014553078
		.999999604170316
		1.00000056448044
		.999999595795334
		1.00000011461771
e_r	7.1e−13	2.9e−07
r_r	1.7e−16	1.4e−16

Tabella 4.7 Esempio 4.5.3: metodo di Householder

che, anche se le soluzioni sono diverse da quelle ottenute con il metodo di Gauss con pivoting parziale, gli errori e i residui sono perfettamente confrontabili. Questo indica che il metodo di Householder non ha bisogno di strategie di pivoting per risolvere un sistema con la massima accuratezza concessa dal condizionamento del problema e dalla precisione di macchina. ■

Esempio 4.5.4. Osserviamo la tabella 4.8. Essa riguarda la risoluzione me-

n	e_r	r_r
10	6.9e−16	2.9e−16
15	2.0e−15	5.4e−16
20	1.1e−15	1.5e−16
25	2.3e−15	3.3e−16
30	2.6e−15	2.9e−16
35	4.7e−15	4.2e−16
40	3.1e−15	2.4e−16
45	3.3e−15	3.6e−16
50	5.9e−15	4.9e−16
55	4.0e−15	2.5e−16
60	5.2e−15	3.5e−16

Tabella 4.8 Esempio 4.5.4: metodo di Householder

diante il metodo di Householder del sistema lineare dell'esempio 4.4.4, per il quale il metodo di Gauss con pivoting parziale risulta instabile. Dalla tabella si vede che all'aumentare di n l'accuratezza non subisce alcun deterioramento. ■

I risultati di questi esempi trovano conferma in importanti risultati teorici. J.H. Wilkinson ha dimostrato infatti che il metodo di Householder è un metodo stabile. Utilizzando tecniche abbastanza complesse per analizzare la propagazione degli errori attraverso le varie fasi del metodo, si dimostra che vale una disuguaglianza simile alla (4.20):

$$\frac{\|x - \tilde{x}\|}{\|\tilde{x}\|} \leq H_n k(A) \epsilon_m,$$

dove $\tilde{x}$ è la soluzione calcolata in precisione finita con il metodo di Householder e H_n è una costante che cresce lentamente con n (orientativamente come n^2). L'aspetto fondamentale è che per questo metodo non possono verificarsi fenomeni di crescita analoghi a quelli del metodo di Gauss; come avevamo preannunciato all'inizio del paragrafo, questo è legato al fatto che le matrici ortogonali non alterano la lunghezza dei vettori.

4.6 Metodi iterativi stazionari

I metodi iterativi per la risoluzione dei sistemi lineari sono utili tutte le volte che la dimensione n del problema è grande e, come spesso succede, la matrice dei coefficienti A presenta molti elementi uguali a zero, disposti in ordine sparso o con una qualche struttura.

Il termine "grande" è decisamente vago. Nel diciannovesimo secolo, ad esempio, quando non esistevano gli elaboratori elettronici e i calcoli dovevano essere fatti fondamentalmente a mano, anche un sistema di 5 equazioni

in 5 incognite poteva essere considerato grande. Nel secolo scorso, con l'evolversi degli strumenti di calcolo, la frontiera del "grande" si è sempre più spostata in avanti. Di epoca in epoca "grande" è un sistema che non può essere risolto in tempi ragionevoli con le macchine disponibili. Come osservato in [36], nel 1950 "grande" equivaleva a $n > 20$, nel 1965 a $n > 200$, nel 1980 a $n > 2000$, nel 1995 a $n > 20000$; attualmente potremmo dire che "grande" è un sistema con $n > 200000$.

In generale, la presenza di molti elementi nulli nella matrice A non può essere sfruttata nei metodi diretti ai fini di ridurre in modo cospicuo il costo computazionale, sia in termini di occupazione di memoria che in termini di numero di operazioni da eseguire. Questo dipende molto semplicemente dal fatto che la trasformazione di A può introdurre un numero diverso da zero laddove prima c'era uno zero. Così, salvo situazioni particolari in cui la matrice è fortemente strutturata [7] e si può sfruttare la struttura per ridurre il costo computazionale, i metodi diretti non sono adatti ai problemi di grandi dimensioni. D'altra parte la presenza di molti zeri in una matrice può essere sicuramente sfruttata in toto per ridurre il costo della moltiplicazione della matrice per un vettore. In questo caso infatti si può risparmiare memoria (memorizzando soltanto gli elementi diversi da zero e le informazioni utili a ricordare la loro posizione nella matrice) e tempo di calcolo (ricordiamo che il vettore $y = Ax$ ha componenti $y_i = \sum_{j=1}^{n} a_{ij}x_j$ e i termini $a_{ij}x_j$ non portano nessun contributo se $a_{ij} = 0$). Allora la domanda diventa: è possibile risolvere un sistema lineare usando soltanto moltiplicazioni fra A (o parte di essa) e opportuni vettori? La risposta è sì, purché ci si accontenti di soluzioni approssimate. Infatti, i metodi che usano soltanto moltiplicazioni fra A e vettori sono tutti metodi iterativi che, partendo da un'approssimazione iniziale, generano una successione di vettori che (si spera) converge alla soluzione cercata. Come si è già visto nel Capitolo 3, un metodo siffatto dovrà essere necessariamente interrotto prima di aver raggiunto la soluzione esatta, in base a qualche opportuno criterio di arresto che ci faccia ritenere l'iterata corrente una approssimazione sufficientemente buona della soluzione. Il vantaggio rispetto ai metodi diretti si avrà se con un numero di iterazioni sufficientemente piccolo (diciamo molto minore di n) si raggiunge la precisione desiderata.

La storia dei metodi iterativi per sistemi lineari è molto lunga e risale al diciannovesimo secolo: nel 1845 e nel 1874 furono pubblicati per la prima volta, nell'ambito di particolari applicazioni, due procedimenti oggi noti come metodi di Jacobi e di Gauss-Seidel e nel 1847 il metodo del gradiente di

[7] Una matrice si dice strutturata quando presenta molti elementi uguali a zero e questi elementi riempiono una parte significativa della matrice. Esempi già incontrati in questo libro sono le matrici diagonali o quelle triangolari. Altri semplici casi che si trovano spesso nelle applicazioni sono le matrici tridiagonali, nelle quali gli elementi diversi da zero sono soltanto sulla diagonale principale e sulle due parallele superiore e inferiore ($a_{ij} = 0$ per $|i - j| > 1$) e più in generale le matrici a banda ($a_{ij} = 0$ per $|i - j| > q$, con $q << n$).

Cauchy per sistemi con matrice dei coefficienti simmetrica definita positiva. Nei primi decenni del Novecento, in particolare negli anni '40-'50, fu molto intensa la ricerca sui metodi per quest'ultima classe di problemi, a cui noi dedicheremo il paragrafo 4.7. A partire da questi studi, intorno agli anni '80 ha ripreso nuovo vigore lo studio di metodi iterativi per sistemi lineari non simmetrici, portando alla definizione di diversi metodi che oggi sono ampiamente utilizzati. In questo paragrafo ci occuperemo dei metodi storici, quelli di Jacobi [8] e Gauss-Seidel [9], tralasciando quelli più moderni la cui comprensione richiederebbe conoscenze di algebra lineare troppo approfondite per gli studenti a cui questo libro è rivolto. Il lettore interessato può consultare comunque la bibliografia citata nel paragrafo introduttivo 4.1. I metodi di Jacobi e Gauss-Seidel partono da una riscrittura del sistema lineare

$$Ax = b, \tag{4.25}$$

in un problema equivalente della forma

$$x = Mx + d, \tag{4.26}$$

con M matrice $n \times n$ e d vettore in $\mathbb{R}^n$. Ad esempio, e questa non è l'unica possibilità, (4.25) è equivalente a $x = (I - A)x + b$. Una volta che il sistema è stato portato nella forma (4.26), si può pensare al seguente procedimento iterativo per risolverlo: data un'approssimazione iniziale $x^{(0)}$ della soluzione, si genera una successione $x^{(1)}, x^{(2)}, \ldots$ di vettori con la formula:

$$x^{(k)} = Mx^{(k-1)} + d. \tag{4.27}$$

Ovviamente dovremo assicurarci che la successione sia convergente e converga proprio alla soluzione di (4.25). Al variare della riformulazione (4.26), ovvero al variare di M e d, si ottengono metodi diversi che vengono comunque chiamati metodi iterativi stazionari perché M e d dipendono esclusivamente dai dati A e b e non cambiano al variare di k. La matrice M è chiamata matrice di iterazione. I due più semplici metodi appartenenti a questa classe sono il metodo di Jacobi e il metodo di Gauss-Seidel.

[8] Carl Gustav Jacobi (1804-1851)

[9] L'idea di questo metodo comparve per la prima volta in un lavoro di Gauss nell'ambito di una applicazione specifica. Successivamente, Jacobi dette da studiare il procedimento al suo allievo Philipp Ludwig Ritter von Seidel (1821-1896), che lo enunciò in una forma più generale. Da qui nasce il doppio nome con cui il metodo è oggi ricordato.

4.6.1 Metodo di Jacobi

Consideriamo la matrice A scomposta nella forma $A = D + E$, dove D è la parte diagonale di A ed E la parte extra-diagonale:

$$D = \begin{pmatrix} a_{11} & 0 & \dots & 0 \\ 0 & a_{22} & \dots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & a_{nn} \end{pmatrix}, \quad E = \begin{pmatrix} 0 & a_{12} & \dots & a_{1n} \\ a_{21} & 0 & \dots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \dots & 0 \end{pmatrix}.$$

Il sistema diventa allora $(D+E)x = b$, ovvero $Dx = -Ex+b$. Supponendo che gli elementi diagonali di A siano tutti diversi da zero [10] , la matrice D è invertibile e quindi il sistema viene riscritto come

$$x = -D^{-1}Ex + D^{-1}b. \tag{4.28}$$

Ci siamo così ricondotti alla forma (4.26), con

$$M = M_J = -D^{-1}E \text{ e } d = D^{-1}b.$$

Siccome

$$D^{-1} = \begin{pmatrix} a_{11}^{-1} & 0 & \dots & 0 \\ 0 & a_{22}^{-1} & \dots & 0 \\ \vdots & \vdots & \vdots & \\ 0 & 0 & \dots & a_{nn}^{-1} \end{pmatrix},$$

non è difficile vedere che il passaggio dalla forma (4.25) alla forma (4.28) equivale semplicemente a riscrivere il sistema esplicitando l'incognita x_1 dalla prima equazione, x_2 dalla seconda e così via fino ad esplicitare x_n dall'ultima equazione. Più precisamente il sistema originale diventa

$$\begin{cases} x_1 = \frac{b_1-(a_{12}x_2+a_{13}x_3+\dots+a_{1n}x_n)}{a_{11}} \\ x_2 = \frac{b_2-(a_{21}x_1+a_{23}x_3+\dots+a_{2n}x_n)}{a_{22}} \\ \vdots \\ x_i = \frac{b_i-(a_{i1}x_1+\dots+a_{i,i-1}x_{i-1}+a_{i,i+1}x_{i+1}+\dots+a_{in}x_n)}{a_{ii}} \\ \vdots \\ x_n = \frac{b_n-(a_{n1}x_1+a_{n2}x_2+a_{n3}x_3+\dots+a_{n,n-1}x_{n-1}}{a_{nn}} \end{cases}$$

[10] Se, come stiamo supponendo, $det(A) \neq 0$, questa non è un'ipotesi restrittiva. Infatti, anche se qualche elemento diagonale fosse uguale a zero, sicuramente potremmo cambiare a priori l'ordine delle equazioni in modo da ricondursi alla situazione voluta. Se così non fosse, vorrebbe dire che la matrice A è singolare.

Una volta che il sistema è stato riscritto, si può usare la formula iterativa (4.27), che, esplicitata, diventa:

$$\begin{cases} x_1^{(k)} = \frac{b_1-(a_{12}x_2^{(k-1)}+a_{13}x_3^{(k-1)}+\ldots+a_{1n}x_n^{(k-1)})}{a_{11}} \\ x_2^{(k)} = \frac{b_2-(a_{21}x_1^{(k-1)}+a_{23}x_3^{(k-1)}+\ldots+a_{2n}x_n^{(k-1)})}{a_{22}} \\ \vdots \\ x_i^{(k)} = \frac{b_i-(a_{i1}x_1^{(k-1)}+\ldots+a_{i,i-1}x_{i-1}^{(k-1)}+a_{i,i+1}x_{i+1}^{(k-1)}+\ldots+a_{in}x_n^{(k-1)})}{a_{ii}} \\ \vdots \\ x_n^{(k)} = \frac{b_n-(a_{n1}x_1^{(k-1)}+a_{n2}x_2^{(k-1)}+\ldots+a_{n,n-1}x_{n-1}^{(k-1)})}{a_{nn}} \end{cases}$$

ovvero

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

Nella formula precedente seguiamo la convenzione in base alla quale una sommatoria nella quale il valore iniziale dell'indice è maggiore del valore finale vale zero.

4.6.2 Metodo di Gauss-Seidel

Nel metodo di Jacobi le componenti del nuovo vettore $x^{(k)}$ non vengono utilizzate fino a quando l'intero vettore non è stato calcolato. In realtà, si può pensare di utilizzarle via via che vengono calcolate nel modo seguente: una volta calcolata $x_1^{(k)}$, la si usa insieme alle vecchie componenti $x_2^{(k-1)}, \ldots, x_n^{(k-1)}$ per produrre $x_2^{(k)}$; una volta che anche $x_2^{(k)}$ è disponibile, il calcolo di $x_3^{(k)}$ avviene sfruttando $x_1^{(k)}, x_2^{(k)}$ e $x_3^{(k-1)}, \ldots, x_n^{(k-1)}$ e così via. In conclusione l'iterata $x^{(k)}$ viene calcolata con le seguenti formule:

$$\begin{cases} x_1^{(k)} = \frac{b_1-(a_{12}x_2^{(k-1)}+a_{13}x_3^{(k-1)}+\ldots+a_{1n}x_n^{(k-1)})}{a_{11}} \\ x_2^{(k)} = \frac{b_2-(a_{21}x_1^{(k)}+a_{23}x_3^{(k-1)}+\ldots+a_{2n}x_n^{(k-1)})}{a_{22}} \\ \vdots \\ x_i^{(k)} = \frac{b_i-(a_{i1}x_1^{(k)}+\ldots+a_{i,i-1}x_{i-1}^{(k)}+a_{i,i+1}x_{i+1}^{(k-1)}+\ldots+a_{in}x_n^{(k-1)})}{a_{ii}} \\ \vdots \\ x_n^{(k)} = \frac{b_n-(a_{n1}x_1^{(k)}+a_{n2}x_2^{(k)}+\ldots+a_{n,n-1}x_{n-1}^{(k)})}{a_{nn}} \end{cases}$$

ovvero

$$x_i^{(k)} = \frac{b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k-1)}}{a_{ii}}, \quad \text{per} \quad i = 1, \ldots, n. \quad (4.29)$$

Abbiamo in questo modo un altro metodo iterativo, il metodo di Gauss-Seidel. Per vedere che esso appartiene alla classe dei metodi stazionari (4.27) scomponiamo la matrice A nella forma $A = L + U$ dove L è la parte triangolare inferiore di A e U la parte strettamente triangolare superiore, cioè:

$$L = \begin{pmatrix} a_{11} & 0 & 0 & \ldots & 0 \\ a_{21} & a_{22} & 0 & \ldots & 0 \\ a_{31} & a_{32} & a_{33} & \ldots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \ldots & a_{nn} \end{pmatrix}, \quad U = \begin{pmatrix} 0 & a_{12} & a_{13} & \ldots & a_{1n} \\ 0 & 0 & a_{23} & \ldots & a_{2n} \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \ldots & a_{n-1,n} \\ 0 & 0 & 0 & \ldots & 0 \end{pmatrix}.$$

Allora il sistema $Ax = b$ è equivalente a

$$Lx = -Ux + b, \quad (4.30)$$

ovvero a un problema dela forma (4.26) con

$$M = M_{GS} = -L^{-1}U \ \text{ e } \ d = L^{-1}b.$$

Consideriamo la relazione (4.30) e impostiamo su questa il procedimento iterativo

$$Lx^{(k)} = -Ux^{(k-1)} + b,$$

nel quale vediamo che la nuova iterata $x^{(k)}$ è soluzione di un sistema triangolare inferiore. Applicando a questo sistema l'algoritmo di sostituzione in avanti (Algoritmo 4.3.2) si vede facilmente che le componenti di $x^{(k)}$ sono date proprio dalla formula (4.29).

4.6.3 Convergenza dei metodi iterativi stazionari

Sia x la soluzione del sistema e indichiamo con $e^{(k)} = x^{(k)} - x$ il vettore errore alla k-esima iterazione. Sottraendo membro a membro le relazioni (4.26) e (4.27) si ottiene

$$e^{(k)} = Me^{(k-1)}, \quad (4.31)$$

ovvero un legame fra l'errore alla k-esima iterazione e quello all'iterazione precedente. Da questa relazione possiamo ricavare informazioni sulle proprietà di convergenza dei metodi iterativi stazionari.

Teorema 4.6.1. Se esiste una norma naturale [11] per la quale $\|M\| < 1$, allora la successione $\{x^{(k)}\}$ generata dal metodo iterativo stazionario (4.27) converge alla soluzione del sistema $Ax = b$ per qualunque scelta dell'approssimazione iniziale $x^{(0)}$.

Dimostrazione. Consideriamo la relazione (4.31). Passando alle norme e sfruttando le proprietà delle norme naturali, si ottiene

$$\|e^{(k)}\| = \|Me^{(k-1)}\| \leq \|M\|\|e^{(k-1)}\|$$

ovvero, ponendo $C = \|M\|$,

$$\|e^{(k)}\| \leq C\|e^{(k-1)}\|.$$

Allora $\|e^{(1)}\| \leq C\|e^{(0)}\|$, $\|e^{(2)}\| \leq C\|e^{(1)}\| \leq C^2\|e^{(0)}\|$ e così via; in generale si ha

$$\|e^{(k)}\| \leq C^k\|e^{(0)}\|.$$

Se usiamo la norma naturale per la quale $C < 1$, passando al limite per $k \to \infty$, si ottiene

$$\lim_{k\to\infty} \|e^{(k)}\| \leq \lim_{k\to\infty} C^k\|e^{(0)}\| = \|e^{(0)}\| \lim_{k\to\infty} C^k = 0,$$

cioè la successione degli errori $\{e^{(k)}\}$ converge a 0 e quindi la successione $\{x^{(k)}\}$ converge alla soluzione del sistema lineare. □

Da questo teorema si deduce il seguente corollario

Corollario 4.6.1. Se la matrice A è a predominanza diagonale per righe, ovvero soddisfa la relazione

$$|a_{ii}| > \sum_{j=1, j\neq i}^{n} |a_{ij}| \qquad \forall i = 1, \ldots, n, \tag{4.32}$$

oppure a predominanza diagonale per colonne [12] , ovvero soddisfa la relazione

$$|a_{jj}| > \sum_{i=1, i\neq j}^{n} |a_{ij}| \qquad \forall j = 1, \ldots, n, \tag{4.33}$$

allora il metodo di Jacobi converge per qualunque scelta dell'approssimazione iniziale $x^{(0)}$

[11] Ad esempio, $\|\cdot\|_1, \|\cdot\|_2$ oppure $\|\cdot\|_\infty$

[12] Alcuni autori indicano le proprietà (4.32) e (4.33) con l'espressione a predominanza diagonale in senso stretto per righe o per colonne, per distinguerla da una proprietà più debole nella quale le disuguaglianze valgono con il $\geq$ e almeno una con il $>$ stretto.

Dimostrazione. La matrice di iterazione M_J per il metodo di Jacobi è $M_J = -D^{-1}E$ e quindi i suoi elementi sono dati da $m_{ij} = -a_{ij}/a_{ii}$ per $i \neq j$ e $m_{ij} = 0$ per $i = j$. Allora

$$\|M_J\|_\infty = \max_{1\leq i\leq n} \sum_{j=1}^{n} |m_{ij}| = \max_{1\leq i\leq n} \sum_{j=1,\, j\neq i}^{n} \frac{|a_{ij}|}{|a_{ii}|} = \max_{1\leq i\leq n} \frac{\sum_{j=1,\, j\neq i}^{n} |a_{ij}|}{|a_{ii}|}$$

e

$$\|M_J\|_1 = \max_{1\leq j\leq n} \sum_{i=1}^{n} |m_{ij}| = \max_{1\leq j\leq n} \sum_{i=1\ i\neq j}^{n} \frac{|a_{ij}|}{|a_{ii}|} = \max_{1\leq j\leq n} \frac{\sum_{i=1\ i\neq j}^{n} |a_{ij}|}{|a_{ii}|}.$$

Pertanto, se vale la (4.32) si ha $\|M_J\|_\infty < 1$ e se vale (4.33) si ha $\|M_J\|_1 < 1$. In entrambi i casi l'ipotesi del Teorema 4.6.1 è soddisfatta. □

Con passaggi più complicati si può dimostrare che (4.32) e (4.33) sono condizioni sufficienti anche per la convergenza globale del metodo di Gauss-Seidel.

Il teorema e il corollario appena dimostrati esprimono condizioni sotto cui i metodi convergono globalmente, ovvero indipendentemente dalla scelta dell'approssimazione iniziale $x^{(0)}$. In questo senso la scelta di $x^{(0)}$ non rappresenta un problema, tanto che in pratica si utilizza spesso $x^{(0)} = 0$. Poiché queste condizioni sono sufficienti per la convergenza, ma non necessarie, i metodi possono convergere anche se esse non sono soddisfatte. Per individuare risultati più generali occorre tirare in ballo gli autovalori della matrice di iterazione M, o meglio il suo raggio spettrale $\rho(M)$, che è definito come il massimo dei moduli degli autovalori di M. In particolare, vale il seguente risultato per la cui dimostrazione rimandiamo a [6] o [20].

Teorema 4.6.2. Un metodo iterativo stazionario converge a partire da qualunque $x^{(0)}$ se e soltanto se $\rho(M) < 1$.

Ricordiamo che il raggio spettrale di una matrice M è l'estremo inferiore dell'insieme delle norme naturali di M e quindi se $\rho(M) < 1$, sicuramente esiste almeno una norma naturale di M minore di 1, ma non è dato sapere quale norma sia.

Consideriamo ora la questione della velocità di convergenza dei metodi di Jacobi e Gauss-Seidel. Sulla base del Teorema 4.6.2 si dimostra [6] che la velocità è strettamente legata a $\rho(M)$: più $\rho(M)$ è vicino a zero, più veloce è la convergenza; viceversa, più $\rho(M)$ è vicino a 1, più lenta è la convergenza. È importante osservare che nelle ipotesi del Teorema 4.6.1 si ha

$$\|e^{(k)}\| \leq \|M\|\|e^{(k-1)}\|;$$

questa relazione esprime il fatto che, nella norma scelta, l'errore diminuisce almeno di un fattore pari a $\|M\|$ ad ogni iterazione e si potrebbe essere tentati di concludere che la velocità con cui le iterate si avvicinano alla soluzione dipende da $\|M\|$. Questa conclusione è falsa, come vedremo in un esempio più avanti.

Concludiamo osservando che le proprietà finora esaminate valgono in precisione infinita. Quando i metodi vengono realizzati in aritmetica floating point, la convergenza può talvolta essere rallentata o addirittura compromessa del tutto da questioni di stabilità. L'analisi della stabilità dei metodi iterativi stazionari è molto complicata e viene portata avanti soltanto per classi particolari di sistemi lineari. Chi fosse interessato a questi aspetti può consultare il libro [22].

4.6.4 Criteri di arresto

La realizzazione pratica dei metodi iterativi stazionari richiede la definizione di opportuni criteri di arresto. Ricordando la discussione svolta nel Capitolo 3 a proposito degli algoritmi per la risoluzione di equazioni non lineari, possiamo ipotizzare in questo nuovo contesto l'uso di diversi criteri.

1) Come criterio di salvaguardia, conviene sempre fissare un numero massimo di iterazioni, per tener conto della possibilità che il metodo non converga e di eventuali scelte errate delle tolleranze.

2) In analogia con il criterio (3.17) si può accettare un'iterata come soluzione quando il vettore residuo è sufficientemente piccolo. Questo criterio viene abitualmente usato nella forma mista

$$\|Ax^{(k)} - b\| \leq \eta_r \|b\| + \eta_a, \tag{4.34}$$

dove la norma scelta può essere una qualunque norma naturale. Ricordiamo che nel contesto della risoluzione di equazioni non lineari, il soddisfacimento del criterio (3.17) da parte di un'iterata non implicava la sua vicinanza alla soluzione e che questa dipendeva dalla pendenza di f. Anche ora possiamo fare un'osservazione analoga e vedere che la vicinanza o meno dell'iterata alla soluzione dipende da $k(A)$. Ricordiamo a questo proposito la relazione (4.13) fra il vettore residuo e l'errore; si vede allora che il criterio (4.34) garantisce che l'errore relativo $\|x^{(k)} - x\|/\|x\|$ è piccolo se $k(A)$ è piccolo. Più precisamente, supponendo per semplicità di porre $\eta_a = 0$, da (4.13) si deduce

$$\frac{\|x^{(k)} - x\|}{\|x\|} \leq k(A)\eta_r.$$

Pertanto la bontà di $x^{(k)}$ come approssimazione della soluzione dipende dalla tolleranza scelta e dal condizionamento del problema.

3) Chiediamoci se in questo contesto sarebbe sensato un criterio di arresto basato sulla vicinanza fra due successive iterate

$$\|x^{(k)} - x^{(k-1)}\| \leq \sigma_r \|x^{(k-1)}\| + \sigma_a. \tag{4.35}$$

Nel Capitolo 3 questo criterio trovava una giustificazione teorica soltanto per metodi a convergenza veloce, almeno superlineare. In generale infatti, non è assolutamente detto che due iterate vicine fra loro siano anche vicine alla soluzione; esse possono essere vicine semplicemente perché il metodo sta procedendo con lentezza. Nel contesto di cui stiamo discutendo ora possiamo fare una analisi rigorosa che, sfruttando implicitamente la linearità del problema, ci porta alla stessa conclusione. A tale scopo supponiamo per semplicità che siano soddisfatte le ipotesi del Teorema 4.6.1; allora, da

$$\begin{aligned} \|x^{(k)} - x^{(k-1)}\| &= \|x^{(k)} - x + x - x^{(k-1)}\| = \\ &= \|e^{(k)} - e^{(k-1)}\| \geq \|e^{(k-1)}\| - \|e^{(k)}\| \geq \\ &\geq \|e^{(k-1)}\| - \|M\|\|e^{(k-1)}\| = (1 - \|M\|)\|e^{(k-1)}\| \end{aligned}$$

e da $\|e^{(k)}\| \leq \|M\|\|e^{(k-1)}\|$ si ottiene la seguente relazione fra $\|x^{(k)} - x^{(k-1)}\|$ e $\|e^{(k)}\|$:

$$\|e^{(k)}\| \leq \frac{\|M\|}{1 - \|M\|} \|x^{(k)} - x^{(k-1)}\|.$$

Supponiamo ora che il criterio (4.35), per semplicità nella forma assoluta con $\sigma_r = 0$, sia soddisfatto. Allora si ha

$$\|e^{(k)}\| \leq \frac{\|M\|}{1 - \|M\|} \sigma_a.$$

Di conseguenza avremo $\|e^{(k)}\| \leq \sigma_a$ se $\|M\| \leq 1/2$; se invece $\|M\| > 1/2$ la distanza di $x^{(k)}$ dalla soluzione può essere maggiore di σ_a e tanto più grande quanto più $\|M\|$ è vicina a 1.

4.6.5 Algoritmi ed esempi

Negli algoritmi di Jacobi e Gauss-Seidel che seguono usiamo il criterio di arresto basato sulla norma del residuo (4.34) e prevediamo un numero massimo K_{max} di iterazioni. Per fissare le idee abbiamo scelto di usare la norma-∞, che può essere calcolata usando l'algoritmo 1.3.3.

Algoritmo 4.6.1. Algoritmo di Jacobi per la risoluzione di $Ax = b$.

Dati: $n,\ A,\ b,\ x^{(0)},\ K_{max},\ \eta_r,\ \eta_a$

Risultati: $esito$ indicatore del risultato:
$esito = 0$ il procedimento è terminato con successo
$esito = 1$ raggiunto il numero massimo di iterazioni
x ultima iterata calcolata
k numero di iterazioni effettuate

1. $x = x^{(0)}$
2. $\gamma = \|b\|_\infty$
3. Per $k = 1, \ldots, K_{max}$
   1. $x^v = x$
   2. Per $i = 1, \ldots, n$
      1. $s = \sum_{j=1\ j\neq i}^{n} a_{ij} x_j^v$
      2. $x_i = (b_i - s)/a_{ii}$

      Fine ciclo su i
   3. $r = Ax - b$
   4. Se $\|r\|_\infty < \eta_r \gamma + \eta_a$, allora:
      $esito = 0$ e fermati

      Fine scelta

   Fine ciclo su k
4. $esito = 1$
5. Fine

L'algoritmo utilizza due vettori x^v e x necessari per la memorizzazione di $x^{(k-1)}$ e $x^{(k)}$ rispettivamente. L'algoritmo di Gauss-Seidel che segue non ha bisogno di due vettori perché usa le nuove componenti di $x^{(k)}$ man mano che le calcola; così ad ogni iterazione il vettore x rappresenta la nuova iterata $x^{(k)}$ nelle componenti già aggiornate e la vecchia iterata $x^{(k-1)}$ in quelle ancora non aggiornate.

Algoritmo 4.6.2. Algoritmo di Gauss-Seidel per la risoluzione di $Ax = b$.

Dati: $n,\ A,\ b,\ x^{(0)},\ K_{max},\ \eta_r,\ \eta_a$;

Risultati: $esito$ indicatore del risultato:
$esito = 0$ il procedimento è terminato con successo
$esito = 1$ raggiunto il numero massimo di iterazioni
x ultima iterata calcolata
k numero di iterazioni effettuate

1. $x = x^{(0)}$
2. $\gamma = \|b\|_\infty$
3. Per $k = 1, \ldots, K_{max}$
   1. Per $i = 1, \ldots, n$
      1. $s = \sum_{j=1\ j\neq i}^{n} a_{ij}x_j$
      2. $x_i = (b_i - s)/a_{ii}$

      Fine ciclo su i
   2. $r = Ax - b$
   3. Se $\|r\|_\infty < \eta_r\gamma + \eta_a$, allora:
      $esito = 0$ e fermati

      Fine scelta

   Fine ciclo su k
4. $esito = 1$
5. Fine

Le condizioni di convergenza dei Teoremi 4.6.1 e 4.6.2 riguardano la matrice di iterazione M. Poiché questa dipende dal metodo e, a parità di matrice A, le due matrici M_J e M_{GS} sono in generale diverse, può darsi che per un dato sistema lineare il metodo di Jacobi converga e quello di Gauss-Seidel no, o viceversa, oppure che entrambi i metodi convergano o nessuno dei due converga. L'esperienza dice che quando entrambi i metodi convergono, quello di Gauss-Seidel risulta spesso più veloce di quello di Jacobi; questo corrisponde all'idea intuitiva che il metodo di Gauss-Seidel, utilizzando le componenti della nuova iterata man mano che le calcola, sia in un certo senso una versione "accelerata"del metodo di Jacobi. D'altra parte, non mancano esempi del contrario, in cui il metodo di Jacobi risulta più veloce.

Esempio 4.6.1. Consideriamo il sistema lineare $Ax = b$ con

$$A = \begin{pmatrix} 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & \ldots & 0 \\ -1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & \ldots & 0 \\ 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & \ldots & 0 \\ 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & \ldots & 0 \\ -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 & \ldots & 0 \\ 0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 \end{pmatrix}$$

e $b = (2, 1, \ldots, 1, 2)^T$, la cui soluzione è $x = (1, 1, \ldots, 1)^T$. Per $n \leq 9$ la matrice A è a predominanza diagonale (sia per righe che per colonne, essendo simmetrica) e la convergenza di entrambi i metodi è assicurata dal Corollario 4.6.1. Per $n \geq 10$, questa proprietà non vale più; nondimeno entrambi i metodi convergono perché i raggi spettrali $\rho(M_J)$ e $\rho(M_{GS})$ sono minori di 1. Nella tabella 4.9 riportiamo per diversi valori di n:
– il numero di condizionamento $k_\infty(A)$,

– il numero di iterazioni K_J e K_{GS} necessarie ai due metodi per soddisfare il criterio di arresto con $\eta_r = \eta_a = 10^{-8}$,
– i raggi spettrali $\rho_J = \rho(M_J)$ e $\rho_{GS} = \rho(M_{GS})$,
– gli errori rispetto alla soluzione esatta e_J e e_{GS} calcolati in norma infinito.

n	$k_\infty(A)$	K_J	K_{GS}	ρ_J	ρ_{GS}	e_J	e_{GS}
5	3.0e+00	27	16	0.5	0.2868	7.5e−09	4.2e−09
10	1.1e+01	72	38	0.7784	0.6166	1.8e−08	1.5e−08
20	3.2e+01	212	108	0.9236	0.8545	6.1e−08	5.6e−08
30	6.5e+01	418	211	0.9622	0.9263	1.3e−07	1.2e−07
50	1.7e+02	1016	510	0.9852	0.9707	3.3e−07	3.3e−07
100	6.3e+02	3501	1753	0.9961	0.9921	1.2e−06	1.2e−06
150	1.4e+03	7280	3642	0.9982	0.9964	2.8e−06	2.8e−06

Tabella 4.9 Esempio 4.6.1: metodi di Jacobi e Gauss-Seidel

Vediamo che per ogni valore di n il metodo di Gauss-Seidel impiega un minor numero di iterazioni per soddisfare il criterio di arresto; questo è coerente con il fatto che ρ_{GS} è sempre minore di ρ_J. Per entrambi i metodi la velocità di convergenza si deteriora al crescere di n perché il raggio spettrale della matrice di iterazione si avvicina sempre di più a 1. Osserviamo anche che gli errori tendono a crescere al crescere di n, laddove (in virtù del criterio di arresto utilizzato e del fatto che $\|b\|_\infty = 2$ per ogni n) i residui hanno tutti norma minore o uguale a 2×10^{-8}: questo comportamento si spiega con il fatto che $k_\infty(A)$ cresce con n. ■

Esempio 4.6.2. I risultati della tabella 4.10 si riferiscono al sistema $Ax = b$ con

$$A = \begin{pmatrix} 11 & -5 & -5 \\ 5 & 12 & 6 \\ 6 & -4 & 11 \end{pmatrix} \quad \text{e} \quad b = \begin{pmatrix} 1 \\ 23 \\ 13 \end{pmatrix},$$

la cui soluzione è $x = (1, 1, 1)^T$.

η	K_J	K_{GS}	e_J	e_{GS}
1e−05	49	65	1.4e−05	9.8e−06
1e−08	78	103	1.6e−08	1.1e−08
1e−10	98	129	1.5e−10	1.1e−10
1e−12	118	155	1.4e−12	1.0e−12

Tabella 4.10 Esempio 4.6.2: metodi di Jacobi e Gauss-Seidel

Essendo A a predominanza diagonale per righe, entrambi i metodi di Jacobi e Gauss-Seidel convergono. Poiché $\|M_J\|_\infty = 0.9167$ e $\|M_{GS}\|_\infty = 0.9091$, si potrebbe pensare in base al Teorema 4.6.1 che la convergenza del metodo di Gauss-Seidel fosse più rapida di quella del metodo di Jacobi. In realtà le cose vanno nel modo opposto perché $\rho(M_J) = 0.7918$ e $\rho(M_{GS}) = 0.8363$.

Nella tabella riportiamo per diversi valori di η il numero di iterazioni K_J e K_{GS} necessarie ai due metodi per soddisfare il criterio di arresto con $\eta_a = \eta_r = \eta$ e gli errori rispetto alla soluzione esatta. ■

4.7 Metodi per sistemi con matrice dei coefficienti simmetrica definita positiva (*)

In molte applicazioni capita di dover risolvere sistemi lineari in cui la matrice dei coefficienti è simmetrica ($A = A^T$) e definita positiva ($x^T Ax > 0$ per qualunque vettore x diverso da zero). Si può dimostrare, e questa è una proprietà che useremo nel seguito, che per le matrici definite positive i pivots del metodo di Gauss $a_{kk}^{(k-1)}$ sono tutti positivi [22].

Per risolvere sistemi di questo tipo si possono ovviamente utilizzare i metodi visti finora, sia diretti che iterativi. Ad esempio, si può dimostrare che il metodo di Gauss-Seidel converge globalmente se A è simmetrica definita positiva [6]. D'altra parte esistono metodi ad-hoc che, sfruttando le caratteristiche del problema, risultano più efficienti di quelli di carattere generale.

4.7.1 Metodo di Cholesky

Il metodo diretto più utilizzato per risolvere sistemi lineari con matrice dei coefficienti simmetrica definita positiva è il metodo di Cholesky [13] che richiede la metà delle operazioni del metodo di Gauss.

L'idea alla base di questo metodo è quella di fattorizzare la matrice A nella forma

$$A = R^T R \tag{4.36}$$

con R triangolare superiore. Data questa fattorizzazione, la risoluzione di $Ax = b$ equivale alla risoluzione in cascata dei due sistemi triangolari

$$R^T y = b$$

e

$$Rx = y.$$

Il fattore di Cholesky R non è univocamente determinato: ad esempio, se (4.36) vale per una certa R, allora vale anche per $-R$. L'unicità può essere ottenuta imponendo qualche ulteriore condizione; quella abitualmente considerata è che gli elementi diagonali r_{ii} siano tutti positivi. La matrice R può essere calcolata direttamente procedendo per righe o per colonne.

[13] André Luis Cholesky visse dal 1875 al 1918. Il suo procedimento fu pubblicato postumo nel 1924 e divenne oggetto di studio e di notorietà soltanto alla fine degli anni '40.

Accenniamo a come procede il calcolo per colonne. Il calcolo stesso dimostra che la fattorizzazione (4.36) esiste. Scriviamo per esteso l'uguaglianza $A = R^T R$ tenendo conto della simmetria di A:

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{12} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ a_{1n} & a_{2n} & \dots & a_{nn} \end{pmatrix} = \begin{pmatrix} r_{11} & 0 & \dots & 0 \\ r_{12} & r_{22} & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ r_{1n} & r_{2n} & \dots & r_{nn} \end{pmatrix} \begin{pmatrix} r_{11} & r_{12} & \dots & r_{1n} \\ 0 & r_{22} & \dots & r_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & r_{nn} \end{pmatrix}.$$

L'elemento r_{11} soddisfa la relazione $a_{11} = r_{11}^2$, che ha senso poiché $a_{11} > 0$, essendo il primo pivot che si incontra nel metodo di Gauss. Si ottiene così

$$r_{11} = \pm\sqrt{a_{11}}$$

e scegliamo il segno $+$. Passiamo alla seconda colonna. Da $a_{12} = r_{11} r_{12}$ ricaviamo

$$r_{12} = \frac{a_{12}}{r_{11}}$$

e, successivamente, da $a_{22} = r_{12}^2 + r_{22}^2$ possiamo calcolare

$$r_{22} = \pm\sqrt{a_{22} - r_{12}^2}.$$

Notiamo che $a_{22} - r_{12}^2$ è positivo perché coincide con l'elemento pivot $a_{22}^{(1)}$ che si otterrebbe facendo un passo del metodo di Gauss. Proseguendo in questo modo possiamo calcolare tutti gli elementi di R. Fissato un indice di colonna j si ricavano uno dopo l'altro gli elementi extradiagonali $r_{1j}, r_{2j}, \dots, r_{j-1,j}$ dalle relazioni $a_{ij} = \sum_{k=1}^{i} r_{ki} r_{kj}$, ottenendo

$$r_{ij} = \frac{a_{ij} - \sum_{k=1}^{i-1} r_{ki} r_{kj}}{r_{ii}};$$

una volta calcolati gli elementi extradiagonali possiamo ottenere r_{jj} da

$$r_{jj} = \sqrt{a_{jj} - \sum_{k=1}^{j-1} r_{kj}^2}. \tag{4.37}$$

Il radicando in (4.37) è sicuramente positivo per ogni $j = 2, \dots, n$ in virtù della relazione che sussiste fra queste espressioni e i pivots del metodo di Gauss [22].

La costruzione ora tratteggiata è sintetizzata nell'algoritmo 4.7.1 che segue. Come si può vedere, l'algoritmo prevede la possibilità che il metodo non arrivi a calcolare la matrice R se, per qualche j, l'espressione sotto radice in (4.37) non è positiva; più precisamente, se il radicando è negativo

non è possibile calcolare r_{jj}, mentre se è uguale a zero ci troveremmo a fare una divisione per zero al passo successivo, a meno che non sia $j = n$. Come abbiamo detto, in teoria questa situazione di errore non può avverarsi se A è definita positiva [22]. D'altra parte non esistono modi pratici per controllare a priori se la matrice data soddisfa questa ipotesi. Da questo punto di vista, il metodo di Cholesky può essere visto come uno strumento algoritmico per verificare se una data matrice simmetrica è definita positiva oppure no.

Algoritmo 4.7.1. Algoritmo per calcolare la fattorizzazione di Cholesky di una matrice A simmetrica definita positiva.
Dati: n, A
Risultati: *esito* indicatore del risultato:
esito $= 0$ il procedimento è terminato con successo
esito $= 1$ il procedimento si è interrotto perchè un radicando è non positivo
R fattore di Cholesky

1. *esito* $= 0$
2. $r_{11} = \sqrt{a_{11}}$
3. Per $j = 2, \ldots, n$
   1. Per $i = 1, \ldots, j-1$
      $r_{ij} = \frac{a_{ij} - \sum_{k=1}^{i-1} r_{ki} r_{kj}}{r_{ii}}$
      Fine ciclo su i
   2. $s = a_{jj} - \sum_{k=1}^{j-1} r_{kj}^2$
   3. Se $s \leq 0$, allora:
      esito $= 1$ e fermati
      Fine scelta
   4. $r_{jj} = \sqrt{s}$

   Fine ciclo su j
4. Fine

Esempio 4.7.1. Consideriamo la matrice 3×3

$$A = \begin{pmatrix} 0.01 & 0.003 & 0.004 \\ 0.003 & 0.0409 & 0.0112 \\ 0.004 & 0.0112 & 0.0941 \end{pmatrix}.$$

L'algoritmo precedente, realizzato in doppia precisione, calcola la fattorizzazione di A con

$$R = \begin{pmatrix} 0.1 & 0.03 & 0.04 \\ 0 & 0.2 & 0.05 \\ 0 & 0 & 0.3 \end{pmatrix}.$$

Usando questa fattorizzazione per risolvere il sistema $Ax = b$, con b scelto in modo che la soluzione sia $x = (1, 1, 1)^T$, si ottiene una soluzione esatta nei limiti della precisione di macchina. Se modifichiamo leggermente la matrice A cambiando l'elemento a_{33} da 0.0941 a 0.0041 si ottiene una matrice

singolare. L'algoritmo di Cholesky calcola la fattorizzazione con R uguale a prima salvo che ora $r_{33} = 0$. ■

Uno degli aspetti positivi del metodo di Cholesky è la sua stabilità. Si può infatti dimostrare [22] che l'errore commesso con il metodo di Cholesky in precisione finita soddisfa una relazione analoga a quella già vista per il metodo di Gauss con pivoting parziale e per il metodo di Householder:

$$\frac{\|x - \tilde{x}\|}{\|\tilde{x}\|} \leq C_n k(A) \epsilon_m, \tag{4.38}$$

dove C_n non può assumere valori troppo grandi. La stabilità di questo metodo, come già abbiamo visto per il metodo di Householder, dipende dal fatto che non sono possibili fenomeni di crescita. Infatti dalla relazione (4.37) si deduce

$$a_{jj} = \sum_{k=1}^{j} r_{kj}^2$$

e quindi

$$|r_{kj}| \leq \sqrt{a_{jj}}$$

per ogni coppia di indici k, j. È allora evidente che gli elementi della matrice R non possono crescere significativamente rispetto a quelli della matrice A.

Quanto detto implica che il metodo di Cholesky è stabile, purchè, ovviamente, riesca a calcolare R. Un risultato di Wilkinson assicura che il successo è garantito se A è sufficientemente lontana dall'essere "numericamente" singolare, ovvero se non è troppo mal condizionata rispetto alla precisione di macchina [14]. Consideriamo ad esempio le matrici di Hilbert che sono simmetriche e definite positive qualunque sia la dimensione. Se risolviamo in doppia precisione con il metodo di Cholesky i sistemi lineari dell' esempio (4.4.3) con $n = 4$ e $n = 8$ otteniamo gli stessi risultati che con il metodo di Gauss con pivoting parziale. Se però proviamo a calcolare la fattorizzazione di Cholesky della matrice di Hilbert di dimensione $n = 20$, l'algoritmo si arresta al calcolo di $r_{14,14}$ perché il radicando risulta negativo. Notiamo che il numero di condizionamento in questo caso è dell'ordine di 10^{18}, a fronte di una precisione di macchina $\epsilon_m \simeq 10^{-16}$.

4.7.2 Metodo del gradiente coniugato

Per sistemi definiti positivi di grande dimensione il metodo di Cholesky, in quanto metodo diretto, può risultare troppo costoso e quindi inefficiente. Occorre allora individuare metodi iterativi che sfruttino le proprietà della

[14] Per l'esattezza, se $20n^{\frac{3}{2}} k_2(A)\epsilon_m \leq 1$, ma nella pratica questo limite risulta un po' pessimistico.

matrice A. Il metodo del gradiente coniugato rientra in questa classe. Esso fu pubblicato nel 1952 da Hestenes e Stiefel [21] e molti anni dopo è stato riletto come membro di una più ampia classe di metodi, noti come metodi di Krylov ([24], [28]). Qui utilizziamo la descrizione originale di Hestenes e Stiefel. Vedremo che questo metodo è in teoria un metodo diretto perché si dimostra che la soluzione viene raggiunta dopo al più n iterazioni, dove n è come al solito la dimensione del problema. Molto spesso però, quando n è grande, il numero di iterazioni necessarie per soddisfare richieste di accuratezza ragionevoli (rispetto alla precisione di macchina) è molto minore di n. Questo fatto, insieme alla considerazione che l'uso dell'aritmetica floating point impedisce comunque di arrivare alla soluzione esatta, fanno sì che il metodo venga utilizzato come metodo iterativo.

Per la descrizione del metodo dovremo fare ricorso a definizioni e concetti relativi a funzioni da $\mathbb{R}^n$ in $\mathbb{R}$ (dette spesso "funzionali"), per le quali rimandiamo ad esempio a [3]. Il punto di partenza è il legame che sussiste fra il sistema lineare $Ax = b$ e il funzionale quadratico f definito da

$$f(x) = \frac{1}{2}x^T Ax - b^T x, \tag{4.39}$$

il cui gradiente è $Ax - b$ e la cui hessiana è A stessa. Allora risolvere il sistema lineare $Ax = b$ equivale a trovare un punto critico (o stazionario) di f. D'altra parte se, come stiamo supponendo, A è definita positiva, il punto stazionario è unico e rappresenta un punto di minimo per f. Possiamo allora pensare di calcolare la soluzione di $Ax = b$ con un metodo iterativo finalizzato alla minimizzazione di f. Un procedimento di questo tipo parte da un punto iniziale x_0 e calcola la successione $\{x_k\}$ mediante una formula iterativa della forma [15]:

$$x_{k+1} = x_k + \alpha_k d_k,$$

dove $d_k \in \mathbb{R}^n$ rappresenta la direzione di movimento a partire da x_k e il parametro $\alpha_k \in \mathbb{R}$ serve per decidere di quanto muoversi lungo d_k. Per scegliere d_k e α_k ragioniamo nel modo seguente. Data la (4.39), per qualsiasi $d \in \mathbb{R}^n$ e $\alpha \in \mathbb{R}$ si ha

$$\begin{array}{rcl} f(x_k + \alpha d) & = & \frac{1}{2}(x_k + \alpha d)^T A(x_k + \alpha d) - b^T(x_k + \alpha d) \\ & = & \frac{1}{2}x_k^T Ax_k - b^T x_k + \frac{1}{2}\alpha^2 d^T Ad + \alpha d^T Ax_k - \alpha b^T d \\ & = & f(x_k) + \frac{1}{2}\alpha^2 d^T Ad + \alpha(Ax_k - b)^T d \\ & = & f(x_k) + \frac{1}{2}\alpha^2 d^T Ad + \alpha r_k^T d, \end{array} \tag{4.40}$$

dove abbiamo indicato con r_k il vettore residuo $Ax_k - b$. Supponiamo per il momento che sia fissata la direzione $d = d_k$. Possiamo osservare che

[15] Poiché in questo paragrafo non avremo bisogno di far riferimento alle singole componenti delle iterate, non usiamo la notazione $x^{(k)}$ utilizzata nella descrizione dei metodi iterativi stazionari e torniamo alla notazione più semplice x_k.

$f(x_k + \alpha d_k)$, vista come funzione di α, è una funzione di secondo grado con derivata prima uguale a $d_k^T A d_k \alpha + r_k^T d_k$ e derivata seconda uguale a $d_k^T A d_k > 0$. Definiamo allora α_k come il valore di α che minimizza $f(x_k + \alpha d_k)$. A questo scopo risolviamo l'equazione $d_k^T A d_k \alpha + r_k^T d_k = 0$ e otteniamo

$$\alpha_k = -\frac{r_k^T d_k}{d_k^T A d_k}. \tag{4.41}$$

Se d_k è tale che

$$r_k^T d_k < 0, \tag{4.42}$$

avremo $\alpha_k > 0$ e, come si può facilmente verificare, $f(x_k + \alpha_k d_k) < f(x_k)$. I vettori d_k che soddisfano la relazione (4.42) sono detti direzioni di discesa per la f in x_k perché $r_k^T d_k$ è la derivata di $f(x_k + \alpha d_k)$ in $\alpha = 0$ e quindi se $r_k^T d_k < 0$ la funzione f risulta decrescente partendo da x_k in direzione d_k. Le direzioni di discesa sono tutti e soli i vettori che formano un angolo fra 0 (incluso) e $\pi/2$ (escluso) con il vettore $-r_k$, detto antigradiente di f in x_k. La scelta più semplice per la direzione di discesa è $d_k = -r_k$, che dà origine al metodo del gradiente, detto anche metodo di Cauchy [16]. Le proprietà di convergenza di questo metodo sono riassunte nella seguente relazione, per la cui dimostrazione rimandiamo a [6]:

$$0 \leq f(x_k) - f(x^*) \leq \left(\frac{k_2(A) - 1}{k_2(A) + 1}\right)^{2k} (f(x_0) - f(x^*)), \tag{4.43}$$

dove x^* è la soluzione del sistema lineare. Questa relazione dice che $f(x_k)$ tende a $f(x^*)$ per $k \to \infty$, il che implica che x_k tende a x^* dal momento che f è un funzionale continuo.

Dalla relazione (4.43) si deduce che il metodo del gradiente raggiunge la soluzione in una sola iterazione quando $k_2(A) = 1$, mentre la convergenza può essere molto lenta quando $k_2(A) >> 1$, ovvero quando il sistema è molto mal condizionato. Questa lentezza è legata al fatto che, per sua definizione, il metodo del gradiente procede a zig-zag, nel senso che ogni direzione di movimento è ortogonale a quella precedente (cfr. figura 4.2); quanto più il problema è mal condizionato, tanto più questo procedere a zig-zag impone passi piccoli [28]. Partendo da questa osservazione, molti autori hanno cercato modi diversi di scegliere le direzioni di discesa allo scopo di accelerare la convergenza rispetto al metodo del gradiente.

Il metodo del gradiente coniugato si pone ad ogni iterazione il seguente obiettivo: determinare d_k e α_k in modo da minimizzare $f(x_k + \alpha d)$ contemporaneamente rispetto a d e ad α. Per capire se e come è possibile

[16] Il metodo del gradiente fu pubblicato da Augustin Louis Cauchy (1789 - 1857) nel 1847. Esso è detto anche metodo di più ripida discesa perché l'antigradiente $-r_k$ è, fra tutte le direzioni di discesa, quella a cui corrisponde il più piccolo valore di $r_k^T d_k$.

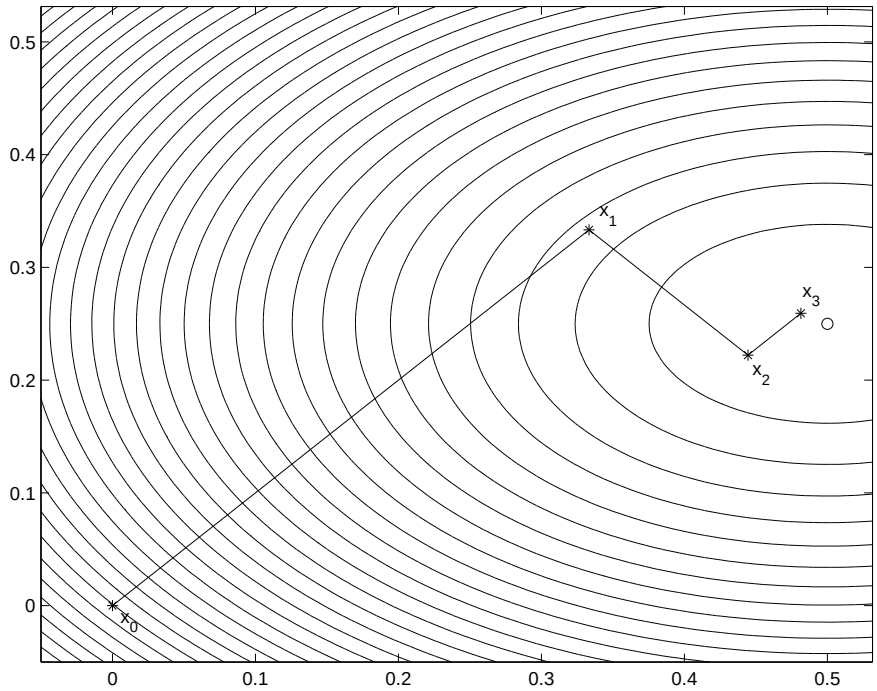

Figura 4.2 Metodo del gradiente

raggiungere questo scopo, teniamo conto del fatto che

$$\begin{aligned} x_k &= x_{k-1} + \alpha_{k-1} d_{k-1} \\ &= x_{k-2} + \alpha_{k-2} d_{k-2} + \alpha_{k-1} d_{k-1} = \dots \\ &= x_0 + \alpha_0 d_0 + \alpha_1 d_1 + \dots + \alpha_{k-1} d_{k-1} \\ &= x_0 + D_{k-1} y, \end{aligned}$$

dove la matrice $D_{k-1} \in \mathbb{R}^{n \times k}$ ha per colonne le direzioni d_0, d_1, ..., d_{k-1} e il vettore $y \in \mathbb{R}^k$ ha come elementi $\alpha_0, \alpha_1, \dots, \alpha_{k-1}$. Allora, ricordando che

$$f(x_k + \alpha d) = f(x_k) + \alpha x_k^T A d + \frac{1}{2}\alpha^2 d^T A d - \alpha b^T d,$$

possiamo scrivere

$$\begin{aligned} f(x_k + \alpha d) &= f(x_k) + \alpha x_0^T A d + \alpha y^T D_{k-1}^T A d + \tfrac{1}{2}\alpha^2 d^T A d - \alpha b^T d \\ &= f(x_k) + \alpha y^T D_{k-1}^T A d + \tfrac{1}{2}\alpha^2 d^T A d + \alpha r_0^T d. \end{aligned}$$

Una strada per definire d_k e α_k può allora essere la seguente: si sceglie d_k in modo che sia

$$d_j^T A d_k = 0 \quad \text{per} \;\; j = 0, \dots, k-1 \tag{4.44}$$

e quindi $y^T D_{k-1}^T A d_k = 0$ e α_k in modo da rendere $\frac{1}{2}\alpha^2 d_k^T A d_k + \alpha r_0^T d_k$ più piccolo possibile, cioè

$$\alpha_k = -\frac{r_0^T d_k}{d_k^T A d_k}. \tag{4.45}$$

Osserviamo subito che questa formula per α_k non è altro che la (4.41); infatti in virtù della (4.44) si ha:

$$\begin{aligned} r_0^T d_k &= (Ax_0 - b)^T d_k = (Ax_0 + AD_{k-1}y - AD_{k-1}y - b)^T d_k \\ &= (Ax_k - b)^T d_k + y^T D_{k-1}^T A d_k = r_k^T d_k. \end{aligned}$$

Un insieme di vettori che soddisfa la (4.44) si dice costituito da vettori A-coniugati.

Il metodo del gradiente coniugato è uno (il più importante) dei metodi che utilizzano direzioni di movimento A-coniugate. In particolare, le direzioni sono scelte nel modo seguente:

$$\begin{cases} d_0 = -r_0 \\ d_k = -r_k + \beta_k d_{k-1}, \quad k = 1, 2, \ldots, \end{cases} \tag{4.46}$$

dove β_k è uno scalare scelto proprio in modo da garantire la (4.44), ossia

$$\beta_k = \frac{d_{k-1} A r_k}{d_{k-1} A d_{k-1}}. \tag{4.47}$$

Un'importante proprietà che discende dalla (4.44) è l'indipendenza lineare dei vettori A-coniugati. Possiamo dimostrare questo fatto nel seguente modo. Supponiamo che una combinazione lineare $z = \sum_{i=0}^{k} c_i d_i$ di $d_0, d_1, \ldots, d_k$ sia uguale a 0. Allora, per ogni $j = 0, \ldots, k$ si ha

$$0 = d_j^T A z = \sum_{i=0}^{k} c_i d_j^T A d_i = c_j d_j^T A d_j;$$

essendo $d_j^T A d_j > 0$, deve essere $c_j = 0$. In altri termini, l'unica combinazione lineare di $d_0, \ldots, d_k$ uguale a 0 è quella ottenuta con coefficienti tutti uguali a 0, ovvero $d_0, \ldots, d_k$ sono linearmente indipendenti. Da questa proprietà segue che il metodo del gradiente coniugato è teoricamente un metodo diretto, che arriva alla soluzione al massimo in n iterazioni. Infatti, poiché le direzioni sono vettori in $\mathbb{R}^n$ e $(n+1)$ vettori in $\mathbb{R}^n$ sono necessariamente linearmente dipendenti, il procedimento può generarne al più n e si interrompe necessariamente (se non è successo prima) quando ha calcolato x_n. Ci chiediamo allora se x_n ha qualche legame con la soluzione x^* del sistema. La risposta è che x_n è la soluzione. Consideriamo infatti

una qualunque direzione d_j, con $j \in \{0, \ldots, n-1\}$; a causa della (4.44), della (4.45) e della (4.46), si ha:

$$\begin{aligned} d_j^T r_n &= d_j^T (Ax_n - b) \\ &= d_j^T (Ax_0 + \alpha_0 Ad_0 + \ldots + \alpha_{n-1} Ad_{n-1} - b) \\ &= d_j^T (Ax_0 - b) + \alpha_j d_j^T Ad_j \\ &= d_j^T r_0 + \alpha_j d_j^T Ad_j = 0. \end{aligned}$$

In altri termini, r_n è ortogonale a tutti gli elementi di una base di $\mathbb{R}^n$ e quindi è il vettore nullo.

Abbiamo quindi dedotto che il metodo del gradiente coniugato ha la cosiddetta proprietà della terminazione finita: entro n iterazioni esso calcola la soluzione del sistema. In realtà, come abbiamo già avuto modo di osservare, si preferisce usarlo come un metodo iterativo, arrestandolo quando è soddisfatto un criterio di arresto adeguato (di solito, il criterio (4.34) sul vettore residuo).

Nel momento in cui il metodo viene visto come metodo iterativo, ci si pone il problema della velocità con cui ci si avvicina alla soluzione e in particolare, ci chiediamo se effettivamente esso è più veloce del metodo del gradiente. La risposta è affermativa poiché si può dimostrare [28] che

$$0 \leq f(x_k) - f(x^*) \leq \left(\frac{\sqrt{k_2(A)} - 1}{\sqrt{k_2(A)} + 1} \right)^{2k} (f(x_0) - f(x^*)),$$

ossia la velocità del metodo non dipende da $k_2(A)$ come succede per il metodo del gradiente (cfr. (4.43)) ma da $\sqrt{k_2(A)}$: quanto più il sistema è mal condizionato, tanto maggiore è lo scarto fra $\sqrt{k_2(A)}$ e $k_2(A)$, ovvero lo scarto fra le velocità dei due metodi.

L'algoritmo 4.7.2 realizza il metodo del gradiente coniugato. In questo algoritmo non usiamo le formule (4.41) e (4.47), ma le seguenti formule alternative ad esse equivalenti:

$$\alpha_k = \frac{r_k^T r_k}{d_k^T Ad_k} \quad \text{e} \quad \beta_k = \frac{r_k^T r_k}{r_{k-1}^T r_{k-1}}.$$

Poiché il calcolo di β_k richiede l'uso di due vettori residui consecutivi, nell'algoritmo utilizziamo i vettori r (per r_k) e r_v (per r_{k-1}). Inoltre, i vettori residui vengono aggiornati con la formula ricorsiva

$$r_k = r_{k-1} + \alpha_k Ad_k.$$

Con queste scelte il costo computazionale del metodo viene ridotto al minimo: l'operazione più costosa di ogni iterazione è il calcolo del prodotto Ad_k, che richiede $\mathcal{O}(n^2)$ operazioni aritmetiche, salvo il risparmio che si può ottenere sfruttando la simmetria di A e la sua eventuale sparsità. Ci preme notare che questo costo computazionale è confrontabile con quello dei metodi di Jacobi e Gauss-Seidel.

Algoritmo 4.7.2. Algoritmo del gradiente coniugato per la risoluzione di $Ax = b$.
Dati: n, A, b, x_0 η_r, η_a
Risultati: $esito$ indicatore del risultato:
$esito = 0$ il procedimento è terminato con successo
$esito = 1$ raggiunto il numero massimo di iterazioni (n)
x ultima iterata calcolata
k numero di iterazioni effettuate

1. $x = x_0$
2. $r = Ax - b$
3. $d = -r$
4. $w = Ad$
5. $\alpha = \frac{r^T r}{d^T w}$
6. $x = x + \alpha d$
7. $r_v = r$
8. $\gamma = \|b\|_\infty$
9. Per $k = 1, \ldots, n-1$
    1. $r = r_v + \alpha w$
    2. Se $\|r\|_\infty < \eta_r \gamma + \eta_a$, allora:
        $esito = 0$ e fermati
       Fine scelta
    3. $\beta = \frac{r^T r}{r_v^T r_v}$
    4. $d = -r + \beta d$
    5. $w = Ad$
    6. $\alpha = \frac{r^T r}{d^T w}$
    7. $x = x + \alpha d$
    8. $r_v = r$

   Fine ciclo su k
10. $esito = 1$
11. Fine

Nell'algoritmo abbiamo previsto la possibilità che il criterio di arresto non sia soddisfatto entro n iterazioni. Questa scelta può apparire in contrasto con la proprietà di terminazione finita. In realtà molti autori (a partire da Wilkinson e Reisch nel 1971 [39]) hanno rilevato che a causa dell'aritmetica floating point le direzioni $d_0, \ldots, d_k$ possono perdere al crescere di k la proprietà di essere A-coniugate da cui dipende la terminazione finita. Questo effetto deleterio degli errori di arrotondamento può essere ridotto rinunciando periodicamente (ad esempio ogni $\sqrt{n}$ iterazioni) ad aggiornare il vettore residuo con la formula ricorsiva e ricalcolandolo con la formula usuale $r_k = Ax_k - b$.

Esempio 4.7.2. Consideriamo il sistema lineare dell'esempio 4.6.1, che può essere risolto con il metodo del gradiente coniugato perché la matrice A è simmetrica e definita positiva per ogni n. Come in quell'esempio, abbiamo risolto il problema per diversi valori di n, ogni volta usando tolleranze

n	K_{GC}	E_r	R_r	$k_\infty(A)$
10	5	2.2e−16	6.3e−17	1.1e+01
50	21	4.5e−09	1.5e−08	1.7e+02
100	32	2.9e−09	9.0e−09	6.3e+02
500	99	2.9e−09	1.3e−08	1.5e+04
1000	183	4.6e−09	2.0e−08	5.9e+04
1500	268	2.6e−09	1.0e−08	1.3e+05

Tabella 4.11 Esempio 4.7.2: metodo del gradiente coniugato

$\eta_r = \eta_a = 10^{-8}$ e approssimazione iniziale $x_0 = 0$. I risultati ottenuti sono riportati nella tabella 4.11, dove K_{GC} è il numero di iterazioni effettuate e E_r e R_r indicano rispettivamente l'errore relativo e il residuo relativo in norma infinito calcolati nell'ultima iterata. Si nota subito che K_{GC} è molto minore di n e che, per i valori di n per cui il confronto è possibile, questo metodo è decisamente vantaggioso rispetto ai metodi di Jacobi e Gauss-Seidel. Possiamo anche osservare che, pur in presenza di numeri di condizionamento che crescono al crescere di n, il metodo del gradiente coniugato riesce a calcolare una soluzione affetta da un errore molto più piccolo di quanto (pessimisticamente) ci si aspetterebbe dalla solita relazione (4.13). ■

5

APPROSSIMAZIONE DI DATI E FUNZIONI

5.1 Introduzione

Questo capitolo è dedicato al problema di approssimare una funzione $f : \mathbb{R} \to \mathbb{R}$, nota per via analitica o per punti, con un'altra funzione m scelta in una classe di funzioni "semplici", dove con "semplice" intendiamo "facilmente trattabile dal punto di vista del calcolo analitico o numerico". Le classi più comunemente usate a questo scopo, quando la funzione f non presenta singolarità, sono essenzialmente due:
–funzioni di natura polinomiale, ad esempio polinomi algebrici

$$P(x) = \sum_{k=0}^{n} a_k x^k, \tag{5.1}$$

funzioni polinomiali a tratti e funzioni spline, che sono funzioni polinomiali a tratti rispondenti a particolari requisiti di regolarità;

–funzioni periodiche, ad esempio polinomi trigonometrici

$$T(x) = a_0 + \sum_{k=1}^{n} (a_k cos(k\omega) + b_k sin(k\omega)).$$

Ogni elemento di ciascuna di queste classi è individuabile fissando particolari valori dei parametri in esso presenti; ad esempio un polinomio è identificabile, una volta fissato il grado n, scegliendo i coefficienti $a_0, a_1, \ldots, a_n$.

La scelta della classe di funzioni da usare dipende da molti fattori, fra cui il numero di dati a disposizione, la possibilità o meno di acquisire all'occorrenza altri dati, lo scopo dell'approssimazione, le informazioni che si hanno sulla funzione f. Per esempio se i dati vengono da una funzione periodica o mostrano un andamento che fa pensare a una certa periodicità, si può utilizzare un polinomio trigonometrico; altrimenti ci si rivolge ad una funzione di tipo polinomiale. Noi tratteremo in dettaglio approssimanti polinomiali e splines, rimandando a [5] o [11] per gli aspetti teorici e pratici dell'uso di polinomi trigonometrici e alla letteratura specializzata per

altri tipi di approssimanti (ad esempio funzioni razionali, che si utilizzano quando f presenta delle singolarità).

In alcune situazioni la funzione f da approssimare non è nota nella sua forma analitica, ma se ne conoscono soltanto alcuni valori. Questa è la situazione tipica in cui si dispone di una tabella di valori di una grandezza, ottenuti ad esempio per via sperimentale, e si vuole determinare la legge che ad ogni valore della variabile indipendente associa il corrispondente valore misurato. Evidentemente, questa legge si identifica con una funzione $y = f(x)$ di cui non conosciamo la forma. Allora si ipotizza un modello matematico $y = m(x)$ del fenomeno e si scelgono i parametri in modo che m approssimi f in base a un qualche prefissato criterio. Una volta trovata la funzione m, essa può essere usata per diversi scopi, tipicamente per stimare il valore della grandezza che stiamo studiando in corrispondenza di valori della variabile indipendente non presenti fra i dati. Il problema di approssimare un insieme di dati con una curva è noto anche come problema del fitting di dati, dal verbo inglese "to fit" che significa "essere della forma giusta per qualcosa".

Esempio 5.1.1. In figura 5.1 diamo un esempio della situazione appena descritta. I dati si riferiscono ad un problema in ambito climatologico studiato

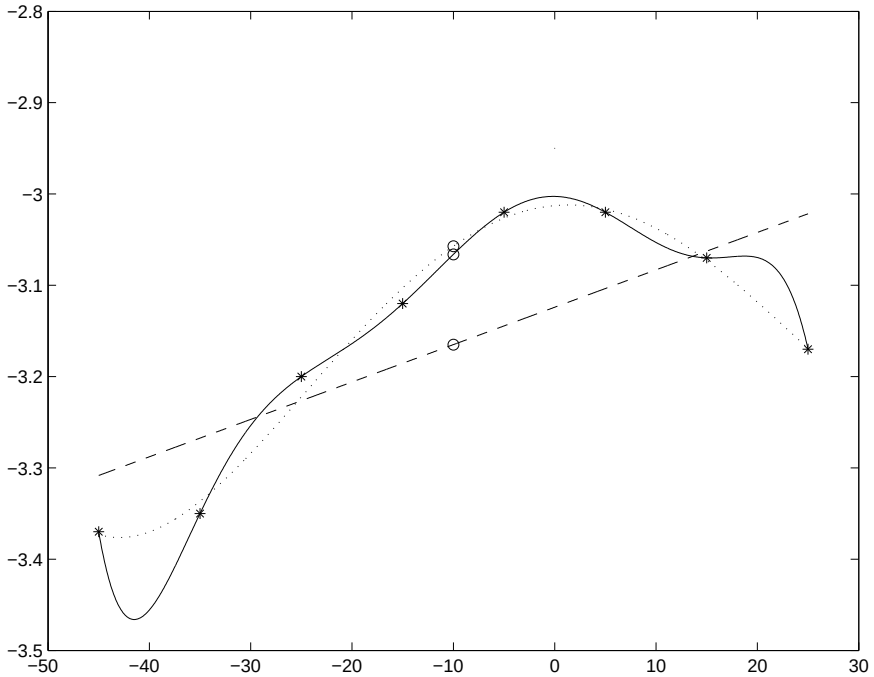

Figura 5.1 Esempi di funzioni approssimanti un insieme di dati

verso la fine dell'800, legato alla scoperta che la temperatura dell'aria al

suolo varia al variare della concentrazione K del biossido di carbonio e della latitudine. Da [4] abbiamo estratto i seguenti dati

$$(x_0, f_0) = (-45, -3.37) \quad (x_1, f_1) = (-35, -3.35)$$

$$(x_2, f_2) = (-25, -3.20) \quad (x_3, f_3) = (-15, -3.12)$$

$$(x_4, f_4) = (-5, -3.02) \quad (x_5, f_5) = (5, -3.02)$$

$$(x_6, f_6) = (15, -3.07) \quad (x_7, f_7) = (25, -3.17)$$

dove le ascisse $x_0, \ldots, x_7$ rappresentano alcune latitudini e le ordinate $f_0, \ldots, f_7$ sono le corripondenti variazioni di temperatura media annua dovute a una variazione di K da 1 a 0.67 [1]. Nella figura 5.1 i dati sono contrassegnati da un $*$. Se vogliamo stimare la variazione annua di temperatura ad una latitudine non presente fra i dati, ad esempio $x = -10$, possiamo trovare una funzione m che approssima i dati e prendere come stima $m(-10)$. Nella figura sono riportati tre possibili modelli polinomiali scelti in base a criteri diversi. Ognuno di questi modelli dà luogo ad una stima diversa del valore cercato, contrassegnata dal simbolo $\circ$. Nel caso della curva descritta da una linea continua (un polinomio di grado 7) si ha $m(-10) \simeq -3.07$; per la linea tratteggiata (un polinomio di grado 1) si ha $m(-10) \simeq -3.16$; infine la terza curva (un polinomio di grado 4) dà $m(-10) \simeq -3.06$. Non entriamo per ora nel merito dei criteri usati per definire i tre modelli, su cui torneremo più avanti, e in considerazioni relative alla bontà di queste stime. ■

La necessità di costruire un modello semplice m di una funzione f può nascere anche quando la funzione è conosciuta nella sua forma analitica. Ad esempio molti programmi per tracciare grafici di funzioni sono basati sul seguente principio: si calcola il valore della funzione f in un certo numero di punti nell'intervallo in cui ci interessa vedere il grafico e si uniscono i punti del piano cartesiano così ottenuti con segmenti rettilinei. Questo equivale ad approssimare f con una funzione m lineare a tratti. Più fitti sono i dati e più il grafico di m è vicino a quello di f. Una funzione lineare a tratti è una funzione continua, ma la sua derivata non lo è: graficamente questo si manifesta nella presenza di punti angolosi, tanto più visibili quanto più radi sono i dati. Se un grafico ottenuto in questo modo non è soddisfacente si possono utilizzare funzioni spline che permettono di ottenere modelli con un certo grado di regolarità, ad esempio con derivata prima continua.

Altre situazioni in cui si ha necessità di approssimare una funzione f nota con un funzione m più maneggevole nascono quando si devono risolvere numericamente alcuni problemi matematici per cui non si dispone di

[1] Qui e nel seguito le ordinate dei punti dati saranno indicate con il simbolo f_i, per $i = 0, 1, \ldots$. Mettiamo in evidenza in questo modo che esse rappresentano valori noti della funzione f che vogliamo approssimare.

formule esplicite. Abbiamo già incontrato una situazione di questo tipo nel Capitolo 3 quando abbiamo studiato il metodo di Newton e il metodo delle secanti per la risoluzione di equazioni non lineari. Questi metodi sono basati sul seguente principio: non avendo un modo per calcolare direttamente una soluzione dell'equazione non lineare $f(x) = 0$, si sostituisce a questo problema una successione di problemi che si sanno risolvere, in ognuno dei quali la funzione f è approssimata con un polinomio di primo grado, di cui è immediato calcolare una radice.
Un altro esempio, che verrà affrontato nell'ultimo paragrafo di questo capitolo, è rappresentato dal calcolo di un integrale definito

$$\int_a^b f(x)dx. \tag{5.2}$$

Tutti sappiamo che spesso può essere difficile, se non impossibile, trovare una primitiva di f per poter calcolare l'integrale in modo analitico. Si può allora pensare di approssimare f con una funzione m di cui si possa calcolare facilmente l'integrale (ad esempio un polinomio che sappiamo facilmente integrare qualunque sia il suo grado) e approssimare (5.2) con

$$\int_a^b m(x)dx.$$

Qualunque sia il nostro problema, una volta scelta la classe di funzioni in cui cercare il modello approssimante, si deve stabilire un criterio in base al quale selezionare una particolare funzione m. Ovviamente questo criterio deve implicare che m riproduce, in un qualche senso, il comportamento dei dati. Consideriamo a questo proposito la figura 5.1. Quello che contraddistingue la curva disegnata con linea continua dalle altre due è il fatto che essa passa esattamente per tutti i dati, mentre le altre passano fra i dati senza toccarne neppure uno (la retta) o toccandone solo alcuni (il polinomio di quarto grado). La prima curva e le altre due sono state determinate con due approcci diversi che prendono il nome di interpolazione e di migliore approssimazione rispettivamente. Più precisamente con il termine interpolazione si intende che la funzione approssimante è scelta in modo che il suo grafico passi esattamente per i punti dati [2]. Si parla invece di migliore approssimazione, in inglese "best fitting", quando la funzione approssimante è scelta in base a qualche criterio di "ottimalità" che non implica il passaggio per i punti dati ma soltanto la riproduzione del loro

[2]Il termine "interpolazione" deriva dal verbo latino interpolare= lisciare fra e viene usato in molte situazioni in cui si raccordano dei dati (parole, numeri,...) allo scopo di chiarire il significato complessivo del contesto. Sembra che questo termine sia stato usato per la prima volta nel senso matematico da John Wallis nel suo libro Arithmetica infinitorum del 1655. Un interessante articolo sulla storia dell'interpolazione è [26].

andamento complessivo. In generale si ricorre all'uno o all'altro dei due approcci in base alla fiducia che si ha nella correttezza dei dati. Se siamo certi che i dati siano esatti, o comunque affetti da errori trascurabili, allora può essere ragionevole utilizzare funzioni interpolanti. Questo succede per esempio quando si deve sostituire una funzione nota complicata da gestire con una più semplice, come nel caso del calcolo approssimato di un integrale definito. Se invece i dati sono affetti da errori non trascurabili, come spesso succede quando si tratta di misure sperimentali soggette all'imprecisione dello strumento con cui sono state effettuate, allora non è opportuno cercare di riprodurli esattamente attraverso una funzione interpolante, ma è consigliabile seguire la strada della migliore approssimazione.

5.2 Interpolazione polinomiale

Consideriamo il seguente problema: dati i punti (x_0, f_0), (x_1, f_1), ..., (x_n, f_n), si vuole determinare un polinomio $P(x)$ interpolante i dati assegnati, ovvero un polinomio che soddisfa le condizioni di interpolazione

$$P(x_i) = f_i, \quad i = 0, 1, \ldots, n.$$

Questo è un problema di interpolazione polinomiale di Lagrange [3]. In certe situazioni le condizioni di interpolazione possono coinvolgere, oltre a valori della funzione, anche valori delle sue derivate di qualche ordine in uno o più punti; in questo caso si parla di problema di interpolazione di Hermite [4]. Noi ci occuperemo soltanto del problema di interpolazione di Lagrange, d'ora in avanti detto semplicemente problema di interpolazione. Per gli aspetti teorici e numerici dell'interpolazione di Hermite, rimandiamo a [11].

I punti x_i sono chiamati punti fondamentali o nodi dell'interpolazione. Noi useremo la prima espressione, mentre riserveremo il termine "nodi" ad un altro uso nel paragrafo 5.3 dedicato alle funzioni spline. Nel seguito supporremo sempre che i punti fondamentali siano distinti, ovvero

$$x_i \neq x_j \ \text{ per } \ i \neq j, \tag{5.3}$$

come è ragionevole supporre quando si tratta di interpolazione di Lagrange: misure sperimentali o tabulazioni di funzioni sono di solito effettuate per valori distinti della variabile indipendente.

5.2.1 Esistenza e unicità del polinomio interpolante

La prima domanda da porsi è se il polinomio interpolante esiste e, in caso di risposta affermativa, se è unico. Per rispondere stabiliamo innanzitutto

[3] Joseph Louis Lagrange nacque a Torino nel 1736 e morì a Parigi nel 1813. Egli fu uno dei più grandi matematici del suo tempo (e non solo) e si occupò di molti campi della matematica e di molte applicazioni, in particolare del problema dell'interpolazione.

[4] Dal nome del matematico francese Charles Hermite (1822-1901).

il grado del polinomio. Tenendo conto che disponiamo in tutto di $n+1$ informazioni, ossia $n+1$ condizioni da imporre, possiamo ragionevolmente cercare un polinomio di grado n che sappiamo dipendere da $n+1$ coefficienti. Questa scelta è confortata da noti risultati della geometria analitica elementare: per due punti passa una retta (ovvero un polinomio di primo grado, o di grado zero) e per tre punti non allineati passa una parabola (ovvero un polinomio di secondo grado). Il nostro obiettivo diventa allora costruire un polinomio $P_n(x)$ di grado n tale che

$$P_n(x_i) = f_i, \quad i = 0, 1, \ldots, n. \tag{5.4}$$

Considerando la forma (5.1) di un generico polinomio di grado n, possiamo esplicitare (5.4), ottenendo il sistema lineare

$$\begin{cases} a_0 + x_0 a_1 + x_0^2 a_2 + \ldots + x_0^n a_n = f_0 \\ a_0 + x_1 a_1 + x_1^2 a_2 + \ldots + x_1^n a_n = f_1 \\ \vdots \\ a_0 + x_n a_1 + x_n^2 a_2 + \ldots + x_n^n a_n = f_n \end{cases} \tag{5.5}$$

nelle incognite $a_0,\ a_1, \ldots, a_n$. Il sistema ammette un'unica soluzione se il determinante della matrice dei coefficienti è diverso da zero.

Esempio 5.2.1. Si vuole determinare un polinomio interpolante i seguenti tre punti:

$$(x_0, f_0) = (-1, -3) \quad (x_1, f_1) = (1, 0) \quad (x_2, f_2) = (2, 4).$$

Imponendo il passaggio per i tre punti otteniamo il seguente sistema lineare:

$$\begin{cases} a_0 - a_1 + a_2 = -3 \\ a_0 + a_1 + a_2 = 0 \\ a_0 + 2a_1 + 4a_2 = 4 \end{cases}$$

la cui matrice dei coefficienti è

$$A = \begin{pmatrix} 1 & -1 & 1 \\ 1 & 1 & 1 \\ 1 & 2 & 4 \end{pmatrix}.$$

Poiché $\det(A) = 6$ il sistema ammette un'unica soluzione, data da

$$a_0 = -\frac{7}{3}, \quad a_1 = \frac{3}{2}, \quad a_2 = \frac{5}{6}$$

e quindi il polinomio interpolante è

$$P_2(x) = \frac{5}{6}x^2 + \frac{3}{2}x - \frac{7}{3}.$$

■

Esempio 5.2.2. Assegnati i cinque punti

$$(x_0, f_0) = (1, 2) \quad (x_1, f_1) = (2, 3) \quad (x_2, f_2) = (3, 4)$$

$$(x_3, f_3) = (4, 5) \quad (x_4, f_4) = (5, 6),$$

si vuole determinare il polinomio $P_4(x)$ che li interpola. Imponendo le condizioni di interpolazione, otteniamo il seguente sistema lineare di 5 equazioni in 5 incognite

$$\begin{cases} a_0 + a_1 + a_2 + a_3 + a_4 = 2 \\ a_0 + 2a_1 + 4a_2 + 8a_3 + 16a_4 = 3 \\ a_0 + 3a_1 + 9a_2 + 27a_3 + 81a_4 = 4 \\ a_0 + 4a_1 + 16a_2 + 64a_3 + 256a_4 = 5 \\ a_0 + 5a_1 + 25a_2 + 125a_3 + 625a_4 = 6 \end{cases}$$

la cui soluzione è

$$a_0 = 1, a_1 = 1, a_2 = 0, a_3 = 0, a_4 = 0.$$

Conseguentemente il polinomio interpolante risulta

$$P_4(x) = x + 1$$

che è di primo grado e non di quarto grado. La spiegazione di questo risultato è nel fatto che i punti dati non sono disposti "casualmente" nel piano cartesiano, ma sono tutti disposti lungo una linea retta. L'equazione di questa retta è proprio $y = 1 + x$. ■

Tornando al caso generale, osserviamo che la matrice dei coefficienti del sistema (5.5) è

$$A = \begin{pmatrix} 1 & x_0 & x_0^2 & \dots & x_0^n \\ 1 & x_1 & x_1^2 & \dots & x_1^n \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^n \end{pmatrix}$$

che riconosciamo essere la matrice di Vandermonde già incontrata nell'esempio 4.2.1 del Capitolo 4. Ricordando che il determinante di A è dato da (4.12)

$$det(A) = \prod_{i=1}^{n-1} \prod_{j=i+1}^{n} (x_j - x_i),$$

possiamo concludere che il sistema (5.5) ammette un'unica soluzione in virtù dell'ipotesi 5.3. In conclusione, abbiamo dimostrato il seguente teorema.

Teorema 5.2.1. Dati i punti (x_i, f_i), $i = 0, 1, \ldots, n$, con $x_i \neq x_j$ per $i \neq j$, esiste un unico polinomio di grado minore o uguale a n che li interpola.

Ci preme notare che nell'enunciato del teorema si parla di grado minore o uguale di n e non di grado n. Infatti, come mostrato nell'esempio 5.2.2, niente assicura che il polinomio interpolante abbia esattamente grado n.

Abbiamo visto che risolvere il problema di interpolazione polinomiale equivale a risolvere il sistema lineare (5.5), caratterizzato da una matrice dei coefficienti di Vandermonde. Questa strada, che serve a dimostrare l'esistenza e unicità del polinomio interpolante, non viene seguita in pratica per determinare il polinomio, ma è comunque utile per avere delle indicazioni sul condizionamento del problema. Infatti, come abbiamo avuto modo di verificare nel Capitolo 4, il condizionamento delle matrici di Vandermonde può essere (molto) cattivo al crescere di n e, da quanto visto nell'esempio 4.2.1, ci possiamo aspettare una forte dipendenza del condizionamento dalla scelta dei punti fondamentali $x_0, \ldots, x_n$. Generalizzando i risultati di quell'esempio possiamo prevedere che punti equidistanti portino un forte mal condizionamento anche per n non troppo grande, mentre scelte diverse potrebbero ridurre la gravità del problema. Esamineremo in dettaglio il condizionamento del problema di interpolazione polinomiale nel paragrafo 5.2.3.

5.2.2 Polinomio interpolante nella formulazione di Lagrange

Consideriamo l'interpolazione di due punti (x_0, f_0) e (x_1, f_1) mediante un polinomio $P_1(x)$ di grado ≤ 1. Dalla geometria analitica elementare conosciamo l'equazione della retta che unisce i due punti:

$$\frac{y - f_0}{f_0 - f_1} = \frac{x - x_0}{x_0 - x_1}. \tag{5.6}$$

Esplicitando y otteniamo un'espressione del polinomio interpolante

$$P_1(x) = f_0 \frac{x - x_1}{x_0 - x_1} + f_1 \frac{x - x_0}{x_1 - x_0} \tag{5.7}$$

che puo' essere facilmente generalizzata al caso dell'interpolazione con $n+1$ dati. A questo scopo osserviamo che, definendo

$$l_0(x) = \frac{x - x_1}{x_0 - x_1} \text{ e } l_1(x) = \frac{x - x_0}{x_1 - x_0},$$

possiamo riscrivere la (5.7) nella forma

$$P_1(x) = l_0(x) f_0 + l_1(x) f_1. \tag{5.8}$$

Le due funzioni l_0 e l_1 sono polinomi di primo grado ed assumono nei due punti fondamentali x_0 e x_1 valori 0 ed 1; più precisamente

$$l_0(x_0) = 1 \ \text{ e } \ l_0(x_1) = 0$$

mentre

$$l_1(x_0) = 0 \ \text{ e } \ l_1(x_1) = 1.$$

La (5.8) mostra che il polinomio interpolante P_1 può essere scritto come combinazione lineare con coefficienti f_0 e f_1 dei due polinomi di primo grado l_0 e l_1. Per estendere questa formula al caso generale in cui si hanno $n+1$ punti (x_i, f_i), con $i = 0, \ldots, n$, dobbiamo cercare $n+1$ polinomi $l_0, l_1, \ldots, l_n$ di grado n tali che

$$l_j(x_i) = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases} \tag{5.9}$$

e quindi scrivere

$$P_n(x) = f_0 l_0(x) + f_1 l_1(x) + \ldots + f_i l_i(x) + \ldots + f_n l_n(x).$$

È infatti banale vedere, sfruttando la (5.9), che questo polinomio soddisfa le condizioni di interpolazione. Come possiamo definire i polinomi $l_0, l_1, \ldots, l_n$? Dalla (5.9) risulta che, fissato j, il polinomio l_j si annulla in tutti i punti fondamentali x_i con $i \neq j$ e deve quindi avere la forma

$$l_j(x) = \alpha_j (x - x_0)(x - x_1) \cdots (x - x_{j-1})(x - x_{j+1}) \cdots (x - x_n)$$

per qualche valore del coefficiente α_j. Il valore di α_j può essere determinato imponendo l'ulteriore condizione $l_j(x_j) = 1$, da cui si ottiene

$$\alpha_j = \frac{1}{(x_j - x_0)(x_j - x_1) \ldots (x_j - x_{j-1})(x_j - x_{j+1}) \ldots (x_j - x_n)}$$

e pertanto

$$l_j(x) = \frac{(x - x_0)(x - x_1) \cdots (x - x_{j-1})(x - x_{j+1}) \cdots (x - x_n)}{(x_j - x_0)(x_j - x_1) \cdots (x_j - x_{j-1})(x_j - x_{j+1}) \cdots (x_j - x_n)}.$$

Riassumendo, il polinomio interpolante P_n è dato da

$$L_n(x) = \sum_{j=0}^{n} f_j l_j(x) \tag{5.10}$$

con

$$l_j(x) = \frac{\prod_{i=1,\ i\neq j}^{n} (x - x_i)}{\prod_{i=1,\ i\neq j}^{n} (x_j - x_i)} \quad j = 0, 1, \ldots, n. \tag{5.11}$$

Nella (5.10) abbiamo usato il simbolo L_n invece di P_n perché questa forma è nota come formulazione di Lagrange del polinomio interpolante. I polinomi $l_0, l_1, \ldots, l_n$ sono detti polinomi fondamentali di Lagrange. Ci preme sottolineare che nell'ipotesi (5.3) il Teorema 5.2.1 assicura l'unicità del polinomio interpolante. Pertanto L_n e il polinomio che si otterebbe risolvendo il sistema (5.5) sono lo stesso polinomio, scritto in due modi diversi.

Esempio 5.2.3. Vogliamo determinare il polinomio in forma di Lagrange che interpola i dati dell'esempio 5.2.1

$$(x_0, f_0) = (-1, -3) \quad (x_1, f_1) = (1, 0) \quad (x_2, f_2) = (2, 4).$$

A questo scopo costruiamo i tre polinomi di Lagrange $l_0(x)$, $l_1(x)$, $l_2(x)$ secondo la (5.11)

$$\begin{array}{rcccl} l_0(x) & = & \frac{(x-1)(x-2)}{(-1-1)(-1-2)} & = & \frac{x^2-3x+2}{6} \\ l_1(x) & = & \frac{(x+1)(x-2)}{(1+1)(1-2)} & = & -\frac{x^2-x-2}{2} \\ l_2(x) & = & \frac{(x+1)(x-1)}{(2+1)(2-1)} & = & \frac{x^2-1}{3} \end{array}$$

ottenendo

$$\begin{aligned} L_2(x) &= f_0 l_0(x) + f_1 l_1(x) + f_2 l_2(x) \\ &= -3\frac{x^2-3x+2}{6} - 0\frac{x^2-x-2}{2} + 4\frac{x^2-1}{3} \\ &= \frac{5}{6}x^2 + \frac{3}{2}x - \frac{7}{3}. \end{aligned}$$

Essendo il polinomio interpolante unico, abbiamo ovviamente trovato lo stesso polinomio dell'esempio 5.2.1. ■

Facciamo ora alcune considerazioni sulla formulazione di Lagrange del polinomio interpolante, mettendo in evidenza gli aspetti positivi e quelli negativi.

1. Prima di tutto osserviamo che nell'esempio 5.2.3 avremmo potuto evitare di calcolare l'espressione di $l_1(x)$ dal momento che $f_1 = 0$. In generale la formulazione di Lagrange è particolarmente vantaggiosa dal punto di vista del calcolo se alcuni dei dati assegnati hanno le ordinate nulle ed ovviamente il vantaggio aumenta all'aumentare del numero di questi dati.
2. Supponiamo di aver misurato diverse grandezze o tabulato diverse funzioni sullo stesso insieme di ascisse (ad esempio abbiamo le misure di temperatura e pressione di un fluido negli stessi istanti di tempo) e di voler interpolare i diversi insiemi di punti così ottenuti. Poiché i polinomi fondamentali di Lagrange dipendono esclusivamente dalle ascisse dei punti di interpolazione e non cambiano se cambiano le ordinate, possiamo calcolarli una sola volta e usarli per creare tutti i polinomi interpolanti che ci interessano.

3. Se, come nell'esempio 5.2.2, siamo di fronte ad una degenerazione del grado del polinomio interpolante, questo diventa noto soltanto al termine dei calcoli.
4. Supponiamo di aver calcolato il polinomio che interpola certi dati e di dover aggiungere un punto. Siccome i polinomi fondamentali di Lagrange dipendono dalle ascisse di tutti i punti su cui si interpola, essi dovranno essere calcolati ex-novo, senza poter riutilizzare nessuno dei calcoli fatti precedentemente.

I primi due punti evidenziano gli aspetti positivi della formulazione di Lagrange, mentre gli altri due riguardano i suoi inconvenienti. Il punto 4 in particolare mette in evidenza un aspetto fortemente negativo di questa forma del polinomio interpolante perché nelle applicazioni succede spesso di usare inizialmente pochi dati e aggiungerne via via altri fino a quando l'approssimazione trovata non risulta soddisfacente. Così, in pratica, la formulazione di Lagrange non viene quasi mai utilizzata e si preferisce ricorrere alla formulazione di Newton, di cui parleremo nel paragrafo 5.2.4. Per questo motivo tralasciamo di descrivere algoritmi relativi alla forma di Lagrange.

5.2.3 Condizionamento del problema di interpolazione polinomiale

Al di là degli svantaggi che essa presenta dal punto di vista pratico, la formulazione di Lagrange è importante dal punto di vista teorico. Ad esempio essa si presta bene per studiare il condizionamento del problema dell'interpolazione polinomiale rispetto a errori nelle misure $f_0, f_1, \ldots, f_n$. Siano $\tilde{f}_i = f_i(1+\epsilon_i)$, per $i = 0, 1, \ldots, n$, i valori perturbati. Chiediamoci come le perturbazioni $\epsilon_0, \epsilon_1, \ldots, \epsilon_n$ si ripercuotono sui valori che il polinomio interpolante assume per valori di x distinti dai punti fondamentali $x_0, x_1, \ldots, x_n$. Più precisamente, sia $\tilde{L}_n(x)$ il polinomio corrispondente al nuovo insieme di dati e $[a, b]$ un intervallo contenente i punti fondamentali, per fissare le idee

$$[a, b] = [\min_{0\le i\le n} x_i, \max_{0\le i\le n} x_i]. \tag{5.12}$$

Vogliamo allora stimare la perturbazione relativa

$$\frac{\max_{x\in[a,b]} |\tilde{L}_n(x) - L_n(x)|}{\max_{x\in[a,b]} |L_n(x)|}$$

in funzione di $\epsilon_{max} = \max_{0\le j\le n} |\epsilon_j|$. Da

$$L_n(x) = \sum_{j=0}^{n} f_j l_j(x) \;\text{ e }\; \tilde{L}_n(x) = \sum_{j=0}^{n} \tilde{f}_j l_j(x),$$

sottraendo termine a termine si ottiene

$$\begin{aligned} |\tilde{L}_n(x) - L_n(x)| &= |\sum_{j=0}^{n}(f_j - \tilde{f}_j)l_j(x)| = |\sum_{j=0}^{n} f_j\epsilon_j l_j(x)| \\ &\leq \epsilon_{max} \max_{0\leq j\leq n} |f_j| \sum_{j=0}^{n} |l_j(x)|. \end{aligned}$$

Poiché $f_j = L_n(x_j)$ per ogni j, si ha

$$\max_{0\leq j\leq n} |f_j| = \max_{0\leq j\leq n} |L_n(x_j)| \leq \max_{x\in[a,b]} |L_n(x)|$$

e quindi

$$\begin{aligned} |\tilde{L}_n(x) - L_n(x)| &\leq \epsilon_{max} \max_{x\in[a,b]} |L_n(x)| \sum_{j=0}^{n} |l_j(x)| \\ &\leq \epsilon_{max} \max_{x\in[a,b]} |L_n(x)| \max_{x\in[a,b]} \sum_{j=0}^{n} |l_j(x)|, \end{aligned}$$

da cui otteniamo la stima desiderata:

$$\frac{\max_{x\in[a,b]} |\tilde{L}_n(x) - L_n(x)|}{\max_{x\in[a,b]} |L_n(x)|} \leq \Lambda_n \epsilon_{max}, \tag{5.13}$$

dove

$$\Lambda_n = \max_{x\in[a,b]} \sum_{j=0}^{n} |l_j(x)|$$

è detta costante di Lebesgue.

Esaminiamo la (5.13). Essa mostra che a piccole perturbazioni dei dati corrispondono piccole perturbazioni del risultato se Λ_n è piccola; viceversa, se Λ_n è grande, a piccole perturbazioni dei dati possono corrispondere grosse perturbazioni del risultato. In poche parole, Λ_n funziona da numero di condizionamento del problema. Le proprietà di Λ_n sono state e sono tuttora molto studiate (cfr. [31] e [34]). Qui ci limitiamo a ricordare le principali. Si può dimostrare che $\Lambda_n \geq 1$ comunque siano scelti i punti fondamentali (da cui Λ_n dipende). Inoltre, per n grande e punti fondamentali equidistanti si ottiene

$$\Lambda_n \geq e^{\frac{n}{2}}.$$

Questi valori crescono molto rapidamente con n e quindi l'uso di punti equidistanti non è assolutamente vantaggioso. Quando possibile, conviene fare scelte diverse. Una buona scelta, relativamente all'intervallo [-1,1], è

$$x_i = cos\left(\frac{2i+1}{2n+2}\pi\right) \quad i = 0, 1, \ldots, n \tag{5.14}$$

che per n grande porta a

$$\Lambda_n \simeq \frac{2}{\pi} log(n),$$

che è il più piccolo valore che si può ottenere. I punti (5.14) sono detti punti di Chebyshev perché sono gli zeri di un particolare polinomio di grado $n+1$, detto polinomio di Chebyshev. I polinomi di Chebyshev $T_0(x), T_1(x), \ldots$, sono una famiglia di polinomi di grado crescente definiti dalla formula ricorsiva

$$T_0(x) = 1 \ , \ \ T_1(x) = x$$

$$T_{k+1}(x) = 2xT_k(x) - T_{k-1}(x), \ \ k = 1, 2, \ldots$$

Si tratta di una famiglia di polinomi ortogonali, concetto che in questo libro non introduciamo, ma che riveste importanza fondamentale in molti campi della matematica teorica e numerica. I polinomi di Chebyshev sono così chiamati dal nome del matematico russo [5] che li scoprì. I polinomi di Chebyshev hanno tutte radici reali e distinte appartenenti all'intervallo [-1,1], definite, per $k = n+1$, dalla formula (5.14). È possibile estendere la definizione dei polinomi di Chebyshev a un intervallo qualunque $[a, b] \neq [-1, 1]$ mediante un opportuno cambiamento di variabile. In tal caso invece della (5.14) si ottiene

$$x_i = \frac{b-a}{2} cos\left(\frac{2i+1}{2n+2}\pi\right) + \frac{a+b}{2}, \ \ i = 0, \ldots n. \tag{5.15}$$

I punti (5.15), che noi chiameremo "punti di Chebyshev generalizzati", mantengono le buone proprietà dei punti di Chebyshev originali.

5.2.4 Polinomio interpolante nella formulazione di Newton

Prendiamo di nuovo in considerazione il problema dell'interpolazione di due punti (x_0, f_0) e (x_1, f_1). Il polinomio che li interpola può essere scritto anche come

$$N_1(x) = f_0 + \frac{f_1 - f_0}{x_1 - x_0}(x - x_0), \tag{5.16}$$

dove usiamo il simbolo $N_1(x)$ perché questa è la formulazione di Newton. Osserviamo che il polinomio di grado 0 interpolante un solo punto (x_0, f_0)

[5] Pafnutii Lvovich Chebyshev (1821-1894). In un articolo di R.Roy comparso su una rivista russa nel 1996, 175^o anniversario della sua nascita, si diceva: Chebyshev was probably the first mathematician to recognise the general concept of orthogonal polynomials. A few particular orthogonal polynomials were known before his work. Legendre and Laplace had encountered the Legendre polynomials in their work on celestial mechanics in the late eighteenth century... It was Chebyshev who saw the possibility of a general theory and its applications. His work arose out of the theory of least squares approximation and probability; he applied his results to interpolation, approximate quadrature and other areas.

è dato da $N_0(x) = f_0$ e quindi da (5.16) segue che

$$N_1(x) = N_0(x) + \frac{f_1 - f_0}{x_1 - x_0}(x - x_0).$$

Ora aggiungiamo un nuovo punto (x_2, f_2) ed estendiamo la forma precedente. Considerando che N_1 è una combinazione lineare del polinomio N_0 di grado 0 con il polinomio $(x - x_0)$ di grado 1, proviamo a scrivere N_2 come combinazione di N_0, $(x - x_0)$ e $(x - x_0)(x - x_1)$, cioè

$$N_2(x) = A_0 + A_1(x - x_0) + A_2(x - x_0)(x - x_1)$$

e a determinare i coefficienti A_0, A_1 e A_2 mediante le condizioni di interpolazione

$$N_2(x_0) = f_0, \;\; N_2(x_1) = f_1, \;\; N_2(x_2) = f_2.$$

Imponendo queste condizioni si ottiene il seguente sistema lineare nelle incognite A_0, A_1 e A_2:

$$\begin{cases} A_0 = f_0 \\ A_0 + A_1(x_1 - x_0) = f_1 \\ A_0 + A_1(x_2 - x_0) + A_2(x_2 - x_0)(x_2 - x_1) = f_2 \end{cases}$$

Trattandosi di un sistema triangolare inferiore, possiamo calcolare la soluzione usando l'algoritmo di sostituzione in avanti:

$$A_0 = f_0$$

$$A_1 = \tfrac{f_1 - A_0}{x_1 - x_0} = \tfrac{f_1 - f_0}{x_1 - x_0}$$

$$A_2 = \tfrac{f_2 - A_0 - A_1(x_2 - x_0)}{(x_2 - x_0)(x_2 - x_1)} = \tfrac{f_2 - f_0 - \frac{f_1 - f_0}{x_1 - x_0}(x_2 - x_0)}{(x_2 - x_0)(x_2 - x_1)}.$$

Riconosciamo in A_0 e A_1 i coefficienti già visti nell'espressione (5.16); per quanto riguarda A_2, possiamo vedere che

$$\begin{aligned} A_2 &= \tfrac{1}{x_2 - x_1}\left[\tfrac{f_2 - f_0}{(x_2 - x_1)} - \tfrac{f_1 - f_0}{(x_1 - x_0)}\right] \\ &= \tfrac{1}{x_2 - x_1}\left[\tfrac{f_2(x_1 - x_0) - f_1(x_1 - x_0) + f_1(x_1 - x_0) - f_0(x_1 - x_0) - f_1(x_2 - x_0) + f_0(x_2 - x_0)}{(x_2 - x_1)(x_1 - x_0)}\right] \\ &= \tfrac{1}{x_2 - x_0}\left[\tfrac{f_2 - f_1}{x_2 - x_1} + \tfrac{f_1(x_1 - x_2) - f_0(x_1 - x_2)}{(x_2 - x_1)(x_1 - x_0)}\right] \\ &= \tfrac{1}{x_2 - x_0}\left[\tfrac{f_2 - f_1}{x_2 - x_1} - \tfrac{f_1 - f_0}{x_1 - x_0}\right]. \end{aligned}$$

I coefficienti A_0, A_1 e A_2 vengono di solito indicati con i simboli $f[x_0]$, $f[x_0, x_1]$ e $f[x_0, x_1, x_2]$ e chiamati differenze divise di f relative a x_0 di

ordine 0, 1 e 2 rispettivamente. Qui f indica al solito la funzione che stiamo approssimando. Possiamo allora scrivere

$$N_2(x) = f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1). \quad (5.17)$$

In generale, dati $n+1$ punti (x_0, f_0), (x_1, f_1), ..., (x_n, f_n), si possono definire differenze divise di f di ordine da 0 a n in modo ricorsivo come segue:

– per $i \in \{0, 1, \ldots, n\}$ le differenze divise di ordine 0 relative a x_i sono definite da

$$f[x_i] = f_i;$$

– per $i \in \{0, 1, \ldots, n-k\}$ le differenze divise di ordine k relative a x_i, sono definite mediante le differenze divise di ordine $k-1$ da

$$f[x_i, x_{i+1}, \ldots, x_{i+k}] = \frac{f[x_{i+1}, x_{i+2}, \ldots, x_{i+k}] - f[x_i, x_{i+1}, \ldots, x_{i+k-1}]}{x_{i+k} - x_i}.$$

In questo modo si arriva fino alla differenza divisa di ordine n relativa a x_0

$$f[x_0, \ldots, x_n] = \frac{f[x_1, \ldots, x_n] - f[x_0, \ldots, x_{n-1}]}{x_n - x_0}.$$

Le differenze divise del primo ordine sono rapporti incrementali e quelle di ordine superiore sono una sorta di rapporti incrementali di ordine superiore al primo. Da questo punto di vista, le differenze divise possono essere considerate una generalizzazione delle derivate per funzioni date per punti.

Utilizzando anche le differenze divise di ordine superiore al secondo, possiamo generalizzare la (5.17) e ottenere il polinomio interpolante N_n:

$$N_n(x) \quad = \quad A_0 + A_1\omega_1(x) + A_2\omega_2(x) + \ldots + A_n\omega_n(x), \quad (5.18)$$

dove

$$A_j = f[x_0, \ldots, x_j] \ , \ \text{per} \ j = 0, 1, \ldots, n \quad (5.19)$$

e

$$\omega_j(x) = \Pi_{i=0}^{j-1}(x - x_i) \ , \ \text{per} \ j = 1, 2, \ldots, n. \quad (5.20)$$

Lasciamo per esercizio la verifica del fatto che il polinomio definito da (5.18)-(5.19)-(5.20) soddisfa le condizioni di interpolazione.

Negli esempi successivi raccoglieremo le differenze divise nella tabella delle differenze divise, così strutturata:

x_i	$f_i = f[x_i]$	$f[x_i, x_{i+1}]$	$f[x_i, x_{i+1}, x_{i+2}]$	$f[x_i, x_{i+1}, x_{i+2}, x_{i+3}]$	$\dots$
x_0	f_0				
		$f[x_0, x_1]$			
x_1	f_1		$f[x_0, x_1, x_2]$		
		$f[x_1, x_2]$		$f[x_0, x_1, x_2, x_3]$	$\dots$
x_2	f_2		$f[x_1, x_2, x_3]$		
		$f[x_2, x_3]$		$\dots$	
x_3	f_3		$\dots$		
		$\dots$			
$\dots$	$\dots$				

Le differenze divise che servono ai fini della costruzione dei polinomi interpolanti sono quelle relative a x_0, che occupano la "diagonale superiore" della tabella.

Esempio 5.2.4. Consideriamo nuovamente i dati già visti nell'esempio 5.2.1, cioè

$$(x_0, f_0) = (-1, -3) \quad (x_1, f_1) = (1, 0) \quad (x_2, f_2) = (2, 4).$$

Per determinare il polinomio interpolante nella forma di Newton costruiamo la tabella delle differenze divise

x_i	f_i	$f[x_i, x_{i+1}]$	$f[x_i, x_{i+1}, x_{i+2}]$
-1	-3		
		$\frac{0+3}{1+1} = \frac{3}{2}$	
1	0		$\frac{4-\frac{3}{2}}{2+1} = \frac{5}{6}$
		$\frac{4-0}{2-1} = 4$	
2	4		

(5.21)

e utilizziamo quelle sulla "diagonale superiore", nell'ordine -3, $\frac{3}{2}$ e $\frac{5}{6}$ per scrivere

$$N_2(x) = -3 + \frac{3}{2}(x+1) + \frac{5}{6}(x-1)(x+1) = \frac{5}{6}x^2 + \frac{3}{2}x - \frac{7}{3}$$

■

Esaminando la forma generale (5.18) del polinomio interpolante di Newton notiamo subito che

$$N_n(x) = N_{n-1}(x) + A_n \omega_n(x),$$

dove N_{n-1} è il polinomio che interpola i primi n punti, da (x_0, f_0) a (x_{n-1}, f_{n-1}). Questo ci dice che la formulazione di Newton del polinomio interpolante permette di ovviare al più grave svantaggio della formulazione di Lagrange. Supponiamo infatti di aver determinato N_{n-1} e di aggiungere il nuovo punto (x_n, f_n). Per determinare il nuovo polinomio N_n non occorre rifare tutti i calcoli, ma è sufficiente calcolare la differenza divisa $f[x_0, \ldots, x_n]$ di ordine n ed aggiungere a N_{n-1} il termine $A_n\omega_n(x)$ di grado n.

Esempio 5.2.5. Consideriamo nuovamente i dati dell'esempio precedente e aggiungiamo il punto $(x_3, f_3) = (0, 0)$. Il nuovo polinomio interpolante sarà di grado 3 ed avrà la forma

$$N_3(x) = N_2(x) + f[-1, 1, 2, 0](x + 1)(x - 1)(x - 2).$$

Per calcolare $f[-1, 1, 2, 0]$ aggiungiamo alla tabella 5.21 la riga con il nuovo punto e calcoliamo le differenze divise di ordine 1, 2 e 3 che lo coinvolgono:

x_i	f_i	$f[x_i, x_{i+1}]$	$f[x_i, x_{i+1}, x_{i+2}]$	$f[x_i, x_{i+1}, x_{i+2}, x_{i+3}]$
-1	-3			
		$\frac{3}{2}$		
1	0		$\frac{5}{6}$	
		4		$\frac{2-\frac{5}{6}}{0+1} = \frac{7}{6}$
2	4		$\frac{2-4}{0-1} = 2$	
		$\frac{0-4}{0-2} = 2$		
0	0			

La differenza divisa che ci occorre è $\frac{7}{6}$, con la quale possiamo formare il nuovo polinomio

$$\begin{aligned} N_3(x) &= N_2(x) + \frac{7}{6}(x + 1)(x - 1)(x - 2) \\ &= \frac{5}{6}x^2 + \frac{3}{2}x - \frac{7}{3} + \frac{7}{6}(x + 1)(x - 1)(x - 2) \\ &= \frac{7}{6}x^3 - \frac{3}{2}x^2 + \frac{1}{3}x. \end{aligned}$$

Vale la pena notare che nell'aggiornare la tabella delle differenze divise non ci siamo preoccupati del fatto che le ascisse non fossero ordinate. È evidente che il polinomio interpolante non cambia cambiando l'ordine dei dati, che quindi possono essere ordinati per ascisse crescenti come no. Nel caso specifico della formulazione di Newton abbiamo inoltre la garanzia che le differenze divise sono invarianti rispetto all'ordine degli argomenti (la dimostrazione è facile: si procede per induzione a partire dalle differenze divise del primo ordine per le quali la proprietà è evidente). ■

Il prossimo esempio mette in luce un'altra buona caratteristica della formulazione di Newton, legata ai casi in cui il polinomio interpolante non ha grado massimo n. Ricordiamo che la formulazione di Lagrange permette di accorgersi di questa situazione soltanto al termine di tutti i calcoli.

Esempio 5.2.6. Abbiamo visto che i punti dell'esempio 5.2.2

$$(x_0, f_0) = (1, 2) \quad (x_1, f_1) = (2, 3) \quad (x_2, f_2) = (3, 4)$$

$$(x_3, f_3) = (4, 5) \quad (x_4, f_4) = (5, 6)$$

appartengono al grafico della retta di equazione $y = x + 1$. Se cerchiamo il polinomio interpolante nella forma di Newton e calcoliamo le differenze divise otteniamo la tabella

x_i	f_i	$f[x_i, x_{i+1}]$	$f[x_i, x_{i+1}, x_{i+2}]$	$f[x_i, \ldots, x_{i+3}]$	$f[x_i, \ldots, x_{i+4}]$
1	2				
		$\frac{3-2}{2-1} = 1$			
2	3		0		
		$\frac{4-3}{3-2} = 1$		0	
3	4		0		0
		$\frac{5-4}{4-3} = 1$		0	
4	5		0		
		$\frac{6-5}{5-4} = 1$			
5	6				

dove si nota che tutte le differenze di ordine superiore al primo sono uguali a zero, il che porta a

$$N_4(x) = 2 + (x - 1) = x + 1.$$

Ovviamente avremmo potuto fermare la costruzione della tabella subito dopo aver calcolato le differenze divise del primo ordine; essendo infatti queste tutte uguali fra loro, si poteva subito dedurre che tutte le differenze divise successive sarebbero venute uguali a zero. ■

5.2.5 Algoritmi

Possiamo ora procedere a descrivere gli algoritmi per costruire il polinomio interpolante in forma di Newton e calcolare il valore che esso assume in punti distinti da quelli fondamentali. Proponiamo due algoritmi distinti: il primo algoritmo assume come dati i punti (x_0, f_0), (x_1, f_1), …, (x_n, f_n) e calcola le differenze divise $A_0, A_1, \ldots, A_n$. Il secondo algoritmo assume come dati le differenze divise e calcola il valore del polinomio in un dato $\hat{x}$. Lasciamo per esercizio ai lettori modificare questo secondo algoritmo in modo che calcoli il valore del polinomio in più di un punto.

Nel primo algoritmo si fanno intervenire i punti (x_i, f_i) uno per volta e, utilizzando un vettore di lavoro g con componenti $g_0, g_1, \ldots, g_n$, si costruiscono le differenze divise nel seguente ordine:

$g_0 = f_0$;

$g_1 = f_1$, $g_0 = f[x_0, x_1]$;

$g_2 = f_2$, $g_1 = f[x_1, x_2]$, $g_0 = f[x_0, x_1, x_2]$;

$g_3 = f_3$, $g_2 = f[x_2, x_3]$, $g_1 = f[x_1, x_2, x_3]$, $g_0 = f[x_0, x_1, x_2, x_3]$;

e così via.

Algoritmo 5.2.1. Algoritmo per il calcolo delle differenze divise $A_0, A_1, \ldots, A_n$.

Dati: n, $(x_0, f_0), (x_1, f_1), \ldots, (x_n, f_n)$
Risultato: $A_0, A_1, \ldots, A_n$
1. Per $i = 0, 1, \ldots, n$
    1. $g_i = f_i$
    2. Per $k = i-1, i-2, \ldots, 0$
        1. $g_k = \frac{g_{k+1} - g_k}{x_i - x_k}$
    Fine ciclo su k
    3. $A_i = g_0$
  Fine ciclo su i
2. Fine

Nel ciclo in cui la variabile k assume valori decrescenti da $i-1$ a 0, facciamo la convenzione che esso venga saltato per $i = 0$, ovvero quando $i - 1 < 0$. Questa convenzione trova riscontro nelle istruzioni che realizzano i cicli finiti nei linguaggi di programmazione.

Per quanto riguarda il calcolo del valore $N_n(\hat{x})$ per un dato $\hat{x}$, nell'algoritmo 5.2.2 utilizziamo la forma annidata

$$\begin{aligned} N_n(x) &= A_0 + (x - x_0)[A_1 + (x - x_1)[A_2 + (x - x_2)[A_3 + \ldots \\ &\quad (x - x_{n-2})[A_{n-1} + A_n(x - x_{n-1})] \ldots]]] \end{aligned}$$

nello stesso spirito dell'algoritmo di Hörner 1.3.8. In questo modo si risparmiano infatti un certo numero di operazioni aritmetiche rispetto a quelle che si farebbero usando la forma (5.18).

Algoritmo 5.2.2. Algoritmo per il calcolo di $N_n(\hat{x})$

Dati: n, $\hat{x}$, $x_0, x_1, \ldots, x_n$, $A_0, A_1, \ldots, A_n$
Risultato: $y = N_n(\hat{x})$
1. $s = A_n$
2. Per $i = n-1, n-2, \ldots, 0$
    1. $s = s(\hat{x} - x_i) + A_i$
   Fine ciclo su i
3. $y = s$
4. Fine

5.2.6 Errore nell'interpolazione polinomiale

Il problema dell'interpolazione nasce quando c'è necessità di approssimare il valore di una funzione in punti diversi da quelli in cui è stata misurata. Di conseguenza è di fondamentale importanza poter valutare l'errore di interpolazione

$$E_n(\hat{x}) = f(\hat{x}) - P_n(\hat{x})$$

in un punto $\hat{x}$ distinto dai punti fondamentali, cioè l'errore che si commette quando approssimiamo il valore $f(\hat{x})$ con il valore assunto in $\hat{x}$ dal polinomio interpolante P_n. Si suppone che $\hat{x}$ appartenga all'intervallo $[a, b]$ definito dai punti fondamentali (cfr. (5.12)). Se così non fosse staremmo usando un polinomio interpolante per estrapolare un valore di f. L'estrapolazione è un processo molto più delicato dal punto di vista degli errori, perché si usano informazioni sul comportamento della funzione f in un intervallo per poi stimarne valori al di fuori di questo intervallo.

Prima di enunciare i principali risultati teorici sull'errore di interpolazione mostriamo alcuni esempi illustrativi che mettono in luce gli aspetti fondamentali della questione.

Esempio 5.2.7. Consideriamo la funzione $f(x) = sin(x)$ e interpoliamola nell'intervallo $[-\pi, \pi]$ usando un numero crescente di punti fondamentali equidistanti, estremi compresi. Nei quattro grafici della figura 5.2 vediamo i polinomi costruiti con 3, 4, 5 e 6 punti. In questi grafici i punti fondamentali sono indicati con un $*$, la funzione f è disegnata con una linea tratteggiata e i polinomi interpolanti con una linea continua, Notiamo che al crescere di n i polinomi approssimano sempre meglio la funzione su tutto l'intervallo e che quello di grado più alto praticamente si sovrappone alla f. ■

Esempio 5.2.8. Consideriamo ora la funzione

$$f(x) = \frac{1}{1 + 25x^2}$$

per $x \in [-1, 1]$. Nella figura 5.3 riportiamo i polinomi interpolanti di grado $n = 4$, $n = 6$, $n = 8$ e $n = 10$ costruiti su punti fondamentali equidistanti. È chiaro che per questa funzione le cose vanno diversamente dall'esempio precedente: al crescere di n il polinomio interpolante approssima sempre

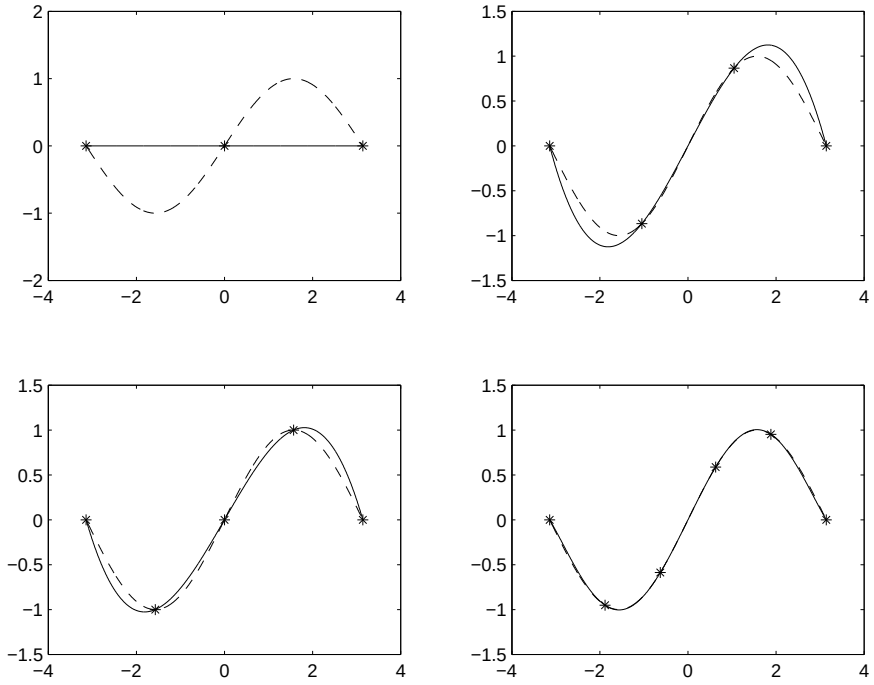

Figura 5.2 Interpolazione di $f(x) = sin(x)$ su punti fondamentali equidistanti

meglio la funzione nella zona centrale dell'intervallo, ma sempre peggio verso gli estremi. La funzione f è nota come funzione di Runge e il fenomeno di crescita degli errori di interpolazione agli estremi dell'intervallo prende il nome di fenomeno di Runge, dal nome del matematico tedesco Carl David Tolmè Runge (1856-1927) che usò proprio questa funzione per evidenziare il comportamento dell'errore di interpolazione con punti fondamentali equidistanti [32]. Poiché abbiamo la funzione in forma analitica, possiamo provare a cambiare distribuzione dei punti fondamentali; scegliamo i punti di Chebyshev (5.14), sapendo che al crescere di n questi punti si addensano particolarmente verso gli estremi dell'intervallo $[-1, 1]$ come mostra la tabella la tabella 5.1 nella quale riportiamo i punti $x_0, \ldots x_{n/2}$ per alcuni valori di n ($x_{n/2+1}$ è sempre uguale a zero e i restanti punti sono simmetrici rispetto a quelli riportati).

La figura 5.4 mostra che con questa scelta dei punti fondamentali l'errore diminuisce su tutto l'intervallo all'aumentare del grado del polinomio interpolante.

■

Quanto visto in questo esempio relativamente alla funzione di Runge trova la sua generalizzazione in importanti risultati teorici. Usando la for-

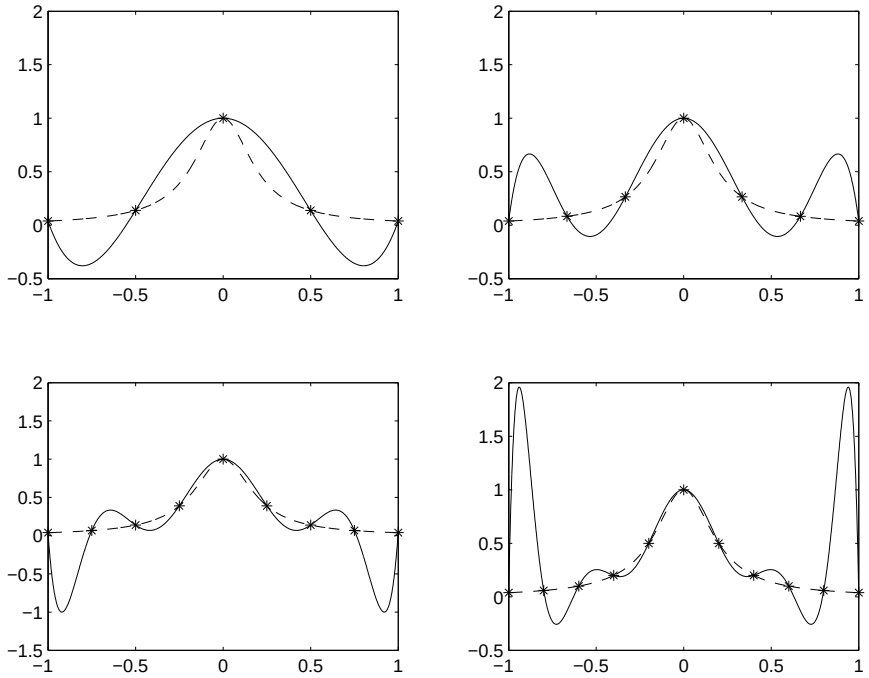

Figura 5.3 Interpolazione della funzione di Runge su punti fondamentali equidistanti

$n=4$	$n=6$	$n=8$	$n=10$
0.9511	0.9749	0.9848	0.9898
0.5878	0.7818	0.8660	0.9096
	0.4339	0.6428	0.7557
		0.3420	0.5406
			0.2817

Tabella 5.1 Punti di Chebyshev

mulazione di Newton (5.18) del polinomio interpolante possiamo agilmente trovare un'espressione dell'errore di interpolazione. Dato $N_n(x)$, aggiungiamo formalmente ai punti fondamentali il punto $(\hat{x}, f(\hat{x}))$ pur sapendo che il valore $f(\hat{x})$ non è noto. Allora il polinomio interpolante diventa

$$N_{n+1}(x) = N_n(x) + f[x_0, x_1, \ldots, x_n, \hat{x}]\omega_{n+1}(x),$$

con $\omega_{n+1}(x) = \Pi_{i=0}^{n}(x - x_i)$. Poiché per definizione $N_{n+1}(\hat{x}) = f(\hat{x})$ possiamo scrivere

$$f(\hat{x}) = N_n(\hat{x}) + f[x_0, x_1, \ldots, x_n, \hat{x}]\omega_{n+1}(\hat{x})$$

e quindi

$$E_n(\hat{x}) = f[x_0, x_1, \ldots, x_n, \hat{x}]\omega_{n+1}(\hat{x}). \tag{5.22}$$

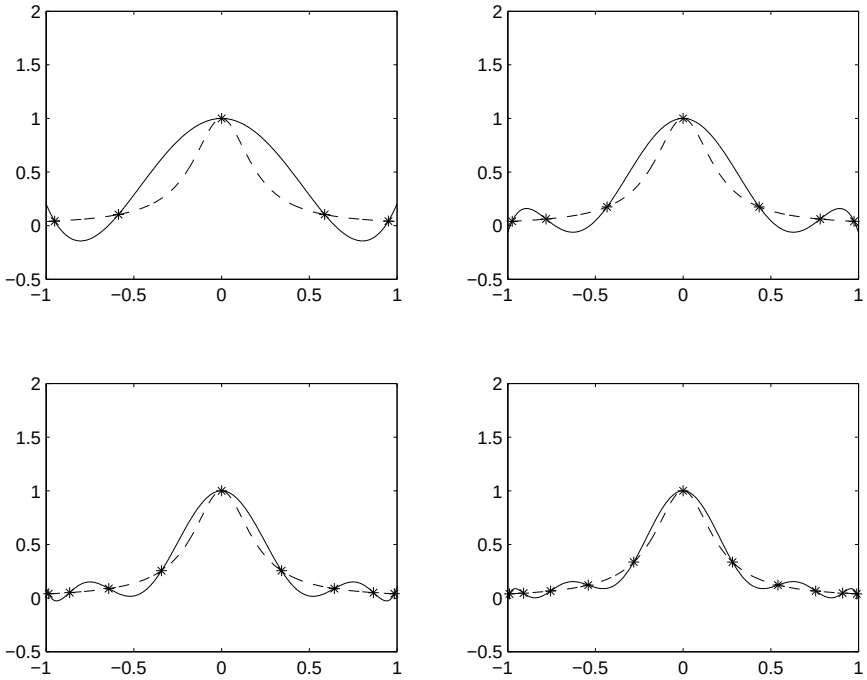

Figura 5.4 Interpolazione della funzione di Runge su punti di Chebyshev

Se supponiamo che f sia derivabile $n+1$ volte in $[a,b]$, allora possiamo dedurre [15] una diversa espressione dell'errore nella quale non interviene una differenza divisa di ordine $n+1$ ma la derivata $(n+1)$–esima:

$$E_n(\hat{x}) = \frac{f^{(n+1)}}{(n+1)!}(\xi)\omega_{n+1}(\hat{x})$$

con $\min\{x_0, x_1, \ldots, x_n, \hat{x}\} < \xi < \max\{x_0, x_1, \ldots, x_n, \hat{x}\}$. Questo mette in evidenza il legame che esiste fra le differenze divise e le derivate. Usando una qualunque delle due forme dell'errore notiamo che, come è lecito attendersi, l'errore dipende sia dal comportamento della funzione f, attraverso il fattore $f[x_0, x_1, \ldots, x_n, \hat{x}]$ o $\frac{f^{(n+1)}}{(n+1)!}(\xi)$, sia dai punti fondamentali, presenti nel suddetto fattore e in $\omega_{n+1}(\hat{x})$. Possiamo allora chiederci se esiste una scelta di questi punti che permetta di mantenere piccolo l'errore, indipendentemente dalla funzione. Poiché il fattore $\omega_{n+1}(\hat{x})$ dipende soltanto dai punti fondamentali e non da f ci si può chiedere più precisamente se esiste una distribuzione di punti che renda minima la quantità $\max_{x\in[a,b]} |\omega_{n+1}(x)|$. La risposta a questa domanda è positiva e si dimostra [15] che i punti di Chebyshev generalizzati (5.15) rappresentano la scelta ottimale qualunque sia l'intervallo di interpolazione $[a,b]$.

Torniamo all'espressione (5.22) dell'errore di interpolazione. Pur trattandosi di un'espressione teorica che, dipendendo dal valore incognito $f(\hat{x})$, non può essere valutata esattamente, essa può risultare utile per ottenere in pratica una stima di $E_n(\hat{x})$ usando un'approssimazione, eventualmente grossolana, di $f(\hat{x})$. Ad esempio, una volta individuato l'intervallo $[x_i, x_{i+1}]$ a cui $\hat{x}$ appartiene, si può approssimare $f[x_0, x_1, \ldots, x_n, \hat{x}]$ usando al posto di $f(\hat{x})$ il valore $P_1(\hat{x})$, dove P_1 è il polinomio di primo grado che interpola i punti (x_i, f_i) e (x_{i+1}, f_{i+1}).

Esempio 5.2.9. Consideriamo ancora una volta i punti dell'esempio 5.2.1

$$(x_0, f_0) = (-1, -3) \quad (x_1, f_1) = (1, 0) \quad (x_2, f_2) = (2, 4),$$

che sono interpolati dal polinomio $N_2(x) = \frac{5}{6}x^2 + \frac{3}{2}x - \frac{7}{3}$. Per $\hat{x} = 0$ si ha $N_2(0) = -\frac{7}{3}$ e vogliamo calcolare una stima dell'errore $E_2(0)$, sapendo che esso è uguale a $f[-1, 1, 2, 0](0+1)(0-1)(0-2) = 2f[-1, 1, 2, 0]$. Per stimare la differenza divisa $f[-1, 1, 2, 0]$, osserviamo che $0 \in [-1, 1]$ e che la retta passante per $(-1, -3)$ e $(1, 0)$ è $y = \frac{3}{2}(x + 1) - 3$; quindi approssimiamo $f(0)$ con $= -\frac{9}{2}$ e completiamo la tabella 5.21 aggiungendo il punto $(0, -\frac{9}{2})$. In questo modo si ottiene $E_2(0) \simeq -2 \times \frac{13}{12} = \frac{13}{6}$. ■

5.3 Interpolazione mediante funzioni spline (*)

Abbiamo visto che i punti di Chebyshev (generalizzati) sono ottimali per l'interpolazione polinomiale, sia dal punto di vista del condizionamento del problema che da quello dell'errore di interpolazione. Ma se la funzione f non è data in forma analitica, come succede ad esempio nell'interpolazione di dati sperimentali, non possiamo scegliere i punti in cui interpolare. In questa situazione potremmo decidere di scegliere pochi punti fra quelli disponibili e usare un polinomio interpolante di grado basso (l'esperienza insegna che in generale, usando punti equidistanti o comunque non scelti ad hoc, non è consigliabile utilizzare polinomi di grado superiore a 5), ma questo non darà in generale una buona approssimazione della funzione f su tutto l'intervallo. Quando si dispone di molti dati conviene seguire una strada diversa, che consente di utilizzarli tutti pur lavorando con polinomi di grado basso. La strategia consiste nell'usare interpolanti polinomiali a tratti.

Consideriamo ad esempio la figura 5.5 relativa alla funzione di Runge. Qui abbiamo diviso l'intervallo $[-1, 1]$ in $n = 4, 6, 8, 10$ sottointervalli di uguale ampiezza come nell'esempio 5.2.8 ma, a differenza di quanto fatto in quel contesto, per ogni sottointervallo abbiamo costruito il polinomio di primo grado che interpola la funzione nei due estremi. Abbiamo fatto cioè un'interpolazione lineare a tratti. Confrontando le figure 5.3 e 5.5, vediamo che le interpolanti a tratti forniscono approssimazioni migliori di quelle

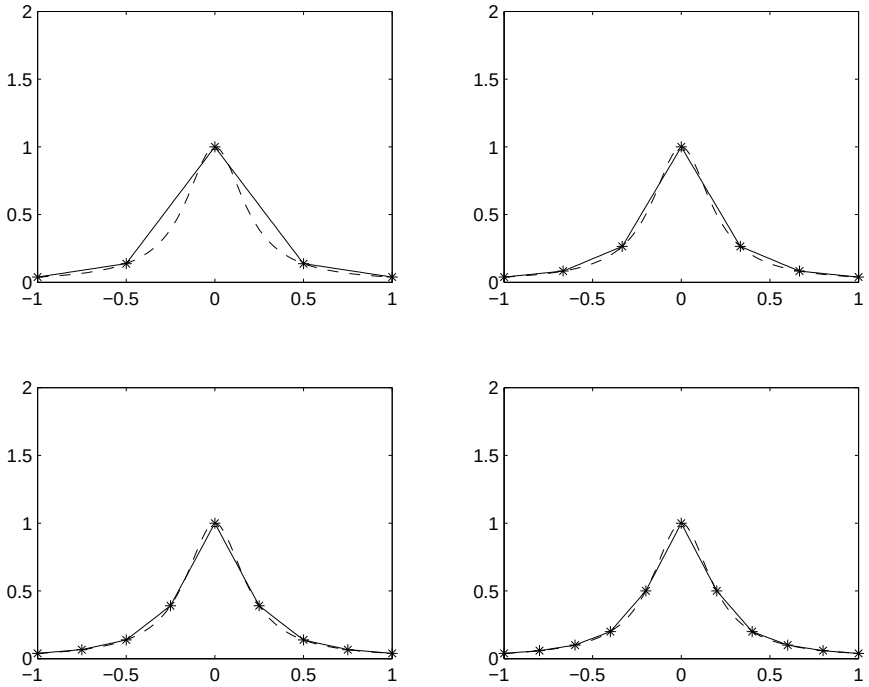

Figura 5.5 Interpolazione della funzione di Runge mediante lineari a tratti

date dai polinomi interpolanti. D'altra parte il raccordo dei vari tratti lineari genera punti angolosi e complessivamente la funzione approssimante risulta continua ma con derivata prima discontinua. Questo può risultare un inconveniente in talune situazioni, vuoi dal punto di vista estetico perché si necessita di un grafico senza spigoli, vuoi dal punto di vista teorico perché si vuol determinare una funzione approssimante con un certo grado di regolarità. Per evitare questo problema continuando a usare funzioni polinomiali a tratti di grado basso, occorre utilizzare le funzioni spline di grado superiore al primo.

5.3.1 Funzioni spline

Dato un intervallo $[a, b]$, fissiamo m punti ad esso interni $y_1 < y_2 < \ldots < y_m$; poniamo poi per comodità $y_0 = a$ e $y_{m+1} = b$, cosí che possiamo vedere l'intervallo $[a, b]$ suddiviso in $(m + 1)$ sottointervalli disgiunti $I_0, I_1, \ldots, I_m$ definiti da

$$I_i = [y_i, y_{i+1}) \text{ per } i = 0, 1, \ldots, m-1$$

$$I_m = [y_m, y_{m+1}].$$

Si definisce funzione spline , o semplicemente spline, di grado p e nodi y_0, y_1, ..., y_{m+1} una funzione polinomiale a tratti tale che su ogni sottointer-

vallo I_i essa coincide con un polinomio di grado p e i tratti di polinomio si raccordano nei nodi in modo che la funzione sia continua su tutto $[a, b]$ insieme alle sue derivate fino a quella di ordine $p - 1$. Il termine "spline" in inglese indica una "bacchetta flessibile di legno, gomma o metallo, usata per disegnare curve". In altri termini, la funzione spline descrive matematicamente lo strumento spline che fino dall'antichità i costruttori delle carene delle navi usavano per tracciarne il profilo. I nodi sono i punti in corrispondenza dei quali lo strumento si piega [6].

Nel seguito indicheremo una spline con il simbolo $S_{p,y}$ e quindi, per definizione, $S_{p,y}$ soddisfa le seguenti proprietà:

$$S_{p,y} \in \mathcal{P}_p, \quad \text{per} \quad x \in I_i, \quad i = 0, 1, \ldots, m,$$

$$S_{p,y} \in C^{p-1}[a, b],$$

dove $\mathcal{P}_p$ indica l'insieme dei polinomi di grado p. Le funzioni lineari a tratti della figura 5.5 sono splines di primo grado i cui nodi coincidono con i punti fondamentali dell'interpolazione; una spline quadratica ($p = 2$) su 3 nodi y_0, y_1 e y_2 è costituita da due archi di parabola che si raccordano in y_1 in modo tale che la funzione e la sua derivata prima siano continue; una spline cubica ($p = 3$) è data da m tratti polinomiali di terzo grado che si raccordano l'uno con l'altro in modo tale da dare una funzione continua su tutto $[a, b]$, insieme alla sua derivata prima e alla derivata seconda.

Chiediamoci quanti gradi di libertà ci sono per individuare univocamente una spline, fissati p ed m. Data la definizione di spline, in ognuno dei nodi interni $y_1, y_2, \ldots, y_m$ devono valere le condizioni di raccordo per la funzione e le sue derivate:

$$\lim_{x \to y_i^-} S_{p,y}^{(j)}(x) = \lim_{x \to y_i^+} S_{p,y}^{(j)}(x), \quad \text{per} \quad j = 0, \ldots, p - 1,$$

per un totale di $m \times p$ condizioni. D'altra parte, in ognuno degli $(m + 1)$ sottointervalli $I_0, I_1, \ldots, I_m$ la spline è rappresentata da un polinomio di grado p ed è quindi definita mediante $(p+1)$ coefficienti. Allora $S_{p,y}$ dipende complessivamente da $(m+1)\times(p+1)$ coefficienti. La differenza fra il numero

[6] Probabilmente le funzioni spline sono state usate per la prima volta nell'ambito dell'approssimazione di funzioni da C. Runge [32] nel 1901. La definizione e lo studio sistematico di queste funzioni è però dovuto a Isaac Jacob Schoenberg (1903-1990) che iniziò ad occuparsene nel 1943, quando lavorava presso la Moore School della Università di Pennsylvania e presso il Ballistic Research Laboratory negli Stati Uniti. Il motivo che lo fece avvicinare a queste problematiche fu la necessità di studiare in modo accurato e stabile le traiettorie dei proiettili usando il primo calcolatore elettronico chiamato ENIAC. Il primo suo articolo sull'interpolazione mediante splines fu pubblicato nel 1946 e per molti anni egli continuò ad occuparsene da solo. Dopo gli anni '60, con l'evolversi dei calcolatori elettronici e il conseguente evolversi del software per il CAGD (Computer Aided Geometric Design), lo studio delle splines è diventato molto più popolare.

di coefficienti e il numero di condizioni di raccordo

$$(m+1)\times(p+1) - m\times n = m+p+1 \tag{5.23}$$

fornisce il numero di gradi di libertà. In pratica allora, una volta fissati i nodi, per individuare una particolare spline si devono fornire $m+p+1$ condizioni. Ad esempio, per individuare in modo univoco una spline di primo grado con tre nodi occorrono 3 condizioni, essendo $p = m = 1$; invece una spline quadratica con 3 nodi ha 4 gradi di libertà e una spline cubica con 3 nodi ne ha 5.

Vediamo ora come possiamo scrivere un'espressione esplicita di $S_{p,y}$. Procederemo ad una costruzione intuitiva, rimandando a [12] e [15] per una trattazione rigorosa. Supponiamo per esempio di avere 4 nodi

$$a = y_0 < y_1 < y_2 < y_3 = b$$

e costruiamo la spline da sinistra a destra, cominciando dall'intervallo $[y_0, y_1)$. In questo intervallo la spline è rappresentata da un polinomio di grado p che indichiamo con $P_p(x)$; si ha quindi

$$S_{p,y}(x) = P_p(x) \ \text{ per } \ x \in [y_0, y_1).$$

Passiamo all'intervallo successivo $[y_1, y_2)$. Possiamo pensare che la spline sia rappresentata in $[y_0, y_2)$ come

$$S_{p,y}(x) = P_p(x) + c_1\widetilde{P}_1(x)$$

dove c_1 è un coefficiente e $\widetilde{P}_1(x)$ è una funzione che vale zero per $x \in [y_0, y_1)$ e si identifica con un polinomio di grado p su $[y_1, y_2)$. Questo può essere ottenuto ponendo

$$\widetilde{P}_1(x) = (x-y_1)_+^p = \begin{cases} 0 & x < y_1 \\ (x-y_1)^p & x \geq y_1. \end{cases}$$

Analogamente, inglobando l'ultimo sottointervallo $[y_2, y_3]$, possiamo pensare che la spline abbia in $[a, b] = [y_0, y_3]$ la forma

$$S_{p,y}(x) = P_p(x) + c_1\widetilde{P}_1(x) + c_2\widetilde{P}_2(x)$$

dove c_2 è un altro coefficiente e $\widetilde{P}_2(x)$ vale zero in $[y_0, y_2)$ e coincide con un polinomio di grado p in $[y_2, y_3]$; di nuovo, possiamo prendere

$$\widetilde{P}_2(x) = (x-y_2)_+^p = \begin{cases} 0 & x < y_2 \\ (x-y_2)^p & x \geq y_2 \end{cases}$$

e quindi la nostra spline avrà l'espressione

$$S_{p,y}(x) = P_p(x) + c_1(x-y_1)_+^p + c_2(x-y_2)_+^p.$$

Generalizzando al caso di m qualunque, possiamo scrivere $S_{p,y}$ nella forma

$$S_{p,y}(x) = \sum_{i=0}^{p} a_i x^i + \sum_{i=1}^{m} c_i (x - y_i)_+^p, \tag{5.24}$$

dove

$$(x - y_i)_+^p = \begin{cases} 0 & x < y_i \\ (x - y_i)^p & x \geq y_i \end{cases} \tag{5.25}$$

prende il nome di potenza troncata di grado p relativa al nodo y_i. Notiamo che nell'espressione (5.24) compaiono $m + p + 1$ coefficienti, tanti quanti sono i gradi di libertà (cfr.(5.23)).

5.3.2 Funzioni spline interpolanti

Il problema dell'interpolazione mediante splines si pone nei termini seguenti: dato un intervallo $[a, b]$ nel quale siano fissati $m + 2$ nodi $y_0 = a < y_1 \ldots < y_m < y_{m+1} = b$ e dati $n + 1$ punti $(x_0, f_0), (x_1, f_1), \ldots, (x_n, f_n)$ con $x_0, x_1, \ldots, x_n$ in $[a, b]$, determinare una spline $S_{p,y}$ per cui

$$S_{p,y}(x_i) = f_i, \quad , \quad i = 0, 1, \ldots, n.$$

Poiché il numero di condizioni di interpolazione $(n+1)$ non può superare il numero di gradi di libertà $(m+p+1)$, se vogliamo essere certi dell'esistenza della spline dobbiamo supporre che m, p e n soddisfino la relazione

$$n \leq m + p.$$

Per quanto riguarda l'insieme dei nodi e l'insieme dei punti fondamentali dell'interpolazione, essi possono essere totalmente distinti, parzialmente coincidenti oppure totalmente coincidenti. È comunque ragionevole che ci debba essere un legame tra la posizione dei nodi e quella dei punti fondamentali. Rimandando a [12] per una trattazione rigorosa, ci limitiamo ad un esempio. Consideriamo una spline lineare: se in un intervallo $[y_i, y_{i+1})$ cadono due punti di interpolazione, in tale intervallo viene definita univocamente una retta interpolante; supponiamo ora che nell'intervallo contiguo $[y_{i+1}, y_{i+2})$ ci siano altri due punti fondamentali e quindi, di nuovo, un tratto di retta univocamente determinato. A questo punto, non c'è modo di imporre il raccordo tra i due tratti di retta adiacenti.

Esempio 5.3.1. Dato l'intervallo $[0, 5]$, fissiamo i nodi $y_0 = 0$, $y_1 = 1$, $y_2 = 2$, $y_3 = 3$, $y_4 = 4$ e $y_5 = 5$. Vogliamo determinare una spline $S_{1,y}$ di primo grado con questi nodi interpolante i punti

$$(x_0, f_0) = (\tfrac{1}{3}, 1) \quad (x_1, f_1) = (\tfrac{2}{3}, 2) \quad (x_2, f_2) = (2, 1)$$

$$(x_3, f_3) = (\tfrac{5}{2}, \tfrac{1}{2}) \quad (x_4, f_4) = (\tfrac{7}{2}, 2) \quad (x_5, f_5) = (5, 1).$$

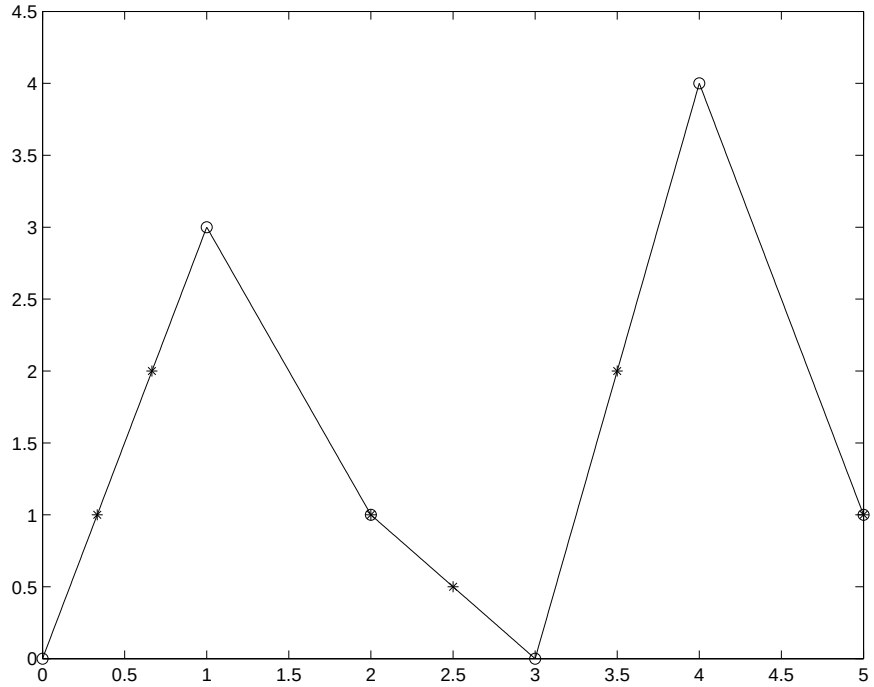

Figura 5.6 Esempio 5.3.1: spline di grado 1 interpolante

Vedremo che la spline cercata è unica ed è quella rappresentata nella figura 5.6, dove con il simbolo '*' indichiamo i dati (x_i, f_i) con $i = 0, \ldots, 5$ e con 'o' i punti $(y_j, S_{1,y}(y_j))$, $j = 0, \ldots, 5$.

La spline è esprimibile nella forma

$$S_{1,y}(x) = a_0 + a_1 x + c_1(x-1)_+ + c_2(x-2)_+ + c_3(x-3)_+ + c_4(x-4)_+.$$

Abbiamo quindi 6 coefficienti da determinare e 6 condizioni di interpolazione. Imponendo queste condizioni e tenendo conto della definizione di potenza troncata (5.25) otteniamo il seguente sistema lineare

$$\begin{cases} a_0 + \frac{1}{3}a_1 & = & 1 \\ a_0 + \frac{2}{3}a_1 & = & 2 \\ a_0 + 2a_1 + c_1 & = & 1 \\ a_0 + \frac{5}{2}a_1 + \frac{3}{2}c_1 + \frac{1}{2}c_2 & = & \frac{1}{2} \\ a_0 + \frac{1}{2}a_1 + \frac{5}{2}c_1 + \frac{3}{2}c_2 + \frac{1}{2}c_3 & = & 2 \\ a_0 + 5a_1 + 4c_1 + 3c_2 + 2c_3 + c_4 & = & 1 \end{cases}$$

che ammette come unica soluzione $a_0 = 0$, $a_1 = 3$, $c_1 = -5$, $c_2 = 1$, $c_3 = 5$, $c_4 = -7$. Sciogliendo la definizione di potenza troncata, possiamo trovare le espressioni di $S_{1,y}(x)$ sottointervallo per sottointervallo, nella

forma seguente:

$$S_{1,y}(x) = \begin{cases} 3x & x \in [0,1) \\ -2x+5 & x \in [1,2) \\ -x+3 & x \in [2,3) \\ 4x-12 & x \in [3,4) \\ -3x+16 & x \in [4,5] \end{cases}$$

da cui possiamo verificare la continuità della funzione. ■

Esempio 5.3.2. Siano dati l'intervallo $[1,4]$ e i nodi $y_0 = 1$, $y_1 = 2$ e $y_2 = 4$. Vogliamo determinare una spline $S_{2,y}$ di grado 2 interpolante i dati

$$(x_0, f_0) = (1,1) \quad (x_1, f_1) = (2, \tfrac{1}{2})$$

$$(x_2, f_2) = (3,2) \quad (x_3, f_3) = (4, \tfrac{1}{4}).$$

La funzione spline in questione è esprimibile nella forma

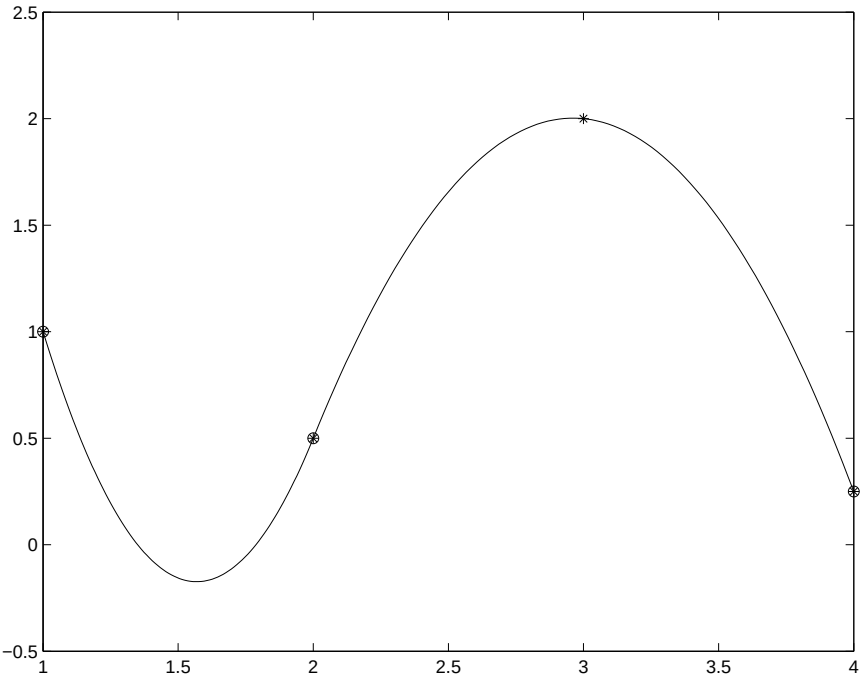

Figura 5.7 Esempio 5.3.2: spline quadratica interpolante

$$S_{2,y}(x) = a_0 + a_1 x + a_2 x^2 + c_1 (x-2)_+^2.$$

Imponendo le condizioni di interpolazione si ottiene il seguente sistema lineare

$$\begin{cases} a_0 + a_1 + a_2 & = & 1 \\ a_0 + 2a_1 + 4a_2 & = & \frac{1}{2} \\ a_0 + 3a_1 + 9a_2 + c_1 & = & 2 \\ a_0 + 4a_1 + 16a_2 + 4c_1 & = & \frac{1}{4} \end{cases}$$

la cui risoluzione porta alla spline

$$S_{2,y}(x) = \frac{35}{4} - \frac{91}{8}x + \frac{29}{8}x^2 - \frac{21}{4}(x-2)_+^2$$

rappresentata in figura 5.7. Per verificare che abbiamo realmente trovato una spline dobbiamo verificare che la funzione trovata è $C^1[1,4]$, ovvero che i due archi di parabola si raccordano nel nodo interno $y_1 = 2$ fino alla derivata prima. In effetti abbiamo

$$\lim_{x\to 2^-} S_{2,y}(x) = \lim_{x\to 2^+} S_{2,y}(x) = \frac{1}{2}$$

e

$$\lim_{x\to 2^-} S'_{2,y}(x) = \lim_{x\to 2^+} S'_{2,y}(x) = \frac{25}{8}.$$

■

Come abbiamo visto negli esempi, l'espressione (5.24) delle funzioni spline mediante potenze troncate è molto semplice da utilizzare, ma presenta alcuni svantaggi. Uno dei principali inconvenienti, più evidente quando sono presenti molti nodi, consiste nel fatto che la valutazione della spline in un punto dell'intervallo $[a,b]$ richiede un costo computazionale crescente man mano che ci si sposta da a verso b. Questo è insito nella definizione di potenza troncata, che diventa diversa da zero a partire dal nodo a cui si riferisce. Per questo motivo, e anche per altri inconvenienti legati a questioni di stabilità, la formulazione con le potenze troncate non viene abitualmente usata e si preferisce una formulazione alternativa, mediante le cosiddette B-splines. La trattazione di questi aspetti va oltre gli scopi di questo testo e rimandiamo chi fosse interessato a [12].

5.3.3 Splines cubiche interpolanti nei nodi

Le splines più comunemente usate nei problemi applicativi di interpolazione sono le splines cubiche; infatti esse sono, per definizione, continue fino alla derivata seconda e questo risulta essere soddisfacente nella maggior parte delle situazioni. Basti pensare che le discontinuità, se presenti, riguardano le derivate dalla terza in poi e non sono quindi visibili ad occhio nudo in un grafico. In particolare, sono spesso usate le splines cubiche interpolanti

nei nodi, cioè splines cubiche per le quali i nodi e i punti fondamentali dell'interpolazione coincidono. Per il calcolo di queste splines esiste un procedimento ad hoc che costituisce una valida alternativa a quello basato sulle potenze troncate.

Il problema che abbiamo di fronte è il seguente: assegnati i dati (x_i, f_i), con $i = 0, \ldots, n$ tali che $a = x_0 < x_1 < \ldots < x_n = b$, si vuole determinare una spline cubica che li interpoli, assumendo come nodi i punti fondamentali stessi. Il problema così posto non definisce univocamente una spline; infatti, essendo $p = 3$ ed essendoci $n - 1$ nodi interni, la (5.23) dice che i gradi di libertà sono $n - 1 + 3 + 1 = n + 3$, mentre noi abbiamo soltanto $n + 1$ condizioni da imporre. Dobbiamo allora aggiungere altre due condizioni. In mancanza di altre informazioni, si impone che la derivata seconda della spline si annulli in x_0 e x_n: la spline così ottenuta prende il nome di spline naturale [7]. Le due condizioni aggiuntive possono essere scelte in modo diverso quando si hanno ulteriori informazioni sulla funzione f che stiamo cercando di approssimare, ad esempio quando si conoscono i valori di f' o di f'' negli estremi, o quando si sa che f è una funzione periodica. Qui considereremo soltanto il caso delle splines naturali.

Per descrivere il procedimento di costruzione della spline, introduciamo una nuova notazione e indichiamo con $s_i(x)$ il tratto di spline relativo all'intervallo $[x_i, x_{i+1}]$, per $i = 0, 1, \ldots, n - 1$. Con questa notazione, le condizioni di continuità della derivata prima e seconda diventano:

$$\begin{array}{rcl} s_i'(x_i) & = & s_{i-1}'(x_i) \\ s_i''(x_i) & = & s_{i-1}''(x_i), \end{array} \tag{5.26}$$

per $i = 1, 2, \ldots, n - 1$; le condizioni di interpolazione diventano

$$\begin{array}{rcl} s_i(x_i) & = & f_i \\ s_i(x_{i+1}) & = & f_{i+1}, \end{array} \tag{5.27}$$

per $i = 0, 1, \ldots, n$; infine le condizioni aggiuntive per la spline naturale diventano

$$s_0''(x_0) = s_{n-1}''(x_n) = 0. \tag{5.28}$$

Per costruire la spline in modo efficiente senza ricorrere all'uso delle potenze troncate, dobbiamo prima di tutto far vedere che le funzioni $s_0, s_1, \ldots, s_{n-1}$ possono essere espresse in termini dei valori, che indicheremo con $z_0, z_1, \ldots, z_n$, assunti dalla derivata seconda nei nodi. Poiché lavoriamo con splines naturali, i due valori estremi z_0 e z_n sono entrambi fissati uguali a zero, mentre i valori $z_1, z_2, \ldots, z_{n-1}$ sono incogniti. Per il momento procediamo formalmente come se conoscessimo tutti questi valori. Allora, sapendo che per ogni i la funzione s_i'' è un polinomio di primo grado

[7] Questo nome deriva dal fatto che all'esterno dell'intervallo $[a, b]$ lo strumento spline usato per disegnare tende naturalmente a proseguire in linea retta.

(poiché s_i è un polinomio di terzo grado), possiamo scriverne l'espressione; si tratta infatti di scrivere l'equazione della retta che passa per i due punti (x_i, z_i) e (x_{i+1}, z_{i+1}). Ponendo

$$h_i = x_{i+1} - x_i$$

si ottiene

$$s_i''(x) = \frac{z_{i+1}}{h_i}(x - x_i) + \frac{z_i}{h_i}(x_{i+1} - x). \tag{5.29}$$

Notiamo che, assumendo la forma (5.29), stiamo implicitamente imponendo la continuità della derivata seconda della spline, ossia la seconda delle (5.26). Integrando la (5.29) due volte, otteniamo un'espressione di $s_i(x)$

$$s_i(x) = \frac{z_{i+1}}{6h_i}(x - x_i)^3 + \frac{z_i}{6h_i}(x_{i+1} - x)^3 + C_i(x - x_i) + D_i(x_{i+1} - x) \tag{5.30}$$

in cui compaiono due costanti di integrazione C_i e D_i. Queste costanti possono essere determinate imponendo le condizioni di interpolazione (5.27), da cui si ottiene

$$C_i = \frac{f_{i+1}}{h_i} - \frac{h_i}{6}z_{i+1} \quad \text{e} \quad D_i = \frac{f_i}{h_i} - \frac{h_i}{6}z_i. \tag{5.31}$$

In (5.30) e (5.31) compaiono i valori incogniti z_i e z_{i+1}. È arrivato ora il momento di determinare questi valori, in modo da poter avere un'espressione di $s_i(x)$ spendibile. A questo scopo imponiamo le ultime condizioni che non abbiamo ancora utilizzato, quelle sulla continuità della derivata prima. Derivando la (5.30) per i e $i-1$ si ottiene

$$s_i'(x) = \frac{z_{i+1}}{2h_i}(x - x_i)^2 - \frac{z_i}{2h_i}(x_{i+1} - x)^2 + C_i - D_i$$

e analogamente

$$s_{i-1}'(x) = \frac{z_i}{2h_{i-1}}(x - x_{i-1})^2 - \frac{z_{i-1}}{2h_{i-1}}(x_i - x)^2 + C_{i-1} - D_{i-1}.$$

La prima condizione delle (5.26)) assume pertanto la forma

$$-\frac{h_i}{6}z_{i+1} - \frac{h_i}{3}z_i + \frac{f_{i+1} - f_i}{h_i} = \frac{h_{i-1}}{6}z_{i-1} + \frac{h_{i-1}}{3}z_i + \frac{f_i - f_{i-1}}{h_{i-1}},$$

ovvero

$$h_{i-1}z_{i-1} + 2(h_{i-1} + h_i)z_i + h_i z_{i+1} = b_i \tag{5.32}$$

con

$$\begin{aligned} b_i &= 6(v_i - v_{i-1}) \\ v_i &= \tfrac{f_{i+1} - f_i}{h_i} \quad , \quad v_{i-1} = \tfrac{f_i - f_{i-1}}{h_{i-1}}. \end{aligned}$$

La relazione (5.32) deve valere per $i = 1, 2, \ldots n-1$. Abbiamo così ricavato $n-1$ condizioni che i valori $z_0, z_1, \ldots z_n$ devono soddisfare. Poiché siamo interessati a splines naturali, sostituiamo il valore zero a z_0 e z_n e otteniamo un sistema lineare di $n-1$ equazioni nelle $n-1$ incognite $z_1, z_2, \ldots z_{n-1}$. La matrice dei coefficienti di questo sistema è

$$A = \begin{pmatrix} d_1 & h_1 & 0 & 0 & \ldots & 0 \\ h_1 & d_2 & h_2 & 0 & \ldots & 0 \\ 0 & h_2 & d_3 & h_3 & \ldots & 0 \\ \vdots & & & & & \\ 0 & \ldots & 0 & h_{n-3} & d_{n-2} & h_{n-2} \\ 0 & \ldots & 0 & 0 & h_{n-2} & d_{n-1} \end{pmatrix} \tag{5.33}$$

con $d_i = 2(h_{i-1}+h_i)$, per $i = 1, 2, \ldots, n-1$. Questa matrice ha una struttura molto particolare: è tridiagonale, simmetrica e inoltre è a predominanza diagonale (cfr. (4.32)). In base al teorema che riportiamo qui di seguito, A è non singolare e quindi il sistema ammette un'unica soluzione. In altri termini, la spline cubica interpolante nei nodi esiste ed è unica.

Teorema 5.3.1. Sia $A \in \mathbb{R}^{m\times m}$ una matrice a predominanza diagonale per righe, ovvero tale che

$$|a_{ii}| > \sum_{j=1, j\neq i}^{m} |a_{ij}| \qquad \forall i = 1, \ldots, m.$$

Allora, A è non singolare.

Dimostrazione. Supponiamo per assurdo che A sia singolare. Allora esiste un vettore $x \in \mathbb{R}^m$ diverso da zero tale che $Ax = 0$. Fissata una componente $x_k \neq 0$ di x possiamo scrivere

$$\sum_{j=1}^{m} a_{kj}x_j = 0 \Rightarrow a_{kk}x_k + \sum_{j=1, j\neq k}^{m} a_{kj}x_j = 0$$

e, dividendo per x_k, si ottiene

$$a_{kk} = \sum_{j=1, j\neq k}^{m} \frac{a_{kj}x_j}{x_k}.$$

Se scegliamo k in modo che $|x_k| = \|x\|_\infty$, i rapporti $\frac{|x_j|}{|x_k|}$ con $j \neq k$ sono tutti minori o uguali di 1. Applicando la disuguaglianza triangolare si ottiene allora

$$|a_{kk}| \leq \sum_{j=1, j\neq i}^{m} |a_{kj}|$$

che è in contraddizione con le ipotesi. □

Lo studio del condizionamento del problema di interpolazione con splines cubiche interpolanti nei nodi, ovvero del condizionamento della matrice (5.33), è abbastanza complesso. Qui ci limitiamo a mostrare alcuni esempi che illustrano gli aspetti salienti della questione. Il condizionamento dipende dalla distribuzione dei nodi, come già avevamo visto nel caso dell'interpolazione polinomiale, ma la situazione è in un certo senso rovesciata.

Se i nodi non sono distribuiti in modo uniforme, in particolare se alcuni sottointervalli (pochi rispetto al numero totale n) hanno ampiezza molto maggiore degli altri, allora il condizionamento può essere anche molto cattivo. Per esemplificare, consideriamo due numeri positivi δ_1 e δ_2 e i punti

$$\begin{array}{lcll} x_i & = & i\delta_1 & \text{per } i = 0, 1, \ldots, 5 \\ x_6 & = & x_5 + \delta_2 & \\ x_i & = & x_6 + (i-6)\delta_1 & \text{per } i = 7, 8, \ldots, 12 \\ x_{13} & = & x_{12} + \delta_2 & \end{array}$$

che individuano due intervalli di ampiezza δ_2 e undici di ampiezza δ_1. Al variare di δ_1 e δ_2 il numero di condizionamento $k_1(A)$ varia come esemplificato nella tabella 5.2. Se $\delta_2 >> \delta_1$ si ha $k_1(A) \simeq \frac{\delta_2}{\delta_1}$.

δ_1	δ_2	$k_1(A)$
1	1e+05	1.36e+05
1	1e+04	1.36e+04
1e−02	1e+03	1.36e+05
1	1e+06	1.36e+06
1e+04	1	4.72e+00
1e+06	1	4.74e+00
1e+03	1e−02	4.74e+00

Tabella 5.2 Condizionamento delle splines cubiche interpolanti nei nodi per distribuzioni non uniformi

Viceversa, se i nodi sono equidistanti ($h_0 = h_1 = \ldots = h_{n-1} = h$) il problema è ben condizionato qualunque sia la loro distanza h. In questo caso infatti la matrice del sistema diventa $A = h\widetilde{A}$, dove $\widetilde{A}$ è una matrice tridiagonale e simmetrica con elementi diagonali uguali a 4 e elementi codiagonali uguali a 1; di conseguenza $k(A) = k(\widetilde{A})$ in qualunque norma si misuri il condizionamento. Ebbene, il condizionamento di $\widetilde{A}$ è ottimo per qualunque n: il valore di $k_1(\widetilde{A})$ calcolato con la funzione condest di MATLAB è circa uguale a 3 per ogni n.

Per quanto riguarda l'errore di interpolazione, le funzioni splines hanno un comportamento decisamente migliore dei polinomi, come già la figura 5.5

relativa a splines lineari lasciava intravedere. In particolare questo è vero per le splines cubiche interpolanti nei nodi. Si può infatti dimostrare [12] che, qualunque sia la distribuzione dei nodi, quando n tende a ∞ l'errore di interpolazione con queste funzioni tende a zero su tutto l'intervallo $[a, b]$.

Esempio 5.3.3. La figura 5.8 mostra le splines cubiche interpolanti la funzione

$$f(x) = \frac{1}{2} e^x sin^2(x)$$

su $n = 4, 6, 8, 10$ punti equidistanti nell'intervallo $[0, 2\pi]$. Come al solito i punti di interpolazione sono contrassegnati da un * e il grafico di f è disegnato con linea tratteggiata. Vediamo che al crescere di n la spline

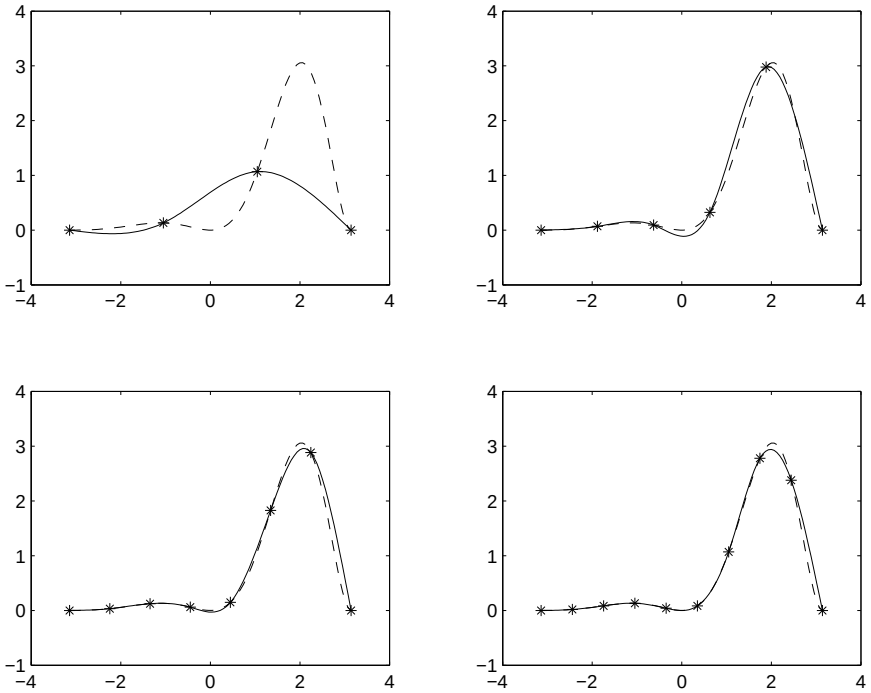

Figura 5.8 Esempio 5.3.3: splines cubiche interpolanti nei nodi

approssima sempre meglio la funzione su tutto l'intervallo. ■

5.3.4 Algoritmi per il calcolo di splines cubiche interpolanti nei nodi

Il fatto che la matrice (5.33) sia tridiagonale, simmetrica e a predominanza diagonale permette di determinare la spline cubica interpolante nei nodi in modo molto efficiente. Infatti si sa [20] che quando si applica il metodo di Gauss a matrici con queste caratteristiche, la tecnica del pivoting

parziale non induce nessuno scambio di righe. Questo implica che possiamo usare direttamente il metodo nella sua forma originale (senza pivoting parziale) la cui realizzazione risulta peraltro notevolmente semplificata a causa della struttura di A: ad ogni passo vengono modificati soltanto un elemento diagonale ed il corrispondente termine noto. Infine, la matrice R a cui si perviene ha una struttura non più triangolare superiore, ma bidiagonale superiore (soltanto la diagonale e la codiagonale superiore contengono elementi diversi da zero). Di tutte queste informazioni facciamo uso nel primo dei due algoritmi seguenti, l'algoritmo 5.3.1, che descrive la costruzione e la risoluzione del sistema lineare per la determinazione di $z = (z_1, z_2, \ldots, z_{n-1})^T$.

Algoritmo 5.3.1. Algoritmo per il calcolo delle derivate seconde $z_1, z_2, \ldots, z_{n-1}$ nei nodi interni per una splina cubica naturale interpolante nei nodi.

Dati: n, $(x_0, f_0), (x_1, f_1), \ldots, (x_n, f_n)$
Risultato: z
1. Per $i = 0, \ldots, n-1$
   1. $h_i = x_{i+1} - x_i$
   2. $v_i = \frac{f_{i+1} - f_i}{h_i}$

   Fine ciclo su i
2. Per $i = 1, \ldots, n-2$
   1. $d_i = 2(h_{i-1} + h_i)$
   2. $b_i = 6(v_i - v_{i-1})$

   Fine ciclo su i
3. $d_{n-1} = 2(h_{n-2} + h_{n-1})$
4. $b_{n-1} = 6(v_{n-1} - v_{n-2})$
5. Per $i = 2, \ldots, n-1$
   1. $d_i = d_i - \frac{h_{i-1}^2}{d_{i-1}}$
   2. $b_i = b_i - \frac{h_{i-1}}{d_{i-1}} b_{i-1}$

   Fine ciclo su i
6. $z_{n-1} = \frac{b_{n-1}}{d_{n-1}}$
7. Per $i = n-2, n-3, \ldots, 1$
   $z_i = \frac{b_i - h_i z_{i+1}}{d_i}$

   Fine ciclo su i
8. Fine

La costruzione della matrice e del termine noto del sistema occupa i passi da 1 a 4; il ciclo al passo 5 realizza il metodo di Gauss senza pivoting per una matrice tridiagonale e simmetrica; infine, ai passi 6 e 7 si risolve il sistema bidiagonale superiore mediante sostituzione all'indietro.

Dato il vettore $z = (0, z_1, z_2, \ldots, z_{n-1}, 0)^T$, il successivo algoritmo 5.3.2 calcola il valore y che la spline cubica assume in un dato punto $\hat{x} \in [x_0, x_n]$: una volta individuato un indice i tale che $\hat{x} \in [x_i, x_{i+1}]$, si calcola $s_i(\hat{x})$. Per il calcolo di y non conviene usare l'espressione (5.30), ma la seguente

forma annidata che richiede meno operazioni aritmetiche:

$$s_i(x) = f_i + (x - x_i)(B_i + (x - x_i)(\frac{z_i}{2} + (x - x_i)\frac{z_{i+1} - z_i}{6h_i})),$$

con

$$B_i = -\frac{h_i}{6}z_{i+1} - \frac{h_i}{3}z_i + \frac{f_{i+1} - f_i}{h_i}.$$

Algoritmo 5.3.2. Algoritmo per il calcolo del valore y che la spline cubica naturale assume in $\hat{x}$.

Dati: n, z,$(x_0, f_0), (x_1, f_1), \ldots, (x_n, f_n)$, $\hat{x}$
Risultato: y
1. Per $i = 0, \ldots, n-1$
    1. Se $x_i \leq \hat{x} \leq x_{i+1}$, allora:
        1. $h = x_{i+1} - x_i$
        2. $t = \hat{x} - x_i$
        3. $B = -\frac{h}{6}z_{i+1} - \frac{h}{3}z_i + \frac{f_{i+1}-f_i}{h}$
        4. $y = f_i + t(B + t(\frac{z_i}{2} + t\frac{z_{i+1}-z_i}{6h}))$
        5. Fermati
    Fine scelta
  Fine ciclo su i
2. Fine

Osserviamo che il risultato y non è definito se il dato in ingresso $\hat{x}$ non appartiene all'intervallo $[x_0, x_n]$. In alcuni linguaggi di programmazione, ad esempio nella versione 7 di MATLAB, questo può in taluni casi determinare una situazione di errore che provoca l'interruzione del programma. Lasciamo per esercizio ai lettori la modifica dell'algoritmo per gestire questa situazione.

5.4 Migliore approssimazione ai minimi quadrati

Come abbiamo accennato nel paragrafo introduttivo 5.1, se i dati di cui disponiamo sono affetti da errori non trascurabili, non è ragionevole interpolarli ed è molto più sensato cercare un modello che ne riproduca in un qualche senso l'andamento complessivo. Per esempio nella figura 5.9 è rappresentata una funzione lineare che approssima un insieme di 50 dati senza interpolarli.

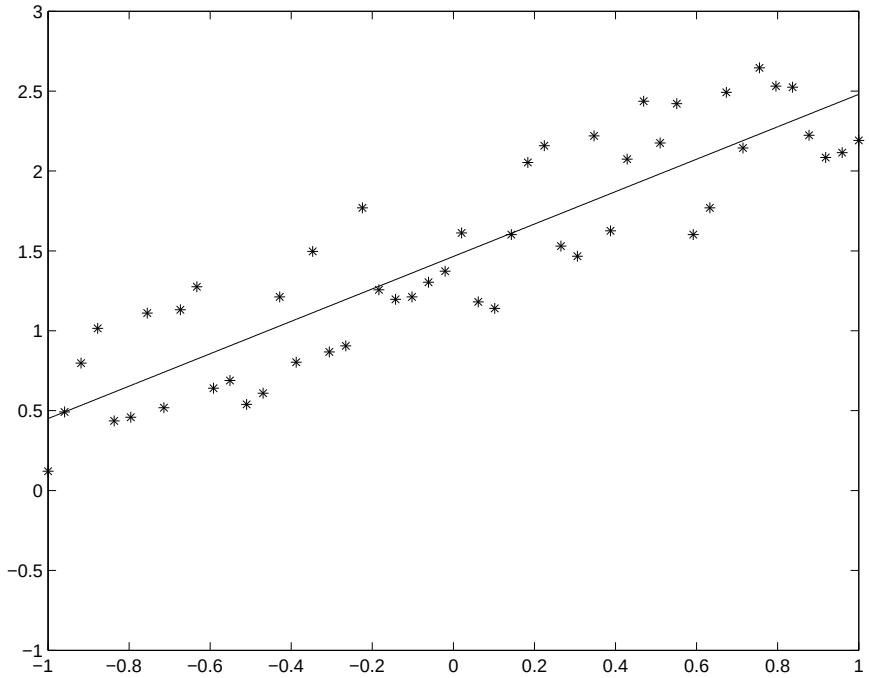

Figura 5.9 Un esempio di retta approssimante

Siano $(x_1, f_1), (x_2, f_2), \ldots, (x_m, f_m)$ i dati e supponiamo di voler utilizzare come modello un polinomio $P_n(x)$ di grado n. In corrispondenza di ogni dato (x_i, f_i) definiamo lo scarto

$$E_i = P_n(x_i) - f_i$$

fra il polinomio P_n e la funzione f. Ogni E_i dipende dai coefficienti $a_0, a_1, \ldots, a_n$ di P_n e per scegliere un particolare polinomio si deve stabilire un criterio che tenda a rendere più piccoli possibile gli scarti $E_1, E_2, \ldots, E_m$ nel loro insieme.

I tre criteri più comunemente usati ricercano i coefficienti $a_0, a_1, \ldots, a_n$ che rendono minima la quantità

$$\sqrt{\sum_{i=1}^{m} E_i^2} \tag{5.34}$$

oppure

$$\max_{1 \le i \le m} |E_i| \tag{5.35}$$

oppure

$$\sum_{i=1}^{m} |E_i|. \tag{5.36}$$

In pratica, se raccogliamo gli scarti in un vettore $E = (E_1, E_2, \ldots, E_m)^T$, si tratta di individuare i valori dei coefficienti $a_0, a_1, \ldots, a_n$ che minimizzano $\|E\|_2$, $\|E\|_\infty$ oppure $\|E\|_1$.

La scelta fra questi tre criteri viene formulata di solito in base a considerazioni di natura statistica, che esulano da questo testo e per le quali rimandiamo a [7] e [9]. Ci limitiamo qui ad osservare che, ad esempio, il criterio basato su (5.35) potrebbe dare molto peso a pochi punti, magari poco significativi, che siano lontani da tutti gli altri: questo può essere un bene o un male a seconda delle situazioni. Inoltre, (5.35) e (5.36) portano a dover risolvere problemi di minimo per funzioni non derivabili in tutto il loro dominio, problemi non facilmente trattabili dal punto di vista numerico. L'unico criterio di cui ci occuperemo è quello basato su (5.34), che è comunque di gran lunga il più usato. Quando si adotta questo criterio, si parla di migliore approssimazione ai minimi quadrati. Nei prossimi paragrafi vedremo che individuare un polinomio di migliore approssimazione ai minimi quadrati equivale a risolvere un opportuno sistema nelle incognite a_0, a_1, ..., a_n; questo sistema è un sistema lineare perché i polinomi dipendono linearmente dai coefficienti. Ci preme notare che il problema della migliore approssimazione può essere formulato non solo per modelli polinomiali ma anche per funzioni spline. Concettualmente questa scelta non porta alcun cambiamento perchè anche le splines dipendono linearmente dai coefficienti sia che le si esprimano mediante potenze troncate (cfr. 5.24)) sia che si usino le B-splines; le differenze si trovano ovviamente a livello di calcoli, come si può vedere in [12].

5.4.1 Retta di migliore approssimazione

Consideriamo il caso più semplice, nel quale si vuole determinare la retta di migliore approssimazione ai minimi quadrati [8]. In questo caso si ha

$$E_i = a_0 + a_1 x_i - f_i$$

per $i = 1, 2, \ldots, m$ e si vogliono quindi determinare i valori dei coefficienti a_0 e a_1 che risolvono il seguente problema di minimo:

$$\min_{a_0, a_1} \frac{1}{2} \sum_{i=1}^{m} |E_i|^2.$$

[8] Questo problema è anche noto come problema della regressione lineare. Il termine "regressione" risale allo studioso Francis Galton (1822-1911), antropologo interessato a problemi di eugenetica, che fece uso di questo procedimento per studiare un particolare fenomeno biologico, in base al quale individui con caratteristiche eccezionali tendono mediamente a generare figli che non possiedono tali caratteristiche, ma che anzi regrediscono verso le caratteristiche più normali di antenati più remoti. La prima applicazione del procedimento di migliore approssimazione ai minimi quadrati risale a Adrien Marie Legendre (1752-1833), il quale coniò l'espressione minimi quadrati (moindres carrées).

Si può notare che abbiamo eliminato la radice quadrata dalla funzione da minimizzare; questa è un'operazione lecita perché i punti di minimo di $\|E\|_2$ sono tutti e soli i punti di minimo di $\|E\|_2^2$. Abbiamo inoltre moltiplicato per il fattore $\frac{1}{2}$ per semplificare i calcoli successivi.

In pratica dobbiamo determinare il punto di minimo di una funzione quadratica di due variabili

$$\mathcal{S}(a_0, a_1) = \frac{1}{2}\sum_{i=1}^{m}(a_0 + a_1 x_i - f_i)^2,$$

che ha gradiente

$$\begin{pmatrix} \frac{\partial \mathcal{S}}{\partial a_0} \\ \\ \frac{\partial \mathcal{S}}{\partial a_1} \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{m}(a_0 + a_1 x_i - f_i) \\ \\ \sum_{i=1}^{m}(a_0 + a_1 x_i - f_i)x_i \end{pmatrix}$$

e matrice hessiana

$$H = \begin{pmatrix} m & \sum_{i=1}^{m} x_i \\ \\ \sum_{i=1}^{m} x_i & \sum_{i=1}^{m} x_i^2 \end{pmatrix}.$$

Vogliamo dimostrare che H è definita positiva, ovvero $z^T H z > 0$ per ogni vettore $z \neq 0$. A questo scopo osserviamo che $H = C^T C$, dove

$$C = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \\ 1 & x_m \end{pmatrix}$$

e quindi

$$z^T H z = z^T C^T C z = \|Cz\|_2^2.$$

Supponendo ancora valida l'ipotesi $x_i \neq x_j$ per $i \neq j$, le colonne di C sono linearmente indipendenti e $Cz \neq 0$ per ogni $z \neq 0$; questo basta per affermare che H è definita positiva. Questo risultato ci permette di affermare che $\mathcal{S}$ è una funzione convessa; il suo (unico) punto di minimo può essere calcolato uguagliando a zero il gradiente, cioè risolvendo il sistema lineare di due equazioni nelle due incognite a_0 e a_1:

$$\begin{cases} m a_0 + \left(\sum_{i=1}^{m} x_i\right) a_1 & = & \sum_{i=1}^{m} f_i \\ \\ \left(\sum_{i=1}^{m} x_i\right) a_0 + \left(\sum_{i=1}^{m} x_i^2\right) a_1 & = & \sum_{i=1}^{m} f_i x_i \end{cases} \tag{5.37}$$

Detta (a_0^*, a_1^*) la soluzione del sistema, la retta di migliore approssimazione ha equazione $y = a_0^* + a_1^* x$. Si può facilmente vedere che questa retta passa per il baricentro dei dati, che ha coordinate

$$\left(\frac{\sum_{i=1}^{m} x_i}{m}, \frac{\sum_{i=1}^{m} f_i}{m}\right).$$

Esempio 5.4.1. Consideriamo i seguenti dati (cfr. [9]):

$$(x_1, f_1) = (1, 0.5) \quad (x_2, f_2) = (2, 2.5) \quad (x_3, f_3) = (3, 2)$$
$$(x_4, f_4) = (4, 4) \quad (x_5, f_5) = (5, 3.5) \quad (x_6, f_6) = (6, 6)$$
$$(x_7, f_7) = (7, 5.5)$$

Volendo determinare la retta che meglio li approssima nel senso dei minimi quadrati, impostiamo il sistema lineare (5.37) che in questo caso risulta essere

$$\begin{cases} 7a_0 + 28a_1 = 24 \\ 28a_0 + 140a_1 = 119.5. \end{cases}$$

La soluzione è $a_0^* \simeq 0.071429$, $a_1^* \simeq 0.839286$; la retta di migliore approssimazione $y = a_1^* x + a_0$ passa per il baricentro dei dati le cui coordinate sono circa (4, 3.42857).

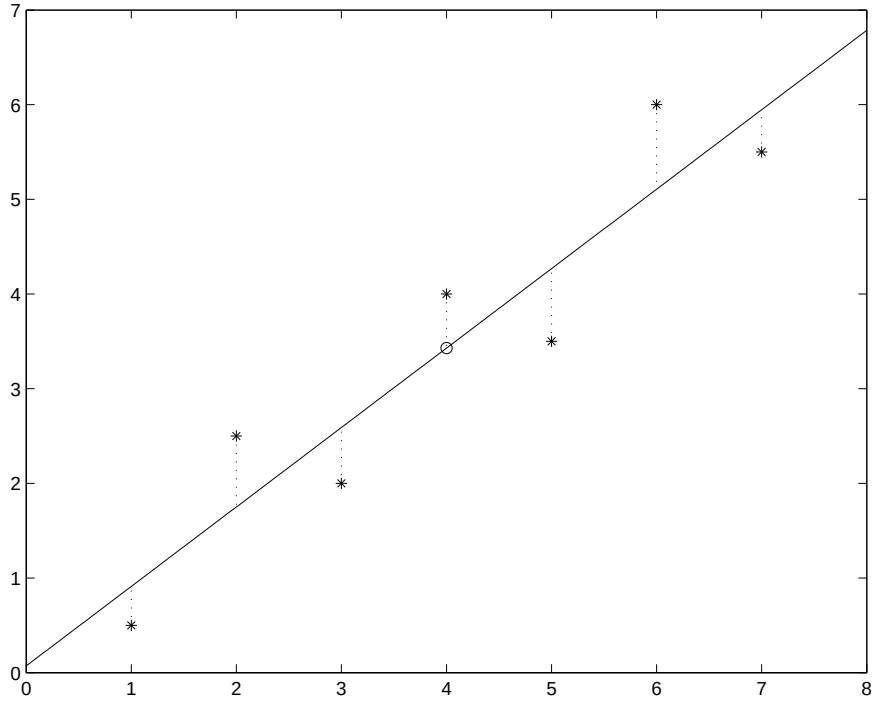

Figura 5.10 Retta di migliore approssimazione ai minimi quadrati

In figura 5.10 sono rappresentati i dati (con il simbolo '*') e il baricentro (con il simbolo 'o'). I segmenti che uniscono ogni dato con il corrispondente

punto sulla retta identificano i singoli scarti

$$\begin{array}{rcll} E_1 & = & a_0 + a_1 x_1 - f_1 & \simeq \ 0.4107 \\ E_2 & = & a_0 + a_1 x_2 - f_2 & \simeq \ -0.7500 \\ E_3 & = & a_0 + a_1 x_3 - f_3 & \simeq \ 0.5893 \\ E_4 & = & a_0 + a_1 x_4 - f_4 & \simeq \ -0.5714 \\ E_5 & = & a_0 + a_1 x_5 - f_5 & \simeq \ 0.7678 \\ E_6 & = & a_0 + a_1 x_6 - f_6 & \simeq \ -0.8929 \\ E_7 & = & a_0 + a_1 x_7 - f_7 & \simeq \ 0.4464 \end{array}$$

a cui corrisponde lo scarto complessivo $\sqrt{\sum_{i=1}^{7} E_i^2} \simeq 1.7295$. ■

5.4.2 Problema dei minimi quadrati lineare

Se le informazioni che si hanno sui dati ci suggeriscono di scegliere come modello un polinomio di grado $n > 1$, il problema della migliore approssimazione ai minimi quadrati si riconduce al seguente problema di minimo

$$\min_{a_0, a_1, \ldots, a_n} \mathcal{S}(a_0, a_1, \ldots, a_n)$$

dove

$$\mathcal{S}(a_0, a_1, \ldots, a_n) = \frac{1}{2} \sum_{i=1}^{m} (a_0 + a_1 x_i + a_2 x_i^2 + \ldots + a_n x_i^n - f_i)^2.$$

In analogia al caso della retta di migliore approssimazione, si procede a uguagliare a zero le $n+1$ derivate parziali di $\mathcal{S}$

$$\begin{array}{rcl} \frac{\partial \mathcal{S}}{\partial a_0} & = & \sum_{i=1}^{m} (a_0 + a_1 x_i + \ldots + a_n x_i^n) - f_i \\ \frac{\partial \mathcal{S}}{\partial a_1} & = & \sum_{i=1}^{m} (a_0 + a_1 x_i + \ldots + a_n x_i^n - f_i) x_i \\ \vdots & & \\ \frac{\partial \mathcal{S}}{\partial a_n} & = & \sum_{i=1}^{m} (a_0 + a_1 x_i + \ldots + a_n x_i^n - f_i) x_i^n \end{array}$$

e si ottiene il seguente sistema lineare di $n+1$ equazioni in $n+1$ incognite (per alleggerire la notazione, abbiamo sottinteso gli estremi delle sommatorie, in cui i varia sempre da 1 a m):

$$\left\{ \begin{array}{lcl} m a_0 + (\sum x_i) a_1 + (\sum x_i^2) a_2 + \ldots + (\sum x_i^n) a_n & = & \sum f_i \\ (\sum x_i) a_0 + (\sum x_i^2) a_1 + (\sum x_i^3) a_2 + \ldots + (\sum x_i^{n+1}) a_n & = & \sum f_i x_i \\ \vdots & & \\ (\sum x_i^n) a_0 + (\sum x_i^{n+1}) a_1 + (\sum x_i^{n+2}) a_2 + \ldots + (\sum x_i^{2n}) a_n & = & \sum f_i x_i^n \end{array} \right.$$

che può essere riscritto nella forma compatta

$$C^T C a = C^T b \tag{5.38}$$

avendo posto

$$C = \begin{pmatrix} 1 & x_1 & x_1^2 & \dots & x_1^n \\ 1 & x_2 & x_2^2 & \dots & x_2^n \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_{m-1} & x_{m-1}^2 & \dots & x_{m-1}^n \\ 1 & x_m & x_m^2 & \dots & x_m^n \end{pmatrix}, \quad a = \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix} \text{ e } b = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_m \end{pmatrix}.$$

Come nel paragrafo precedente, si può dimostrare che la matrice dei coefficienti è definita positiva e quindi il sistema (5.38), detto sistema normale, ha un'unica soluzione.

In realtà, quando m e/o n non sono piccoli, non è consigliabile risolvere il sistema normale per calcolare i coefficienti del polinomio di migliore approssimazione e si preferisce usare una strada diversa, fondamentalmente a causa di problemi di cattivo condizionamento. Se osserviamo infatti la matrice C, vediamo che essa diventa una matrice di Vandermonde quando $m = n + 1$ e ne è una generalizzazione rettangolare quando, come abitualmente succede, m è maggiore di $n+1$. Come abbiamo avuto modo di vedere in altre occasioni, le matrici di Vandermonde possono risultare molto mal condizionate; anche nel caso rettangolare le colonne della matrice C possono essere molto vicine alla dipendenza lineare [20], da cui segue che la matrice dei coefficienti del sistema normale, $C^T C$, può essere molto vicina alla singolarità e quindi mal condizionata (cfr. (4.11)).

Per cercare di limitare l'inconveniente del cattivo condizionamento si può usare un approccio diverso, che passa attraverso una nuova formulazione del problema e non porta al sistema normale. Fissato un indice i fra 1 e m si ha

$$E_i = a_0 + a_1 x_i + \ldots + a_n x_i^m - f_i = \sum_{j=0}^{n} c_{ij} a_j - f_i$$

e quindi

$$\|E\|_2 = \sum_{i=1}^{m} E_i^2 = \sum_{i=1}^{m} \left(\sum_{j=0}^{n} c_{ij} a_j - f_i \right)^2 = \|Ca - b\|_2^2.$$

Un problema della forma

$$\min_a \|Ca - b\|_2^2 \tag{5.39}$$

viene detto in letteratura problema ai minimi quadrati lineare. I metodi abitualmente usati per risolvere questo tipo di problemi sono basati su

fattorizzazioni QR, come accenniamo nel prossimo paragrafo, oppure su tecniche più avanzate di algebra lineare numerica basate sulla cosiddetta "decomposizione ai valori singolari"(SVD). Per dettagli rimandiamo a [20].

5.4.3 Metodo di Householder per il problema dei minimi quadrati lineari (*)

Per risolvere il problema (5.39) possiamo usare un procedimento basato sulle matrici di Householder viste nel Capitolo 4. Ricordiamo la costruzione fatta nel paragrafo 4.5.2 per triangolarizzare una matrice quadrata $n \times n$ e trasformare quindi un sistema $Ax = b$ in uno equivalente $Rx = d$ con matrice dei coefficienti R triangolare superiore. In modo analogo, possiamo ora individuare $n + 1$ matrici ortogonali $Q_1, Q_2, \ldots, Q_{n+1}$ di dimensione $m \times m$ tali che

$$Q_{n+1} Q_n \ldots Q_1 C = \begin{pmatrix} R \\ 0 \end{pmatrix}$$

dove $R \in \mathbb{R}^{(n+1)\times(n+1)}$ è una matrice triangolare superiore e il simbolo zero indica una matrice di dimensione $(m - n - 1) \times (n + 1)$ costituita da tutti elementi nulli. Se ora indichiamo con Q^T il prodotto $Q_{n+1} Q_n \cdots Q_1$ e ricordiamo che le matrici ortogonali lasciano inalterata la norma-2 dei vettori, si ha

$$\|Ca - b\|_2^2 = \|Q^T(Ca - b)\|_2^2. \tag{5.40}$$

Inoltre, indicando con $\tilde{b}$ il vettore $Q^T b$, possiamo scrivere

$$\tilde{b} = \begin{pmatrix} \tilde{b}_1 \\ \tilde{b}_2 \end{pmatrix}$$

con $\tilde{b}_1 \in \mathbb{R}^{n+1}$ e $\tilde{b}_2 \in \mathbb{R}^{m-n-1}$ e quindi, riprendendo la (5.40), si ottiene

$$\|Ca - b\|_2^2 = \|Q^T(Ca - b)\|_2^2 = \|Ra - \tilde{b}_1\|_2^2 + \|\tilde{b}_2\|_2^2.$$

La matrice R è sicuramente invertibile dal momento che C ha colonne linearmente indipendenti [20]. Possiamo allora concludere che il vettore a che risolve il nostro problema è il vettore soluzione del sistema lineare triangolare superiore di dimensione $n + 1$

$$Ra = \tilde{b}_1 \tag{5.41}$$

e lo scarto corrispondente è $\|E\|_2^2 = \|\tilde{b}_2\|_2^2$.

Con questo procedimento si evita la costruzione della matrice $C^T C$ e la risoluzione di un sistema lineare il cui condizionamento può essere molto peggiore di quello insito nel problema. D'altra parte se il problema è mal condizionato, questo si manifesterà anche attraverso il sistema triangolare (5.41). La cosa importante da tenere presente è che il nuovo procedimento è stabile perché utilizza trasformazioni ortogonali. Per un approfondimento su questioni teoriche e numeriche legate alla risoluzione dei problemi ai minimi quadrati lineari rimandiamo a [20] e [10].

5.5 Calcolo numerico di integrali definiti

Nel calcolo di un integrale definito

$$I[f] = \int_a^b f(x)dx$$

si possono verificare diverse situazioni. Se l'espressione della funzione integranda è data, si può tentare di trovarne una primitiva per procedere all'integrazione per via analitica; ma soltanto in pochi casi si riesce agilmente a seguire questa strada: più spesso questo non è possibile e si deve pensare ad un approccio numerico, che inevitabilmente fornirà un valore approssimato dell'integrale. Quando poi la funzione f è nota solo per punti, o soltanto valutabile per un valore fissato di x tramite un algoritmo, la strada numerica è l'unica percorribile. Le formule di approssimazione di integrali definiti sono chiamate formule di quadratura.

5.5.1 Formule di quadratura interpolatorie

Consideriamo l'integrale

$$I[f] = \int_1^5 (6x^3 - 42x^2 + 100x + 6sin(x))dx. \tag{5.42}$$

Poiché conosciamo la primitiva

$$F(x) = 1.5x^4 - 14x^3 + 50x^2 - 6cos(x),$$

possiamo facilmente calcolare $I[f] = F(5) - F(1) \simeq 401.5398$. Vediamo come sarebbe possibile approssimare questo integrale usando soltanto i valori della funzione f in alcuni punti dell'intervallo $[a, b] = [1, 5]$ e quanto queste eventuali approssimazioni si avvicinano al valore esatto $I[f]$.

Iniziamo supponendo di usare soltanto il valore della funzione nel punto di mezzo dell'intervallo $c = \frac{a+b}{2}$. In questo caso (estremo) possiamo pensare di approssimare f con il polinomio di grado zero $P_0(x) = f(\frac{a+b}{2})$ e quindi $I[f]$ con $\int_1^5 P_0(x)dx$. Si ottiene in questo modo

$$I[f] \simeq (b-a)f\left(\frac{a+b}{2}\right) \tag{5.43}$$

che porta ad un valore approssimato $\simeq 339.3869$. Geometricamente, l'operazione che abbiamo fatto equivale ad approssimare l'area sottostante al grafico di f in $[a, b]$ con l'area del rettangolo di base $(b - a)$ ed altezza $f(\frac{a+b}{2})$ (cfr. il primo grafico in alto a sinistra nella figura 5.11).

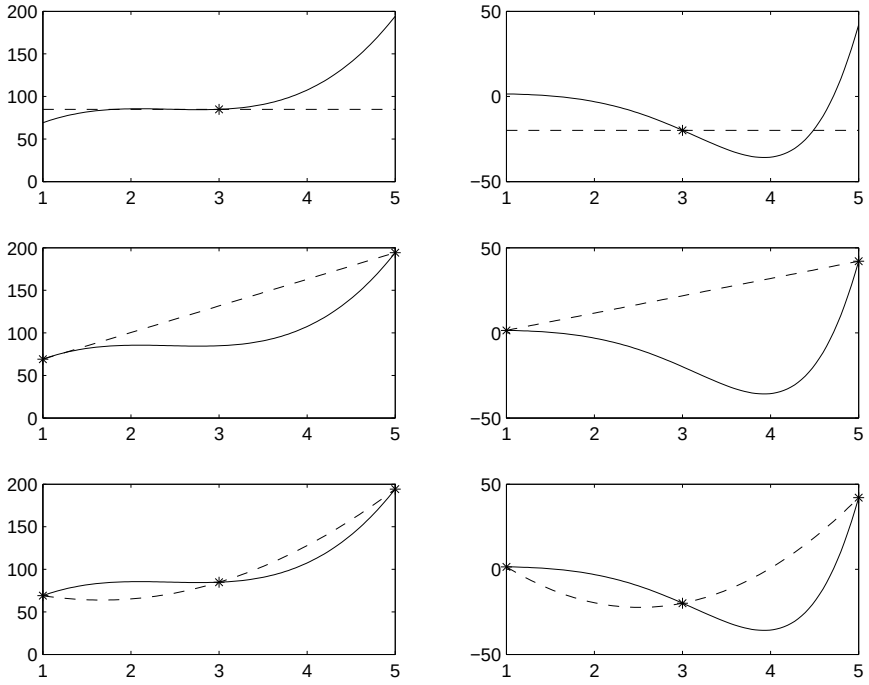

Figura 5.11 Approssimazione di $\int_1^5 (6x^3 - 42x^2 + 100x + 6sin(x))dx$ (a sinistra) e $\int_1^5 (e^x cos(x))dx$ (a destra)

Supponiamo ora di usare i due valori della funzione agli estremi dell'intervallo $f(a)$ e $f(b)$. Allora possiamo approssimare l'integrale con l'area sottostante alla retta che unisce i due punti $(a, f(a))$ e $(b, f(b))$, cioè l'area del trapezio che ha come altezza $(b - a)$ e $f(a)$ e $f(b)$ come basi (cfr. grafico centrale a sinistra nella figura 5.11). Così facendo otteniamo l'approssimazione

$$I[f] \simeq \frac{b-a}{2}[f(a) + f(b)] \tag{5.44}$$

che dà un valore $\simeq 526.591$. Sia dai grafici che dai valori numerici, vediamo che le formule (5.43) e (5.44) non forniscono buone approssimazioni di $I[f]$.

Proviamo ora ad usare tutti e tre i valori $f(a)$, $f(c)$ e $f(b)$. Analogamente a quanto fatto in precedenza possiamo pensare di approssimare $I[f]$ con l'integrale della parabola $P_2(x)$ che interpola f in a, b e c (cfr. grafico in basso a sinistra nella figura 5.11); otteniamo così

$$I[f] \simeq \frac{b-a}{6}\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right] \tag{5.45}$$

che dà un valore $\simeq 401.7881$, molto migliore dei due precedenti. Non è detto in generale che usare tre punti nel modo in cui li abbiamo usati porti ad

una approssimazione dell'integrale migliore di quella che si ottiene con un punto o con due punti; questo dipende dall'andamento della funzione integranda nell'intervallo $[a, b]$. Se per esempio ripetiamo i calcoli precedenti per approssimare

$$\int_1^5 e^x cos(x) dx$$

si ottengono i tre valori -79.5381, 87.1358 e -23.9801 ripettivamente (cfr grafici sul lato destro della figura 5.11), a fronte del valore esatto -51.9869 calcolabile per via analitica usando la primitiva $F(x) = 0.5e^x(cos(x) + sin(x))$; in questo caso il valore approssimato ottenuto con un solo punto è leggermente migliore di quello ottenuto con tre punti.

Le formule (5.43), (5.44) e (5.45) si chiamano formula del punto di mezzo, formula dei trapezi e formula di Simpson [9] rispettivamente. Esse fanno parte di una famiglia di formule di quadratura che si ottengono approssimando $I[f]$ con l'integrale di un polinomio interpolante. Più precisamente, si costruisce il polinomio $P_n(x)$ che interpola f in $n + 1$ punti x_0, x_1, ..., x_n appartenenti ad $[a, b]$ e si approssima $I[f]$ con $\int_a^b P_n(x)dx$. Scrivendo il polinomio interpolante nella forma di Lagrange si ottiene

$$I[f] \simeq \int_a^b P_n(x)dx = \int_a^b \sum_{i=0}^{n} f(x_i)l_i(x)dx = \sum_{i=0}^{n} f(x_i) \int_a^b l_i(x)dx,$$

ovvero, ponendo $w_i = \int_a^b l_i(x)dx$ per ogni i,

$$I[f] \simeq \sum_{i=0}^{n} w_i f(x_i).$$

Le formule così generate sono chiamate formule di quadratura interpolatorie. I punti $x_0, x_1, \ldots, x_n$ sono i nodi della formula di quadratura e i numeri w_0, w_1, ..., w_n, che dipendono dai nodi ma non dalla funzione f, sono detti pesi. Le formule di quadratura interpolatorie costruite su nodi equidistanti prendono il nome di formule di Newton-Cotes. Le formule del punto di mezzo, dei trapezi e di Simpson sono particolari formule di Newton-Cotes costruite su 1, 2 e 3 nodi rispettivamente. In particolare, la formula del punto di mezzo, che non include fra i nodi gli estremi dell'intervallo, è detta di tipo aperto; quelle dei trapezi e di Simpson che invece usano gli estremi dell'intervallo come nodi, sono di tipo chiuso.

[9] Thomas Simpson (1710-1761) è conosciuto soprattutto per questa formula di quadratura. In realtà la formula era già stata utilizzata da Newton e ancor prima, nella sua formulazione geometrica, dal matematico italiano Bonaventura Cavalieri (1598-1647); per questo motivo in molti testi italiani si suole rendere omaggio anche a Cavalieri parlando di formula di Cavalieri-Simpson.

Approssimando $I[f]$ con una formula di quadratura interpolatoria commettiamo un errore

$$R_n = I[f] - \int_a^b P_n(x)dx$$

strettamente legato all'errore di interpolazione $E_n(x) = f(x) - P_n(x)$; è infatti evidente che

$$R_n = \int_a^b E_n(x)dx.$$

Per quanto riguarda le tre formule viste precedentemente, si può dimostrare [15] che se la funzione f è sufficientemente regolare (in particolare $f \in C^2[a,b]$ per la formula del punto di mezzo e quella dei trapezi e $f \in C^4[a,b]$ per la formula di Simpson) gli errori hanno le seguenti espressioni

$$\begin{array}{ll} \text{Formula del punto di mezzo} & R_0 = \frac{1}{3} f''(\xi) \left(\frac{b-a}{2}\right)^3 \\ \text{Formula dei trapezi} & R_1 = -\frac{1}{12} f''(\eta)(b-a)^3 \\ \text{Formula di Simpson} & R_2 = -\frac{1}{90} f^{iv}(\theta) \left(\frac{b-a}{2}\right)^5 \end{array} \tag{5.46}$$

dove ξ, η e θ sono punti interni ad $[a,b]$.

Ovviamente non possiamo pensare che in generale, aumentando il numero dei nodi nella formula di quadratura si ottenga maggiore accuratezza nell'approssimazione dell'integrale. Se non bastassero tutte le considerazioni fatte nel paragrafo precedente sull'errore di interpolazione e gli esempi svolti all'inizio di questo paragrafo, possiamo tornare a osservare la figura 5.3, dalla quale intuiamo facilmente che la differenza fra $I[f] = \int_{-1}^{1} \frac{1}{1+x^2} dx$ e i valori approssimati calcolati con le formule di Newton Cotes di tipo chiuso con n crescente tende ad aumentare. Le formule di Newton-Cotes con molti nodi sono del resto sconsigliate anche per motivi di stabilità: al crescere di n, i pesi tendono ad assumere valori di segno alterno e questo può provocare in aritmetica floating point un'amplificazione incontrollata degli errori di arrotondamento [15].

Sappiamo che i possibili approcci per ottenere funzioni interpolanti accurate sono due: scegliere opportunamente i punti fondamentali, ad esempio i punti di Chebyshev, oppure utilizzare funzioni polinomiali a tratti, eventualmente splines, di grado basso. Questi sono gli approcci usati anche nelle formule di quadratura quando si ha necessità e/o possibilità di usare un numero elevato di nodi. Con la prima strada si arriva alle formule interpolatorie di tipo gaussiano, che utilizzano come nodi gli zeri di opportuni polinomi ortogonali, di cui i polinomi di Chebyshev che abbiamo già incontrato sono un particolare esempio. I polinomi ortogonali sono uno strumento molto importante nel campo dell'approssimazione di dati e funzioni, ma una loro trattazione va oltre gli scopi di questo libro; quindi

rimandiamo chi fosse interessato alle formule gaussiane a [15] o a [27]. La seconda strada, quella delle interpolanti polinomiali a tratti, porta a nuove formule di quadratura, molto usate nella pratica, che prendono il nome di formule di quadratura composite.

5.5.2 Formule di quadratura composite

Se dividiamo l'intervallo di integrazione $[a, b]$ in un certo numero di sottointervalli $I_1, I_2, \ldots$, le proprietà degli integrali permettono di scrivere

$$\int_a^b f(x)dx = \sum_i \int_{I_i} f(x)dx$$

dove il simbolo $\int_{I_i}$ indica l'integrale esteso ad I_i. Possiamo allora approssimare ognuno degli integrali al secondo membro con una formula di quadratura che usi pochi nodi e $I[f]$ con la somma dei valori così ottenuti. In questo modo si ottiene una formula di quadratura composita. Ovviamente possiamo costruire formule di tipo composito a partire da qualunque formula di quadratura. Le più usate in pratica sono quelle basate sulla formula dei trapezi e sulla formula di Simpson. Ad esempio, se dividiamo $[a, b]$ in due sottointervalli $[a, x_1]$ e $[x_1, b]$ di uguale ampiezza h e usiamo la formula dei trapezi su ciascuno di essi, si ottiene

$$I[f] \simeq \frac{h}{2}[f(a) + f(x_1)] + \frac{h}{2}[f(x_1) + f(b)] = \frac{h}{2}[f(a) + 2f(x_1) + f(b)].$$

Supponiamo invece di fissare cinque punti $a = x_0 < x_1 < x_2 < x_3 < x_4 = b$ a distanza $h = \frac{b-a}{4}$ l'uno dall'altro e applicare la formula di Simpson nei due intervalli $[x_0, x_2]$ e $[x_2, x_4]$; tenendo conto che ognuno di questi intervalli ha ampiezza $2h$ si ottiene

$$\begin{aligned} I[f] &\simeq \frac{h}{3}\left[f(x_0) + 4f(x_1) + f(x_2)\right] + \frac{h}{3}\left[f(x_2) + 4f(x_3) + f(x_4)\right] \\ &= \frac{h}{3}\left[f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + f(x_4)\right]. \end{aligned}$$

Generalizzando, dividiamo $[a, b]$ in n sottointervalli $[x_i, x_{i+1}]$ mediante punti equidistanti $x_i = a + ih$ per $i = 0, 1, \ldots, n$, con $h = \frac{b-a}{n}$. Applicando su ciascuno di essi la formula dei trapezi, si ottiene la formula composita dei trapezi:

$$I[f] \simeq \frac{h}{2}[f(a) + 2\sum_{i=1}^{n-1} f(a + ih) + f(b)],$$

dove abbiamo tenuto conto del fatto che $x_0 = a$ e $x_n = b$.

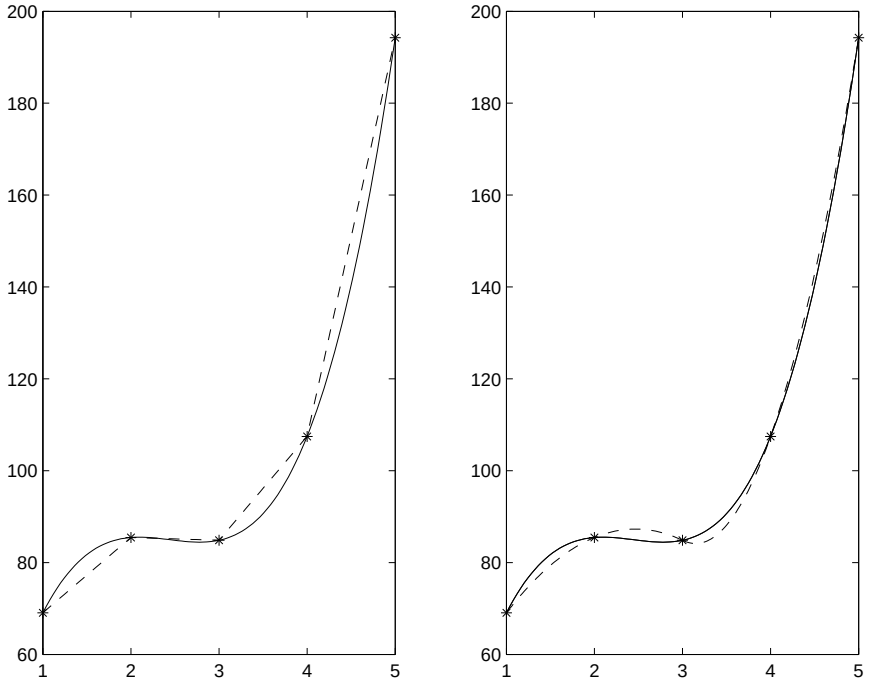

Figura 5.12 Approssimazione di $\int_1^5 (6x^3 - 42x^2 + 100x + 6sin(x))dx$: formule composite dei trapezi e di Simpson con 5 nodi

Se abbiamo l'accortezza di scegliere n pari, possiamo applicare la formula di Simpson su ciascun intervallo $[x_{2i}, x_{2i+2}]$ di ampiezza $2h$ per $i = 0, 1, \ldots, \frac{n}{2} - 1$ e sommare, ottenendo la formula composita di Simpson

$$I[f] \simeq \frac{h}{3}\left[f(a) + 2\sum_{i=1}^{\frac{n}{2}-1} f(a+2ih) + 4\sum_{i=1}^{\frac{n}{2}} f(a+(2i-1)h) + f(b)\right].$$

La figura 5.12 descrive graficamente l'approssimazione dell'integrale (5.42) mediante la formula composita dei trapezi (a sinistra) e la formula composita di Simpson (a destra) con 5 nodi. In pratica, stiamo approssimando f con una funzione lineare a tratti nel primo caso e con una quadratica a tratti nel secondo. Nel seguito indicheremo con I_T e I_S i valori approssimati di $I[f]$ ottenuti con la formula composita dei trapezi e con quella di Simpson rispettivamente, e con R_T e R_S i corrispondenti errori.

Esempio 5.5.1. Consideriamo

$$\int_{-1}^{1} \frac{1}{1+x^2} dx.$$

Poiché conosciamo una primitiva $F(x) = arctg(x)$, siamo in grado di calcolare per via analitica il valore esatto $I[f] = \frac{\pi}{2} \simeq 1.570796326794897$. Se applichiamo le formule composite dei trapezi e Simpson otteniamo le approssimazioni riportate nella tabella 5.3. Dalla tabella vediamo che entram-

n	I_T	$\lvert R_T \rvert$	I_S	$\lvert R_S \rvert$
1	1.000000000000000	5.71e−01	1.666666666666667	9.59e−02
2	1.500000000000000	7.08e−02	1.566666666666667	4.13e−03
4	1.550000000000000	2.08e−02	1.570784313725490	1.20e−05
8	1.565588235294118	5.21e−03	1.570796251229353	7.56e−08
16	1.569494247245545	1.30e−03	1.570796325612411	1.18e−09
32	1.570470806020695	3.26e−04	1.570796326776418	1.85e−11
64	1.570714946587486	8.14e−05	1.570796326794607	2.89e−13
128	1.570775981742828	2.03e−05	1.570796326794892	4.22e−15

Tabella 5.3 Esempio 5.5.1: formule composite dei trapezi e Simpson

be le formule forniscono valori sempre più accurati al crescere di n, ma con un andamento molto diverso: l'errore della formula di Simpson diminuisce molto più velocemente di quello relativo alla formula dei trapezi.

■

Se la funzione f è sufficientemente regolare, possiamo sfruttare le espressioni degli errori relativi alle formule di base date in (5.46) per determinare quelle di R_T e R_S. Iniziamo con la formula dei trapezi, supponendo $f \in C^2[a,b]$. Per le proprietà degli integrali, R_T è la somma degli errori commessi sui singoli sottointervalli esprimibile come

$$R_T = -\frac{1}{12}h^3 \sum_{i=0}^{n-1} f''(\eta_i),$$

con $\eta_i \in (x_i, x_{i+1})$. Ricordiamo ora che $h = \frac{b-a}{n}$ e riscriviamo R_T nella forma

$$R_T = -\frac{1}{12}h^2(b-a)\frac{\sum_{i=0}^{n-1} f''(\eta_i)}{n},$$

dove si è evidenziato il termine $\frac{\sum_{i=0}^{n-1} f''(\eta_i)}{n}$ che rappresenta la media aritmetica dei valori della derivata seconda in η_0, η_1, ..., η_{n-1}. Sfruttando la continuità di f'' in $[a,b]$ è facile vedere che esiste un η interno ad $[a,b]$ tale che

$$\frac{\sum_{i=0}^{n-1} f''(\eta_i)}{n} = f''(\eta);$$

da questo segue

$$R_T = -\frac{1}{12}h^2(b-a)f''(\eta). \tag{5.47}$$

Consideriamo ora la formula composita di Simpson, tenendo conto che in questo caso R_S è la somma degli errori di quadratura su $\frac{n}{2}$ intervalli di ampiezza $2h$. Supponendo $f \in C^4[a,b]$, in modo analogo a prima si può dimostrare che

$$R_S = -\frac{1}{180}h^4(b-a)f^{iv}(\theta) \tag{5.48}$$

dove $\theta \in (a,b)$. Quindi, quando h tende a zero, sia R_T che R_S tendono a zero, ma con diverse velocità: R_T è un $\mathcal{O}(h^2)$ e R_S è un $\mathcal{O}(h^4)$. Per questo, e per la sua semplicità, la formula di Simpson è alla base delle procedure di integrazione numerica più comunemente usate, disponibili ad esempio in MATLAB e in FORTRAN. Torneremo su questi aspetti più avanti.

L'esempio successivo, tratto da [27], mostra una situazione particolare, in cui la formula composita dei trapezi funziona meglio di quanto previsto da (5.47).

Esempio 5.5.2. Abbiamo approssimato con la formula composita dei trapezi i due integrali seguenti:

$$I_1[f] = \int_0^{2\pi} cos^2(x)e^{sin(2x)}dx,$$

per cui assumiamo come valore esatto il valore 3.977463167069391 calcolato con la funzione quad di MATLAB, e

$$I_2[f] = \int_0^1 cos^2(x) e^{sin(2x)} dx,$$

il cui valore esatto, ancora calcolato con quad, è 1.429777357029912. Nella tabella 5.4 riportiamo le approssimazioni $I_{1,T}$ (per $I_1[f]$) e $I_{2,T}$ (per $I_2[f]$) e i corrispondenti errori $R_{1,T}$ e $R_{2,T}$ al variare di n. Notiamo che al crescere

n	$I_{1,T}$	$\lvert R_{1,T}\rvert$	$I_{2,T}$	$\lvert R_{2,T}\rvert$
2	6.283185307179585	2.31e+00	1.324472005554721	1.05e−01
4	3.141592653589792	8.36e−01	1.404217767571056	2.56e−02
8	3.994661719911019	1.72e−02	1.423433227037202	6.34e−03
16	3.977463886350889	7.19e−07	1.428194056490323	1.58e−03
32	3.977463260506423	9.34e−08	1.429381606668820	3.96e−04

Tabella 5.4 Esempio 5.5.2: formula composita dei trapezi per funzioni periodiche

di n l'errore $R_{1,T}$ diminuisce molto più velocemente di quanto suggerirebbe l'espressione (5.47). Questo comportamento è dovuto al fatto che l'errore della formula composita dei trapezi va a zero più velocemente con h quando la funzione integranda è periodica (come la funzione che stiamo considerando) e l'intervallo di integrazione coincide con l'intero periodo [27]. Il fatto che l'intervallo $[a, b]$ coincida con un periodo è fondamentale. A titolo di riprova, è sufficiente osservare i risultati relativi a $I_2[f]$: l'andamento di $R_{2,T}$ è simile a quello riscontrato nella tabella 5.3. ■

5.5.3 Estrapolazione di Richardson

Le espressioni degli errori associati alle diverse formule di quadratura viste nei paragrafi precedenti hanno validità soltanto teorica perché dipendono da una derivata della funzione integranda; quand'anche si potesse agevolmente calcolare la derivata, non si avrebbe idea del punto in cui calcolarla. Così resta il problema, fissata la formula composita che si vuole usare, di scegliere h in modo da ottenere una buona approssimazione di $I[f]$. Un modo di procedere potrebbe essere il seguente. Si calcolano due approssimazioni I_1 e I_2 di $I[f]$ con due diverse suddivisioni dell'intervallo, diciamo con sottointervalli di ampiezza h_1 e h_2 con $h_2 < h_1$, in modo che I_2 sia, in linea di principio, più accurata di I_1; in pratica converrà scegliere $h_2 = \frac{h_1}{2}$ così da poter utilizzare nel calcolo di I_2 valori della funzione già usati nel calcolo di I_1. Se la differenza fra I_2 e I_1 è minore di una tolleranza fissata, si accetta I_2 come valore dell'integrale, altrimenti si diminuisce l'ampiezza dei sottointervalli e si ripete il procedimento. Questa strategia può essere migliorata utilizzando una tecnica nota come estrapolazione di

Richardson che, dati I_1 e I_2, permette di stimare l'errore associato a I_2 in modo abbastanza accurato [10].

Prendiamo come base la formula composita dei trapezi. Indichiamo con $I_{T,h}$ e $I_{T,h/2}$ i risultati ottenuti con due diverse suddivisioni dell'intervallo $[a,b]$, una con sottointervalli di ampiezza h e l'altra con $\frac{h}{2}$; siano $R_{T,h}$ e $R_{T,h/2}$ gli errori corrispondenti. Usando la (5.47) possiamo scrivere

$$R_{T,h} = -\frac{1}{12}(b-a)f''(\eta_1)h^2 \tag{5.49}$$

e

$$R_{T,h/2} = -\frac{1}{12}(b-a)f''(\eta_2)\left(\frac{h}{2}\right)^2, \tag{5.50}$$

con $\eta_1, \eta_2 \in (a,b)$. Assumiamo come ipotesi di lavoro che f'' non vari troppo in $[a,b]$, così che $f''(\eta_1) \simeq f''(\eta_2)$; questo ci permette di ottenere da (5.49) e (5.50) la seguente relazione

$$R_{T,h/2} \simeq \frac{1}{4}R_{T,h}$$

da cui, ricordando che $R_{T,h} = I[f] - I_{T,h}$ e $R_{T,h/2} = I[f] - I_{T,h/2}$, segue

$$R_{T,h/2} \simeq \frac{1}{4}(I[f] - I_{T,h} + I_{T,h/2} - I_{T,h/2}) = \frac{1}{4}R_{T,h/2} + \frac{1}{4}(I_{T,h/2} - I_{T,h})$$

e quindi

$$R_{T,h/2} \simeq \frac{I_{T,h/2} - I_{T,h}}{3}. \tag{5.51}$$

La (5.51) è utile da due punti di vista: prima di tutto fornisce una stima effettivamente calcolabile dell'errore commesso con passo $h/2$; secondariamente, essa consente di correggere $I_{T,h/2}$ e ottenere un'altra approssimazione $\bar{I}_T$ dell'integrale:

$$\bar{I}_T = I_{T,h/2} + \frac{I_{T,h/2} - I_{T,h}}{3}. \tag{5.52}$$

[10] L'estrapolazione di Richardson è una strategia di carattere generale, che può essere applicata in molti contesti, non solo in quello delle formula di quadratura. Il suo inventore stesso, Lewis Fry Richardson (1881-1953) applicò le sue idee in molti campi, in particolare nel campo della meteorologia. A proposito di Richardson, ci piace ricordare che egli fu un ardente pacifista: fece obiezione di coscienza durante la prima guerra mondiale e questo gli costò l'allontanamento dal mondo accademico; interruppe alcuni studi nel campo della meteorologia, distruggendo i risultati non ancora pubblicati, quando si rese conto che essi avrebbero potuto essere utilizzati nella progettazione di armi chimiche; infine, egli applicò le sue conoscenze matematiche allo studio delle radici dei conflitti internazionali, motivo per cui è considerato uno dei padri dell'analisi scientifica dei conflitti. Chi fosse interessato a saperne di più, può consultare T.W. Korner, The pleasure of counting, Cap. 8 (A Quacker mathematician) e Cap. 9 (Richardson on war), Cambridge U.P. 1996.

Se la stima (5.51) è buona, $\bar{I}_T$ è un'approssimazione di $I[f]$ molto più accurata di $I_{T,h/2}$ e $I_{T,h}$.

Esempio 5.5.3. Come nell'esempio 5.5.1, abbiamo approssimato

$$I[f] = \int_{-1}^{1} \frac{1}{1+x^2} dx$$

con la formula composita dei trapezi per n crescente. Per ogni n, usando i valori ottenuti per $h = \frac{b-a}{n}$ e $h = \frac{b-a}{2n}$, abbiamo calcolato l'approssimazione $\bar{I}_T$ secondo la formula (5.52). Dalla tabella 5.5 si vede che l'errore $\bar{R}_T$ asso-

n	I_T	$\lvert R_T\rvert$	$\bar{I}_T$	$\lvert\bar{R}_T\rvert$
1	1.000000000000000	5.71e−01		
2	1.500000000000000	7.08e−02	1.666666666666667	9.59e−02
4	1.550000000000000	2.08e−02	1.566666666666667	4.13e−03
8	1.565588235294118	5.21e−03	1.570784313725490	1.20e−05
16	1.569494247245545	1.30e−03	1.570796251229354	7.56e−08
32	1.570470806020695	3.26e−04	1.570796325612411	1.18e−09
64	1.570714946587486	8.14e−05	1.570796326776417	1.85e−11
128	1.570775981742828	2.03e−05	1.570796326794608	2.89e−13

Tabella 5.5 Esempio 5.5.3: estrapolazione di Richardson

ciato a $\bar{I}_T$ va a zero molto più rapidamente di R_T. Questo implicitamente assicura che la stima dell'errore (5.51) è accurata.

■

L'estrapolazione di Richardson può essere utilizzata anche a partire dalla formula composita di Simpson. Tenendo conto dell'espressione (5.48) dell'errore e facendo passaggi del tutto analoghi ai precedenti si ottiene (con ovvio significato dei simboli)

$$R_{S,h/2} \simeq \frac{I_{S,h/2} - I_{S,h}}{15} \tag{5.53}$$

e

$$\bar{I}_S = I_{S,h/2} + \frac{I_{S,h/2} - I_{S,h}}{15}. \tag{5.54}$$

Finora abbiamo supposto di usare partizioni uniformi dell'intervallo di integrazione, ma questo approccio può essere dispendioso ed inutile, perché può succedere di usare molti nodi anche in zone di $[a, b]$ in cui un andamento poco vario della funzione ne richiederebbe pochi. In pratica è conveniente procedere con strategie che provvedano a suddividere l'intervallo di integrazione in sottointervalli di ampiezza diversa, ad applicare in questi una formula base con pochi nodi e addensare i nodi soltanto dove la

funzione presenta grosse variazioni, collocandone pochi altrove. Tali strategie vengono chiamate adattative perché "si adattano" al comportamento della funzione.

I procedimenti adattativi di uso più comune, come quelli realizzati nelle funzioni quad e quadl in MATLAB o nei sottoprogrammi adattativi contenuti nel package quadpack in FORTRAN, sono basati sulla formula composita di Simpson oppure su formule composite gaussiane. La realizzazione di questi procedimenti non è assolutamente semplice; per avere un'idea delle difficoltà legate, ad esempio, alla scelta dei criteri di arresto, rimandiamo all'articolo [16]. Qui ci limitiamo a descrivere a grandi linee il funzionamento di questi procedimenti, facendo riferimento alla formula di Simpson. Si inizia suddividendo $[a, b]$ in due (o tre) sottointervalli; per ciascuno di questi si calcolano le approssimazioni I_1 e I_2 con la formula di Simpson con 3 e 5 nodi equidistanti e si stima l'errore di I_2 mediante la formula (5.53), cioè $\frac{I_2-I_1}{15}$. Se l'errore così stimato risulta minore della tolleranza prefissata, si accetta I_2 come integrale esteso al sottointervallo su cui stiamo lavorando; altrimenti, il sottointervallo viene diviso in due parti, su ciascuna delle quali si ripete il procedimento. Alla fine, l'approssimazione numerica dell'integrale su $[a, b]$ viene ottenuto sommando le approssimazioni che via via vengono accettate sui vari sottointervalli. In questo modo, nelle zone di $[a, b]$ in cui la funzione varia di più sarà necessario procedere a un grande numero di suddivisioni mentre nelle zone in cui la funzione presenta meno variazioni il numero di suddivisioni sarà inferiore: questo porta in generale a una distribuzione non uniforme dei nodi.

A

Appendice: RICHIAMI DI ALGEBRA LINEARE

In questa appendice raccogliamo in forma sintetica alcune definizioni e risultati di algebra lineare che riteniamo debbano far parte del bagaglio culturale degli studenti che utilizzano questo testo. Per un eventuale approfondimento su questi argomenti rimandiamo ai numerosi libri di algebra lineare disponibili sia in italiano che in inglese, come ad esempio [1] o [25].

Una matrice $A \in \mathbb{R}^{n\times m}$ è una tabella di $n \times m$ numeri disposti su n righe e m colonne:

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{pmatrix}.$$

La matrice si riduce ad un vettore riga in $\mathbb{R}^m$ quando $n = 1$ e ad un vettore colonna in $\mathbb{R}^n$ quando $m = 1$. Per convenzione, una matrice è di solito indicata con una lettera maiuscola e i suoi elementi con la corrispondente lettera minuscola. Per contro, i vettori sono indicati con lettere minuscole. Identificando ogni colonna (riga) di A con un vettore, una matrice può essere vista come un insieme di vettori colonna (vettori riga).
Quando $m = n$, si dice che A è una matrice quadrata $n \times n$ e n è detto dimensione o ordine della matrice.

Matrici quadrate con struttura particolare

Una matrice è strutturata quando ha molti elementi uguali a zero disposti in modo non casuale, bensì a riempire una parte significativa della matrice. Alcuni esempi che si incontrano frequentemente sono i seguenti:

– matrici diagonali: $a_{ij} = 0$ per $i \neq j$

$$A = \begin{pmatrix} a_{11} & 0 & \dots & 0 \\ 0 & a_{22} & \dots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & a_{nn} \end{pmatrix}$$

Una matrice diagonale viene anche indicata con $A = diag(a_{11}, a_{22}, \ldots, a_{nn})$. Un caso particolare molto importante di matrice diagonale è la matrice identità

$$I = diag(1, 1, \ldots, 1)$$

– matrici triangolari superiori: $a_{ij} = 0$ per $i > j$

$$A = \begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ 0 & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \ldots & a_{nn} \end{pmatrix}$$

– matrici triangolari inferiori: $a_{ij} = 0$ per $i < j$

$$A = \begin{pmatrix} a_{11} & 0 & \ldots & 0 \\ a_{21} & a_{22} & \ldots & 0 \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \ldots & a_{nn} \end{pmatrix}$$

– matrici tridiagonali: $a_{ij} = 0$ per $|i - j| > 1$

$$A = \begin{pmatrix} a_{11} & a_{12} & 0 & 0 & \ldots & 0 \\ a_{21} & a_{22} & a_{23} & 0 & \ldots & 0 \\ \vdots & & & & & \\ 0 & \ldots & 0 & a_{n-2,n-1} & a_{n-1,n-1} & a_{n-1,n} \\ 0 & \ldots & 0 & 0 & a_{n-1,n} & a_{n,n} \end{pmatrix}$$

Matrice trasposta

La trasposta di una matrice A, indicata con A^T, si ottiene scambiando le righe con le colonne: la prima riga diventa la prima colonna, la seconda riga diventa la seconda colonna, e così via.

$$A = \begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1m} \\ a_{21} & a_{22} & \ldots & a_{2m} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \ldots & a_{nm} \end{pmatrix} \Rightarrow A^T = \begin{pmatrix} a_{11} & a_{21} & \ldots & a_{n1} \\ a_{12} & a_{22} & \ldots & a_{n2} \\ \vdots & \vdots & & \vdots \\ a_{1m} & a_{2m} & \ldots & a_{nm} \end{pmatrix}.$$

Se A è una matrice $n \times m$ la sua trasposta A^T è una matrice $m \times n$. Da questo segue che il trasposto di un vettore colonna è un vettore riga

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \Rightarrow x^T = (x_1 \;\; x_2 \;\; \ldots \;\; x_n)$$

e viceversa. Ricordiamo a questo proposito che, per convenzione, i vettori in matematica sono sempre vettori colonna e quindi un vettore riga è sempre visto come il trasposto di un vettore (colonna).

Una matrice quadrata A è simmetrica quando $A = A^T$, ovvero $a_{ij} = a_{ji}$ per ogni $i, j = 1, \ldots, n$, dove n è la dimensione.

Data una matrice $A \in \mathbb{R}^{n\times m}$, la matrice $A^T A$ è quadrata, di dimensione n e simmetrica. Il prodotto fra matrici è definito più avanti.

Una matrice quadrata A è ortogonale quando $AA^T = A^T A = I$.

Operazioni fra matrici

– Addizione e sottrazione (si può fare soltanto fra matrici delle stesse dimensioni): date $A, B \in \mathbb{R}^{n\times m}$, la matrice somma o differenza $C = A \pm B$ è ancora una matrice in $\mathbb{R}^{n\times m}$ e i suoi elementi sono definiti da

$$c_{ij} = a_{ij} \pm b_{ij}, \quad i, j = 1, \ldots n.$$

  L'addizione fra matrici mantiene le proprietà associativa e commutativa dell'addizione scalare
– Moltiplicazione per uno scalare: data $A \in \mathbb{R}^{n\times m}$, la matrice αA è la matrice ottenuta moltiplicando per α tutti gli elementi di A
– Moltiplicazione righe per colonne (si può fare soltanto se il numero di colonne della prima matrice è uguale al numero di righe della seconda): date $A \in \mathbb{R}^{n\times m}$ e $B \in \mathbb{R}^{m\times p}$, la matrice prodotto $C = A \times B$ appartiene a $\mathbb{R}^{n\times p}$ e i suoi elementi sono definiti da

$$c_{ij} = \sum_{k=1}^{m} a_{ik} b_{kj}, \quad i = 1, \ldots, n \ \text{ e } \ j = 1, \ldots, p.$$

  Un caso particolarmente importante è il prodotto fra una matrice $A \in \mathbb{R}^{n\times m}$ e un vettore $x \in \mathbb{R}^m$. In questo caso il risultato è il vettore $y = Ax \in \mathbb{R}^n$ le cui componenti sono date da:

$$y_i = \sum_{k=1}^{m} a_{ik} x_k.$$

  Un altro caso particolare molto importante è il prodotto scalare fra due vettori $x, y \in \mathbb{R}^n$, definito da

$$x^T y = y^T x = \sum_{i=1}^{n} x_i y_i.$$

  Due vettori il cui prodotto scalare sia uguale a zero sono detti ortogonali.

Dati una matrice quadrata $A \in \mathbb{R}^{n\times n}$ e un vettore $x \in \mathbb{R}^n$, la quantità $x^T Ax$ è uno scalare. A è definita positiva se $x^T Ax > 0$ per ogni $x \neq 0$, e semidefinita positiva se $x^T Ax \geq 0$ per ogni $x \neq 0$.

Date due matrici A e B quadrate della stessa dimensione, si ha

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

Il prodotto scalare è un'operazione commutativa, ma in generale la moltiplicazione fra matrici non mantiene questa proprietà, anche quando i due fattori A e B sono matrici quadrate e quindi le matrici prodotto AB e BA sono entrambe definite ed hanno la stessa dimensione. Per esempio, date

$$A = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \quad \text{e} \quad B = \begin{pmatrix} 1 & 1 \\ -1 & -1 \end{pmatrix}, \tag{A.1}$$

si ha:

$$AB = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \quad \text{e} \quad BA = \begin{pmatrix} 2 & 2 \\ -2 & -2 \end{pmatrix}.$$

Questo esempio mostra anche che la legge di annullamento del prodotto valida per la moltiplicazione fra scalari non si estende al caso matriciale. Infatti il prodotto fra due matrici può essere uguale a zero, ovvero una matrice con tutti elementi uguali a zero, anche se nessuna delle due è la matrice nulla.

L'elemento neutro della moltiplicazione è la matrice identità; infatti per qualunque matrice A si ha $AI = IA = A$ e, ovviamente, per qualunque vettore x si ha $Ix = x$.

Combinazioni lineari e basi

Dati k vettori $x^{(1)}, x^{(2)}, \ldots, x^{(k)}$ in $\mathbb{R}^n$ e k numeri reali $\alpha_1, \alpha_2, \ldots, \alpha_k$, il vettore

$$y = \alpha_1 x^{(1)} + \alpha_2 x^{(2)} + \ldots + \alpha_k x^{(k)} = \sum_{i=1}^{k} \alpha_i x^{(i)}$$

è ancora un vettore di $\mathbb{R}^n$ e si dice che y è una combinazione lineare dei vettori $x^{(1)}, x^{(2)}, \ldots, x^{(k)}$. Ricordando la definizione del prodotto righe per colonne, il vettore y può essere scritto in modo compatto come

$$y = X\alpha,$$

dove X è la matrice $n \times k$ che ha i vettori $x^{(1)}, x^{(2)}, \ldots, x^{(k)}$ come colonne e $\alpha \in \mathbb{R}^k$ è il vettore di componenti $\alpha_1, \alpha_2, \ldots, \alpha_k$.
I vettori $x^{(1)}, x^{(2)}, \ldots, x^{(k)}$ si dicono linearmente indipendenti quando

$$y = 0 \Leftrightarrow \alpha_1 = \alpha_2 = \ldots = \alpha_k = 0$$

e linearmente dipendenti in caso contrario, cioè se esiste una k-upla di numeri $\alpha_1, \alpha_2, \ldots, \alpha_k$ non tutti nulli tali che $y = 0$.
Si dimostra che un insieme di vettori linearmente indipendenti in $\mathbb{R}^n$ non può essere costituito da più di n vettori.
Un insieme di n vettori linearmente indipendenti in R^n costituisce una base per R^n e qualunque altro vettore di $\mathbb{R}^n$ può essere espresso in un unico modo come loro combinazione lineare. Ad esempio i vettori $e^{(1)} = (1, 0, \ldots, 0)^T$, $e^{(2)} = (0, 1, 0, \ldots, 0)^T$, ..., $e^{(n)} = (0, 0, \ldots, 0, 1)^T$ rappresentano una base di R^n, comunemente indicata come base canonica.

Determinante di matrici quadrate

Ad ogni matrice quadrata A è associato un numero, detto determinante di A e indicato con il simbolo $\det(A)$, definito in maniera ricorsiva nel modo seguente.
Per una matrice 2×2

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix},$$

si definisce

$$\det(A) = a_{11}a_{22} - a_{21}a_{12}.$$

Per una matrice $n \times n$ il determinante è dato dalla seguente formula (di Laplace), che utilizza determinanti di sottomatrici $(n-1) \times (n-1)$:

$$\det(A) = \sum_{j=1}^{n} (-1)^{i+j} \det(A_{ij}) a_{ij}, \tag{A.2}$$

dove i è un qualunque indice di riga fissato e A_{ij} è la sottomatrice di ordine $(n-1)$ che si ottiene da A togliendo la riga i-esima e la colonna j-esima. Il prodotto $(-1)^{i+j} \det(A_{ij})$ è detto "complemento algebrico" dell'elemento a_{ij}. Il valore del determinante è lo stesso qualunque sia l'indice i fissato. Inoltre, si può ottenere il determinante con una formula analoga alla (A.2) fissando un qualunque indice di colonna j e facendo la sommatoria rispetto agli indici di riga.
In base alla formula (A.2), si deduce facilmente che il calcolo del determinante di una matrice $n \times n$ ha in generale un costo di circa $n!$ operazioni, il che lo rende proibitivo. Fanno eccezione le matrici diagonali o triangolari, per le quali il determinante è dato dal prodotto degli elementi diagonali.
Le principali proprietà del determinante sono le seguenti:

- $\det(A) \neq 0$ se e soltanto se le righe (e le colonne) di A sono linearmente indipendenti. Quindi $\det(A) = 0$ se, ad esempio, una riga o una colonna ha tutti elementi uguali a zero, o, meno banalmente, se una riga (o colonna) è multipla di un'altra o combinazione lineare di due o più delle altre;
- $\det(\alpha A) = \alpha \det(A)$;

– $\det(AB) = \det(A)\det(B)$ (Teorema di Binet);
– se si scambiano fra loro due righe (o due colonne) il determinante cambia segno;
– se ad una riga (colonna) si sostituisce una combinazione lineare fra la stessa e un'altra riga (colonna), il determinante non cambia.

Matrici invertibili e sistemi lineari

Una matrice quadrata $A \in \mathbb{R}^{n\times n}$ si dice invertibile o non singolare se esiste una matrice, indicata con A^{-1}, tale che

$$AA^{-1} = A^{-1}A = I.$$

La matrice A^{-1} si chiama matrice inversa di A e, se esiste, è unica. Si può dimostrare che una matrice è invertibile se e soltanto se le sue colonne rappresentano una base di $\mathbb{R}^n$, ovvero

$$A \text{ è invertibile } \Leftrightarrow \det(A) \neq 0$$

e in questo caso

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

Gli elementi della matrice inversa non sono in generale facilmente calcolabili perché sono definiti tramite determinanti: l'elemento di indici i, j di A^{-1} è dato da

$$\frac{(-1)^{i+j}\det(A_{ij})}{\det(A)}.$$

L'inversa è immediatamente disponibile quando la matrice A è ortogonale, nel qual caso $A^{-1} = A^T$, e quando è diagonale, $A = diag(a_{11}, a_{22}, \ldots, a_{nn})$, nel qual caso $A^{-1} = diag(a_{11}^{-1}\ \ a_{22}^{-1} \ldots a_{nn}^{-1})$.

Date due matrici A e B della stessa dimensione entrambe invertibili, la matrice AB è anch'essa invertibile e

$$(AB)^{-1} = B^{-1}A^{-1}.$$

Consideriamo ora un sistema lineare di n equazioni in n incognite:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \ldots + a_{2n}x_n &= b_2 \\ \vdots & \vdots \\ a_{i1}x_1 + a_{i2}x_2 + a_{i3}x_3 + \ldots + a_{in}x_n &= b_i \\ \vdots & \vdots \\ a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \ldots + a_{nn}x_n &= b_n. \end{cases}$$

Tenendo conto delle regole di moltiplicazione fra matrici e vettori, il sistema può essere scritto in forma compatta

$$Ax = b,$$

dove

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix}$$

è la matrice dei coefficienti,

$$b = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

è il vettore dei termini noti e

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

è il vettore delle incognite. Il problema è quindi interpretabile nel seguente modo: si tratta di trovare un vettore x che, moltiplicato per una matrice data A, dia come risultato un vettore dato b. Un'altra chiave di lettura viene dalla seguente osservazione. Indicando con $a_1, a_2, \dots, a_n$ le colonne di A, il sistema lineare può essere scritto nella forma:

$$x_1 a_1 + x_2 a_2 + \ldots + x_n a_n = b,$$

da cui segue l'interpretazione alternativa: si tratta di riuscire ad esprimere il vettore dato b come combinazione lineare di n vettori dati $a_1, a_2, \dots, a_n$; i coefficienti di questa combinazione sono le incognite $x_1, x_2, \dots, x_n$.
Il sistema lineare $Ax = b$ ha soluzione unica se e soltanto se $\det(A) \neq 0$, ovvero se A è invertibile. In questo caso si ottiene:

$$Ax = b \;\; \Rightarrow \;\; A^{-1}Ax = A^{-1}b \;\; \Rightarrow \;\; Ix = A^{-1}b \;\; \Rightarrow \;\; x = A^{-1}b.$$

Se $det(A) = 0$, il sistema può essere inconsistente (nessuna soluzione) o indeterminato (infinite soluzioni). In particolare, un sistema omogeneo

$$Ax = 0$$

con matrice A invertibile ammette l'unica soluzione $x = 0$. Viceversa, un sistema omogeneo può ammettere soluzioni diverse da zero se e soltanto se $\det(A) = 0$.

Autovalori e autovettori di matrici quadrate

Data una matrice $A \in \mathbb{R}^{n\times n}$, un vettore $\bar{x} \in \mathbb{R}^n$ diverso da zero è detto autovettore di A se esiste uno scalare $\bar{\lambda}$, reale o complesso, tale che $A\bar{x} = \bar{\lambda}\bar{x}$. In questo caso, $\bar{\lambda}$ è detto autovalore di A corrispondente a $\bar{x}$. Se $\bar{\lambda}$ è un autovalore di A, per definizione il sistema lineare $(A - \bar{\lambda}I)x = 0$ ha almeno una soluzione diversa da zero, $\bar{x}$, e quindi $\det(A - \bar{\lambda}I) = 0$. Da questo segue che gli autovalori di A possono essere definiti anche come le radici dell'equazione

$$\det(A - \lambda I) = 0. \tag{A.3}$$

La funzione $\det(A - \lambda I)$ è un polinomio in λ di grado n, detto polinomio caratteristico di A, ed ha quindi n radici, reali o complesse, $\lambda_1, \lambda_2, \ldots, \lambda_n$. La (A.3) è detta equazione caratteristica di A.

Elenchiamo alcune fra le principali proprietà degli autovalori:

– A è singolare se e soltanto se $\lambda = 0$ è un suo autovalore;
– Se A è non singolare, gli autovalori di A^{-1} sono i reciproci degli autovalori di A;
– A e A^T hanno gli stessi autovalori;
– se A è simmetrica, i suoi autovalori sono tutti reali;
– se A è simmetrica e tutti gli autovalori sono positivi, A è definita positiva.

L'insieme degli autovalori di A è detto spettro di A e

$$\rho(A) = \max_{1\le i\le n} |\lambda_i| \tag{A.4}$$

è il raggio spettrale di A.

Norme vettoriali

In molte occasioni si ha la necessità di misurare la "lunghezza", o "grandezza", di un vettore. Ad esempio, dati $x, y \in \mathbb{R}^n$, si può voler quantificare quanto x e y sono vicini, ovvero quanto è "grande" la loro differenza $x - y$. Nel caso scalare, quando x e y sono due numeri, c'è un solo modo di misurare la grandezza di un numero, e cioè tramite il valore assoluto. Per misurare i vettori non c'è un solo modo: si può usare una qualsiasi norma, ovvero una qualsiasi funzione che associ ad ogni vettore x un numero, indicato con $\|x\|$, rispettando i seguenti vincoli, che rispecchiano le proprietà fondamentali della funzione valore assoluto:

1) $\|x\| \ge 0$ e $\|x\| = 0 \Leftrightarrow x = 0$;

2) $\|\alpha x\| = |\alpha|\|x\|$;

3) $\|x + y\| \le \|x\| + \|y\|$.

La proprietà 1) dice che l'unico vettore di lunghezza uguale a zero è il vettore con componenti tutte uguali a zero; tutti gli altri vettori hanno lunghezza positiva.

La proprietà 2) assicura molte relazioni che il nostro intuito dice debbano essere soddisfatte. Ad esempio, essa assicura che x e $-x$ hanno la stessa lunghezza e che il vettore $10x$ ha lunghezza 10 volte superiore a quella di x, mentre il vettore $0.5x$ ha lunghezza pari alla metà di quella di x.

La proprietà 3) è nota come disuguaglianza triangolare ed esprime la ben nota proprietà geometrica per cui la lunghezza di un lato di un triangolo non è mai superiore alla somma delle lunghezze degli altri due lati.

Gli studenti conoscono, probabilmente sotto il nome di "modulo", la norma euclidea o norma-2, definita da:

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

In realtà ci sono infiniti modi di definire norme vettoriali. Le norme vettoriali più frequentemente usate nella pratica, oltre alla norma-2, sono la norma-1 e la norma-∞ definite da

$$\begin{aligned} norma-1 \quad &: \quad \|x\|_1 = \textstyle\sum_{i=1}^{n} |x_i| \\ norma-\infty \quad &: \quad \|x\|_\infty = \max_{1\le i\le n} |x_i|. \end{aligned}$$

Queste norme vettoriali sono tutte equivalenti, nel senso che comunque se ne scelgano due, diciamo $\|\cdot\|_\alpha$ e $\|\cdot\|_\beta$, esistono due numeri m e M tali che

$$m\|x\|_\alpha \le \|x\|_\beta \le M\|x\|_\alpha.$$

Ad esempio,

$$\begin{aligned} \|x\|_\infty &\le \|x\|_1 \le n\|x\|_\infty \\ \|x\|_\infty &\le \|x\|_2 \le \sqrt{n}\|x\|_\infty \\ \tfrac{1}{\sqrt{n}}\|x\|_1 &\le \|x\|_2 \le \|x\|_1. \end{aligned}$$

Così, per valori di n non eccessivamente grandi, se un vettore è "piccolo" o "grande" secondo una certa norma, lo è anche secondo le altre norme, anche se i valori specifici di queste norme sono diversi fra loro.

Norme di matrici

Anche per le matrici si pone lo stesso problema che abbiamo considerato per i vettori: come misurare la "grandezza" di una matrice $n \times m$? Un modo

per rispondere alla domanda è il seguente: si "srotola" la matrice mettendo una dietro l'altra le righe o le colonne e si definisce "norma della matrice" una qualsiasi norma del vettore di $n \times m$ componenti così ottenuto. Quando si considera la norma-2, si ottiene in questo modo la cosiddetta norma di Frobenius per le matrici:

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

Non in tutte le situazioni questo si rivela un modo efficace per misurare la grandezza di una matrice. Definizioni di norme matriciali diverse nascono dal vedere una matrice $n \times m$ come una funzione (operatore) che trasforma vettori di $\mathbb{R}^m$ in vettori di $\mathbb{R}^n$: ad un dato $x \in \mathbb{R}^m$ questo operatore associa il vettore $y = Ax$. In questa ottica un modo generale per definire una norma matriciale diventa il seguente. Si fissa una qualunque norma vettoriale $\|\cdot\|_p$ e si definisce la norma matriciale indotta con la formula

$$\|A\|_p = max_{\|x\|_p=1} \|Ax\|_p.$$

Tutte le norme così definite godono delle seguenti proprietà:

1) $\|A\|_p \geq 0$ e $\|A\|_p = 0 \Leftrightarrow A = 0$;

2) $\|\alpha A\|_p = |\alpha| \|A\|_p$;

3) $\|A + B\|_p \leq \|A\|_p + \|B\|_p$;

4) $\|AB\|_p \leq \|A\|_p \|B\|_p$ (in particolare $\|Ax\|_p \leq \|A\|_p \|x\|_p$);

5) $\|I\|_p = 1$.

Le norme matriciali indotte da quelle vettoriali sono spesso dette norme naturali. L'equivalenza fra le norme vettoriali si estende naturalmente a queste norme matriciali.
Un importante teorema relativo alle norme naturali afferma che il raggio spettrale $\rho(A)$ è l'estremo inferiore dell'insieme delle norme naturali di A, cioè per ogni $\epsilon > 0$ esiste una norma naturale $\|A\|_p$, dipendente da ϵ, tale che

$$\rho(A) \leq \|A\|_p \leq \rho(A) + \epsilon.$$

Le norme naturali che più interessano sono quelle indotte dalla norma-1, norma-2 e norma-∞ vettoriali. Si può dimostrare che $\|A\|_1$ e $\|A\|_\infty$ sono facilmente calcolabili, in quanto

$$\|A\|_1 = \max_{1 \leq j \leq m} \sum_{i=1}^{n} |a_{ij}|$$

e

$$\|A\|_\infty = \max_{1\leq i\leq n} \sum_{j=1}^{m} |a_{ij}|,$$

mentre non esiste una formula per $\|A\|_2$ di altrettanto facile calcolo. Per quest'ultima norma si può vedere che

$$\|A\|_2 = \sqrt{\rho(A^T A)},$$

dove $\rho(A^T A)$ è il raggio spettrale della matrice $A^T A$; per questo motivo, la norma-2 di matrici è anche detta norma spettrale. Per matrici simmetriche la formula si semplifica e diventa $\|A\|_2 = \rho(A)$. In ogni caso il calcolo non è in generale semplice perchè richiede la conoscenza degli autovalori di una matrice. Poiché gli autovalori sono radici di un'equazione non lineare (l'equazione caratteristica (A.3) di A), i metodi per calcolarli sono necessariamente metodi iterativi. Il lettore interessato può consultare a questo proposito [6], [15], o [20].

Bibliografia

[1] M. Abate e C. de Fabritiis, Geometria analitica con elementi di algebra lineare, McGraw Hill, 2006

[2] T.M. Apostol, Calcolo, Vol. 1^o, Analisi 1, Boringheri, 1978

[3] T.M. Apostol, Calcolo, Vol. 3^o, Analisi 2, Boringheri, 1978

[4] S. Arrhenius, On the influence of carbonic acid in the air upon the temperature of the ground, the London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science, series V, 41 (1896), 237-276

[5] R. Bevilacqua, D. Bini, M. Capovani e O. Menchi, Metodi numerici, Zanichelli, 1992

[6] D. Bini, M. Capovani e O. Menchi, Metodi numerici per l'algebra lineare, Zanichelli, 1988

[7] S. Boyd e L. Vanderberghe, Convex optimization, Cambridge University press, 2004

[8] J.L. Chabert, A history of algorithms. Form the pebble to the microchip, Springer, 1999

[9] S.C. Chapra e R.P. Canale, Metodi Numerici per l'ingegneria, McGraw-Hill, 1988

[10] W. Cheney e D. Kincaid, Numerical mathematics and computing, 5^a edizione, Brooks/Cole Publishing Co., 2004

[11] S.D. Conte e C. de Boor, Elementary numerical analysis. An algorithmic approach, McGraw-Hill Int., 1981

[12] C. de Boor, A practical guide to splines, Series in Applied Mathemathical Sciences, Springer, 2001

[13] T.J. Dekker, Finding a zero by means of successive linear interpolation, in Constructive aspects of the fundamental theorem of algebra, B. Dejon e P. Henrici editori, Whiley-Interscience, NY, 1969

[14] J. E. Dennis e R. B. Schnabel, Numerical methods for unconstrained optimization and nonlinear equations, SIAM, 1996

[15] F. Fontanella e A. Pasquali, Calcolo numerico. Metodi ed algoritmi, Pitagora editrice, 1982

[16] W. Gander e W. Gautschi, Adaptive quadrature revisited, BIT, 40 (2000) 84-101

[17] P.E. Gill, W. Murray e M.H.Wright, Numerical Linear Algebra and Optimization, vol.1, Addison Weslwy Pub. Comp., 1991

[18] D. Goldberg, What every computer scientist should know about floating-point arithmetic, ACM Computer Surveys, 23 (1991), 5-48

[19] H.H. Goldstine, A history of numerical analysis from the 16th to the 19th century, Springer, 1977
[20] G.H. Golub e C.F. Van Loan, Matrix computations, Johns Hopkins University Press, Baltimore, 1983
[21] M.R. Hestenes e E. Stiefel, Methods of conjugate gradients for solving linear systems, J. Res. Nat. Bureau Standards, 49 (1952), 409-436
[22] N.J. Higham, Accuracy and stability of numerical algorithms, SIAM, 1996
[23] D. Kahaner, C. Moler e S. Nash, Numerical methods and software, Prentice-Hall International Ed., 1989
[24] C.T. Kelley, Iterative methods for linear and nonlinear equations, SIAM, 1995
[25] S. Lang, Algebra lineare, Bollati Boringhieri, 1970
[26] E. Meijering, A chronology of interpolation: from ancient astronomy to modern signal and image processing, Proceedings of IEEE, 90 (2002), 319-342
[27] G. Monegato, Fondamenti di calcolo numerico, CLUT Editrice, Torino, 1988
[28] J. Nocedal e S.J. Wright, Numerical Optimization, Springer series in Operations Research, 1999
[29] M.L. Overton, Numerical computing with IEEE floating point arithmetic, SIAM, 2001
[30] W. C. Rheinboldt e J. V. Burkardt, A locally parameterized continuation process, ACM Transactions on Mathematical Software, 9 (1983), 215-235
[31] T.J. Rivlin, An introduction to approximation of functions, Blaisdell, Waltham, Mass., 1969
[32] C. Runge, Über empirische Funktionen und die Interpolation zwischen äquidistanten Ordinaten, Zeitschrift für Mathematik und Physik, 46 (1901), 224-243
[33] Y. Saad, Iterative methods for sparse linear systems, seconda edizione, SIAM, 2003
[34] S.J. Smith, Lebesgue constants in polynomial interpolation, Annales Mathematicae et Informaticae, 33 (2006), 109-123
[35] G.W. Stewart, Afternotes in numerical analysis, SIAM, 1996
[36] L.N. Trefethen e D. Bau III, Numerical Linear Algebra, SIAM, 1997
[37] J.H. Wilkinson, The evaluation of the zeros of ill-conditioned polynomials, Numer. Math, 1 (1959), 150-180
[38] J.H. Wilkinson, Rounding errors in Algebraic Processes, Notes on Applied Science No 32, Her Majesty's Stationery Office, London, 1963, ristampato da Dover, New York, nel 1994
[39] J.H. Wilkinson e C. Reinsch, Handbook for automatic computation, Vol. 2, Linear Algebra, Springer-Verlag, NY, 1971