

A. Quarteroni
F. Saleri

Introduzione al **CALCOLO SCIENTIFICO**

Esercizi e problemi risolti con MATLAB



3^a edizione



Springer

TEXAS

Introduzione al Calcolo Scientifico

A. Quarteroni
F. Saleri

Introduzione al Calcolo Scientifico

Esercizi e problemi risolti con MATLAB

3^a edizione



ALFIO QUARTERONI
MOX - Dipartimento di Matematica
Politecnico di Milano e
Ecole Polytechnique Fédérale de Lausanne

FAUSTO SALERI
MOX - Dipartimento di Matematica
Politecnico di Milano

Le simulazioni numeriche riportate in copertina sono state realizzate presso
l'Ecole Polytechnique Fédérale del Lausanne
da Nicola Parolini e Mark L. Sawley

Springer-Verlag Italia fa parte di Springer Science+Business Media

springer.com

© Springer-Verlag Italia, Milano 2006

ISBN 10 88-470-0480-2

ISBN 13 978-88-470-0480-1

Quest'opera è protetta dalla legge sul diritto d'autore. Tutti i diritti, in particolare quelli relativi alla traduzione, alla ristampa, all'uso di figure e tavole, alla citazione orale, alla trasmissione radiofonica o televisiva, alla riproduzione su microfilm o in database, alla diversa riproduzione in qualsiasi altra forma (stampa o elettronica) rimangono riservati anche nel caso di utilizzo parziale. Una riproduzione di quest'opera, oppure di parte di questa, è anche nel caso specifico solo ammessa nei limiti stabiliti dalla legge sul diritto d'autore, ed è soggetta all'autorizzazione dell'Editore. La violazione delle norme comporta le sanzioni previste dalla legge.

L'utilizzo di denominazioni generiche, nomi commerciali, marchi registrati, ecc, in quest'opera, anche in assenza di particolare indicazione, non consente di considerare tali denominazioni o marchi liberamente utilizzabili da chiunque ai sensi della legge sul marchio.

Riprodotto da copia camera-ready fornita dagli Autori
Progetto grafico della copertina: Simona Colombo, Milano
Stampato in Italia: Signum Srl, Bollate (Milano)

Prefazione

Questo testo è una introduzione al Calcolo Scientifico. In esso vengono illustrati metodi numerici per la risoluzione con il calcolatore di alcune classi di problemi della Matematica che non si possono risolvere con “carta e penna”. In particolare, mostreremo come calcolare gli zeri o l’integrale di funzioni continue, come risolvere sistemi lineari, come approssimare funzioni con polinomi, ma anche come trovare delle buone approssimazioni della soluzione di equazioni differenziali ordinarie e di problemi ai limiti.

A tale scopo, nel Capitolo 1 illustreremo le principali regole del gioco che i calcolatori seguono quando memorizzano i numeri reali ed i numeri complessi, i vettori e le matrici, e come operano con essi.

Al fine di rendere concreta ed incisiva la nostra trattazione adotteremo il linguaggio di programmazione MATLAB®¹ come fedele compagno di viaggio. Scopriremo gradualmente i suoi principali comandi e costrutti. Grazie ad esso mostreremo come rendere esecutivi tutti gli algoritmi che via via introdurremo e potremo immediatamente fornire un riscontro “quantitativo” alle loro proprietà teoriche, quali stabilità, accuratezza e complessità. Saremo inoltre in grado di risolvere al calcolatore numerosi quesiti e problemi che verranno posti attraverso esercizi ed esempi, anche con riferimento a specifiche applicazioni.

Per rendere più agevole la lettura useremo alcuni accorgimenti tipografici.² A margine del testo riporteremo il comando MATLAB in corrispondenza della linea in cui tale comando è richiamato per la prima volta. Inoltre, useremo il simbolo  per segnalare degli esercizi, il sim-

¹ MATLAB è un marchio registrato di The MathWorks, Inc. Per ulteriori informazioni su MATLAB si prega di contattare: The MathWorks, 3 Apple Hill Drive, Natick, MA 01760 20098, Tel: 001+508-647-7000, Fax: 001+508-647-7001.

² Per le icone utilizzate si veda il sito <http://www.iconarchive.com>

bolo per segnalare un programma ed il simbolo per attirare l'attenzione su un comportamento critico o sorprendente di un algoritmo o di un procedimento. Le formule particolarmente rilevanti sono incorniciate. Infine, il simbolo segnala la presenza di una scheda riassuntiva dei concetti e delle conclusioni esposte nei paragrafi immediatamente precedenti.

Alla fine di ogni capitolo è situato un paragrafo nel quale si menzionano gli argomenti non trattati e si indicano dei riferimenti bibliografici per l'approfondimento del materiale presentato. Le soluzioni di tutti gli esercizi sono raccolte nel capitolo conclusivo.

Faremo spesso riferimento ai testi [QSS04] e [QSS02] per i rimandi di carattere teorico o per gli approfondimenti, mentre per una descrizione completa di MATLAB rimandiamo a [HH00]. Tutti i programmi presenti nel volume possono essere trovati all'indirizzo:

mox.polimi.it/qss.

Questo testo è espressamente concepito per i corsi brevi del nuovo ordinamento delle Facoltà di Ingegneria e di Scienze. Non è richiesto nessun particolare requisito, fatta eccezione ovviamente per un corso elementare di Analisi Matematica.

In ogni caso nel primo capitolo richiamiamo i principali risultati di Analisi e di Geometria di cui verrà fatto uso nel testo. Gli argomenti meno elementari, non indispensabili cioè ad un percorso formativo introduttivo, sono segnalati con il simbolo .

Questa terza edizione si differenzia dalla precedente per la presenza di un maggior numero di problemi applicativi e per diverse integrazioni riguardanti la risoluzione di sistemi lineari e non lineari e l'approssimazione di equazioni differenziali ordinarie. Desideriamo ringraziare tutti i nostri colleghi e collaboratori del MOX (Centro di Modellistica e Calcolo Scientifico) del Politecnico di Milano che hanno consentito di rendere più ricco ed interessante questo volume. Ringraziamo inoltre Paola Gervasio, Carlo D'Angelo e Nicola Parolini che si sono prestati ad un'attenta rilettura della terza edizione, contribuendo a migliorarne la chiarezza espositiva.

Losanna e Milano, febbraio 2006

Alfio Quarteroni, Fausto Saleri

Indice

1	Quel che non si può non sapere	1
1.1	I numeri reali	1
1.1.1	Come si rappresentano	2
1.1.2	Come si opera con i numeri floating-point	4
1.2	I numeri complessi	6
1.3	Le matrici	8
1.3.1	I vettori	12
1.4	Le funzioni	14
1.4.1	Gli zeri	15
1.4.2	I polinomi	17
1.4.3	L'integrale e la derivata	19
1.5	Errare non è solo umano	21
1.5.1	Parliamo di costi	24
1.6	Qualche parola in più su MATLAB	26
1.6.1	Statements MATLAB	29
1.6.2	Programmare in MATLAB	30
1.7	Cosa non vi abbiamo detto	34
1.8	Esercizi	34
2	Equazioni non lineari	37
2.1	Il metodo di bisezione	39
2.2	Il metodo di Newton	42
2.2.1	Come arrestare il metodo di Newton	44
2.2.2	Il metodo di Newton per sistemi di equazioni non lineari	47
2.3	Iterazioni di punto fisso	49
2.3.1	Come arrestare un'iterazione di punto fisso	53
2.4	Accelerazione con il metodo di Aitken	54
2.5	Polinomi algebrici	58
2.5.1	Il metodo di Hörner	59

2.5.2 Il metodo di Newton-Hörner	61
2.6 Cosa non vi abbiamo detto	64
2.7 Esercizi	65
3 Approssimazione di funzioni e di dati	69
3.1 Interpolazione	71
3.1.1 Interpolazione polinomiale di Lagrange	73
3.1.2 Interpolazione di Chebyshev	78
3.1.3 Interpolazione trigonometrica e FFT	79
3.2 Interpolazione lineare composita	84
3.3 Approssimazione con funzioni <i>spline</i>	86
3.4 Il metodo dei minimi quadrati	90
3.5 Cosa non vi abbiamo detto	95
3.6 Esercizi	96
4 Differenziazione ed integrazione numerica	99
4.1 Approssimazione delle derivate	101
4.2 Integrazione numerica	103
4.2.1 La formula del punto medio	104
4.2.2 La formula del trapezio	106
4.2.3 La formula di Simpson	107
4.3 Formule di quadratura interpolatorie	109
4.4 La formula di Simpson adattiva	113
4.5 Cosa non vi abbiamo detto	117
4.6 Esercizi	117
5 Sistemi lineari	121
5.1 Il metodo di fattorizzazione LU	124
5.2 La tecnica del pivoting	132
5.3 Quanto è accurata la fattorizzazione LU?	134
5.4 Come risolvere un sistema tridiagonale	138
5.5 Sistemi sovradeterminati	139
5.6 Cosa si nasconde nel comando \ di MATLAB	141
5.7 Metodi iterativi	142
5.7.1 Come costruire un metodo iterativo	143
5.8 Il metodo di Richardson e del gradiente	148
5.9 Il metodo del gradiente coniugato	150
5.9.1 Il caso non simmetrico	153
5.10 Quando arrestare un metodo iterativo?	153
5.11 Ed ora: metodi diretti o iterativi?	156
5.11.1 Un sistema lineare sparso	156
5.11.2 Un sistema lineare con banda estesa	157
5.11.3 Un sistema con matrice piena	158
5.11.4 Un sistema lineare con matrice sparsa non a banda e non simmetrica	159

5.11.5 In conclusione	160
5.12 Cosa non vi abbiamo detto	161
5.13 Esercizi	161
6 Autovalori ed autovettori	165
6.1 Il metodo delle potenze	168
6.1.1 Analisi di convergenza	171
6.2 Generalizzazione del metodo delle potenze	172
6.3 Come calcolare lo shift	174
6.4 Calcolo di tutti gli autovalori	177
6.5 Cosa non vi abbiamo detto	181
6.6 Esercizi	181
7 Equazioni differenziali ordinarie	185
7.1 Il problema di Cauchy	188
7.2 I metodi di Eulero	189
7.2.1 Analisi di convergenza	191
7.3 Il metodo di Crank-Nicolson	195
7.4 Zero-stabilità	197
7.5 Stabilità su intervalli illimitati	199
7.5.1 La regione di assoluta stabilità	202
7.5.2 L'assoluta stabilità controlla le perturbazioni	203
7.6 Metodi di ordine elevato	209
7.7 I metodi predictor-corrector	214
7.8 Sistemi di equazioni differenziali	216
7.9 Alcuni esempi	222
7.9.1 Il pendolo sferico	222
7.9.2 Il problema dei tre corpi	225
7.9.3 Alcuni problemi stiff	227
7.10 Cosa non vi abbiamo detto	231
7.11 Esercizi	231
8 Metodi numerici per problemi ai limiti	235
8.1 Approssimazione di problemi ai limiti	238
8.1.1 Approssimazione con differenze finite	238
8.1.2 Approssimazione con elementi finiti	240
8.2 Le differenze finite in 2 dimensioni	243
8.2.1 Consistenza e convergenza	249
8.3 Che cosa non vi abbiamo detto	251
8.4 Esercizi	251
9 Soluzione degli esercizi proposti	255
9.1 Capitolo 1	255
9.2 Capitolo 2	258
9.3 Capitolo 3	264

9.4 Capitolo 4	268
9.5 Capitolo 5	273
9.6 Capitolo 6	278
9.7 Capitolo 7	281
9.8 Capitolo 8	290
Riferimenti bibliografici.....	295
Indice dei programmi MATLAB	299
Indice analitico	301

1

Quel che non si può non sapere

In questo testo si incontreranno continuamente entità matematiche elementari che dovrebbero far parte del bagaglio culturale del lettore, ma il cui ricordo è talvolta appannato.

Approfittiamo di questo capitolo introduttivo per rinfrescare quelle nozioni che saranno utili nella trattazione successiva. Non solo, introdurremo anche nuovi concetti tipici del Calcolo Scientifico ed inizieremo ad esplorarne il significato e l'utilità avvalendoci del programma MATLAB. Di esso proporremo nel paragrafo 1.6 una breve introduzione, rimandando al manuale [HH00] per una sua completa descrizione.

Si tratta dunque di un capitolo nel quale sono condensate nozioni proprie di Analisi, Algebra e Geometria, riconfigurate in funzione del loro utilizzo nel calcolo scientifico.

1.1 I numeri reali

Cosa sia l'insieme \mathbb{R} dei numeri reali è a tutti noto. Forse meno nota è la modalità di trattamento dei numeri reali da parte di un calcolatore. Essendo impossibile rappresentare su una macchina (le cui risorse sono necessariamente finite) l'infinità dei numeri reali, ci si dovrà accontentare di rappresentarne soltanto un sottoinsieme di dimensione finita che indicheremo con \mathbb{F} e chiameremo insieme dei *numeri floating-point*. Peraltro \mathbb{F} è caratterizzato da proprietà diverse rispetto a quelle dell'insieme \mathbb{R} , come vedremo nel paragrafo 1.1.2. Ciò è dovuto al fatto che ogni singolo numero reale x viene rappresentato dalla macchina con un numero arrotondato, che si indica con $fl(x)$ e viene detto *numero macchina*, che non coincide necessariamente con il numero x di partenza.

1.1.1 Come si rappresentano

Per renderci conto delle differenze fra \mathbb{R} e \mathbb{F} diamo uno sguardo, tramite alcuni semplici esperimenti MATLAB, al modo con il quale un calcolatore (un PC ad esempio) tratta i numeri reali. Utilizzare MATLAB piuttosto che un altro linguaggio di programmazione (come, ad esempio, Fortran o C) è solo una scelta di comodo: il risultato degli esperimenti dipende infatti in modo essenziale da come il calcolatore lavora e, in misura assai minore, dal linguaggio utilizzato.

Consideriamo il numero razionale $x = 1/7$, la cui rappresentazione decimale è $0.\overline{142857}$. Si osservi che il punto separa la parte decimale da quella intera ed è dunque sostitutivo della virgola. Tale rappresentazione è infinita, nel senso che esistono infinite cifre non nulle dopo il punto. Per rappresentare in macchina tale numero introduciamo dopo il *prompt* (il simbolo **>>**) la frazione 1/7 ottenendo

```
>> 1/7
ans =
0.1429
```

cioè un numero costituito apparentemente da sole 4 cifre decimali, l'ultima delle quali inesatta rispetto alla quarta cifra del numero reale. Se ora digitiamo 1/3 troviamo 0.3333, nel quale anche la quarta cifra è esatta. Questo comportamento è dovuto al fatto che i numeri reali sul calcolatore vengono *arrotondati*, viene cioè memorizzato solo un numero fissato a priori di cifre decimali ed inoltre, l'ultima cifra decimale memorizzata risulta incrementata di 1 rispetto alla corrispondente cifra decimale del numero originario qualora la cifra successiva in quest'ultimo risulti maggiore od uguale a 5.

La prima considerazione che potremmo fare è che usare solo 4 cifre decimali per rappresentare i numeri è oltremodo grossolano. Possiamo tranquillizzarci: il numero che abbiamo “visto” è solo un possibile formato di *output* del sistema che non coincide con la sua rappresentazione interna che utilizza ben 16 cifre decimali (più altri correttivi dei quali non è qui il caso di discutere). Lo stesso numero assume espressioni diverse se fatto precedere da opportune dichiarazioni di formato: ad esempio,

format per 1/7, alcuni possibili formati di *output* sono

```
format long    produce 0.14285714285714,
format short e   "    1.4286e - 01,
format long e   "    1.428571428571428e - 01,
format short g   "    0.14286,
format long g   "    0.142857142857143.
```

Talune di queste rappresentazioni (ad esempio, il formato **long e**) sono più fedeli di altre al formato interno dell'elaboratore. Quest'ultimo memorizza generalmente i numeri nel modo seguente

$$x = (-1)^s \cdot (0.a_1a_2 \dots a_t) \cdot \beta^e = (-1)^s \cdot m \cdot \beta^{e-t}, \quad a_1 \neq 0, \quad (1.1)$$

dove s vale 0 o 1, β (un numero intero positivo maggiore od uguale a 2) è la *base*, m è un intero detto *mantissa* la cui lunghezza t è il numero massimo di cifre a_i (con $0 \leq a_i \leq \beta - 1$) memorizzabili ed e è un numero intero detto *esponente*. Il formato `long` e' quello che più assomiglia a questa rappresentazione (la e sta proprio per esponente, le cui cifre, precedute dal segno, in questo formato vengono riportate immediatamente a destra del carattere e). I numeri di macchina nel formato (1.1) sono detti numeri *floating-point* essendo variabile la posizione del punto decimale. Le cifre $a_1a_2 \dots a_p$ (con $p \leq t$) vengono generalmente chiamate le prime p cifre significative di x .

La condizione $a_1 \neq 0$ impedisce che lo stesso numero possa avere più rappresentazioni. Ad esempio, senza questa condizione, $1/10$ in base 10 potrebbe essere rappresentato come $0.1 \cdot 10^0$ o $0.01 \cdot 10^1$ e così via.

L'insieme \mathbb{F} è dunque completamente caratterizzato dalla base β , dal numero di cifre significative t e dall'intervallo (L, U) (con $L < 0$ ed $U > 0$) di variabilità dell'esponente e . Viene perciò anche indicato con $\mathbb{F}(\beta, t, L, U)$: ad esempio, in MATLAB si utilizza $\mathbb{F}(2, 53, -1021, 1024)$ (in effetti 53 cifre significative in base 2 corrispondono alle 15 cifre significative mostrate in base 10 da MATLAB con il comando `format long`).

Fortunatamente l'inevitabile *errore di roundoff* che si commette sostituendo ad un numero reale $x \neq 0$ il suo rappresentante $fl(x)$ in \mathbb{F} , è generalmente piccolo, avendosi

$$\frac{|x - fl(x)|}{|x|} \leq \frac{1}{2} \epsilon_M, \quad (1.2)$$

dove $\epsilon_M = \beta^{1-t}$ rappresenta la distanza fra 1 ed il più vicino numero *floating-point* maggiore di 1. Si osservi che ϵ_M dipende da β e da t . Ad esempio, in MATLAB, dove ϵ_M è calcolabile con il comando `eps`, si ha $\epsilon_M = 2^{-52} \simeq 2.22 \cdot 10^{-16}$. Si osservi che nella (1.2) si stima l'*errore relativo* su x , certamente più significativo dell'*errore assoluto* $|x - fl(x)|$, in quanto quest'ultimo non tiene conto dell'ordine di grandezza di x .

Il numero 0 non appartiene a \mathbb{F} , poiché per esso $a_1 = 0$ nella (1.1): viene pertanto trattato a parte. Essendo inoltre L ed U finiti non si potranno rappresentare numeri in valore assoluto arbitrariamente piccoli o grandi. Di fatto, il più piccolo ed il più grande numero positivo di \mathbb{F} sono

$$x_{min} = \beta^{L-1}, \quad x_{max} = \beta^U (1 - \beta^{-t}).$$

In MATLAB attraverso i comandi `realmin` e `realmax` è possibile determinare tali valori che sono

$$x_{min} = 2.225073858507201 \cdot 10^{-308}, \\ x_{max} = 1.7976931348623158 \cdot 10^{+308}.$$

`eps`

`realmin`
`realmax`

Un numero positivo inferiore ad x_{min} produce una segnalazione di *underflow* e viene trattato o come 0 o in un modo speciale (si veda ad esempio [QSS04], capitolo 2). Un numero positivo maggiore di x_{max} produce invece una segnalazione di *overflow* e viene memorizzato nella variabile `Inf` (la rappresentazione al calcolatore dell'infinito positivo).

Il fatto che x_{min} e x_{max} siano gli estremi di un intervallo molto vasto della retta reale non deve trarre in inganno: i numeri di \mathbb{F} sono molto addensati vicino a x_{min} , diventando sempre più radi all'avvicinarsi di x_{max} . Ci si può rendere immediatamente conto di questa proprietà osservando che il numero di \mathbb{F} immediatamente precedente a x_{max} e quello immediatamente successivo a x_{min} sono rispettivamente

$$\begin{aligned}x_{max}^- &= 1.7976931348623157 \cdot 10^{+308} \\x_{min}^+ &= 2.225073858507202 \cdot 10^{-308},\end{aligned}$$

dunque $x_{min}^+ - x_{min}^- \simeq 10^{-323}$, mentre $x_{max} - x_{max}^- \simeq 10^{292}$ (!). La distanza relativa resta comunque piccola, come si deduce dalla (1.2).

1.1.2 Come si opera con i numeri floating-point

Veniamo alle operazioni elementari fra numeri di \mathbb{F} : essendo \mathbb{F} soltanto un sottoinsieme di \mathbb{R} , esse non godono di tutte le proprietà delle analoghe operazioni definite su \mathbb{R} . Precisamente, permangono valide la commutatività fra addendi (cioè $fl(x + y) = fl(y + x) \forall x, y \in \mathbb{F}$) o fra fattori ($fl(xy) = fl(yx) \forall x, y \in \mathbb{F}$), ma vengono violate l'unicità dello zero, la proprietà associativa e distributiva.

Per renderci conto della non unicità dello zero, assegnamo ad una variabile `a` un valore qualunque, ad esempio 1, ed eseguiamo il seguente programma

```
>> a = 1; b=1; while a+b ~= a; b=b/2; end
```

In esso la variabile `b` viene dimezzata ad ogni passo finché la somma di `a` e di `b` si mantiene diversa (non uguale, `~=`) da `a`. Evidentemente, se stessimo utilizzando i numeri reali, il programma non si arresterebbe mai; invece nel nostro caso il programma si arresta dopo un numero finito di passi e fornisce per `b` il seguente valore: $1.1102e-16 = \epsilon_M/2$. Esiste dunque almeno un numero `b` diverso da 0 tale che `a+b=a`. Questo può accadere per la struttura dell'insieme \mathbb{F} , costituito come abbiamo visto da elementi isolati. In generale in MATLAB, per un numero positivo `a` qualsiasi, il numero `a+eps(a)` è il più piccolo numero di \mathbb{F} successivo ad `a`. Di conseguenza, comando ad `a` un numero `b` minore di `eps(a)` si avrà `a+b` uguale ad `a`.

Per quanto riguarda l'associatività, essa è violata quando si presenta una situazione di *overflow* o di *underflow*: prendiamo ad esempio `a=1.0e+308`, `b=1.1e+308` e `c=-1.001e+308` ed eseguiamone la somma in due modi diversi. Troviamo

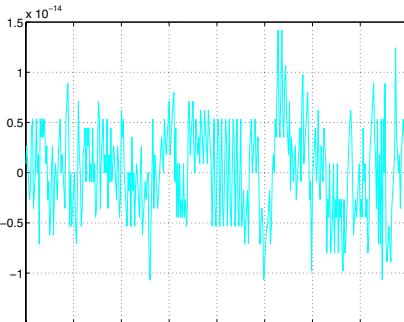


Figura 1.1. Andamento oscillante della funzione (1.3) causato dagli errori di cancellazione di cifre significative

$$a + (b + c) = 1.0990e+308, \quad (a + b) + c = \text{Inf.}$$

Questo è un caso particolare del fenomeno che si verifica quando si sommano tra loro numeri che hanno all'incirca lo stesso modulo, ma segno opposto. In tal caso il risultato della somma può essere assai impreciso e ci si riferisce a questa situazione con l'espressione *cancellazione di cifre significative*. Ad esempio, consideriamo in MATLAB la seguente operazione $((1+x)-1)/x$ con $x \neq 0$, il cui risultato esatto è ovviamente 1 per ogni $x \neq 0$. Troviamo invece

```
>> x = 1.e-15; ((1+x)-1)/x
ans =
1.1102
```

Come si nota il risultato ottenuto è molto inaccurato nel senso che l'errore assoluto è grande.

Un ulteriore esempio di cancellazione di cifre significative si incontra nella valutazione della funzione

$$f(x) = x^7 - 7x^6 + 21x^5 - 35x^4 + 35x^3 - 21x^2 + 7x - 1 \quad (1.3)$$

in 401 punti equispaziati nell'intervallo $[1 - 2 \cdot 10^{-8}, 1 + 2 \cdot 10^{-8}]$. Si ottiene il grafico caotico riportato in Figura 1.1 (l'andamento reale è quello di $(x-1)^7$ cioè una funzione sostanzialmente nulla e costante in questo minuscolo intorno di $x = 1$). Vedremo nel paragrafo 1.4 i comandi utilizzati per generare il grafico.

Si noti infine che in \mathbb{F} non possono trovar posto le cosiddette forme indeterminate come $0/0$ o ∞/∞ : la loro comparsa nel corso dei calcoli produce i cosiddetti *non numeri* (`NaN` in MATLAB) per i quali non possono più valere le usuali regole di calcolo.



Osservazione 1.1 È vero che gli errori di arrotondamento sono generalmente piccoli, tuttavia, se ripetuti all'interno di algoritmi lunghi e complessi, possono

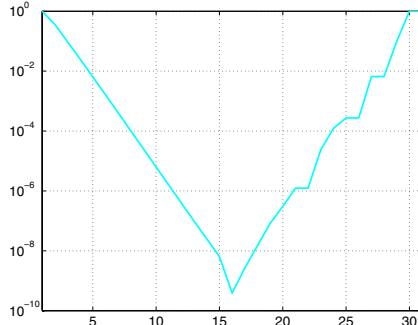


Figura 1.2. Logaritmo dell'errore relativo $|\pi - z_n|/|\pi|$ al variare di n

avere effetti catastrofici. Due casi eclatanti riguardano l'esplosione del missile Arianne il 4 giugno del 1996, causata dalla comparsa di un *overflow* nel computer di bordo, e quello di un missile americano Patriot caduto, durante la prima guerra del Golfo del 1991, su una caserma americana a causa di un errore di arrotondamento nel calcolo della sua traiettoria (per una descrizione accurata di questi e di altri casi, si veda ad esempio, mox.polimi.it/qs).

Un esempio con conseguenze meno catastrofiche, ma comunque inquietanti, è costituito dalla seguente successione

$$z_2 = 2, z_{n+1} = \sqrt{1 - \sqrt{1 - 4^{1-n} z_n^2}}, n = 2, 3, \dots \quad (1.4)$$

la quale converge a π quando n tende all'infinito. Se utilizziamo MATLAB per calcolare z_n , troveremo che l'errore relativo fra π e z_n decresce per 15 iterazioni per poi cominciare a crescere a causa degli errori di arrotondamento (come mostrato in Figura 1.2).



Si vedano gli Esercizi 1.1-1.2.

1.2 I numeri complessi

I numeri complessi, il cui insieme viene indicato con il simbolo \mathbb{C} , hanno la forma $z = x + iy$, dove $i = \sqrt{-1}$ è l'unità immaginaria (cioè $i^2 = -1$), mentre $x = \operatorname{Re}(z)$ e $y = \operatorname{Im}(z)$ sono rispettivamente la parte reale e la parte immaginaria di z . Generalmente essi vengono rappresentati dal calcolatore come coppie di numeri reali.

A meno che non vengano ridefinite diversamente, le variabili MATLAB `i` e `j` denotano indifferentemente l'unità immaginaria. Per introdurre un numero complesso di parte reale `x` e parte immaginaria `y` basta pertanto scrivere `x+i*y`; alternativamente, si può usare il comando `complex(x,y)`. Ricordiamo anche le rappresentazioni esponenziale e trigonometrica di un numero complesso z (equivalenti grazie alla *formula di Eulero*) per cui

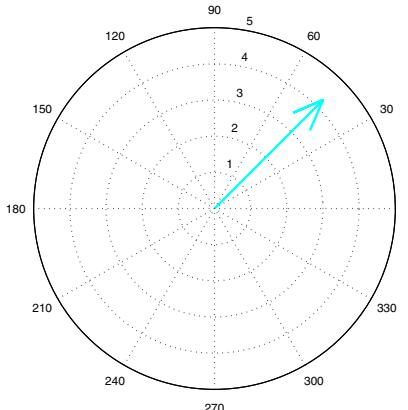


Figura 1.3. Output del comando MATLAB compass

$$z = \rho e^{i\theta} = \rho(\cos \theta + i \sin \theta), \quad (1.5)$$

essendo $\rho = \sqrt{x^2 + y^2}$ il modulo del numero complesso (esso è ottenibile tramite il comando `abs(z)`) e θ l'argomento, cioè l'angolo formato dalla semiretta con origine nello 0 e passante per z , visto come un punto di componenti (x, y) , con il semiasse positivo delle ascisse. L'angolo θ può essere trovato con il comando `angle(z)`. Pertanto la rappresentazione (1.5) si scrive

$$\text{abs}(z) * (\cos(\text{angle}(z)) + i * \sin(\text{angle}(z))).$$

Una rappresentazione grafica polare (cioè in funzione di ρ e di θ) di uno o più numeri complessi si ha con il comando `compass(z)`, dove z è un singolo numero complesso od un vettore di numeri complessi. Ad esempio, digitando

```
>> z = 3+i*3; compass(z);
```

si ottiene il grafico riportato in Figura 1.3.

Dato un numero complesso z , se ne può estrarre la parte reale e quella immaginaria con i comandi `real(z)` e `imag(z)`. Infine, il complesso coniugato $\bar{z} = x - iy$ di z , si trova semplicemente scrivendo `conj(z)`.

In MATLAB *tutte le operazioni vengono eseguite supponendo implicitamente che il risultato e gli operandi possano essere complessi*. Così non deve stupire se calcolando in MATLAB la radice cubica di -5 come $(-5)^{(1/3)}$, anziché il numero reale $-1.7099\dots$ si trovi il numero complesso $0.8550 + 1.4809i$. (Il simbolo \wedge serve per eseguire l'elevamento a potenza.) In effetti, tutti i numeri della forma $\rho e^{i(\theta+2k\pi)}$, con k intero, sono indistinguibili da $z = \rho e^{i\theta}$. Se ora calcoliamo $\sqrt[3]{z}$ troviamo $\sqrt[3]{\rho} e^{i(\theta/3+2k\pi/3)}$, vale a dire le tre radici distinte

`abs`

`angle`

`compass`

`real`
`imag`
`conj`

\wedge

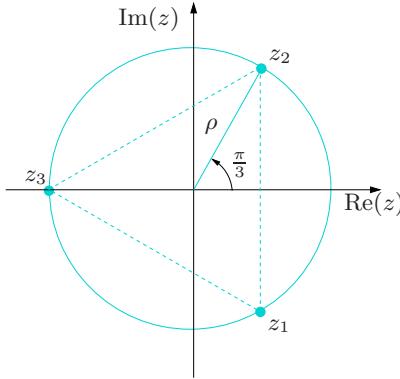


Figura 1.4. Rappresentazione nel piano di Gauss delle radici cubiche complesse di -5

$$z_1 = \sqrt[3]{\rho} e^{i\theta/3}, z_2 = \sqrt[3]{\rho} e^{i(\theta/3+2\pi/3)}, z_3 = \sqrt[3]{\rho} e^{i(\theta/3+4\pi/3)}.$$

MATLAB sceglierà come radice la prima incontrata scandendo il piano complesso in senso antiorario a partire dall'asse reale. Essendo $z = -5$ della forma $\rho e^{i\theta}$ con $\rho = 5$ e $\theta = -\pi$, le tre radici valgono

$$\begin{aligned} z_1 &= \sqrt[3]{5}(\cos(-\pi/3) + i \sin(-\pi/3)) \simeq 0.8550 - 1.4809i, \\ z_2 &= \sqrt[3]{5}(\cos(\pi/3) + i \sin(\pi/3)) \simeq 0.8550 + 1.4809i, \\ z_3 &= \sqrt[3]{5}(\cos(-\pi) + i \sin(-\pi)) \simeq -1.7100. \end{aligned}$$

La seconda è la radice prescelta (si veda la Figura 1.4 per la rappresentazione di z_1 , z_2 e z_3 nel piano di Gauss).

Ricordiamo infine che, grazie alla (1.5), si ha

$$\cos(\theta) = \frac{1}{2} (e^{i\theta} + e^{-i\theta}), \quad \sin(\theta) = \frac{1}{2i} (e^{i\theta} - e^{-i\theta}). \quad (1.6)$$

1.3 Le matrici

Se indichiamo con n e m due numeri interi positivi, una matrice A con m righe e n colonne è un insieme di $m \times n$ elementi a_{ij} con $i = 1, \dots, m$, $j = 1, \dots, n$, rappresentato dalla tabella

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}. \quad (1.7)$$

In maniera compatta scriveremo $A = (a_{ij})$. Scriveremo $A \in \mathbb{R}^{m \times n}$ se gli elementi di A sono numeri reali, $A \in \mathbb{C}^{m \times n}$ se invece sono complessi. Se inoltre $n = m$ la matrice si dice *quadrata* di dimensione n . Una matrice con una sola colonna viene detta *vettore colonna*, in contrapposizione ad una matrice con una sola riga che viene detta *vettore riga*.

In MATLAB per introdurre una matrice è sufficiente digitarne gli elementi dalla prima riga all'ultima, introducendo al termine di ogni riga il carattere di separazione `;`. Così ad esempio, il comando

```
>> A = [ 1 2 3; 4 5 6]
```

produce

```
A =
1   2   3
4   5   6
```

cioè una matrice a 2 righe e 3 colonne dagli elementi indicati. La matrice di dimensione $m \times n$ con tutti elementi nulli è indicata con 0 e costruita con `zeros(m,n)`. Il comando MATLAB `eye(m,n)` genera invece una matrice rettangolare i cui elementi sono tutti nulli ad eccezione di quelli della diagonale principale che sono pari a 1 (ricordiamo che la diagonale principale di una matrice A di dimensione $m \times n$ è l'insieme degli elementi a_{ii} con $i = 1, \dots, \min(m, n)$). Un caso particolare è il comando `eye(n)` (che è una versione abbreviata di `eye(n,n)`): esso produce una matrice quadrata di dimensione n con elementi diagonali unitari, chiamata matrice *identità* e denotata con I . Infine, con il comando `A=[]` si inizializza una matrice vuota.

`zeros`
`eye`

Sulle matrici possiamo definire alcune operazioni elementari:

1. se $A = (a_{ij})$ e $B = (b_{ij})$ sono due matrici $m \times n$, allora la *somma* di A con B è la matrice $A + B = (a_{ij} + b_{ij})$;
2. il *prodotto* di una matrice A per un numero λ (reale o complesso) è la matrice $\lambda A = (\lambda a_{ij})$.
3. il *prodotto fra due matrici* può essere eseguito soltanto se esse hanno dimensioni compatibili, precisamente se A è una matrice $m \times p$ e B è $p \times n$, per un intero positivo p . La matrice prodotto è in tal caso la matrice $C = AB$ di dimensione $m \times n$ di elementi:

$$c_{ij} = \sum_{k=1}^p a_{ik} b_{kj}, \text{ per } i = 1, \dots, m, j = 1, \dots, n.$$

Nel seguito, riportiamo un esempio di somma e prodotto di due matrici:

```
>> A=[1 2 3; 4 5 6]; B=[7 8 9; 10 11 12]; C=[13 14; 15 16; 17 18];
```

```
>> A+B
```

```
ans =
```

```
8   10   12
```

```

14   16   18
>> A*C
ans =
 94  100
229 244

```

Si noti che il tentativo di eseguire operazioni fra matrici di dimensione incompatibile porta ad un messaggio di errore. Ad esempio,

```

>> A+C
??? Error using ==> +
Matrix dimensions must agree.
>> A*B
??? Error using ==> *
Inner matrix dimensions must agree.

```

Per quanto riguarda l'*inversione* di una matrice quadrata di dimensione n , cioè il calcolo dell'unica matrice $X = A^{-1}$ tale che $AX = XA = I$, ricordiamo che X esiste se il *determinante* di A è non nullo, cioè se i vettori colonna di A sono linearmente indipendenti. Il calcolo dell'inversa può essere realizzato attraverso il comando `inv(A)`, mentre per il calcolo del determinante si può usare il comando `det(A)`. A questo proposito si ricordi che il determinante di una matrice quadrata è un numero definito ricorsivamente come segue (*regola di Laplace*)

$$\det(A) = \begin{cases} a_{11} & \text{se } n = 1, \\ \sum_{j=1}^n \Delta_{ij} a_{ij}, & \text{per } n > 1, \forall i = 1, \dots, n, \end{cases} \quad (1.8)$$

dove $\Delta_{ij} = (-1)^{i+j} \det(A_{ij})$ e A_{ij} è la matrice che si trova dalla matrice A per soppressione della i -esima riga e della j -esima colonna. (Il risultato non dipende dalla riga i scelta.)

Se $A \in \mathbb{R}^{1 \times 1}$ si porrà pertanto $\det(A) = a_{11}$, se $A \in \mathbb{R}^{2 \times 2}$ si ha

$$\det(A) = a_{11}a_{22} - a_{12}a_{21},$$

mentre se $A \in \mathbb{R}^{3 \times 3}$ otteniamo

$$\begin{aligned} \det(A) = & a_{11}a_{22}a_{33} + a_{31}a_{12}a_{23} + a_{21}a_{13}a_{32} \\ & - a_{11}a_{23}a_{32} - a_{21}a_{12}a_{33} - a_{31}a_{13}a_{22}. \end{aligned}$$

Infine, se $A = BC$, allora $\det(A) = \det(B)\det(C)$.

Vediamo un esempio di inversione di una matrice 2×2 e di calcolo del suo determinante

```
>> A=[1 2; 3 4];
>> inv(A)
ans =
-2.0000  1.0000
 1.5000 -0.5000
>> det(A)
ans =
-2
```

Se la matrice è singolare MATLAB segnala il problema e restituisce un messaggio diagnostico, seguito da una matrice con elementi uguali a **Inf**, come si vede nel seguente esempio

```
>> A=[1 2; 0 0];
>> inv(A)
Warning: Matrix is singular to working precision.
ans =
  Inf  Inf
  Inf  Inf
```

Facciamo notare fin d'ora che, per alcune classi di matrici quadrate, il calcolo dell'inversa e del determinante è particolarmente semplice. Iniziamo dalle *matrici diagonali* per le quali cioè gli a_{kk} , con $k = 1, \dots, n$, sono gli unici elementi che possono essere non nulli. Tali elementi formano la cosiddetta diagonale principale della matrice e si ha

$$\det(A) = a_{11}a_{22} \cdots a_{nn}.$$

Le matrici diagonali sono non singolari se $a_{kk} \neq 0$ per ogni k . In tal caso, l'inversa è ancora una matrice diagonale di elementi a_{kk}^{-1} .

La costruzione di una matrice diagonale di dimensione **n** in MATLAB è semplice, basta digitare il comando **diag(v)**, essendo **v** un vettore di dimensione **n** contenente i soli elementi diagonali. Scrivendo invece **diag(v,m)** si genera una matrice quadrata di dimensione **n+abs(m)** che presenta l'**m**-esima sopra-diagonale (o sotto-diagonale, se **m** è negativo) con elementi uguali a quelli contenuti nel vettore **v**.

Ad esempio, se **v** = [1 2 3] si avrà

```
>> A=diag(v,-1)
A =
  0   0   0   0
  1   0   0   0
  0   2   0   0
  0   0   3   0
```

Altre matrici per le quali il calcolo del determinante è elementare, sono quelle *triangolari superiori* o *triangolari inferiori*: una matrice quadrata di dimensione **n** è triangolare inferiore (rispettivamente, superiore)

se ha nulli tutti gli elementi che stanno al di sopra (rispettivamente, al di sotto) della diagonale principale. Il suo determinante è semplicemente il prodotto degli elementi diagonali.

In MATLAB, tramite i comandi `tril(A)` e `triu(A)`, è possibile estrarre dalla matrice A di dimensione n la sua parte triangolare inferiore e superiore, rispettivamente. Anch'essi, nelle forme `tril(A,m)` o `triu(A,m)`, con m che varia tra $-n$ e n , consentono di estrarre le parti triangolari aumentate o diminuite da sovra (o sotto) diagonali.

Ad esempio, considerata la matrice $A = [3 \ 1 \ 2; -1 \ 3 \ 4; -2 \ -1 \ 3]$, con il comando `L1=tril(A)` troviamo la matrice triangolare inferiore

`L1 =`

$$\begin{matrix} 3 & 0 & 0 \\ -1 & 3 & 0 \\ -2 & -1 & 3 \end{matrix}$$

Se invece, scriviamo `L2=tril(A,1)`, otteniamo la seguente matrice

`L2 =`

$$\begin{matrix} 3 & 1 & 0 \\ -1 & 3 & 4 \\ -2 & -1 & 3 \end{matrix}$$

Un'operazione propria delle matrici è la *trasposizione*: data una matrice $A \in \mathbb{R}^{n \times m}$ indichiamo con $A^T \in \mathbb{R}^{m \times n}$ la matrice trasposta, ottenuta scambiando tra loro le righe con le colonne di A . Se $A = A^T$, allora A è detta *simmetrica*. In MATLAB, se A è una matrice reale, `A'` denota la sua trasposta. Se invece A è una matrice complessa `A'` è la sua trasposta coniugata.

1.3.1 I vettori

I vettori verranno indicati con lettere in grassetto; così \mathbf{v} denota sempre un vettore colonna, la cui componente i -esima verrà indicata con v_i . Se un vettore ha come componenti n numeri reali si scriverà semplicemente $\mathbf{v} \in \mathbb{R}^n$.

MATLAB tratta i vettori come casi particolari di matrici. Per introdurre un vettore colonna basta riportare fra parentesi quadre i valori delle componenti del vettore stesso separati da un punto e virgola, mentre per un vettore riga è sufficiente riportare i valori separati da spazi bianchi o virgolette. Così ad esempio, le istruzioni `v = [1;2;3]` e `w = [1 2 3]` inizializzano rispettivamente un vettore colonna ed un vettore riga di dimensione 3. Il comando `zeros(n,1)` (rispettivamente, `zeros(1,n)`) produce un vettore colonna (rispettivamente, riga) di dimensione n con elementi tutti nulli: esso verrà denotato nel testo con **0**. Analogamente, il comando `ones(n,1)` genera un vettore colonna con tutte le componenti pari a 1, indicato perciò con **1**.

Tra i vettori saranno particolarmente importanti quelli tra loro *linearmente indipendenti*: ricordiamo che un sistema di vettori $\{\mathbf{y}_1, \dots, \mathbf{y}_m\}$ si dice linearmente indipendente se la relazione

$$\alpha_1 \mathbf{y}_1 + \dots + \alpha_m \mathbf{y}_m = \mathbf{0}$$

è soddisfatta solo se tutti i coefficienti $\alpha_1, \dots, \alpha_m$ sono nulli. Un insieme di n vettori $\mathcal{B} = \{\mathbf{y}_1, \dots, \mathbf{y}_n\}$ linearmente indipendenti di \mathbb{R}^n (o \mathbb{C}^n) forma una *base* per \mathbb{R}^n (rispettivamente, \mathbb{C}^n), gode cioè della proprietà che un qualunque vettore \mathbf{w} di \mathbb{R}^n può essere scritto in modo unico come combinazione lineare dei vettori della base,

$$\mathbf{w} = \sum_{k=1}^n w_k \mathbf{y}_k.$$

I numeri w_k sono dette le *componenti* di \mathbf{w} rispetto alla base \mathcal{B} . Ad esempio, la base canonica per \mathbb{R}^n è quella costituita dai vettori $\{\mathbf{e}_1, \dots, \mathbf{e}_n\}$, dove \mathbf{e}_i ha la i -esima componente pari a 1 e le restanti nulle. Questa non è l'unica base per \mathbb{R}^n , ma è quella che verrà generalmente utilizzata.

Per quanto riguarda le operazioni fra vettori della stessa dimensione ricordiamo in particolare il *prodotto scalare* ed il *prodotto vettore*. Il primo è definito come

$$\mathbf{v}, \mathbf{w} \in \mathbb{R}^n, (\mathbf{v}, \mathbf{w}) = \mathbf{w}^T \mathbf{v} = \sum_{k=1}^n v_k w_k,$$

essendo $\{v_k\}$ e $\{w_k\}$ le componenti di \mathbf{v} e \mathbf{w} , rispettivamente. In MATLAB tale operazione tra vettori colonna si esegue scrivendo $\mathbf{w}' * \mathbf{v}$, dove l'apice esegue l'operazione di trasposizione del vettore \mathbf{w} (o utilizzando l'apposito comando `dot(v,w)`). Il modulo di un vettore \mathbf{v} è allora dato da

$$\|\mathbf{v}\| = \sqrt{(\mathbf{v}, \mathbf{v})} = \sqrt{\sum_{k=1}^n v_k^2}$$

e viene calcolato con il comando `norm(v)`. Il prodotto vettore fra due vettori $\mathbf{v}, \mathbf{w} \in \mathbb{R}^n$ è invece dato dal vettore $\mathbf{u} \in \mathbb{R}^n$ (denotato con $\mathbf{u} = \mathbf{v} \times \mathbf{w}$ o $\mathbf{u} = \mathbf{v} \wedge \mathbf{w}$) ortogonale sia a \mathbf{v} che a \mathbf{w} e di modulo $|\mathbf{u}| = |\mathbf{v}| |\mathbf{w}| \sin(\alpha)$, dove α è l'angolo compreso fra \mathbf{v} e \mathbf{w} . In MATLAB si calcola con il comando `cross(v,w)`. La visualizzazione di vettori in MATLAB può essere effettuata con i comandi `quiver` per i vettori di \mathbb{R}^2 e `quiver3` per quelli di \mathbb{R}^3 .

Talvolta nei programmi MATLAB che proporremo compariranno delle operazioni fra vettori precedute da un punto, come ad esempio $\mathbf{x}.*\mathbf{y}$ o $\mathbf{x}.^2$. Questo è solo un modo per segnalare all'elaboratore che

`norm``cross`
`quiver`
`quiver3``.*``.^`

l'operazione non va eseguita nel senso usuale, ma componente per componente. Così $\mathbf{x}.*\mathbf{y}$ non è il prodotto scalare fra i vettori \mathbf{x} e \mathbf{y} , ma restituisce ancora un vettore con la componente i -esima pari a $x_i y_i$. Ad esempio, se definiamo i vettori

```
>> v = [1; 2; 3]; w = [4; 5; 6];
```

il prodotto scalare ed il prodotto componente per componente sono dati rispettivamente da

```
>> w'*v
ans =
 32
>> w.*v
ans =
   4
  10
  18
```

Si noti che il prodotto $w*v$ non è neppure definito, non avendo i vettori le dimensioni corrette.

Ricordiamo infine che un vettore $\mathbf{v} \in \mathbb{R}^n$, con $\mathbf{v} \neq \mathbf{0}$, è un *autovettore* di una matrice $A \in \mathbb{R}^{n \times n}$ associato al numero complesso λ se

$$A\mathbf{v} = \lambda\mathbf{v}.$$

Il numero λ viene detto *autovalore* di A . Il calcolo degli autovalori di una matrice è generalmente assai complicato; fanno eccezione le matrici diagonali e quelle triangolari per le quali gli autovalori sono gli elementi diagonali stessi.



Si vedano gli Esercizi 1.3-1.6.

1.4 Le funzioni

Le funzioni reali a variabile reale saranno le protagoniste di alcuni capitoli di questo libro. In particolare, data una funzione f definita su un intervallo (a, b) , vorremo calcolarne gli zeri, il suo integrale e la sua derivata, nonché conoscerne in maniera approssimata l'andamento.

fplot

Il comando **fplot(fun,lims)** visualizza il grafico di una funzione precisata nella stringa **fun** sull'intervallo **(lims(1),lims(2))**. Ad esempio, se si vuole rappresentare $f(x) = 1/(1 + x^2)$ su $(-5, 5)$, basta scrivere

```
>> fun ='1/(1+x^2)'; lims=[-5,5]; fplot(fun,lims);
```

In alternativa, si può scrivere direttamente

```
>> fplot('1/(1+x^2)',[-5 5]);
```

Il grafico ottenuto è una rappresentazione approssimata, a meno del 0.2%, del grafico di f ed è ottenuto campionando la funzione su un opportuno insieme non equispaziato di ascisse. Per aumentare l'accuracy della rappresentazione è sufficiente richiamare `fplot` nel modo seguente

```
>> fplot(fun,lims,tol,n,LineSpec)
```

essendo `tol` la tolleranza relativa richiesta. Il parametro `n` (≥ 1) assicura che il grafico della funzione sia disegnato utilizzando almeno `n+1` punti; `LineSpec` è una stringa che specifica invece il tratto grafico (od il colore) della linea utilizzata nel tracciamento del grafico (ad esempio, `LineSpec='--'` per una linea tratteggiata, `LineSpec='r-.'` per una linea tratto-punto rossa). Per usare i valori *di default* (ovvero preassegnati) per una qualsiasi di tali variabili si può passare una matrice vuota ([]).

Il valore di `fun` in un punto `x` (o su un insieme di punti, memorizzati nuovamente nel vettore `x`), si trova utilizzando il comando `y=eval(fun)`, dopo aver inizializzato `x`. In `y` vengono raccolte le corrispondenti ordinate. Per usare questo comando è necessario che la stringa `fun` sia un'espressione della variabile `x`. Come vedremo nell'Osservazione 1.2 in alternativa a `eval` si può usare il comando più generale `feval`.

Infine, per introdurre una griglia di riferimento come quella che compare nella Figura 1.1 basta dare, dopo il comando `fplot`, il comando `grid on`.

`eval`
`feval`
`grid`

1.4.1 Gli zeri

Ricordiamo che se $f(\alpha) = 0$, α si dice *zero* di f , o equivalentemente, *radice* dell'equazione $f(x) = 0$. Uno zero viene inoltre detto *semplice* se $f'(\alpha) \neq 0$, *multiplo* in caso contrario. Dal grafico di una funzione è possibile ricavare, seppur in maniera approssimata, i suoi zeri.

Il calcolo diretto degli zeri di una data funzione non è sempre possibile. Ad esempio, nel caso in cui la funzione sia un polinomio a coefficienti reali di grado n , cioè sia della forma

$$p_n(x) = a_0 + a_1x + a_2x^2 + \dots + a_nx^n = \sum_{k=0}^n a_kx^k, \quad a_k \in \mathbb{R}, \quad a_n \neq 0,$$

è possibile calcolarne l'unico zero $\alpha = -a_0/a_1$, quando $n = 1$ (ovvero il grafico di p_1 è una retta), o i due zeri, α_+ e α_- , quando $n = 2$ (il grafico di p_2 è una parabola)

$$\alpha_{\pm} = \frac{-a_1 \pm \sqrt{a_1^2 - 4a_0a_2}}{2a_2}.$$

È anche noto che, se $n \geq 5$, non esistono formule generali che con un numero finito di operazioni consentano di calcolare le radici di un polinomio p_n qualunque. Indicheremo nel seguito con \mathbb{P}_n lo spazio dei polinomi di grado minore o uguale a n .

Anche il numero di zeri di una funzione non è determinabile in modo elementare a priori; fanno eccezione i polinomi per i quali il numero di radici (reali o complesse) è uguale al grado del polinomio stesso. Si sa inoltre che se un polinomio a coefficienti reali di grado $n \geq 2$ ammette una radice complessa $\alpha = x + iy$ con $y \neq 0$, allora esso deve presentare come radice anche il complesso coniugato $\bar{\alpha} = x - iy$ di α .

Il calcolo di uno zero (non di tutti) di una funzione `fun` vicino ad un certo valore `x0`, reale o complesso, può essere fatto in MATLAB tramite `fzero`. In uscita, oltre al valore approssimato dello zero, viene fornito l'intervallo entro il quale il programma ha cercato lo zero. In alternativa, richiamando il comando `fzero(fun, [x0 x1])` viene cercato uno zero di `fun` nell'intervallo di estremi `x0`, `x1`, purché f cambi di segno in `x0` e `x1`.

Ad esempio, consideriamo la funzione $f(x) = x^2 - 1 + e^x$; da uno studio grafico si deduce che essa presenta due zeri nell'intervallo $(-1, 1)$ per calcolare i quali basta eseguire le seguenti istruzioni

```
>> fun='x^2 - 1 + exp(x)';
>> fzero(fun,1)
Zero found in the interval: [-0.28, 1.9051].
ans =
    6.0953e-18
>> fzero(fun,-1)
Zero found in the interval: [-1.2263, -0.68].
ans =
    -0.7146
```

Alternativamente, avendo osservato per via grafica che uno zero si trova in $[-1, -0.2]$ e l'altro in $[-0.2, 1]$ avremmo potuto anche scrivere

```
>> fzero(fun,[-0.2 1])
Zero found in the interval: [-0.2, 1].
ans =
    -4.2658e-17
>> fzero(fun,[-1 -0.2])
Zero found in the interval: [-1, -0.2].
ans =
    -0.7146
```

Come si vede il risultato ottenuto per la prima radice non è uguale a quello calcolato in precedenza (pur essendo entrambi di fatto assimilabili allo 0): ciò è dovuto alla diversa inizializzazione nei due casi dell'algorit-

mo implementato in `fzero`. Vedremo nel capitolo 2 alcuni metodi per il calcolo degli zeri di una funzione.

1.4.2 I polinomi

Come abbiamo già avuto modo di dire i polinomi sono funzioni abbastanza particolari e per essi sono stati approntati comandi MATLAB specifici, raccolti nel toolbox¹ `polyfun`.

Accenniamo ai principali. Il comando `polyval` serve per valutare un polinomio in uno o più punti e riceve in ingresso due vettori, `p` e `x`. In `p` devono essere memorizzati i coefficienti del polinomio ordinati da a_n fino ad a_0 , mentre in `x` si devono specificare le ascisse nelle quali si vuole che il polinomio sia valutato. Il risultato potrà essere salvato in un vettore `y` scrivendo

```
>> y = polyval(p,x)
```

Ad esempio, per il polinomio $p(x) = x^7 + 3x^2 - 1$, i valori assunti nei nodi equispaziati $x_k = -1 + k/4$ per $k = 0, \dots, 8$, si trovano scrivendo

```
>> p = [1 0 0 0 0 3 0 -1]; x = [-1:0.25:1];
>> y = polyval(p,x)
y =
    Columns 1 through 7
    1.0000    0.5540   -0.2578   -0.8126   -1.0000   -0.8124   -0.2422
    Columns 8 through 9
    0.8210    3.0000
```

Evidentemente per valutare un polinomio si potrebbe usare anche il comando `feval`; esso è però generalmente scomodo perché obbligherebbe a riportare nella stringa che definisce la funzione da disegnare l'espressione analitica del polinomio e non solo i suoi coefficienti.

Il programma `roots` serve invece per calcolare in maniera approssimata gli zeri di un polinomio e richiede in ingresso il solo vettore `p`. Ad esempio, per il polinomio $p(x) = x^3 - 6x^2 + 11x - 6$ calcoliamo gli zeri scrivendo

```
>> p = [1 -6 11 -6]; format long;
>> roots(p)
ans =
    3.000000000000000
    2.000000000000000
    1.000000000000000
```

¹ Un toolbox è una raccolta di programmi MATLAB relativi ad uno specifico argomento

In tal caso si ottengono gli zeri esatti.

Non sempre però il risultato è così accurato: ad esempio, per il polinomio $p(x) = (x + 1)^7$, il cui unico zero con molteplicità 7 è $\alpha = -1$, si trovano i seguenti zeri (alcuni dei quali addirittura complessi)

```
>> p = [1 7 21 35 35 21 7 1];
>> roots(p)
ans =
-1.0101
-1.0063 + 0.0079i
-1.0063 - 0.0079i
-0.9977 + 0.0099i
-0.9977 - 0.0099i
-0.9909 + 0.0044i
-0.9909 - 0.0044i
```

Una spiegazione di questo comportamento risiede nel fatto che i metodi numerici solitamente usati per calcolare le radici di un polinomio sono particolarmente sensibili agli errori di arrotondamento in presenza di radici multiple (si veda il paragrafo 2.5.2).

conv Con il comando `p=conv(p1,p2)` si calcolano i coefficienti del polinomio ottenuto come prodotto dei polinomi i cui coefficienti sono precisati in `p1` e `p2`. Invece, il comando `[q,r]=deconv(p1,p2)` calcola i coefficienti del quoziente e del resto della divisione fra `p1` e `p2`, cioè `q` e `r` tali che `p1 = conv(p2,q) + r`.

deconv Ad esempio, consideriamo i polinomi $p_1(x) = x^4 - 1$ e $p_2(x) = x^3 - 1$ e calcoliamone il prodotto e la divisione

```
>> p1 = [1 0 0 0 -1];
>> p2 = [1 0 0 -1];
>> p=conv(p1,p2)
p =
    1     0     0    -1   -1     0     0     1
>> [q,r]=deconv(p1,p2)
q =
    1     0
r =
    0     0     0     1   -1
```

Troviamo pertanto i polinomi $p(x) = p_1(x)p_2(x) = x^7 - x^4 - x^3 + 1$, $q(x) = x$ e $r(x) = x - 1$ tali che $p_1(x) = q(x)p_2(x) + r(x)$.

polyint Infine i comandi `polyint(p)` e `polyder(p)` forniscono rispettivamente i coefficienti della primitiva (che si annulla in $x = 0$) e quelli della derivata del polinomio i cui coefficienti sono dati dalle componenti del vettore `p`.



Riassumiamo i comandi precedenti nella Tabella 1.1: in essa, x è un vettore di ascisse, mentre p , p_1 , p_2 sono i vettori contenenti i coefficienti dei polinomi p , p_1 e p_2 , rispettivamente.

comando	risultato
<code>y=polyval(p,x)</code>	$y = \text{valori di } p(x)$
<code>z=roots(p)</code>	$z = \text{radici di } p \text{ tali che } p(z) = 0$
<code>p=conv(p1,p2)</code>	$p = \text{coefficienti del polinomio } p_1 p_2$
<code>[q,r]=deconv(p1,p2)</code>	$q = \text{coefficienti di } q, r = \text{coefficienti di } r \text{ tali che } p_1 = qp_2 + r$
<code>y=polyder(p)</code>	$y = \text{coefficienti di } p'$
<code>y=polyint(p)</code>	$y = \text{coefficienti di } \int_0^x p(t) dt$

Tabella 1.1. Principali comandi MATLAB relativi ai polinomi

Un ulteriore comando, `polyfit`, consente di calcolare gli $n+1$ coefficienti di un polinomio p di grado n una volta noti i valori di p in $n+1$ punti distinti (si veda il paragrafo 3.1.1).

`polyfit`

1.4.3 L'integrale e la derivata

Per quanto riguarda l'integrazione, riteniamo utile ricordare i due seguenti risultati:

- il *teorema fondamentale del calcolo integrale* per il quale se f è una funzione continua nell'intervallo $[a, b]$, allora la funzione integrale

$$F(x) = \int_a^x f(t) dt \quad \forall x \in [a, b],$$

detta *primitiva* di f , è derivabile e si ha, $\forall x \in [a, b]$,

$$F'(x) = f(x);$$

- il *teorema della media integrale* per il quale se f è una funzione continua nell'intervallo $[a, b]$ e se $x_1, x_2 \in [a, b]$ con $x_2 > x_1$, allora $\exists \xi \in (x_1, x_2)$ tale che

$$f(\xi) = \frac{1}{x_2 - x_1} \int_{x_1}^{x_2} f(t) dt.$$

Il calcolo analitico della primitiva non è sempre possibile e comunque potrebbe non essere conveniente da un punto di vista computazionale. Ad esempio, sapere che l'integrale di $1/x$ è $\ln|x|$ non è rilevante se non sappiamo come calcolare efficientemente il logaritmo. Nel Capitolo 4 vedremo metodi di approssimazione in grado di calcolare l'integrale di una funzione continua con l'accuratezza desiderata, a prescindere dalla conoscenza della sua primitiva.

Ricordiamo che una funzione f definita su un intervallo $[a, b]$ è derivabile in un punto $\bar{x} \in (a, b)$ se esiste finito il limite

$$f'(\bar{x}) = \lim_{h \rightarrow 0} \frac{f(\bar{x} + h) - f(\bar{x})}{h}. \quad (1.9)$$

Il valore $f'(\bar{x})$ fornisce la pendenza della retta tangente a f in \bar{x} .

Diremo che una funzione derivabile con derivata continua su tutto un intervallo $[a, b]$ appartiene allo spazio $C^1([a, b])$. In generale, una funzione derivabile con derivate continue fino all'ordine p (intero positivo) si dice appartenente a $C^p([a, b])$. In particolare, una funzione soltanto continua appartiene a $C^0([a, b])$.

Un risultato dell'Analisi che utilizzeremo spesso è il *teorema del valor medio* secondo il quale, se $f \in C^1([a, b])$, allora esiste un punto $\xi \in (a, b)$ tale che

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

È infine utile ricordare che, data una funzione con derivate continue fino all'ordine $n+1$ in un intorno di un punto x_0 , essa può essere approssimata in un intorno di x_0 dal cosiddetto *polinomio di Taylor di grado n*, costruito rispetto al punto x_0

$$\begin{aligned} T_n(x) &= f(x_0) + (x - x_0)f'(x_0) + \dots + \frac{1}{n!}(x - x_0)^n f^{(n)}(x_0) \\ &= \sum_{k=0}^n \frac{(x - x_0)^k}{k!} f^{(k)}(x_0). \end{aligned}$$

Si tenga infine presente che in MATLAB il toolbox `symbolic` consente, attraverso i comandi `diff`, `int` e `taylor`, di calcolare analiticamente la derivata, l'integrale indefinito (ovvero la primitiva) ed il polinomio di Taylor di semplici funzioni. In particolare, definita nella stringa `f` l'espressione della funzione sulla quale si intende operare, `diff(f,n)` ne calcola la derivata di ordine `n`, `int(f)` l'integrale e `taylor(f,x,n+1)` il polinomio di Taylor di grado `n` in un intorno di $x_0 = 0$. La variabile `x` che compare deve essere dichiarata *simbolica* con il comando `syms x`. In tal modo, essa potrà essere manipolata algebricamente senza dover essere necessariamente valutata.

`diff``int``taylor``syms`

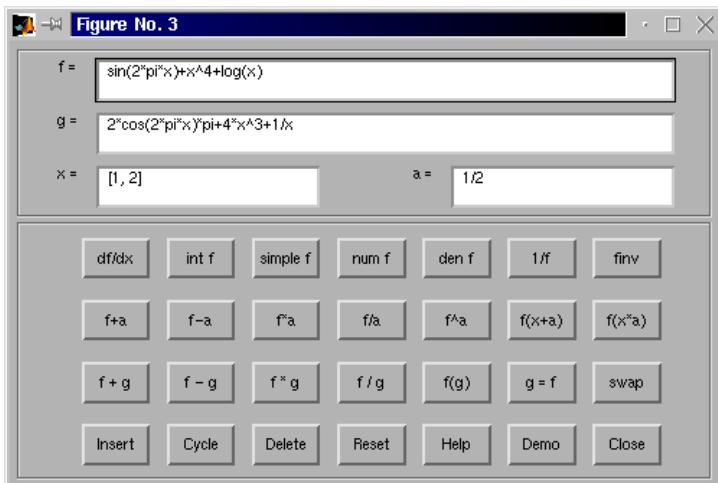


Figura 1.5. Interfaccia grafica del comando `funtool`

Ad esempio, per calcolare la derivata, l'integrale indefinito ed il polinomio di Taylor del quinto'ordine della funzione $f(x) = (x^2+2x+2)/(x^2-1)$, basta digitare i comandi

```
>> f = '(x^2+2*x+2)/(x^2-1)';
>> syms x
>> diff(f)
(2*x+2)/(x^2-1)-2*(x^2+2*x+2)/(x^2-1)^2*x
>> int(f)
x+5/2*log(x-1)-1/2*log(1+x)
>> taylor(f,x,6)
-2-2*x-3*x^2-2*x^3-3*x^4-2*x^5
```

Con il comando `simple` è possibile semplificare le espressioni generate da `diff`, `int` e `taylor` in modo da renderle più semplici possibili. Il comando `funtool` consente infine, attraverso l'interfaccia grafica riportata in Figura 1.5, di manipolare simbolicamente delle funzioni e di studiarne le principali caratteristiche.

Si vedano gli Esercizi 1.7-1.8.

`simple`
`funtool`



1.5 Errare non è solo umano

In effetti, parafrasando il motto latino, si potrebbe dire che nel Calcolo Scientifico errare è addirittura inevitabile. Come abbiamo visto infatti il semplice uso di un calcolatore per rappresentare i numeri reali introduce

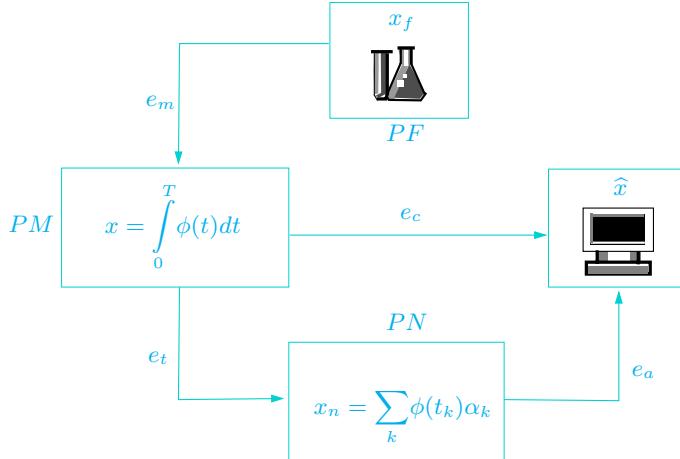


Figura 1.6. I vari tipi di errore nel processo computazionale

degli errori. L’importante perciò non è annullare gli errori, ma imparare a controllarne la grandezza.

Molto in generale possiamo distinguere diversi livelli di errore che accompagnano il processo di approssimazione e risoluzione di un problema fisico (si veda la Figura 1.6).

Al livello più alto, stanno gli errori e_m che si commettono rappresentando la realtà fisica (PF sta per problema fisico e x_f ne è la soluzione) attraverso un qualche modello matematico (PM, la cui soluzione è x). Essi limiteranno l’applicabilità del modello matematico a determinate situazioni e sfuggono al controllo del Calcolo Scientifico.

Il modello matematico (sia esso esprimibile tramite un integrale come nel caso dell’esempio in Figura 1.6, un’equazione algebrica o differenziale, un sistema lineare o non lineare) non è in generale risolubile analiticamente. La sua risoluzione al calcolatore comporterà certamente almeno l’introduzione e la propagazione degli errori di arrotondamento negli algoritmi utilizzati. Chiamiamo questi errori e_a . D’altra parte spesso ad essi è necessario aggiungere altri errori legati alla necessità di eliminare dal modello matematico ogni operazione che richieda passaggi al limite: tali operazioni non possono essere infatti realizzate su un calcolatore se non in maniera approssimata (si pensi ad esempio al calcolo della somma di una serie che dovrà necessariamente arrestarsi alla troncata di un certo ordine). Si dovrà pertanto introdurre un problema numerico, PN , la cui soluzione x_n differisce da x per un errore e_t che viene detto *errore di troncamento*. Tali errori sono assenti soltanto in quei modelli matematici che sono già di dimensione finita (ad esempio, nella risoluzione di un sistema lineare). Gli errori e_a e e_t costituiscono nel loro insieme l’*errore computazionale* e_c che è la quantità di nostro interesse.

Se, come detto, indichiamo con x la soluzione esatta del modello matematico e con \hat{x} la soluzione ottenuta al termine del processo numerico, l'errore computazionale assoluto sarà dunque

$$e_c^{ass} = |x - \hat{x}|,$$

mentre quello relativo sarà (se $x \neq 0$)

$$e_c^{rel} = |x - \hat{x}|/|x|,$$

dove $|\cdot|$ denota il modulo (o un'altra misura di grandezza a seconda del significato di x).

Generalmente il processo numerico è una approssimazione del modello matematico ottenuta in funzione di un parametro di discretizzazione, che indicheremo con h e supporremo positivo, con la speranza che per h tendente a 0 il processo numerico restituiscia la soluzione del modello matematico. Diremo in tal caso che il processo numerico è *convergente*. Se l'errore, assoluto o relativo, può essere limitato in funzione di h come

$$e_c \leq Ch^p, \quad (1.10)$$

dove C è indipendente da h e p è un numero (generalmente intero), diremo che il metodo è *convergente di ordine p* . Talvolta si potrà addirittura sostituire il simbolo \leq con il simbolo \simeq , nel caso in cui, oltre alla maggiorazione (1.10), valga anche una minorazione $C'h^p \leq e_c$, essendo C' un'altra costante ($\leq C$) indipendente da h e p .

Esempio 1.1 Supponiamo di approssimare la derivata di una funzione f in un punto \bar{x} con il rapporto incrementale che compare nella (1.9). Evidentemente se f è derivabile in \bar{x} , per h che tende a 0 l'errore commesso sostituendo a $f'(\bar{x})$ tale rapporto tende a 0. Come vedremo però nel paragrafo 4.1 esso può essere maggiorato da Ch solo se $f \in C^2$ in un intorno di \bar{x} .

Negli studi di convergenza spesso ci capiterà di dover leggere dei grafici che riportano l'errore in funzione di h in scala logaritmica, cioè che presentano sull'asse delle ascisse $\log(h)$ e sull'asse delle ordinate $\log(e_c)$. Il vantaggio di questa rappresentazione è presto detto: se $e_c \simeq Ch^p$ allora $\log e_c \simeq \log C + p \log h$. Di conseguenza, p in scala logaritmica rappresenta la pendenza della retta $\log e_c$ e quindi, se abbiamo due metodi da confrontare, quello che produrrà la retta con maggiore pendenza sarà quello di ordine più elevato. In MATLAB per ottenere grafici in scala logaritmica è sufficiente invocare il comando `loglog(x,y)`, essendo x e y i vettori contenenti le ascisse e le ordinate dei dati che si vogliono rappresentare.

Ad esempio, in Figura 1.7 vengono riportate le rette relative all'andamento degli errori in due diversi metodi. Quello in linea continua risulta del prim'ordine, mentre quello in linea tratteggiata è del second'ordine.

`loglog`

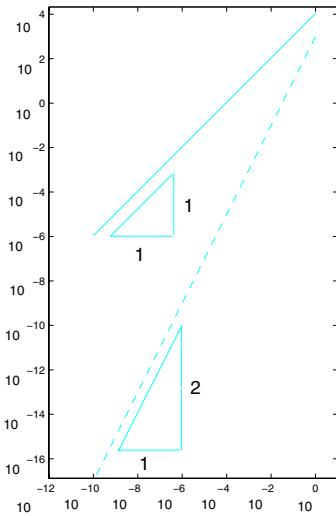


Figura 1.7. Grafici in scala logaritmica

Un modo alternativo a quello grafico per stabilire l'ordine di un metodo è il seguente. Nel caso siano noti gli errori e_i per certi valori h_i del parametro di discretizzazione, con $i = 1, \dots, N$, si ipotizza che $e_i \simeq Ch_i^p$ con C indipendente da i . A questo punto si può stimare p attraverso i valori

$$p_i = \log(e_i/e_{i-1}) / \log(h_i/h_{i-1}), \quad i = 2, \dots, N. \quad (1.11)$$

A ben guardare l'errore non è una quantità calcolabile in quanto dipende dall'incognita stessa del problema. È dunque necessario introdurre delle quantità calcolabili che possono essere utilizzate per stimare l'errore stesso, i cosiddetti *stimatori dell'errore*. Ne vedremo degli esempi nei paragrafi 2.2, 2.3 e 4.4.

1.5.1 Parliamo di costi

In generale la risoluzione di un problema su un calcolatore viene effettuata attraverso un algoritmo, ovvero una direttiva, sotto forma di un testo finito, che precisi in maniera univoca tutti i passi necessari per risolvere il problema. Siamo interessati ad algoritmi che richiedano un numero finito di passi.

Per *costo computazionale* di un algoritmo si intende di solito il numero di operazioni aritmetiche che esso richiede per la sua esecuzione. A tale numero è infatti correlato uno degli indicatori sulla velocità di un elaboratore, cioè il numero di operazioni *floating-point* che vengono eseguite in un secondo. Esso si misura in *flops* e nei suoi multipli

(mega-flops pari a 10^6 flops, giga-flops pari a 10^9 flops, tera-flops pari a 10^{12} flops). I calcolatori attualmente più potenti possono effettuare decine di tera-flops.

In genere, non serve conoscere esattamente il numero delle operazioni aritmetiche, ma basta quantificarne la grandezza in funzione di un parametro d legato alla dimensione del problema che si sta risolvendo. Così diremo che un algoritmo ha una complessità *costante* se richiede un numero di operazioni indipendente da d cioè se richiede $\mathcal{O}(1)$ operazioni, *lineare* se richiede $\mathcal{O}(d)$ operazioni e, più in generale, *polinomiale* se richiede $\mathcal{O}(d^m)$ operazioni con m intero positivo. Alcuni algoritmi presentano complessità più elevate, di tipo *esponenziale* ($\mathcal{O}(c^d)$ operazioni) o *fattoriale* ($\mathcal{O}(d!)$ operazioni).

Ricordiamo che la simbologia $\mathcal{O}(d^m)$ (che si legge “ O grande di d^m ”) sta per “si comporta, per d grandi, come una costante per d^m ”.

Esempio 1.2 (il prodotto matrice-vettore) Si consideri una matrice quadrata A di dimensione n : vogliamo quantificare il costo computazionale dell’usuale algoritmo per eseguire il prodotto $A\mathbf{v}$ dove $\mathbf{v} \in \mathbb{R}^n$. Osserviamo che il calcolo della componente j -esima del vettore prodotto, data da

$$a_{j1}v_1 + a_{j2}v_2 + \dots + a_{jn}v_n,$$

richiede n prodotti e $n - 1$ somme. In tutto dobbiamo calcolare n componenti e dovremo quindi eseguire $n(2n - 1)$ operazioni. Dunque questo algoritmo richiede $\mathcal{O}(n^2)$ operazioni ed ha pertanto complessità quadratica rispetto al parametro n . Con lo stesso procedimento sono necessarie $\mathcal{O}(n^3)$ operazioni per eseguire il prodotto di 2 matrici di ordine n . Esiste tuttavia un algoritmo, detto algoritmo di Strassen, che ne richiede “solo” $\mathcal{O}(n^{\log_2 7})$, ed un altro, dovuto a Winograd e Coppersmith, che richiede $\mathcal{O}(n^{2.376})$ operazioni.

Esempio 1.3 (il calcolo del determinante) Come abbiamo ricordato, il determinante di una matrice di dimensione n può essere calcolato tramite la formula ricorsiva (1.8).

Si può però verificare che l’algoritmo corrispondente ha una complessità fattoriale rispetto a n , ovvero richiede $\mathcal{O}(n!)$ operazioni. Teniamo presente che algoritmi con una tale complessità non possono essere eseguiti neppure sui calcolatori più avanzati oggi disponibili se non per n piccoli. Ad esempio, se $n = 24$, su un elaboratore in grado di eseguire 1 peta-flops (cioè 10^{15} operazioni al secondo) servirebbero circa 20 anni per terminare il calcolo. In effetti il solo aumento della potenza di calcolo non consente la risoluzione di un qualunque problema: è necessario studiare ed approntare metodi numerici che presentino un costo computazionale accessibile. Per esempio esiste un algoritmo di tipo ricorsivo che consente di ridurre il calcolo del determinante a quello del prodotto di matrici, dando così luogo ad una complessità di $\mathcal{O}(n^{\log_2 7})$ operazioni se si ricorre all’algoritmo di Strassen (si veda [BB96]).

Il numero di operazioni richiesto da un algoritmo è dunque un parametro importante nell’analisi teorica dell’algoritmo stesso. Quando però un algoritmo viene codificato in un programma possono intervenire anche altri fattori che ne condizionano l’efficacia (quali ad esempio l’accesso

alle memorie): una misura delle prestazioni di un programma è allora il tempo di esecuzione o meglio, il *tempo di CPU* (CPU sta per *central processing unit*). Si tratta del tempo impiegato dall'unità centrale del calcolatore per eseguire un determinato programma e non tiene conto dei tempi di attesa per acquisire i dati necessari ad iniziare l'elaborazione (la cosiddetta fase di *input*) o per salvare i risultati ottenuti (la fase di *output*). È quindi diverso dal tempo che intercorre tra il momento in cui un programma è stato mandato in esecuzione ed il suo completamento. Quest'ultimo è noto (in inglese) come *elapsed time*. In MATLAB il tempo di CPU viene misurato in secondi attraverso il comando `cputime`, mentre l'*elapsed time* si misura (sempre in secondi) con il comando `etime`.

Esempio 1.4 Misuriamo il tempo di esecuzione del prodotto di una matrice quadrata e di un vettore. A tale scopo eseguiamo le seguenti istruzioni

```
>> n=4000; step=50; A=rand(n,n); v=rand(n);
>> T=[ ]; sizeA=[ ];
>> for k = 50:step:n
    AA = A(1:k,1:k); vv = v(1:k)';
    t = cputime; b = AA*vv; tt = cputime - t;
    T = [T, tt]; sizeA = [sizeA,k];
end
```

Con l'istruzione `a:step:b` che compare nel ciclo `for` si generano tutti i numeri della forma `a+step*k` con `k` intero che va da 0 fino al massimo valore `kmax` per il quale `a+step*kmax` è minore o uguale a `b` (nel caso in esame `a=50`, `b=4000` e `step=50`). Il comando `rand(n,m)` inizializza una matrice $n \times m$ i cui elementi sono numeri casuali. Infine, nelle componenti del vettore `T` vengono memorizzati i tempi di CPU necessari per eseguire ogni prodotto matrice-vettore. `cputime` restituisce il tempo complessivo impiegato da MATLAB per eseguire tutti i processi di una sessione. Il tempo necessario per eseguire un singolo processo è dunque la differenza tra il tempo di CPU attuale e quello calcolato prima del processo in esame, memorizzato nel caso in esame nella variabile `t`. Il grafico della Figura 1.8 (ottenuto con il comando `plot(sizeA,T,'o')`) mostra come il tempo di CPU tenda effettivamente a crescere proporzionalmente al quadrato della dimensione `n` della matrice.

1.6 Qualche parola in più su MATLAB



MATLAB è un ambiente integrato per il calcolo scientifico e per la visualizzazione, scritto in linguaggio C e distribuito dalla MathWorks (si veda il sito www.mathworks.com). Il nome sta per *MATrix LABoratory* in quanto fu originariamente sviluppato per consentire un accesso immediato a pacchetti di software appositamente sviluppati per il calcolo matriciale.

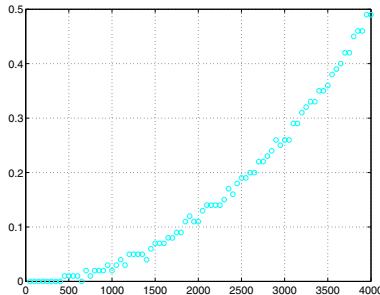


Figura 1.8. Tempo di CPU (in secondi) necessario per eseguire un prodotto matrice-vettore in funzione della dimensione n della matrice su un processore a 2.53 GHz

Una volta installato (l'eseguibile si trova generalmente nella sotto-directory *bin* della directory principale *matlab*), l'esecuzione di MATLAB consente di accedere ad un ambiente di lavoro caratterizzato dal *prompt* `>>`. Ad esempio, eseguendo MATLAB a noi compare

```
< M A T L A B >
Copyright 1984-2004 The MathWorks, Inc.
Version 7.0.0.19901 (R14)
May 06, 2004
```

To get started, select MATLAB Help or Demos from the Help menu.

`>>`

Tutto ciò che scriveremo dopo il *prompt* verrà, premuto il tasto *enter* (o *return*), interpretato:² MATLAB si domanderà se ciò che abbiamo scritto rientra fra le variabili definite e, se questo non accade, se è il nome di uno dei programmi o dei comandi presenti in MATLAB. Se anche questo controllo fallisce, MATLAB segnala un messaggio d'errore. In caso contrario il comando viene eseguito producendo eventualmente un *output*. In entrambi i casi il sistema ripropone al termine il *prompt* in attesa di un nuovo comando. Il programma si chiude scrivendo il comando `quit` (o `exit`) e battendo *enter*. D'ora in poi sottintenderemo che per eseguire una certa istruzione sia necessario battere *enter* e useremo i termini comando, programma o *function* in modo equivalente.

Quando il comando che abbiamo scritto è una delle strutture elementari definite in MATLAB (come i numeri o le stringhe di caratteri che si precisano tra apici) viene restituito in *output* nella variabile di default `ans` (che sta per *answer*, cioè risposta). Ad esempio, se digitiamo la stringa di caratteri '`casa`' abbiamo

`>>`

`quit`
`exit`

`ans`

² Di conseguenza, un programma MATLAB non deve essere compilato come in altri linguaggi come, ad esempio, Fortran o C, anche se, per aumentare la velocità di esecuzione dei codici, si può ricorrere ad un compilatore MATLAB con il comando `mcc`.

```
>> 'casa'
ans =
casa
```

Se ora digitiamo un'altra stringa o un numero, `ans` assumerà il nuovo valore. Per disabilitare questo *output* automatico è sufficiente scrivere un punto e virgola dopo la variabile o il comando di interesse. Così l'esecuzione di '`casa`'; si limita a riporre il *prompt*, assegnando comunque il valore `casa` alla variabile `ans`.

= Il comando `=` serve per assegnare ad una data variabile un valore. Ad esempio, volendo assegnare la stringa '`casa`' alla variabile `a` basterà scrivere

```
>> a='casa';
```

Come si vede non è necessario dichiarare il tipo della variabile `a`; sarà MATLAB che automaticamente e dinamicamente allocherà le variabili che di volta in volta utilizzeremo. Ad esempio, se decidessimo di voler utilizzare la `a` che abbiamo prima inizializzato per memorizzare il numero 5, non dovremmo far altro che scrivere `a=5`. Questa estrema semplicità d'uso ha però un prezzo. Supponiamo ad esempio di chiamare una variabile `quit` e di porla pari a 5. Abbiamo quindi creato una variabile che ha lo stesso nome del comando MATLAB `quit`; così facendo non possiamo più eseguire il comando `quit` in quanto MATLAB per interpretare un comando prima controlla se è una variabile e, solo nel caso non lo sia, se è uno dei comandi definiti. Bisogna quindi evitare di assegnare a variabili, o a programmi, nomi di variabili o programmi già definiti in MATLAB.

`clear` In ogni caso, con il comando `clear` seguito dal nome della variabile è possibile cancellare una variabile, ad esempio la nostra `quit`, dal sistema e riaccedere quindi, nel nostro caso, al comando `quit`.

`save` Utilizzando il comando `save` tutte le variabili della sessione (che sono memorizzate nel cosiddetto *base workspace*) vengono salvate nel file

`load` binario `matlab.mat`. Analogamente, il comando `load` ripristina nella sessione corrente tutte le variabili memorizzate nel file binario `matlab.mat`. Il nome del file nel quale si salvano (o si caricano) le variabili può essere precisato facendo seguire al comando `save` (rispettivamente, `load`) il nome prescelto per il file stesso. Se poi si volessero salvare solo alcune variabili, diciamo `v1`, `v2` e `v3`, in uno specifico file, di nome ad esempio `area.mat`, basterà dare il comando `save area v1 v2 v3`.

I comandi disponibili e le variabili predefinite sono individuabili attraverso il comando `help`: una volta invocato, esso presenta una lista di tutti i pacchetti (inclusi i cosiddetti *toolbox* che sono insiemi di programmi specialistici) di comandi disponibili durante l'esecuzione. Tra i moltissimi ricordiamo quelli che definiscono le funzioni elementari seno

`sin cos` (`sin(x)`), coseno (`cos(x)`), radice quadrata (`sqrt(x)`) ed esponenziale (`exp(x)`).

`sqrt exp`

Ci sono inoltre dei caratteri speciali che non possono comparire nel nome di una variabile o di un comando; ad esempio, gli operandi delle operazioni elementari di addizione, sottrazione, moltiplicazione e divisione (+, -, * e /), gli operatori logici *and* (&), *or* (|), *not* (^), gli operatori relazionali di maggiore (>), maggiore o uguale (>=), minore (<), minore o uguale (<=), uguale (==). Infine, nessun nome può cominciare con un numero, né contenere una parentesi o un carattere della punteggiatura.

+ - * /
& | ^ >
>= < <=
==

1.6.1 Statements MATLAB

Lo speciale linguaggio di programmazione fornito in MATLAB permette all'utilizzatore di scrivere nuovi programmi. Sebbene non sia richiesto al fine di usare i numerosi programmi che introdurremo in questo testo, la sua conoscenza può tuttavia consentire al lettore di modificarli o di costruirne di completamente nuovi. Il linguaggio MATLAB presenta diverse istruzioni o costrutti (che noi, coerentemente con la versione originale, chiameremo *statements*) fra i quali citiamo i *condizionali* ed i *cicli*.

Lo *statement* condizionale *if-elseif-else* ha la seguente forma generale

```
if cond(1)
    statement(1)
elseif cond(2)
    statement(2)
...
else
    statement(n)
end
```

dove *cond(1)*, *cond(2)*, ... rappresentano espressioni logiche che possono assumere i valori 0 o 1 (falso o vero). L'intera costruzione permette l'esecuzione dello *statement* corrispondente alla prima condizione che assume valore uguale a 1. Se tutte le condizioni fossero false sarebbe eseguito l'ultimo *statement* (lo *statement(n)*). Infatti se *cond(k)* è falsa lo *statement(k)* non viene eseguito ed il controllo passa oltre.

Ad esempio, per calcolare le radici del polinomio $ax^2 + bx + c$ si possono usare le seguenti istruzioni (il comando *disp()* permette di visualizzare il valore assunto dalla variabile riportata nelle parentesi)

```
>> if a ~= 0
    sq = sqrt(b * b - 4 * a * c);
    x(1) = 0.5 * (-b + sq)/a;
    x(2) = 0.5 * (-b - sq)/a;
elseif b ~= 0
    x(1) = -c/b;
elseif c ~= 0
    disp('Equazione impossibile');
(1.12)
```

```

else
    disp('L"equazione data e" un"identita"')
end

```

Il doppio apice nelle stringhe serve per visualizzare gli accenti (o gli apostrofi) ed è necessario dato che il singolo apice è un comando MATLAB. Notiamo inoltre che, digitando l'intera sequenza di istruzioni, essa non verrà eseguita finché l'intera costruzione non sia stata chiusa dallo *statement end*.

for
while In MATLAB sono disponibili due tipi di ciclo: **for** (simile al ciclo **do** del linguaggio Fortran o al ciclo **for** del linguaggio C) e **while**. Un ciclo **for** ripete le istruzioni presenti nel ciclo stesso per tutti i valori dell'indice contenuti in un certo vettore riga. Ad esempio, al fine di calcolare i primi 6 elementi della successione di Fibonacci $\{f_i = f_{i-1} + f_{i-2}, i \geq 2\}$ con $f_1 = 0$ e $f_2 = 1$, si può far ricorso alle seguenti istruzioni

```

>> f(1) = 0; f(2) = 1;
>> for i = [3 4 5 6]
    f(i) = f(i-1) + f(i-2);
end

```

Si noti inoltre che la riga contenente l'istruzione **for** può essere sostituita dall'istruzione equivalente **>> for i = 3:6**. Il ciclo **while** viene invece eseguito sino a quando una data espressione logica è vera. Ad esempio, il seguente insieme di istruzioni può essere usato in alternativa al precedente

```

>> f(1) = 0; f(2) = 1; k = 3;
>> while k <= 6
    f(k) = f(k-1) + f(k-2); k = k + 1;
end

```

Statements di uso meno frequente sono **switch**, **case** e **otherwise**: per la loro descrizione rimandiamo il lettore al corrispondente *help*.

1.6.2 Programmare in MATLAB

In questo paragrafo vogliamo fornire qualche indicazione per la stesura di programmi MATLAB. Un nuovo programma deve essere salvato in un file, chiamato genericamente *m-file* ed avente estensione **.m**. Questo file potrà essere richiamato in MATLAB purché si trovi in una delle cartelle (o *directories*) nelle quali MATLAB cerca i propri m-file. Una lista di tali cartelle, tra le quali è contemplata sempre la cartella corrente, può essere ottenuta usando il comando **path**. Rinviamo all'*help* corrispondente per vedere come aggiungere una cartella a tale lista.

A livello di programmi, è importante distinguere tra *scripts* e *functions*. I primi sono solo una semplice raccolta di istruzioni o comandi

MATLAB, senza un’interfaccia di input/output. Ad esempio, l’insieme delle istruzioni (1.12), una volta salvato in un m-file, di nome ad esempio `equation.m`, diventa uno *script*. Per eseguirlo è sufficiente digitare subito dopo il prompt `>>` il comando `equation`. Riportiamo di seguito due esempi di utilizzo

```
>> a = 1; b = 1; c = 1;
>> equation
ans =
-0.5000 + 0.8660i -0.5000 - 0.8660i

>> a = 0; b = 1; c = 1;
>> equation
ans =
-1
```

Non avendo nessuna interfaccia di input/output tutte le variabili usate in uno *script* sono anche variabili della sessione di lavoro e vengono quindi cancellate solo dietro un esplicito comando (`clear`), caratteristica per nulla soddisfacente quando si intendono scrivere programmi complicati con molte variabili temporanee e relativamente poche variabili di input e di output, le sole che si intendono effettivamente conservare una volta terminata l’esecuzione del programma stesso. Per questo motivo si ricorre ad una forma di programma decisamente più flessibile di uno *script*, chiamata *function*. Una *function* è ancora definita in un m-file, ad esempio `nome.m`, ma possiede una ben precisa interfaccia di input/output introdotta con il comando `function`

```
function [out1,...,outn]=nome(in1,...,inm)
```

dove `out1, ..., outn` sono le variabili di output e `in1, ..., inm` quelle di input.

Il seguente file, chiamato `det23.m`, è un esempio di *function*: in esso viene definita una nuova *function*, chiamata `det23`, che calcola, secondo la formula data nel paragrafo 1.3, il determinante di una matrice quadrata la cui dimensione può essere 2 o 3

```
function [det]=det23(A)
%DET23 calcola il determinante di una matrice quadrata di dimensione 2 o 3
[n,m]=size(A);
if n==m
    if n==2
        det = A(1,1)*A(2,2)-A(2,1)*A(1,2);
    elseif n == 3
        det = A(1,1)*det23(A[2,3],[2,3]))-A(1,2)*det23(A([2,3],[1,3]))+...
            A(1,3)*det23(A[2,3],[1,2]));
    else
        disp(' Solo matrici 2x2 o 3x3 ');
    end
end
```

```

end
else
    disp(' Solo matrici quadrate ');
end
return

```

...
% Si noti l'uso dei caratteri di continuazione ... a significare che l'istruzione continua nella linea seguente e del carattere % per denotare una riga di commento. L'istruzione $A([i,j],[k,1])$ consente la costruzione di una matrice 2×2 i cui elementi sono quelli della matrice originaria A giacenti alle intersezioni delle righe i -esima e j -esima con le colonne k -esima e 1-esima.

Quando si invoca una *function*, MATLAB crea un'area di lavoro locale (il *function's workspace*) nella quale memorizza le variabili richiamate all'interno della *function* stessa. Di conseguenza, le istruzioni contenute in una *function* non possono riferirsi a variabili dichiarate nel *base workspace* a meno che queste non rientrino fra i parametri in input.³ In particolare, tutte le variabili usate in una *function* vanno perdute a fine esecuzione a meno che non siano tra i parametri di output.

A questo proposito facciamo osservare che l'esecuzione di una *function* termina quando si raggiunge l'ultima istruzione o quando si incontra per la prima volta il comando **return**. Ad esempio, al fine di approssimare il valore $\alpha = 1.6180339887\dots$, che rappresenta il limite per $k \rightarrow \infty$ del rapporto f_k/f_{k-1} nella successione di Fibonacci, iterando sino a quando due frazioni consecutive differiscano per meno di 10^{-4} , possiamo costruire la seguente *function*

```

function [golden,k]=fibonacci
f(1) = 0; f(2) = 1; goldenold = 0; kmax = 100; tol = 1.e-04;
for k = 3:kmax
    f(k) = f(k-1) + f(k-2);
    golden = f(k)/f(k-1);
    if abs(golden - goldenold) <= tol
        return
    end
    goldenold = golden;
end
return

```

La sua esecuzione si interrompe o dopo **kmax=100** iterazioni o quando il valore assoluto della differenza fra due iterate consecutive è minore di **tol=1.e-04**. Possiamo eseguire questa *function* scrivendo

³ È disponibile un terzo tipo di *workspace*, il *global workspace* nel quale vengono memorizzate le variabili dichiarate come **global**. Tali variabili possono essere usate in una *function* anche se non rientrano tra i parametri in input.

```
>> [alpha,niter]=fibonacci
alpha =
    1.618055555555556
niter =
    14
```

Dunque, dopo 14 iterazioni la *function* restituisce un valore approssimato che condivide con il vero α le prime 5 cifre significative.

Il numero di parametri di input e di output di una *function* MATLAB può variare. Per esempio, la *function* **fibonacci** appena vista potrebbe modificarsi come segue

```
function [golden,k]=fibonacci(tol,kmax)
if nargin == 0
    kmax = 100; tol = 1.e-04; % valori di default
elseif nargin == 1
    kmax = 100; % valore di default per kmax
end
f(1) = 0; f(2) = 1; goldenold = 0;
for k = 3:kmax
    f(k) = f(k-1) + f(k-2);
    golden = f(k)/f(k-1);
    if abs(golden - goldenold) <= tol
        return
    end
    goldenold = golden;
end
return
```

La *function* **nargin** conta il numero di parametri di input (in modo analogo, con **nargout** si contano i parametri di output). In questa nuova versione della *function* **fibonacci** possiamo prescrivere il massimo numero di iterazioni consentite (**kmax**) ed una specifica tolleranza **tol** oppure, non passando tali variabili in input, accettare i valori di *default* predisposti all'interno della *function* stessa (**kmax=100** e **tol=1.e-04**). Un uso possibile allora è il seguente

```
>> [alpha,niter]=fibonacci(1.e-6,200)
alpha =
    1.61803381340013
niter =
    19
```

Si noti che avendo scelto una tolleranza più restrittiva sul criterio d'arresto abbiamo calcolato una nuova approssimazione che condivide con il vero α ben 8 cifre significative. La *function* **nargin** può anche essere usata esternamente ad una *function* per ottenere il numero massimo di parametri di input di un'altra *function*. Ad esempio

nargin
nargout

```
>> nargin('fibonacci')
ans =
2
```

inline **Osservazione 1.2 (inline functions)** Il comando **inline**, la cui sintassi più semplice è `g=inline(expr,arg1,arg2,...,argn)`, dichiara una *function* g che dipende dalle stringhe `arg1,arg2,...,argn`. La stringa `expr` contiene l'espressione di g. Ad esempio, `g=inline('sin(r)','r')` dichiara la funzione $g(r) = \sin(r)$. La forma abbreviata `g=inline(expr)` assume implicitamente che l'espressione `expr` dipenda dalla sola variabile `x`. Una volta dichiarata, una *inline function* può essere valutata su un qualunque insieme di variabili attraverso il comando **feval**. Ad esempio, per valutare g nei punti `z=[0 1]` possiamo scrivere

```
>> feval('g',z);
```

Notiamo come, contrariamente al caso del comando **eval**, con **feval** il nome della variabile (`z`) non deve necessariamente coincidere con il nome simbolico (`r`) assegnato tramite il comando **inline**.

Dopo questa breve introduzione, l'invito è quello di esplorare MATLAB utilizzandone l'*help*. Ad esempio, scrivendo `help for` non solo si riceve una corposa descrizione di questa istruzione, ma al termine vengono anche indicate altre istruzioni simili a `for` (in questo caso `if`, `while`, `switch`, `break`, `end`). Invocandone l'*help* potremo dunque ampliare progressivamente la nostra conoscenza di MATLAB.



Si vedano gli Esercizi 1.9-1.14.

1.7 Cosa non vi abbiamo detto

Una trattazione più sistematica dei numeri *floating-point* può essere trovata in [QSS04] o in [Üeb97].

Per quanto riguarda la complessità computazionale e l'algoritmica in generale, rimandiamo a [BC98] e a [Pan92] per gli approfondimenti.

Per una sistematica introduzione a MATLAB il lettore interessato può consultare il manuale MATLAB [HH00] o [Mat94], ma anche monografie quali [HLR01], [EKH02], [Pal04] o [MH03].

1.8 Esercizi

Esercizio 1.1 Da quanti numeri è costituito l'insieme $\mathbb{F}(2, 2, -2, 2)$? Quanto vale ϵ_M per tale insieme?

Esercizio 1.2 Si verifichi che in generale l'insieme $\mathbb{F}(\beta, t, L, U)$ contiene $2(\beta - 1)\beta^{t-1}(U - L + 1)$ numeri.

Esercizio 1.3 Si dimostri che i^i è un numero reale e si verifichi il risultato in MATLAB.

Esercizio 1.4 Si costruiscano in MATLAB una matrice triangolare superiore ed una triangolare inferiore di dimensione 10 con 2 sulla diagonale principale e -3 sulla seconda sopra (rispettivamente, sotto) diagonale.

Esercizio 1.5 Si scrivano le istruzioni MATLAB che consentono di scambiare fra loro la terza e la settima riga delle matrici costruite nell'Esercizio 1.4, indi quelle per scambiare l'ottava con la quarta colonna.

Esercizio 1.6 Si stabilisca se i seguenti vettori di \mathbb{R}^4 sono fra loro linearmente indipendenti

$$\mathbf{v}_1 = [0 \ 1 \ 0 \ 1], \mathbf{v}_2 = [1 \ 2 \ 3 \ 4], \mathbf{v}_3 = [1 \ 0 \ 1 \ 0], \mathbf{v}_4 = [0 \ 0 \ 1 \ 1].$$

Esercizio 1.7 Si scrivano in MATLAB le seguenti funzioni e si calcolino con il toolbox simbolico derivata prima e seconda ed integrale indefinito

$$f(x) = \sqrt{x^2 + 1}, \quad g(x) = \sin(x^3) + \cosh(x).$$

Esercizio 1.8 Dato un vettore \mathbf{v} di dimensione n , scrivendo `c=poly(v)` è [poly](#) possibile costruire i coefficienti, memorizzati nel vettore \mathbf{c} , di un polinomio di grado n con coefficiente relativo a x^n uguale a 1, che abbia come radici proprio i valori memorizzati in \mathbf{v} . Ci si aspetta pertanto di trovare che $\mathbf{v} = \text{roots}(\text{poly}(\mathbf{c}))$. Si provi a calcolare `roots(poly([1:n]))` dove n varia da 2 fino a 25 e si commentino i risultati ottenuti.

Esercizio 1.9 Si scriva un programma per il calcolo della seguente successione

$$I_0 = \frac{1}{e}(e - 1),$$

$$I_{n+1} = 1 - (n + 1)I_n, \text{ per } n = 0, 1, \dots, 21.$$

Sapendo che $I_n \rightarrow 0$ per $n \rightarrow \infty$, si commentino i risultati ottenuti.

Esercizio 1.10 Si spieghi il comportamento della successione (1.4) quando implementata con MATLAB.

Esercizio 1.11 Per il calcolo di π si può usare la seguente tecnica: si generano n coppie (x_k, y_k) di numeri casuali compresi fra 0 e 1 e di questi si calcola il numero m di punti che cadono nel primo quarto del cerchio di centro l'origine e raggio 1. Si ha che π è il limite per n che tende all'infinito dei rapporti $\pi_n = 4m/n$. Si scriva un programma che esegua questo calcolo e si verifichi la correttezza del risultato al crescere di n .

Esercizio 1.12 Sempre per il calcolo di π si può utilizzare una troncata della seguente serie

$$\pi = \sum_{n=0}^{\infty} 16^{-n} \left(\frac{4}{8n+1} - \frac{2}{8n+4} - \frac{1}{8n+5} - \frac{1}{8n+6} \right).$$

Si realizzi una *function* MATLAB che ne calcola la somma fino ad un certo n fissato. Quanto grande deve essere n per ottenere un valore di π confrontabile con quello memorizzato nella variabile `pi`?

Esercizio 1.13 Si scriva un programma per il calcolo del coefficiente binomiale $\binom{n}{k} = n!/(k!(n-k)!)$, dove n e k sono numeri naturali con $k \leq n$.

Esercizio 1.14 Si realizzi una *function* che calcola l'elemento f_n della successione di Fibonacci in forma ricorsiva. Osservando poi che

$$\begin{bmatrix} f_i \\ f_{i-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} f_{i-1} \\ f_{i-2} \end{bmatrix} \quad (1.13)$$

si realizzi un'altra *function* che calcola f_n sfruttando questa relazione. Si confrontino i tempi di calcolo.

2

Equazioni non lineari

Il calcolo degli *zeri* di una funzione f reale di variabile reale o delle *radici* dell'equazione $f(x) = 0$ è un problema assai ricorrente nel calcolo scientifico. In generale non è possibile approntare metodi numerici che calcolino gli zeri di una generica funzione in un numero finito di passi. Abbiamo ad esempio ricordato nel paragrafo 1.4.1 che un teorema dell'Algebra esclude la possibilità di calcolare con un numero finito di operazioni gli zeri di un generico polinomio di grado maggiore di 4. La situazione è ancor più complicata quando f è una funzione non polinomiale.

I metodi numerici per la risoluzione di questo problema sono pertanto necessariamente *iterativi*. A partire da uno o più dati iniziali, scelti convenientemente, essi generano una successione di valori $x^{(k)}$ che, sotto opportune ipotesi, convergerà ad uno zero α della funzione f studiata.

Problema 2.1 (Piano di investimento) Si vuol calcolare il tasso medio di rendita I di un fondo di investimento su più anni. Supponiamo che si investano nel fondo v euro all'inizio di ogni anno e che alla fine dell'ennesimo anno si sia accumulato un montante pari a M euro. Essendo M legato a I dalla seguente relazione

$$M = v \sum_{k=1}^n (1+I)^k = v \frac{1+I}{I} [(1+I)^n - 1],$$

deduciamo che I è la radice dell'equazione non lineare

$$f(I) = 0, \quad \text{dove } f(I) = M - v \frac{1+I}{I} [(1+I)^n - 1].$$

Per la soluzione di questo problema, si veda l'Esempio 2.1.

Problema 2.2 (Equazione di stato di un gas) Si vuole determinare il volume V occupato da un gas ad una temperatura T e soggetto ad una pressione p . L'equazione di stato (ossia l'equazione che lega p , V e T) è

$$[p + a(N/V)^2] (V - Nb) = kNT, \quad (2.1)$$

nella quale a e b sono dei coefficienti che dipendono dallo specifico tipo di gas, N è il numero di molecole di gas contenute nel volume V e k è la cosiddetta costante di Boltzmann. Dobbiamo quindi risolvere un'equazione non lineare la cui radice è V . Per la soluzione di questo problema si veda l'Esercizio 2.2.

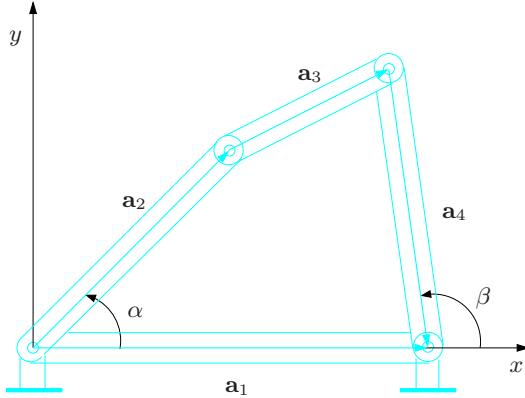


Figura 2.1. Il sistema di quattro aste del Problema 2.3

Problema 2.3 (Statica) Consideriamo il sistema meccanico costituito dalle quattro aste rigide \mathbf{a}_i di Figura 2.1; si vuole stabilire, in corrispondenza di un fissato angolo β , quale sia l'angolo α fra le aste \mathbf{a}_1 e \mathbf{a}_2 . A partire dall'identità vettoriale

$$\mathbf{a}_1 - \mathbf{a}_2 - \mathbf{a}_3 - \mathbf{a}_4 = \mathbf{0}$$

ed osservando che l'asta \mathbf{a}_1 è sempre allineata con l'asse delle ascisse, è possibile ricavare la seguente relazione tra β e α

$$\frac{a_1}{a_2} \cos(\beta) - \frac{a_1}{a_4} \cos(\alpha) - \cos(\beta - \alpha) = -\frac{a_1^2 + a_2^2 - a_3^2 + a_4^2}{2a_2a_4}, \quad (2.2)$$

avendo indicato con a_i la lunghezza dell' i -esima asta. Evidentemente tale equazione, detta di Freudenstein, si può riscrivere come: $f(\alpha) = 0$, essendo

$$f(x) = \frac{a_1}{a_2} \cos(\beta) - \frac{a_1}{a_4} \cos(x) - \cos(\beta - x) + \frac{a_1^2 + a_2^2 - a_3^2 + a_4^2}{2a_2a_4}. \quad (2.3)$$

Essa può essere risolta analiticamente solo per particolari valori di β . Si tenga inoltre conto che non per tutti i valori di β la soluzione esiste o, se esiste, è unica. Per la sua risoluzione nel caso generale in cui β assuma un valore arbitrario compreso fra 0 e π si dovrà ricorrere ad un metodo numerico (si veda l'Esercizio 2.9).

Problema 2.4 (Dinamica delle popolazioni) Nello studio della dinamica delle popolazioni (di batteri, ad esempio) l'equazione $x^+ = \phi(x) = xR(x)$ stabilisce un legame fra il numero x di individui di una generazione ed il numero x^+ di individui della generazione seguente. La funzione $R(x)$ modella il tasso di variazione della popolazione in esame e può essere scelta in vari modi. Tra i più noti, ricordiamo:

1. il modello di Malthus (Thomas Malthus, 1766-1834),

$$R(x) = R_M(x) = r, \quad r > 0;$$

2. il modello di crescita in presenza di risorse limitate (proposto da Pierre Francois Verhulst, 1804-1849)

$$R(x) = R_V(x) = \frac{r}{1 + xK}, \quad r > 0, K > 0, \quad (2.4)$$

che migliora il modello di Malthus tenendo conto del fatto che la crescita della popolazione è limitata dalle risorse disponibili;

3. il modello predatore/presa con saturazione

$$R(x) = R_P = \frac{rx}{1 + (x/K)^2}, \quad (2.5)$$

che può essere visto come l'evoluzione del modello di Verhulst in presenza di una popolazione antagonista.

La dinamica di una popolazione è quindi descritta dal processo iterativo

$$x^{(k)} = \phi(x^{(k-1)}), \quad k > 0, \quad (2.6)$$

dove $x^{(k)}$ rappresenta il numero di individui presenti k generazioni dopo la generazione iniziale $x^{(0)}$. Inoltre, gli stati stazionari (o di equilibrio) x^* della popolazione considerata sono definiti come le soluzioni del problema

$$x^* = \phi(x^*),$$

o, equivalentemente, $x^* = x^* R(x^*)$, ovvero $R(x^*) = 1$. La (2.6) è un esempio di metodo di punto fisso (si veda la Sezione 2.3).

2.1 Il metodo di bisezione

Sia f una funzione continua in $[a, b]$ e tale che $f(a)f(b) < 0$. Sotto tali ipotesi f ammette almeno uno zero in (a, b) . Supponiamo per semplicità che ne abbia uno solo che indicheremo con α . Nel caso in cui f presenti più zeri è sempre possibile, ad esempio attraverso uno studio grafico con il comando `fplot`, individuare un intervallo che ne contenga uno solo.

La strategia del metodo di bisezione consiste nel dimezzare l'intervallo di partenza, selezionare tra i due sotto-intervalli ottenuti quello nel quale f cambia di segno agli estremi ed applicare ricorsivamente questa procedura all'ultimo intervallo selezionato. Più precisamente, detto $I^{(0)} = (a, b)$ e, più in generale, $I^{(k)}$ il sotto-intervallo selezionato al passo k -esimo, si sceglie come $I^{(k+1)}$ il semi-intervallo di $I^{(k)}$ ai cui estremi f cambia di segno. Con tale procedura si è certi che ogni $I^{(k)}$ così individuato conterrà α . La successione $\{x^{(k)}\}$ dei punti medi dei sotto-intervalli $I^{(k)}$ dovrà ineluttabilmente convergere a α , in quanto la lunghezza dei sotto-intervalli tende a 0 per k che tende all'infinito.

Formalizziamo questa idea, ponendo

$$a^{(0)} = a, b^{(0)} = b, I^{(0)} = (a^{(0)}, b^{(0)}), x^{(0)} = (a^{(0)} + b^{(0)})/2.$$

Al generico passo $k \geq 1$ il metodo di bisezione calcolerà allora il semi-intervallo $I^{(k)} = (a^{(k)}, b^{(k)})$ dell'intervallo $I^{(k-1)} = (a^{(k-1)}, b^{(k-1)})$ nel modo seguente:

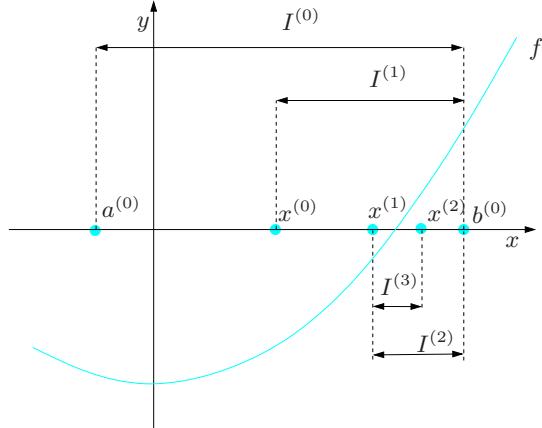


Figura 2.2. Alcune iterazioni del metodo di bisezione

dato $x^{(k-1)} = (a^{(k-1)} + b^{(k-1)})/2$, se $f(x^{(k-1)}) = 0$, allora $\alpha = x^{(k-1)}$ ed il metodo si arresta;

altrimenti,

se $f(a^{(k-1)})f(x^{(k-1)}) < 0$ si pone $a^{(k)} = a^{(k-1)}$, $b^{(k)} = x^{(k-1)}$;

se $f(x^{(k-1)})f(b^{(k-1)}) < 0$ si pone $a^{(k)} = x^{(k-1)}$, $b^{(k)} = b^{(k-1)}$.

In entrambi i casi si definisce $x^{(k)} = (a^{(k)} + b^{(k)})/2$ e si incrementa k di uno.

Ad esempio, nel caso rappresentato in Figura 2.2 se $f(x) = x^2 - 1$, a partire da $a^{(0)} = -0.25$ e $b^{(0)} = 1.25$, otterremmo

$$\begin{aligned} I^{(0)} &= (-0.25, 1.25), & x^{(0)} &= 0.5, \\ I^{(1)} &= (0.5, 1.25), & x^{(1)} &= 0.875, \\ I^{(2)} &= (0.875, 1.25), & x^{(2)} &= 1.0625, \\ I^{(3)} &= (0.875, 1.0625), & x^{(3)} &= 0.96875. \end{aligned}$$

Si noti che ognuno degli intervalli $I^{(k)}$ contiene lo zero $\alpha = 1$. Inoltre, la successione $\{x^{(k)}\}$ converge necessariamente allo zero α in quanto ad ogni passo l'ampiezza $|I^{(k)}| = b^{(k)} - a^{(k)}$ dell'intervalle $I^{(k)}$ si dimezza. Essendo allora $|I^{(k)}| = (1/2)^k |I^{(0)}|$, l'errore al passo k sarà tale che

$$|e^{(k)}| = |x^{(k)} - \alpha| < \frac{1}{2} |I^{(k)}| = \left(\frac{1}{2}\right)^{k+1} (b - a).$$

Al fine di garantire che $|e^{(k)}| < \varepsilon$ per una assegnata tolleranza ε , basta allora fermarsi dopo k_{min} iterazioni, essendo k_{min} il primo intero che soddisfa la diseguaglianza

$$k_{min} > \log_2 \left(\frac{b-a}{\varepsilon} \right) - 1. \quad (2.7)$$

Naturalmente, questa diseguaglianza non dipende dalla particolare funzione f scelta in precedenza.

Il metodo di bisezione è implementato nel Programma 1: `fun` è una *function* (o una *inline function*) che specifica la funzione f , `a` e `b` sono gli estremi dell'intervallo di ricerca, `tol` la tolleranza ε e `nmax` il massimo numero consentito di iterazioni. `fun` oltre al primo argomento relativo alla variabile indipendente, può accettare altri argomenti opzionali impiegati nella definizione di f .

In uscita, `zero` contiene lo zero calcolato, `res` il residuo, ovvero il valore assunto da f in `zero`, e `niter` il numero di iterazioni effettuate. Il comando `find(fx==0)` serve per trovare gli indici del vettore `fx` corrispondenti ad elementi nulli, mentre il comando `varargin` permette alla *function* `fun` di accettare un numero di parametri d'ingresso variabile.

Programma 1 - bisection : il metodo di bisezione

`find`
`varargin`



```
function [zero,res,niter]=bisection(fun,a,b,tol,nmax,varargin)
%BISECTION Trova uno zero di una funzione.
% ZERO=BISECTION(FUN,A,B,TOL,NMAX) approssima uno zero della
% funzione FUN nell'intervallo [A,B] con il metodo di bisezione. FUN accetta
% come input uno scalare reale x e restituisce uno scalare reale. Se la ricerca
% dello zero di FUN fallisce, il programma restituisce un messaggio d'errore.
% FUN puo' essere una inline function.
% ZERO=BISECTION(FUN,A,B,TOL,NMAX,P1,P2,...) passa i parametri P1,
% P2,... alla funzione FUN(X,P1,P2,...).
% [ZERO,RES,NITER]= BISECTION(FUN,...) restituisce il valore del residuo
% RES in ZERO ed il numero di iterazioni effettuate per calcolare il valore ZERO.
x = [a, (a+b)*0.5, b]; fx = feval(fun,x,varargin{:});
if fx(1)*fx(3)>0
    error(' Il segno della funzione agli estremi dell''intervallo [A,B] deve essere diverso');
elseif fx(1) == 0
    zero = a; res = 0; niter = 0; return
elseif fx(3) == 0
    zero = b; res = 0; niter = 0; return
end
niter = 0; I = (b - a)*0.5;
while I >= tol & niter <= nmax
    niter = niter + 1;
    if fx(1)*fx(2) < 0
        x(3) = x(2); x(2) = x(1)+(x(3)-x(1))*0.5;
        fx = feval(fun,x,varargin{:}); I = (x(3)-x(1))*0.5;
    elseif fx(2)*fx(3) < 0
        x(1) = x(2); x(2) = x(1)+(x(3)-x(1))*0.5;
        fx = feval(fun,x,varargin{:}); I = (x(3)-x(1))*0.5;
```

```

else
    x(2) = x(find(fx==0)); l = 0;
end
end
if niter > nmax
    fprintf(['Il metodo di bisezione si e'' arrestato senza soddisfare la tolleranza richiesta',...
        ' avendo raggiunto il numero massimo di iterazioni\n']);
end
zero = x(2); x = x(2); res = feval(fun,x,varargin{:});
return

```

Esempio 2.1 (Piano di investimento) Risolviamo con il metodo di bisezione il Problema 2.1, supponendo che v sia pari a 1000 euro e che, dopo 5 anni, M sia uguale a 6000 euro. Dal grafico della funzione f , ottenuto con le seguenti istruzioni

```

>> f=inline('M-v*(1+l).*((1+l).^5 - 1)./l','l','M','v');
>> fplot(f,[0.01,0.3],[],[],[],6000,1000)

```

si ricava che f presenta un'unica radice nell'intervallo $(0.01, 0.1)$, pari a circa 0.06. Eseguiamo quindi il Programma 1 con $\text{tol} = 10^{-12}$, $a = 0.01$ e $b = 0.1$ con il comando

```

>> [zero,res,niter]=bisection(f,0.01,0.1,1.e-12,1000,6000,1000);

```

Il metodo converge dopo 36 iterazioni al valore 0.06140241153618, in perfetto accordo con la stima (2.7) per la quale $k_{min} = 36$. Si può quindi concludere che il tasso di interesse I è pari al 6.14%.

Il metodo di bisezione non garantisce una riduzione progressiva dell'errore, ma solo il dimezzamento dell'ampiezza dell'intervallo all'interno del quale si cerca lo zero. Per questo motivo possono essere inavvertitamente scartate approssimazioni di α assai accurate se si usa come unico criterio d'arresto quello sulla lunghezza dell'intervallo $I^{(k)}$.

Questo metodo non tiene infatti conto del reale andamento di f ed in effetti, a meno che l'intervallo di partenza non sia simmetrico rispetto allo zero cercato, esso non converge allo zero in un solo passo neppure se f è una funzione lineare.



Si vedano gli Esercizi 2.1-2.5.

2.2 Il metodo di Newton

Il metodo di bisezione si limita ad utilizzare il segno che la funzione f assume in certi punti (gli estremi dei sotto-intervalli). Vogliamo ora introdurre un metodo che sfrutta maggiori informazioni su f , precisamente

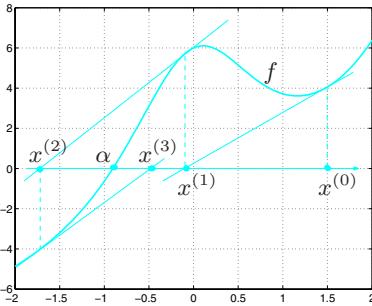


Figura 2.3. Prime iterate generate dal metodo di Newton a partire dal dato iniziale $x^{(0)}$ per la funzione $f(x) = x + e^x + 10/(1+x^2) - 5$

i suoi valori e quelli della sua derivata (nell'ipotesi che quest'ultima esista). A tal fine ricordiamo che l'equazione della retta tangente alla curva $(x, f(x))$ nel punto $x^{(k)}$ è

$$y(x) = f(x^{(k)}) + f'(x^{(k)})(x - x^{(k)}).$$

Se cerchiamo $x^{(k+1)}$ tale che $y(x^{(k+1)}) = 0$, troviamo la seguente formula

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{f'(x^{(k)})}, \text{ purché } f'(x^{(k)}) \neq 0, k \geq 0 \quad (2.8)$$

La (2.8) consente di calcolare una successione di valori $x^{(k)}$ a partire da un dato iniziale $x^{(0)}$. Il metodo così ottenuto è noto come *metodo di Newton* ed equivale a calcolare lo zero di f sostituendo localmente a f la sua retta tangente (si veda la Figura 2.3).

In effetti, se sviluppiamo f in serie di Taylor in un intorno di un generico punto $x^{(k)}$ troviamo

$$f(x^{(k+1)}) = f(x^{(k)}) + \delta^{(k)} f'(x^{(k)}) + \mathcal{O}((\delta^{(k)})^2), \quad (2.9)$$

dove $\delta^{(k)} = x^{(k+1)} - x^{(k)}$. Imponendo che $f(x^{(k+1)})$ sia nullo e trascurando il termine $\mathcal{O}((\delta^{(k)})^2)$, possiamo ricavare $x^{(k+1)}$ in funzione di $x^{(k)}$ come nella (2.8). In questo senso la (2.8) può essere vista come una approssimazione della (2.9).

Evidentemente, (2.8) converge allo zero in un solo passo quando f è lineare, cioè della forma $f(x) = a_1 x + a_0$.

Esempio 2.2 Risolviamo con il metodo di Newton lo stesso caso dell'Esempio 2.1 a partire dal dato iniziale $x^{(0)} = 0.3$. Il metodo converge allo zero cercato e dopo 6 iterazioni la differenza fra due iterate successive è minore di 10^{-12} .

 La convergenza del metodo di Newton non è garantita per ogni scelta di $x^{(0)}$, ma soltanto per valori di $x^{(0)}$ sufficientemente vicini ad α . Questa richiesta a prima vista sembra insensata: per trovare l'incognita α abbiamo bisogno di scegliere $x^{(0)}$ sufficientemente vicino a α , quando α è proprio il valore sconosciuto!

In pratica, tali valori possono essere ottenuti utilizzando ad esempio poche iterazioni del metodo di bisezione, oppure attraverso uno studio del grafico di f . Se $x^{(0)}$ è stato scelto opportunamente e se lo zero α è semplice, ovvero se $f'(\alpha) \neq 0$, allora il metodo di Newton converge. Inoltre, nel caso in cui f è derivabile con continuità due volte, otteniamo il seguente risultato di convergenza (si veda l'Esercizio 2.8)

$$\lim_{k \rightarrow \infty} \frac{x^{(k+1)} - \alpha}{(x^{(k)} - \alpha)^2} = \frac{f''(\alpha)}{2f'(\alpha)} \quad (2.10)$$

La (2.10) afferma che se $f'(\alpha) \neq 0$ il metodo di Newton converge almeno *quadraticamente* o con ordine 2 nel senso che, per k sufficientemente grande, l'errore al passo $(k + 1)$ -esimo si comporta come il quadrato dell'errore al passo k -esimo, moltiplicato per una costante indipendente da k .

Se lo zero ha invece molteplicità m maggiore di 1, il metodo ha ordine di convergenza pari a 1 (si veda l'Esercizio 2.15). In tal caso, l'ordine 2 può essere ancora recuperato usando anziché (2.8) la relazione

$$x^{(k+1)} = x^{(k)} - m \frac{f(x^{(k)})}{f'(x^{(k)})}, \text{ purché } f'(x^{(k)}) \neq 0, k \geq 0. \quad (2.11)$$

Naturalmente, questo *metodo di Newton modificato* richiede una conoscenza a priori di m . In mancanza di tale informazione si può formulare un *metodo di Newton adattivo*, ancora di ordine 2, come riportato in [QSS04], paragrafo 6.6.2.

Esempio 2.3 La funzione $f(x) = (x - 1) \log(x)$ ha un solo zero, $\alpha = 1$, di molteplicità $m = 2$. Calcoliamolo con il metodo di Newton (2.8) e con la sua versione modificata (2.11). Nel grafico di Figura 2.4 viene riportato l'errore ottenuto con i due metodi in funzione del numero di iterazioni. Come si vede, nel caso del metodo classico (2.8) l'errore decresce solo linearmente.

2.2.1 Come arrestare il metodo di Newton

Il metodo di Newton, quando converge, restituisce il valore esatto di α solo dopo un numero infinito di iterazioni. D'altra parte in generale ci si accontenta di ottenere α a meno di una tolleranza fissata ε : è quindi sufficiente arrestarsi alla prima iterata k_{min} in corrispondenza della quale si abbia

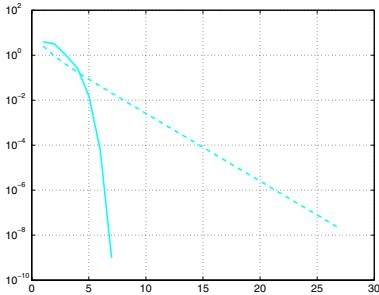


Figura 2.4. Errore in funzione del numero di iterazioni per la funzione dell’Esempio 2.3. La curva tratteggiata corrisponde al metodo di Newton (2.8), quella continua al metodo di Newton modificato (2.11) (con $m = 2$)

$$|e^{(k_{min})}| = |\alpha - x^{(k_{min})}| < \varepsilon.$$

Si tratta di un test sull’errore. Sfortunatamente essendo l’errore incognito, è necessario impiegare in sua vece degli *stimatori dell’errore* vale a dire delle quantità facilmente calcolabili grazie alle quali sia possibile maggiorare l’errore stesso. Come vedremo al termine del paragrafo 2.3, come stimatore dell’errore per il metodo di Newton possiamo prendere la *differenza fra due iterate consecutive* e ci si arresta cioè in corrispondenza del più piccolo intero k_{min} per il quale

$$|x^{(k_{min})} - x^{(k_{min}-1)}| < \varepsilon. \quad (2.12)$$

Si tratta di un test sull’incremento. Come vedremo nel paragrafo 2.3.1, questo è un buon criterio quando lo zero cercato è semplice. Uno stimatore alternativo, anche per metodi iterativi diversi da quello di Newton volti a trovare gli zeri di una funzione f , è dato dal *residuo* definito come $r^{(k)} = f(x^{(k)})$ che è nullo quando $x^{(k)}$ è uno zero di f .

Il metodo viene in tal caso arrestato alla prima iterata k_{min} per cui

$$|r^{(k_{min})}| = |f(x^{(k_{min})})| < \varepsilon.$$

Il residuo fornisce una stima accurata dell’errore solo quando $|f'(x)|$ è circa pari a 1 in un intorno I_α dello zero α cercato (si veda la Figura 2.5). In caso contrario, porterà ad una sovrastima dell’errore se $|f'(x)| \gg 1$ per $x \in I_\alpha$ o ad una sottostima se $|f'(x)| \ll 1$ (si veda anche l’Esercizio 2.6).

Nel Programma 2 viene riportata una implementazione del metodo di Newton nella sua forma (2.8) (per utilizzare la forma modificata è sufficiente inserire, invece di f' , la funzione f'/m). I parametri `fun` e `dfun` sono le stringhe contenenti la funzione f e la sua derivata prima, mentre `x0` è il dato iniziale. Il metodo viene arrestato se il valore assoluto

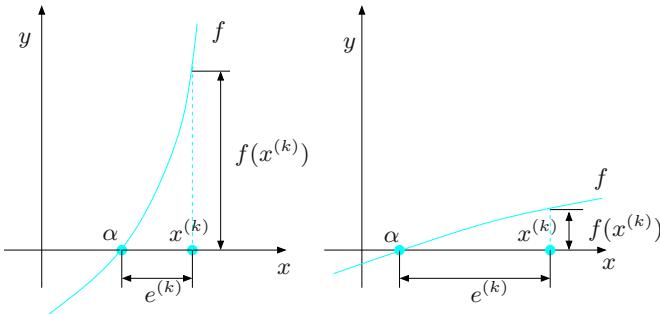


Figura 2.5. Le due possibili situazioni nelle quali il residuo non è un buon stimatore dell'errore: $|f'(x)| \gg 1$ a sinistra, $|f'(x)| \ll 1$ a destra, con x appartenente ad un intervallo contenente α

della differenza fra due iterate consecutive è minore della tolleranza `tol` o se è stato oltrepassato il massimo numero di iterazioni consentito, `nmax`.



Programma 2 - `newton` : il metodo di Newton

```
function [zero,res,niter]=newton(fun,dfun,x0,tol,nmax,varargin)
%NEWTON Trova uno zero di una funzione.
% ZERO=NEWTON(FUN,DFUN,X0,TOL,NMAX) approssima lo zero ZERO
% della funzione definita nella function FUN, continua e derivabile, piu' vicino
% a X0 usando il metodo di Newton. FUN e la sua derivata DFUN accettano
% in ingresso uno scalare x e restituiscono un valore scalare. Se la ricerca dello
% zero fallisce, il programma restituisce un messaggio d'errore. FUN e DFUN
% possono essere inline functions.
% ZERO=NEWTON(FUN,DFUN,X0,TOL,NMAX,P1,P2,...) passa i parametri
% P1,P2,... alle funzioni FUN(X,P1,P2,...) e DFUN(X,P1,P2,...).
% [ZERO,RES,NITER]= NEWTON(FUN,...) restituisce il valore del residuo RES
% in ZERO ed il numero di iterazioni NITER necessario per calcolare ZERO.
x = x0; fx = feval(fun,x,varargin{:}); dfx = feval(dfun,x,varargin{:});
niter = 0; diff = tol+1;
while diff >= tol & niter <= nmax
    niter = niter + 1;
    diff = - fx/dfx;
    x = x + diff;
    diff = abs(diff);
    fx = feval(fun,x,varargin{:});
    dfx = feval(dfun,x,varargin{:});
end
if niter > nmax
    fprintf(['newton si e'' arrestato senza aver soddisfatto l''accuratezza richiesta',...
        ' avendo raggiunto il massimo numero di iterazioni\n']);
end
zero = x; res = fx;
return
```

2.2.2 Il metodo di Newton per sistemi di equazioni non lineari

Consideriamo il seguente sistema di equazioni non lineari

$$\begin{cases} f_1(x_1, x_2, \dots, x_n) = 0, \\ f_2(x_1, x_2, \dots, x_n) = 0, \\ \vdots \\ f_n(x_1, x_2, \dots, x_n) = 0, \end{cases} \quad (2.13)$$

dove f_1, \dots, f_n sono funzioni non lineari. Se poniamo $\mathbf{f} \equiv (f_1, \dots, f_n)^T$ e $\mathbf{x} \equiv (x_1, \dots, x_n)^T$, possiamo riscrivere il sistema (2.13) nella forma

$$\mathbf{f}(\mathbf{x}) = \mathbf{0}. \quad (2.14)$$

Un semplice esempio di sistema non lineare è il seguente

$$\begin{cases} f_1(x_1, x_2) = x_1^2 + x_2^2 - 1 = 0, \\ f_2(x_1, x_2) = \sin(\pi x_1/2) + x_2^3 = 0. \end{cases} \quad (2.15)$$

Al fine di estendere il metodo di Newton al caso di un sistema, sostituiamo alla derivata prima della funzione scalare f la matrice jacobiana J_f della funzione vettoriale \mathbf{f} , le cui componenti sono

$$(J_f)_{ij} \equiv \frac{\partial f_i}{\partial x_j}, \quad i, j = 1, \dots, n.$$

Il simbolo $\partial f_i / \partial x_j$ rappresenta la derivata parziale di f_i rispetto a x_j (si veda la definizione 8.3). Con questa notazione, il metodo di Newton per (2.14) diventa: dato $\mathbf{x}^{(0)} \in \mathbb{R}^n$, per $k = 0, 1, \dots$, fino a convergenza

$$\begin{aligned} &\text{risolvere } J_f(\mathbf{x}^{(k)}) \boldsymbol{\delta}\mathbf{x}^{(k)} = -\mathbf{f}(\mathbf{x}^{(k)}); \\ &\text{porre } \mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \boldsymbol{\delta}\mathbf{x}^{(k)} \end{aligned} \quad (2.16)$$

Di conseguenza, esso richiede ad ogni passo la soluzione di un sistema lineare di matrice $J_f(\mathbf{x}^{(k)})$.

Il Programma 3 implementa il metodo di Newton per un sistema non lineare usando il comando `\` di MATLAB (si veda il paragrafo 5.6) per risolvere il sistema lineare sulla matrice jacobiana. In ingresso, è necessario definire un vettore che rappresenta il dato iniziale e due *functions*, `Ffun` e `Jfun`, che calcolano, rispettivamente, il vettore colonna `F`, contenente la valutazione di `f` su un generico vettore `x`, e la matrice jacobiana `Jf`, anch'essa valutata su un vettore `x`. Il metodo si arresta quando la

differenza in norma euclidea fra due iterate consecutive è minore di `tol` o quando viene raggiunto il massimo numero di iterazioni consentito `maxiter`.



Programma 3 - newtonsys : il metodo di Newton per un sistema non lineare

```
function [x,F,iter] = newtonsys(Ffun,Jfun,x0,tol,maxiter,varargin)
%NEWTONSYS calcola uno zero di un sistema non lineare
% [ZERO,F,ITER]=NEWTONSYS(FFUN,JFUN,X0,TOL,MAXITER) calcola il
% vettore ZERO, zero di un sistema non lineare definito dalla function FFUN
% con matrice jacobiana definita nella funciton JFUN a partire dal vettore X0.
iter = 0; err = tol + 1; x = x0;
while err > tol & iter <= maxiter
    J = feval(Jfun,x,varargin{:});
    F = feval(Ffun,x,varargin{:});
    delta = - J\F;
    x = x + delta;
    err = norm(delta);
    iter = iter + 1;
end
F = norm(feval(Ffun,x,varargin{:}));
if iter >= maxiter
    fprintf(' Il metodo non converge nel massimo numero di iterazioni\n ');
    fprintf(' L''ultima iterata calcolata ha residuo relativo pari a %e\n',F);
else
    fprintf(' Il metodo converge in %i iterazioni con un residuo pari a %e\n',iter,F);
end
return
```

Esempio 2.4 Consideriamo il sistema non lineare (2.15) che ammette le due soluzioni (individuabili, ad esempio, per via grafica) $(0.4761, -0.8794)$ e $(-0.4761, 0.8794)$ (riportiamo le sole prime 4 cifre significative). Per usare il Programma 3 definiamo le seguenti *functions*:

```
function J=Jfun(x)
pi2 = 0.5*pi;
J(1,1) = 2*x(1);
J(1,2) = 2*x(2);
J(2,1) = pi2*cos(pi2*x(1));
J(2,2) = 3*x(2)^2;
return

function F=Ffun(x)
F(1,1) = x(1)^2 + x(2)^2 - 1;
F(2,1) = sin(pi*x(1)/2) + x(2)^3;
return
```

Usando il Programma 3 nel modo seguente (il carattere speciale @ serve per segnalare a `newtonsys` che `Ffun` e `Jfun` sono delle *function*)

```
[x,F,iter] = newtonsys(@Ffun,@Jfun,x0,tol,maxiter);
```

con `x0=[1;1]`, troviamo che il metodo di Newton converge in 8 iterazioni al vettore

```
4.760958225338114e-01  
-8.793934089897496e-01
```

avendo scelto `tol=1.e-05` e `maxiter=10`. Si noti che per far convergere il metodo all'altra radice basta scegliere come dato iniziale `x0=[-1,-1]`. In generale, esattamente come nel caso scalare, la convergenza del metodo di Newton dipende dalla scelta del dato iniziale $\mathbf{x}^{(0)}$ ed in particolare bisogna garantire che $\det(\mathbf{J}_f(\mathbf{x}^{(0)})) \neq 0$.

Riassumendo



- Il calcolo degli zeri di una funzione f viene condotto attraverso metodi iterativi;
- il metodo di bisezione è un metodo elementare che consente di approssimare uno zero di una funzione “incapsulandolo” in intervalli la cui ampiezza viene dimezzata ad ogni iterazione. Esso converge sempre allo zero purché f sia continua nell’intervallo di partenza e cambi di segno agli estremi;
- il metodo di Newton è un metodo nel quale l’approssimazione dello zero viene condotta utilizzando i valori assunti da f e dalla sua derivata prima. Esso generalmente converge solo per valori del dato iniziale sufficientemente vicini allo zero cercato;
- quando converge, il metodo di Newton ha ordine 2 se lo zero è semplice, 1 se lo zero è multiplo;
- il metodo di Newton può essere esteso al caso del calcolo degli zeri di un sistema di equazioni non lineari.

Si vedano gli Esercizi 2.6-2.14.



2.3 Iterazioni di punto fisso

Con una calcolatrice si può facilmente verificare che applicando ripetutamente la funzione coseno partendo dal numero 1 si genera la seguente successione

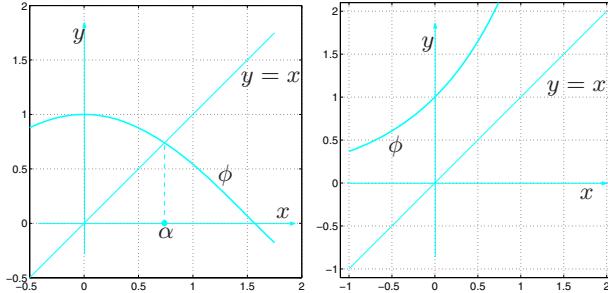


Figura 2.6. La funzione $\phi(x) = \cos x$ ammette un solo punto fisso (a sinistra), mentre la funzione $\phi(x) = e^x$ non ne ammette nessuno (a destra)

$$\begin{aligned} x^{(1)} &= \cos(1) = 0.54030230586814, \\ x^{(2)} &= \cos(x^{(1)}) = 0.85755321584639, \\ &\vdots \\ x^{(10)} &= \cos(x^{(9)}) = 0.74423735490056, \\ &\vdots \\ x^{(20)} &= \cos(x^{(19)}) = 0.73918439977149, \end{aligned}$$

che tende al valore $\alpha = 0.73908513\dots$. Essendo per costruzione $x^{(k+1)} = \cos(x^{(k)})$ per $k = 0, 1, \dots$ (con $x^{(0)} = 1$), α è tale che $\cos(\alpha) = \alpha$: per questa ragione esso viene detto un *punto fisso* della funzione coseno. L'interesse per un metodo che sfrutta iterazioni di questo tipo è evidente: se α è punto fisso per il coseno, allora esso è uno zero della funzione $f(x) = x - \cos(x)$ ed il metodo appena proposto è un metodo per il calcolo degli zeri di f . D'altra parte non tutte le funzioni ammettono punti fissi; ad esempio, se si ripete l'esperimento precedente con la funzione esponenziale a partire da $x^{(0)} = 1$, dopo solo 4 passi si giunge ad una situazione di *overflow* (si veda la Figura 2.6). Dobbiamo quindi precisare meglio questa idea intuitiva. Consideriamo pertanto il seguente problema: data una funzione $\phi : [a, b] \rightarrow \mathbb{R}$, trovare $\alpha \in [a, b]$ tale che

$$\alpha = \phi(\alpha).$$

Se un tale α esiste, viene detto un punto fisso di ϕ e lo si può determinare come limite della seguente successione

$$x^{(k+1)} = \phi(x^{(k)}), k \geq 0$$

(2.17)

dove $x^{(0)}$ è un dato iniziale. Questo algoritmo è detto delle *iterazioni di punto fisso* e ϕ ne è detta la *funzione di iterazione*. L'esempio introduttivo è dunque un algoritmo di iterazioni di punto fisso in cui $\phi(x) = \cos(x)$.

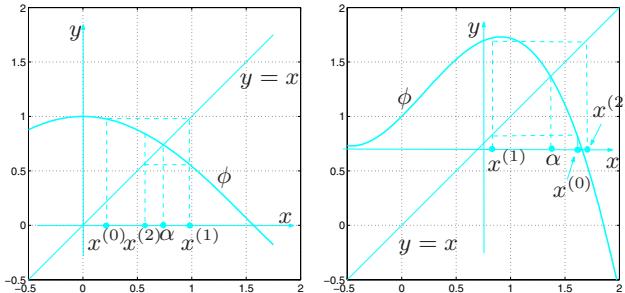


Figura 2.7. Rappresentazione delle prime iterazioni di punto fisso per due funzioni di iterazione. A sinistra, le iterazioni convergono verso il punto fisso α , a destra, invece, se ne allontanano

Un'interpretazione geometrica della (2.17) viene riportata nel grafico di sinistra di Figura 2.7. Si intuisce che, se ϕ è una funzione continua e se esiste il limite della successione $\{x^{(k)}\}$, allora tale limite è un punto fisso di ϕ . Preciseremo bene questo risultato nelle Proposizioni 2.1 e 2.2.

Esempio 2.5 Il metodo di Newton (2.8) può essere riletto come un algoritmo di iterazioni di punto fisso con funzione di iterazione

$$\phi(x) = x - \frac{f(x)}{f'(x)}. \quad (2.18)$$

Tale funzione verrà d'ora in poi indicata con il simbolo ϕ_N . Il metodo di bisezione non è invece un'iterazione di punto fisso, in quanto la generica iterata $x^{(k+1)}$ può non dipendere dalla sola $x^{(k)}$, ma anche da $x^{(k-1)}$.

Come mostrato dalla Figura 2.7 (a destra), non tutte le funzioni di iterazione garantiscono che l'iterazione di punto fisso converga. Vale infatti il seguente risultato:

Proposizione 2.1 Consideriamo la successione (2.17). Supponiamo che valgano le seguenti ipotesi:

1. $\phi(x) \in [a, b]$ per ogni $x \in [a, b]$;
2. ϕ è derivabile in $[a, b]$;
3. $\exists K < 1$ tale che $|\phi'(x)| \leq K$ per ogni $x \in [a, b]$.

In tal caso, ϕ ha un unico punto fisso $\alpha \in [a, b]$ e la successione definita nella (2.17) converge a α , qualunque sia la scelta del dato iniziale $x^{(0)}$ in $[a, b]$. Inoltre,

$$\lim_{k \rightarrow \infty} \frac{x^{(k+1)} - \alpha}{x^{(k)} - \alpha} = \phi'(\alpha). \quad (2.19)$$

Dalla (2.19) si deduce che le iterazioni di punto fisso convergono almeno *linearmente* cioè che, per k sufficientemente grande, l'errore al passo $k+1$ si comporta come l'errore al passo k moltiplicato per una costante $\phi'(\alpha)$ indipendente da k ed il cui valore assoluto è minore di 1.

Esempio 2.6 La funzione $\phi(x) = \cos(x)$ soddisfa le ipotesi della Proposizione 2.1 in quanto $|\phi'(\alpha)| = |\sin(\alpha)| \simeq 0.67 < 1$ e, di conseguenza per continuità, esiste un intorno I_α di α nel quale $|\phi'(x)| < 1$ per ogni $x \in I_\alpha$. La funzione $\phi(x) = x^2 - 1$, pur possedendo due punti fissi $\alpha_{\pm} = (1 \pm \sqrt{5})/2$, non verifica le ipotesi per nessuno dei due in quanto $|\phi'(\alpha_{\pm})| = |1 \pm \sqrt{5}| > 1$. La corrispondente iterazione di punto fisso non sarà pertanto convergente.

Esempio 2.7 (Dinamica di una popolazione) Applichiamo le iterazioni di punto fisso alla funzione $\phi_V(x) = rx/(1+xK)$ del modello di Verhulst (2.4) ed alla funzione $\phi_P(x) = rx^2/(1+(x/K)^2)$ del modello predatore/preda (2.5) scegliendo $r = 3$ e $K = 1$. Se partiamo dal dato iniziale $x^{(0)} = 1$ troviamo il punto fisso $\alpha = 2$ nel primo caso e $\alpha = 2.6180$ nel secondo (si veda la Figura 2.8). Il punto fisso $\alpha = 0$ può essere calcolato solo per il modello predatore/preda in quanto $|\phi'_V(\alpha)| = r > 1$, mentre $\phi'_P(\alpha) = 0$. Analogamente il punto fisso $\alpha = 0.3820\dots$ del modello predatore/preda non può essere calcolato in quanto $|\phi'_P(\alpha)| > 1$.

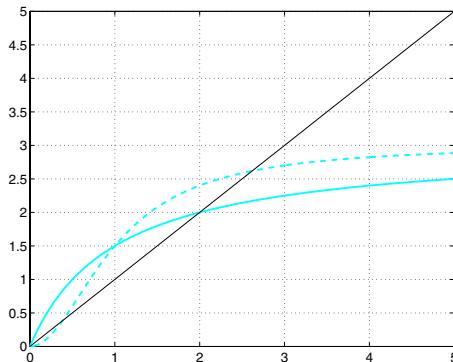


Figura 2.8. I punti fissi per due diversi modelli di dinamica delle popolazioni: il modello di Verhulst (in linea piena) e quello predatore/preda (in linea tratteggiata)

La convergenza quadratica non è prerogativa del solo metodo di Newton. In generale, vale infatti la seguente proprietà:

Proposizione 2.2 Si suppongano valide le ipotesi della Proposizione 2.1. Se, inoltre, ϕ è derivabile due volte e se

$$\phi'(\alpha) = 0, \phi''(\alpha) \neq 0,$$

allora il metodo di punto fisso (2.17) è convergente di ordine 2 e si ha

$$\lim_{k \rightarrow \infty} \frac{x^{(k+1)} - \alpha}{(x^{(k)} - \alpha)^2} = \frac{1}{2} \phi''(\alpha). \quad (2.20)$$

L'Esempio 2.5 mostra che le iterazioni di punto fisso (2.17) possono servire anche per il calcolo degli zeri di funzioni. Naturalmente, data una funzione f , la ϕ definita in (2.18) non è l'unica funzione di iterazione possibile. Ad esempio, per la soluzione dell'equazione $\log(x) = \gamma$, posto $f(x) = \log(x) - \gamma$, la scelta (2.18) condurrebbe alla funzione di iterazione

$$\phi_N(x) = x(1 - \log(x) + \gamma).$$

Un altro metodo di punto fisso si trova sommando x ad ambo i membri dell'equazione $f(x) = 0$. La funzione di iterazione associata è ora $\phi_1(x) = x + \log(x) - \gamma$. Un terzo metodo può essere infine ricavato moltiplicando per x l'equazione e scegliendo $\phi_2(x) = x \log(x)/\gamma$. Non tutti questi metodi sono convergenti; ad esempio, se $\gamma = -2$, i metodi con funzioni di iterazione ϕ_N e ϕ_2 sono entrambi convergenti, mentre quello con funzione ϕ_1 non lo è in quanto $|\phi'_1(x)| > 1$ in un intorno del punto fisso.

2.3.1 Come arrestare un'iterazione di punto fisso

In generale, le iterazioni di punto fisso verranno arrestate quando il valore assoluto della *differenza fra due iterate* è minore di una tolleranza ε fissata.

Essendo $\alpha = \phi(\alpha)$ e $x^{(k+1)} = \phi(x^{(k)})$, usando il teorema del valor medio (introdotto nel paragrafo 1.4.3) troviamo

$$\alpha - x^{(k+1)} = \phi(\alpha) - \phi(x^{(k)}) = \phi'(\xi^{(k)}) (\alpha - x^{(k)}) \text{ con } \xi^{(k)} \in I_{\alpha, x^{(k)}},$$

essendo $I_{\alpha, x^{(k)}}$ l'intervallo di estremi α e $x^{(k)}$. Usando l'identità

$$\alpha - x^{(k)} = (\alpha - x^{(k+1)}) + (x^{(k+1)} - x^{(k)}),$$

concludiamo che

$$\alpha - x^{(k)} = \frac{1}{1 - \phi'(\xi^{(k)})} (x^{(k+1)} - x^{(k)}). \quad (2.21)$$

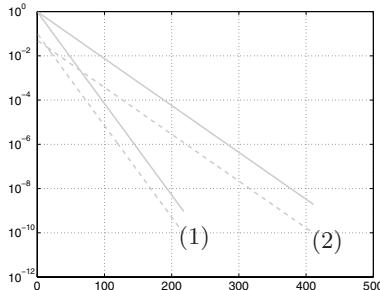


Figura 2.9. Logaritmo dei valori assoluti degli errori (in linea continua) e logaritmo del valore assoluto della differenza fra iterate (in linea tratteggiata) in funzione del numero di iterazioni per il caso dell’Esempio 2.8: le curve (1) si riferiscono a $m = 11$, mentre le (2) a $m = 21$

Di conseguenza, se $\phi'(x) \simeq 0$ in un intorno di α , l’errore viene stimato accuratamente dalla differenza fra due iterate consecutive. Questo accade per tutti i metodi di ordine 2 e quindi, in particolare, per il metodo di Newton. In caso contrario, tanto più ϕ' è prossimo a 1, tanto più stimare l’errore con la differenza fra le iterate sarà insoddisfacente.

Esempio 2.8 Calcoliamo con il metodo di Newton lo zero $\alpha = 1$ della funzione $f(x) = (x - 1)^{m-1} \log(x)$ per $m = 11$ e $m = 21$. Questo zero ha molteplicità pari a m . In tal caso l’ordine di convergenza del metodo di Newton decade a 1; inoltre, si può provare (si veda l’Esercizio 2.15) che $\phi'_N(\alpha) = 1 - 1/m$, essendo ϕ_N la funzione di iterazione del metodo stesso, visto come iterazione di punto fisso. Quindi, al crescere di m , la stima dell’errore fornita dalla differenza fra le iterate diventa sempre meno affidabile. È quello che si verifica sperimentalmente: nei grafici della Figura 2.9 vengono paragonati gli errori e la differenza fra le iterate in valore assoluto per $m = 11$ e $m = 21$. Come si vede lo scarto fra le due quantità è maggiore per $m = 21$.

2.4 Accelerazione con il metodo di Aitken

In questo paragrafo illustriamo una tecnica che consente di accelerare la convergenza di una successione ottenuta a partire da iterazioni di punto fisso. Supponiamo pertanto che $x^{(k)} = \phi(x^{(k-1)})$, $k \geq 1$. Se la successione $\{x^{(k)}\}$ converge *linearmente* ad un punto fisso α di ϕ , dalla (2.19) si ricava che, per k fissato, dovrà esistere un valore λ (da determinare) tale che

$$\phi(x^{(k)}) - \alpha = \lambda(x^{(k)} - \alpha), \quad (2.22)$$

dove volutamente non abbiamo identificato $\phi(x^{(k)})$ con $x^{(k+1)}$. L’idea del metodo di Aitken consiste infatti nel definire un nuovo valore per $x^{(k+1)}$

(e, di conseguenza, una nuova successione) che sia un'approssimazione di α migliore di quella data da $\phi(x^{(k)})$. In effetti, dalla (2.22) ricaviamo che

$$\alpha = \frac{\phi(x^{(k)}) - \lambda x^{(k)}}{1 - \lambda} = \frac{\phi(x^{(k)}) - \lambda x^{(k)} + x^{(k)} - x^{(k)}}{1 - \lambda}$$

ovvero

$$\alpha = x^{(k)} + (\phi(x^{(k)}) - x^{(k)})/(1 - \lambda) \quad (2.23)$$

Si tratta a questo punto di calcolare λ . Per fare questo introduciamo la seguente successione

$$\lambda^{(k)} = \frac{\phi(\phi(x^{(k)})) - \phi(x^{(k)})}{\phi(x^{(k)}) - x^{(k)}} \quad (2.24)$$

e verifichiamo che vale la seguente proprietà:

Lemma 2.1 Se la successione di elementi $x^{(k+1)} = \phi(x^{(k)})$ converge a α , allora $\lim_{k \rightarrow \infty} \lambda^{(k)} = \phi'(\alpha)$.

Dimostrazione 2.1 Se $x^{(k+1)} = \phi(x^{(k)})$, allora $x^{(k+2)} = \phi(\phi(x^{(k)}))$ e quindi, dalla (2.24), si ricava che $\lambda^{(k)} = (x^{(k+2)} - x^{(k+1)})/(x^{(k+1)} - x^{(k)})$ ovvero

$$\lambda^{(k)} = \frac{x^{(k+2)} - \alpha - (x^{(k+1)} - \alpha)}{x^{(k+1)} - \alpha - (x^{(k)} - \alpha)} = \frac{\frac{x^{(k+2)} - \alpha}{x^{(k+1)} - \alpha} - 1}{1 - \frac{x^{(k)} - \alpha}{x^{(k+1)} - \alpha}}$$

da cui, passando al limite e ricordando la (2.19), si perviene alla tesi, ovvero

$$\lim_{k \rightarrow \infty} \lambda^{(k)} = \frac{\phi'(\alpha) - 1}{1 - 1/\phi'(\alpha)} = \phi'(\alpha).$$

Grazie al Lemma 2.1 possiamo concludere che, per k fissato, $\lambda^{(k)}$ può essere considerato come un'approssimazione del valore incognito λ , introdotto in precedenza. Utilizziamo allora la (2.24) nella (2.23) e definiamo un nuovo $x^{(k+1)}$ nel modo seguente

$$x^{(k+1)} = x^{(k)} - \frac{(\phi(x^{(k)}) - x^{(k)})^2}{\phi(\phi(x^{(k)})) - 2\phi(x^{(k)}) + x^{(k)}}, \quad k \geq 0 \quad (2.25)$$

Questa espressione è nota come *formula di estrappolazione di Aitken* e, per la (2.25) può essere considerata come *nuova iterazione di punto fisso* in cui si ponga come funzione di iterazione

$$\phi_{\Delta}(x) = \frac{x\phi(\phi(x)) - [\phi(x)]^2}{\phi(\phi(x)) - 2\phi(x) + x}$$

(tale metodo è noto talvolta anche con il nome di metodo di Steffensen).

Evidentemente la funzione ϕ_{Δ} è indeterminata per $x = \alpha$ in quanto tanto il numeratore che il denominatore si annullano. Tuttavia, assumendo che ϕ sia derivabile con $\phi'(\alpha) \neq 1$ ed applicando la formula di de l'Hôpital si trova

$$\begin{aligned}\lim_{x \rightarrow \alpha} \phi_{\Delta}(x) &= \frac{\phi(\phi(\alpha)) + \alpha\phi'(\phi(\alpha))\phi'(\alpha) - 2\phi(\alpha)\phi'(\alpha)}{\phi'(\phi(\alpha))\phi'(\alpha) - 2\phi'(\alpha) + 1} \\ &= \frac{\alpha + \alpha[\phi'(\alpha)]^2 - 2\alpha\phi'(\alpha)}{[\phi'(\alpha)]^2 - 2\phi'(\alpha) + 1} = \alpha.\end{aligned}$$

Di conseguenza, $\phi_{\Delta}(x)$ può essere estesa per continuità in $x = \alpha$ con $\phi_{\Delta}(\alpha) = \alpha$. Quando $\phi(x) = x - f(x)$ il caso $\phi'(\alpha) = 1$ corrisponde ad una radice di molteplicità almeno 2 per f (in quanto $\phi'(\alpha) = 1 - f'(\alpha)$). Anche in questa situazione si può però dimostrare, passando al limite, che $\phi_{\Delta}(\alpha) = \alpha$. Infine, si può anche verificare che i punti fissi di ϕ_{Δ} sono tutti e soli i punti fissi di ϕ .

Il metodo di Aitken può essere quindi applicato ad un metodo di punto fisso qualsiasi. Vale infatti il seguente teorema:

Teorema 2.1 Siano $x^{(k+1)} = \phi(x^{(k)})$ delle iterazioni di punto fisso, con $\phi(x) = x - f(x)$, per l'approssimazione delle radici di f . Allora, se f è sufficientemente regolare abbiamo che:

- se le iterazioni di punto fisso convergono linearmente ad una radice semplice di f , allora il metodo di Aitken converge quadraticamente alla stessa radice;
- se le iterazioni di punto fisso convergono con ordine $p \geq 2$ ad una radice semplice di f , allora il metodo di Aitken converge alla stessa radice con ordine $2p - 1$;
- se le iterazioni di punto fisso convergono linearmente ad una radice di molteplicità $m \geq 2$ di f , allora il metodo di Aitken converge linearmente alla stessa radice con un fattore di convergenza asintotico $C = 1 - 1/m$.

In particolare, se $p = 1$ e la radice di f è semplice il metodo di estrappolazione di Aitken converge anche se le corrispondenti iterazioni di punto fisso divergono.

Nel Programma 4 riportiamo un'implementazione del metodo di Aitken. In esso `phi` è una *function* (o una *inline function*) che precisa l'espressione della funzione di iterazione del metodo di punto fisso cui

viene applicata la tecnica di estrapolazione di Aitken. Il dato iniziale viene precisato nella variabile `x0`, mentre `tol` e `kmax` sono rispettivamente la tolleranza sul criterio d'arresto (sul valore assoluto della differenza fra due iterate consecutive) ed il numero massimo di iterazioni consentite. Se non precisati, vengono assunti i valori di `default` pari a `kmax=100` e `tol=1.e-04`.

Programma 4 - `aitken` : il metodo di Aitken

```
function [x,niter]=aitken(phi,x0,tol,kmax,varargin)
%AITKEN Estrapolazione di Aitken
% [ALPHA,NITER]=AITKEN(PHI,X0) calcola un'approssimazione di un
% punto fisso ALPHA della funzione PHI a partire dal dato iniziale X0 con il
% metodo di estrapolazione di Aitken. Il metodo si arresta dopo 100
% iterazioni o dopo che il valore assoluto della differenza fra due iterate
% consecutive e' minore di 1.e-04. PHI deve essere precisata come function o
% come inline function.
% [ALPHA,NITER]=AITKEN(PHI,X0,TOL,KMAX) consente di definire la
% tolleranza sul criterio d'arresto ed il numero massimo di iterazioni.
if nargin == 2
    tol = 1.e-04; kmax = 100;
elseif nargin == 3
    kmax = 100;
end
x = x0;
diff = tol + 1;
niter = 0;
while niter <= kmax & diff >= tol
    gx = feval(phi,x,varargin{:});
    ggx = feval(phi,gx,varargin{:});
    xnew = (x*ggx-gx^2)/(ggx-2*gx+x); diff = abs(x-xnew); x = xnew;
    niter = niter + 1;
end
return
```

Esempio 2.9 Per il calcolo della radice semplice $\alpha = 1$ della funzione $f(x) = e^x(x-1)$ applichiamo il metodo di Aitken a partire dalle due seguenti funzioni di iterazione

$$\phi_0(x) = \log(xe^x), \quad \phi_1(x) = \frac{e^x + x}{e^x + 1}.$$

Utilizziamo il Programma 4 con `tol=1.e-10`, `kmax=100`, `x0=2` e definiamo le due funzioni di iterazione come segue

```
>> phi0 = inline('log(x*exp(x))','x');
>> phi1 = inline('((exp(x)+x)/(exp(x)+1))','x');
```

A questo punto eseguiamo il Programma 4 nel modo seguente



```

>> [alpha,niter]=aitken(phi0,x0,tol,kmax)
alpha =
    1.0000 + 0.0000i
niter =
    10
>> [alpha,niter]=aitken(phi1,x0,tol,kmax)
alpha =
    1
niter =
    4

```

Come si vede la convergenza del metodo è estremamente rapida (per confronto il metodo di punto fisso con funzione di iterazione ϕ_1 e con lo stesso criterio d'arresto avrebbe richiesto 18 iterazioni, mentre il metodo corrispondente a ϕ_0 non sarebbe stato convergente in quanto $|\phi'_0(1)| = 2$).



Riassumendo

1. Un valore α tale che $\phi(\alpha) = \alpha$ si dice punto fisso della funzione ϕ . Per il suo calcolo si usano metodi iterativi della forma $x^{(k+1)} = \phi(x^{(k)})$, che vengono detti iterazioni di punto fisso;
2. le iterazioni di punto fisso convergono sotto precise condizioni su ϕ e sulla sua derivata prima. Tipicamente la convergenza è lineare, diventa quadratica qualora $\phi'(\alpha) = 0$;
3. è possibile utilizzare le iterazioni di punto fisso anche per il calcolo degli zeri di una funzione f ;
4. data un'iterazione di punto fisso $x^{(k+1)} = \phi(x^{(k)})$, anche non convergente, è sempre possibile costruire una nuova iterazione di punto fisso convergente tramite il metodo di Aitken.



Si vedano gli Esercizi 2.15-2.18.

2.5 Polinomi algebrici

In questo paragrafo consideriamo il caso in cui f sia un polinomio di grado $n \geq 0$ cioè della forma

$$p_n(x) = \sum_{k=0}^n a_k x^k, \quad (2.26)$$

dove gli $a_k \in \mathbb{R}$ sono coefficienti assegnati. Come già osservato, lo spazio di tutti i polinomi di grado al più n della forma (2.26) viene indicato con il simbolo \mathbb{P}_n . Si noti che, essendo i coefficienti a_k reali, se $\alpha \in \mathbb{C}$

con $\operatorname{Im}(\alpha) \neq 0$ è una radice di p_n , allora lo è anche la sua complessa coniugata $\bar{\alpha}$.

Il teorema di Abel assicura che per ogni $n \geq 5$ non esiste una forma esplicita per calcolare tutti gli zeri di un generico polinomio p_n . Questo fatto motiva ulteriormente l'uso di metodi numerici per il calcolo delle radici di p_n .

Come abbiamo visto in precedenza per tali metodi è importante la scelta di un buon dato iniziale $x^{(0)}$ o di un opportuno intervallo di ricerca $[a, b]$ per la radice. Nel caso dei polinomi ciò è talvolta possibile sulla base dei seguenti risultati.

Teorema 2.2 (Regola dei segni di Cartesio) *Sia $p_n \in \mathbb{P}_n$. Indichiamo con ν il numero di variazioni di segno nell'insieme dei coefficienti $\{a_j\}$ e con k il numero di radici reali positive di p_n ciascuna contata con la propria molteplicità. Si ha allora che $k \leq \nu$ e $\nu - k$ è pari.*

Esempio 2.10 Il polinomio $p_6(x) = x^6 - 2x^5 + 5x^4 - 6x^3 + 2x^2 + 8x - 8$ ha come zeri $\{\pm 1, \pm 2i, 1 \pm i\}$ e quindi ammette 1 radice reale positiva ($k = 1$). In effetti, il numero di variazioni di segno ν dei coefficienti è 5 e quindi $k \leq \nu$ e $\nu - k = 4$ è pari.

Teorema 2.3 (di Cauchy) *Tutti gli zeri di p_n sono inclusi nel cerchio Γ del piano complesso*

$$\Gamma = \{z \in \mathbb{C} : |z| \leq 1 + \eta\}, \text{ dove } \eta = \max_{0 \leq k \leq n-1} |a_k/a_n|. \quad (2.27)$$

Questa proprietà è di scarsa utilità quando $\eta \gg 1$ (per il polinomio p_6 dell'Esempio 2.10 si ha ad esempio $\eta = 8$, mentre le radici stanno tutte all'interno di cerchi di raggio decisamente minore).

2.5.1 Il metodo di Hörner

Illustriamo in questo paragrafo un metodo per la valutazione efficiente di un polinomio (e della sua derivata) in un punto assegnato z . Tale algoritmo consente di generare un procedimento automatico, detto metodo di *deflazione*, per l'approssimazione progressiva di *tutte* le radici di un polinomio. Da un punto di vista algebrico la (2.26) è equivalente alla seguente rappresentazione

$$p_n(x) = a_0 + x(a_1 + x(a_2 + \dots + x(a_{n-1} + a_n x) \dots)). \quad (2.28)$$

Tuttavia, mentre la (2.26) richiede n addizioni e $2n - 1$ moltiplicazioni per valutare $p_n(x)$ (per x fissato), la (2.28) richiede solo n addizioni più n moltiplicazioni. L'espressione (2.28), nota anche come algoritmo delle moltiplicazioni annidate, sta alla base del metodo di Horner. Quest'ultimo consente la valutazione efficiente del polinomio p_n in un punto z mediante il seguente algoritmo di *divisione sintetica*:

$$\begin{aligned} b_n &= a_n, \\ b_k &= a_k + b_{k+1}z, \quad k = n-1, n-2, \dots, 0 \end{aligned} \quad (2.29)$$

Nella (2.29) tutti i coefficienti b_k con $k \leq n-1$ dipendono da z e possiamo verificare che $b_0 = p_n(z)$. Il polinomio

$$q_{n-1}(x; z) = b_1 + b_2x + \dots + b_nx^{n-1} = \sum_{k=1}^n b_kx^{k-1}, \quad (2.30)$$

di grado pari a $n-1$ nella variabile x , dipende dal parametro z (attraverso i coefficienti b_k) e si dice il *polinomio associato* a p_n . L'algoritmo (2.29) è stato implementato nel Programma 5. I coefficienti a_j del polinomio da valutare sono memorizzati nel vettore **a** a partire da a_n fino ad a_0 .



Programma 5 - horner : il metodo di divisione sintetica

```
function [y,b] = horner(a,z)
%HORNER Metodo di Horner
% Y=HORNER(A,Z) calcola
%   Y = A(1)*Z^N + A(2)*Z^(N-1) + ... + A(N)*Z + A(N+1)
%   con il metodo di divisione sintetica di Horner.
n = length(a)-1;
b = zeros(n+1,1);
b(1) = a(1);
for j=2:n+1
    b(j) = a(j)+b(j-1)*z;
end
y = b(n+1);
b = b(1:end-1);
return
```

Vogliamo a questo punto introdurre un algoritmo efficiente che, nota una radice di un polinomo (od una sua approssimazione), sia in grado di eliminarla e consentire quindi il calcolo della successiva fino all'esaurimento di tutte le radici.

A questo proposito conviene ricordare la seguente proprietà sulla *divisione tra polinomi*:

Proposizione 2.3 *Dati due polinomi $h_n \in \mathbb{P}_n$ e $g_m \in \mathbb{P}_m$ con $m \leq n$, esistono un unico polinomio $\delta \in \mathbb{P}_{n-m}$ ed un unico polinomio $\rho \in \mathbb{P}_{m-1}$ tali che*

$$h_n(x) = g_m(x)\delta(x) + \rho(x). \quad (2.31)$$

Dividendo allora un polinomio $p_n \in \mathbb{P}_n$ per $x - z$, grazie alla (2.31) si deduce che

$$p_n(x) = b_0 + (x - z)q_{n-1}(x; z),$$

avendo indicato con q_{n-1} il quoziente e con b_0 il resto della divisione. Se z è una radice di p_n , allora si ha $b_0 = p_n(z) = 0$ e quindi $p_n(x) = (x - z)q_{n-1}(x; z)$. In tal caso l'equazione algebrica $q_{n-1}(x; z) = 0$ fornisce le $n - 1$ radici restanti di $p_n(x)$. Questa osservazione suggerisce di adottare il seguente procedimento, detto di *deflazione*, per il calcolo di tutte le radici di p_n .

Per $m = n, n - 1, \dots, 1$, con passo -1 :

1. si trova una radice r_m di p_m con un opportuno metodo di approssimazione;
2. si calcola $q_{m-1}(x; r_m)$ tramite le (2.29)-(2.30) (posto $z = r_m$);
3. si pone $p_{m-1} = q_{m-1}$.

Nel paragrafo che segue proponiamo il metodo più noto di questa famiglia, che utilizza per l'approssimazione delle radici il metodo di Newton.

2.5.2 Il metodo di Newton-Hörner

Come suggerisce il nome, il metodo di Newton-Hörner implementa il procedimento di deflazione appena descritto facendo uso del metodo di Newton per il calcolo delle radici r_m . Il vantaggio risiede nel fatto che l'implementazione del metodo di Newton sfrutta convenientemente l'algoritmo di Hörner (2.29). Infatti, se q_{n-1} è il polinomio associato a p_n definito nella (2.30), poiché

$$p'_n(x) = q_{n-1}(x; z) + (x - z)q'_{n-1}(x; z),$$

si ha

$$p'_n(z) = q_{n-1}(z; z).$$

Grazie a questa identità il metodo di Newton-Hörner per l'approssimazione di una radice (reale o complessa) r_j di p_n ($j = 1, \dots, n$) prende la forma seguente:

data una stima iniziale $r_j^{(0)}$ della radice, calcolare per ogni $k \geq 0$ fino a convergenza

$$r_j^{(k+1)} = r_j^{(k)} - \frac{p_n(r_j^{(k)})}{p'_n(r_j^{(k)})} = r_j^{(k)} - \frac{p_n(r_j^{(k)})}{q_{n-1}(r_j^{(k)}; r_j^{(k)})} \quad (2.32)$$

A questo punto si utilizza la tecnica di deflazione, sfruttando il fatto che $p_n(x) = (x - r_j)p_{n-1}(x)$. Si può quindi passare all'approssimazione di uno zero di p_{n-1} e così via sino all'esaurimento di tutte le radici di p_n .

Si tenga conto che quando $r_j \in \mathbb{C}$ è necessario condurre i calcoli in aritmetica complessa, prendendo $r_j^{(0)}$ con parte immaginaria non nulla. In caso contrario, infatti, il metodo di Newton-Hörner genererebbe una successione $\{r_j^{(k)}\}$ di numeri reali.

Il metodo di Newton-Hörner è stato implementato nel Programma 6. I coefficienti a_j del polinomio del quale si intendono calcolare le radici sono memorizzati nel vettore **a** a partire da a_n fino ad a_0 . Gli altri parametri di input, **tol** e **kmax**, sono rispettivamente la tolleranza sul criterio d'arresto (sul valore assoluto della differenza fra due iterate consecutive) ed il numero massimo di iterazioni consentite. Se non diversamente precisati, vengono assunti i valori di *default* pari a **kmax=100** e **tol=1.e-04**. In output, il programma restituisce nei vettori **radici** e **iter** le radici calcolate ed il numero di iterazioni che è stato effettuato per calcolare ciascun valore.



Programma 6 - newtonhorner : il metodo di Newton-Hörner

```
function [radici,iter]=newtonhorner(a,x0,tol,kmax)
%NEWTONHORNER Metodo di Newton-Horner
% [RADICI,ITER]=NEWTONHORNER(A,X0) calcola le radici del polinomio
% P(X) = A(1)*X^N + A(2)*X^(N-1) + ... + A(N)*X + A(N+1)
% con il metodo di Newton-Horner a partire dal dato iniziale X0. Il metodo
% si arresta per ogni radice dopo 100 iterazioni o dopo che il valore assoluto
% della differenza fra due iterate consecutive e' minore di 1.e-04.
% [RADICI,ITER]=NEWTONHORNER(A,X0,TOL,KMAX) consente di
% definire la tolleranza sul criterio d'arresto ed il numero massimo di iterazioni.
if nargin == 2
    tol = 1.e-04; kmax = 100;
elseif nargin == 3
    kmax = 100;
end
n=length(a)-1; radici = zeros(n,1); iter = zeros(n,1);
for k = 1:n
    % Iterazioni di Newton
    niter = 0; x = x0; diff = tol + 1;
    while niter <= kmax & diff >= tol
        [pz,b] = horner(a,x); [dpz,b] = horner(b,x);
```

```

xnew = x - pz/dpz;      diff = abs(xnew-x);
niter = niter + 1;      x = xnew;
end
% Deflazione
[pz,a] = horner(a,x); radici(k) = x; iter(k) = niter;
end
return

```

Osservazione 2.1 Onde minimizzare la propagazione degli errori di arrotondamento, nel processo di deflazione conviene approssimare per prima la radice r_1 di modulo minimo e procedere quindi al calcolo delle successive radici r_2, r_3, \dots , sino a quella di modulo massimo (per approfondimenti si veda ad esempio [QSS04]).

Esempio 2.11 Richiamiamo il Programma 6 per calcolare le radici $\{1, 2, 3\}$ del polinomio $p_3(x) = x^3 - 6x^2 + 11x - 6$. Usiamo le seguenti istruzioni

```

>> a=[1 -6 11 -6]; [x,niter]=newtonhorner(a,0,1.e-15,100)
x =
    1
    2
    3
niter =
    8
    8
    2

```

Come si vede il metodo calcola accuratamente ed in poche iterazioni tutte e tre le radici. Come notato nell’Osservazione 2.1 non sempre il metodo è però così efficiente.

Ad esempio, per il calcolo delle radici del polinomio $p_4(x) = x^4 - 7x^3 + 15x^2 - 13x + 4$ (che presenta una radice pari a 1 con molteplicità 3 ed una radice semplice pari a 3) si trovano i seguenti risultati

```

>> a=[1 -7 15 -13 4]; format long; [x,niter]=newtonhorner(a,0,1.e-15,100)
x =
    1.00000693533737
    0.99998524147571
    1.00000782324144
    3.99999999994548
niter =
    61
    101
    6
    2

```

dai quali risulta un evidente deterioramento nell’accuratezza del calcolo della radice multipla. In effetti si può dimostrare che questa perdita di accuratezza è tanto maggiore quanto più grande è la molteplicità della radice (si veda [QSS04]).

2.6 Cosa non vi abbiamo detto

I metodi più complessi per il calcolo accurato degli zeri di una generica funzione si ottengono combinando fra loro diversi algoritmi. Segnaliamo **fzero** a questo proposito il comando **fzero** (già introdotto nel paragrafo 1.4.1) che, nella sua forma più semplice **fzero(fun,x0)**, a partire da un dato **x0** calcola lo zero di una funzione **fun** (data come stringa o come *inline function*) più vicino a **x0**. Questa *function* è basata sul metodo di Dekker-Brent (si veda [QSS04, Capitolo 6]).

Ad esempio, risolviamo il problema dell'Esempio 2.1 anche con **fzero**, prendendo come valore iniziale **x0=0.3** (lo stesso che abbiamo scelto quando abbiamo usato il metodo di Newton). È sufficiente dare le seguenti istruzioni

```
>> f=inline('6000 - 1000*(1+l)/l*((1+l)^5 - 1)', 'l'); x0=0.3;
>> [alpha,res,flag,iter]=fzero(f,x0);
```

per trovare **alpha=0.06140241153653** e residuo pari a **res=9.0949e-13** dopo **iter=29** iterazioni. Si noti che quando il parametro di uscita **flag** assume un valore negativo significa che **fzero** ha fallito nella ricerca dello zero. Per confronto, osserviamo che il metodo di Newton converge in 6 iterazioni al valore **0.06140241153652** con un residuo pari a **2.3646e-11**, richiedendo tuttavia anche la conoscenza della derivata prima di *f*.

Per il calcolo degli zeri di un polinomio oltre al metodo di Newton-Hörner citiamo i metodi basati sulle successioni di Sturm, il metodo di Müller, (si vedano [Atk89], [Com95] o [QSS04]) ed il metodo di Bairstow ([RR85], pag.371 e seguenti). Un altro approccio consiste nel caratterizzare gli zeri di un polinomio come gli autovalori di una particolare matrice, detta *companion*, e nel calcolare quest'ultimi con tecniche opportune. Questo è l'approccio adottato dalla funzione MATLAB **roots**, introdotta nel paragrafo 1.4.2.

Nel paragrafo 2.2.2 abbiamo mostrato come si possa adattare il metodo di Newton al caso di sistemi di equazioni non lineari. In generale, ogni iterazione di punto fisso può essere facilmente estesa al calcolo delle radici di un sistema di equazioni non lineari. Per questo tipo di problemi ricordiamo inoltre anche il metodo di Broyden ed i metodi quasi-Newton che possono essere visti come una generalizzazione del metodo di Newton (si veda [DS83]).

fsolve

L'istruzione MATLAB ,

```
zero = fsolve('fun', x0)
```

permette di calcolare uno zero di un sistema non lineare definito attraverso la *function* **fun** (costruita dall'utilizzatore) e partendo da un vettore iniziale **x0**. La *function* **fun** restituisce i valori $f_i(\bar{x}_1, \bar{x}_2, \dots, \bar{x}_n)$, $i = 1, \dots, n$, per ogni vettore in ingresso $(\bar{x}_1, \bar{x}_2, \dots, \bar{x}_n)^T$.

Ad esempio, per il sistema non lineare (2.15) la corrispondente *function* MATLAB (da richiamare in `fsolve`) è

```
function fx=systemnl(x)
fx(1) = x(1)^2+x(2)^2-1;
fx(2) = sin(pi*0.5*x(1))+x(2)^3;
return
```

Le istruzioni MATLAB per risolvere il sistema dato sono allora

```
>> x0 = [1 1];
>> alpha=fsolve('systemnl',x0)
alpha =
    0.4761 -0.8794
```

Usando questa procedura abbiamo trovato solo una delle 2 radici. L'altra può essere calcolata usando come dato iniziale $-x_0$.

2.7 Esercizi

Esercizio 2.1 Data la funzione $f(x) = \cosh x + \cos x - \gamma$, per $\gamma = 1, 2, 3$ si individui un intervallo contenente uno zero (se esistente) e lo si calcoli con il metodo di bisezione con una accuratezza pari a 10^{-10} .

Esercizio 2.2 (Equazione di stato di un gas) Per l'anidride carbonica (CO_2) i coefficienti a e b della (2.1) valgono rispettivamente $a = 0.401\text{Pa m}^6$ e $b = 42.7 \cdot 10^{-6}\text{m}^3$ (Pa sta per Pascal). Si trovi il volume occupato da 1000 molecole di anidride carbonica poste ad una temperatura $T = 300\text{K}$ e ad una pressione $p = 3.5 \cdot 10^7\text{Pa}$ utilizzando il metodo di bisezione con una accuratezza pari a 10^{-12} (la costante di Boltzmann è pari a $k = 1.3806503 \cdot 10^{-23}\text{Joule K}^{-1}$).

Esercizio 2.3 Si consideri un piano la cui inclinazione varia con velocità costante ω . Su di esso si trova un oggetto che all'istante iniziale è fermo; dopo t secondi la sua posizione è

$$s(t, \omega) = \frac{g}{2\omega^2} [\sinh(\omega t) - \sin(\omega t)],$$

dove $g = 9.8\text{m/s}^2$ è l'accelerazione di gravità. Supponiamo che il corpo si sia mosso di un metro in un secondo; si ricavi il corrispondente valore di ω con una accuratezza dell'ordine di 10^{-5} .

Esercizio 2.4 Si ricavi la diseguaglianza (2.7).

Esercizio 2.5 Nel Programma 1 per calcolare il punto medio è stata utilizzata la seguente istruzione: $x(2) = x(1)+(x(3)-x(1))*0.5$, invece della più naturale: $x(2) = (x(1)+x(3))*0.5$. Per quale motivo?

Esercizio 2.6 Si ripeta per il metodo di Newton l’Esercizio 2.1. Perché per $\gamma = 2$ il metodo risulta inaccurato?

Esercizio 2.7 Utilizzando il metodo di Newton si costruisca un algoritmo per il calcolo della radice quadrata di un numero positivo a . Si proceda in modo analogo per il calcolo della radice cubica di a .

Esercizio 2.8 Supponendo il metodo di Newton convergente, si dimostri la (2.10) con α radice semplice di $f(x) = 0$ e f derivabile due volte con continuità in un intorno di α .

Esercizio 2.9 (Statica) Si risolva il Problema 2.3, al variare di $\beta \in [0, \pi]$, supponendo che le aste abbiano le seguenti lunghezze $a_1 = 10$ cm, $a_2 = 13$ cm, $a_3 = 8$ cm, $a_4 = 10$ cm, usando il metodo di Newton e richiedendo una tolleranza pari a 10^{-5} . Per ogni valore di β si considerino due possibili valori iniziali pari rispettivamente a -0.1 e a $2\pi/3$.

Esercizio 2.10 Si osservi che la funzione $f(x) = e^x - 2x^2$ ha 3 zeri, $\alpha_1 < 0$ e α_2 e α_3 positivi. Per quali valori di $x^{(0)}$ il metodo di Newton converge a α_1 ?

Esercizio 2.11 Si applichi il metodo di Newton per il calcolo dello zero di $f(x) = x^3 - 3x^22^{-x} + 3x4^{-x} - 8^{-x}$ in $[0, 1]$ e si analizzi sperimentalmente l’ordine di convergenza. La convergenza non risulta di ordine 2. Perché?

Esercizio 2.12 Un proiettile che viene lanciato ad una velocità v_0 con una inclinazione α in un tunnel di altezza h , raggiunge la massima gittata quando α è tale che $\sin(\alpha) = \sqrt{2gh/v_0^2}$, dove $g = 9.8\text{m/s}^2$ è l’accelerazione di gravità. Si calcoli α con il metodo di Newton, quando $v_0 = 10\text{m/s}$ e $h = 1\text{m}$.

Esercizio 2.13 (Piano di investimento) Si risolva, a meno di una tolleranza `tol=1.e-12`, il Problema 2.1 con il metodo di Newton, supponendo che $M = 6000$ euro, $v = 1000$ euro, $n = 5$ ed utilizzando un dato iniziale pari al risultato ottenuto dopo cinque iterazioni del metodo di bisezione sull’intervallo $(0.01, 0.1)$.

Esercizio 2.14 Un corridoio ha la pianta indicata in Figura 2.10. La lunghezza massima L di un’asta che possa passare da un estremo all’altro strisciando per terra è data da

$$L = l_2 / (\sin(\pi - \gamma - \alpha)) + l_1 / \sin(\alpha),$$

dove α è la soluzione della seguente equazione non lineare

$$l_2 \frac{\cos(\pi - \gamma - \alpha)}{\sin^2(\pi - \gamma - \alpha)} - l_1 \frac{\cos(\alpha)}{\sin^2(\alpha)} = 0. \quad (2.33)$$

Si determini con il metodo di Newton α quando $l_2 = 10$, $l_1 = 8$ e $\gamma = 3\pi/5$.

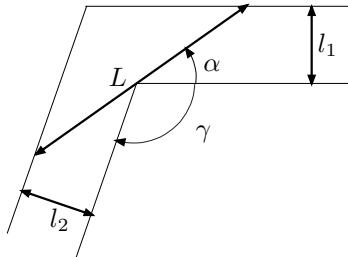


Figura 2.10. Problema dello scorrimento di un'asta in un corridoio

Esercizio 2.15 Verificare che, indicata al solito con ϕ_N la funzione di iterazione del metodo di Newton considerato come metodo di punto fisso, se α è uno zero di f di molteplicità m , allora $\phi'_N(\alpha) = 1 - 1/m$. Se ne deduca che il metodo di Newton converge quadraticamente se α è uno zero semplice di $f(x) = 0$, linearmente negli altri casi.

Esercizio 2.16 Si tracci il grafico della funzione $f(x) = x^3 + 4x^2 - 10$ e se ne deduca che essa ammette un unico zero reale α . Per il suo calcolo si usino le seguenti iterazioni di punto fisso: dato $x^{(0)}$, trovare $x^{(k+1)}$ tale che

$$x^{(k+1)} = \frac{2(x^{(k)})^3 + 4(x^{(k)})^2 + 10}{3(x^{(k)})^2 + 8x^{(k)}}, \quad k \geq 0.$$

Se ne studi la convergenza a α .

Esercizio 2.17 Si studi la convergenza delle seguenti iterazioni di punto fisso

$$x^{(k+1)} = \frac{x^{(k)}[(x^{(k)})^2 + 3a]}{3(x^{(k)})^2 + a}, \quad k \geq 0,$$

per il calcolo della radice quadrata di un numero positivo a .

Esercizio 2.18 Si ripetano i calcoli effettuati nell'Esercizio 2.11 usando come criterio d'arresto quello sul residuo. È più accurato il risultato ottenuto ora o quello ricavato precedentemente?

3

Approssimazione di funzioni e di dati

Approssimare una funzione f significa trovare una funzione \tilde{f} di forma più semplice che verrà usata come surrogato di f . Questa strategia è frequentemente utilizzata nell'integrazione numerica in cui invece di calcolare $\int_a^b f(x)dx$ si calcola $\int_a^b \tilde{f}(x)dx$ ove \tilde{f} sia una funzione facile da integrare (ad esempio, un polinomio), come mostreremo nel prossimo capitolo. In altri contesti, la funzione f potrebbe essere nota solo parzialmente attraverso i valori che essa assume in determinati punti. In tal caso la determinazione di \tilde{f} consentirà di approssimare con una funzione continua l'andamento della “legge f ” che ha generato l'insieme di dati. Gli esempi che seguono danno un'idea di questo approccio.

Problema 3.1 (Climatologia) La temperatura dell'aria in prossimità del suolo dipende dalla concentrazione K dell'acido carbonico. In particolare, in Tabella 3.1 (tratta da Philosophical Magazine 41, 237 (1896)) vengono riportate, in corrispondenza di 4 diversi valori di K (e per diverse latitudini) le variazioni della temperatura media che si avrebbero nel globo rispetto a quella attuale (normalizzata al valore di riferimento $K = 1$). In questo caso possiamo costruire una funzione che, sulla base dei dati disponibili, fornisce un'approssimazione dei valori della temperatura media per ogni possibile latitudine (si veda l'Esempio 3.1).

Problema 3.2 (Finanza) In Figura 3.1 viene riportato l'andamento del prezzo di una particolare azione alla Borsa di Milano su due anni. La curva è stata ottenuta semplicemente congiungendo con un segmento i prezzi fissati ogni due giorni alla chiusura del mercato. Questa semplice rappresentazione assume implicitamente che il prezzo cambi linearmente durante il giorno (anticipiamo che questa approssimazione è nota come *interpolazione composita lineare*). Ci si può chiedere se da questo grafico si possa dedurre una previsione del prezzo dell'azione in esame per un breve periodo di tempo successivo all'ultima quotazione disponibile. Come vedremo nel paragrafo 3.4, informazioni di questo genere possono essere ottenute facendo uso di una tecnica nota come l'approssimazione di funzioni nel senso dei *minimi quadrati* (si veda l'Esempio 3.9).

Latitudine	$K = 0.67$	$K = 1.5$	$K = 2.0$	$K = 3.0$
65	-3.1	3.52	6.05	9.3
55	-3.22	3.62	6.02	9.3
45	-3.3	3.65	5.92	9.17
35	-3.32	3.52	5.7	8.82
25	-3.17	3.47	5.3	8.1
15	-3.07	3.25	5.02	7.52
5	-3.02	3.15	4.95	7.3
-5	-3.02	3.15	4.97	7.35
-15	-3.12	3.2	5.07	7.62
-25	-3.2	3.27	5.35	8.22
-35	-3.35	3.52	5.62	8.8
-45	-3.37	3.7	5.95	9.25
-55	-3.25	3.7	6.1	9.5

Tabella 3.1. Variazioni della temperatura media annua del globo al variare della concentrazione K di acido carbonico e della latitudine

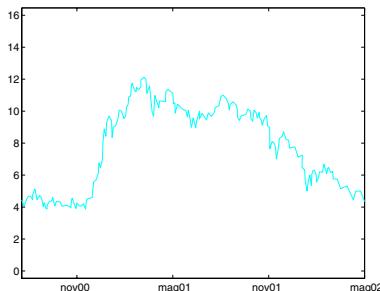


Figura 3.1. Andamento del prezzo di un'azione nell'arco di due anni

Test	Sforzo σ	Deformazione ϵ	Test	Sforzo σ	Deformazione ϵ
1	0.00	0.00	5	0.31	0.23
2	0.06	0.08	6	0.47	0.25
3	0.14	0.14	7	0.60	0.28
4	0.25	0.20	8	0.70	0.29

Tabella 3.2. Valori di deformazione relativi a diversi sforzi applicati ad un disco intervertebrale

Problema 3.3 (Biomeccanica) Nella Tabella 3.2 vengono riportati i risultati di un esperimento (P.Komarek, Capitolo 2 di *Biomechanics of Clinical Aspects of Biomedicine*, 1993, J.Valenta ed., Elsevier) eseguito per individuare il legame fra lo sforzo e la relativa deformazione di un campione di tessuto biologico (un disco intervertebrale, si veda la rappresentazione schematica di Figura 3.2). Partendo dai dati riportati in tabella si vuole stimare la deformazione corrispondente ad uno sforzo $\sigma = 0.9$ MPa ($\text{MPa} = 100 \text{ N/cm}^2$). Si veda per la risoluzione l'Esempio 3.10.

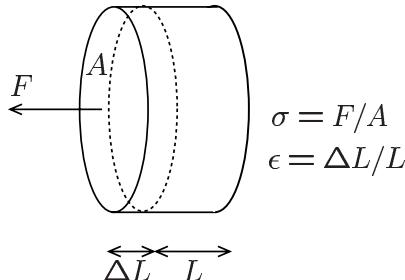


Figura 3.2. Una rappresentazione schematica di un disco intervertebrale

Problema 3.4 (Robotica) Si intende determinare nel piano xy la traiettoria seguita da un robot che viene impiegato per un ciclo di lavorazione in un'industria. Il robot deve rispettare determinati vincoli di movimento: in particolare, si vuole che al tempo iniziale ($t = 0$) il robot si trovi fermo nella posizione $(0, 0)$, al tempo $t = 1$ passi per il punto $(1, 2)$, raggiunga al tempo $t = 2$ il punto $(4, 4)$ con velocità nulla, riparta quindi per raggiungere il punto $(3, 1)$ al tempo $t = 3$ e ritorni al punto di partenza al tempo $t = 5$, fermandosi per poi iniziare un nuovo ciclo lavorativo. Si suppone che il robot sia assimilabile ad un punto materiale. Vedremo nell'Esempio 3.7 come risolvere questo problema.

Come è noto, una funzione f può essere approssimata in un intervallo dal suo polinomio di Taylor di un certo grado n , introdotto nel paragrafo 1.4.3. Tale procedura è assai costosa in quanto richiede la conoscenza di f e delle sue derivate fino all'ordine n in un dato punto x_0 . Inoltre, il polinomio di Taylor può non approssimare accuratamente f quando ci si discosta da x_0 . Ad esempio, in Figura 3.3 si confronta l'andamento della funzione $f(x) = 1/x$ con quello del suo polinomio di Taylor di grado 10 relativo al punto $x_0 = 1$. Questa figura mostra anche l'interfaccia grafica del programma MATLAB `taylortool` che consente di calcolare il polinomio di Taylor di grado arbitrario di ogni data funzione f . Come si vede più ci si allontana da x_0 più il polinomio di Taylor si discosta dalla funzione. Per altre funzioni ciò fortunatamente non si verifica; è il caso ad esempio della funzione esponenziale per la quale il polinomio di Taylor relativo al punto $x_0 = 0$ rappresenta una buona approssimazione per ogni valore di $x \in \mathbb{R}$ purché il grado n sia sufficientemente grande.

Servono quindi in generale dei metodi di approssimazione alternativi che illustreremo nei prossimi paragrafi.

`taylortool`

3.1 Interpolazione

Come abbiamo potuto notare dai problemi proposti, in molte applicazioni concrete si conosce una funzione solo attraverso i suoi valori in

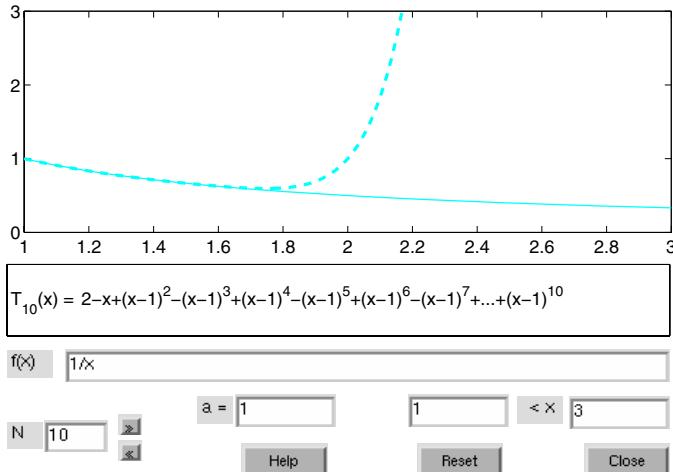


Figura 3.3. Confronto tra la funzione $f(x) = 1/x$ (in linea piena) ed il suo polinomio di Taylor di grado 10 rispetto al punto $x_0 = 1$ (in tratteggio). L'espressione del polinomio di Taylor è riportata in figura

determinati punti. Supponiamo pertanto di conoscere $n + 1$ coppie di valori $\{(x_i, y_i)\}$, $i = 0, \dots, n$, dove i punti x_i , tutti distinti, sono detti *nodi*.

Ad esempio, con riferimento alla Tabella 3.1, n è uguale a 12, i nodi x_i sono i valori della latitudine riportati nella prima colonna, mentre gli y_i sono i valori corrispondenti (della temperatura) che troviamo in una qualunque delle restanti colonne.

In tal caso, può apparire naturale richiedere che la funzione approssimante \tilde{f} soddisfi le seguenti uguaglianze

$$\tilde{f}(x_i) = y_i, \quad i = 0, 1, \dots, n \quad (3.1)$$

Una tale funzione \tilde{f} è detta *interpolatore* di f e le equazioni (3.1) sono le condizioni di interpolazione.

Si possono immaginare vari tipi di interpolatori, ad esempio:

- l'*interpolatore polinomiale*:

$$\tilde{f}(x) = a_0 + a_1 x + a_2 x^2 + \dots + a_n x^n;$$

- l'*interpolatore trigonometrico*:

$$\tilde{f}(x) = a_{-M} e^{-i M x} + \dots + a_0 + \dots + a_M e^{i M x},$$

dove M è un intero pari a $n/2$ se n è pari, $(n-1)/2$ se n è dispari, e i è l'unità immaginaria;

- l'interpolatore razionale:

$$\tilde{f}(x) = \frac{a_0 + a_1 x + \dots + a_k x^k}{a_{k+1} + a_{k+2} x + \dots + a_{k+n+1} x^n}.$$

Per semplicità considereremo soltanto quegli interpolatori che dipendono linearmente dai coefficienti incogniti a_i . Ad esempio, l'interpolazione polinomiale e quella trigonometrica rientrano in questa categoria, mentre quella razionale no.

3.1.1 Interpolazione polinomiale di Lagrange

Concentriamo la nostra attenzione sull'interpolazione polinomiale. Vale il seguente risultato:

Proposizione 3.1 *Per ogni insieme di coppie $\{x_i, y_i\}$, $i = 0, \dots, n$, con i nodi x_i distinti fra loro, esiste un unico polinomio di grado minore od uguale a n , che indichiamo con Π_n e chiamiamo polinomio interpolatore dei valori y_i nei nodi x_i , tale che*

$$\Pi_n(x_i) = y_i, \quad i = 0, \dots, n. \quad (3.2)$$

Quando i valori $\{y_i, i = 0, \dots, n\}$, rappresentano i valori assunti da una funzione continua f (ovvero $y_i = f(x_i)$), Π_n è detto polinomio interpolatore di f (in breve, interpolatore di f) e viene indicato con $\Pi_n f$.

Per verificare l'unicità procediamo per assurdo supponendo che esistano due polinomi distinti di grado n , Π_n e Π_n^* , che soddisfino entrambi le relazioni nodali (3.2). La loro differenza, $\Pi_n - \Pi_n^*$, sarà ancora un polinomio di grado n che si annulla in $n + 1$ punti distinti. Per un noto teorema dell'Algebra, esso deve essere identicamente nullo e, quindi, Π_n^* coincide con Π_n , da cui l'assurdo.

Per ottenere un'espressione di Π_n , iniziamo da una funzione molto speciale per la quale tutti gli y_i sono nulli fuorché quello per $i = k$ (per un dato k) per il quale $y_k = 1$. Posto allora $\varphi_k(x) = \Pi_n(x)$, si dovrà avere (si veda la Figura 3.4)

$$\varphi_k \in \mathbb{P}_n, \quad \varphi_k(x_j) = \delta_{jk} = \begin{cases} 1 & \text{se } j = k, \\ 0 & \text{altrimenti} \end{cases}$$

(δ_{jk} è il simbolo di Kronecker).

Le funzioni φ_k possono essere scritte come

$$\varphi_k(x) = \prod_{\substack{j=0 \\ j \neq k}}^n \frac{x - x_j}{x_k - x_j}, \quad k = 0, \dots, n. \quad (3.3)$$

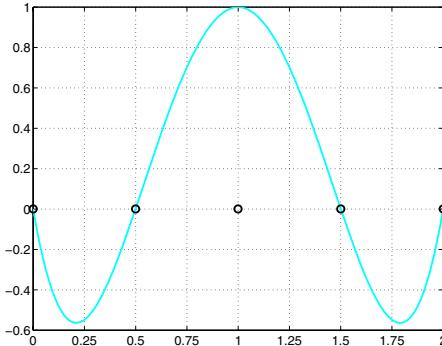


Figura 3.4. Il polinomio $\varphi_2 \in \mathbb{P}_4$ associato ai 5 nodi equispaziati 0, 0.5, 1, 1.5 e 2

Mettiamoci ora nel caso generale in cui $\{y_i, i = 0, \dots, n\}$ sia un insieme di valori arbitrari. Abbiamo

$$\Pi_n(x) = \sum_{k=0}^n y_k \varphi_k(x) \quad (3.4)$$

In effetti, questo polinomio soddisfa le condizioni di interpolazione (3.2) in quanto

$$\Pi_n(x_i) = \sum_{k=0}^n y_k \varphi_k(x_i) = \sum_{k=0}^n y_k \delta_{ik} = y_i, \quad i = 0, \dots, n.$$

Per il ruolo peculiare giocato dalle funzioni $\{\varphi_k\}$, esse sono dette *polinomi caratteristici* e la (3.4) è nota come *forma di Lagrange* del polinomio interpolatore.

In MATLAB possiamo memorizzare le $n+1$ coppie $\{(x_i, y_i)\}$ in due vettori, ad esempio x e y , e con l'istruzione `c=polyfit(x,y,n)` possiamo generare i coefficienti del polinomio interpolatore. In particolare, $c(1)$ conterrà il coefficiente di x^n , $c(2)$ quello di x^{n-1} , ... e $c(n+1)$ il valore di $\Pi_n(0)$. (Maggiori dettagli su questo comando sono contenuti nel paragrafo 3.4.)

Come abbiamo visto nel Capitolo 1, noti i coefficienti, attraverso l'istruzione `p=polyval(c,z)` è poi possibile calcolare i valori $p(j)$ del polinomio interpolatore in m punti arbitrari $z(j)$, $j=1, \dots, m$.

Nel caso in cui invece sia nota la funzione da interpolare, memorizzata ad esempio in una stringa f in funzione di x , basta costruire i nodi, memorizzati nel vettore x , nei quali f deve essere interpolata e porre `y=eval(f)`. A questo punto possiamo procedere come nel caso precedente avendo a disposizione i vettori x e y .

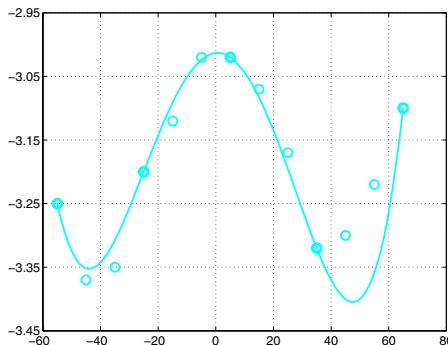


Figura 3.5. Il polinomio interpolatore di grado 4 introdotto nell’Esempio 3.1

Esempio 3.1 (Climatologia) Calcoliamo il polinomio interpolatore di grado 4 per i dati del Problema 3.1 relativi ad una concentrazione K di acido carbonico pari a 0.67 (prima colonna), utilizzando i valori della temperatura corrispondenti alle sole latitudini 65, 35, 5, -25, -55. Possiamo utilizzare le seguenti istruzioni MATLAB

```
>> x = [-55 -25 5 35 65];
>> y = [-3.25 -3.2 -3.02 -3.32 -3.1];
>> format short e;
>> c=polyfit(x,y,4)
c =
    8.2819e-08 -4.5267e-07 -3.4684e-04  3.7757e-04 -3.0132e+00
```

Il grafico del polinomio interpolatore può allora essere generato come segue

```
>> z=linspace(x(1),x(end),100);
>> p=polyval(c,z);
>> plot(z,p,x,y,'o');
```

Si noti che al solo scopo di ottenere una rappresentazione “liscia” il polinomio è stato valutato in 101 punti equispaziati nell’intervallo $[-55, 65]$ (in effetti, quando MATLAB disegna una curva si limita a congiungere due punti consecutivi con un segmento). L’istruzione $x(\text{end})$ consente di accedere direttamente all’ultima componente del vettore x , senza bisogno di conoscerne la lunghezza. In Figura 3.5 i cerchietti (pieni e vuoti) indicano tutti i dati disponibili, mentre quelli pieni solo i dati utilizzati per costruire il polinomio di interpolazione. Si può apprezzare il buon accordo a livello qualitativo fra il polinomio interpolatore e la distribuzione dei dati.

Grazie al risultato seguente possiamo quantificare l’errore che si commette sostituendo ad una funzione f il suo polinomio interpolatore $\Pi_n f$:

Proposizione 3.2 *Sia I un intervallo limitato, e si considerino $n + 1$ nodi di interpolazione distinti $\{x_i, i = 0, \dots, n\}$ in I . Sia f derivabile con continuità fino all'ordine $n + 1$ in I . Allora $\forall x \in I \exists \xi \in I$ tale che*

$$E_n f(x) = f(x) - \Pi_n f(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} \prod_{i=0}^n (x - x_i) \quad (3.5)$$

Ovviamente, $E_n f(x_i) = 0$, $i = 0, \dots, n$.

Il risultato (3.5) può essere meglio specificato nel caso di una distribuzione uniforme di nodi, ovvero quando $x_i = x_{i-1} + h$ per $i = 1, \dots, n$, per un dato $h > 0$ ed un dato x_0 . In tal caso, si veda l'Esercizio 3.1, $\forall x \in (x_0, x_n)$ si può verificare che

$$\left| \prod_{i=0}^n (x - x_i) \right| \leq n! \frac{h^{n+1}}{4}, \quad (3.6)$$

e quindi

$$\max_{x \in I} |E_n f(x)| \leq \frac{\max_{x \in I} |f^{(n+1)}(x)|}{4(n+1)} h^{n+1}. \quad (3.7)$$

Purtroppo non si può dedurre dalla (3.7) che l'errore tende a 0 per $n \rightarrow \infty$, nonostante $h^{n+1}/[4(n+1)]$ tenda a 0. Infatti, come mostra l'Esempio 3.2, esistono funzioni f per le quali addirittura tale limite può essere infinito, ovvero

$$\lim_{n \rightarrow \infty} \max_{x \in I} |E_n f(x)| = \infty.$$

Questo risultato indica che ad un aumento del grado n del polinomio interpolatore non corrisponde necessariamente un miglioramento nella ricostruzione di una funzione f . Ad esempio, se interpolassimo tutti i dati della seconda colonna della Tabella 3.1, troveremmo il polinomio Π_{12} , rappresentato in Figura 3.6, il cui comportamento, nelle vicinanze dell'estremo di sinistra dell'intervallo è assai meno soddisfacente di quello mostrato in Figura 3.5 utilizzando un numero inferiore di nodi. Si può riscontrare un comportamento ancor più insoddisfacente per particolari funzioni, come risulta dall'esempio seguente.

Esempio 3.2 (Runge) Se interpoliamo la funzione $f(x) = 1/(1+x^2)$ (detta di Runge) su un insieme di nodi equispaziati nell'intervallo $I = [-5, 5]$, l'errore $\max_{x \in I} |E_n f(x)|$ tende all'infinito quando $n \rightarrow \infty$. Questo è dovuto al fatto che per $n \rightarrow \infty$ l'ordine di infinito di $\max_{x \in I} |f^{(n+1)}(x)|$ supera quello di infinitesimo di $h^{n+1}/[4(n+1)]$.

Possiamo verificare questa conclusione calcolando il massimo delle derivate di f fino all'ordine 21 con le seguenti istruzioni

```
>> syms x; n=20; f=1/(1+x^2); df=diff(f,1);
>> cdf = char(df);
>> for i = 1:n+1, df = diff(df,1); cdfn = char(df);
    x = fzero(cdfn,0); M(i) = abs(eval(cdf)); cdf = cdfn;
end
```

I massimi del valore assoluto delle funzioni $f^{(n)}$, $n = 1, \dots, 21$, sono stati memorizzati nel vettore M . Si noti che il comando `char` converte la variabile simbolica `df` in una stringa che possa poi essere valutata dalla funzione `fzero`. In particolare, i valori assoluti di $f^{(n)}$ per $n = 3, 9, 15, 21$ sono

```
>> format short e; M([3,9,15,21]) =
ans =
4.6686e+00 3.2426e+05 1.2160e+12 4.8421e+19
```

mentre i corrispondenti valori assoluti di $\prod_{i=0}^n (x - x_i)/(n + 1)!$ sono

```
>> z = linspace(-5,5,10000);
>> for n=0:20; h=10/(n+1); x=[-5:h:5];
    c=poly(x); r(n+1)=max(polyval(c,z));r(n+1)=r(n+1)/prod([1:n+2]); end
>> r([3,9,15,21])
ans =
2.8935e+00 5.1813e-03 8.5854e-07 2.1461e-11
```

dove `c=poly(x)` è un vettore i cui elementi sono i coefficienti del polinomio che ha come radici proprio gli elementi del vettore `x`. Ne consegue che $\max_{x \in I} |E_n f(x)|$ assume i seguenti valori

```
1.3509e+01 1.6801e+03 1.0442e+06 1.0399e+09
```

rispettivamente per $n = 3, 9, 15, 21$.

La mancanza di convergenza si manifesta nelle forti oscillazioni, presenti nel grafico del polinomio interpolatore rispetto a quello di f , che tendono ad amplificarsi in prossimità degli estremi dell'intervallo (si veda la Figura 3.6 a destra). Questo comportamento è noto come *fenomeno di Runge*.

Osservazione 3.1 Si può anche dimostrare la seguente diseguaglianza

$$\max_{x \in I} |f'(x) - (\Pi_n f)'(x)| \leq Ch^n \max_{x \in I} |f^{(n+1)}(x)|,$$

dove C è una costante indipendente da h . Quindi se approssimiamo la derivata prima di f con la derivata prima di $\Pi_n f$, dobbiamo aspettarci di perdere un ordine di convergenza rispetto a h . Si noti che $(\Pi_n f)'$ può essere calcolato tramite il comando MATLAB `[d]=polyder(c)`, dove il parametro `c` di *input* è il vettore che memorizza i coefficienti del polinomio interpolatore, mentre `d` è il vettore dei coefficienti della sua derivata (si veda il paragrafo 1.4.2).

`polyder`

Si vedano gli Esercizi 3.1-3.4.



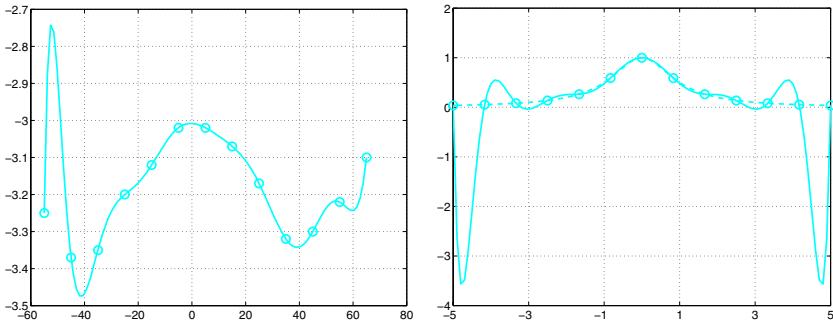


Figura 3.6. Due esemplificazioni del fenomeno di Runge: a sinistra, $\Pi_{12}f$ calcolato per l'insieme di dati della Tabella 3.1, colonna $K = 0.67$; a destra, $\Pi_{12}f$ (in linea continua) calcolato su 13 nodi equispaziati nel caso della funzione di Runge $f(x) = 1/(1+x^2)$ (in linea tratteggiata)

3.1.2 Interpolazione di Chebyshev

Il fenomeno di Runge può essere evitato utilizzando opportune distribuzioni di nodi. In particolare, su un arbitrario intervallo $[a, b]$ consideriamo i cosiddetti *nodi di Chebyshev*

$$x_i = \frac{a+b}{2} + \frac{b-a}{2}\widehat{x}_i, \text{ dove } \widehat{x}_i = -\cos(\pi i/n), i = 0, \dots, n. \quad (3.8)$$

(Naturalmente $x_i = \widehat{x}_i$, $i = 0, \dots, n$ quando $[a, b] = [-1, 1]$.) Si può dimostrare che se f è una funzione continua e derivabile con continuità in $[a, b]$, il polinomio interpolatore $\Pi_n f$ associato a questa particolare distribuzione di nodi converge a f per $n \rightarrow \infty$, per ogni $x \in [a, b]$.

I nodi di Chebyshev, che sono le ascisse di nodi equispaziati sulla semicirconferenza di raggio uno, appartengono all'intervallo $[a, b]$ e si addensano vicino agli estremi dell'intervallo (si veda la Figura 3.7 a destra).

Un'altra distribuzione di nodi sull'intervallo (a, b) , per la quale si hanno le stesse proprietà di convergenza, è data da

$$x_i = \frac{a+b}{2} - \frac{b-a}{2} \cos \left(\frac{2i+1}{n+1} \frac{\pi}{2} \right), i = 0, \dots, n.$$

Esempio 3.3 Riprendiamo la funzione di Runge ed interpoliamola utilizzando gli $n + 1$ nodi di Chebyshev nell'intervallo $[-5, 5]$. Per generarli possiamo usare i seguenti comandi

```
>> xc = -cos(pi*[0:n]/n); x = (a+b)*0.5+(b-a)*xc*0.5;
```

dove a e b sono gli estremi dell'intervallo di interpolazione (nel nostro caso porremo $a=-5$ e $b=5$).

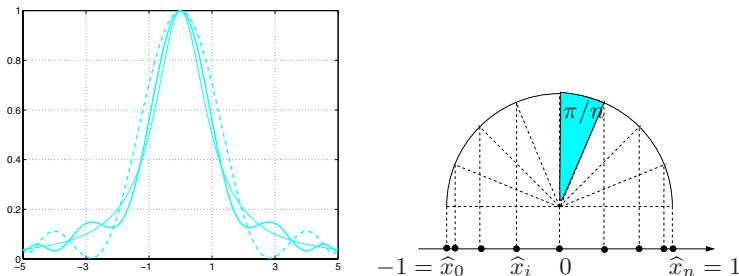


Figura 3.7. A sinistra, la funzione di Runge $f(x) = 1/(1+x^2)$ (in linea continua sottile) a confronto con i polinomi interpolatori sui nodi di Chebyshev di grado 8 (linea tratteggiata) e 12 (linea continua più marcata). Si noti come ora, al crescere del grado, le oscillazioni si smorzino e l'approssimazione divenga sempre più accurata. A destra, riportiamo la distribuzione dei nodi di Chebyshev nell'intervallo $[-1, 1]$

Quindi, il polinomio interpolatore si genererà con le seguenti istruzioni

```
>> f= '1./(1+x.^2)'; y = eval(f); c = polyfit(x,y,n);
```

Valutiamo a questo punto il valore assoluto delle differenze fra f ed il suo polinomio interpolatore di Chebyshev in 1001 punti equispaziati nell'intervallo $[-5, 5]$ e prendiamone il massimo

```
>> x = linspace(-5,5,1000); p=polyval(c,x); fx = eval(f); err = max(abs(p-fx));
```

Come si vede in Tabella 3.3, il massimo dell'errore descresce quando n cresce.

n	5	10	20	40
E_n	0.6386	0.1322	0.0177	0.0003

Tabella 3.3. L'errore di interpolazione per la funzione di Runge qualora si utilizzino i nodi di Chebyshev

3.1.3 Interpolazione trigonometrica e FFT

Vogliamo approssimare una funzione $f : [0, 2\pi] \rightarrow \mathbb{C}$ periodica, cioè tale che $f(0) = f(2\pi)$, con un polimomio trigonometrico \tilde{f} che la interpoly negli $n + 1$ nodi $x_j = 2\pi j / (n + 1)$, $j = 0, \dots, n$, ovvero tale che

$$\tilde{f}(x_j) = f(x_j), \text{ per } j = 0, \dots, n. \quad (3.9)$$

L'interpolatore trigonometrico \tilde{f} si ottiene attraverso una combinazione lineare di seni e coseni.

In particolare, \tilde{f} assumerà le forme seguenti: se n è pari



$$\tilde{f}(x) = \frac{a_0}{2} + \sum_{k=1}^M [a_k \cos(kx) + b_k \sin(kx)], \quad (3.10)$$

dove $M = n/2$ mentre, se n è dispari,

$$\tilde{f}(x) = \frac{a_0}{2} + \sum_{k=1}^M [a_k \cos(kx) + b_k \sin(kx)] + a_{M+1} \cos((M+1)x), \quad (3.11)$$

dove $M = (n-1)/2$. Possiamo riscrivere (3.10) come

$$\tilde{f}(x) = \sum_{k=-M}^M c_k e^{ikx}, \quad (3.12)$$

dove i è l'unità immaginaria. I coefficienti complessi c_k sono legati ai coefficienti a_k e b_k , anch'essi complessi, dalle relazioni

$$a_k = c_k + c_{-k}, \quad b_k = i(c_k - c_{-k}), \quad k = 0, \dots, M. \quad (3.13)$$

Infatti, dalla (1.5) segue che $e^{ikx} = \cos(kx) + i \sin(kx)$ e

$$\begin{aligned} \sum_{k=-M}^M c_k e^{ikx} &= \sum_{k=-M}^M c_k (\cos(kx) + i \sin(kx)) \\ &= \sum_{k=1}^M [c_k(\cos(kx) + i \sin(kx)) + c_{-k}(\cos(kx) - i \sin(kx))] + c_0, \end{aligned}$$

da cui segue la (3.10), grazie alla (3.13).

Analogamente, se n è dispari, la (3.11) assume la forma

$$\tilde{f}(x) = \sum_{k=-(M+1)}^{M+1} c_k e^{ikx}, \quad (3.14)$$

dove i coefficienti c_k per $k = 0, \dots, M$ sono determinati come prima, mentre $c_{M+1} = c_{-(M+1)} = a_{M+1}/2$. In entrambi i casi, potremo scrivere

$$\tilde{f}(x) = \sum_{k=-(M+\mu)}^{M+\mu} c_k e^{ikx}, \quad (3.15)$$

con $\mu = 0$ se n è pari, $\mu = 1$ se n è dispari. Nel caso in cui f sia una funzione a valori reali, i coefficienti c_k soddisfano la relazione $c_{-k} = \bar{c}_k$ e, dalla (3.13), segue che i coefficienti a_k e b_k sono tutti reali.

Per la sua analogia con lo sviluppo in serie di Fourier, \tilde{f} è detta anche *serie discreta di Fourier* di f . Imponendo le condizioni di interpolazione nei nodi $x_j = jh$, con $h = 2\pi/(n+1)$, troviamo che

$$\sum_{k=-(M+\mu)}^{M+\mu} c_k e^{ikjh} = f(x_j), \quad j = 0, \dots, n. \quad (3.16)$$

Per il calcolo dei coefficienti $\{c_k\}$ moltiplichiamo ambo i membri della (3.16) per $e^{-imx_j} = e^{-imjh}$ con m intero fra 0 e n , e sommiamo poi su j

$$\sum_{j=0}^n \sum_{k=-(M+\mu)}^{M+\mu} c_k e^{ikjh} e^{-imjh} = \sum_{j=0}^n f(x_j) e^{-imjh}. \quad (3.17)$$

Consideriamo ora l'identità

$$\sum_{j=0}^n e^{ijh(k-m)} = (n+1)\delta_{km}.$$

Essa è ovvia se $k = m$. Quando $k \neq m$, abbiamo

$$\sum_{j=0}^n e^{ijh(k-m)} = \frac{1 - (e^{i(k-m)h})^{n+1}}{1 - e^{i(k-m)h}},$$

ma il numeratore a destra è nullo in quanto

$$1 - e^{i(k-m)h(n+1)} = 1 - e^{i(k-m)2\pi} = 1 - \cos((k-m)2\pi) - i \sin((k-m)2\pi).$$

Di conseguenza, dalla (3.17) ricaviamo un'espressione esplicita dei coefficienti di \tilde{f}

$$c_k = \frac{1}{n+1} \sum_{j=0}^n f(x_j) e^{-ikjh}, \quad k = -(M+\mu), \dots, M+\mu \quad (3.18)$$

Il calcolo di tutti i coefficienti $\{c_k\}$ può essere effettuato con un costo computazionale dell'ordine di $n \log_2 n$ operazioni se si ricorre alla *trasformata rapida di Fourier* o FFT, implementata in MATLAB nel programma **fft** (si veda l'Esempio 3.4). Un costo analogo ha la trasformata inversa attraverso la quale si trovano i valori $\{f(x_j)\}$ a partire dai coefficienti $\{c_k\}$. Essa è implementata nella sua versione rapida nel programma **ifft**.

fft**ifft**

Esempio 3.4 Consideriamo la funzione $f(x) = x(x-2\pi)e^{-x}$ per $x \in [0, 2\pi]$. Per usare il comando MATLAB **fft**, campioniamo la funzione nei nodi $x_j = j\pi/5$ per $j = 0, \dots, 9$. A questo punto tramite i seguenti comandi (ricordiamo che $.*$ è il prodotto fra vettori, componente per componente)

```
>> x=pi/5*[0:9]; y=x.*(x-2*pi).*exp(-x); Y=fft(y);
```

troviamo gli $n + 1$ valori

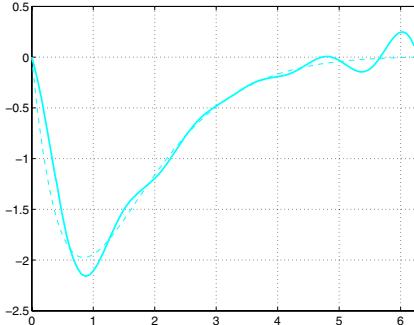


Figura 3.8. La funzione $f(x) = x(x - 2\pi)e^{-x}$ (in linea tratteggiata) ed il corrispondente interpolatore trigonometrico (in linea continua) relativo a 10 nodi equispaziati

```
>> Y =
Columns 1 through 3
-6.5203e+00      -4.6728e-01 + 4.2001e+00i 1.2681e+00 + 1.6211e+00i
Columns 4 through 6
1.0985e+00 + 6.0080e-01i 9.2585e-01 + 2.1398e-01i 8.7010e-01 - 1.3887e-
16i
Columns 7 through 9
9.2585e-01 - 2.1398e-01i 1.0985e+00 - 6.0080e-01i 1.2681e+00 - 1.6211e+00i
Column 10
-4.6728e-01 - 4.2001e+00i
```

dove $\mathbf{Y} = (n+1)[c_0, \dots, c_{M+\mu}, c_{-M}, \dots, c_{-1}]$.

Si noti che il programma `ifft`, seppur utilizzabile per ogni valore di n , raggiunge tuttavia il massimo dell'efficienza computazionale quando n è una potenza di 2.

interpft

Il comando `interpft` calcola l'interpolatore trigonometrico di un insieme di dati. Richiede come parametri d'ingresso un intero m ed un vettore le cui componenti sono i valori assunti da una funzione (periodica di periodo p) nei punti $x_j = jp/(n+1)$, $j = 0, \dots, n$. Il programma `interpft` restituisce gli m valori dell'interpolatore trigonometrico, ottenuto con la trasformata di Fourier, nei nodi $t_i = ip/m$, $i = 0, \dots, m-1$. Ad esempio, riconsideriamo la funzione dell'Esempio 3.4 in $[0, 2\pi]$ e valutiamola in 10 nodi equispaziati $x_j = j\pi/5$, $j = 0, \dots, 9$. I valori dell'interpolatore trigonometrico, ad esempio nei 100 nodi equispaziati $t_i = i\pi/100$, $i = 0, \dots, 99$, si possono ottenere nel modo seguente (si veda la Figura 3.8)

```
>> x=pi/5*[0:9]; y=x.*(x-2*pi).*exp(-x); z=interpft(y,100);
```

L'accuratezza dell'interpolazione trigonometrica può in certe situazioni subire un forte degrado come mostrato nell'esempio seguente.

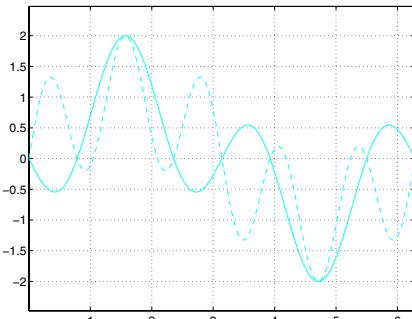


Figura 3.9. Gli effetti dell’aliasing: confronto tra la funzione $f(x) = \sin(x) + \sin(5x)$ (in linea continua) ed il suo interpolatore trigonometrico (3.10) con $M = 3$ (linea tratteggiata)

Esempio 3.5 Approssimiamo la funzione $f(x) = f_1(x) + f_2(x)$ dove $f_1(x) = \sin(x)$ e $f_2(x) = \sin(5x)$, usando 9 nodi equispaziati nell’intervallo $[0, 2\pi]$. Il risultato viene riportato in Figura 3.9. Si noti che in certi intervalli l’approssimante trigonometrica presenta un’inversione di fase rispetto a f .

Questa comportamento può essere spiegato osservando che nei nodi considerati, la funzione f_2 è indistinguibile dalla funzione $f_3(x) = -\sin(3x)$ che ha una frequenza più bassa (si veda la Figura 3.10). La funzione che viene approssimata è quindi $F(x) = f_1(x) + f_3(x)$ e non $f(x)$ (in effetti, il grafico in tratteggiato della Figura 3.9 è quello di F).

Questo fenomeno prende il nome di *aliasing* e si può manifestare ogni volta che in una stessa funzione coesistono componenti con frequenza diversa: finché il numero di nodi non è sufficientemente alto per risolvere le frequenze più elevate, queste ultime potranno interferire con le frequenze più basse, dando origine ad approssimazioni inaccurate. Solo aumentando il numero di nodi sarà possibile approssimare correttamente le funzioni di frequenza più elevata.

Un esempio concreto di *aliasing* è dato dall’apparente inversione del senso di rotazione di ruote munite di raggi: raggiunta una certa velocità critica, il nostro cervello non è più in grado di campionare in modo sufficientemente accurato l’immagine in movimento e, di conseguenza, produce immagini distorte.

Riassumendo

1. Approssimare un insieme di dati o una funzione f in $[a, b]$ significa trovare un’opportuna funzione \tilde{f} sufficientemente rappresentativa;
2. il processo di interpolazione consiste nel trovare una funzione \tilde{f} tale che $\tilde{f}(x_i) = y_i$, dove $\{x_i\}$ sono nodi assegnati e $\{y_i\}$ possono essere o i valori $\{f(x_i)\}$ o un insieme di valori assegnati;

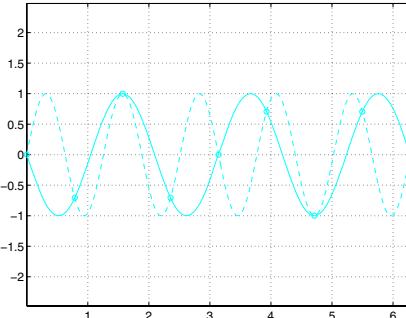


Figura 3.10. Il fenomeno dell’*aliasing*: le funzioni $\sin(5x)$ (in linea tratteggiata) e $-\sin(3x)$ (in linea continua) assumono gli stessi valori nei nodi di interpolazione. Questo spiega la forte perdita di accuratezza mostrata in Figura 3.9

3. se gli $n+1$ nodi $\{x_i\}$ sono distinti, allora esiste un unico polinomio di grado minore o uguale a n che interpola un insieme di valori assegnati $\{y_i\}$ nei nodi $\{x_i\}$;
4. per una distribuzione di nodi equispaziati in $[a, b]$ l’errore di interpolazione in un generico punto di $[a, b]$ non tende necessariamente a 0 quando n tende all’infinito. Tuttavia, esistono delle speciali distribuzioni di nodi, come ad esempio quella di Chebyshev, per le quali la convergenza a zero dell’errore di interpolazione è garantita per tutte le funzioni continue e derivabili;
5. l’interpolazione trigonometrica è una forma di interpolazione ideale per funzioni periodiche nella quale si sceglie \tilde{f} come una combinazione lineare di seni e coseni. La FFT è un algoritmo particolarmente efficiente per il calcolo dei coefficienti di Fourier dell’interpolatore trigonometrico a partire dai suoi valori nodali. Esso ammette un algoritmo inverso ugualmente efficiente, la IFFT.

3.2 Interpolazione lineare composita

Se f è una funzione di cui si conosce l’espressione analitica, l’interpolazione di Chebyshev fornisce uno strumento di approssimazione ampiamente soddisfacente. In tutti quei casi, invece, in cui f sia nota solo attraverso i suoi valori in un insieme assegnato di punti (che potrebbero non coincidere con i nodi di Chebyshev) si può ricorrere ad un metodo di interpolazione differente, detto interpolazione composita lineare.

Precisamente, data una distribuzione di nodi $x_0 < x_1 < \dots < x_{n-1} < x_n$, non necessariamente equispaziati, indichiamo con I_i l’intervallo $[x_i, x_{i+1}]$. Approssimiamo f con una funzione globalmente continua che, su ciascun intervallo, è data dal segmento congiungente i punti

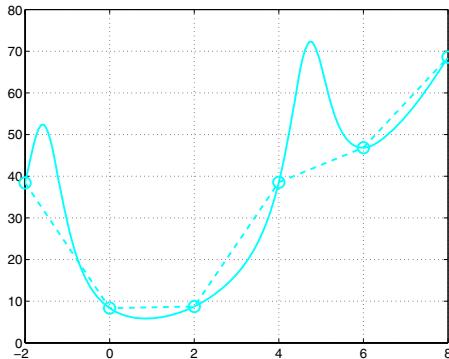


Figura 3.11. La funzione $f(x) = x^2 + 10/(\sin(x) + 1.2)$ (in linea continua) ed il suo interpolatore lineare composito $\Pi_1^H f$ (in linea tratteggiata)

$(x_i, f(x_i))$ e $(x_{i+1}, f(x_{i+1}))$ (si veda la Figura 3.11). Tale funzione, denotata con $\Pi_1^H f$, è detta *polinomio interpolatore composito lineare* ed assume la seguente espressione

$$\Pi_1^H f(x) = f(x_i) + \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}(x - x_i) \text{ per } x \in I_i.$$

L'indice H rappresenta la massima lunghezza degli intervalli I_i .

Il seguente risultato può essere dedotto dalla (3.7) ponendo $n = 1$ e $h = H$:

Proposizione 3.3 *Se $f \in C^2(I)$, dove $I = [x_0, x_n]$, allora*

$$\max_{x \in I} |f(x) - \Pi_1^H f(x)| \leq \frac{H^2}{8} \max_{x \in I} |f''(x)|.$$

Di conseguenza, per ogni x nell'intervallo di interpolazione, $\Pi_1^H f(x)$ tende a $f(x)$ quando $H \rightarrow 0$, purché f sia sufficientemente regolare.

Tramite il comando `s1=interp1(x,y,z)` si possono calcolare i valori in un insieme arbitrario di punti, memorizzati nel vettore `z`, assunti dall'interpolatore lineare composito che interpola i valori `y(i)` nei nodi `x(i)`, per $i = 1, \dots, n+1$. Quando i nodi di interpolazione sono dati in ordine crescente (*i.e.* $x(i+1) > x(i)$, per $i=1, \dots, n$) allora si può usare la versione computazionalmente più economica `interp1q` (in inglese `q` sta per *quickly*).

Facciamo notare che il comando `fplot`, che viene utilizzato per disegnare il grafico di una funzione f su un dato intervallo $[a, b]$, di fatto disegna il grafico dell'interpolatore lineare composito di f . L'insieme dei nodi di interpolazione viene generato automaticamente dal programma

`interp1`

`interp1q`

MATLAB, seguendo il criterio di infittire i nodi laddove f varia più rapidamente. Una procedura di questo tipo è detta *adattiva*.

3.3 Approssimazione con funzioni *spline*



Naturalmente si può definire anche un'interpolazione composita di grado ≥ 1 , ad esempio quadratica (che indicheremo con $\Pi_2^H f$) ossia una funzione continua che, ristretta ad ogni intervallo I_i sia un polinomio di grado 2. Se $f \in C^3(I)$, l'errore generato è questa volta un infinitesimo di ordine 3 rispetto a H . Tuttavia, la principale controindicazione dell'interpolazione composita (lineare o di grado $k \geq 1$) è che la funzione $\Pi_k^H f$ nei punti $\{x_i\}$ è solo continua. D'altra parte, in molte applicazioni, come ad esempio in *computer graphics*, è necessario utilizzare funzioni approssimanti che siano almeno derivabili con continuità.

A questo scopo, costruiamo una funzione s_3 che abbia le seguenti caratteristiche:

1. su ogni intervallo $I_i = [x_i, x_{i+1}]$, per $i = 0, \dots, n - 1$, s_3 deve essere un polinomio di grado 3 che interpola le coppie di valori $(x_j, f(x_j))$ per $j = i, i + 1$;
2. s_3 deve avere derivata prima e seconda continua in ogni punto x_i , $i = 1, \dots, n - 1$.

Per la sua completa determinazione è necessario assegnare 4 condizioni su ciascun intervallo e, conseguentemente, $4n$ equazioni in tutto che possiamo così individuare:

- $n + 1$ condizioni dovute alla richiesta che s_3 interpoli i dati nei nodi x_i , $i = 0, \dots, n$;
- $n - 1$ condizioni discendono dalla richiesta che s_3 sia continua nei nodi interni x_1, \dots, x_{n-1} ;
- imponendo anche la continuità della derivata prima e della derivata seconda negli stessi nodi otteniamo $2(n - 1)$ equazioni addizionali.

Restano ancora da individuare 2 equazioni che possono ad esempio essere date da

$$s_3''(x_0) = 0, s_3''(x_n) = 0. \quad (3.19)$$

La funzione s_3 così caratterizzata è detta *spline cubica interpolatoria naturale*.

Scegliendo opportunamente le incognite per rappresentare s_3 (si veda [QSS02], paragrafo 7.6), si può determinare s_3 risolvendo un sistema lineare di dimensione $(n + 1)$ con matrice tridiagonale e le cui incognite sono i valori $s''(x_i)$, per $i = 0, \dots, n$. Tale soluzione può essere ottenuta con un numero di operazioni proporzionale alla dimensione del sistema

stesso (come vedremo nel paragrafo 5.4) attraverso il Programma 7 i cui parametri d'ingresso obbligatori sono i vettori \mathbf{x} e \mathbf{y} dei dati da interpolare ed il vettore \mathbf{z} delle ascisse nelle quali si vuole che venga valutata s_3 . La scelta (3.19) non è l'unica possibile per completare il sistema di equazioni. Un'alternativa a (3.19) consiste nel richiedere che la derivata prima sia assegnata in x_0 ed in x_n .

Se non viene precisato alcun altro parametro d'ingresso il Programma 7 calcola la spline cubica interpolatoria naturale. I parametri opzionali **type** e **der** (un vettore di due componenti) servono per selezionare altri tipi di spline. Precisamente, se **type=0** viene calcolata la spline cubica interpolatoria con derivata prima assegnata agli estremi e pari a **der(1)** nell'estremo di sinistra dell'intervallo considerato, a **der(2)** in quello di destra. Se **type=1** viene invece calcolata la spline cubica interpolatoria con derivata seconda assegnata agli estremi e pari a **der(1)** e **der(2)**, rispettivamente.

Diversamente, nel comando MATLAB **spline** (si veda anche il toolbox **splines**) si impone che la derivata terza di s_3 sia continua nei nodi x_1 e x_{n-1} ; a questa condizione viene attribuito il curioso nome di *not-a-knot condition*. I parametri di ingresso del comando **spline** sono i vettori \mathbf{x} e \mathbf{y} dei dati da interpolare ed il vettore \mathbf{z} delle ascisse nelle quali si vuole che venga valutata s_3 . I comandi **mkpp** e **ppval** servono per costruire e valutare efficientemente in MATLAB un polinomio composito.

spline**mkpp**
ppval

Programma 7 - **cubicspline** : spline cubica interpolante

```
function s=cubicspline(x,y,zi,type,der)
%CUBICSPLINE calcola una spline cubica
% S=CUBICSPLINE(X,Y,ZI) calcola le valutazioni nei nodi ZI della spline
% cubica naturale che interpola i valori Y relativi ai nodi X.
% S=CUBICSPLINE(X,Y,ZI,TYPE,DER) se TYPE=0 calcola le valutazioni nei
% nodi ZI della spline cubica interpolante i valori Y con derivata prima
% assegnata agli estremi (DER(1) e DER(2)). Se TYPE=1 i valori DER(1) e DER(2)
% si riferiscono invece ai valori della derivata seconda.
[n,m]=size(x);
if n == 1
    x = x'; y = y'; n = m;
end
if nargin == 3
    der0 = 0; dern = 0; type = 1;
else
    der0 = der(1); dern = der(2);
end
h = x(2:end)-x(1:end-1);
e = 2*[h(1); h(1:end-1)+h(2:end); h(end)];
A = spdiags([[h; 0] e [0; h]],-1:1,n,n);
d = (y(2:end)-y(1:end-1))/h;
rhs = 3*(d(2:end)-d(1:end-1));
```



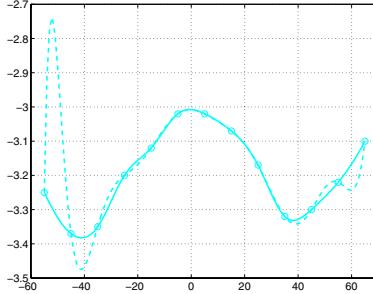


Figura 3.12. Confronto fra la spline cubica ed il polinomio interpolatore di Lagrange per il caso discusso nell’Esempio 3.6

```

if type == 0
    A(1,1) = 2*h(1); A(1,2) = h(1);
    A(n,n) = 2*h(end); A(end,end-1) = h(end);
    rhs = [3*(d(1)-der0); rhs; 3*(dern-d(end))];
else
    A(1,:) = 0; A(1,1) = 1;
    A(n,:) = 0; A(n,n) = 1;
    rhs = [der0; rhs; dern];
end
S = zeros(n,4);
S(:,3) = A\rhs;
for m = 1:n-1
    S(m,4) = (S(m+1,3)-S(m,3))/3/h(m);
    S(m,2) = d(m) - h(m)/3*(S(m + 1,3)+2*S(m,3));
    S(m,1) = y(m);
end
S = S(1:n-1, 4:-1:1); pp = mkpp(x,S); s = ppval(pp,zi);
return

```

Esempio 3.6 Riprendiamo i dati della Tabella 3.1, colonna corrispondente a $K = 0.67$ e calcoliamo su di essi la spline cubica interpolatoria s_3 . Se siamo interessati a valutare $s_3(z_i)$, dove $z_i = -55 + i$, $i = 0, \dots, 120$, possiamo procedere nel modo seguente

```

>> x = [-55:10:65];
>> y = [-3.25 -3.37 -3.35 -3.2 -3.12 -3.02 -3.02 ...
   -3.07 -3.17 -3.32 -3.3 -3.22 -3.1];
>> z = [-55:1:65];
>> s = spline(x,y,z);

```

Il grafico di s_3 , riportato in Figura 3.12, appare più plausibile di quello generato dall’interpolatore di Lagrange negli stessi nodi.

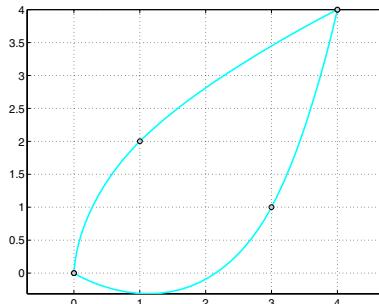


Figura 3.13. La traiettoria nel piano xy del robot descritto nel Problema 3.4. I pallini rappresentano le posizioni dei punti attraverso cui doveva transitare il robot durante il suo movimento

Esempio 3.7 (Robotica) Troviamo una rappresentazione parametrica della funzione che descrive la traiettoria del robot del Problema 3.4 nel piano xy . Dobbiamo determinare due funzioni $x = x(t)$ e $y = y(t)$ con $t \in (0, 5)$ che rispettino i vincoli imposti. Risolviamo il problema dividendo l'intervallo temporale nei due sottointervalli $(0, 2)$ e $(2, 5)$. Cerchiamo in ciascun intervallo due *spline* cubiche interpolanti nei piani tx e ty i valori dati, che presentino derivata prima nulla agli estremi per garantire che la velocità del robot sia nulla in tali posizioni. Usando il Programma 7, per ottenere il risultato desiderato basta scrivere le seguenti istruzioni

```
>> x1 = [0 1 2]; y1 = [0 2 4]; t1 = [0 1 2]; ti1 = [0:0.01:2];
>> x2 = [0 3 4]; y2 = [0 1 4]; t2 = [0 2 3]; ti2 = [0:0.01:3]; d=[0,0];
>> six1 = cubicspline(t1,x1,ti1,0,d); siy1 = cubicspline(t1,y1,ti1,0,d);
>> six2 = cubicspline(t2,x2,ti2,0,d); siy2 = cubicspline(t2,y2,ti2,0,d);
```

La traiettoria ottenuta è stata riportata in Figura 3.13.

L'errore che si commette approssimando una funzione f (derivabile con continuità fino al quart'ordine) con la *spline* cubica interpolatoria naturale s_3 soddisfa le seguenti disuguaglianze

$$\max_{x \in I} |f^{(r)}(x) - s_3^{(r)}(x)| \leq C_r H^{4-r} \max_{x \in I} |f^{(4)}(x)|, \quad r = 0, 1, 2, 3,$$

dove $I = [x_0, x_n]$ e $H = \max_{i=0, \dots, n-1} (x_{i+1} - x_i)$, mentre C_r è una opportuna costante che dipende da r , ma non da H . È dunque evidente che non solo f , ma anche le sue derivate, prima, seconda e terza, vengono bene approssimate dalla funzione s_3 quando H tende a 0.

Osservazione 3.2 Le *spline* cubiche in generale non preservano eventuali proprietà di monotonia di f tra nodi adiacenti. Ad esempio, se si approssimasce l'arco di circonferenza unitario del primo quadrante usando le coppie di punti $\{(x_k = \sin(k\pi/6), y_k = \cos(k\pi/6)), \text{ per } k = 0, \dots, 3\}$, otterremmo la *spline* oscillante di Figura 3.14. In casi come questo conviene utilizzare altre tecniche di approssimazione. Ad esempio, il comando MATLAB `pchip` genera un `pchip`

interpolatore cubico composito $\Pi_3^H f$ che intercala anche la derivata prima di f nei nodi $\{x_i, i = 1, \dots, n - 1\}$ e, soprattutto, garantisce la monotonia locale dell'interpolatore stesso (si veda la Figura 3.14). Tale interpolatore, detto di Hermite, si ricava attraverso i seguenti comandi:

```
>> t = linspace(0,pi/2,4)
>> x = cos(t); y = sin(t);
>> xx = linspace(0,1,40);
>> plot(x,y,'s',xx,[pchip(x,y,xx);spline(x,y,xx)])
```

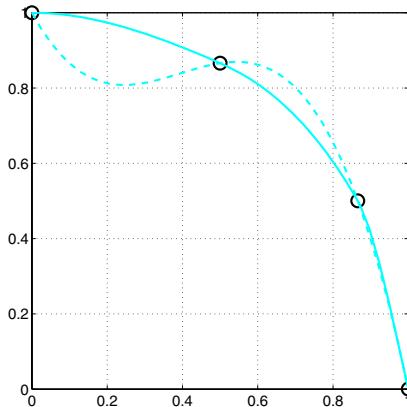


Figura 3.14. Approssimazione del primo quarto di circonferenza del cerchio unitario usando solo 4 nodi. La linea tratteggiata è il grafico della spline cubica intercalante, mentre la linea continua è il corrispondente interpolatore cubico composito di Hermite



Si vedano gli Esercizi 3.5-3.8.



3.4 Il metodo dei minimi quadrati

Abbiamo già notato che al crescere del grado del polinomio l'interpolazione polinomiale di Lagrange non garantisce una maggiore accuratezza nell'approssimazione di una funzione. Questo problema può essere superato con l'interpolazione polinomiale composita (come ad esempio quella lineare a pezzi o con funzioni spline). Essa tuttavia mal si presta ad essere utilizzata per estrarre informazioni da dati noti, cioè per generare nuove valutazioni in punti che giacciono al di fuori dell'intervallo di interpolazione.

Esempio 3.8 (Finanza) Dai dati riportati sinteticamente in Figura 3.1, siamo interessati a capire se il prezzo dell'azione tenderà a salire o scendere nei giorni immediatamente successivi all'ultima seduta di borsa. L'interpolazione polinomiale di Lagrange non è utilizzabile in pratica in quanto richiederebbe un polinomio (tremendamente oscillante) di grado 719 che conduce a predizioni fasulle. D'altra parte, l'interpolatore polinomiale composito di grado 1, il cui grafico è riportato in Figura 3.1, calcola un valore estrapolato sfruttando esclusivamente gli ultimi due valori disponibili, trascurando di conseguenza tutta la storia passata. Per ottenere il risultato cercato, rinunciamo al requisito alla base della interpolazione, procedendo come indicato nel seguito.

Supponiamo di disporre di un insieme di dati $\{(x_i, y_i), i = 0, \dots, n\}$ (dove gli y_i potrebbero eventualmente essere i valori $f(x_i)$ che una funzione assume nei nodi x_i). Cerchiamo un polinomio \tilde{f} di grado al più $m \geq 1$ (in genere, m sarà decisamente minore di n) che soddisfi la seguente disugualianza

$$\sum_{i=0}^n [y_i - \tilde{f}(x_i)]^2 \leq \sum_{i=0}^n [y_i - p_m(x_i)]^2 \quad (3.20)$$

per ogni polinomio p_m di grado al più m . Diremo che \tilde{f} (quando esiste) approssima l'insieme di dati *nel senso dei minimi quadrati*. Se $m < n$ non sarà ora più possibile garantire che $\tilde{f}(x_i) = y_i$ per $i = 0, \dots, n$.

Ponendo

$$\tilde{f}(x) = a_0 + a_1 x + \dots + a_m x^m, \quad (3.21)$$

dove i coefficienti a_0, \dots, a_m sono incogniti, il problema (3.20) si può riformulare come segue: determinare a_0, a_1, \dots, a_m tali che

$$\Phi(a_0, a_1, \dots, a_m) = \min_{\{b_i, i=0, \dots, m\}} \Phi(b_0, b_1, \dots, b_m)$$

dove

$$\Phi(b_0, b_1, \dots, b_m) = \sum_{i=0}^n [y_i - (b_0 + b_1 x_i + \dots + b_m x_i^m)]^2.$$

Risolviamo questo problema quando $m = 1$. Essendo

$$\Phi(b_0, b_1) = \sum_{i=0}^n [y_i^2 + b_0^2 + b_1^2 x_i^2 + 2b_0 b_1 x_i - 2b_0 y_i - 2b_1 x_i y_i],$$

il grafico della funzione Φ è un paraboloide convesso il cui punto di minimo (a_0, a_1) si trova imponendo le condizioni

$$\frac{\partial \Phi}{\partial b_0}(a_0, a_1) = 0, \quad \frac{\partial \Phi}{\partial b_1}(a_0, a_1) = 0,$$

dove il simbolo $\partial \Phi / \partial b_j$ denota la derivata parziale di Φ rispetto a b_j (si veda la Definizione 8.3).

Calcolando esplicitamente le due derivate parziali troviamo le seguenti 2 equazioni nelle 2 incognite a_0 ed a_1

$$\sum_{i=0}^n [a_0 + a_1 x_i - y_i] = 0, \quad \sum_{i=0}^n [a_0 x_i + a_1 x_i^2 - x_i y_i] = 0,$$

ovvero

$$\begin{aligned} a_0(n+1) + a_1 \sum_{i=0}^n x_i &= \sum_{i=0}^n y_i, \\ a_0 \sum_{i=0}^n x_i + a_1 \sum_{i=0}^n x_i^2 &= \sum_{i=0}^n y_i x_i. \end{aligned} \tag{3.22}$$

Ponendo $D = (n+1) \sum_{i=0}^n x_i^2 - (\sum_{i=0}^n x_i)^2$, la soluzione è

$$\begin{aligned} a_0 &= \frac{1}{D} \left[\sum_{i=0}^n y_i \sum_{j=0}^n x_j^2 - \sum_{j=0}^n x_j \sum_{i=0}^n x_i y_i \right], \\ a_1 &= \frac{1}{D} \left[(n+1) \sum_{i=0}^n x_i y_i - \sum_{j=0}^n x_j \sum_{i=0}^n y_i \right]. \end{aligned}$$

Il corrispondente polinomio $\tilde{f}(x) = a_0 + a_1 x$ è noto come *retta dei minimi quadrati*, o *retta di regressione*.

L'approccio precedente può essere generalizzato al caso in cui m sia un intero arbitrario (con $m < n$). Il sistema lineare quadrato di dimensione $m+1$ cui si perviene, che è simmetrico, avrà la forma seguente

$$\begin{aligned} a_0(n+1) + a_1 \sum_{i=0}^n x_i &+ \dots + a_m \sum_{i=0}^n x_i^m &= \sum_{i=0}^n y_i, \\ a_0 \sum_{i=0}^n x_i &+ a_1 \sum_{i=0}^n x_i^2 &+ \dots + a_m \sum_{i=0}^n x_i^{m+1} &= \sum_{i=0}^n x_i y_i, \\ \vdots &\vdots &\vdots &\vdots \\ a_0 \sum_{i=0}^n x_i^m &+ a_1 \sum_{i=0}^n x_i^{m+1} &+ \dots + a_m \sum_{i=0}^n x_i^{2m} &= \sum_{i=0}^n x_i^m y_i. \end{aligned}$$

Quando $m = n$, il polinomio dei minimi quadrati \tilde{f} coincide con quello prodotto dall'interpolazione polinomiale di Lagrange, $\Pi_n f$ (si veda l'Esercizio 3.9).

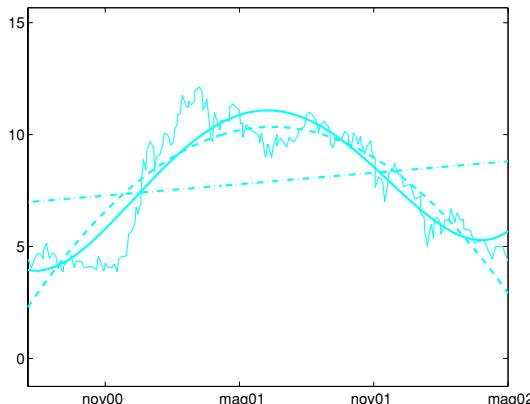


Figura 3.15. Approssimazioni nel senso dei minimi quadrati dei dati del Problema 3.2 di grado 1 (linea tratto-punto), di grado 2 (linea tratteggiata) e di grado 4 (linea continua spessa). I dati esatti del problema sono rappresentati in linea sottile

Il comando MATLAB `c=polyfit(x,y,m)` calcola *di default* i coefficienti del polinomio di grado m che approssima le $n+1$ coppie di dati $(x(i), y(i))$ nel senso dei minimi quadrati. Come già notato in precedenza, quando m è uguale a n esso calcola il polinomio interpolatore. `polyfit`

Esempio 3.9 (Finanza) In Figura 3.15 vengono riportati i grafici dei polinomi di grado 1, 2 e 4 che approssimano i dati di Figura 3.1 nel senso dei minimi quadrati. Il polinomio di grado 4 ben rappresenta l'andamento del prezzo dell'azione nel periodo di tempo considerato e suggerisce, in risposta al quesito del Problema 3.2, che, in un prossimo futuro, il valore di questo titolo possa risalire.

Esempio 3.10 (Biomeccanica) Usando il metodo dei minimi quadrati possiamo dare una risposta alla domanda del Problema 3.3 e scoprire che la linea che meglio approssima i dati dell'esperimento ha equazione $\epsilon(\sigma) = 0.3471\sigma + 0.0654$ (si veda la Figura 3.16). Di conseguenza, si trova una stima di 0.2915 per la deformazione ϵ corrispondente a $\sigma = 0.9$.

Un'ulteriore generalizzazione dell'approssimazione nel senso dei minimi quadrati consiste nell'usare funzioni di tipo non polinomiale nella (3.20). Precisamente, nel problema di minimizzazione (3.20) sia \tilde{f} che p_n sono funzioni di uno spazio V_n i cui elementi si ottengono combinando linearmente $m+1$ funzioni indipendenti $\{\psi_j, j = 0, \dots, m\}$. Esempi sono dati dalle funzioni goniometriche $\psi_j(x) = \cos(\gamma_j x)$ (per un dato parametro $\gamma \neq 0$), da quelle esponenziali $\psi_j = e^{\delta_j x}$ (per un opportuno $\delta > 0$) o da un opportuno insieme di funzioni *spline*.

La scelta del miglior insieme di funzioni $\{\psi_j\}$ è guidata generalmente da una qualche congettura sulla natura della legge che sottostà all'insieme

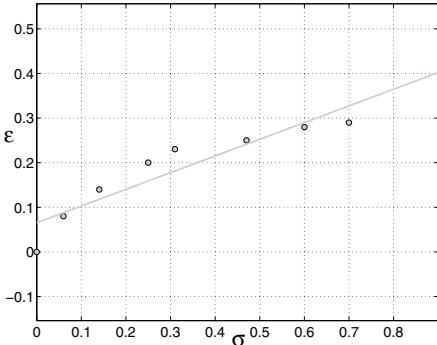


Figura 3.16. L'approssimazione lineare ai minimi quadrati per i dati del Problema 3.3

me dei dati che si vuole approssimare. Ad esempio, in Figura 3.17 abbiamo riportato il grafico dell'approssimazione nel senso dei minimi quadrati dei dati dell'Esempio 3.1 calcolata usando le funzioni goniometriche $\psi_j(x) = \cos(jt(x))$, $j = 0, \dots, 4$, con $t(x) = (2\pi/120)(x + 55)$.

Lasciamo al lettore di verificare che i coefficienti incogniti a_i che compaiono nell'espressione di \tilde{f} ,

$$\tilde{f}(x) = \sum_{j=0}^m a_j \psi_j(x),$$

sono le soluzioni del seguente sistema (di equazioni normali)

$$\mathbf{B}^T \mathbf{B} \mathbf{a} = \mathbf{B}^T \mathbf{y}$$

dove \mathbf{B} è una matrice rettangolare $(n+1) \times (m+1)$ di coefficienti $b_{ij} = \psi_j(x_i)$, \mathbf{a} è il vettore di coefficienti incogniti, mentre \mathbf{y} è il vettore dei dati.



Riassumendo

1. L'interpolatore lineare composito di una funzione f è una funzione continua e lineare a pezzi \tilde{f} , che interpola f in un dato insieme di punti $\{x_i\}$. In questo modo non si incorre nei fenomeni oscillatori del tipo di Runge quando il numero di punti cresce;
2. l'interpolazione tramite funzioni spline cubiche consente di ottenere una funzione \tilde{f} interpolatrice che sia un polinomio di grado 3 a tratti, continuo e con derivate prima e seconda continue;
3. nell'approssimazione nel senso dei minimi quadrati si cerca un polinomio \tilde{f} di grado $m < n$ tale da minimizzare la somma degli scarti quadratici $\sum_{i=0}^n [y_i - \tilde{f}(x_i)]^2$. Lo stesso criterio di minimo si può

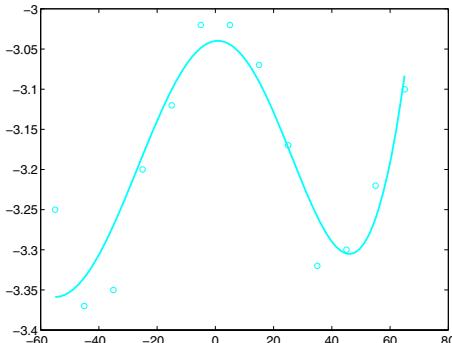


Figura 3.17. L'approssimazione nel senso dei minimi quadrati dei dati dell'Esempio 3.17 usando una base di coseni. I valori esatti sono rappresentati dai cerchietti

applicare in una classe di funzioni \tilde{f} non necessariamente di tipo polinomiale.

Si vedano gli Esercizi 3.9-3.14.



3.5 Cosa non vi abbiamo detto

Per una presentazione più generale della teoria dell'interpolazione e dell'approssimazione rimandiamo ad esempio a [Dav63], [Mei67] e [Gau97].

L'interpolazione polinomiale può essere estesa per approssimare funzioni o dati in più dimensioni. In particolare, l'interpolazione composita lineare o con funzioni spline si presta bene a questo compito a patto di sostituire la decomposizione dell'intervallo I in sotto-intervalli con una decomposizione della corrispondente regione bidimensionale Ω in poligoni (triangoli o quadrilateri) o tridimensionale (tetraedri o prismi).

Una situazione particolarmente semplice è quella in cui Ω sia di forma rettangolare o parallelepipedo. In tal caso in MATLAB si possono usare i comandi `interp2`, se Ω è un rettangolo e `interp3`, se Ω è un parallelepipedo. Entrambi questi comandi suppongono che la funzione che si vuole interpolare su una griglia regolare (ottenuta cioè come prodotto cartesiano di griglie monodimensionali) sia nota su un'altra griglia, anch'essa regolare, in generale di passo più grande.

`interp2`
`interp3`

Ad esempio, supponiamo di voler interpolare con una spline cubica i valori di $f(x, y) = \sin(2\pi x) \cos(2\pi y)$, noti su una griglia di 36 nodi con ascisse ed ordinate equispaziate sul quadrato $[0, 1]^2$ e generati con i seguenti comandi

```
>> [x,y]=meshgrid(0:0.2:1,0:0.2:1); z=sin(2*pi*x).*cos(2*pi*y);
```

La spline cubica interpolatoria, valutata su una griglia più fitta di 441 nodi (21 equispaziati in entrambe le direzioni), si ricava nel modo seguente

```
>> xi = [0:0.05:1]; yi=[0:0.05:1];
>> [xf,yf]=meshgrid(xi,yi); pi3=interp2(x,y,z,xf,yf,'spline');
```

meshgrid Il comando `meshgrid` trasforma l'insieme di tutti i punti della forma $(xi(k), yi(j))$ nelle due matrici `xf` e `yf` che possono essere utilizzate per valutare funzioni di due variabili e per effettuare grafici di superfici tridimensionali in MATLAB. Le righe della matrice `xf` sono copie del vettore `xi`, mentre le colonne della matrice `yf` sono copie del vettore `yi`.

griddata Alternativamente si può usare la funzione `griddata`, disponibile anche per dati tridimensionali (`griddata3`) o per approssimazione di superfici n -dimensionali (`griddatan`).

pdetool Se Ω è una regione bidimensionale di forma generica, se ne può ottenere una decomposizione in triangoli utilizzando l'interfaccia grafica `pdetool`.

Per una presentazione generale delle funzioni spline si veda, ad esempio, [Die93] e [PBP02]. Il toolbox MATLAB `splines` consente inoltre di esplorare svariate applicazioni delle funzioni spline. In particolare, con il comando `spdemos` vengono esemplificate le proprietà delle principali famiglie di funzioni spline. Tramite i comandi `rpmak` e `rsmak` si possono inoltre richiamare funzioni spline razionali che sono cioè date dal quoziente di due spline. Un esempio notevole di spline razionali è dato dalle cosiddette NURBS, comunemente impiegate nel CAGD (*Computer Assisted Geometric Design*).

Nel medesimo contesto dell'approssimazione di Fourier, segnaliamo le approssimazioni basate sull'uso di *ondine* (o *wavelet*), ampiamente usate nel campo della ricostruzione e della compressione delle immagini e nell'analisi di segnali (per una introduzione si vedano ad esempio [DL92], [Urb02]). Una vasta raccolta di wavelet (ed esempi di loro applicazioni) si trova nel toolbox MATLAB `wavelet`.

3.6 Esercizi

Esercizio 3.1 Si ricavi la diseguaglianza (3.6).

Esercizio 3.2 Si maggiori l'errore di interpolazione di Lagrange per le seguenti funzioni:

$$\begin{aligned} f_1(x) &= \cosh(x), & f_2(x) &= \sinh(x), & x_k &= -1 + 0.5k, & k &= 0, \dots, 4, \\ f_3(x) &= \cos(x) + \sin(x), & & & x_k &= -\pi/2 + \pi k/4, & k &= 0, \dots, 4. \end{aligned}$$

Esercizio 3.3 I dati della tabella che segue sono relativi alle aspettative di vita (in anni) per i cittadini di 2 regioni europee:

	1975	1980	1985	1990
Europa occidentale	72.8	74.2	75.2	76.4
Europa orientale	70.2	70.2	70.3	71.2

Si usi il polinomio di grado 3 che interpola questi dati per stimare le aspettative di vita nel 1970, 1983 e 1988. Si estrapoli quindi un valore per l'anno 1995. In un secondo momento, sapendo che nel 1970 l'attesa di vita per gli abitanti dell'Europa occidentale era di 71.8 anni e di 69.6 anni per quelli dell'Europa orientale, si stimi la bontà della predizione precedentemente effettuata per il 1995.

Esercizio 3.4 Il prezzo in euro di una rivista ha avuto il seguente andamento:

Nov.87	Dic.88	Nov.90	Gen.93	Gen.95	Gen.96	Nov.96	Nov.00
4.5	5.0	6.0	6.5	7.0	7.5	8.0	8.0

Si stimi il prezzo a novembre del 2001 estrapolando questi dati.

Esercizio 3.5 Si ripetano i calcoli effettuati nell'Esercizio 3.3 usando la spline cubica interpolatoria generata con il comando `spline`. Si confrontino i risultati ottenuti con i due approcci.

Esercizio 3.6 Nella tabella seguente sono riportate alcune misure della densità ρ dell'acqua di mare (in Kg/m^3) in funzione della temperatura T (in gradi Celsius)

T	4°	8°	12°	16°	20°
ρ	1000.7794	1000.6427	1000.2805	999.7165	998.9700

Si calcoli la spline cubica interpolatoria sui 4 sottointervalli dell'intervallo di temperatura $[4, 20]$. Si confronti il risultato ottenuto con i dati seguenti (che corrispondono ad ulteriori misurazioni di T):

T	6°	10°	14°	18°
ρ	1000.74088	1000.4882	1000.0224	999.3650

Esercizio 3.7 La produzione italiana di agrumi ha subito le seguenti variazioni:

anno	1965	1970	1980	1985	1990	1991
produzione ($\times 10^5$ Kg)	17769	24001	25961	34336	29036	33417

Si usino spline cubiche interpolatorie di varia natura per stimare la produzione nel 1962, nel 1977 e nel 1992 e la si confronti con la produzione reale che è stata, rispettivamente, pari a 12380, 27403 e 32059 migliaia di quintali. Si confrontino i risultati ottenuti con le *spline* con ciò che si otterrebbe usando il polinomio di interpolazione di Lagrange.

Esercizio 3.8 Si valuti la funzione $f(x) = \sin(2\pi x)$ in 21 nodi equispaziati nell'intervallo $[-1, 1]$. Si calcolino il polinomio interpolatore di Lagrange e la spline cubica interpolatoria e si confrontino i grafici di tali curve con quello di f sull'intervallo dato. Si ripetano i calcoli usando il seguente insieme di dati perturbati $\{f(x_i) + (-1)^{i+1}10^{-4}\}$ e si osservi che il polinomio interpolatore di Lagrange è più sensibile alle piccole perturbazioni di quanto non lo sia la spline cubica.

Esercizio 3.9 Si verifichi che se $m = n$ e se $y_i = f(x_i)$ (per una opportuna funzione f) allora il polinomio dei minimi quadrati nei nodi x_0, \dots, x_n coincide con $\Pi_n f$.

Esercizio 3.10 Si calcoli il polinomio di grado 4 che approssima nel senso dei minimi quadrati i dati di K riportati nelle colonne della Tabella 3.1.

Esercizio 3.11 Si ripetano i calcoli eseguiti nell'Esercizio 3.7 usando il polinomio dei minimi quadrati di grado 3.

Esercizio 3.12 Si esprimano i coefficienti del sistema (3.22) in funzione della *media* $M = \frac{1}{(n+1)} \sum_{i=0}^n x_i$, della *varianza* $v = \frac{1}{(n+1)} \sum_{i=0}^n (x_i - M)^2$ e dell'insieme di dati $\{x_i, i = 0, \dots, n\}$.

Esercizio 3.13 Si verifichi che la retta di regressione passa per il punto (\bar{x}, \bar{y}) con

$$\bar{x} = \frac{1}{n+1} \sum_{i=0}^n x_i, \quad \bar{y} = \frac{1}{n+1} \sum_{i=0}^n y_i.$$

Esercizio 3.14 I valori seguenti

portata	0	35	0.125	5	0	5	1	0.5	0.125	0
---------	---	----	-------	---	---	---	---	-----	-------	---

rappresentano le misure della portata del sangue in una sezione della carotide durante un battito cardiaco. La frequenza di acquisizione dei dati è costante e pari a $10/T$, dove $T = 1$ s è il periodo del battito. Si descrivano questi dati con una funzione continua di periodo T .

4

Differenziazione ed integrazione numerica

In questo capitolo proponiamo metodi per l'approssimazione numerica di derivate ed integrali di funzioni. Per quanto riguarda l'integrazione, non sempre si riesce a trovare in forma esplicita la primitiva di una funzione. Anche nel caso in cui la si conosca, potrebbe essere complicato valutarla. Ad esempio nel caso in cui $f(x) = \cos(4x) \cos(3 \sin(x))$, si ha

$$\int_0^\pi f(x) dx = \pi \left(\frac{3}{2}\right)^4 \sum_{k=0}^{\infty} \frac{(-9/4)^k}{k!(k+4)!};$$

il problema del calcolo di un integrale si è trasformato in quello (altrattanto problematico) della somma di una serie. Talvolta inoltre la funzione che si vuole integrare o derivare potrebbe essere nota solo per punti (rappresentanti ad esempio il risultato di una misura sperimentale), esattamente come avviene nel caso dell'approssimazione di funzioni discussa nel Capitolo 3.

In tutte queste situazioni è dunque necessario approntare metodi numerici in grado di restituire un valore approssimato della quantità di interesse, indipendentemente da quanto complessa sia la funzione da integrare o da differenziare.

Problema 4.1 (Idraulica) Ad intervalli di 5 secondi è stata misurata in metri la quota $q(t)$ raggiunta da un fluido all'interno di un cilindro retto di raggio $R = 1$ m, che presenta sul fondo un foro circolare di raggio $r = 0.1$ m, ottenendo i seguenti valori:

t	0	5	10	15	20
$q(t)$	0.6350	0.5336	0.4410	0.3572	0.2822

Si vuole fornire una stima della velocità di svuotamento $q'(t)$ del cilindro, da confrontare con quella attesa dalla legge di Torricelli: $q'(t) = -\gamma(r/R)^2 \sqrt{2gq(t)}$, dove g è il modulo dell'accelerazione di gravità e $\gamma = 0.6$

è un fattore correttivo che tiene conto della cosiddetta *strozzatura di vena*, cioè del fatto che il flusso dell'acqua che fuoriesce dall'apertura ha una sezione che è minore di quella dell'apertura stessa. Per la risoluzione di questo problema si veda l'Esempio 4.1.

Problema 4.2 (Ottica) Per il progetto di una camera a raggi infrarossi si è interessati a calcolare l'energia emessa da un corpo nero (cioè un oggetto capace di irradiare in tutto lo spettro alla temperatura ambiente) nello spettro (infrarosso) compreso tra le lunghezze d'onda $3\mu\text{m}$ e $14\mu\text{m}$. La risoluzione di questo problema si ottiene calcolando il seguente integrale

$$E(T) = 2.39 \cdot 10^{-11} \int_{3 \cdot 10^{-4}}^{14 \cdot 10^{-4}} \frac{dx}{x^5(e^{1.432/(Tx)} - 1)}, \quad (4.1)$$

che è l'equazione di Planck per l'energia E , dove x è la lunghezza d'onda (in cm) e T la temperatura (in gradi Kelvin) del corpo nero. Per il calcolo dell'integrale che compare nella (4.1) si veda l'Esercizio 4.17.

Problema 4.3 (Elettromagnetismo) Consideriamo una sfera conduttrice di raggio indefinito r e di conducibilità σ assegnata. Si vuol determinare l'andamento della densità di corrente \mathbf{j} in funzione di r e di t (il tempo), conoscendo la distribuzione iniziale della densità di corrente $\rho(r)$. Il problema si risolve utilizzando le relazioni che legano la densità di corrente, il campo elettrico e la densità di carica ed osservando che, per la simmetria del problema, $\mathbf{j}(r, t) = j(r, t)\mathbf{r}/|\mathbf{r}|$, dove $j = |\mathbf{j}|$. Si trova

$$j(r, t) = \gamma(r)e^{-\sigma t/\varepsilon_0}, \quad \gamma(r) = \frac{\sigma}{\varepsilon_0 r^2} \int_0^r \rho(\xi)\xi^2 d\xi, \quad (4.2)$$

dove $\varepsilon_0 = 8.859 \cdot 10^{-12}$ farad/m è la costante dielettrica del vuoto.

Per il calcolo di $j(r)$ si veda l'Esercizio 4.16.

Problema 4.4 (Demografia) Consideriamo una popolazione formata da un numero M grande di individui. La distribuzione $N(h)$ della loro altezza può essere rappresentata da una funzione a campana caratterizzata dal valor medio \bar{h} dell'altezza e da una deviazione standard σ ,

$$N(h) = \frac{M}{\sigma\sqrt{2\pi}} e^{-(h-\bar{h})^2/(2\sigma^2)}.$$

Allora

$$N = \int_h^{h+\Delta h} N(s) ds \quad (4.3)$$

rappresenta il numero di individui la cui altezza è compresa fra h e $h + \Delta h$ (per un $\Delta h > 0$). Riportiamo in Figura 4.1 un esempio che corrisponde ad aver preso $M = 200$ individui con $\bar{h} = 1.7$ m, $\sigma = 0.1$ m. L'area della regione ombreggiata fornisce il numero di individui la cui altezza è compresa fra 1.8 e 1.9 m.

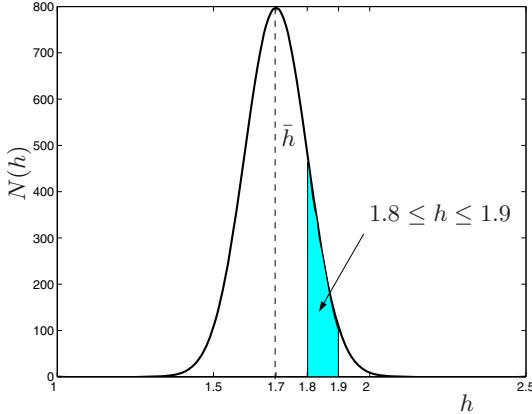


Figura 4.1. Distribuzione dell'altezza per una popolazione formata da $M = 200$ individui

4.1 Approssimazione delle derivate

Consideriamo una funzione $f : [a, b] \rightarrow \mathbb{R}$ che sia derivabile con continuità in $[a, b]$. Vogliamo approssimarne la derivata prima in un generico punto \bar{x} di (a, b) .

Grazie alla definizione (1.9), si può ritenere che, per h sufficientemente piccolo e positivo, la quantità

$$(\delta_+ f)(\bar{x}) = \frac{f(\bar{x} + h) - f(\bar{x})}{h} \quad (4.4)$$

che viene detta *differenza finita in avanti*, rappresenti una approssimazione di $f'(\bar{x})$. Per quantificare l'errore commesso, se $f \in C^2(a, b)$, è sufficiente sviluppare f in serie di Taylor, ottenendo

$$f(\bar{x} + h) = f(\bar{x}) + h f'(\bar{x}) + \frac{h^2}{2} f''(\xi), \quad (4.5)$$

dove ξ è un punto opportuno in $(\bar{x}, \bar{x} + h)$. Pertanto

$$(\delta_+ f)(\bar{x}) = f'(\bar{x}) + \frac{h}{2} f''(\xi), \quad (4.6)$$

e quindi, $(\delta_+ f)(\bar{x})$ approssima $f'(\bar{x})$ a meno di un errore che tende a 0 come h (cioè l'approssimante è accurato al prim'ordine) purché $f \in C^2(a, b)$. In maniera del tutto analoga, dal seguente sviluppo

$$f(\bar{x} - h) = f(\bar{x}) - h f'(\bar{x}) + \frac{h^2}{2} f''(\eta) \quad (4.7)$$

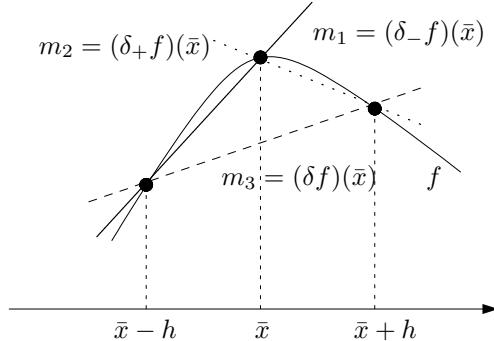


Figura 4.2. Approssimazione alle differenze finite di $f'(\bar{x})$: all'indietro (linea continua), in avanti (linea punteggiata) e centrale (linea tratteggiata). m_1 , m_2 e m_3 sono le pendenze delle rette indicate

con $\eta \in (\bar{x}-h, \bar{x})$, possiamo ottenere la seguente formula, detta *differenza finita all'indietro*

$$(\delta_- f)(\bar{x}) = \frac{f(\bar{x}) - f(\bar{x}-h)}{h} \quad (4.8)$$

sempre accurata di ordine 1, purché $f \in C^2(a, b)$. Si noti che le formule (4.4) e (4.8) si possono anche ottenere derivando il polinomio interpolatore lineare di f , calcolato sui nodi $\{\bar{x}, \bar{x}+h\}$ o $\{\bar{x}-h, \bar{x}\}$, rispettivamente. In effetti, le formule introdotte approssimano $f'(\bar{x})$ con il coefficiente angolare della retta che passa per i punti $(\bar{x}, f(\bar{x}))$ e $(\bar{x}+h, f(\bar{x}+h))$, o $(\bar{x}-h, f(\bar{x}-h))$ e $(\bar{x}, f(\bar{x}))$, rispettivamente (si veda la Figura 4.2).

Introduciamo infine la formula della *differenza finita centrale*

$$(\delta f)(\bar{x}) = \frac{f(\bar{x}+h) - f(\bar{x}-h)}{2h} \quad (4.9)$$

che è un'approssimazione del second'ordine di $f'(\bar{x})$ rispetto a h se $f \in C^3(a, b)$. Infatti, sviluppando $f(\bar{x}+h)$ e $f(\bar{x}-h)$ in serie di Taylor fino all'ordine 3 in un intorno di \bar{x} e sommando le due espressioni trovate, abbiamo

$$f'(\bar{x}) - (\delta f)(\bar{x}) = -\frac{h^2}{12}[f'''(\xi) + f'''(\eta)], \quad (4.10)$$

dove η e ξ sono punti opportuni negli intervalli $(\bar{x}-h, \bar{x})$ e $(\bar{x}, \bar{x}+h)$, rispettivamente (si veda l'Esercizio 4.1).

Quando si usa la (4.9), di fatto $f'(\bar{x})$ viene approssimata dal coefficiente angolare della retta passante per i punti $(\bar{x}-h, f(\bar{x}-h))$ e $(\bar{x}+h, f(\bar{x}+h))$.

Esempio 4.1 (Idraulica) Risolviamo il Problema 4.1, utilizzando le formule (4.4), (4.8) e (4.9) con $h = 5$ per approssimare $q'(t)$ in 5 punti diversi. Otteniamo

t	0	5	10	15	20
$q'(t)$	-0.0212	-0.0194	-0.0176	-0.0159	-0.0141
$\delta_+ q$	-0.0203	-0.0185	-0.0168	-0.0150	--
$\delta_- q$	--	-0.0203	-0.0185	-0.0168	-0.0150
δq	--	-0.0194	-0.0176	-0.0159	--

Come si vede l'accordo fra la derivata esatta e quella calcolata con le formule alle differenze finite con $h = 5$ è più soddisfacente quando si usi la (4.9) rispetto alle (4.8) o (4.4).

In generale possiamo supporre che siano disponibili le valutazioni di una certa funzione f in $n + 1$ punti equispaziati $x_i = x_0 + ih$, per $i = 0, \dots, n$ con $h > 0$. Nella derivazione numerica sostituiremo a $f'(x_i)$ una qualunque delle sue approssimazioni (4.4), (4.8) o (4.9) con $\bar{x} = x_i$.

Va osservato che la formula centrata (4.9) è applicabile nei soli punti x_1, \dots, x_{n-1} e non nei punti estremi x_0 e x_n . In questi ultimi punti si possono usare le formule modificate

$$\begin{aligned} \frac{1}{2h} [-3f(x_0) + 4f(x_1) - f(x_2)] &\quad \text{in } x_0, \\ \frac{1}{2h} [3f(x_n) - 4f(x_{n-1}) + f(x_{n-2})] &\quad \text{in } x_n, \end{aligned} \tag{4.11}$$

ancora del second'ordine rispetto a h . Esse sono state ottenute calcolando nel punto x_0 (rispettivamente, x_n) la derivata prima del polinomio interpolatore di f di grado 2 relativo ai nodi x_0, x_1, x_2 (rispettivamente, x_{n-2}, x_{n-1}, x_n).

Si vedano gli Esercizi 4.1-4.4.



4.2 Integrazione numerica

In questo paragrafo introduciamo metodi numerici adatti per approssimare l'integrale

$$I(f) = \int_a^b f(x) dx,$$

essendo f un'arbitraria funzione continua in $[a, b]$. Ricaveremo prima alcune semplici formule, per poi osservare che esse sono parte della più ampia famiglia delle cosiddette formule di Newton-Cotes. Successivamente introdurremo le cosiddette formule Gaussiane che garantiscono il massimo grado di esattezza per un dato numero di valutazioni della funzione f .

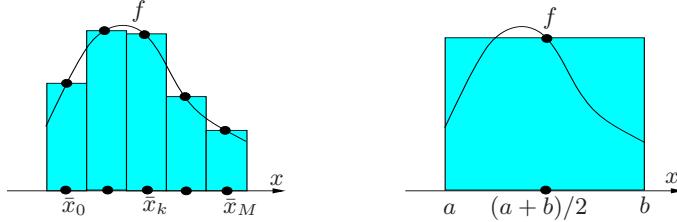


Figura 4.3. Formule del punto medio composito (a sinistra) e del punto medio (a destra)

4.2.1 La formula del punto medio

Una semplice procedura per approssimare $I(f)$ consiste nel suddividere l'intervallo $[a, b]$ in sottointervalli $I_k = [x_{k-1}, x_k]$, $k = 1, \dots, M$, con $x_k = a + kH$, $k = 0, \dots, M$, $H = (b - a)/M$. Poiché

$$I(f) = \sum_{k=1}^M \int_{I_k} f(x) dx, \quad (4.12)$$

su ogni sotto-intervallo I_k si sostituisce l'integrale di f con l'integrale di un polinomio \bar{f} che approssimi f su I_k . La soluzione più semplice consiste nello scegliere \bar{f} come il polinomio costante che interpola f nel punto medio dell'intervallo I_k

$$\bar{x}_k = \frac{x_{k-1} + x_k}{2}.$$

In tal modo si ottiene la formula di *quadratura composita del punto medio*

$$I_{pm}^c(f) = H \sum_{k=1}^M f(\bar{x}_k) \quad (4.13)$$

Il pedice *pm* sta per punto medio, mentre l'apice *c* sta per composita. Essa è accurata al second'ordine rispetto a H , più precisamente se f è derivabile con continuità in $[a, b]$ fino al second'ordine, si ha

$$I(f) - I_{pm}^c(f) = \frac{b-a}{24} H^2 f''(\xi), \quad (4.14)$$

dove ξ è un opportuno punto in $[a, b]$ (si veda l'Esercizio 4.6). La formula (4.13) è anche nota come formula di quadratura *composita del rettangolo* per la sua interpretazione geometrica, che è evidente in Figura 4.3.

La *formula del punto medio* classica (nota anche come *formula del rettangolo*) si ottiene prendendo $M = 1$ nella (4.13), ovvero usando la formula del punto medio direttamente sull'intervallo (a, b)

$$I_{pm}(f) = (b - a)f[(a + b)/2] \quad (4.15)$$

L'errore ora è dato da

$$I(f) - I_{pm}(f) = \frac{(b - a)^3}{24} f''(\xi), \quad (4.16)$$

dove ξ è un opportuno punto in (a, b) .

La (4.16) segue come caso particolare della (4.14), ma può anche essere dimostrata direttamente osservando che, posto $\bar{x} = (a + b)/2$, si ha

$$\begin{aligned} I(f) - I_{pm}(f) &= \int_a^b [f(x) - f(\bar{x})] dx \\ &= \int_a^b f'(\bar{x})(x - \bar{x}) dx + \frac{1}{2} \int_a^b f''(\eta(x))(x - \bar{x})^2 dx, \end{aligned}$$

essendo $\eta(x)$ un punto opportuno compreso fra x e \bar{x} . La (4.16) segue in quanto $\int_a^b (x - \bar{x}) dx = 0$ ed inoltre, per il teorema della media integrale, $\exists \xi \in [a, b]$ tale che

$$\frac{1}{2} \int_a^b f''(\eta(x))(x - \bar{x})^2 dx = \frac{1}{2} f''(\xi) \int_a^b (x - \bar{x})^2 dx = \frac{(b - a)^3}{24} f''(\xi).$$

Il *grado di esattezza* di una formula di quadratura è il più grande intero $r \geq 0$ per il quale l'integrale approssimato (prodotto dalla formula di quadratura) di un polinomio generico di grado r è uguale all'integrale esatto. Come si deduce dalle (4.14) e (4.16), le formule del punto medio hanno grado di esattezza 1 in quanto integrano esattamente tutti i polinomi di grado minore od uguale a 1 (ma non tutti quelli di grado 2).

La formula composita del punto medio è stata implementata nel Programma 8. I parametri d'ingresso sono gli estremi dell'intervallo di integrazione **a** e **b**, il numero di sottointervalli **M** e la *function f* che contiene l'espressione della funzione integranda *f*.

Programma 8 - midpointc : formula composita del punto medio

```
function Imp=midpointc(a,b,M,f,varargin)
%MIDPOINTC Formula composita del punto medio.
% IMP = MIDPOINTC(A,B,M,FUN) calcola un'approssimazione dell'integrale
% della funzione FUN tramite la formula composita del punto medio (su M
% intervalli equispaziati). FUN e' una function che riceve in ingresso un vettore x
% e restituisce un vettore reale. FUN puo' essere una inline function.
% IMP = MIDPOINT(A,B,M,FUN,P1,P2,...) passa alla function FUN i parametri
```



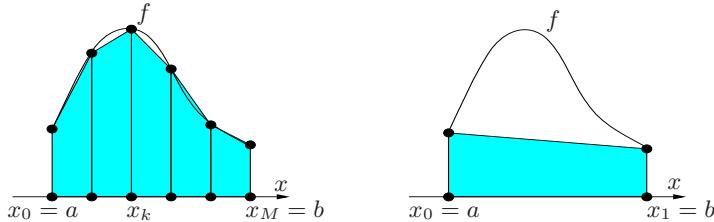


Figura 4.4. Formule del trapezio composita (a sinistra) e del trapezio (a destra)

```
% opzionali P1,P2,... come FUN(X,P1,P2,...).
H=(b-a)/M;
x = linspace(a+H/2,b-H/2,M);
fmp=feval(f,x,varargin{:}).*ones(1,M);
Imp=H*sum(fmp);
return
```



Si vedano gli Esercizi 4.5-4.8.

4.2.2 La formula del trapezio

Si può ottenere un'altra formula di quadratura sostituendo su ogni I_k con il suo interpolatore lineare composito nei nodi x_{k-1} e x_k (equivolentemente, sostituendo in $[a, b]$ f con l'interpolatore lineare composito $\Pi_1^H f$, si veda il paragrafo 3.2). Si perviene alla formula seguente, detta *formula del trapezio composita*

$$\begin{aligned} I_t^c(f) &= \frac{H}{2} \sum_{k=1}^M [f(x_{k-1}) + f(x_k)] \\ &= \frac{H}{2} [f(a) + f(b)] + H \sum_{k=1}^{M-1} f(x_k) \end{aligned} \tag{4.17}$$

Essa è accurata al second'ordine rispetto a H , più precisamente

$$I(f) - I_t^c(f) = -\frac{b-a}{12} H^2 f''(\xi) \tag{4.18}$$

per un opportuno $\xi \in [a, b]$, purché $f \in C^2([a, b])$. Utilizzando (4.17) con $M = 1$, si trova la formula

$$I_t(f) = \frac{b-a}{2} [f(a) + f(b)] \tag{4.19}$$

detta *formula del trapezio* per via della sua interpretazione geometrica (si veda la Figura 4.4). L'errore che si commette vale

$$I(f) - I_t(f) = -\frac{(b-a)^3}{12} f''(\xi), \quad (4.20)$$

con ξ opportuno in $[a, b]$. Si deduce che (4.19) ha grado di esattezza uguale ad 1, come la formula del punto medio.

La formula composita del trapezio (4.17) è implementata nei programmi MATLAB `trapz` e `cumtrapz`. In particolare, se indichiamo con \mathbf{x} il vettore con componenti gli x_k e con \mathbf{y} il vettore delle $f(x_k)$, $\mathbf{z}=\text{cumtrapz}(\mathbf{x}, \mathbf{y})$ restituisce un vettore \mathbf{z} che ha come componenti i valori $z_k = \int_a^{x_k} f(x) dx$, approssimati con la formula composita del trapezio. Di conseguenza, il valore dell'integrale tra a e b è contenuto nell'ultima componente di \mathbf{z} .

`trapz`
`cumtrapz`

Si vedano gli Esercizi 4.9-4.11.



4.2.3 La formula di Simpson

La formula di Simpson si ottiene sostituendo su ogni I_k l'integrale di f con quello del polinomio interpolatore di grado 2 di f relativo ai nodi x_{k-1} , $\bar{x}_k = (x_{k-1} + x_k)/2$ e x_k

$$\begin{aligned} I_2 f(x) &= \frac{2(x - \bar{x}_k)(x - x_k)}{H^2} f(x_{k-1}) \\ &+ \frac{4(x_{k-1} - x)(x - x_k)}{H^2} f(\bar{x}_k) + \frac{2(x - \bar{x}_k)(x - x_{k-1})}{H^2} f(x_k). \end{aligned}$$

La formula risultante è nota come la *formula di quadratura composita di Simpson*, ed è data da

$$I_s^c(f) = \frac{H}{6} \sum_{k=1}^M [f(x_{k-1}) + 4f(\bar{x}_k) + f(x_k)] \quad (4.21)$$

Si può dimostrare che essa introduce un errore pari a

$$I(f) - I_s^c(f) = -\frac{b-a}{180} \frac{H^4}{16} f^{(4)}(\xi), \quad (4.22)$$

dove ξ è un punto opportuno in $[a, b]$, purché $f \in C^4([a, b])$. Si tratta quindi di una formula accurata di ordine 4 rispetto a H . Quando (4.21) viene applicata ad un solo intervallo (a, b) , otteniamo la *formula di quadratura di Simpson*

$$I_s(f) = \frac{b-a}{6} [f(a) + 4f((a+b)/2) + f(b)] \quad (4.23)$$

L'errore ora vale

$$I(f) - I_s(f) = -\frac{1}{16} \frac{(b-a)^5}{180} f^{(4)}(\xi), \quad (4.24)$$

per un opportuno $\xi \in [a, b]$. Il grado di esattezza è quindi uguale a 3.

La formula composita di Simpson è implementata nel Programma 9.



Programma 9 - simpsonc : formula composita di Simpson

```
function [Isic]=simpsonc(a,b,M,f,varargin)
%SIMPSONC Formula composita di Simpson
% ISIC = SIMPONC(A,B,M,FUN) calcola un'approssimazione dell'integrale
% della funzione FUN tramite la formula composita di Simpson (su M
% intervalli equispaziati). FUN e' una function che riceve in ingresso un vettore x
% e restituisce un vettore reale. FUN puo' essere una inline function.
% ISIC = SIMPONC(A,B,M,FUN,P1,P2,...) passa alla function FUN i parametri
% opzionali P1,P2,... come FUN(X,P1,P2,...).
H=(b-a)/M;
x=linspace(a,b,M+1);
fpm=feval(f,x,varargin{:}).*ones(1,M+1);
fpm(2:end-1) = 2*fpm(2:end-1);
Isic=H*sum(fpm)/6;
x=linspace(a+H/2,b-H/2,M);
fpm=feval(f,x,varargin{:}).*ones(1,M);
Isic = Isic+2*H*sum(fpm)/3;
return
```

Esempio 4.2 (Demografia) Consideriamo il Problema 4.4. Per determinare il numero di individui la cui altezza è compresa fra 1.8 e 1.9 m, dobbiamo calcolare l'integrale (4.3) per $h = 1.8$ e $\Delta h = 0.1$. Usiamo la formula composita di Simpson con 100 sotto-intervalli:

```
>> N = inline('M/(sigma * sqrt(2*pi)) * exp(-(h - hbar).^2./(2*sigma.^2))',...
    'h', 'M', 'hbar', 'sigma');
>> M = 200; hbar = 1.7; sigma = 0.1;
>> int = simpsonc(1.8, 1.9, 100, N, M, hbar, sigma)
ans =
27.1810
```

Si stima quindi che il numero di individui con altezza nell'intervallo indicato è 27.1810, corrispondente al 15.39 % della popolazione.

Esempio 4.3 Vogliamo confrontare le approssimazioni dell'integrale $I(f) = \int_0^{2\pi} xe^{-x} \cos(2x) dx = -1/25(10\pi - 3 + 3e^{2\pi})/e^{2\pi} \simeq -0.122122604618968$ ottenute usando le formule composite del punto medio, del trapezio e di Simpson. In Figura 4.5 riportiamo in scala logaritmica l'andamento degli errori in funzione di H .

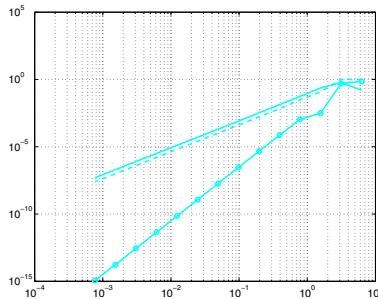


Figura 4.5. Rappresentazione in scala logaritmica degli errori (rispetto a H) per le formule composite di Simpson (linea piena con cerchietti), del punto medio (linea piena) e del trapezio (linea tratteggiata)

Come osservato nel paragrafo 1.5, in questo tipo di grafici a rette di pendenza maggiore corrispondono metodi di ordine più elevato. Come previsto dalla teoria le formule composite del punto medio e del trapezio sono accurate di ordine 2, mentre quella di Simpson è di ordine 4.

4.3 Formule di quadratura interpolatorie

Tutte le formule di quadratura che abbiamo precedentemente introdotto sono esempi notevoli della forma generale

$$I_{appr}(f) = \sum_{j=0}^n \alpha_j f(y_j) \quad (4.25)$$

I numeri reali $\{\alpha_j\}$ sono detti *pesi*, mentre i punti $\{y_j\}$ sono detti *nodi*. Limitiamoci al caso di formule non composite, ovvero corrispondenti alla scelta $M = 1$, come ad esempio nelle formule (4.15), (4.19) e (4.23). In generale, si richiede che la formula (4.25) integri esattamente almeno le funzioni costanti: questa proprietà è garantita se $\sum_{j=0}^n \alpha_j = b - a$. Avremo invece sicuramente un grado di esattezza (almeno) pari a n se

$$I_{appr}(f) = \int_a^b \Pi_n f(x) dx,$$

dove $\Pi_n f \in \mathbb{P}_n$ è il polinomio interpolatore di Lagrange di una funzione f nei nodi y_i , $i = 0, \dots, n$, dato nella (3.4). I pesi avranno di conseguenza la seguente espressione

$$\alpha_i = \int_a^b \varphi_i(x) dx, \quad i = 0, \dots, n,$$

dove $\varphi_i \in \mathbb{P}_n$ è l' i -esimo polinomio caratteristico di Lagrange tale che $\varphi_i(y_j) = \delta_{ij}$, per $i, j = 0, \dots, n$, definito nella (3.3).

Esempio 4.4 Per la formula del trapezio (4.19) abbiamo $n = 1$, $y_0 = a$, $y_1 = b$ e

$$\alpha_0 = \int_a^b \varphi_0(x) dx = \int_a^b \frac{x-b}{a-b} dx = \frac{b-a}{2},$$

$$\alpha_1 = \int_a^b \varphi_1(x) dx = \int_a^b \frac{x-a}{b-a} dx = \frac{b-a}{2}.$$

La domanda che ci poniamo ora è se sia possibile determinare una formula di quadratura interpolatoria che, grazie ad una opportuna scelta dei nodi, abbia un grado di esattezza maggiore di n , precisamente pari a $r = n + m$ per un opportuno $m > 0$. Per semplicità restringiamo la nostra discussione all'intervallo di riferimento $(-1, 1)$. Una volta determinato un insieme di nodi di quadratura $\{\bar{y}_j\}$ (e, conseguentemente di pesi $\{\bar{\alpha}_j\}$) sull'intervallo $(-1, 1)$, utilizzando il cambio di variabile (3.8) troveremo immediatamente i corrispondenti nodi e pesi su un intervallo (a, b) generico,

$$y_j = \frac{a+b}{2} + \frac{b-a}{2}\bar{y}_j, \quad \alpha_j = \frac{b-a}{2}\bar{\alpha}_j.$$

La risposta al nostro quesito è contenuta nel risultato che segue (per la cui dimostrazione rimandiamo a [QSS04, Capitolo 10]):

Proposizione 4.1 *Per un dato $m > 0$, la formula di quadratura $\sum_{j=0}^n \bar{\alpha}_j f(\bar{y}_j)$ ha grado di esattezza $n + m$ se e soltanto se è di tipo interpolatorio e se il polinomio nodale $\omega_{n+1}(x) = \prod_{i=0}^n (x - \bar{y}_i)$ associato ai nodi $\{\bar{y}_i\}$ è tale che*

$$\int_{-1}^1 \omega_{n+1}(x) p(x) dx = 0, \quad \forall p \in \mathbb{P}_{m-1}. \quad (4.26)$$

Si può dimostrare che il valore massimo che m può assumere è $n + 1$ e viene raggiunto quando ω_{n+1} è proporzionale al cosiddetto polinomio di Legendre di grado $n + 1$, $L_{n+1}(x)$. I polinomi di Legendre sono calcolabili ricorsivamente tramite la seguente relazione a tre termini

$$L_0(x) = 1, \quad L_1(x) = x,$$

$$L_{k+1}(x) = \frac{2k+1}{k+1} x L_k(x) - \frac{k}{k+1} L_{k-1}(x), \quad k = 1, 2, \dots,$$

n	$\{\bar{y}_j\}$	$\{\bar{\alpha}_j\}$
1	$\{\pm 1/\sqrt{3}\}$	$\{1\}$
2	$\{\pm \sqrt{15}/5, 0\}$	$\{5/9, 8/9\}$
3	$\left\{ \pm(1/35)\sqrt{525 - 70\sqrt{30}}, \pm(1/35)\sqrt{525 + 70\sqrt{30}} \right\}$	$\{(1/36)(18 + \sqrt{30}), (1/36)(18 - \sqrt{30})\}$
4	$\left\{ 0, \pm(1/21)\sqrt{245 - 14\sqrt{70}}, \pm(1/21)\sqrt{245 + 14\sqrt{70}} \right\}$	$\{128/225, (1/900)(322 + 13\sqrt{70}) (1/900)(322 - 13\sqrt{70})\}$

Tabella 4.1. Nodi e pesi di alcune formule di quadratura di Gauss-Legendre sull'intervallo $(-1, 1)$. I pesi corrispondenti a coppie di nodi simmetrici rispetto allo 0 vengono riportati una volta sola

e si può dimostrare che un qualsiasi polinomio di grado minore od uguale a n può essere scritto come una combinazione lineare dei polinomi di Legendre L_0, L_1, \dots, L_n . Si può inoltre verificare che L_{n+1} è ortogonale a tutti i polinomi di grado minore od uguale a n nel senso che $\int_{-1}^1 L_{n+1}(x)L_j(x) dx = 0$ per $j = 0, \dots, n$ e, di conseguenza, la (4.26) risulta verificata. Il massimo grado di esattezza conseguibile è quindi pari a $2n + 1$, e si ottiene con la cosiddetta formula di Gauss-Legendre (in breve I_{GL}) i cui nodi e pesi sono:

$$\begin{cases} \bar{y}_j \text{ zero di } L_{n+1}(x), \\ \bar{\alpha}_j = \frac{2}{(1 - \bar{y}_j^2)[L'_{n+1}(\bar{y}_j)]^2}, j = 0, \dots, n. \end{cases} \quad (4.27)$$

I pesi $\bar{\alpha}_j$ sono tutti positivi ed i nodi sono tutti interni all'intervallo $(-1, 1)$. In Tabella 4.1 riportiamo i nodi ed i pesi delle formule di quadratura di Gauss-Legendre per $n = 1, 2, 3, 4$.

Se $f \in C^{(2n+2)}([-1, 1])$, l'errore corrispondente è dato da

$$I(f) - I_{GL}(f) = \frac{2^{2n+3}((n+1)!)^4}{(2n+3)((2n+2)!)^3} f^{(2n+2)}(\xi),$$

dove ξ è un opportuno punto in $[-1, 1]$.

Spesso è utile includere tra i nodi di quadratura i punti estremi dell'intervallo di integrazione. In tal caso, la formula con il massimo grado di esattezza (pari a $2n - 1$) è quella che usa i cosiddetti nodi di *Gauss-Legendre-Lobatto* (in breve GLL): per $n \geq 1$

$$\bar{y}_0 = -1, \bar{y}_n = 1, \bar{y}_j \text{ zero di } L'_n(x), j = 1, \dots, n-1, \quad (4.28)$$

$$\bar{\alpha}_j = \frac{2}{n(n+1)} \frac{1}{[L_n(\bar{y}_j)]^2}, \quad j = 0, \dots, n.$$

n	$\{\bar{y}_j\}$	$\{\bar{\alpha}_j\}$
1	$\{\pm 1\}$	$\{1\}$
2	$\{\pm 1, 0\}$	$\{1/3, 4/3\}$
3	$\{\pm 1, \pm \sqrt{5}/5\}$	$\{1/6, 5/6\}$
4	$\{\pm 1, \pm \sqrt{21}/7, 0\}$	$\{1/10, 49/90, 32/45\}$

Tabella 4.2. Nodi e pesi di alcune formule di quadratura di Gauss-Legendre-Lobatto sull'intervallo $[-1, 1]$. I pesi corrispondenti a coppie di nodi simmetrici rispetto allo 0 vengono riportati una volta sola

Se $f \in C^{(2n)}([-1, 1])$, l'errore corrispondente è pari a

$$I(f) - I_{GLL}(f) = -\frac{(n+1)n^{32}2^{2n+1}((n-1)!)^4}{(2n+1)((2n)!)^3} f^{(2n)}(\xi),$$

per un opportuno $\xi \in [-1, 1]$. In Tabella 4.2 riportiamo i valori dei nodi e dei pesi delle formule di Gauss-Legendre-Lobatto sull'intervallo di riferimento $[-1, 1]$ per $n = 1, 2, 3, 4$ (per $n = 1$ si trova la formula del trapezio).

quadl

Usando il comando MATLAB `quadl(fun,a,b)` è possibile approssimare un integrale con una formula di quadratura *composita* di Gauss-Lobatto. La funzione da integrare deve essere precisata in input in una *function* (che può essere anche una *inline function*). Ad esempio, per integrare $f(x) = 1/x$ in $[1, 2]$, definiamo prima la seguente *function* `fun`

```
fun=inline('1./x','x');
```

per poi eseguire `quadl(fun,1,2)`. Si noti che nella definizione di `fun` abbiamo fatto uso di un'operazione elemento per elemento: in effetti `quadl` valuterà l'espressione specificata in `fun` su di un vettore di nodi di quadratura.

Come si vede, nel richiamare `quadl` non abbiamo specificato il numero di intervalli di quadratura da utilizzare nella formula composita, né, conseguentemente, la loro ampiezza H . Tale decomposizione viene automaticamente calcolata in modo che l'errore di quadratura si mantenga al di sotto di una tolleranza prefissata (pari di default a 10^{-3}). Con il comando `quadl(fun,a,b,tol)` si può precisare una tolleranza `tol` diversa. Nel paragrafo 4.4 introdurremo un metodo per stimare l'errore di quadratura e, conseguentemente, per cambiare H in modo adattivo.



Riassumendo

1. Una formula di quadratura calcola in modo approssimato l'integrale di una funzione continua f su un intervallo $[a, b]$;
2. essa è generalmente costituita dalla combinazione lineare dei valori di f in determinati punti (detti *nodi di quadratura*) moltiplicati per opportuni coefficienti (detti *pesi di quadratura*);

3. il *grado di esattezza* di una formula di quadratura è il grado più alto dei polinomi che vengono integrati esattamente dalla formula stessa. Tale grado è pari a 1 per le formule del punto medio e del trapezio, a 3 per la formula di Simpson, a $2n + 1$ per la formula di Gauss-Legendre con $n + 1$ nodi di quadratura e a $2n - 1$ per la formula di Gauss-Legendre-Lobatto con $n + 1$ nodi di quadratura;
4. una formula di quadratura composita ha *ordine di accuratezza* p se l'errore tende a zero per H che tende a zero come H^p , dove H è l'ampiezza dei sotto-intervalli. L'ordine di accuratezza è pari a 2 per le formule composite del punto medio e del trapezio, a 4 per la formula composita di Simpson.

Si risolvano gli Esercizi 4.12-4.18.



4.4 La formula di Simpson adattiva



Il passo di integrazione H di una formula di quadratura composita può essere scelto in modo da garantire che l'errore sia inferiore ad una tolleranza $\varepsilon > 0$ prestabilita. A tal fine se usassimo ad esempio la formula di Simpson composita (4.21), grazie alla (4.22) basterebbe richiedere che

$$\frac{b-a}{180} \frac{H^4}{16} \max_{x \in [a,b]} |f^{(4)}(x)| < \varepsilon, \quad (4.29)$$

dove $f^{(4)}$ denota al solito la derivata quarta di f . D'altra parte, se $f^{(4)}$ è in valore assoluto grande solo in una piccola porzione dell'intervallo di integrazione, il più grande valore di H per il quale la (4.29) è soddisfatta sarà presumibilmente troppo piccolo. L'obiettivo della formula di Simpson adattiva è quello di calcolare un'approssimazione di $I(f)$ a meno di una tolleranza ε fissata facendo uso di una distribuzione *non uniforme* dei sotto-intervalli nell'intervallo $[a, b]$. In tal modo si garantisce la stessa accuratezza della formula composita con nodi equispaziati, ma con un numero inferiore di intervalli (e, quindi, di valutazioni di f).

Per implementare un algoritmo adattivo serviranno uno stimatore dell'errore di quadratura ed una procedura che modifichi, conseguentemente al soddisfacimento della tolleranza richiesta, il passo di integrazione H . Analizziamo dapprima il secondo punto, che è indipendente dalla formula di quadratura usata.

Al primo passo della procedura adattiva, calcoliamo una approssimazione $I_s(f)$ di $I(f) = \int_a^b f(x) dx$. Poniamo $H = b - a$ e cerchiamo di stimare l'errore di quadratura. Se l'errore è minore della tolleranza richiesta, la procedura adattiva si arresta, in caso contrario si dimezza il passo di integrazione H finché non si calcola l'integrale $\int_a^{a+H} f(x) dx$ con l'accuracy desiderata. A questo punto si considera l'intervallo $(a + H, b)$

e si ripete la procedura, scegliendo come primo passo di integrazione la lunghezza $b - (a + H)$ dell'intervallo di integrazione.

Introduciamo le seguenti notazioni:

1. A : l'intervallo di integrazione *attivo* cioè quell'intervallo sul quale stiamo effettivamente approssimando l'integrale;
2. S : l'intervallo di integrazione *già esaminato* nel quale sappiamo che l'errore commesso sta al di sotto della tolleranza richiesta;
3. N : l'intervallo di integrazione *ancora da esaminare*.

All'inizio del processo di integrazione abbiamo $N = [a, b]$, $A = N$ e $S = \emptyset$, mentre ad un passo intermedio avremo una situazione analoga a quella descritta nella Figura 4.6. Indichiamo con $J_S(f)$ l'approssimazione calcolata di $\int_a^\alpha f(x) dx$ (avendo posto $J_S(f) = 0$ all'inizio del processo). Se l'algoritmo termina con successo $J_S(f)$ fornirà l'approssimazione cercata di $I(f)$. Indichiamo inoltre con $J_{(\alpha, \beta)}(f)$ l'integrale approssimato di f sull'intervallo attivo $[\alpha, \beta]$, in bianco in Figura 4.6. Il generico passo del metodo di integrazione adattivo viene realizzato come segue:

1. se la stima dell'errore garantisce che esso sia inferiore alla tolleranza richiesta, allora:
 - (i) $J_S(f)$ viene incrementato di $J_{(\alpha, \beta)}(f)$, ossia $J_S(f) \leftarrow J_S(f) + J_{(\alpha, \beta)}(f)$;
 - (ii) poniamo $S \leftarrow S \cup A$, $A = N$ (corrispondente al cammino (I) in Figura 4.6) e $\alpha \leftarrow \beta$, $\beta \leftarrow b$;
2. se la stima dell'errore non ha l'accuratezza richiesta, allora:
 - (j) A viene dimezzato ed il nuovo intervallo attivo viene posto pari a $A = [\alpha, \alpha']$ con $\alpha' = (\alpha + \beta)/2$ (corrispondente al cammino (II) in Figura 4.6);
 - (jj) poniamo $N \leftarrow N \cup [\alpha', \beta]$, $\beta \leftarrow \alpha'$;
 - (jjj) si stima nuovamente l'errore.

Naturalmente, per evitare che l'algoritmo proposto generi passi di integrazione troppo piccoli, conviene controllare la lunghezza di A ed avvertire l'utilizzatore qualora tale grandezza scenda al di sotto di un valore di soglia (questo potrebbe accadere ad esempio in un intorno di una singolarità della funzione integranda).

Resta ora da scegliere un opportuno stimatore dell'errore. A tal fine, poniamoci su un generico sotto-intervallo di integrazione $[\alpha, \beta]$ e calcoliamo $I_s(f)$ su $[\alpha, \beta] \subset [a, b]$: evidentemente, se su tale generico intervallo l'errore sarà minore di $\varepsilon(\beta - \alpha)/(b - a)$, allora l'errore su tutto $[a, b]$ sarà minore della tolleranza assegnata ε . Poiché dalla (4.24) segue che

$$E_s(f; \alpha, \beta) = \int_{\alpha}^{\beta} f(x) dx - I_s(f) = -\frac{(\beta - \alpha)^5}{2880} f^{(4)}(\xi),$$

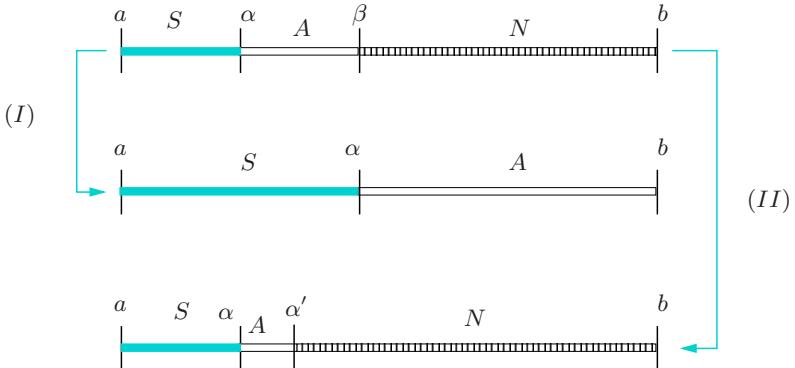


Figura 4.6. Distribuzione degli intervalli di integrazione ad un passo intermedio del processo di integrazione adattiva

per assicurarsi il raggiungimento della accuratezza desiderata basterà richiedere che $E_s(f; \alpha, \beta)$ sia minore di $\varepsilon(\beta - \alpha)/(b - a)$. In pratica questa richiesta non è semplice da soddisfare perché il punto ξ di $[\alpha, \beta]$ è sconosciuto.

Per stimare l'errore $E_s(f; \alpha, \beta)$ senza ricorrere esplicitamente al valore di $f^{(4)}(\xi)$, usiamo ora la formula di quadratura composita di Simpson per calcolare $\int_{\alpha}^{\beta} f(x) dx$, ma con passo $H = (\beta - \alpha)/2$. Per la (4.22) con $a = \alpha$ e $b = \beta$, troviamo che

$$\int_{\alpha}^{\beta} f(x) dx - I_s^c(f) = -\frac{(\beta - \alpha)^5}{46080} f^{(4)}(\eta), \quad (4.30)$$

per un opportuno η diverso da ξ . Sottraendo membro a membro le due ultime equazioni, si trova allora

$$\Delta I = I_s^c(f) - I_s(f) = -\frac{(\beta - \alpha)^5}{2880} f^{(4)}(\xi) + \frac{(\beta - \alpha)^5}{46080} f^{(4)}(\eta). \quad (4.31)$$

Assumiamo ora che $f^{(4)}(x)$ sia approssimativamente costante sull'intervallo $[\alpha, \beta]$. In tal caso, $f^{(4)}(\xi) \approx f^{(4)}(\eta)$. Ricavando $f^{(4)}(\eta)$ dalla (4.31) e sostituendolo nella (4.30), si trova la seguente stima dell'errore:

$$\int_{\alpha}^{\beta} f(x) dx - I_s^c(f) \approx \frac{1}{15} \Delta I.$$

Il passo di integrazione $(\beta - \alpha)/2$ (quello impiegato per il calcolo di $I_s^c(f)$) verrà allora accettato se $|\Delta I|/15 < \varepsilon(\beta - \alpha)/[2(b - a)]$ (la divisione per 2 è fatta per via cautelativa). La formula che combina questo criterio

sul passo con il processo adattivo descritto in precedenza, prende il nome di *formula di Simpson adattiva*.

Essa è stata implementata nel Programma 10 nel quale **f** è la *function* che precisa la funzione integranda, **a** e **b** sono gli estremi dell'intervallo di integrazione, **tol** la tolleranza richiesta sull'errore e **hmin** il minimo passo di integrazione consentito (per evitare che il processo di dimezzamento del passo continui indefinitamente).



Programma 10 - simpadpt : formula di Simpson adattiva

```

function [JSf,nodes]=simpadpt(f,a,b,tol,hmin,varargin)
%SIMPADPT Formula adattiva di Simpson.
% JSF = SIMPADPT(FUN,A,B,TOL,HMIN) approssima l'integrale di FUN
% nell'intervallo (A,B) garantendo che il valore assoluto dell'errore sia
% inferiore a TOL. FUN e' una function che riceve in ingresso un vettore x
% e restituisce un vettore reale. FUN puo' essere una inline function.
% JSF = SIMPADPT(FUN,A,B,TOL,HMIN,P1,P2,...) passa alla function FUN
% i parametri opzionali P1,P2,... come FUN(X,P1,P2,...).
% [JSF,NODES] = SIMPADPT(...) restituisce la distribuzione di nodi
% usati nel processo di quadratura.
A=[a,b]; N=[]; S=[]; JSf = 0; ba = b - a; nodes=[];
while ~isempty(A),
    [deltaI,ISc]=caldeltai(A,f,varargin{:});
    if abs(deltaI) <= 15*tol*(A(2)-A(1))/ba;
        JSf = JSf + ISc; S = union(S,A);
        nodes = [nodes, A(1) (A(1)+A(2))*0.5 A(2)];
        S = [S(1), S(end)]; A = N; N = [];
    elseif A(2)-A(1) < hmin
        JSf=JSf+ISc; S = union(S,A);
        S = [S(1), S(end)]; A=N; N=[];
        warning('Passo di integrazione troppo piccolo');
    else
        Am = (A(1)+A(2))*0.5; A = [A(1) Am]; N = [Am, b];
    end
end
nodes=unique(nodes);
return

function [deltaI,ISc]=caldeltai(A,f,varargin)
L=A(2)-A(1);
t=[0; 0.25; 0.5; 0.5; 0.75; 1];
x=L*t+A(1);
L=L/6;
w=[1; 4; 1];
fx=feval(f,x,varargin{:}).*ones(6,1);
IS=L*sum(fx([1 3 6]).*w);
ISc=0.5*L*sum(fx.*[w;w]);
deltaI=IS-ISc;
return

```

Esempio 4.5 Calcoliamo $I(f) = \int_{-1}^1 e^{-10(x-1)^2} dx$ con la formula adattiva di Simpson. Eseguendo il Programma 10 con

```
>> fun=inline('exp(-10*(x-1).^2)'); tol = 1.e-04; hmin = 1.e-03;
```

troviamo il valore approssimato $I_S^A = 0.28024765884708$, mentre il valore esatto è 0.28024956081990. Il risultato ottenuto soddisfa la tolleranza richiesta, in quanto $|I(f) - I_S^A| \simeq 10^{-5}$. Si noti che per ottenere questo risultato bastano soltanto 10 sotto-intervalli non uniformi. La formula di Simpson composita con passo uniforme avrebbe richiesto circa 22 sotto-intervalli per ottenere la stessa accuratezza.

4.5 Cosa non vi abbiamo detto

Le formule del punto medio, del trapezio e di Simpson sono casi particolari di un'ampia famiglia di formule di quadratura, note come *formule di Newton-Côtes*. Per una loro presentazione rimandiamo a [QSS04, Capitolo 9]. In maniera del tutto analoga, le formule di Gauss-Legendre e di Gauss-Legendre-Lobatto sono solo esempi dell'importante famiglia di formule di quadratura Gaussiane: esse hanno la peculiarità di raggiungere il massimo grado di esattezza, una volta fissato il numero di nodi. Rimandiamo a [QSS04, Capitolo 10] o a [RR85] per la loro trattazione. Per ulteriori approfondimenti sull'integrazione numerica citiamo anche [DR75] e [PdDKÜK83].

L'integrazione numerica può essere realizzata anche per integrali su intervalli illimitati, come ad esempio per calcolare $\int_0^\infty f(x) dx$. Una possibilità consiste nel trovare un punto α tale che il valore di $\int_\alpha^\infty f(x) dx$ possa essere trascurato rispetto a quello di $\int_0^\alpha f(x) dx$; ci si limita poi a calcolare quest'ultimo con una formula di quadratura. Alternativamente si può ricorrere a formule di quadratura di Gauss per intervalli illimitati (si veda [QSS04], capitolo 10).

Infine, l'integrazione numerica può essere estesa ad integrali su domini multidimensionali. Il comando MATLAB `dblquad('f',xmin,xmax, ymin,ymax)` consente ad esempio di approssimare l'integrale di una data funzione, precisata attraverso una variabile `f`, sul dominio rettangolare `[xmin,xmax] × [ymin,ymax]`. `f` è una *function* che deve avere come parametri d'ingresso almeno le due variabili rispetto alle quali si calcola l'integrale doppio, `x` e `y`.

4.6 Esercizi

Esercizio 4.1 Si verifichi che se $f \in C^3$ in un intorno di \bar{x} l'errore della formula (4.9) è dato dalla (4.10).

Esercizio 4.2 Si verifichi che, se $f \in C^3$ in un intorno I_0 di x_0 (rispettivamente, I_n di x_n) l'errore nella formula (4.11) è pari a $-\frac{1}{3}f'''(\xi_0)h^2$ (rispettivamente, $-\frac{1}{3}f'''(\xi_n)h^2$), dove ξ_0 e ξ_n sono due punti opportuni in I_0 e I_n , rispettivamente.

Esercizio 4.3 Si ricavi l'ordine di accuratezza rispetto a h delle seguenti formule di differenziazione numerica per l'approssimazione di $f'(x_i)$:

$$\begin{aligned} a. \quad & \frac{-11f(x_i) + 18f(x_{i+1}) - 9f(x_{i+2}) + 2f(x_{i+3})}{6h}, \\ b. \quad & \frac{f(x_{i-2}) - 6f(x_{i-1}) + 3f(x_i) + 2f(x_{i+1})}{6h}, \\ c. \quad & \frac{-f(x_{i-2}) - 12f(x_i) + 16f(x_{i+1}) - 3f(x_{i+2})}{12h}. \end{aligned}$$

Esercizio 4.4 I valori seguenti rappresentano l'evoluzione al variare del tempo t del numero di individui $n(t)$ di una certa popolazione caratterizzata da un tasso di natalità costante $b = 2$ e da un tasso di mortalità $d(t) = 0.01n(t)$:

t (mesi)	0	0.5	1	1.5	2	2.5	3
n	100	147	178	192	197	199	200

Si usino questi dati per determinare il più accuratamente possibile il tasso di variazione della popolazione. Si confrontino i risultati ottenuti con la velocità teorica data da $n'(t) = 2n(t) - 0.01n^2(t)$.

Esercizio 4.5 Si calcoli il minimo numero M di intervalli necessari per approssimare, a meno di un errore di 10^{-4} , l'integrale delle seguenti funzioni negli intervalli indicati:

$$\begin{aligned} f_1(x) &= \frac{1}{1 + (x - \pi)^2} \text{ in } [0, 5], \\ f_2(x) &= e^x \cos(x) \quad \text{in } [0, \pi], \\ f_3(x) &= \sqrt{x(1 - x)} \quad \text{in } [0, 1], \end{aligned}$$

utilizzando la formula composita del punto medio. Si verifichino sperimentalmente i risultati ottenuti tramite il Programma 8.

Esercizio 4.6 Si dimostri la (4.14) a partire dalla (4.16).

Esercizio 4.7 Si giustifichi la perdita di un ordine di convergenza che si ha passando dalla formula del punto medio a quella del punto medio composita.

Esercizio 4.8 Si verifichi che se f è un polinomio di grado minore od uguale a 1, allora $I_{pm}(f) = I(f)$ cioè che la formula del punto medio ha grado di esattezza uguale ad 1.

Esercizio 4.9 Per la funzione f_1 dell’Esercizio 4.5, si valutino numericamente i valori di M che garantiscono un errore di quadratura inferiore a 10^{-4} nel caso in cui si usino le formule composite del trapezio e di Gauss.

Esercizio 4.10 Siano I_1 e I_2 due approssimazioni, ottenute utilizzando la formula composita del trapezio con due passi di quadratura diversi, H_1 e H_2 , di $I(f) = \int_a^b f(x)dx$. Se $f^{(2)}$ non varia molto in (a, b) , il seguente valore

$$I_R = I_1 + (I_1 - I_2)/(H_2^2/H_1^2 - 1) \quad (4.32)$$

costituisce una approssimazione di $I(f)$ migliore di quelle date da I_1 e I_2 . Questo metodo è noto come *metodo di estrapolazione di Richardson*. Usando la (4.18) si ricavi la (4.32).

Esercizio 4.11 Si verifichi che tra le formule del tipo $I_{approx}(f) = \alpha f(\bar{x}) + \beta f(\bar{z})$ dove $\bar{x}, \bar{z} \in [a, b]$ sono nodi incogniti e α e β coefficienti da determinare, la formula con $n = 1$ della Tabella 4.1 è quella con grado di esattezza massimo.

Esercizio 4.12 Per le prime due funzioni dell’Esercizio 4.5, si valuti il minimo numero di intervalli necessari per ottenere un integrale approssimato con la formula di Simpson composita a meno di un errore di 10^{-4} .

Esercizio 4.13 Si calcoli $\int_0^2 e^{-x^2/2} dx$ con la formula di Simpson (4.23) e con la formula di Gauss-Legendre di Tabella 4.1 per $n = 1$ e si confrontino i risultati ottenuti.

Esercizio 4.14 Per il calcolo degli integrali $I_k = \int_0^1 x^k e^{x-1} dx$ per $k = 1, 2, \dots$, si può utilizzare la seguente formula ricorsiva: $I_k = 1 - kI_{k-1}$ con $I_1 = 1/e$. Si calcoli I_{20} con la formula di Simpson composita in modo da garantire un errore inferiore a 10^{-3} . Si confronti il risultato ottenuto con quello fornito dall’uso della formula ricorsiva suddetta.

Esercizio 4.15 Si applichi la formula di estrapolazione di Richardson (4.32) per l’approssimazione dell’integrale $\int_0^2 e^{-x^2/2} dx$ con $H_1 = 1$ e $H_2 = 0.5$ usando prima la formula di Simpson (4.23) e quindi la formula di Gauss-Legendre per $n = 1$ di Tabella 4.1. Si verifichi che in entrambi i casi I_R è sempre più accurato di I_1 e I_2 .

Esercizio 4.16 (Elettromagnetismo) Si approssimi con la formula composita di Simpson la funzione $j(r, 0)$ definita nella (4.2) per $r = k/10$ m con $k = 1, \dots, 10$, $\rho(r) = \exp(r)$ e $\sigma = 0.36$ W/(mK). Si garantisca che l’errore commesso sia inferiore a 10^{-10} (ricordiamo che m=metri, W=watts, K=gradi Kelvin).

Esercizio 4.17 (Ottica) Si calcoli la funzione $E(T)$ definita nella (4.1) per T pari a 213 K (cioè -60 gradi Celsius) con almeno 10 cifre significative esatte utilizzando le formule composite di Simpson e di Gauss-Legendre con $n = 1$.

Esercizio 4.18 Si proponga una strategia per il calcolo di $I(f) = \int_0^1 |x^2 - 0.25| dx$ con la formula di Simpson composita tale da garantire che l'errore sia complessivamente inferiore a 10^{-2} .

5

Sistemi lineari

Nelle scienze applicate la risoluzione di problemi, anche complessi, viene spesso ricondotta alla risoluzione di uno o più sistemi lineari della forma

$$A\mathbf{x} = \mathbf{b}, \quad (5.1)$$

dove A è una matrice quadrata di dimensione $n \times n$ di elementi a_{ij} , reali o complessi, mentre \mathbf{x} e \mathbf{b} sono vettori colonna di dimensione n che rappresentano rispettivamente il vettore soluzione ed il vettore termine noto. Il sistema (5.1) può essere riscritto per componenti come segue

$$a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1,$$

$$a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2,$$

$$\vdots \qquad \vdots \qquad \vdots$$

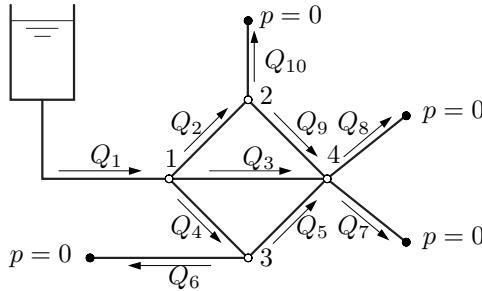
$$a_{n1}x_1 + a_{n2}x_2 + \dots + a_{nn}x_n = b_n.$$

Presentiamo tre problemi che danno luogo a sistemi lineari.

Problema 5.1 (Idraulica) Consideriamo un sistema idraulico formato da 10 condotte, disposte come in Figura 5.1, ed alimentato da un bacino d'acqua posto ad una pressione costante pari a $p_r = 10$ bar. In questo problema, i valori delle pressioni corrispondono alla differenza fra la pressione effettiva e quella atmosferica. Nella condotta j -esima vale la seguente relazione fra la portata Q_j (in m^3/s) e la differenza di pressione Δp_j all'estremità della condotta

$$Q_j = kL\Delta p_j, \quad (5.2)$$

dove k è la resistenza idraulica (misurata (in m^2 /(bar s)) e L la lunghezza (in m) della condotta. Supponiamo che nelle condotte terminali (quelle delimitate ad un estremo da un pallino nero) l'acqua esca alla pressione atmosferica, posta, per coerenza con la precedente convenzione, pari a 0 bar.

**Figura 5.1.** La rete di condotte del Problema 5.1

Un problema tipico consiste nel determinare i valori di pressione nei nodi interni 1, 2, 3 e 4 del sistema. A tal fine, per ogni $j = 1, 2, 3, 4$ possiamo integrare la relazione (5.2) con il fatto che la somma algebrica delle portate nel nodo j -esimo deve essere nulla (una portata negativa indicherà che l'acqua esce dal nodo).

Denotando con $\mathbf{p} = (p_1, p_2, p_3, p_4)^T$ il vettore delle pressioni nei nodi interni, otteniamo un sistema di 4 equazioni e 4 incognite della forma $A\mathbf{p} = \mathbf{b}$.

Nella seguente tabella riassumiamo le caratteristiche principali delle diverse condotte.

condotta	k	L	condotta	k	L	condotta	k	L
1	0.01	20	2	0.005	10	3	0.005	14
4	0.005	10	5	0.005	10	6	0.002	8
7	0.002	8	8	0.002	8	9	0.005	10
10	0.002	8						

Corrispondentemente, A e \mathbf{b} assumeranno i seguenti valori (abbiamo riportato le sole prime 4 cifre significative):

$$A = \begin{bmatrix} -0.370 & 0.050 & 0.050 & 0.070 \\ 0.050 & -0.116 & 0 & 0.050 \\ 0.050 & 0 & -0.116 & 0.050 \\ 0.070 & 0.050 & 0.050 & -0.202 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} -2 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

La soluzione di questo sistema verrà data nell'Esempio 5.5.

Problema 5.2 (Spettrometria) Esaminiamo una miscela di gas costituita da n componenti sconosciute che non si combinano chimicamente tra loro. Usando uno spettrometro di massa si bombarda il gas con elettroni a bassa energia: la corrispondente miscela di ioni viene analizzata da un galvanometro collegato all'apparecchio che mostra dei picchi in corrispondenza di specifici rapporti di massa su carica. Consideriamo soltanto gli n picchi più rilevanti. Si può ipotizzare che l'altezza h_i dell' i -esimo picco sia una combinazione lineare dei valori $\{p_j, j = 1, \dots, n\}$, dove p_j è la pressione parziale della componente j -esima (cioè della pressione esercitata da un singolo gas quando è parte di una miscela):

$$\sum_{j=1}^n s_{ij} p_j = h_i, \quad i = 1, \dots, n \quad (5.3)$$

e dove gli s_{ij} sono i cosiddetti coefficienti di sensitività. La determinazione delle pressioni parziali richiede quindi la risoluzione di un sistema lineare. Per la risoluzione di questo problema si veda l'Esempio 5.3.

Problema 5.3 (Economia: analisi di input-output) Si vuole trovare la condizione di equilibrio fra la domanda e l'offerta di determinati beni, assumendo valido un modello di produzione con n beni e $m \geq n$ imprese. Ogni impresa necessita nella sua attività produttiva di alcuni beni per produrne altri; chiamiamo *input* i beni consumati dal processo produttivo ed *output* quelli prodotti. Per semplicità supponiamo che il modello sia lineare, di conseguenza la quantità prodotta di un certo *output* è proporzionale alla quantità degli *input* utilizzati. L'attività delle imprese è quindi completamente descritta dalla matrice degli *input* $C \in \mathbb{R}^{n \times m}$ e dalla matrice degli *output* $P \in \mathbb{R}^{n \times m}$. Il valore c_{ij} (risp. p_{ij}) rappresenta la quantità del bene i -esimo consumato (risp. prodotto) dall'impresa j -sima, quando essa lavora al livello unitario di produzione per un fissato periodo di tempo. La matrice $A = P - C$, detta *matrice di input-output*, descrive dunque i consumi e le produzioni nette di beni da parte delle imprese: il coefficiente a_{ij} se positivo indica la quantità netta del bene i -esimo prodotto dall'impresa j -esima lavorando al livello unitario di produzione, se negativo rappresenta invece la quantità netta del bene i -esimo consumato nelle stesse condizioni. Il sistema dovrà infine avere un certo obiettivo produttivo costituito ad esempio dalla domanda di beni da parte del mercato che può essere rappresentato da un vettore \mathbf{b} di \mathbb{R}^n (detto della *domanda finale*). Il coefficiente b_i rappresenta dunque la quantità del bene i -esimo richiesta al sistema. Se indichiamo con x_i il livello di produzione raggiunto dall'impresa i -esima, il sistema economico sarà in equilibrio se

$$\mathbf{Ax} = \mathbf{b}, \text{ dove } A = P - C. \quad (5.4)$$

Nel modello di Leontief (1930) si assume che l'impresa i -esima produca il solo bene i -esimo (si veda la Figura 5.2). Di conseguenza, $n = m$ e $P = I$. Per la soluzione del sistema (5.4) si veda l'Esercizio 5.17.

La soluzione del sistema (5.1) esiste se e solo se la matrice A è non singolare. In linea di principio, la si potrebbe calcolare tramite la *regola di Cramer*

$$x_i = \frac{\det(A_i)}{\det(A)}, \quad i = 1, \dots, n,$$

dove A_i è la matrice ottenuta da A sostituendo la i -esima colonna con \mathbf{b} e $\det(A)$ è il determinante di A . Il calcolo degli $n + 1$ determinanti con lo sviluppo di Laplace (si veda l'Esercizio 5.1) richiede circa $2(n + 1)!$ operazioni intendendo, come al solito, per operazione la singola somma, sottrazione, moltiplicazione o divisione. Ad esempio, su un calcolatore in grado di eseguire 10^9 flops (i.e. 1 gigaflops) servirebbero circa 12 ore per

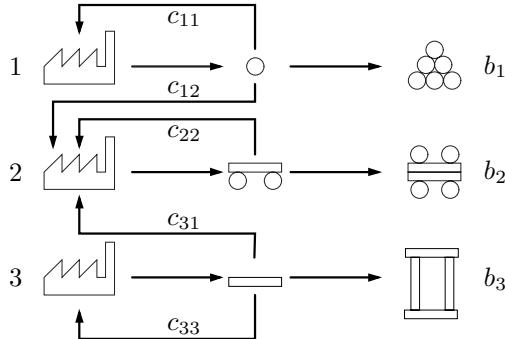


Figura 5.2. Lo schema di interazione fra 3 industrie ed il mercato descritto nel Problema 5.3

risolvere un sistema di dimensione $n = 15,3240$ anni se $n = 20$ e 10^{143} anni se $n = 100$. Il costo computazionale può essere drasticamente ridotto e portato all'ordine di circa $n^{3.8}$ operazioni se gli $n + 1$ determinanti vengono calcolati con l'algoritmo citato nell'Esempio 1.3. Tuttavia, tale costo risulterebbe ancora troppo elevato per le applicazioni pratiche e servono pertanto dei metodi alternativi.

Considereremo due approcci: quello dei metodi *diretti*, con cui la soluzione del sistema viene calcolata dopo un numero finito di passi, e quello dei metodi *iterativi*, che richiedono un numero (teoricamente) infinito di passi. Analizzeremo prima i metodi diretti e poi, a partire dal paragrafo 5.7, quelli iterativi. Mettiamo in guardia il lettore che nella scelta fra un metodo diretto ed uno iterativo intervengono molteplici fattori legati non solo all'efficienza teorica dello schema, ma anche al particolare tipo di matrice, alle richieste di occupazione di memoria ed infine al tipo di computer disponibile (si veda il paragrafo 5.11 per maggiori dettagli).

Facciamo notare che in generale un sistema lineare non potrà essere risolto con meno di n^2 operazioni. Infatti, se le equazioni del sistema sono tra loro veramente accoppiate, è lecito aspettarsi che ognuno degli n^2 elementi della matrice venga almeno una volta interessato da una operazione.

5.1 Il metodo di fattorizzazione LU

Sia A una matrice quadrata di ordine n . Supponiamo che esistano due opportune matrici L ed U , triangolare inferiore e superiore, rispettivamente, tali che

$$A = LU. \quad (5.5)$$

La (5.5) è detta *fattorizzazione* (o decomposizione) LU di A . Osserviamo che se A è non singolare tali matrici devono essere anch'esse non singolari;

in particolare ciò assicura che i loro elementi diagonali siano non nulli (come osservato nel paragrafo 1.3).

In tal caso, risolvere $\mathbf{Ax} = \mathbf{b}$ conduce alla risoluzione dei due seguenti sistemi triangolari

$$\boxed{\mathbf{Ly} = \mathbf{b}, \mathbf{Ux} = \mathbf{y}} \quad (5.6)$$

Entrambi i sistemi sono semplici da risolvere. Infatti, essendo L triangolare inferiore, la prima riga del sistema $\mathbf{Ly} = \mathbf{b}$ avrà la forma

$$l_{11}y_1 = b_1,$$

da cui si ricava il valore di y_1 essendo $l_{11} \neq 0$. Sostituendo il valore trovato per y_1 nelle successive $n - 1$ equazioni troviamo un sistema le cui incognite sono y_2, \dots, y_n , per le quali possiamo procedere allo stesso modo. Procedendo in avanti, equazione per equazione, calcoliamo tutte le incognite con il seguente algoritmo, detto delle *sostituzioni in avanti*

$$\boxed{\begin{aligned} y_1 &= \frac{1}{l_{11}}b_1, \\ y_i &= \frac{1}{l_{ii}} \left(b_i - \sum_{j=1}^{i-1} l_{ij}y_j \right), \quad i = 2, \dots, n \end{aligned}} \quad (5.7)$$

Quantifichiamo il numero di operazioni richiesto da (5.7) osservando che, per calcolare l'incognita y_i , si devono effettuare $i - 1$ somme, $i - 1$ prodotti ed una divisione. Si ottiene un numero totale di operazioni pari a

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

In maniera del tutto analoga potrà essere risolto il sistema $\mathbf{Ux} = \mathbf{y}$: in tal caso, la prima incognita ad essere calcolata sarà x_n e poi, a ritroso, verranno calcolate tutte le restanti incognite x_i per i che varia da $n - 1$ fino a 1

$$\boxed{\begin{aligned} x_n &= \frac{1}{u_{nn}}y_n, \\ x_i &= \frac{1}{u_{ii}} \left(y_i - \sum_{j=i+1}^n u_{ij}x_j \right), \quad i = n - 1, \dots, 1 \end{aligned}} \quad (5.8)$$

Questo algoritmo viene chiamato delle *sostituzioni all'indietro* e richiede ancora n^2 operazioni. Si tratta a questo punto di trovare un algoritmo che consenta di calcolare effettivamente L ed U a partire da A . Illustriamo una procedura generale a partire da una coppia di esempi.

Esempio 5.1 Scriviamo la relazione (5.5) per una generica matrice $A \in \mathbb{R}^{2 \times 2}$

$$\begin{bmatrix} l_{11} & 0 \\ l_{21} & l_{22} \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} \\ 0 & u_{22} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}.$$

I 6 elementi incogniti di L e di U dovranno allora soddisfare le seguenti equazioni (non lineari)

$$\begin{aligned} (e_1) \quad l_{11}u_{11} &= a_{11}, & (e_2) \quad l_{11}u_{12} &= a_{12}, \\ (e_3) \quad l_{21}u_{11} &= a_{21}, & (e_4) \quad l_{21}u_{12} + l_{22}u_{22} &= a_{22}. \end{aligned} \quad (5.9)$$

Questo sistema è *sottodeterminato*, presentando più incognite che equazioni. Per eliminare l'indeterminazione fissiamo *arbitrariamente* pari a 1 gli elementi diagonali di L, aggiungendo perciò le equazioni $l_{11} = 1$ e $l_{22} = 1$. A questo punto, il sistema (5.9) può essere risolto procedendo nel modo seguente: dalle (e₁) ed (e₂) ricaviamo gli elementi della prima riga di U, u_{11} ed u_{12} . Se u_{11} è non nullo, da (e₃) si trova allora l_{21} (cioè la prima colonna di L, essendo l_{11} già fissato pari a 1) e quindi, da (e₄), u_{22} (ossia l'unico elemento non nullo della seconda riga di U).

Esempio 5.2 Ripetiamo gli stessi calcoli per una matrice 3×3 . Per i 12 coefficienti incogniti di L e U abbiamo le seguenti 9 equazioni

$$\begin{aligned} (e_1) \quad l_{11}u_{11} &= a_{11}, & (e_2) \quad l_{11}u_{12} &= a_{12}, & (e_3) \quad l_{11}u_{13} &= a_{13}, \\ (e_4) \quad l_{21}u_{11} &= a_{21}, & (e_5) \quad l_{21}u_{12} + l_{22}u_{22} &= a_{22}, & (e_6) \quad l_{21}u_{13} + l_{22}u_{23} &= a_{23}, \\ (e_7) \quad l_{31}u_{11} &= a_{31}, & (e_8) \quad l_{31}u_{12} + l_{32}u_{22} &= a_{32}, & (e_9) \quad l_{31}u_{13} + l_{32}u_{23} &+ l_{33}u_{33} = a_{33}. \end{aligned}$$

Completiamo tale sistema con le equazioni $l_{ii} = 1$ per $i = 1, 2, 3$. Nuovamente, il sistema ottenuto può essere facilmente risolto calcolando tramite le (e₁), (e₂) e (e₃) i coefficienti della prima riga di U; utilizzando quindi (e₄) e (e₇), possiamo determinare i coefficienti l_{21} e l_{31} della prima colonna di L. Noti questi ultimi, da (e₅) ed (e₆) si ricavano i coefficienti u_{22} ed u_{23} della seconda riga di U e poi, tramite (e₈), il coefficiente l_{32} della seconda colonna di L. Infine, l'ultima riga di U (ridotta al solo elemento u_{33}) viene determinata risolvendo (e₉).

Per una matrice di dimensione arbitraria n possiamo procedere nel modo seguente:

- gli elementi di L e di U soddisfano il seguente sistema non lineare di equazioni

$$\sum_{r=1}^{\min(i,j)} l_{ir}u_{rj} = a_{ij}, \quad i, j = 1, \dots, n; \quad (5.10)$$

- il sistema (5.10) è sottodeterminante, essendovi n^2 equazioni e n^2+n incognite; di conseguenza, la fattorizzazione LU in generale non sarà unica (ovvero possono esistere diverse coppie di matrici L e U che soddisfano (5.10));
- imponendo che gli n elementi diagonali di L siano pari a 1, (5.10) diviene un sistema quadrato determinato che può essere risolto con

il seguente *algoritmo di Gauss*: posto $A^{(1)} = A$ ovvero $a_{ij}^{(1)} = a_{ij}$ per $i, j = 1, \dots, n$, si calcoli

$$\begin{aligned}
 & \text{per } k = 1, \dots, n-1 \\
 & \quad \text{per } i = k+1, \dots, n \\
 & \quad l_{ik} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}, \\
 & \quad \text{per } j = k+1, \dots, n \\
 & \quad a_{ij}^{(k+1)} = a_{ij}^{(k)} - l_{ik} a_{kj}^{(k)}
 \end{aligned} \tag{5.11}$$

Gli elementi $a_{kk}^{(k)}$ devono essere tutti diversi da zero e sono detti *elementi pivot*. Per ogni $k = 1, \dots, n-1$ la matrice $A^{(k+1)} = (a_{ij}^{(k+1)})$ ha $n-k$ righe e colonne.

Al termine di questo processo gli elementi della matrice triangolare superiore di U sono dati da $u_{ij} = a_{ij}^{(i)}$ per $i = 1, \dots, n$ e $j = i, \dots, n$, mentre quelli di L sono dati dai coefficienti l_{ij} generati dall'algoritmo. In (5.11) non vengono calcolati esplicitamente gli elementi diagonali di L in quanto già sappiamo che sono pari a 1.

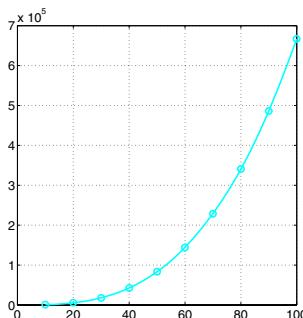
Questa fattorizzazione è detta di Gauss; il calcolo dei coefficienti dei fattori L ed U richiede circa $2n^3/3$ operazioni (si veda l'Esercizio 5.4).

Esempio 5.3 (Spettrometria) Riprendiamo il Problema 5.2 e consideriamo una miscela di gas che, ad un esame spettroscopico, presenta i seguenti 7 picchi più rilevanti: $h_1 = 17.1$, $h_2 = 65.1$, $h_3 = 186.0$, $h_4 = 82.7$, $h_5 = 84.2$, $h_6 = 63.7$ e $h_7 = 119.7$. Vogliamo confrontare la pressione totale misurata, pari a $38.78 \mu\text{m}$ di Hg (che tiene conto anche di componenti che abbiamo eventualmente trascurato nella nostra semplificazione), con quella ottenuta usando le relazioni (5.3) con $n = 7$, dove i coefficienti di sensitività sono riportati in Tabella 5.1 (tratti da [CLW69, pag.331]). Le pressioni parziali, che si trovano risolvendo il sistema (5.3) per $n = 7$ con la fattorizzazione LU, valgono

```
parzpress=
0.6525
2.2038
0.3348
6.4344
2.9975
0.5505
25.6317
```

e conducono ad una stima della pressione totale (`sum(parzpress)`) che presenta una differenza pari a 0.0252 rispetto al valore misurato di $-38.78 \mu\text{m}$ di Hg.

Indice del picco	Componente e indice						
	Idrogeno 1	Metano 2	Etilene 3	Etano 4	Propilene 5	Propano 6	<i>n</i> -Pentano 7
1	16.87	0.1650	0.2019	0.3170	0.2340	0.1820	0.1100
2	0.0	27.70	0.8620	0.0620	0.0730	0.1310	0.1200
3	0.0	0.0	22.35	13.05	4.420	6.001	3.043
4	0.0	0.0	0.0	11.28	0.0	1.110	0.3710
5	0.0	0.0	0.0	0.0	9.850	1.1684	2.108
6	0.0	0.0	0.0	0.0	0.2990	15.98	2.107
7	0.0	0.0	0.0	0.0	0.0	0.0	4.670

Tabella 5.1. I coefficienti di sensitività per una particolare miscela gassosa**Figura 5.3.** Il numero di operazioni, in funzione di n , necessario per generare la fattorizzazione LU di Gauss della matrice di Vandermonde. Questa funzione è una cubica ottenuta approssimando nel senso dei minimi quadrati i valori (rappresentati da pallini) corrispondenti a $n = 10, 20, \dots, 100$ **Esempio 5.4** Consideriamo la seguente matrice di Vandermonde

$$A = (a_{ij}) \text{ con } a_{ij} = x_i^{n-j}, \quad i, j = 1, \dots, n,$$

dove gli x_i sono n valori distinti. Essa può essere costruita usando il comando MATLAB `vander`. In Figura 5.3 riportiamo il numero di operazioni *floating-point* necessarie per calcolare la fattorizzazione di Gauss di A in corrispondenza di diversi valori della dimensione n della matrice, precisamente $n = 10, 20, \dots, 100$. La curva riportata in Figura 5.3 è un polinomio in n di terzo grado che rappresenta l'approssimazione ai minimi quadrati di tali valori. Il calcolo delle operazioni è stato effettuato ricorrendo ad un comando (`flops`) che era presente in MATLAB sino alla versione 5.3. Il coefficiente del monomio n^3 è proprio $2/3$.

Naturalmente non è necessario memorizzare tutte le matrici $A^{(k)}$. In effetti conviene sovrapporre gli $(n - k) \times (n - k)$ elementi di $A^{(k+1)}$ ai corrispondenti $(n - k) \times (n - k)$ ultimi elementi della matrice originale A . Inoltre, poiché al k -esimo passo gli elementi sottodiagonali della k -esima colonna non influenzano la matrice finale U , essi possono essere rimpiazzati dagli elementi della k -esima colonna di L , così come fatto

nel Programma 11. Di conseguenza, al k -esimo passo del processo gli elementi memorizzati al posto dei coefficienti originali della matrice A sono

$$\left[\begin{array}{cccccc} a_{11}^{(1)} & a_{12}^{(1)} & \dots & \dots \dots & a_{1n}^{(1)} \\ l_{21} & a_{22}^{(2)} & & & a_{2n}^{(2)} \\ \vdots & \ddots & \ddots & & \vdots \\ l_{k1} & \dots & l_{k,k-1} & a_{kk}^{(k)} & \dots & a_{kn}^{(k)} \\ \vdots & & \vdots & \vdots & & \vdots \\ l_{n1} & \dots & l_{n,k-1} & a_{nk}^{(k)} & \dots & a_{nn}^{(k)} \end{array} \right],$$

dove la matrice nel riquadro è $A^{(k)}$.

La fattorizzazione di Gauss è alla base dei seguenti comandi MATLAB:

- **[L,U]=lu(A)** le cui modalità di impiego verranno discusse nel paragrafo 5.2;
- **inv** che consente il calcolo dell'inversa di una matrice;
- **** tramite il quale si risolve un sistema lineare di matrice A e termine noto b scrivendo semplicemente $A \backslash b$.

Osservazione 5.1 (Calcolo del determinante) Tramite la fattorizzazione LU si può calcolare il determinante di una matrice A con un costo computazione di $\mathcal{O}(n^3)$ operazioni, osservando che (si veda il paragrafo 1.3)

$$\det(A) = \det(L) \det(U) = \prod_{k=1}^n u_{kk}.$$

Questa procedura è alla base del comando MATLAB **det**.

det

Nel Programma 11 abbiamo implementato l'algoritmo (5.11). Per evitare sprechi di memoria la matrice L (privata della diagonale che sappiamo essere costituita da elementi tutti uguali a 1) viene memorizzata nella parte triangolare inferiore di A, mentre la matrice U (inclusa la diagonale) in quella superiore. Dopo l'esecuzione del programma, i due fattori possono essere ricostruiti semplicemente scrivendo: $L = \text{eye}(n) + \text{tril}(A, -1)$ and $U = \text{triu}(A)$, dove n è la dimensione di A.

Programma 11 - lu_gauss : la fattorizzazione di Gauss



```
function A=lu_gauss(A)
%LU_GAUSS    Fattorizzazione LU senza pivoting
% A = LU_GAUSS(A) calcola la fattorizzazione LU di Gauss della matrice
% A memorizzando nella parte triangolare inferiore stretta di A la matrice L (gli
% elementi diagonali di L sono tutti uguali a 1) ed in quella superiore il fattore U
[n,m]=size(A);
if n~=m; error('A non e'' una matrice quadrata'); else
```

```

for k = 1:n-1
    for i = k+1:n
        A(i,k) = A(i,k)/A(k,k);
        if A(k,k) == 0, error('Un elemento pivot si e'' annullato'); end
        j = [k+1:n]; A(i,j) = A(i,j) - A(i,k)*A(k,j);
    end
end
end

```

Esempio 5.5 Per risolvere il sistema ottenuto nel problema 5.1 utilizziamo la fattorizzazione LU ed i metodi delle sostituzioni in avanti ed all'indietro

```

>> A=lu_gauss(A);
>> y(1)=b(1); for i = 2:4; y = [y; b(i)-A(i,1:i-1)*y(1:i-1)]; end
>> x(4)=y(4)/A(4,4);
>> for i = 3:-1:1; x(i)=(y(i)-A(i,i+1:4)*x(i+1:4))/A(i,i); end

```

Il risultato è $\mathbf{p} = (8.1172, 5.9893, 5.9893, 5.7779)^T$.

Esempio 5.6 La soluzione del sistema $\mathbf{Ax} = \mathbf{b}$, con

$$A = \begin{bmatrix} 1 & 1 - \varepsilon & 3 \\ 2 & 2 & 2 \\ 3 & 6 & 4 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 5 - \varepsilon \\ 6 \\ 13 \end{bmatrix}, \quad \varepsilon \in \mathbb{R}, \quad (5.12)$$

è $\mathbf{x} = (1, 1, 1)^T$ (indipendente da ε).

Per $\varepsilon = 1$ la fattorizzazione di Gauss di A , ottenuta con il Programma 11, è

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 3 & 1 \end{bmatrix}, \quad U = \begin{bmatrix} 1 & 0 & 3 \\ 0 & 2 & -4 \\ 0 & 0 & 7 \end{bmatrix}.$$

Se poniamo $\varepsilon = 0$, anche se A è non singolare, la fattorizzazione di Gauss non può essere calcolata in quanto l'algoritmo (5.11) comporta una divisione per 0.

L'esempio precedente mostra che, sfortunatamente, la fattorizzazione di Gauss potrebbe non esistere anche se la matrice A è non singolare. In tal senso si può dimostrare il seguente risultato:

Proposizione 5.1 *Data una matrice $A \in \mathbb{R}^{n \times n}$, la sua fattorizzazione di Gauss esiste ed è unica se e solo se le sottomatrici principali A_i di A di ordine $i = 1, \dots, n-1$ (cioè quelle ottenute limitando A alle sole prime i righe e colonne) sono non singolari.*

Tornando all'Esempio 5.6, possiamo in effetti notare che quando $\varepsilon = 0$ la seconda sottomatrice A_2 di A è singolare.

Possiamo identificare alcune classi di matrici per le quali le ipotesi della Proposizione 5.1 sono soddisfatte. In particolare, ricordiamo:

1. le matrici simmetriche e definite positive. Ricordiamo che una matrice $A \in \mathbb{R}^{n \times n}$ è *definita positiva* se

$$\forall \mathbf{x} \in \mathbb{R}^n \text{ con } \mathbf{x} \neq \mathbf{0}, \quad \mathbf{x}^T \mathbf{A} \mathbf{x} > 0;$$

2. le matrici a dominanza diagonale. Una matrice è *a dominanza diagonale per righe* se

$$|a_{ii}| \geq \sum_{\substack{j=1 \\ j \neq i}}^n |a_{ij}|, \quad i = 1, \dots, n,$$

per colonne se

$$|a_{ii}| \geq \sum_{\substack{j=1 \\ j \neq i}}^n |a_{ji}|, \quad i = 1, \dots, n.$$

Quando nelle precedenti diseguaglianze possiamo sostituire il segno \geq con quello $>$ diremo che A è a dominanza diagonale *stretta* (per righe o per colonne, rispettivamente).

Se A è una matrice simmetrica e definita positiva, è inoltre possibile trovarne una fattorizzazione speciale

$$A = H H^T, \tag{5.13}$$

essendo H una matrice triangolare inferiore con elementi positivi sulla diagonale.

La (5.13) è nota come *fattorizzazione di Cholesky*. Il calcolo di H richiede circa $n^3/3$ operazioni (cioè la metà di quelle richieste per calcolare le due matrici della fattorizzazione LU di Gauss). Si noti inoltre che per la simmetria di A , ne verrà memorizzata la sola parte triangolare inferiore ed H potrà essere memorizzata nella stessa area di memoria.

Gli elementi di H possono essere calcolati tramite il seguente algoritmo: poniamo $h_{11} = \sqrt{a_{11}}$ e, per $i = 2, \dots, n$,

$$\begin{aligned} h_{ij} &= \frac{1}{h_{jj}} \left(a_{ij} - \sum_{k=1}^{j-1} h_{ik} h_{jk} \right), \quad j = 1, \dots, i-1 \\ h_{ii} &= \sqrt{a_{ii} - \sum_{k=1}^{i-1} h_{ik}^2} \end{aligned} \tag{5.14}$$

La fattorizzazione di Cholesky è richiamabile in MATLAB con la sintassi `R=chol(A)`, essendo R il fattore triangolare superiore H^T .

`chol`

Si vedano gli Esercizi 5.1-5.5.



5.2 La tecnica del pivoting

Vogliamo introdurre un metodo che consenta di portare a compimento il processo di fattorizzazione LU per una qualunque matrice A non singolare, anche nel caso in cui le ipotesi della Proposizione 5.1 non siano soddisfatte.

Riprendiamo la matrice dell'Esempio 5.6 nel caso in cui $\varepsilon = 0$. Poniamo al solito $A^{(1)} = A$ ed eseguiamo solo il primo passo ($k = 1$) di tale metodo; i nuovi coefficienti di A sono

$$\left[\begin{array}{ccc|c} 1 & 1 & 3 \\ 2 & \mathbf{0} & -4 \\ 3 & 3 & -5 \end{array} \right]. \quad (5.15)$$

Essendo nullo il *pivot* a_{22} , la procedura non può proseguire oltre. D'altra parte, se prima di procedere con la fattorizzazione avessimo scambiato la seconda riga di A con la terza, avremmo ottenuto la matrice

$$\left[\begin{array}{ccc|c} 1 & 1 & 3 \\ 3 & \mathbf{3} & -5 \\ 2 & 0 & -4 \end{array} \right]$$

e la fattorizzazione avrebbe potuto terminare senza incontrare divisioni per 0.

Quindi, *permutando* (cioè scambiando) opportunamente le righe della matrice A di partenza, è possibile portare a termine il processo di fattorizzazione anche quando le ipotesi della Proposizione 5.1 non sono soddisfatte, ma nella sola ipotesi che $\det(A) \neq 0$. Sfortunatamente non è possibile stabilire a priori quali siano le righe che dovranno essere tra loro scambiate; tuttavia questa decisione può essere presa ad ogni passo k durante il quale si generino elementi $a_{kk}^{(k)}$ nulli.

Riprendiamo la matrice (5.15) che presenta l'elemento pivot nullo (in posizione $(2, 2)$). Osservato che il corrispondente elemento della terza riga non è nullo, scambiamo quest'ultima riga con la seconda e procediamo con il processo di fattorizzazione. Così facendo si trova proprio la matrice che si sarebbe ottenuta permutando a priori le medesime due righe nella matrice A, ovvero possiamo effettuare gli scambi tra righe quando si rendono necessari, senza preoccuparci di doverli individuare a priori.

Siccome lo scambio tra le righe comporta un cambiamento dei *pivot*, questa tecnica viene chiamata *pivoting per righe*. La fattorizzazione che si trova restituiscce la matrice A di partenza a meno di una permutazione fra le righe. Precisamente,

$$PA = LU,$$

essendo P una opportuna *matrice di permutazione*. Essa è posta uguale all'identità all'inizio del processo di fattorizzazione: se nel corso della fattorizzazione le righe r e s di A vengono scambiate, uno scambio analogo deve essere fatto sulle righe di P . Corrispondentemente, dovremo ora risolvere i seguenti sistemi triangolari

$$Ly = Pb, \quad Ux = y. \quad (5.16)$$

Dalla seconda equazione nella (5.11) si comprende anche che elementi $a_{kk}^{(k)}$ piccoli, pur non impedendo la corretta conclusione del calcolo della fattorizzazione, possono comportare gravi perdite di accuratezza nel risultato finale, in quanto gli eventuali errori di arrotondamento presenti nei coefficienti $a_{kj}^{(k)}$ potrebbero risultare fortemente amplificati.

Esempio 5.7 Consideriamo la matrice non singolare seguente

$$A = \begin{bmatrix} 1 & 1 + 0.5 \cdot 10^{-15} & 3 \\ 2 & 2 & 20 \\ 3 & 6 & 4 \end{bmatrix}.$$

Durante il calcolo della fattorizzazione con il Programma 11 non si generano elementi *pivot* nulli. Nonostante ciò, i fattori L ed U calcolati sono assai inaccordati, come si verifica calcolando $A - LU$ (che in aritmetica esatta sarebbe uguale alla matrice nulla)

$$A - LU = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 4 \end{bmatrix}.$$

È pertanto consigliabile eseguire il *pivoting* ad ogni passo della procedura di fattorizzazione, cercando fra tutti i *pivot* disponibili $a_{ik}^{(k)}$ con $i = k, \dots, n$, quello di modulo massimo. L'algoritmo (5.11) con il *pivoting* per righe, eseguito ad ogni passo, diventa allora:

per $k = 1, \dots, n-1$,

per $i = k+1, \dots, n$

trovare \bar{r} tale che $|a_{\bar{r}k}^{(k)}| = \max_{r=k, \dots, n} |a_{rk}^{(k)}|$,

scambiare la riga k con la riga \bar{r} ,

(5.17)

$l_{ik} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}$,

per $j = k+1, \dots, n$

$a_{ij}^{(k+1)} = a_{ij}^{(k)} - l_{ik} a_{kj}^{(k)}$

La funzione MATLAB `lu`, cui abbiamo accennato in precedenza, calcola la fattorizzazione di Gauss con *pivoting* per righe. La sua sintassi completa è infatti `[L, U, P] = lu(A)`, dove P è una matrice di permutazione.

Quando richiamata con la sintassi abbreviata $[L, U] = \text{lu}(A)$, essa produce una matrice L che è uguale a $P \cdot M$, dove M è triangolare inferiore e P è una matrice di permutazione generata dal *pivoting* per righe. Il comando `lu` attiva automaticamente il *pivoting* per righe quando viene generato un elemento *pivot* nullo (o molto piccolo).



Si vedano gli Esercizi 5.6-5.8.



5.3 Quanto è accurata la fattorizzazione LU?

Come abbiamo già avuto modo di notare nell'Esempio 5.7, a causa degli errori di arrotondamento il prodotto LU non riproduce esattamente la matrice A . Tuttavia, l'uso del *pivoting* consente di contenere questo genere di errori e sembrerebbe quindi ragionevole attendersi con tale tecnica una soluzione accurata del sistema da risolvere. Purtroppo ciò non sempre è vero, come mostra l'esempio seguente.

Esempio 5.8 Consideriamo il sistema lineare $A_n \mathbf{x}_n = \mathbf{b}_n$ dove $A_n \in \mathbb{R}^{n \times n}$ è la cosiddetta *matrice di Hilbert* di elementi

$$a_{ij} = 1/(i+j-1), \quad i, j = 1, \dots, n,$$

mentre \mathbf{b}_n è scelto in modo tale che la soluzione esatta del sistema sia $\mathbf{x}_n = (1, 1, \dots, 1)^T$. La matrice A_n è chiaramente simmetrica e si può dimostrare che essa è anche definita positiva. Per diversi valori di n usiamo in MATLAB la funzione `lu` per calcolare la fattorizzazione di Gauss di A_n con *pivoting* per righe. Risolviamo quindi i sistemi lineari associati (5.16), indicando con $\hat{\mathbf{x}}_n$ la soluzione calcolata. Nella Figura 5.4 riportiamo l'andamento dell'errore relativo al variare di n ,

$$E_n = \|\mathbf{x}_n - \hat{\mathbf{x}}_n\| / \|\mathbf{x}_n\|, \quad (5.18)$$

avendo indicato con $\|\cdot\|$ la norma Euclidea introdotta nel paragrafo 1.3.1. Abbiamo $E_n \geq 10$ se $n \geq 13$ (ovvero un errore relativo sulla soluzione superiore al 1000%!), mentre $R_n = L_n U_n - P_n A_n$ è una matrice nulla (rispetto alla precisione di macchina), qualunque sia il valore di n considerato.

Sulla base della precedente osservazione, si può allora ipotizzare che quando si risolve numericamente il sistema lineare $A \mathbf{x} = \mathbf{b}$ in pratica si stia trovando la soluzione *esatta* $\hat{\mathbf{x}}$ di un sistema *perturbato* della forma

$$(A + \delta A) \hat{\mathbf{x}} = \mathbf{b} + \delta \mathbf{b}, \quad (5.19)$$

dove δA e $\delta \mathbf{b}$ sono rispettivamente una matrice ed un vettore che dipendono dallo specifico metodo numerico impiegato nella risoluzione del sistema. Incominciamo ad analizzare il caso in cui $\delta A = 0$ e $\delta \mathbf{b} \neq \mathbf{0}$, in quanto più semplice del caso generale. Supponiamo inoltre per semplicità che A sia simmetrica e definita positiva.

Confrontando (5.1) e (5.19) troviamo $\mathbf{x} - \hat{\mathbf{x}} = -A^{-1} \delta \mathbf{b}$, e dunque

$$\|\mathbf{x} - \hat{\mathbf{x}}\| = \|A^{-1} \delta \mathbf{b}\|. \quad (5.20)$$

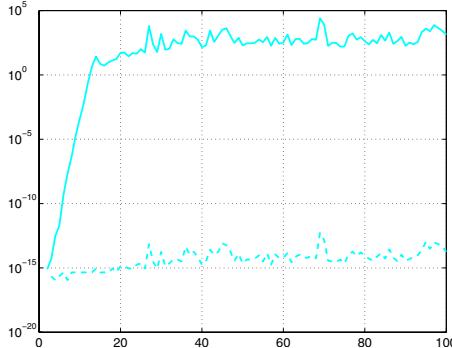


Figura 5.4. Andamento di E_n (linea continua) e di $\max_{i,j=1,\dots,n} |r_{ij}|$ (linea tratteggiata) rispetto a n per il sistema di Hilbert dell’Esempio 5.8. Gli (r_{ij}) sono i coefficienti della matrice R

Per trovare un limite superiore del termine di destra della (5.20) procediamo nel modo seguente. Essendo A simmetrica e definita positiva, esiste una base ortonormale $\{\mathbf{v}_i\}_{i=1}^n$ di \mathbb{R}^n formata dagli autovettori \mathbf{v}_i di A (si veda ad esempio [QSS04, Capitolo 5]). Questo comporta che

$$\begin{aligned} A\mathbf{v}_i &= \lambda_i \mathbf{v}_i, \quad i = 1, \dots, n, \\ \mathbf{v}_i^T \mathbf{v}_j &= \delta_{ij}, \quad i, j = 1, \dots, n, \end{aligned}$$

dove λ_i è l’autovalore di A associato a \mathbf{v}_i e δ_{ij} è il simbolo di Kronecker. Di conseguenza, un generico vettore $\mathbf{w} \in \mathbb{R}^n$ può essere scritto come

$$\mathbf{w} = \sum_{i=1}^n w_i \mathbf{v}_i,$$

per un opportuno (ed unico) insieme di coefficienti $w_i \in \mathbb{R}$. Abbiamo

$$\begin{aligned} \|A\mathbf{w}\|^2 &= (A\mathbf{w})^T (A\mathbf{w}) \\ &= [w_1(A\mathbf{v}_1)^T + \dots + w_n(A\mathbf{v}_n)^T][w_1 A \mathbf{v}_1 + \dots + w_n A \mathbf{v}_n] \\ &= (\lambda_1 w_1 \mathbf{v}_1^T + \dots + \lambda_n w_n \mathbf{v}_n^T)(\lambda_1 w_1 \mathbf{v}_1 + \dots + \lambda_n w_n \mathbf{v}_n) \\ &= \sum_{i=1}^n \lambda_i^2 w_i^2. \end{aligned}$$

Denotiamo con λ_{\max} il più grande autovalore di A . Poiché $\|\mathbf{w}\|^2 = \sum_{i=1}^n w_i^2$, concludiamo che

$$\|A\mathbf{w}\| \leq \lambda_{\max} \|\mathbf{w}\|, \quad \forall \mathbf{w} \in \mathbb{R}^n. \quad (5.21)$$

In modo analogo, otteniamo

$$\|A^{-1}\mathbf{w}\| \leq \frac{1}{\lambda_{\min}} \|\mathbf{w}\|,$$

in quanto gli autovalori di A^{-1} sono i reciproci di quelli di A . Questa diseguaglianza consente di dedurre dalla (5.20) che

$$\frac{\|\mathbf{x} - \hat{\mathbf{x}}\|}{\|\mathbf{x}\|} \leq \frac{1}{\lambda_{min}} \frac{\|\delta \mathbf{b}\|}{\|\mathbf{x}\|}. \quad (5.22)$$

Usando nuovamente la (5.21) e ricordando che $A\mathbf{x} = \mathbf{b}$, otteniamo infine

$$\frac{\|\mathbf{x} - \hat{\mathbf{x}}\|}{\|\mathbf{x}\|} \leq \frac{\lambda_{max}}{\lambda_{min}} \frac{\|\delta \mathbf{b}\|}{\|\mathbf{b}\|} \quad (5.23)$$

Quindi l'errore relativo sulla soluzione dipende dall'errore relativo sui dati attraverso la seguente costante (≥ 1)

$$K(A) = \frac{\lambda_{max}}{\lambda_{min}} \quad (5.24)$$

che è detta *numero di condizionamento (spettrale) della matrice A*. $K(A)$ può essere calcolato in MATLAB con il comando `cond`. Altre definizioni per il numero di condizionamento sono disponibili per matrici non simmetriche, si veda ad esempio [QSS04, Capitolo 3].

Osservazione 5.2 Il comando `cond(A)` consente di calcolare (in modo approssimato) il numero di condizionamento di una matrice A generica, anche quando non è simmetrica o definita positiva. Un altro comando disponibile è `rcond(A)` che fornisce un'approssimazione dell'inverso del numero di condizionamento. Il comando `condeest(A)` consente invece di calcolarne un'approssimazione di basso costo computazionale quando A è una matrice sparsa. Se la matrice A è mal condizionata (cioè $K(A) \gg 1$) anche il calcolo del suo numero di condizionamento potrà essere inaccurato. Ad esempio, consideriamo le matrici tridiagonali $A_n = \text{tridiag}(-1, 2, -1)$ di n righe e colonne al variare di n . A_n è simmetrica e definita positiva ed ha autovalori $\lambda_j = 2 - 2 \cos(j\theta)$, per $j = 1, \dots, n$, con $\theta = \pi/(n+1)$: di conseguenza, $K(A_n)$ può essere calcolato esattamente. In Figura 5.5 riportiamo il valore dell'errore $E_K(n) = |K(A_n) - \text{cond}(A_n)|/K(A_n)$.

Una dimostrazione più elaborata avrebbe condotto ad un risultato più generale nel caso in cui anche δA fosse stata diversa da 0. Precisamente, supponendo δA simmetrica e definita positiva e tale che $\lambda_{max}(\delta A) < \lambda_{min}(A)$, si può dimostrare che

$$\frac{\|\mathbf{x} - \hat{\mathbf{x}}\|}{\|\mathbf{x}\|} \leq \frac{K(A)}{1 - \lambda_{max}(\delta A)/\lambda_{min}(A)} \left(\frac{\lambda_{max}(\delta A)}{\lambda_{max}(A)} + \frac{\|\delta \mathbf{b}\|}{\|\mathbf{b}\|} \right).$$

Se $K(A)$ è “piccolo”, cioè dell'ordine dell'unità, la matrice A viene detta *ben condizionata* ed a piccoli errori sui dati corrisponderanno errori dello stesso ordine di grandezza sulla soluzione. Se invece è grande, la matrice si dirà *mal condizionata* e potrebbe accadere che a piccole perturbazioni sui dati corrispondano grandi errori.

cond

rcond
condeest

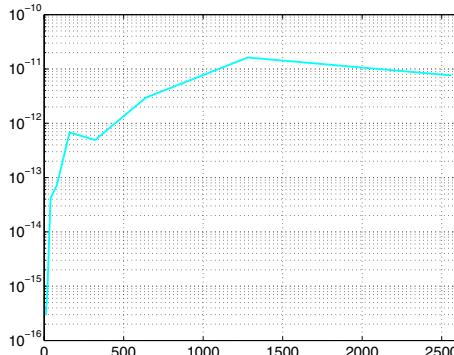


Figura 5.5. Riportiamo il valore $E_K(n)$ al variare di n

Esempio 5.9 Per la matrice di Hilbert introdotta nell’Esempio 5.8, $K(A_n)$ è una funzione rapidamente crescente di n . Abbiamo $K(A_4) > 15000$, mentre se $n > 13$, MATLAB segnala che la matrice ha un numero di condizionamento così elevato da essere ritenuta singolare. Non dobbiamo perciò sorprenderci dei cattivi risultati ottenuti nell’Esempio 5.8.

La diseguaglianza (5.23) può essere riformulata introducendo il *residuo*

$$\mathbf{r} = \mathbf{b} - \mathbf{A}\hat{\mathbf{x}}. \quad (5.25)$$

Evidentemente se $\hat{\mathbf{x}}$ fosse la soluzione esatta, il residuo sarebbe nullo. Di conseguenza, \mathbf{r} può essere ritenuto uno *stimatore* dell’errore commesso $\mathbf{x} - \hat{\mathbf{x}}$. La sua efficacia come stimatore dipende dalla grandezza del numero di condizionamento di A . Infatti, applicando la (5.23) ed osservando che $\delta\mathbf{b} = \mathbf{A}(\hat{\mathbf{x}} - \mathbf{x}) = \mathbf{A}\hat{\mathbf{x}} - \mathbf{b} = -\mathbf{r}$, si ricava

$$\frac{\|\mathbf{x} - \hat{\mathbf{x}}\|}{\|\mathbf{x}\|} \leq K(A) \frac{\|\mathbf{r}\|}{\|\mathbf{b}\|} \quad (5.26)$$

Quindi se $K(A)$ è “piccolo”, avremo la certezza che l’errore sarà piccolo quando lo è il residuo, mentre ciò non è necessariamente vero quando $K(A)$ è “grande”.

Esempio 5.10 Se calcoliamo la norma del residuo per i sistemi dell’Esempio 5.8, troviamo quantità che variano tra 10^{-16} e 10^{-11} in corrispondenza di soluzioni che nulla hanno a che fare con la soluzione esatta del sistema.

Si vedano gli Esercizi 5.9-5.10.



5.4 Come risolvere un sistema tridiagonale

In molte applicazioni (si veda ad esempio il Capitolo 8) è necessario risolvere un sistema con una matrice della forma

$$A = \begin{bmatrix} a_1 & c_1 & & 0 \\ e_2 & a_2 & \ddots & \\ \ddots & & c_{n-1} & \\ 0 & e_n & a_n & \end{bmatrix}.$$

La matrice A viene detta *tridiagonale* in quanto gli unici elementi che possono essere non nulli appartengono alla diagonale principale ed alla prima sopra e sotto-diagonale.

Supponiamo che i coefficienti della matrice siano numeri reali. Allora se la fattorizzazione di Gauss di A esiste, i fattori L e U sono due matrici *bidiagonali* (inferiore e superiore, rispettivamente) della forma

$$L = \begin{bmatrix} 1 & & & 0 \\ \beta_2 & 1 & & \\ \ddots & \ddots & \ddots & \\ 0 & & \beta_n & 1 \end{bmatrix}, \quad U = \begin{bmatrix} \alpha_1 & c_1 & & 0 \\ & \alpha_2 & \ddots & \\ & & \ddots & c_{n-1} \\ 0 & & & \alpha_n \end{bmatrix}.$$

I coefficienti incogniti α_i e β_i possono essere determinati imponendo l'uguaglianza $LU = A$. In tal modo si trovano le seguenti relazioni ricorsive

$$\alpha_1 = a_1, \quad \beta_i = \frac{e_i}{\alpha_{i-1}}, \quad \alpha_i = a_i - \beta_i c_{i-1}, \quad i = 2, \dots, n.$$

Grazie ad esse, possiamo risolvere i due sistemi bidiagonali $Ly = b$ e $Ux = y$, ottenendo le seguenti formule

$$(Ly = b) \quad y_1 = b_1, \quad y_i = b_i - \beta_i y_{i-1}, \quad i = 2, \dots, n, \quad (5.27)$$

$$(Ux = y) \quad x_n = \frac{y_n}{\alpha_n}, \quad x_i = \frac{y_i - c_i x_{i+1}}{\alpha_i}, \quad i = n-1, \dots, 1. \quad (5.28)$$

Questa procedura è nota come *algoritmo di Thomas* e consente di risolvere un sistema tridiagonale con un costo dell'ordine di n operazioni.

spdiags

Il comando MATLAB **spdiags** permette di costruire facilmente una matrice tridiagonale, memorizzando le sole tre diagonali non nulle. Ad esempio, attraverso i comandi

```
>> b=ones(10,1); a=2*b; c=3*b;
>> T=spdiags([b a c],-1:1,10,10);
```

si ottiene la matrice tridiagonale $T \in \mathbb{R}^{10 \times 10}$ con coefficienti pari a 2 sulla diagonale principale, 1 sulla prima sotto-diagonale e 3 sulla prima sopra-diagonale.

Si noti che T viene memorizzata in un *formato sparso*: quest'ultimo prende in considerazione solo gli elementi diversi da 0 ed è precisabile in MATLAB con il comando `sparse`. In effetti, una matrice è detta *sparsa* se ha un numero di coefficienti non nulli dell'ordine della dimensione della matrice stessa.

`sparse`

Quando si incontra un sistema la cui matrice è memorizzata in tale formato MATLAB richiama automaticamente tecniche di risoluzione opportune che consentono di ottimizzare i tempi di calcolo e l'occupazione di memoria (in particolare, il metodo di Thomas se la matrice è tridiagonale).

5.5 Sistemi sovradeterminati

Un sistema lineare $\mathbf{Ax} = \mathbf{b}$ con $A \in \mathbb{R}^{m \times n}$ viene detto *sovradeterminato* se $m > n$, *sottodeterminato* se $m < n$.

In generale, un sistema sovradeterminato non ha soluzione a meno che il termine noto \mathbf{b} non sia un elemento del range(A), definito come

$$\text{range}(A) = \{\mathbf{y} \in \mathbb{R}^m : \mathbf{y} = A\mathbf{x} \text{ per } \mathbf{x} \in \mathbb{R}^n\}. \quad (5.29)$$

Per un termine noto \mathbf{b} arbitrario possiamo cercare un vettore $\mathbf{x}^* \in \mathbb{R}^n$ che minimizzi la norma euclidea del residuo, cioè tale che

$$\Phi(\mathbf{x}^*) = \|A\mathbf{x}^* - \mathbf{b}\|_2^2 \leq \min_{\mathbf{x} \in \mathbb{R}^n} \|A\mathbf{x} - \mathbf{b}\|_2^2 = \min_{\mathbf{x} \in \mathbb{R}^n} \Phi(\mathbf{x}). \quad (5.30)$$

Il vettore \mathbf{x}^* quando esiste è detto *soluzione nel senso dei minimi quadrati* del sistema sovradeterminato $\mathbf{Ax} = \mathbf{b}$.

In modo del tutto analogo a quanto fatto nel paragrafo 3.4, si può trovare la soluzione di (5.30) imponendo che il gradiente della funzione Φ sia nullo in \mathbf{x}^* . Con calcoli del tutto simili si trova che \mathbf{x}^* è di fatto la soluzione del sistema lineare

$$A^T A \mathbf{x}^* = A^T \mathbf{b}, \quad (5.31)$$

detto sistema delle *equazioni normali*. Il sistema (5.31) è non singolare se A è a *rango pieno*, cioè se $\text{rank}(A) = \min(m,n)$, dove $\text{rank}(A)$ è il massimo ordine dei determinanti non nulli estratti da A . In tal caso $B = A^T A$ è una matrice simmetrica e definita positiva, e di conseguenza la soluzione nel senso dei minimi quadrati esiste unica. Per calcolarla si può usare la fattorizzazione di Cholesky (5.13) applicata alla matrice B .

Si tenga comunque conto che a causa degli errori di arrotondamento il calcolo di $A^T A$ può introdurre una perdita di cifre significative con

la conseguente perdita di definita positività della matrice. Anche per questa ragione è più conveniente usare, al posto della fattorizzazione di Cholesky, la cosiddetta fattorizzazione QR. Ogni matrice $A \in \mathbb{R}^{m \times n}$, con $m \geq n$, ammette un'unica *fattorizzazione QR*, esiste cioè una matrice $Q \in \mathbb{R}^{m \times m}$ ortogonale ($Q^T Q = I$) ed una matrice $R \in \mathbb{R}^{m \times n}$ triangolare superiore con tutte le righe nulle, a partire dalla $n+1$ -esima, tali che

$$A = QR. \quad (5.32)$$

L'unica soluzione di (5.30) è allora data da

$$\mathbf{x}^* = \tilde{\mathbf{R}}^{-1} \tilde{\mathbf{Q}}^T \mathbf{b} \quad (5.33)$$

dove $\tilde{\mathbf{R}} \in \mathbb{R}^{n \times n}$ e $\tilde{\mathbf{Q}} \in \mathbb{R}^{m \times n}$ sono le seguenti matrici

$$\tilde{\mathbf{Q}} = Q(1:m, 1:n), \quad \tilde{\mathbf{R}} = R(1:n, 1:n).$$

Facciamo notare che $\tilde{\mathbf{R}}$ è non singolare.

Esempio 5.11 Consideriamo un approccio alternativo al problema di trovare la retta di regressione $\epsilon(\sigma) = a_1\sigma + a_0$ (si veda il paragrafo 3.4) per i dati del Problema 3.3. Usando i dati della Tabella 3.2 e imponendo le condizioni di interpolazione otteniamo il sistema sovradeterminato $\mathbf{A}\mathbf{a} = \mathbf{b}$, dove $\mathbf{a} = (a_1, a_0)^T$ e

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ 0.06 & 1 \\ 0.14 & 1 \\ 0.25 & 1 \\ 0.31 & 1 \\ 0.47 & 1 \\ 0.60 & 1 \\ 0.70 & 1 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 0 \\ 0.08 \\ 0.14 \\ 0.20 \\ 0.23 \\ 0.25 \\ 0.28 \\ 0.29 \end{bmatrix}.$$

Per calcolarne la soluzione nel senso dei minimi quadrati usiamo le seguenti istruzioni MATLAB

```
>> [Q,R]=qr(A);
>> Qt=Q(:,1:5); Rt=R(1:5,:);
>> xstar = Rt \ (Qt'*b)
xstar =
    0.3741
    0.0654
```

che sono precisamente i coefficienti della retta di regressione calcolata nell'Esempio 3.10. Si noti che questa procedura è automaticamente implementata nel comando \: l'istruzione `xstar = A\b` produce infatti il medesimo vettore `xstar` calcolato in precedenza.

5.6 Cosa si nasconde nel comando \ di MATLAB

È importante sapere che nel comando \ di MATLAB viene richiamato uno specifico algoritmo a seconda delle caratteristiche della matrice A del sistema lineare che si intende risolvere. A grandi linee, MATLAB segue la seguente procedura:

1. se A è una matrice sparsa e a banda, vengono usati dei risolutori per matrici a banda (come l'algoritmo di Thomas del paragrafo 5.4). Diciamo che una matrice $A \in \mathbb{R}^{m \times n}$ (o in $\mathbb{C}^{m \times n}$) ha *banda inferiore* p se $a_{ij} = 0$ per $i > j + p$ e *banda superiore* q se $a_{ij} = 0$ per $j > i + q$. Il massimo fra p e q viene detto larghezza di banda della matrice.
2. Se A è una matrice triangolare superiore o inferiore (o una permutazione di una matrice triangolare), il sistema viene risolto con il metodo delle sostituzioni all'indietro nel caso superiore o con quello delle sostituzioni in avanti nel caso inferiore. Il controllo sulla triangularità della matrice viene fatto controllando la disposizione degli elementi nulli se la matrice è piena, accedendo direttamente ai dati memorizzati nella struttura di sparsità della matrice se la matrice è sparsa.
3. Se A è simmetrica ed ha coefficienti diagonali reali e positivi (il che non implica che A sia simmetrica e definita positiva) in prima battuta MATLAB impiega la fattorizzazione di Cholesky (`chol`). Se inoltre A è sparsa, essa viene preventivamente riordinata.
4. Se nessuno dei precedenti criteri è soddisfatto ed A è una matrice quadrata non memorizzata in formato sparso, viene calcolata una generica fattorizzazione di Gauss con pivoting parziale (`lu`).
5. Se A è quadrata, non a banda e sparsa, viene richiamata la libreria UMFPACK per risolvere il sistema.
6. Infine se A è una matrice rettangolare, vengono richiamati metodi opportuni per sistemi sovradeterminati basati sulla fattorizzazione QR (si veda il paragrafo 5.5).

Riassumendo



1. La fattorizzazione LU di A consiste nel calcolare due matrici, L, triangolare inferiore, U, triangolare superiore, tali che $A = LU$;
2. se esiste, la fattorizzazione LU non è unica e viene determinata fissando n condizioni addizionali. Ad esempio, ponendo i coefficienti diagonali di L pari a 1, si trova la cosiddetta fattorizzazione di Gauss;
3. la fattorizzazione di Gauss può essere calcolata solo se le sottomatrici principali di A di ordine da 1 fino a $n - 1$ sono tutte non singolari; in caso contrario, almeno un *pivot* risulterà nullo; è inoltre non singolare se anche $\det(A) \neq 0$;

4. in presenza di un *pivot* nullo, si deve individuare un nuovo *pivot* scambiando opportunamente fra loro le righe o le colonne del sistema (questa strategia è detta *pivoting*);
5. per calcolare i fattori L e U sono richieste circa $2n^3/3$ operazioni. Nel caso di sistemi tridiagonali tale costo scende ad un ordine di n operazioni;
6. per le matrici simmetriche e definite positive esiste la fattorizzazione di Cholesky, $A = HH^T$, con H triangolare inferiore. Il relativo costo computazionale è dell'ordine di $n^3/3$ operazioni;
7. la sensibilità della soluzione di un sistema lineare alle perturbazioni sui dati dipende dal numero di condizionamento della matrice. Precisamente, quando quest'ultimo è grande, piccole perturbazioni sui coefficienti della matrice e del termine noto possono dar luogo a soluzioni molto inaccurate;
8. la soluzione di un sistema lineare sovradeterminato deve essere intesa nel senso dei minimi quadrati e può essere calcolata attraverso la fattorizzazione QR.

5.7 Metodi iterativi

Un metodo iterativo per la risoluzione del sistema lineare (5.1) consiste nel costruire una successione di vettori $\{\mathbf{x}^{(k)}, k \geq 0\}$ di \mathbb{R}^n che si spera convergente alla soluzione esatta \mathbf{x} , ossia tale che

$$\lim_{k \rightarrow \infty} \mathbf{x}^{(k)} = \mathbf{x} \quad (5.34)$$

per un qualunque vettore iniziale $\mathbf{x}^{(0)} \in \mathbb{R}^n$. Per realizzare questo processo una possibile strategia è quella di definire ricorsivamente

$$\mathbf{x}^{(k+1)} = \mathbf{B}\mathbf{x}^{(k)} + \mathbf{g}, \quad k \geq 0, \quad (5.35)$$

essendo \mathbf{B} una matrice opportuna (dipendente da \mathbf{A}) e \mathbf{g} un vettore opportuno (dipendente da \mathbf{A} e da \mathbf{b}), scelti in modo tale da garantire la *condizione di consistenza*

$$\mathbf{x} = \mathbf{B}\mathbf{x} + \mathbf{g}. \quad (5.36)$$

Essendo $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$, necessariamente dovrà aversi $\mathbf{g} = (\mathbf{I} - \mathbf{B})\mathbf{A}^{-1}\mathbf{b}$.

Detto $\mathbf{e}^{(k)} = \mathbf{x} - \mathbf{x}^{(k)}$ l'errore al passo k , sottraendo la (5.35) dalla (5.36), si ottiene

$$\mathbf{e}^{(k+1)} = \mathbf{B}\mathbf{e}^{(k)}.$$

Per tale ragione \mathbf{B} è detta *matrice di iterazione* del metodo (5.35). Se \mathbf{B} è simmetrica e definita positiva, grazie alla (5.21) otteniamo

$$\|\mathbf{e}^{(k+1)}\| = \|\mathbf{B}\mathbf{e}^{(k)}\| \leq \rho(\mathbf{B})\|\mathbf{e}^{(k)}\|, \quad \forall k \geq 0,$$

dove $\rho(\mathbf{B}) = \max |\lambda(\mathbf{B})|$ è il *raggio spettrale* di \mathbf{B} . Per matrici simmetriche e definite positive esso coincide con il massimo autovalore. Iterando a ritroso la stessa disuguaglianza, si trova

$$\|\mathbf{e}^{(k)}\| \leq [\rho(\mathbf{B})]^k \|\mathbf{e}^{(0)}\|, \quad k \geq 0. \quad (5.37)$$

Se $\rho(\mathbf{B}) < 1$, allora $\mathbf{e}^{(k)} \rightarrow \mathbf{0}$ per $k \rightarrow \infty$ per ogni $\mathbf{e}^{(0)}$ (e, conseguentemente, per ogni $\mathbf{x}^{(0)}$). Pertanto il metodo iterativo converge. Si può inoltre dimostrare che questa ipotesi è anche necessaria per la convergenza.

Facciamo notare che se si conoscesse $\rho(\mathbf{B})$, dalla (5.37) sarebbe possibile ricavare il minimo numero di iterazioni k_{min} necessario per abbattere l'errore iniziale di un dato fattore ε . Infatti, k_{min} sarebbe il più piccolo intero positivo per cui $[\rho(\mathbf{B})]^{k_{min}} \leq \varepsilon$.

In generale, per una matrice \mathbf{B} qualunque vale la seguente proprietà:

Proposizione 5.2 *Un metodo iterativo della forma (5.35) la cui matrice di iterazione \mathbf{B} soddisfi la (5.36), è convergente per ogni $\mathbf{x}^{(0)}$ se e soltanto se $\rho(\mathbf{B}) < 1$. Inoltre, minore è $\rho(\mathbf{B})$, minore è il numero di iterazioni necessario per ridurre l'errore iniziale di un dato fattore.*

5.7.1 Come costruire un metodo iterativo

Una tecnica generale per costruire un metodo iterativo è basata sulla seguente *decomposizione additiva* (o *splitting*) della matrice \mathbf{A} , $\mathbf{A} = \mathbf{P} - (\mathbf{P} - \mathbf{A})$, dove \mathbf{P} è una opportuna matrice non singolare che chiameremo *precondizionatore* di \mathbf{A} . Di conseguenza,

$$\mathbf{P}\mathbf{x} = (\mathbf{P} - \mathbf{A})\mathbf{x} + \mathbf{b}$$

è un sistema della forma (5.36) purché si ponga $\mathbf{B} = \mathbf{P}^{-1}(\mathbf{P} - \mathbf{A}) = \mathbf{I} - \mathbf{P}^{-1}\mathbf{A}$ e $\mathbf{g} = \mathbf{P}^{-1}\mathbf{b}$. Questa identità suggerisce la definizione del seguente metodo iterativo

$$\mathbf{P}(\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}) = \mathbf{r}^{(k)}, \quad k \geq 0,$$

dove $\mathbf{r}^{(k)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(k)}$ denota il residuo alla k -esima iterazione. Una generalizzazione di questo metodo iterativo è

$$\mathbf{P}(\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}) = \alpha_k \mathbf{r}^{(k)}, \quad k \geq 0 \quad (5.38)$$

dove $\alpha_k \neq 0$ è un parametro che può cambiare ad ogni iterazione k e che, a priori, servirà a migliorare le proprietà di convergenza della successione $\{\mathbf{x}^{(k)}\}$.

Il metodo (5.38) richiede ad ogni passo di trovare il cosiddetto *residuo precondizionato* $\mathbf{z}^{(k)}$ dato dalla soluzione del sistema lineare

$$P\mathbf{z}^{(k)} = \mathbf{r}^{(k)}. \quad (5.39)$$

Di conseguenza, la nuova iterata è definita da $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k \mathbf{z}^{(k)}$. Per questa ragione la matrice P deve essere scelta in modo tale che il costo computazionale richiesto dalla risoluzione di (5.39) sia modesto (ogni matrice P diagonale, tridiagonale o triangolare andrebbe bene a questo scopo). Introduciamo ora alcuni esempi particolari di metodi iterativi della forma (5.38).

Il metodo di Jacobi

Se i coefficienti diagonali di A sono non nulli, possiamo scegliere $P = D = \text{diag}\{a_{11}, a_{22}, \dots, a_{nn}\}$. Il metodo di Jacobi corrisponde a questa scelta supponendo $\alpha_k = 1$ per ogni k . Di conseguenza, dalla (5.38), otteniamo

$$D\mathbf{x}^{(k+1)} = \mathbf{b} - (A - D)\mathbf{x}^{(k)}, \quad k \geq 0,$$

che, per componenti, assume la forma

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1, j \neq i}^n a_{ij} x_j^{(k)} \right), \quad i = 1, \dots, n \quad (5.40)$$

per ogni $k \geq 0$, essendo $\mathbf{x}^{(0)} = (x_1^{(0)}, x_2^{(0)}, \dots, x_n^{(0)})^T$ il vettore iniziale.

La matrice di iterazione è allora

$$B = D^{-1}(D - A) = \begin{bmatrix} 0 & -a_{12}/a_{11} & \dots & -a_{1n}/a_{11} \\ -a_{21}/a_{22} & 0 & & -a_{2n}/a_{22} \\ \vdots & & \ddots & \vdots \\ -a_{n1}/a_{nn} & -a_{n2}/a_{nn} & \dots & 0 \end{bmatrix}. \quad (5.41)$$

Per il metodo di Jacobi vale il seguente risultato che consente di verificare la Proposizione 5.2 senza calcolare esplicitamente $\rho(B)$:

Proposizione 5.3 *Se la matrice A del sistema (5.1) è a dominanza diagonale stretta per righe, allora il metodo di Jacobi converge.*

Verifichiamo infatti che in tal caso $\rho(B) < 1$, con B data nella (5.41), cioè che tutti gli autovalori di B hanno modulo minore di 1. Iniziamo con l'osservare che la dominanza diagonale stretta garantisce che la diagonale di A non può presentare elementi nulli. Siano λ e \mathbf{x} una coppia autovalore-autovettore di B . Allora

$$\sum_{j=1}^n b_{ij}x_j = \lambda x_i, \quad i = 1, \dots, n.$$

Supponiamo per semplicità che $\max_{k=1, \dots, n} |x_k| = 1$ (questa ipotesi non è restrittiva in quanto ogni autovettore è definito a meno di una costante moltiplicativa) e sia x_i la componente che ha modulo pari a 1. Allora

$$|\lambda| = \left| \sum_{j=1}^n b_{ij}x_j \right| = \left| \sum_{j=1, j \neq i}^n b_{ij}x_j \right| \leq \sum_{j=1, j \neq i}^n \left| \frac{a_{ij}}{a_{ii}} \right|,$$

avendo osservato che B ha elementi diagonali tutti nulli. Possiamo quindi concludere che $|\lambda| < 1$ grazie alle ipotesi fatte su A .

Il metodo di Jacobi è richiamabile nel Programma 12 ponendo come parametro d'ingresso $P='J'$. I restanti parametri di ingresso sono: la matrice del sistema A , il termine noto \mathbf{b} , il vettore iniziale \mathbf{x}_0 ed il massimo numero $nmax$ di iterazioni consentite. La procedura iterativa si arresta non appena il rapporto fra la norma del residuo corrente ed il residuo iniziale sia inferiore ad una tolleranza fissata tol (si veda a questo riguardo il paragrafo 5.10).

Programma 12 - itermeth : un metodo iterativo generale



```
function [x, iter] = itermeth(A, b, x0, nmax, tol, P)
%ITERMETH Un metodo iterativo generale
% X = ITERMETH(A,B,X0,NMAX,TOL,P) cerca di risolvere iterativamente
% il sistema di equazioni lineari A*X=B su X. La matrice A di N-per-N
% coefficienti deve essere non singolare ed il termine noto B deve
% avere lunghezza N. Se P='J' viene usato il metodo di Jacobi, se
% P='G' viene invece selezionato il metodo di Gauss-Seidel. Altrimenti,
% P e' una matrice N-per-N non singolare che gioca il ruolo di
% precondizionatore in un metodo di Richardson a parametro dinamico.
% Il metodo si arresta quando il rapporto fra la
% norma del residuo corrente ed quella del residuo iniziale e' minore
% di TOL. NMAX prescrive il numero massimo di iterazioni consentite.
% Se P non viene precisata, viene usato il metodo di Richardson non
% precondizionato con parametro di rilassamento uguale a 1.
[n,n]=size(A);
if nargin == 6
    if ischar(P)==1
        if P=='J'
            L = diag(diag(A)); U = eye(n); beta = 1; alpha = 1;
        else
            L = diag(diag(A));
            U = eye(n) - L;
            beta = 1/(1+sqrt(1+4*beta));
            alpha = 2*beta;
        end
    else
        L = diag(diag(A));
        U = eye(n) - L;
        beta = 1/(1+sqrt(1+4*beta));
        alpha = 2*beta;
    end
    if tol < 0
        tol = 1e-6;
    end
    if nmax < 0
        nmax = 100;
    end
    if nargin < 6
        tol = 1e-6;
        nmax = 100;
    end
    if nargin < 5
        tol = 1e-6;
    end
    if nargin < 4
        tol = 1e-6;
    end
    if nargin < 3
        tol = 1e-6;
    end
    if nargin < 2
        tol = 1e-6;
    end
    if nargin < 1
        tol = 1e-6;
    end
end
if ~isequal(size(U), [n, n])
    error('U must be a square matrix');
end
if ~isequal(size(L), [n, n])
    error('L must be a square matrix');
end
if ~isequal(size(b), [n, 1])
    error('b must be a column vector');
end
if ~isequal(size(x0), [n, 1])
    error('x0 must be a column vector');
end
if ~isequal(size(A), [n, n])
    error('A must be a square matrix');
end
if ~isequal(size(beta), [1, 1])
    error('beta must be a scalar');
end
if ~isequal(size(alpha), [1, 1])
    error('alpha must be a scalar');
end
if ~isequal(size(tol), [1, 1])
    error('tol must be a scalar');
end
if ~isequal(size(nmax), [1, 1])
    error('nmax must be a scalar');
end
if ~isequal(size(P), [1, 1])
    error('P must be a scalar');
end
if P == 'J'
    method = 'Jacobi';
else
    method = 'Gauss-Seidel';
end
if method == 'Jacobi'
    for iter = 1:nmax
        x = U\beta;
        residual = norm(x - x0);
        if residual < tol
            break;
        end
        x0 = x;
        beta = U*x + b;
    end
else
    for iter = 1:nmax
        x = alpha*(U\b) + beta*L\beta;
        residual = norm(x - x0);
        if residual < tol
            break;
        end
        x0 = x;
        beta = alpha*(U\b) + beta*L\beta;
    end
end
if iter > nmax
    warning('The iteration has not converged');
end
x = x0;
```

```

elseif P == 'G'
    L = tril(A); U = eye(n); beta = 1; alpha = 1;
    end
else
    [L,U]=lu(P); beta = 0;
    end
else
    L = eye(n); U = L; beta = 0;
end
iter = 0; r = b - A * x0; r0 = norm(r); err = norm (r); x = x0;
while err > tol & iter < nmax
    iter = iter + 1;
    z = L\r; z = U\z;
    if beta == 0
        alpha = z'*r/(z'*A*z);
    end
    x = x + alpha*z; r = b - A * x;
    err = norm (r) / r0;
end
return

```

Il metodo di Gauss-Seidel

Nel metodo di Jacobi le componenti del vettore $\mathbf{x}^{(k+1)}$ vengono calcolate indipendentemente le une dalle altre. Questo fatto può suggerire che si potrebbe avere una convergenza più rapida se per il calcolo di $x_i^{(k+1)}$ venissero usate le nuove componenti già disponibili $x_j^{(k+1)}$, $j = 1, \dots, i-1$, assieme con le vecchie $x_j^{(k)}$, $j \geq i$. Si modifica allora il metodo (5.40) come segue: per $k \geq 0$ (supponendo ancora $a_{ii} \neq 0$ per $i = 1, \dots, n$)

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^n a_{ij} x_j^{(k)} \right), \quad i = 1, \dots, n \quad (5.42)$$

L'aggiornamento delle componenti deve essere pertanto effettuato in modo *sequenziale*, mentre nell'originale metodo di Jacobi può essere fatto *simultaneamente* (o in parallelo). Questo nuovo metodo, noto come *metodo di Gauss-Seidel*, corrisponde ad aver scelto $P = D - E$ e $\alpha_k = 1$, $k \geq 0$, in (5.38), dove E è una matrice triangolare inferiore i cui soli elementi non nulli sono $e_{ij} = -a_{ij}$, $i = 2, \dots, n$, $j = 1, \dots, i-1$. La corrispondente matrice di iterazione ha la forma

$$\mathbf{B} = (\mathbf{D} - \mathbf{E})^{-1}(\mathbf{D} - \mathbf{E} - \mathbf{A}).$$

Una generalizzazione di questo metodo è il cosiddetto *metodo di rilassamento* nel quale $P = \frac{1}{\omega}D - E$, dove $\omega \neq 0$ è un parametro di rilassamento, e $\alpha_k = 1$, $k \geq 0$ (si veda l'Esercizio 5.13).

Anche per il metodo di Gauss-Seidel esistono delle classi di matrici per le quali la Proposizione 5.2 è certamente verificata. Tra di esse menzioniamo:

1. le matrici a dominanza diagonale stretta per righe;
2. le matrici simmetriche e definite positive.

Il metodo di Gauss-Seidel è richiamabile nel Programma 12 ponendo il parametro di ingresso P uguale a ' G' '.

Non ci sono risultati generali che consentono di affermare che il metodo di Gauss-Seidel converga sempre più rapidamente di quello di Jacobi, a parte casi particolari come quello facente oggetto della seguente proposizione:

Proposizione 5.4 *Se A è una matrice tridiagonale di dimensione n non singolare con $a_{ii} \neq 0$, $i = 1, \dots, n$, allora i metodi di Jacobi e di Gauss-Seidel sono entrambi convergenti o entrambi divergenti. Nel caso di convergenza, il metodo di Gauss-Seidel converge più velocemente di quello di Jacobi: precisamente, il raggio spettrale della matrice di iterazione del metodo di Gauss-Seidel è il quadrato del raggio spettrale di quello del metodo di Jacobi.*

Esempio 5.12 Consideriamo un sistema lineare $Ax = b$ dove A è la matrice tridiagonale di dimensione $n = 10$ con elementi pari a 3 sulla diagonale principale, -2 sulla sopradiagonale e -1 sulla sottodiagonale, mentre b è scelto in modo tale che la soluzione sia il vettore unitario $(1, 1, \dots, 1)^T$. Entrambi i metodi di Jacobi e di Gauss-Seidel convergono in quanto i raggi spettrali delle matrici di iterazione sono minori di 1. In particolare, il metodo di Jacobi converge in 277 iterazioni contro le 143 richieste dal metodo di Gauss-Seidel (si è posto $tol = 10^{-12}$, $nmax=400$ e si è partiti da un dato iniziale nullo). Le istruzioni necessarie per ottenere questo risultato sono

```
>> n=10; A = 3*eye(n) - 2*diag(ones(n-1,1),1) - diag(ones(n-1,1),-1);
>> b=A*ones(n,1);
>> [x,iter]=itermeth(A,b,zeros(n,1),400,1.e-12,'J'); iter
iter =
    277
>> [x,iter]=itermeth(A,b,zeros(n,1),400,1.e-12,'G'); iter
iter =
    143
```

Si vedano gli Esercizi 5.11-5.14.



5.8 Il metodo di Richardson e del gradiente

Consideriamo in questo paragrafo metodi della forma (5.38) con parametri di accelerazione α_k non nulli. Diciamo *stazionario* il caso in cui $\alpha_k = \alpha$ (una costante assegnata) per ogni $k \geq 0$, *dinamico* il caso in cui α_k può cambiare ad ogni iterazione. In questo ambito la matrice non singolare P è detta ancora *precondizionatore* di A .

Il problema cruciale sta naturalmente nella scelta dei parametri. A questo proposito, vale il seguente risultato (si veda, ad esempio, [QV94, Capitolo 2], [Axe94]).

Proposizione 5.5 *Se P ed A sono entrambe simmetriche e definite positive il metodo stazionario di Richardson converge per ogni possibile scelta $\mathbf{x}^{(0)}$ se e solo se $0 < \alpha < 2/\lambda_{\max}(> 0)$, dove $\lambda_{\max}(> 0)$ è l'autovalore massimo di $P^{-1}A$. Inoltre, il raggio spettrale $\rho(B_\alpha)$ della matrice di iterazione $B_\alpha = I - \alpha P^{-1}A$ è minimo quando $\alpha = \alpha_{opt}$, dove*

$$\alpha_{opt} = \frac{2}{\lambda_{min} + \lambda_{max}} \quad (5.43)$$

essendo λ_{min} l'autovalore minimo di $P^{-1}A$.

Sotto le stesse ipotesi su P e A , il metodo dinamico di Richardson converge se, ad esempio, α_k è scelto nel modo seguente

$$\alpha_k = \frac{(\mathbf{z}^{(k)})^T \mathbf{r}^{(k)}}{(\mathbf{z}^{(k)})^T A \mathbf{z}^{(k)}} \quad \forall k \geq 0 \quad (5.44)$$

dove $\mathbf{z}^{(k)} = P^{-1}\mathbf{r}^{(k)}$ è il residuo precondizionato definito nella (5.39). Il metodo (5.38) con questa scelta di α_k è detto *metodo del gradiente precondizionato o, semplicemente, metodo del gradiente quando P è la matrice identità*.

In entrambi i casi, (5.43) e (5.44), vale la seguente stima di convergenza

$$\|\mathbf{e}^{(k)}\|_A \leq \left(\frac{K(P^{-1}A) - 1}{K(P^{-1}A) + 1} \right)^k \|\mathbf{e}^{(0)}\|_A, \quad k \geq 0 \quad (5.45)$$

dove $\|\mathbf{v}\|_A = \sqrt{\mathbf{v}^T A \mathbf{v}}$, $\forall \mathbf{v} \in \mathbb{R}^n$, è la cosiddetta norma dell'energia associata alla matrice A .

Il metodo non stazionario è da preferirsi al metodo stazionario in quanto non richiede il calcolo degli autovalori estremi di $P^{-1}A$ e determina il parametro α_k in funzione di quantità già calcolate al passo k .

Il metodo del gradiente precondizionato può essere scritto in modo efficiente attraverso il seguente algoritmo (la cui derivazione è lasciata per esercizio): dato $\mathbf{x}^{(0)}$, per ogni $k \geq 0$ si calcolino

$$\begin{aligned} P\mathbf{z}^{(k)} &= \mathbf{r}^{(k)} \\ \alpha_k &= \frac{(\mathbf{z}^{(k)})^T \mathbf{r}^{(k)}}{(\mathbf{z}^{(k)})^T A \mathbf{z}^{(k)}} \\ \mathbf{x}^{(k+1)} &= \mathbf{x}^{(k)} + \alpha_k \mathbf{z}^{(k)} \\ \mathbf{r}^{(k+1)} &= \mathbf{r}^{(k)} - \alpha_k A \mathbf{z}^{(k)} \end{aligned} \quad (5.46)$$

Lo stesso algoritmo può essere usato per implementare il metodo di Richardson stazionario semplicemente sostituendo α_k con il valore costante α .

Dalla (5.45), si deduce che se $P^{-1}A$ è mal condizionata la convergenza sarà molto lenta anche scegliendo $\alpha = \alpha_{opt}$ (in quanto $\rho(B_{\alpha_{opt}}) \simeq 1$). È dunque importante che P venga scelta opportunamente. È per questo motivo che P viene detta precondizionatore o matrice di precondizionamento.

Per una generica matrice A non è possibile stabilire una matrice di precondizionamento che garantisca un bilanciamento ottimale tra abbattimento del numero di condizionamento e sforzo computazionale richiesto per risolvere i sistemi (5.39). La scelta dovrà essere fatta caso per caso, tenuto conto del tipo di matrice A in esame.

Il metodo di Richardson dinamico è implementato nel Programma 12 nel quale il parametro d'ingresso P è la matrice di precondizionamento (se è assente il programma implementa il metodo non precondizionato ponendo $P=I$).

Esempio 5.13 Questo esempio, di puro interesse teorico, ha lo scopo di confrontare la convergenza dei metodi di Jacobi, Gauss-Seidel e del gradiente quando applicati alla soluzione del seguente (mini) sistema lineare

$$2x_1 + x_2 = 1, \quad x_1 + 3x_2 = 0 \quad (5.47)$$

con vettore iniziale $\mathbf{x}^{(0)} = (1, 1/2)^T$. Si noti che la matrice di questo sistema è simmetrica e definita positiva, e che la soluzione esatta è $\mathbf{x} = (3/5, -1/5)^T$. Riportiamo in Figura 5.6 l'andamento del residuo relativo $E^{(k)} = \|\mathbf{r}^{(k)}\|/\|\mathbf{r}^{(0)}\|$, al variare dell'indice di iterazione k , per i tre metodi citati. Le iterazioni vengono arrestate alla prima iterazione k_{min} per la quale $E^{(k_{min})} \leq 10^{-14}$. Il metodo del gradiente è quello che converge più rapidamente.

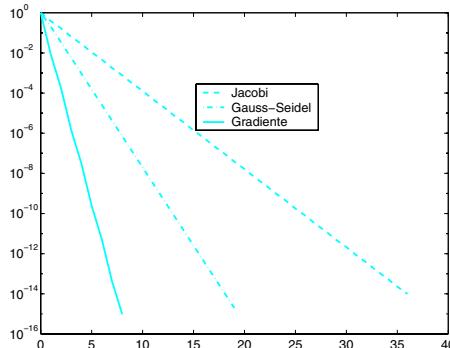


Figura 5.6. Storia di convergenza dei metodi di Jacobi, Gauss-Seidel e del gradiente applicati al sistema (5.47)

Esempio 5.14 Consideriamo il sistema $\mathbf{Ax} = \mathbf{b}$ con $\mathbf{A} \in \mathbb{R}^{100 \times 100}$ pentadiagonale. Le uniche diagonali non nulle di \mathbf{A} , oltre a quella principale che ha tutti elementi pari a 4, sono la prima e la terza sopra e sotto la diagonale principale che hanno elementi pari a -1 . Come sempre \mathbf{b} è scelto in modo tale che la soluzione esatta del sistema sia $\mathbf{x} = (1, \dots, 1)^T$. Poniamo \mathbf{P} uguale alla matrice tridiagonale di elementi diagonali pari a 2 ed elementi sopra e sotto diagonali pari a -1 . Sia \mathbf{A} che \mathbf{P} sono simmetriche e definite positive. Con tale precondizionatore il Programma 12 implementa il metodo di Richardson precondizionato. Fissiamo $\text{tol}=1.e-05$, $\text{nmax}=1000$, $\mathbf{x0}=\text{zeros}(100,1)$. Il metodo converge in 18 iterazioni; il Programma 12 con $\mathbf{P}='G'$ (che implementa il metodo di Gauss-Seidel) richiede invece ben 2421 iterazioni per soddisfare lo stesso criterio d'arresto.

5.9 Il metodo del gradiente coniugato

In un metodo iterativo come (5.46) la nuova iterata $\mathbf{x}^{(k+1)}$ viene ottenuta aggiungendo alla vecchia iterata $\mathbf{x}^{(k)}$ un vettore $\mathbf{z}^{(k)}$ che coincide con il residuo o con il residuo precondizionato. Una domanda naturale che ci si può porre è se sia possibile trovare invece di $\mathbf{z}^{(k)}$ una successione ottimale di vettori, diciamo $\mathbf{p}^{(k)}$, che assicuri la convergenza del metodo in un minimo numero di iterazioni.

Quando la matrice \mathbf{A} è simmetrica e definita positiva il metodo del gradiente coniugato (in breve, GC) utilizza una successione di vettori che sono *A-ortogonali* (o *A-coniugati*), cioè, tali che $\forall k \geq 1$,

$$(\mathbf{Ap}^{(j)})^T \mathbf{p}^{(k)} = 0, \quad j = 0, 1, \dots, k-1. \quad (5.48)$$

Ponendo allora $\mathbf{r}^{(0)} = \mathbf{b} - \mathbf{Ax}^{(0)}$ e $\mathbf{p}^{(0)} = \mathbf{r}^{(0)}$, la k -esima iterazione del metodo del gradiente coniugato assume la seguente forma

$$\begin{aligned}
\alpha_k &= \frac{\mathbf{p}^{(k)T} \mathbf{r}^{(k)}}{\mathbf{p}^{(k)T} \mathbf{A} \mathbf{p}^{(k)}} \\
\mathbf{x}^{(k+1)} &= \mathbf{x}^{(k)} + \alpha_k \mathbf{p}^{(k)} \\
\mathbf{r}^{(k+1)} &= \mathbf{r}^{(k)} - \alpha_k \mathbf{A} \mathbf{p}^{(k)} \\
\beta_k &= \frac{(\mathbf{A} \mathbf{p}^{(k)})^T \mathbf{r}^{(k+1)}}{(\mathbf{A} \mathbf{p}^{(k)})^T \mathbf{p}^{(k)}} \\
\mathbf{p}^{(k+1)} &= \mathbf{r}^{(k+1)} - \beta_k \mathbf{p}^{(k)}. \tag{5.49}
\end{aligned}$$

La costante α_k garantisce che l'errore sia minimizzato lungo la direzione di discesa $\mathbf{p}^{(k)}$, mentre β_k viene scelto in modo che la nuova direzione $\mathbf{p}^{(k+1)}$ sia A-coniugata con $\mathbf{p}^{(k)}$. Per una completa derivazione del metodo, si veda ad esempio [QSS04, Chapter 4] o [Saa96]. Si può dimostrare il seguente importante risultato di convergenza:

Proposizione 5.6 *Sia A una matrice simmetrica e definita positiva. Allora, il metodo del gradiente coniugato per risolvere (5.1) converge al più in n iterazioni (in aritmetica esatta). Inoltre, l'errore $\mathbf{e}^{(k)}$ alla k-esima iterazione (con $k < n$) è ortogonale a $\mathbf{p}^{(j)}$, per $j = 0, \dots, k-1$ e*

$$\|\mathbf{e}^{(k)}\|_A \leq \frac{2c^k}{1+c^{2k}} \|\mathbf{e}^{(0)}\|_A, \text{ con } c = \frac{\sqrt{K_2(A)} - 1}{\sqrt{K_2(A)} + 1}. \tag{5.50}$$

Di conseguenza, in assenza di errori di arrotondamento, il metodo GC può essere visto come un metodo diretto in quanto termina dopo un numero finito di iterazioni. D'altra parte, per matrici di grande dimensione, viene usualmente impiegato come un metodo iterativo ed arrestato quando uno stimatore dell'errore (come ad esempio il residuo) è minore di una tolleranza assegnata. Grazie alla (5.50), la dipendenza del fattore di riduzione dell'errore dal numero di condizionamento della matrice è più favorevole di quella del metodo del gradiente (per la presenza della radice quadrata di $K_2(A)$).

Anche per il metodo GC si può considerare una versione precondizionata (il metodo GCP) con un precondizionatore P simmetrico e definito positivo: dato $\mathbf{x}^{(0)}$ e ponendo $\mathbf{r}^{(0)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(0)}$, $\mathbf{z}^{(0)} = \mathbf{P}^{-1}\mathbf{r}^{(0)}$ e $\mathbf{p}^{(0)} = \mathbf{z}^{(0)}$, la k-esima iterazione diventa

$$\begin{aligned}
\alpha_k &= \frac{\mathbf{p}^{(k)T} \mathbf{r}^{(k)}}{\mathbf{p}^{(k)T} A \mathbf{p}^{(k)}} \\
\mathbf{x}^{(k+1)} &= \mathbf{x}^{(k)} + \alpha_k \mathbf{p}^{(k)} \\
\mathbf{r}^{(k+1)} &= \mathbf{r}^{(k)} - \alpha_k A \mathbf{p}^{(k)} \\
P \mathbf{z}^{(k+1)} &= \mathbf{r}^{(k+1)} \\
\beta_k &= \frac{(A \mathbf{p}^{(k)})^T \mathbf{z}^{(k+1)}}{(A \mathbf{p}^{(k)})^T \mathbf{p}^{(k)}} \\
\mathbf{p}^{(k+1)} &= \mathbf{z}^{(k+1)} - \beta_k \mathbf{p}^{(k)}
\end{aligned} \tag{5.51}$$

pcg Il metodo GCP è implementato nella *function* MATLAB **pcg**

Esempio 5.15 (Metodi Diretti vs metodi iterativi) Riprendiamo l'Esempio 5.8 sulla matrice di Hilbert A_n e risolviamo il sistema per diversi valori di n con i metodi precondizionati del gradiente e del gradiente coniugato. Come precondizionatore abbiamo scelto una matrice diagonale D la cui diagonale principale coincide con quella della matrice di Hilbert. Prendiamo $\mathbf{x}^{(0)} = \mathbf{0}^T$ ed arrestiamo il metodo quando il residuo relativo è minore in norma di 10^{-6} . Nella Tabella 5.2 riportiamo gli errori assoluti (rispetto alla soluzione esatta) ottenuti con i metodi iterativi precedentemente indicati e l'errore che si ottiene utilizzando il comando \backslash di MATLAB. Per quest'ultimo metodo l'errore degenera al crescere di n . D'altro canto possiamo apprezzare il benefico effetto sul numero di iterazioni richieste dovuto ad una opportuna scelta del metodo iterativo (nella fattispecie, il gradiente coniugato precondizionato GCP).

		\	GP		GCP	
n	$K(A_n)$	Errore	Errore	N.ro iterazioni	Errore	N.ro Iterazioni
4	1.55e+04	2.96e-13	1.74e-02	995	2.24e-02	3
6	1.50e+07	4.66e-10	8.80e-03	1813	9.50e-03	9
8	1.53e+10	4.38e-07	1.78e-02	1089	2.13e-02	4
10	1.60e+13	3.79e-04	2.52e-03	875	6.98e-03	5
12	1.79e+16	0.24e+00	1.76e-02	1355	1.12e-02	5
14	4.07e+17	0.26e+02	1.46e-02	1379	1.61e-02	5

Tabella 5.2. Errori calcolati utilizzando i metodi iterativi GP e GCP ed il metodo diretto implementato nel comando \backslash di MATLAB per la soluzione del sistema di Hilbert al variare della dimensione n della matrice A_n . Per i metodi iterativi è stato riportato anche il numero di iterazioni effettuate

5.9.1 Il caso non simmetrico

Il metodo GC è un esempio dei cosiddetti metodi di *Krylov* (o di Lanczos) che possono essere usati per la soluzione di sistemi non necessariamente simmetrici. Alcuni di essi condividono con il metodo GC la proprietà di terminazione finita, ossia in aritmetica esatta restituiscono la soluzione esatta del sistema in un numero finito di iterazioni, anche per un sistema non simmetrico. Un esempio notevole in questo senso è il metodo *GMRES* (Generalized Minimum RESidual). Per la sua descrizione rimandiamo ad esempio a [Axe94], [Saa96] e [vdV03]. Tale metodo è disponibile nel toolbox MATLAB `sparfun` sotto il nome di `gmres`. Un altro metodo di questa famiglia, che non ha la proprietà di terminazione finita, ma che richiede un minor sforzo computazionale, è il metodo del *gradiente coniugato quadrato* (CGS). Tra le sue varianti citiamo il metodo Bi-CGStab che è caratterizzato da una convergenza più regolare del CGS. Tutti questi metodi sono presenti nel toolbox MATLAB `sparfun`.

gmres

Si vedano gli Esercizi 5.15-5.17.



5.10 Quando arrestare un metodo iterativo?

Teoricamente, per convergere alla soluzione esatta, i metodi iterativi necessitano di un numero infinito di iterazioni. Nella pratica ciò non è né ragionevole, né necessario. Infatti, in generale non serve la soluzione esatta, ma una sua approssimazione $\mathbf{x}^{(k)}$ per la quale si possa garantire che l'errore sia inferiore ad una tolleranza ε desiderata. Tuttavia, poiché l'errore è a sua volta una quantità incognita (dipendendo dalla soluzione esatta), serve un opportuno stimatore dell'errore *a posteriori*, cioè determinabile a partire da quantità già calcolate.

Un primo stimatore è costituito dal *residuo* ad ogni iterazione

$$\mathbf{r}^{(k)} = \mathbf{b} - \mathbf{Ax}^{(k)},$$

essendo $\mathbf{x}^{(k)}$ la soluzione approssimata alla k -esima iterazione.

Più precisamente, potremmo arrestare il nostro metodo iterativo al primo passo k_{min} in corrispondenza del quale

$$\|\mathbf{r}^{(k_{min})}\| \leq \varepsilon \|\mathbf{b}\|.$$

Ponendo $\hat{\mathbf{x}} = \mathbf{x}^{(k_{min})}$ e $\mathbf{r} = \mathbf{r}^{(k_{min})}$ in (5.26) otterremo

$$\frac{\|\mathbf{e}^{(k_{min})}\|}{\|\mathbf{x}\|} \leq \varepsilon K(\mathbf{A}),$$

ovvero una stima per l'errore relativo. Pertanto, il controllo del residuo è significativo solo se il numero di condizionamento della matrice \mathbf{A} del sistema è ragionevolmente piccolo.

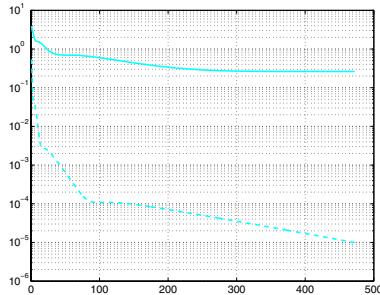


Figura 5.7. Andamento al variare di k di $\|\mathbf{r}^{(k)}\|/\|\mathbf{r}^{(0)}\|$ (in linea tratteggiata) e dell'errore $\|\mathbf{x} - \mathbf{x}^{(k)}\|$ (in linea continua) per il metodo di Gauss-Seidel applicato al sistema di Hilbert dell'Esempio 5.16

Esempio 5.16 Consideriamo il sistema lineare (5.1) in cui $A = A_{20}$ è la matrice di Hilbert di dimensione 20 introdotta nell'Esempio 5.8 e \mathbf{b} viene costruito in modo tale che la soluzione esatta sia $\mathbf{x} = (1, 1, \dots, 1)^T$. Essendo A simmetrica e definita positiva, il metodo di Gauss-Seidel sicuramente converge. Utilizziamo il Programma 12 per risolvere il sistema con \mathbf{x}_0 uguale al vettore nullo e richiedendo una tolleranza tol sul residuo pari a 10^{-5} . Il metodo converge in 472 iterazioni, ma l'errore calcolato è pari in norma a 0.26. Questo comportamento è dovuto al fatto che la matrice è estremamente mal condizionata, essendo $K(A) \simeq 10^{17}$. In Figura 5.7 viene mostrata l'evoluzione della norma del residuo (diviso per la norma del residuo iniziale) e dell'errore $\mathbf{e}^{(k)}$ al variare del numero di iterazioni.

Un criterio alternativo è basato sull'uso di un altro stimatore, il cosiddetto *incremento* $\delta^{(k)} = \mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}$. Più precisamente, il metodo iterativo viene arrestato al primo passo k_{min} per il quale

$$\|\delta^{(k_{min})}\| \leq \varepsilon \|\mathbf{b}\|.$$

Nel caso particolare in cui B sia simmetrica e definita positiva, avremo

$$\|\mathbf{e}^{(k)}\| = \|\mathbf{e}^{(k+1)} - \delta^{(k)}\| \leq \rho(B)\|\mathbf{e}^{(k)}\| + \|\delta^{(k)}\|$$

e, dovendo avere $\rho(B) < 1$ per garantire la convergenza, possiamo scrivere

$$\|\mathbf{e}^{(k)}\| \leq \frac{1}{1 - \rho(B)} \|\delta^{(k)}\| \quad (5.52)$$

Si può quindi concludere che il controllo dell'incremento è significativo soltanto se $\rho(B)$ è molto più piccolo di uno, poiché in tal caso l'errore sarà dello stesso ordine di grandezza dell'incremento.

La stessa conclusione vale anche qualora B non sia simmetrica e definita positiva (com'è ad esempio il caso dei metodi di Jacobi e di Gauss-Seidel), anche se in tal caso non vale più la (5.52).

Esempio 5.17 Consideriamo un sistema con matrice $A \in \mathbb{R}^{50 \times 50}$ tridiagonale simmetrica avente elementi diagonali pari a 2.001 e sottodiagonali pari a 1; al solito, il termine noto del sistema verrà scelto in modo che il vettore $(1, \dots, 1)^T$ sia la soluzione esatta. Essendo A tridiagonale a dominanza diagonale stretta il metodo di Gauss-Seidel convergerà due volte più rapidamente di quello di Jacobi (come osservato nella Proposizione 5.4). Utilizziamo il Programma 12 per risolvere il sistema con l'accortezza di sostituire al criterio d'arresto basato sul residuo quello basato sull'incremento. Partendo da un vettore iniziale nullo e richiedendo una tolleranza $\text{tol} = 10^{-5}$, il programma restituisce dopo ben 1604 iterazioni una soluzione affetta da un errore piuttosto grande pari a 0.0029. Il motivo risiede nel fatto che il raggio spettrale della matrice di iterazione è pari a 0.9952, cioè molto vicino a 1. Se gli elementi diagonali fossero stati pari a 3, avremmo invece ottenuto convergenza in 17 iterazioni con un errore dell'ordine di 10^{-5} ; ora infatti il raggio spettrale della matrice di iterazione è pari a 0.4428.

Si vedano gli Esercizi 5.15-5.17.

Riassumendo



1. Un metodo iterativo per la risoluzione di un sistema lineare costruisce, a partire da un vettore iniziale $\mathbf{x}^{(0)}$, una successione di vettori $\mathbf{x}^{(k)}$ ai quali si richiede di convergere alla soluzione esatta per $k \rightarrow \infty$;
2. condizione necessaria e sufficiente affinché un metodo iterativo converga per ogni possibile scelta di $\mathbf{x}^{(0)}$ è che il raggio spettrale della matrice di iterazione sia minore di 1;
3. i metodi iterativi più classici sono quelli di Jacobi e di Gauss-Seidel. Condizione sufficiente per la convergenza di tali metodi è che la matrice del sistema da risolvere sia a dominanza diagonale stretta (ma anche simmetrica e definita positiva nel caso del metodo di Gauss-Seidel);
4. nel metodo di Richardson la convergenza viene accelerata introducendo ad ogni iterazione un parametro e (eventualmente) una opportuna matrice di precondizionamento;
5. con il metodo del gradiente coniugato la soluzione esatta di un sistema simmetrico definito positivo viene calcolata in un numero finito di iterazioni (in aritmetica esatta). Questo metodo può essere generalizzato al caso di un sistema non simmetrico;
6. due sono i possibili criteri d'arresto per un metodo iterativo: il controllo del residuo ed il controllo dell'incremento. Il primo è significativo se la matrice del sistema è ben condizionata, il secondo se il raggio spettrale della matrice di iterazione è decisamente < 1 .

5.11 Ed ora: metodi diretti o iterativi?

In questa sezione proponiamo un confronto fra metodi diretti e metodi iterativi su alcuni semplici casi test. Premettiamo che per la risoluzione di sistemi di piccole dimensioni, il problema non è così critico come nel caso in cui le matrici siano molto grandi, anche se la scelta fra un metodo iterativo ed uno diretto è un dilemma piuttosto frequente nelle applicazioni. Essa sarà in generale basata sull'esperienza e dipenderà primariamente dal tipo di matrice del sistema lineare in questione (di grande o di media dimensione, simmetrica o non simmetrica, sparsa o piena,...), ma anche dal tipo di risorse a disposizione (rapidità di accesso a grandi memorie, processori veloci, ecc.) Inoltre, nei nostri test il confronto non sarà del tutto leale: il solver diretto presente in MATLAB (implementato nella built-in function \) è infatti ottimizzato e compilato, mentre i risolutori iterativi, come `pcg`, non lo sono. Ciononostante potremo trarre qualche interessante conclusione. I tempi di CPU riportati sono stati ottenuti su un processore Intel Pentium M 1.60GHz con una cache di 2048KB ed una memoria RAM di 1 GByte.

5.11.1 Un sistema lineare sparso

In questo primo caso test risolveremo sistemi sparsi generati dalla discretizzazione con il metodo delle differenze finite del problema di Poisson sul quadrato $(-1, 1)^2$ (si veda la Sezione 8.2). In particolare, le matrici verranno generate a partire da una decomposizione del quadrato in reticolati uniformi di passo $h = 1/N$, con N crescente. La dimensione di ciascun sistema lineare sarà N^2 (per la generazione delle matrici è stato utilizzato il Programma 26). Il grafico di Figura 5.8 a sinistra riporta la struttura della matrice di dimensione $N^2 = 256$ (ottenuta con il comando `spy`): come si vede si tratta di una matrice sparsa con 5 diagonali non nulle. Le matrici considerate sono tutte simmetriche e definite positive ed hanno un numero di condizionamento che si comporta come h^{-2} , sono quindi tanto più malcondizionate quanto più h decresce. Per risolvere i corrispondenti sistemi lineari confronteremo il metodo di fattorizzazione di Cholesky ed il metodo del gradiente coniugato precondizionato (GCP) con una fattorizzazione incompleta di Cholesky (un precondizionatore algebrico ottenuto usando il comando `cholinc`), nonché il metodo implementato nel comando \ di MATLAB che, in tal caso, si traduce in un metodo *ad hoc* per matrici a banda simmetriche. Le matrici sono state tutte memorizzate con il formato `sparse` di MATLAB. Per il metodo GCP richiederemo che a convergenza il residuo relativo sia in norma minore di 10^{-14} e conteggeremo nel tempo di calcolo anche il tempo necessario per costruire il precondizionatore.

Nel grafico di destra di Figura 5.8 confrontiamo i tempi di calcoli dei tre metodi in esame al crescere della dimensione della matrice. Come si

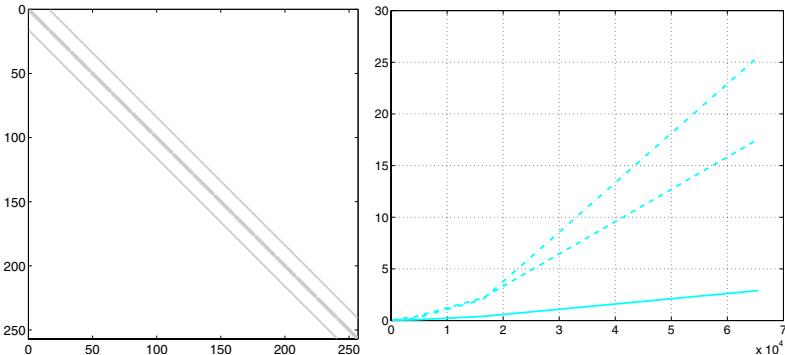


Figura 5.8. La struttura della matrice del primo caso test (a sinistra). Nel grafico di destra, il confronto fra tempi di CPU necessari per la risoluzione dei sistemi lineari corrispondenti: in linea piena usando il comando `\`, in linea tratto-punto la fattorizzazione di Cholesky ed in linea tratteggiata il metodo GCP

vede il metodo diretto usato dal comando `\` è di gran lunga il migliore: in effetti questa variante del metodo di eliminazione di Gauss è particolarmente efficace nel trattamento di matrici sparse con banda ragionevolmente ristretta. Il metodo GCP è a sua volta vincente rispetto alla fattorizzazione di Cholesky. A questo riguardo è importante osservare quanto essenziale sia la scelta del precondizionatore: non precondizionato il GC richiederebbe un numero di iterazioni decisamente più elevato per soddisfare il criterio d'arresto (ad esempio, per $N^2 = 4096$ sono richieste 325 iterazioni contro le 19 del caso precondizionato) ed i tempi di calcolo crescerebbero rendendo di fatto preferibile la fattorizzazione di Cholesky.

5.11.2 Un sistema lineare con banda estesa

I sistemi lineari che consideriamo si riferiscono ancora alla discretizzazione del problema di Poisson sul quadrato $(-1, 1)^2$, ma sono stati generati impiegando una discretizzazione basata su metodi spettrali con formule di quadratura di tipo Gauss-Lobatto-Legendre (si vedano ad esempio [Qua06] e [CHQZ06]). Anche se il numero di punti che formano il reticolo è lo stesso utilizzato con le differenze finite, contrariamente a quanto accade con quest'ultimo metodo, l'approssimazione delle derivate di una funzione coinvolge buona parte dei nodi di discretizzazione e, di conseguenza, la struttura della matrice del sistema lineare corrispondente è decisamente meno sparsa della precedente (anche se continueremo a memorizzare queste matrici con il formato `sparse` di MATLAB). Per dare un'idea abbiamo riportato in Figura 5.9 a sinistra la struttura di una matrice spettrale di dimensione $N^2 = 256$: se confrontata con la struc-

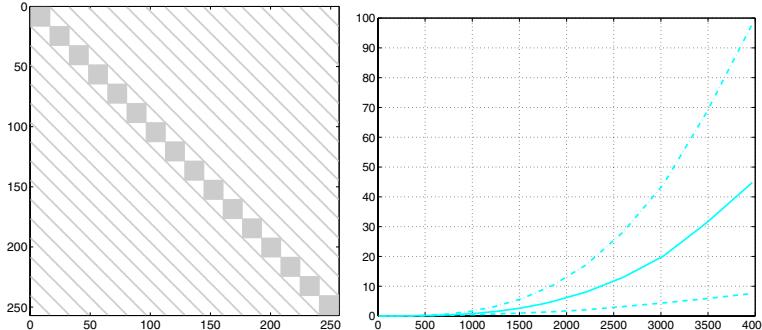


Figura 5.9. La struttura di una delle matrici usate nel secondo caso test (a sinistra). A destra, il confronto fra tempi di CPU: in linea piena usando il metodo \backslash , in linea tratto-punto la fattorizzazione di Cholesky ed in linea tratteggiata il metodo GCP

tura di quella delle differenze finite di Figura 5.8 possiamo visivamente renderci conto di quanto questa sia più piena e con banda maggiore. In effetti quest'ultima presenta 7936 coefficienti non nulli contro i 1216 della matrice delle differenze finite.

I tempi di calcolo riportati nel grafico di destra in Figura 5.9 mostrano come per questo sistema il metodo GCP, precondizionato con una fattorizzazione incompleta di Cholesky, risulti di gran lunga il migliore.

Una prima conclusione che possiamo trarre è che per sistemi con matrice simmetrica e definita positiva, sparsi, ma con bande grandi, il metodo GCP risulta più efficiente del miglior metodo diretto implementato in MATLAB (che è comunque uno dei migliori disponibili e che, in questo caso, non coincide con la fattorizzazione di Cholesky, avendo usato il formato di memorizzazione `sparse` per le matrici). Questo naturalmente a patto di disporre di un efficiente precondizionatore, requisito non sempre facilmente realizzabile.

Teniamo inoltre conto che i metodi diretti richiedono in generale una maggior quantità di memoria rispetto a quelli iterativi e, di conseguenza, possono diventare problematici in applicazioni di grandi dimensioni.

5.11.3 Un sistema con matrice piena

In MATLAB è possibile accedere ad una raccolta di matrici di varia natura utilizzando il comando `gallery`. In particolare, per i nostri scopi selezioniamo con il comando `A=gallery('riemann',n)` la cosiddetta matrice di Riemann di dimensione `n`, cioè una matrice $n \times n$ tale che $\det(A) = \mathcal{O}(n!n^{-1/2+\epsilon})$ per ogni $\epsilon > 0$. Questa matrice è piena e non simmetrica: per quest'ultimo motivo come metodo iterativo useremo invece del metodo GCP il metodo GMRES (si veda il paragrafo 5.9.1).

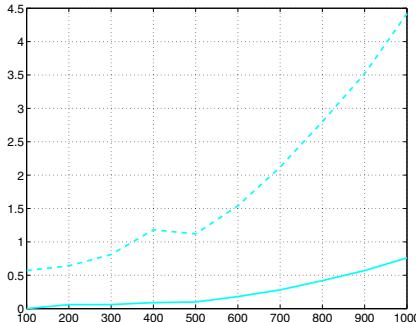


Figura 5.10. Confronto fra tempi di CPU per il terzo caso test: in linea piena usando il metodo \backslash ed in linea tratteggiata il metodo GMRES

Essendo inoltre non mal condizionata (ha un condizionamento che al massimo vale circa 10000 per dimensione 1000), utilizzeremo questo metodo senza alcun precondizionatore. Per arrestare il metodo richiederemo che il residuo sia in norma minore di 10^{-14} . Per quanto riguarda l'approccio diretto useremo soltanto il metodo di eliminazione implementato nel comando \backslash (essenzialmente il metodo di fattorizzazione LU).

Al crescere di n risolviamo i sistemi lineari di matrice A e termine noto fatto in modo tale che la soluzione esatta del sistema sia il vettore $\mathbf{1}^T$. In Figura 5.10 riportiamo i tempi di CPU (in secondi) ottenuti per n compresi tra 100 e 1000. Come si vede il metodo diretto è più favorevole del metodo iterativo.

5.11.4 Un sistema lineare con matrice sparsa non a banda e non simmetrica

Consideriamo sistemi lineari generati dall'approssimazione di un problema di diffusione-trasporto-reazione bidimensionale, simile a quello indicato nella (8.12) nel caso monodimensionale. Come metodo di approssimazione utilizziamo gli elementi finiti lineari (che verranno introdotti, sempre per il caso monodimensionale, nel paragrafo 8.12). Essi conducono ad un sistema lineare con matrice e termine noto opportuni. In due dimensioni il metodo degli elementi finiti richiede che il dominio di calcolo venga preliminarmente decomposto in triangoli non sovrapposti che lo ricoprono completamente (e che costituiscono la griglia di calcolo). Le incognite degli elementi finiti lineari sono i valori assunti dalla funzione approssimante nei vertici di tali triangoli. La matrice associata è sparsa. La distribuzione degli elementi non nulli dipende dal modo in cui vengono numerati i vertici della griglia (si veda la Figura 5.11 a sinistra per un esempio). Inoltre, la mancanza di simmetria della matrice non è evidente dalla rappresentazione della sola struttura che è per costruzione solitamente simmetrica. Minore sarà il *diametro h* dei

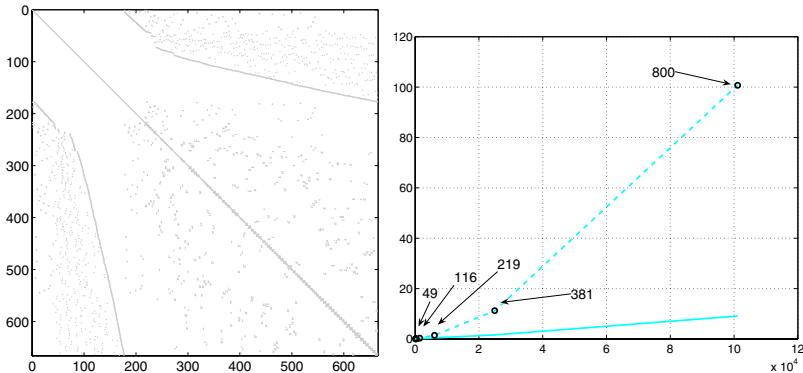


Figura 5.11. La struttura di una delle matrici usate nel quarto caso test (a sinistra). A destra, il confronto fra tempi di CPU: in linea piena usando il comando `\` ed in linea tratteggiata usando il metodo iterativo Bi-CGstab. I numeri indicati si riferiscono al numero di iterazioni effettuate dal Bi-CGstab

triangoli (ossia la lunghezza del lato maggiore), maggiore sarà la dimensione del sistema lineare da risolvere. Abbiamo confrontato i tempi di CPU necessari per risolvere i sistemi che si trovano per h pari a 0.1, 0.05, 0.025, 0.0125 e 0.0063 con la libreria UMFPACK (richiamata in questo caso automaticamente dal comando `\`) e con il metodo iterativo Bi-CGStab non precondizionato, che è uno dei metodi implementati in MATLAB per la risoluzione di sistemi non simmetrici. In ascissa abbiamo riportato il numero di incognite che vanno da 64 per $h = 0.1$ a 101124 per $h = 0.0063$. Come si vede il metodo diretto è in questo caso decisamente più conveniente del metodo iterativo. Se si usasse come precondizionatore per il metodo Bi-CGstab una fattorizzazione LU incompleta (un precondizionatore algebrico ottenuto usando il comando `luinc`) il numero di iterazioni diminuirebbe, ma i tempi di calcolo del metodo iterativo crescerebbero ulteriormente.

`luinc`

5.11.5 In conclusione

Il confronto considerato, seppur estremamente ridotto, mette in luce alcuni aspetti importanti: i metodi diretti nelle loro versioni più sofisticate (come quella implementata nel comando `\` di MATLAB) in generale sono computazionalmente più efficienti dei metodi iterativi quando per questi ultimi non si usano precondizionatori efficaci. Soffrono tuttavia in modo maggiore il malcondizionamento delle matrici (si veda anche l’Esempio 5.15) e richiedono notevoli risorse in termini di memoria. Un aspetto infine da tener presente, che non è emerso dai nostri semplici esempi, consiste nel fatto che per impiegare un metodo diretto è necessario conoscere i coefficienti della matrice del sistema, mentre per un

metodo iterativo basta saper valutare l'effetto della matrice quando applicata ad un vettore noto (come ad esempio il residuo). Questo aspetto rende i metodi iterativi particolarmente interessanti in quei problemi in cui la matrice non si genera esplicitamente.

5.12 Cosa non vi abbiamo detto

Per sistemi lineari di grande dimensione sono disponibili diverse varianti efficienti della fattorizzazione LU di Gauss. Tra quelle più note, ricordiamo il cosiddetto *metodo multifrontale*, basato su un opportuno rordinamento delle incognite del sistema in modo da garantire che i fattori L ed U siano i più sparsi possibile. Questo metodo è alla base della libreria UMFPACK richiamata, come abbiamo visto, dal comando \ in MATLAB in certe circostanze. Per approfondimenti si vedano [GL89] e [DD95].

Per quanto riguarda i metodi iterativi, ricordiamo che il metodo del gradiente coniugato e il metodo GMRES sono due esempi della famiglia dei *metodi di Krylov*. Per una loro descrizione si vedano ad esempio [Axe94], [Saa96] e [vdV03].

Come abbiamo avuto modo di osservare, un metodo iterativo, per quanto efficiente, convergerà lentamente se la matrice del sistema è mal condizionata. Per questo motivo sono state sviluppate numerose strategie di precondizionamento (si veda ad esempio [dV89] e [vdV03]). Tra di esse ve ne sono alcune di tipo puramente algebrico, basate sulle fattorizzazioni LU (o di Cholesky) incomplete ed implementate in MATLAB nelle funzioni `luinc` e `cholinc`. Altre strategie vengono invece sviluppate *ad hoc* avvantaggiandosi della conoscenza del problema fisico che ha originato il sistema lineare da risolvere.

Infine, ricordiamo gli algoritmi di tipo *multigrid* che sono basati sulla risoluzione di una gerarchia di sistemi di dimensione variabile, somiglianti al sistema di partenza, scelti in modo da perseguire una strategia di riduzione progressiva dell'errore (si vedano ad esempio [Hac85], [Wes04] e [Hac94]).

5.13 Esercizi

Esercizio 5.1 Data una matrice $A \in \mathbb{R}^{n \times n}$ si determini al variare di n il numero di operazioni richiesto per il calcolo del determinante con la formula ricorsiva (1.8).

Esercizio 5.2 Si usi il comando MATLAB `magic(n)`, $n=3, 4, \dots, 500$, per costruire i quadrati magici di ordine n , cioè quelle matrici i cui coefficienti sono tali che la somma per righe, per colonne o per diagonali si mantiene

costante. Per ogni n si calcolino il determinante con il comando `det`, introdotto nel paragrafo 1.3 ed il tempo di CPU utilizzato per tale operazione tramite il comando `cputime`. Si approssimino i dati così ottenuti con il metodo dei minimi quadrati e si deduca che i tempi di CPU crescono approssimativamente come n^3 .

Esercizio 5.3 Per quali valori di ε la matrice definita nella (5.12) non soddisfa le ipotesi della Proposizione 5.1? Per quali valori essa è singolare? È comunque possibile calcolare in tal caso la fattorizzazione LU?

Esercizio 5.4 Si verifichi che il numero di operazioni necessario per calcolare la fattorizzazione LU di una matrice quadrata A di dimensione n è approssimativamente $2n^3/3$.

Esercizio 5.5 Si mostri che la fattorizzazione LU di una matrice A può essere usata per calcolarne l'inversa (si osservi che il j -esimo vettore colonna \mathbf{y}_j di A^{-1} soddisfa il sistema lineare $A\mathbf{y}_j = \mathbf{e}_j$, dove \mathbf{e}_j è il vettore le cui componenti sono tutte nulle fuorché la j -esima che vale 1).

Esercizio 5.6 Si calcolino i fattori L ed U per la matrice dell'Esempio 5.7 e si verifichi che la fattorizzazione LU è inaccurata.

Esercizio 5.7 Si spieghi per quale motivo la strategia del pivoting per righe non è conveniente nel caso di matrici simmetriche.

Esercizio 5.8 Si consideri il sistema lineare $A\mathbf{x} = \mathbf{b}$ con

$$A = \begin{bmatrix} 2 & -2 & 0 \\ \varepsilon - 2 & 2 & 0 \\ 0 & -1 & 3 \end{bmatrix},$$

\mathbf{b} tale per cui la soluzione sia $\mathbf{x} = (1, 1, 1)^T$ e ε un numero reale positivo. Si calcoli la fattorizzazione di Gauss di A e si verifichi che l'elemento $l_{32} \rightarrow \infty$ quando $\varepsilon \rightarrow 0$. Ciò nonostante, si verifichi al calcolatore che la soluzione del sistema lineare ottenuta tramite il processo di fattorizzazione è accurata.

Esercizio 5.9 Si considerino i sistemi lineari $A_i\mathbf{x}_i = \mathbf{b}_i$, $i = 1, 2, 3$, con

$$A_1 = \begin{bmatrix} 15 & 6 & 8 & 11 \\ 6 & 6 & 5 & 3 \\ 8 & 5 & 7 & 6 \\ 11 & 3 & 6 & 9 \end{bmatrix}, A_i = (A_1)^i, i = 2, 3,$$

e \mathbf{b}_i tali che la soluzione sia sempre $\mathbf{x}_i = (1, 1, 1, 1)^T$. Si risolvano tali sistemi utilizzando la fattorizzazione di Gauss con *pivoting* per righe e si commentino i risultati ottenuti.

Esercizio 5.10 Si dimostri che per una matrice A simmetrica e definita positiva si ha $K(A^2) = (K(A))^2$.

Esercizio 5.11 Si analizzino le proprietà di convergenza dei metodi di Jacobi e di Gauss-Seidel per la soluzione di sistemi lineari di matrice

$$A = \begin{bmatrix} \alpha & 0 & 1 \\ 0 & \alpha & 0 \\ 1 & 0 & \alpha \end{bmatrix}, \quad \alpha \in \mathbb{R}.$$

Esercizio 5.12 Si dia una condizione sufficiente sul numero reale β affinché i metodi di Jacobi e di Gauss-Seidel convergano entrambi quando applicati alla risoluzione di un sistema di matrice

$$A = \begin{bmatrix} -10 & 2 \\ \beta & 5 \end{bmatrix}. \quad (5.53)$$

Esercizio 5.13 Per la risoluzione del sistema lineare $A\mathbf{x} = \mathbf{b}$ con $A \in \mathbb{R}^{n \times n}$, si consideri il *metodo di rilassamento*: dato $\mathbf{x}^{(0)} = (x_1^{(0)}, \dots, x_n^{(0)})^T$, per $k = 0, 1, \dots$ si calcoli

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

per $i = 1, \dots, n$, dove ω è un parametro reale. Si riporti la matrice di iterazione e si verifichi che la condizione $0 < \omega < 2$ è necessaria per la convergenza di questo metodo. Si noti che per $\omega = 1$ esso coincide con il metodo di Gauss-Seidel. Se $1 < \omega < 2$ il metodo è noto come *SOR (successive over-relaxation)*.

Esercizio 5.14 Si consideri il sistema lineare $A\mathbf{x} = \mathbf{b}$ con $A = \begin{bmatrix} 3 & 2 \\ 2 & 6 \end{bmatrix}$ e si stabilisca, senza calcolare il raggio spettrale della matrice di iterazione, se il metodo di Gauss-Seidel converge.

Esercizio 5.15 Si calcoli la prima iterazione per i metodi di Jacobi, Gauss-Seidel e gradiente precondizionato con la diagonale di A quando applicati alla soluzione del sistema (5.47), posto $\mathbf{x}^{(0)} = (1, 1/2)^T$.

Esercizio 5.16 Si dimostri (5.43) e che

$$\rho(B_{\alpha_{opt}}) = \frac{\lambda_{max} - \lambda_{min}}{\lambda_{max} + \lambda_{min}} = \frac{K(P^{-1}A) - 1}{K(P^{-1}A) + 1} \quad (5.54)$$

Esercizio 5.17 Consideriamo un sistema di $n = 20$ industrie che producono 20 beni diversi. Riferendosi al modello di Leontief, introdotto nel Problema 5.3, si supponga che la matrice C abbia i seguenti coefficienti $c_{ij} = i + j$ per $i, j = 1, \dots, n$, mentre $b_i = i$, per $i = 1, \dots, 20$. Si dica se è possibile risolvere tale sistema con il metodo del gradiente. Osservando che se A è una matrice non singolare, allora la matrice $A^T A$ è simmetrica e definita positiva, si proponga comunque un metodo basato sul gradiente per risolvere il sistema dato.

6

Autovalori ed autovettori

Consideriamo il seguente problema: data una matrice quadrata $A \in \mathbb{C}^{n \times n}$, trovare uno scalare λ (reale o complesso) ed un vettore $\mathbf{x} \in \mathbb{C}^n$ non nullo tali che

$$A\mathbf{x} = \lambda\mathbf{x}. \quad (6.1)$$

Ogni λ che soddisfi (6.1) è detto *autovalore* di A , mentre \mathbf{x} è un corrispondente *autovettore*. Evidentemente \mathbf{x} non è unico in quanto se \mathbf{x} è autovettore anche $\alpha\mathbf{x}$ lo è, qualunque sia il numero $\alpha \neq 0$, reale o complesso. Qualora sia noto \mathbf{x} , λ può essere calcolato usando il *quoziente di Rayleigh* $\mathbf{x}^* A \mathbf{x} / (\mathbf{x}^* \mathbf{x})$, dove $\mathbf{x}^* = \bar{\mathbf{x}}^T$ è il vettore con componente i -esima pari a \bar{x}_i .

Un numero λ è autovalore di A se è radice del seguente polinomio di grado n (detto *polinomio caratteristico* di A)

$$p_A(\lambda) = \det(A - \lambda I).$$

Pertanto una matrice quadrata di dimensione n ha esattamente n autovalori (reali o complessi), non necessariamente distinti fra loro. Facciamo notare che se gli elementi di A sono numeri reali, allora $p_A(\lambda)$ ha coefficienti reali e, di conseguenza, se A ammette come autovalore un numero complesso λ , anche il suo coniugato $\bar{\lambda}$ sarà un autovalore.

Diciamo infine che una matrice $A \in \mathbb{C}^{n \times n}$ è diagonalizzabile se esiste una matrice $U \in \mathbb{C}^{n \times n}$ invertibile tale che

$$U^{-1}AU = \Lambda = \text{diag}(\lambda_1, \dots, \lambda_n), \quad (6.2)$$

in particolare, le colonne di U sono gli autovettori di A e formano una base per \mathbb{C}^n .

Consideriamo ora una matrice rettangolare $A \in \mathbb{C}^{m \times n}$. In tal caso non si parla più di autovalori, ma si può dimostrare che esistono sempre due matrici $U \in \mathbb{C}^{m \times m}$ e $V \in \mathbb{C}^{n \times n}$ unitarie (cioè con $U^{-1} = U^*$ e $V^{-1} = V^*$) tali che

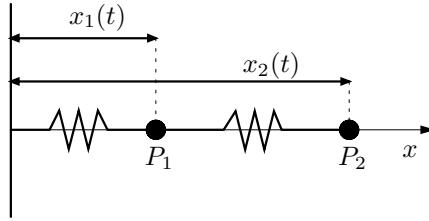


Figura 6.1. Un sistema di due corpi puntiformi di ugual massa collegati da molle

$$\mathbf{U}^* \mathbf{A} \mathbf{V} = \boldsymbol{\Sigma} = \text{diag}(\sigma_1, \dots, \sigma_p) \in \mathbb{R}^{m \times n} \quad \text{con } p = \min(m, n) \quad (6.3)$$

e $\sigma_1 \geq \dots \geq \sigma_p \geq 0$. La (6.3) è detta *decomposizione in valori singolari* (o *singular value decomposition*, in breve, SVD) di \mathbf{A} ed i σ_i sono detti i *valori singolari* di \mathbf{A} .

Problema 6.1 (Dinamica) Consideriamo il sistema di Figura 6.1 formato da due corpi puntiformi P_1 e P_2 , entrambi di massa m , collegati fra loro da due molle uguali e liberi di muoversi lungo la direzione individuata dalla retta che li congiunge. Indichiamo con $x_i(t)$ la posizione occupata dal punto P_i al tempo t per $i = 1, 2$. Allora, per la seconda legge della dinamica, si ha

$$m \ddot{x}_1 = K(x_2 - x_1) - Kx_1,$$

$$m \ddot{x}_2 = K(x_1 - x_2),$$

dove K è il coefficiente di elasticità di entrambe le molle. Siamo interessati alle oscillazioni forzate periodiche cui corrisponde la soluzione $x_i = a_i \sin(\omega t + \phi)$, $i = 1, 2$, con $a_i \neq 0$. In tal caso, si trovano le relazioni

$$\begin{aligned} -ma_1\omega^2 &= K(a_2 - a_1) - Ka_1, \\ -ma_2\omega^2 &= K(a_1 - a_2), \end{aligned} \quad (6.4)$$

cioè un sistema 2×2 omogeneo che ha soluzione non banale $\mathbf{a} = (a_1, a_2)^T$ se e soltanto se il numero $\lambda = m\omega^2/K$ è un autovalore della matrice

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

Infatti, con questa definizione di λ , (6.4) diventa $\mathbf{A}\mathbf{a} = \lambda\mathbf{a}$. Poiché $p_{\mathbf{A}}(\lambda) = (2 - \lambda)(1 - \lambda) - 1$, i due autovalori sono $\lambda_1 \simeq 2.618$ e $\lambda_2 \simeq 0.382$ che corrispondono alle frequenze di oscillazione $\omega_i = \sqrt{K\lambda_i}/m$ ammesse dal sistema in esame.

Problema 6.2 (Viabilità interurbana) Consideriamo n città e sia \mathbf{A} una matrice i cui coefficienti a_{ij} valgono 1 se la città i è collegata con la città j , zero altrimenti. Si può dimostrare che le componenti dell'autovettore \mathbf{x} (di norma unitaria) associato all'autovalore di modulo massimo forniscono una misura della facilità d'accesso alle varie città. Nell'Esempio 6.2, sulla base della schematica rete ferroviaria della Lombardia riportata in Figura 6.2, determineremo in questo modo la città capoluogo di provincia più accessibile.

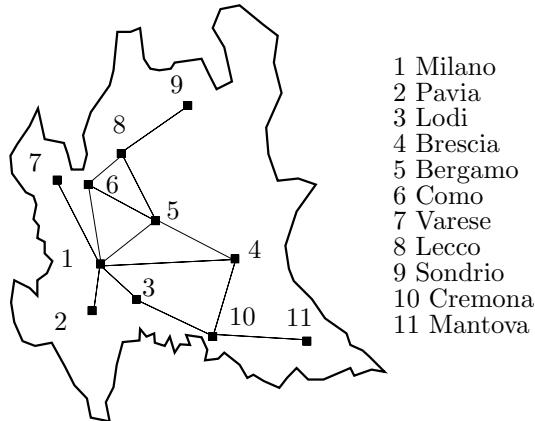


Figura 6.2. Una rappresentazione schematica delle connessioni ferroviarie in Lombardia fra i capoluoghi di provincia

Problema 6.3 (Demografia) La possibilità di prevedere l'andamento della popolazione di una certa specie (umana od animale che sia) ha portato in ambito demografico alla nascita di svariati modelli matematici. Il più semplice modello di popolazione, proposto da Lotka nel 1920 e formalizzato da Leslie vent'anni dopo, è basato sui tassi di mortalità e di fecondità per fasce di età. Supponiamo di avere $n + 1$ fasce d'età indicizzate da 0 a n . Denotiamo con $x_i^{(t)}$ il numero di femmine la cui età al tempo t si colloca nell' i -esima fascia. I valori $x_i^{(0)}$ sono noti. Denotiamo inoltre con s_i il tasso di sopravvivenza delle femmine con età che cade nella fascia i -esima e con m_i il numero medio di femmine generate da una femmina che ha età appartenente alla fascia i -esima. Il modello di Lotka e Leslie è descritto dalle seguenti equazioni

$$\begin{aligned} x_{i+1}^{(t+1)} &= \frac{x_i^{(t)}}{n} s_i, \quad i = 0, \dots, n-1, \\ x_0^{(t+1)} &= \sum_{i=0}^{n-1} x_i^{(t)} m_i. \end{aligned}$$

Le prime n equazioni descrivono l'andamento della popolazione, l'ultima la sua riproduzione. In forma matriciale abbiamo $\mathbf{x}^{(t+1)} = \mathbf{A}\mathbf{x}^{(t)}$, dove $\mathbf{x}^{(t)} = (x_0^{(t)}, \dots, x_n^{(t)})^T$ e \mathbf{A} è la *matrice di Leslie* data da

$$\mathbf{A} = \begin{bmatrix} m_0 & m_1 & \dots & \dots & m_n \\ s_0 & 0 & \dots & \dots & 0 \\ 0 & s_1 & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & s_{n-1} & 0 \end{bmatrix}.$$

Vedremo nel paragrafo 6.1 che la dinamica della popolazione è completamente determinata dall'autovalore di modulo massimo λ_1 di \mathbf{A} , mentre la distribuzione degli individui nelle differenti fasce d'età (normalizzata rispetto all'intera popolazione) si ottiene come limite di $\mathbf{x}^{(t)}$ per $t \rightarrow \infty$ ed è tale che $\mathbf{A}\mathbf{x} = \lambda_1\mathbf{x}$. Questo problema verrà risolto nell'Esercizio 6.2.

Problema 6.4 (Compressione di immagini) Il problema della compressione di un'immagine può essere affrontato utilizzando la decomposizione a valori singolari. In effetti, un'immagine in bianco e nero può essere memorizzata in una matrice A rettangolare $m \times n$ a coefficienti reali, dove m e n rappresentano il numero di *pixel* presenti nella direzione orizzontale ed in quella verticale rispettivamente, ed il coefficiente a_{ij} rappresenta l'intensità di grigio del pixel ij . Grazie alla (6.3) avremo allora che

$$A = \sigma_1 \mathbf{u}_1 \mathbf{v}_1^T + \sigma_2 \mathbf{u}_2 \mathbf{v}_2^T + \dots + \sigma_p \mathbf{u}_p \mathbf{v}_p^T, \quad (6.5)$$

avendo denotando con \mathbf{u}_i e \mathbf{v}_i l' i -simo vettore colonna delle matrici U e V , rispettivamente. La matrice A può quindi essere approssimata con la matrice A_k , ottenuta troncando la somma (6.5) ai primi k addendi con l'idea che, essendo i valori singolari ordinati in ordine decrescente, gli ultimi addendi siano quelli che meno influiscono sulla qualità dell'immagine. Parliamo di compressione in quanto, ad esempio, per trasmettere l'immagine approssimata A_k tra due computer è sufficiente trasferire i soli vettori \mathbf{u}_i e \mathbf{v}_i , nonché i valori singolari σ_i , con $i = 1, \dots, k$ e non invece tutti i coefficienti di A . Nell'Esempio 6.9 vedremo in azione questa tecnica.

Nel caso speciale in cui A sia una matrice diagonale o triangolare, gli autovalori sono dati direttamente dagli elementi diagonali. Qualora però A sia una matrice di forma generale con n sufficientemente grande, per il calcolo dei suoi autovalori (e dei corrispondenti autovettori) non è conveniente cercare di approssimare gli zeri di $p_A(\lambda)$. Per tale ragione si ricorre ad algoritmi numerici specifici che illustreremo nei prossimi paragrafi.

6.1 Il metodo delle potenze

Come abbiamo visto nei Problemi 6.2 e 6.3, non sempre è necessario conoscere lo *spettro* di A (cioè l'insieme di tutti i suoi autovalori); spesso, ci si può limitare ad individuare gli autovalori *estremi*, quelli cioè di modulo massimo e minimo.

Supponiamo di voler calcolare l'autovalore di modulo massimo di una matrice A quadrata di dimensione n . Indichiamolo con λ_1 ed ordiniamo i restanti autovalori di A come segue

$$|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \dots \geq |\lambda_n|. \quad (6.6)$$

In particolare, supponiamo che $|\lambda_1|$ sia distinto dai moduli dei restanti autovalori di A . Denotiamo con \mathbf{x}_1 l'autovettore (di lunghezza unitaria) associato a λ_1 . Se gli autovettori di A sono linearmente indipendenti, λ_1 e \mathbf{x}_1 possono essere calcolati tramite la seguente procedura, nota come *metodo delle potenze*:

dato un vettore iniziale arbitrario $\mathbf{x}^{(0)} \in \mathbb{C}^n$ e posto $\mathbf{y}^{(0)} = \mathbf{x}^{(0)} / \|\mathbf{x}^{(0)}\|$, per $k \geq 1$ si calcola

$$\mathbf{x}^{(k)} = \mathbf{A}\mathbf{y}^{(k-1)}, \mathbf{y}^{(k)} = \frac{\mathbf{x}^{(k)}}{\|\mathbf{x}^{(k)}\|}, \lambda^{(k)} = (\mathbf{y}^{(k)})^* \mathbf{A} \mathbf{y}^{(k)} \quad (6.7)$$

Si noti che $\mathbf{y}^{(k)} = \beta^{(k)} \mathbf{A}^k \mathbf{y}^{(0)}$, essendo $\beta^{(k)} = (\prod_{i=1}^k \|\mathbf{x}^{(i)}\|)^{-1}$ per $k \geq 1$. La presenza delle potenze di \mathbf{A} giustifica il nome del metodo.

Come vedremo nel prossimo paragrafo questo metodo genera una successione di vettori $\{\mathbf{y}^{(k)}\}$ di lunghezza unitaria tali da allinearsi, per $k \rightarrow \infty$, alla direzione dell'autovettore \mathbf{x}_1 . L'errore $\|\mathbf{y}^{(k)} - \mathbf{x}_1\|$ è proporzionale al rapporto $|\lambda_2/\lambda_1|$ per una matrice \mathbf{A} generica, a $|\lambda_2/\lambda_1|^2$ se \mathbf{A} è *hermitiana*, cioè tale che $\mathbf{A}^* = \mathbf{A}$ (e quindi simmetrica se \mathbf{A} è una matrice reale). Di conseguenza, si dimostra che $\lambda^{(k)} \rightarrow \lambda_1$ per $k \rightarrow \infty$.

Un'implementazione del metodo delle potenze è fornita nel Programma 13. La procedura iterativa si arresta alla prima iterazione k in corrispondenza della quale si ha

$$|\lambda^{(k)} - \lambda^{(k-1)}| < \varepsilon |\lambda^{(k)}|,$$

dove ε è una tolleranza assegnata. I parametri d'ingresso sono la matrice \mathbf{A} , il vettore iniziale \mathbf{x}_0 , la tolleranza tol impiegata nel criterio d'arresto ed il numero massimo di iterazioni consentite nmax . I parametri in uscita sono l'autovalore di modulo massimo lambda , l'autovettore associato ed il numero di iterazioni che sono state effettuate. Sia la matrice \mathbf{A} che il vettore \mathbf{x}_0 possono essere formati da numeri complessi.

Programma 13 - eigpower : il metodo delle potenze



```
function [lambda,x,iter]=eigpower(A,tol,nmax,x0)
%EIGPOWER Approssima l'autovalore di modulo massimo di una matrice.
% LAMBDA = EIGPOWER(A) calcola con il metodo delle potenze l'autovalore
% di una matrice A di modulo massimo a partire da un dato iniziale pari al
% vettore unitario.
% LAMBDA = EIGPOWER(A,TOL,NMAX,X0) arresta il metodo quando la
% differenza fra due iterate consecutive e' minore di TOL (il valore di default
% e' 1.E-06) o quando il massimo numero di iterazioni NMAX (il valore di default
% e' 100) e' stato raggiunto.
% [LAMBDA,X,ITER] = EIGPOWER(A,TOL,NMAX,X0) restituisce anche
% l'autovettore unitario X tale che A*X=LAMBDA*X ed il numero di iterazioni
% effettuate per calcolare X.
[n,m] = size(A);
if n ~= m, error('Solo per matrici quadrate'); end
if nargin == 1
    tol = 1.e-06; x0 = ones(n,1); nmax = 100;
end
x0 = x0/norm(x0);
pro = A*x0;
lambda = x0'*pro;
err = tol*abs(lambda) + 1;
```

```

iter = 0;
while err > tol*abs(lambda) & abs(lambda) ~= 0 & iter <= nmax
    x = pro;           x = x/norm(x);
    pro = A*x;         lambdanew = x'*pro;
    err = abs(lambdanew - lambda);
    lambda = lambdanew; iter = iter + 1;
end
return

```

Esempio 6.1 Consideriamo la seguente famiglia di matrici

$$A(\alpha) = \begin{bmatrix} \alpha & 2 & 3 & 13 \\ 5 & 11 & 10 & 8 \\ 9 & 7 & 6 & 12 \\ 4 & 14 & 15 & 1 \end{bmatrix}, \quad \alpha \in \mathbb{R}.$$

Vogliamo approssimare l'autovalore di modulo massimo con il metodo delle potenze. Quando $\alpha = 30$, gli autovalori della matrice sono $\lambda_1 = 39.396$, $\lambda_2 = 17.8208$, $\lambda_3 = -9.5022$ e $\lambda_4 = 0.2854$ (abbiamo riportato solo le prime quattro cifre significative). Il metodo approssima λ_1 in 22 iterazioni con una tolleranza $\varepsilon = 10^{-10}$ ed avendo scelto $\mathbf{x}^{(0)} = \mathbf{1}^T$. Se invece $\alpha = -30$, ci vogliono ben 708 iterazioni. Il diverso comportamento può essere spiegato osservando che in quest'ultimo caso $\lambda_1 = -30.643$, $\lambda_2 = 29.7359$, $\lambda_3 = -11.6806$ e $\lambda_4 = 0.5878$. Di conseguenza, $|\lambda_2|/|\lambda_1| = 0.9704$, che è molto vicino all'unità.

Esempio 6.2 (Viabilità urbana) Indichiamo con $A \in \mathbb{R}^{11 \times 11}$ la matrice associata alla rete ferroviaria di Figura 6.2 generata ponendo $a_{ij} = 1$ se tra la città i e la città j esiste una connessione ferroviaria diretta. Prendendo `tol=1.e-12`, `x0=ones(11,1)`, dopo 26 iterazioni il Programma 13 restituisce la seguente approssimazione dell'autovettore di modulo unitario associato all'autovalore di modulo massimo di A

```

x =
0.5271
0.1590
0.2165
0.3580
0.4690
0.3861
0.1590
0.2837
0.0856
0.1906
0.0575

```

Si ricava che la città più facilmente raggiungibile è quella associata alla prima componente di \mathbf{x} (la più grande in modulo), dunque è Milano, mentre la peggiore è Mantova, associata all'ultima componente di \mathbf{x} (la più piccola in modulo).

Naturalmente questa analisi non tiene conto della frequenza delle connessioni (men che meno dei ritardi), ma solo dell'esistenza o meno di un collegamento fra le diverse città.

6.1.1 Analisi di convergenza

Essendo gli autovettori $\mathbf{x}_1, \dots, \mathbf{x}_n$ di A linearmente indipendenti, essi formano una base per \mathbb{C}^n . Di conseguenza, i vettori $\mathbf{x}^{(0)}$ e $\mathbf{y}^{(0)}$ possono essere scritti come

$$\mathbf{x}^{(0)} = \sum_{i=1}^n \alpha_i \mathbf{x}_i, \quad \mathbf{y}^{(0)} = \beta^{(0)} \sum_{i=1}^n \alpha_i \mathbf{x}_i \quad \text{con } \beta^{(0)} = 1/\|\mathbf{x}^{(0)}\| \text{ e } \alpha_i \in \mathbb{C}.$$

Al primo passo il metodo delle potenze fornisce

$$\mathbf{x}^{(1)} = A\mathbf{y}^{(0)} = \beta^{(0)} A \sum_{i=1}^n \alpha_i \mathbf{x}_i = \beta^{(0)} \sum_{i=1}^n \alpha_i \lambda_i \mathbf{x}_i$$

e, analogamente,

$$\mathbf{y}^{(1)} = \beta^{(1)} \sum_{i=1}^n \alpha_i \lambda_i \mathbf{x}_i, \quad \beta^{(1)} = \frac{1}{\|\mathbf{x}^{(0)}\| \|\mathbf{x}^{(1)}\|}.$$

Al generico passo k avremo dunque

$$\mathbf{y}^{(k)} = \beta^{(k)} \sum_{i=1}^n \alpha_i \lambda_i^k \mathbf{x}_i, \quad \beta^{(k)} = \frac{1}{\|\mathbf{x}^{(0)}\| \dots \|\mathbf{x}^{(k)}\|}$$

e quindi

$$\mathbf{y}^{(k)} = \lambda_1^k \beta^{(k)} \left(\alpha_1 \mathbf{x}_1 + \sum_{i=2}^n \alpha_i \frac{\lambda_i^k}{\lambda_1^k} \mathbf{x}_i \right).$$

Poiché $|\lambda_i/\lambda_1| < 1 \ \forall i = 2, \dots, n$, il vettore $\mathbf{y}^{(k)}$ tende ad allinearsi lungo la stessa direzione dell'autovettore \mathbf{x}_1 quando k tende a $+\infty$, purché $\alpha_1 \neq 0$. La condizione su α_1 , seppur teoricamente impossibile da assicurare essendo \mathbf{x}_1 una delle incognite del problema, non è di fatto restrittiva. L'insorgere degli errori di arrotondamento comporta infatti la comparsa di una componente nella direzione di \mathbf{x}_1 , anche se questa non era presente nel vettore iniziale $\mathbf{x}^{(0)}$ (questo è evidentemente uno dei rari casi in cui gli errori di arrotondamento giuocano a nostro favore).

Esempio 6.3 Consideriamo la matrice $A(\alpha)$, definita nell'Esempio 6.1, in corrispondenza di $\alpha = 16$. L'autovettore \mathbf{x}_1 di lunghezza unitaria associato all'autovalore λ_1 è $(1/2, 1/2, 1/2, 1/2)^T$. Scegliamo (di proposito!) il vettore iniziale $(2, -2, 3, -3)^T$ che è ortogonale a \mathbf{x}_1 .

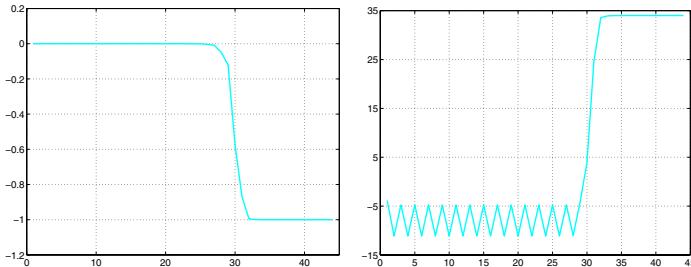


Figura 6.3. A sinistra $(\mathbf{y}^{(k)})^T \mathbf{x}_1 / (\|\mathbf{y}^{(k)}\| \|\mathbf{x}_1\|)$, a destra i valori $\lambda^{(k)}$, entrambi per $k = 1, \dots, 44$

In Figura 6.3 viene riportato $\cos(\theta^{(k)}) = (\mathbf{y}^{(k)})^T \mathbf{x}_1 / (\|\mathbf{y}^{(k)}\| \|\mathbf{x}_1\|)$. Come si nota dopo circa 30 iterazioni questo valore tende a -1 e l'angolo θ tra i due vettori tende rapidamente a π , mentre la successione dei $\lambda^{(k)}$ approssima $\lambda_1 = 34$. Il metodo delle potenze ha quindi generato, "grazie" agli errori di arrotondamento, una successione di vettori $\{\mathbf{y}^{(k)}\}$ con componente lungo la direzione di \mathbf{x}_1 crescente al crescere di k .

Osservazione 6.1 Si può dimostrare che il metodo delle potenze converge anche se λ_1 è una radice multipla di $p_A(\lambda)$. Non converge invece se esistono due autovalori distinti, aventi entrambi modulo massimo. In tal caso la successione dei $\lambda^{(k)}$ non tende ad alcun limite, ma oscilla fra i due autovalori.



Si vedano gli Esercizi 6.1-6.3.

6.2 Generalizzazione del metodo delle potenze

Una prima generalizzazione possibile per matrici non singolari consiste nell'applicare il metodo delle potenze alla matrice inversa: in tal caso infatti, essendo gli autovalori di A^{-1} i reciproci degli autovalori di A , il metodo potrà approssimare l'autovalore di A di modulo minimo. Si giunge così al *metodo delle potenze inverse*:

dato un vettore iniziale $\mathbf{x}^{(0)}$ e posto $\mathbf{y}^{(0)} = \mathbf{x}^{(0)} / \|\mathbf{x}^{(0)}\|$, per $k \geq 1$ si calcola

$$\mathbf{x}^{(k)} = A^{-1} \mathbf{y}^{(k-1)}, \quad \mathbf{y}^{(k)} = \frac{\mathbf{x}^{(k)}}{\|\mathbf{x}^{(k)}\|}, \quad \mu^{(k)} = (\mathbf{y}^{(k)})^* A^{-1} \mathbf{y}^{(k)} \quad (6.8)$$

Se A ammette n autovettori linearmente indipendenti, e se l'autovalore di modulo minimo λ_n è distinto dagli altri, allora

$$\lim_{k \rightarrow \infty} \mu^{(k)} = 1/\lambda_n$$

(i.e. $(\mu^{(k)})^{-1}$ tende a λ_n per $k \rightarrow \infty$).

Al k -esimo passo di questo metodo si deve risolvere un sistema lineare della forma $Ax^{(k)} = y^{(k-1)}$. È pertanto conveniente generare la fattorizzazione LU di A (o la sua fattorizzazione di Cholesky se A è simmetrica e definita positiva) una volta per tutte, all'inizio del processo, e risolvere ad ogni iterazione due sistemi triangolari.

Facciamo notare che il comando `lu` di MATLAB è in grado di calcolare la fattorizzazione LU anche di una matrice a coefficienti complessi.

Esempio 6.4 Applichiamo il metodo delle potenze inverse per il calcolo dell'autovalore di modulo minimo della matrice $A(30)$ definita nell'Esempio 6.1. Il metodo converge in 7 iterazioni al valore 3.5037 che approssima l'autovalore di modulo massimo di A^{-1} . Di conseguenza, l'approssimazione dell'autovalore di modulo minimo di $A(30)$ sarà $1/3.5037 \simeq 0.2854$.

Il metodo delle potenze può essere utilizzato anche per calcolare l'autovalore di A più vicino ad un dato numero $\mu \in \mathbb{C}$. Infatti, posto $A_\mu = A - \mu I$ osserviamo che $\lambda(A_\mu) = \lambda(A) - \mu$. Pertanto, per calcolare l'autovalore λ_μ di A più prossimo a μ , basta approssimare l'autovalore $\lambda_{min}(A_\mu)$ di modulo minimo di A_μ e quindi calcolare $\lambda_\mu = \lambda_{min}(A_\mu) + \mu$. Questo algoritmo è noto come il *metodo delle potenze inverse con shift* ed il numero μ è chiamato lo *shift*.

Nel Programma 14 viene implementato il metodo delle potenze inverse con *shift*; il metodo delle potenze inverse si ottiene ponendo semplicemente lo *shift* uguale a 0. I parametri d'ingresso `A`, `tol`, `nmax`, `x0` sono gli stessi del Programma 13, mentre `mu` denota il valore dello *shift*. I parametri in uscita sono l'approssimazione di λ_μ , l'autovettore associato `x` ed il numero di iterazioni effettuate per calcolarlo.

Programma 14 - `invshift` : il metodo delle potenze inverse con *shift*

```
function [lambda,x,iter]=invshift(A,mu,tol,nmax,x0)
%INVSIFT    Approssima l'autovalore di modulo minimo.
%  LAMBDA = INVSIFT(A) calcola con il metodo delle potenze l'autovalore
% di una matrice A di modulo minimo a partire da un dato iniziale pari al
% vettore unitario.
%  LAMBDA = INVSIFT(A,MU) calcola l'autovalore di A piu' vicino ad un dato
% numero (reale o complesso) MU.
%  LAMBDA = INVSIFT(A,MU,TOL,NMAX,X0) arresta il metodo
% quando la differenza fra due iterate consecutive e' minore di TOL
% (il valore di default e' 1.E-06) o quando il massimo numero di iterazioni
% NMAX (il valore di default e' 100) e' stato raggiunto.
%  [LAMBDA,X,ITER] = INVSIFT(A,MU,TOL,NMAX,X0) restituisce anche
% l'autovettore unitario X tale che A*X=LAMBDA*X ed il numero di iterazioni
% effettuate per calcolare X.
[n,m]=size(A);
if n ~= m, error('Solo per matrici quadrate'); end
```



```

if nargin == 1
    x0 = rand(n,1); nmax = 100; tol = 1.e-06; mu = 0;
elseif nargin == 2
    x0 = rand(n,1); nmax = 100; tol = 1.e-06;
end
[L,U]=lu(A-mu*eye(n));
if norm(x0) == 0
    x0 = rand(n,1);
end
x0=x0/norm(x0);
z0=L\x0;
pro=U\z0;
lambda=x0'*pro;
err=tol*abs(lambda)+1;      iter=0;
while err > tol*abs(lambda) & abs(lambda) ~= 0 & iter <= nmax
    x = pro; x = x/norm(x);
    z=L\x;   pro=U\z;
    lambdanew = x'*pro;
    err = abs(lambdanew - lambda);
    lambda = lambdanew;
    iter = iter + 1;
end
lambda = 1/lambda + mu;
return

```

Esempio 6.5 Cerchiamo fra tutti gli autovalori della matrice A(30) dell'Esempio 6.1 quello più vicino al numero reale 17. Utilizziamo il Programma 14 con `mu=17` ed una tolleranza `tol =10-10` e `x0=[1;1;1;1]`. Dopo 8 iterazioni, il programma ritorna il valore `lambda=17.82079703055703`.

Il valore dello *shift* potrebbe essere modificato nel corso delle iterazioni, ponendo $\mu = \lambda^{(k)}$. Ciò comporta una diminuzione del numero di iterazioni necessarie per soddisfare il test d'arresto, ma anche un considerevole aumento del costo computazionale in quanto la matrice A_μ cambia ad ogni passo e non sarà più possibile fattorizzarla una volta per tutte all'inizio del programma.



Si vedano gli Esercizi 6.4-6.6.



6.3 Come calcolare lo shift

Al fine di applicare il metodo delle potenze inverse con *shift* è necessario localizzare (in modo più o meno accurato) gli autovalori di A nel piano complesso. A tale scopo introduciamo la seguente definizione.

Sia A una matrice quadrata di dimensione n . I *cerchi di Gershgorin* $C_i^{(r)}$ e $C_i^{(c)}$ associati rispettivamente alla i -esima riga ed alla i -esima colonna di A sono definiti come

$$C_i^{(r)} = \{z \in \mathbb{C} : |z - a_{ii}| \leq \sum_{j=1, j \neq i}^n |a_{ij}|\},$$

$$C_i^{(c)} = \{z \in \mathbb{C} : |z - a_{ii}| \leq \sum_{j=1, j \neq i}^n |a_{ji}|\}.$$

$C_i^{(r)}$ è chiamato l' i -esimo *cerchio riga*, mentre $C_i^{(c)}$ l' i -esimo *cerchio colonna*.

Con il Programma 15 è possibile visualizzare in due finestre (aperte dal comando `figure`) sia i cerchi riga che quelli colonna. Il comando `hold on` consente di sovrapporre due o più grafici nella stessa figura (nel nostro caso i cerchi riga e colonna che vengono disegnati sequenzialmente) e viene neutralizzato dal comando `hold off`. I comandi `title`, `xlabel` e `ylabel` hanno lo scopo di visualizzare nella figura un titolo e le etichette degli assi.

Il comando `patch` serve per colorare i cerchi, mentre il comando `axis image` serve a scalare opportunamente l'immagine.

`figure`
`hold on`
`title`
`xlabel`
`ylabel`
`patch`
`axis`



Programma 15 - gershcircles : i cerchi di Gershgorin

```
function gershcircles(A)
%GERSHCIRCLES disegna i cerchi di Gershgorin
% GERSHGORINCIRCLES(A) disegna i cerchi di Gershgorin associati
% alla matrice A ed alla sua trasposta.
n = size(A);
if n(1) ~= n(2)
    error('Solo matrici quadrate');
else
    n = n(1); circler = zeros(n,201); circlec = circler;
end
center = diag(A);
radiic = sum(abs(A-diag(center))); radiir = sum(abs(A'-diag(center)));
one = ones(1,201); cosisin = exp(i*[0:pi/100:2*pi]);
figure(1); title('Cerchi riga'); xlabel('Re'); ylabel('Im');
figure(2); title('Cerchi colonna'); xlabel('Re'); ylabel('Im');
for k = 1:n
    circlec(k,:) = center(k)*one + radiic(k)*cosisin;
    circler(k,:) = center(k)*one + radiir(k)*cosisin;
    figure(1); patch(real(circler(k,:)),imag(circler(k,:)),'red'); hold on
    plot(real(circler(k,:)),imag(circler(k,:)),'k-',real(center(k)),imag(center(k)),'kx');
    figure(2); patch(real(circlec(k,:)),imag(circlec(k,:)),'green'); hold on
    plot(real(circlec(k,:)),imag(circlec(k,:)),'k-',real(center(k)),imag(center(k)),'kx');
end
for k = 1:n
    figure(1);
    plot(real(circler(k,:)),imag(circler(k,:)),'k-',real(center(k)),imag(center(k)),'kx');
    figure(2);
    plot(real(circlec(k,:)),imag(circlec(k,:)),'k-',real(center(k)),imag(center(k)),'kx');
```

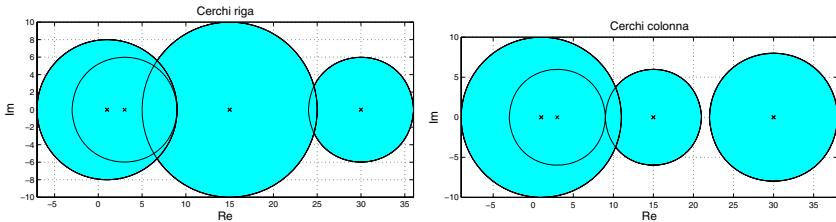


Figura 6.4. Cerchi riga (a sinistra) e colonna (a destra) per la matrice dell’Esempio 6.6

```
end
figure(1); axis image; hold off;
figure(2); axis image; hold off
return
```

Esempio 6.6 In Figura 6.4 sono stati riportati i cerchi di Gershgorin per la matrice

$$A = \begin{bmatrix} 30 & 1 & 2 & 3 \\ 4 & 15 & -4 & -2 \\ -1 & 0 & 3 & 5 \\ -3 & 5 & 0 & 1 \end{bmatrix}.$$

Con una croce sono stati evidenziati i centri dei cerchi.

Come già anticipato i cerchi di Gershgorin possono essere usati per localizzare gli autovalori di una matrice. Vale infatti il seguente risultato:

Proposizione 6.1 Tutti gli autovalori di una data matrice $A \in \mathbb{C}^{n \times n}$ appartengono alla regione del piano complesso individuata dall’intersezione tra l’unione dei cerchi riga e l’unione dei cerchi colonna.

Se inoltre m cerchi riga (o colonna) con $1 \leq m \leq n$, sono sconnessi dall’unione dei restanti $n - m$ cerchi riga (o colonna), allora la loro unione contiene esattamente m autovalori.

La Proposizione 6.1 non esclude che possano esistere cerchi vuoti (ovvero che non contengono alcun autovalore), purché non isolati. In generale inoltre l’informazione restituita dai cerchi di Gershgorin è abbastanza grossolana e garantisce solo un valore preliminare per lo *shift*.

Esempio 6.7 Dall’analisi dei cerchi riga della matrice $A(30)$ dell’Esempio 6.1 si può dedurre che la matrice ha autovalori con parte reale compresa fra -32 e 48 . Se utilizziamo il Programma 14 per calcolare l’autovalore di modulo massimo scegliendo come *shift* $\mu = 48$, troviamo che il metodo converge in 16 iterazioni, contro le 24 richieste dal metodo delle potenze a parità di dato iniziale $x_0 = [1; 1; 1; 1]$ e di tolleranza $\text{tol}=1.e-10$.



Riassumendo

- Il metodo delle potenze è una procedura iterativa che calcola l'autovalore di modulo massimo di una data matrice;
- il metodo delle potenze inverse consente il calcolo dell'autovalore di modulo minimo; per una efficace implementazione è opportuno fattorizzare la matrice data;
- il metodo delle potenze inverse con *shift* consente di calcolare l'autovalore più vicino ad un numero assegnato (lo *shift*); una sua efficiente applicazione richiede una qualche conoscenza *a priori* sulla localizzazione degli autovalori della matrice nel piano complesso; questo tipo di informazione può essere ottenuta esaminando i cerchi di Gershgorin.

Si vedano gli Esercizi 6.7-6.8.



6.4 Calcolo di tutti gli autovalori



Due matrici quadrate A e B della stessa dimensione sono dette *simili* se esiste una matrice P invertibile tale che

$$P^{-1}AP = B.$$

Matrici simili hanno gli stessi autovalori: infatti, se λ è un autovalore di A e $\mathbf{x} \neq \mathbf{0}$ è un autovettore associato, si ha

$$BP^{-1}\mathbf{x} = P^{-1}A\mathbf{x} = \lambda P^{-1}\mathbf{x},$$

cioè, λ è anche autovalore di B ed un suo autovettore associato è $\mathbf{y} = P^{-1}\mathbf{x}$.

I metodi che consentono di approssimare simultaneamente tutti gli autovalori di una matrice A sono generalmente basati sull'idea di trasformare A (dopo un numero infinito di passi) in una matrice simile ad A, ma con struttura diagonale o triangolare i cui autovalori sono quindi rappresentati dagli elementi diagonali.

Tra questi metodi ricordiamo il *metodo delle iterazioni QR*, che sta alla base della function MATLAB `eig`. Più precisamente, se richiamata nella forma `D=eig(A)`, dove A è una matrice quadrata, la function `eig` restituisce un vettore D contenente tutti gli autovalori di A. Se viene invece richiamata con due parametri di uscita, `[X,D]=eig(A)`, essa restituisce due matrici: D, la matrice diagonale i cui elementi sono gli autovalori di A, e X, la matrice i cui vettori colonna sono gli autovettori di A. Le matrici D e X soddisfano la relazione di similitudine `A*X=X*D`.

`eig`

Il metodo delle iterazioni QR è così chiamato perché impiega la fattorizzazione QR, introdotta nel paragrafo 5.5. In questo paragrafo presentiamo il metodo QR solo nel caso di matrici reali e nella sua forma più elementare che, come vedremo, non garantisce la convergenza agli autovalori di A . Per una descrizione completa dell'algoritmo rimandiamo a [QSS04, Capitolo 5], mentre per la sua estensione al caso complesso rimandiamo a [GL89, Sect.5.2.10] e a [Dem97, Sect.4.2.1].

L'idea di questo metodo consiste nel costruire una successione di matrici $A^{(k)}$, tutte simili alla matrice A di cui si vogliono calcolare gli autovalori. Posto $A^{(0)} = A$, si calcolano per $k = 0, 1, \dots$, con la fattorizzazione QR le matrici quadrate $Q^{(k+1)}$ e $R^{(k+1)}$ tali che

$$Q^{(k+1)}R^{(k+1)} = A^{(k)}$$

e si pone quindi $A^{(k+1)} = R^{(k+1)}Q^{(k+1)}$.

Si può dimostrare che le matrici $A^{(k)}$ sono tutte simili fra loro e, di conseguenza, essendo $A^{(0)} = A$, hanno in comune con quest'ultima gli autovalori (si veda l'Esercizio 6.9). Inoltre, se $A \in \mathbb{R}^{n \times n}$ ed i suoi autovalori soddisfano $|\lambda_1| > |\lambda_2| > \dots > |\lambda_n|$, allora si ha che

$$\lim_{k \rightarrow +\infty} A^{(k)} = T = \begin{bmatrix} \lambda_1 & t_{12} & \dots & t_{1n} \\ 0 & \ddots & \ddots & \vdots \\ \vdots & & \lambda_{n-1} & t_{n-1,n} \\ 0 & \dots & 0 & \lambda_n \end{bmatrix}. \quad (6.9)$$

Per quanto riguarda la velocità di convergenza a zero dei coefficienti $a_{i,j}^{(k)}$ con $i > j$ e k che tende all'infinito, si ha che essa dipende da $\max_i |\lambda_{i+1}/\lambda_i|$. Praticamente, si decide di arrestare il metodo quando $\max_{i>j} |a_{i,j}^{(k)}| \leq \epsilon$, dove $\epsilon > 0$ è una tolleranza fissata. Sotto l'ulteriore ipotesi che A sia simmetrica, la successione $\{A^{(k)}\}$ converge ad una matrice diagonale.

Nel Programma 16 viene proposta una implementazione del metodo delle iterazioni QR: i parametri d'ingresso sono la matrice A di cui si vogliono calcolare gli autovalori, la tolleranza tol ed il numero massimo di iterazioni consentito, nmax .



Programma 16 - basisQR : il metodo delle iterazioni QR

```
function D=qrbasis(A,tol,nmax)
%QRBASES calcola gli autovalori di una matrice
% D=QRBASES(A,TOL,NMAX) calcola con il metodo delle iterazioni QR tutti
% gli autovalori della matrice A a meno di una tolleranza TOL ed in un
% numero massimo di iterazioni NMAX. La convergenza di questo metodo non
% e' in generale garantita.
```

```
[n,m]=size(A);
if n ~= m, error('La matrice deve essere quadrata'); end
T = A; niter = 0; test = norm(tril(A,-1),inf);
while niter <= nmax & test >= tol
    [Q,R]=qr(T); T = R*Q;
    niter = niter + 1;
    test = norm(tril(T,-1),inf);
end
if niter > nmax
    warning('Il metodo non converge nel massimo numero di iterazioni permesso');
else
    fprintf('Il metodo converge in %i iterazioni\n',niter);
end
D = diag(T);
return
```

Esempio 6.8 Consideriamo la matrice $A(30)$ dell'Esempio 6.1 ed usiamo il Programma 16 per calcolarne gli autovalori. Otteniamo

```
>> D=qrbasis(A,1.e-14,100)
Il metodo e' andato a convergenza in 41 iterazioni
D =
39.3960
17.8208
-9.5022
0.2854
```

Questi autovalori sono in buon accordo con quelli riportati nell'Esempio 6.1 (ottenuti con il comando `eig`). Come abbiamo osservato in precedenza, la velocità di convergenza cala quando ci sono autovalori molto vicini in modulo: in effetti, per la matrice con $\alpha = -30$ che presenta due autovalori di modulo simile troviamo che il metodo per soddisfare lo stesso criterio d'arresto richiede ora ben 843 iterazioni

```
>> D=qrbasis(A,1.e-14,1000)
Il metodo e' andato a convergenza in 843 iterazioni
D =
-30.6430
29.7359
-11.6806
0.5878
```

Un caso particolare è quello in cui si vogliono approssimare gli autovalori di una matrice sparsa, presumibilmente di grande dimensione. In tal caso, se si memorizza la matrice in una variabile A nel formato sparso di MATLAB, con il comando `eigs(A,k)` si possono calcolare i primi k autovalori di modulo più grande della matrice stessa.

Affrontiamo infine brevemente il problema del calcolo dei valori singolari di una matrice rettangolare. MATLAB mette a disposizione due

`eigs`



Figura 6.5. A sinistra, l'immagine originale, al centro quella ricostruita sulla base dei primi 20 valori singolari e a destra quella ottenuta usando i primi 40

function: `svd` e `svds`. La prima calcola tutti i valori singolari di una matrice e la seconda soltano i k più grandi, con k da precisare in ingresso (per default $k=6$). Rimandiamo a [ABB⁺99] per una esaustiva presentazione dell'algoritmo che viene utilizzato.

`svd`
`svds`

Esempio 6.9 (Compressione di immagini) Con il comando MATLAB `A=imread('pout.tif')` carichiamo un'immagine in bianco e nero presente di default nell'Image Processing toolbox di MATLAB. La variabile `A` è una matrice di 291 per 240 interi a 8 bit (`uint8`) che rappresentano delle intensità di grigio. Eseguendo il comando `imshow(A)` ci apparirà l'immagine a sinistra di Figura 6.5. Per calcolare la SVD di `A` dobbiamo prima convertire `A` in una matrice di numeri in doppia precisione (i numeri *floating-point* che usa normalmente MATLAB): basta dare il comando `A=double(A)`. A questo punto il calcolo della SVD viene fatto ponendo `[U,S,V]=svd(A)`. Abbiamo riportato in Figura 6.5 al centro l'immagine ricostruita usando solo i primi 20 valori singolari di `S`, ottenuta dando cioè i comandi

```
>> X=U(:,1:20)*S(1:20,:)*V(:,1:20)'; imshow(uint8(X));
```

a destra di Figura 6.5 appare l'immagine ricostruita con i primi 40 valori singolari. Si osserva che l'immagine di destra è un'ottima approssimazione dell'immagine originale, e richiede la memorizzazione di 21280 coefficienti (due matrici di 291 per 40 e 240 per 40 coefficienti, più i 40 primi valori singolari) invece dei 69840 richiesti dall'immagine originale.



Riassumendo

1. Il metodo delle iterazioni QR è una tecnica iterativa globale che consente di approssimare tutti gli autovalori di una data matrice A ;
2. la convergenza del metodo nella sua versione di base è garantita se la matrice A ha coefficienti reali ed autovalori tutti distinti; la sua velocità di convergenza dipende asintoticamente dal modulo dei quozienti di due autovalori consecutivi.

Si vedano gli Esercizi 6.9-6.10.



6.5 Cosa non vi abbiamo detto

Non abbiamo mai accennato al condizionamento del problema del calcolo degli autovalori di una matrice, cioè alla sensibilità degli autovalori a variazioni degli elementi della matrice. Il lettore interessato può riferirsi a [Wil65], [GL89] e [QSS04], Capitolo 5.

Osserviamo soltanto che il calcolo degli autovalori di una matrice malcondizionata non è necessariamente un problema malcondizionato. Questo è, ad esempio, il caso della matrice di Hilbert (introdotta nell'Esempio 5.9) che ha un numero di condizionamento enorme, ma è ben condizionata per quanto concerne il calcolo degli autovalori essendo simmetrica e definita positiva.

Per il calcolo simultaneo di tutti gli autovalori, oltre al metodo QR segnaliamo il metodo di Jacobi che trasforma una matrice simmetrica in una matrice diagonale, eliminando, attraverso trasformazioni di similitudine, in modo sistematico tutti gli elementi extra-diagonali. Il metodo non termina dopo un numero finito di passi in quanto, azzerando uno specifico elemento extra-diagonale non si conservano necessariamente nulli gli elementi extra-diagonali già precedentemente azzerati.

Altri metodi sono il metodo di Lanczos ed il metodo delle successioni di Sturm. Per una panoramica di tutti questi metodi si veda [Saa92].

La libreria ARPACK (inclusa in MATLAB e disponibile con il comando `arpackc`) può essere infine usata per calcolare gli autovalori di una matrice di grandi dimensioni. La *function* MATLAB `eigs` è una subroutine di questa libreria.

`arpackc`

Menzioniamo infine che un uso appropriato di tecniche di *deflazione* (cioè l'eliminazione successiva di autovalori già calcolati) consente di accelerare la convergenza dei metodi e di ridurne il costo computazionale.

6.6 Esercizi

Esercizio 6.1 Si approssimi con il metodo delle potenze, con una tolleranza pari a $\varepsilon = 10^{-10}$, l'autovalore di modulo massimo per le seguenti matrici, a partire dal vettore $\mathbf{x}^{(0)} = (1, 2, 3)^T$

$$\mathbf{A}_1 = \begin{bmatrix} 1 & 2 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \quad \mathbf{A}_2 = \begin{bmatrix} 0.1 & 3.8 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \quad \mathbf{A}_3 = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}.$$

Si commenti la convergenza del metodo nei tre casi.

Esercizio 6.2 Le caratteristiche di una popolazione di pesci sono descritte nella seguente matrice di Leslie, introdotta nel Problema 6.3:

Intervallo d'età (mesi)	$\mathbf{x}^{(0)}$	m_i	s_i
0-3	6	0	0.2
3-6	12	0.5	0.4
6-9	8	0.8	0.8
9-12	4	0.3	-

Si calcoli il vettore \mathbf{x} della distribuzione normalizzata di questa popolazione per diversi intervalli d'età, in accordo con quanto mostrato nel Problema 6.3.

Esercizio 6.3 Si mostri che il metodo delle potenze non converge per una matrice che ha un autovalore di modulo massimo pari a $\lambda_1 = \gamma e^{i\vartheta}$ ed un altro $\lambda_2 = \gamma e^{-i\vartheta}$, dove $i = \sqrt{-1}$ e $\gamma, \vartheta \in \mathbb{R} \setminus \{0\}$.

Esercizio 6.4 Si mostri che se A è invertibile gli autovalori di A^{-1} sono i reciproci degli autovalori di A .

Esercizio 6.5 Si verifichi che per la seguente matrice il metodo delle potenze non è in grado di calcolare l'autovalore di modulo massimo e se ne fornisca una spiegazione

$$A = \begin{bmatrix} \frac{1}{3} & \frac{2}{3} & 2 & 3 \\ 1 & 0 & -1 & 2 \\ 0 & 0 & -\frac{5}{3} & -\frac{2}{3} \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

Esercizio 6.6 Usando il metodo delle potenze inverse con *shift*, si approssimi gli autovalori estremi della matrice di Wilkinson

$$A = \begin{bmatrix} 3 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 3 \end{bmatrix},$$

`wilkinson` ottenuta con il comando `wilkinson(7)`.

Esercizio 6.7 Utilizzando i cerchi di Gershgorin, si stimi il massimo numero di autovalori complessi delle matrici seguenti

$$A = \begin{bmatrix} 2 & -\frac{1}{2} & 0 & -\frac{1}{2} \\ 0 & 4 & 0 & 2 \\ -\frac{1}{2} & 0 & 6 & \frac{1}{2} \\ 0 & 0 & 1 & 9 \end{bmatrix}, B = \begin{bmatrix} -5 & 0 & 1/2 & 1/2 \\ 1/2 & 2 & 1/2 & 0 \\ 0 & 1 & 0 & 1/2 \\ 0 & 1/4 & 1/2 & 3 \end{bmatrix}.$$

Esercizio 6.8 Utilizzando il risultato della Proposizione 6.1 si trovi un opportuno *shift* per il calcolo dell'autovalore di modulo massimo di

$$A = \begin{bmatrix} 5 & 0 & 1 & -1 \\ 0 & 2 & 0 & -\frac{1}{2} \\ 0 & 1 & -1 & 1 \\ -1 & -1 & 0 & 0 \end{bmatrix}.$$

Si confrontino il numero di iterazioni ed il costo computazionale del metodo delle potenze e del metodo delle potenze inverse con *shift* richiedendo una tolleranza pari a 10^{-14} .

Esercizio 6.9 Si dimostri che le matrici $A^{(k)}$ generate dalle iterazioni QR sono tutte simili alla matrice A.

Esercizio 6.10 Si calcolino con il comando `eig` tutti gli autovalori delle matrici proposte nell'Esercizio 6.7 e si verifichi la bontà delle conclusioni tratte sulla base della Proposizione 6.1.

Equazioni differenziali ordinarie

Un'equazione differenziale è un'equazione che coinvolge una o più derivate di una funzione incognita. Se tutte le derivate sono fatte rispetto ad una sola variabile indipendente avremo un'*equazione differenziale ordinaria*, mentre avremo un'*equazione alle derivate parziali* quando sono presenti derivate rispetto a più variabili indipendenti.

L'equazione differenziale (ordinaria o alle derivate parziali) ha *ordine* p se p è l'ordine massimo delle derivate che vi compaiono. Dedicheremo il prossimo capitolo allo studio di equazioni alle derivate parziali di ordine 1 e 2, mentre in questo capitolo ci dedicheremo alle equazioni differenziali ordinarie di ordine 1.

Le equazioni differenziali ordinarie consentono di descrivere l'evoluzione di numerosi fenomeni nei campi più svariati. Vediamone alcuni esempi.

Problema 7.1 (Termodinamica) Consideriamo un corpo di temperatura interna T posto in un ambiente a temperatura costante T_e . Supponiamo che tutta la massa m del corpo sia concentrata in un punto. Allora il trasferimento di calore tra il corpo e l'ambiente esterno può essere descritto dalla legge di Stefan-Boltzmann

$$v(t) = \epsilon\gamma S(T^4(t) - T_e^4),$$

dove t è la variabile tempo, ϵ è la costante di Boltzmann (pari a $5.6 \cdot 10^{-8} \text{ J/m}^2 \text{K}^4 \text{s}$, dove J sta per Joule, K per Kelvin e, naturalmente, m per metri e s per secondi), γ è la costante di emissività del corpo, S l'area della superficie del corpo e v la velocità di trasferimento del calore. La velocità di variazione dell'energia $E(t) = mCT(t)$ (dove C è il calore specifico del materiale che costituisce il corpo) egualia, in valore assoluto, la velocità v di trasferimento del calore. Di conseguenza, ponendo $T(0) = T_0$, il calcolo di $T(t)$ richiede la risoluzione della seguente equazione differenziale ordinaria

$$\frac{dT}{dt} = -\frac{v(t)}{mC}. \quad (7.1)$$

Per la sua soluzione si veda l'Esercizio 7.15.

Problema 7.2 (Biologia) Consideriamo una popolazione di batteri posta in un ambiente limitato nel quale non possono convivere più di B batteri. Supponiamo che inizialmente la popolazione abbia un numero di individui pari a $y_0 \ll B$ e che il fattore di crescita dei batteri sia pari ad una costante positiva C . In tal caso, la velocità di cambiamento della popolazione di batteri nel tempo sarà proporzionale al numero di batteri preesistenti, con la restrizione che il numero complessivo di batteri sia minore di B . Ciò è esprimibile dall'equazione differenziale

$$\frac{dy}{dt} = Cy \left(1 - \frac{y}{B}\right), \quad (7.2)$$

la cui soluzione $y = y(t)$ esprime il numero di batteri presenti al tempo t .

Se ora supponiamo che due popolazioni batteriche, y_1 e y_2 , siano in competizione tra loro, all'equazione differenziale (7.2) si dovranno sostituire le equazioni seguenti

$$\begin{aligned} \frac{dy_1}{dt} &= C_1 y_1 (1 - b_1 y_1 - d_1 y_2), \\ \frac{dy_2}{dt} &= -C_2 y_2 (1 - b_2 y_2 - d_2 y_1), \end{aligned} \quad (7.3)$$

dove C_1 e C_2 sono i fattori di crescita (positivi) delle due popolazioni batteriche. I coefficienti d_1 e d_2 governano il tipo di interazione tra le due popolazioni, mentre b_1 e b_2 sono legati alla disponibilità dei nutrienti. Le equazioni (7.3) sono note come equazioni di Lotka-Volterra e sono alla base di numerose applicazioni. Per una loro soluzione si veda l'Esempio 7.7.

Problema 7.3 (Sport) Si vuole simulare la traiettoria di una palla da baseball dal lanciatore al battitore; la distanza fra i due è di circa 18.44 m. Se si adotta un sistema di coordinate come quello indicato in Figura 7.1 le equazioni del moto della palla sono date dal seguente sistema di equazioni differenziali ordinarie (si vedano [Ada90] e [Gio97])

$$\begin{aligned} \frac{d\mathbf{x}}{dt} &= \mathbf{v}, \\ \frac{d\mathbf{v}}{dt} &= \mathbf{F}, \end{aligned}$$

dove $\mathbf{x} = (x, y, z)^T$ è la posizione della palla, $\mathbf{v} = (v_x, v_y, v_z)^T$ la sua velocità di modulo v , \mathbf{F} è il vettore di componenti

$$\begin{aligned} F_x &= -F(v)vv_x + B\omega(v_z \sin \phi - v_y \cos \phi), \\ F_y &= -F(v)vv_y + B\omega v_x \cos \phi, \\ F_z &= -g - F(v)vv_z - B\omega v_x \sin \phi, \end{aligned} \quad (7.4)$$

$B = 4.1 \cdot 10^{-4}$, ϕ è l'angolo di lancio, mentre ω è il modulo della velocità angolare impressa alla palla dal lanciatore. Si tenga conto che per una normale palla da baseball il coefficiente $F(v)$ che compare nella (7.4) e che tiene conto dell'effetto d'attrito dell'aria, è pari a

$$F(v) = 0.0039 + \frac{0.0058}{1 + e^{(v-35)/5}}.$$

Per la risoluzione numerica di questo problema si veda l'Esercizio 7.20.

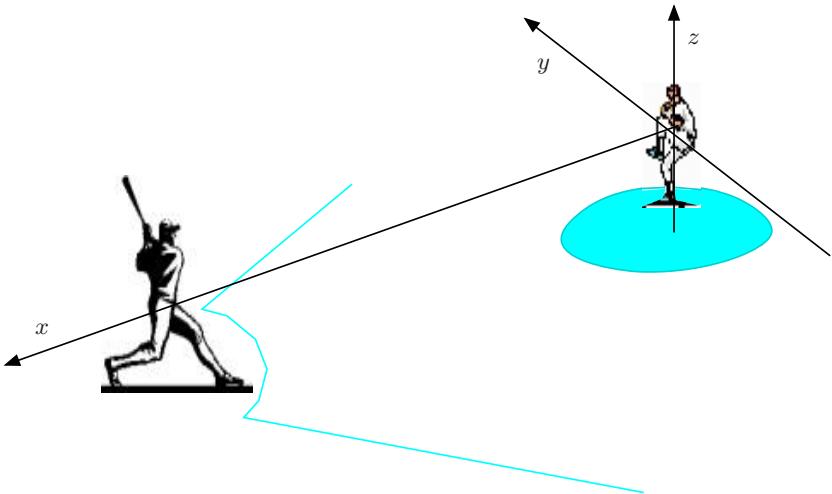


Figura 7.1. Il sistema di riferimento adottato per il Problema 7.3

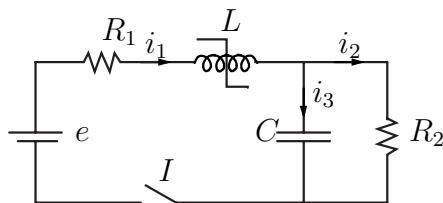


Figura 7.2. Il circuito elettrico del Problema 7.4

Problema 7.4 (Elettrotecnica) Consideriamo il circuito elettrico indicato in Figura 7.2. Si vuole studiare l'andamento della differenza di potenziale $v(t)$ ai capi del condensatore C a partire dal tempo $t = 0$ in cui viene chiuso l'interruttore I . Supponiamo che l'induttanza L possa essere espressa come una funzione esplicita dell'intensità di corrente i , cioè che $L = L(i)$. Per la legge di Ohm si ha

$$e - \frac{d(i_1 L(i_1))}{dt} = i_1 R_1 + v,$$

dove R_1 è una resistenza. Assumendo le correnti dirette come in Figura 7.2, derivando rispetto a t entrambi i membri della legge di Kirchoff $i_1 = i_2 + i_3$ ed osservando che $i_3 = Cdv/dt$ e $i_2 = v/R_2$, si trova l'ulteriore equazione

$$\frac{di_1}{dt} = C \frac{d^2v}{dt^2} + \frac{1}{R_2} \frac{dv}{dt}.$$

Abbiamo dunque trovato un sistema di due equazioni differenziali la cui soluzione consente di descrivere l'andamento delle incognite v e i_1 al variare del tempo. Come si vede la seconda di queste equazioni è di ordine 2. Per una sua soluzione si veda l'Esempio 7.8.

7.1 Il problema di Cauchy

Ci limitiamo al caso di un'equazione differenziale ordinaria del prim'ordine. Un'equazione differenziale di ordine $p > 1$ può sempre essere ridotta ad un sistema di p equazioni del prim'ordine. Il caso dei sistemi verrà affrontato nel paragrafo 7.8.

Un'equazione differenziale ordinaria ammette in generale infinite soluzioni. Per fissarne una è necessario imporre una condizione che prescriva il valore assunto dalla soluzione in un punto dell'intervallo di integrazione. Ad esempio, l'equazione (7.2) ammette la seguente famiglia di soluzioni $y(t) = B\Psi(t)/(1 + \Psi(t))$ con $\Psi(t) = e^{Ct+K}$, essendo K una costante arbitraria. Se imponiamo la condizione $y(0) = 1$, selezioniamo l'unica soluzione corrispondente al valore $K = \ln[1/(B - 1)]$.

Ci occuperemo della risoluzione dei cosiddetti *problemi di Cauchy*, ossia di problemi della forma:

trovare $y : I \rightarrow \mathbb{R}$ tale che

$$\begin{cases} y'(t) = f(t, y(t)) \quad \forall t \in I, \\ y(t_0) = y_0, \end{cases} \quad (7.5)$$

dove I è un intervallo di \mathbb{R} , $f : I \times \mathbb{R} \rightarrow \mathbb{R}$ una funzione assegnata e y' indica la derivata di y rispetto a t . Infine, t_0 è un punto di I e y_0 è un valore assegnato detto *dato iniziale*.

Nella seguente proposizione riportiamo un risultato classico dell'Analisi Matematica per tali problemi:

Proposizione 7.1 *Supponiamo che la funzione $f(t, y)$ sia*

1. continua rispetto ad entrambi gli argomenti;
2. lipschitziana rispetto al secondo argomento, ossia esista una costante L positiva tale che

$$|f(t, y_1) - f(t, y_2)| \leq L|y_1 - y_2|, \quad \forall t \in I, \quad \forall y_1, y_2 \in \mathbb{R}.$$

Allora la soluzione del problema di Cauchy (7.5) esiste, è unica ed è di classe C^1 su I .

Sfortunatamente solo un limitato numero di equazioni differenziali ordinarie ammette soluzione in forma esplicita. In molti altri casi, la soluzione è disponibile solo implicitamente. Questo è ad esempio il caso dell'equazione $y'(t) = (y-t)/(y+t)$ le cui soluzioni verificano la relazione implicita

$$\frac{1}{2} \ln(t^2 + y^2) + \operatorname{arctg} \frac{y}{t} = C,$$

dove C è una costante arbitraria. In certe situazioni addirittura la soluzione non è rappresentabile nemmeno in forma implicita. È questo ad esempio il caso dell'equazione $y' = e^{-t^2}$ la cui soluzione è esprimibile solo tramite uno sviluppo in serie.

Cerchiamo quindi dei metodi numerici in grado di approssimare la soluzione di *ogni* classe di equazioni differenziali ordinarie che ammettano una soluzione.

La strategia generale di tali metodi consiste nel dividere l'intervallo di integrazione $I = [t_0, T]$, con $T < +\infty$, in N_h sottointervalli di ampiezza $h = (T - t_0)/N_h$; h è detto il *passo di discretizzazione*. Indi, si cerca per ogni nodo $t_n = t_0 + nh$ ($1 \leq n \leq N_h$) il valore incognito u_n che approssimi $y_n = y(t_n)$. L'insieme dei valori $\{u_0, u_1, \dots, u_{N_h}\}$ forma la *soluzione numerica*.

7.2 I metodi di Eulero

Un metodo classico, il cosiddetto *metodo di Eulero in avanti*, genera la successione seguente

$$u_{n+1} = u_n + hf_n, \quad n = 0, \dots, N_h - 1 \quad (7.6)$$

avendo usato la notazione semplificata $f_n = f(t_n, u_n)$. Questo metodo è derivato dall'equazione differenziale (7.5) considerata in ogni nodo t_n con $n = 1, \dots, N_h$, qualora si approssimi la derivata esatta $y'(t_n)$ con il rapporto incrementale (4.4).

Procedendo in maniera analoga, ma utilizzando questa volta il rapporto incrementale (4.8) per approssimare $y'(t_{n+1})$, si ottiene il *metodo di Eulero all'indietro*

$$u_{n+1} = u_n + hf_{n+1}, \quad n = 0, \dots, N_h - 1 \quad (7.7)$$

Si tratta di due esempi di metodi *ad un passo* in quanto per calcolare la soluzione numerica nel nodo t_{n+1} necessitano solo delle informazioni legate al nodo precedente t_n .

Più precisamente, mentre nel metodo di Eulero in avanti la soluzione numerica u_{n+1} dipende *esclusivamente* dal valore precedentemente calcolato u_n , nel metodo di Eulero all'indietro dipende, tramite f_{n+1} , anche da se stessa. Per tale motivo, il primo metodo è detto *esplicito* ed il secondo *implicito* (e per questo essi sono noti anche con i nomi di Eulero esplicito e di Eulero隐式, rispettivamente).

Ad esempio, la discretizzazione di (7.2) con il metodo di Eulero in avanti richiede ad ogni passo il calcolo di

$$u_{n+1} = u_n + hCu_n(1 - u_n/B),$$

mentre se si usasse il metodo di Eulero all'indietro si dovrebbe risolvere l'equazione non lineare

$$u_{n+1} = u_n + hCu_{n+1} (1 - u_{n+1}/B).$$

Di conseguenza, i metodi impliciti sono assai più costosi di quelli esplicativi, in quanto se la funzione f del problema (7.5) è non lineare in y , ad ogni livello temporale t_{n+1} essi richiedono la soluzione di un problema non lineare per calcolare u_{n+1} . D'altra parte, vedremo che i metodi impliciti godono di miglior caratteristiche di stabilità degli schemi esplicativi.

Il metodo di Eulero in avanti è implementato nel Programma 17; l'intervallo di integrazione è `tspan = [t0,tfinal]`, `odefun` è una *function* o una *inline function* che precisa la funzione $f(t,y(t))$ che dipende dalle variabili `t` e `y` (e da eventuali altri parametri opzionali) ed i cui primi due argomenti si riferiscono a `t` e `y`.



Programma 17 - feuler : il metodo di Eulero in avanti

```
function [t,y]=feuler(odefun,tspan,y,Nh,varargin)
%FEULER Risolve equazioni differenziali usando il metodo di Eulero in avanti.
% [T,Y] = FEULER(ODEFUN,TSPAN,Y0,NH) con TSPAN = [T0 TFINAL]
% integra il sistema di equazioni differenziali y' = f(t,y) dal tempo T0 a
% TFINAL con condizione iniziale Y0 usando il metodo di Eulero in avanti su
% una griglia di NH intervalli equispaziati.
% La funzione ODEFUN(T,Y) deve ritornare un vettore colonna corrispondente
% a f(t,y). Ogni riga dell'array soluzione Y corrisponde alla soluzione calcolata
% per il corrispondente vettore colonna T.
% [T,Y] = FEULER(ODEFUN,TSPAN,Y0,NH,P1,P2,...) passa i parametri addizionali
% P1,P2,... alla funzione ODEFUN come ODEFUN(T,Y,P1,P2...).
h=(tspan(2)-tspan(1))/Nh;
tt=linspace(tspan(1),tspan(2),Nh+1);
for t = tt(1:end-1)
    y = [y; y(end,:)+h*feval(odefun,t,y(end,:),varargin{:})];
end
t=tt;
return
```

Il metodo di Eulero all'indietro è implementato nel Programma 18. Si noti che abbiamo utilizzato la funzione `fzero` per la soluzione del problema non lineare che appare ad ogni passo. Come dato iniziale per `fzero` utilizziamo l'ultimo valore disponibile per la soluzione approssimata.



Programma 18 - beuler : il metodo di Eulero all'indietro

```
function [t,u]=beuler(odefun,tspan,y,Nh,varargin)
%BEULER Risolve equazioni differenziali usando il metodo di Eulero all'indietro.
% [T,Y] = BEULER(ODEFUN,TSPAN,Y0,NH) con TSPAN = [T0 TFINAL]
```

```
% integra il sistema di equazioni differenziali y' = f(t,y) dal tempo T0 a
% TFINAL con condizione iniziale Y0 usando il metodo di Eulero all'indietro su
% una griglia di NH intervalli equispaziati.
% La funzione ODEFUN(T,Y) deve ritornare un vettore colonna corrispondente
% a f(t,y). Ogni riga dell'array soluzione Y corrisponde alla soluzione calcolata
% per il corrispondente vettore colonna T.
% [T,Y] = BEULER(ODEFUN,TSPAN,Y0,NH,P1,P2,...) passa i parametri addizionali
% P1,P2,... alla funzione ODEFUN come ODEFUN(T,Y,P1,P2...).
h=(tspan(2)-tspan(1))/Nh;
tt=linspace(tspan(1),tspan(2),Nh+1);
u(1,:)=y;

fid=fopen('myfun.m','w');
fprintf(fid,'function z=myfun(w,t,y,h,odefun)\n');
fprintf(fid,'z=w-y-h*feval(odefun,t,w);\n');
fprintf(fid,'return\n');
fclose(fid);
options=optimset;
options.TolFun=1.e-06;
options.MaxFunEvals=10000;
for t=tt(2:end)
    w = fsolve(@(w) myfun(w,t,y,h,odefun),y,options);
    u = [u; w];
    y = w;
end
t=tt;
return
```

7.2.1 Analisi di convergenza

Un metodo numerico si dice *convergente* se

$$\forall n = 0, \dots, N_h, \quad |y_n - u_n| \leq C(h) \quad (7.8)$$

dove $C(h)$ è un infinitesimo rispetto a h per h che tende a 0. Se $C(h) = \mathcal{O}(h^p)$ per qualche $p > 0$, diremo che il metodo converge con *ordine p*. Per verificare che il metodo di Eulero in avanti è convergente, scriviamo l'errore nel seguente modo

$$e_n = y_n - u_n = (y_n - u_n^*) + (u_n^* - u_n), \quad (7.9)$$

dove

$$u_n^* = y_{n-1} + h f(t_{n-1}, y_{n-1})$$

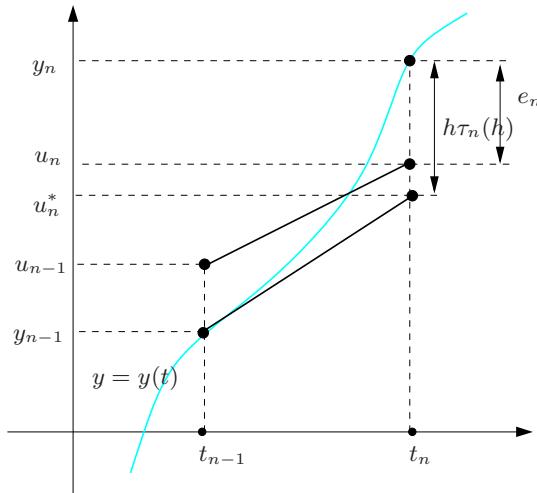


Figura 7.3. Rappresentazione geometrica di un passo del metodo di Eulero in avanti

denota la soluzione numerica calcolata in t_n a partire dalla soluzione esatta al tempo t_{n-1} ; si veda la Figura 7.3. Il termine $y_n - u_n^*$ nella (7.9) rappresenta l'errore prodotto da un passo del metodo di Eulero in avanti, mentre il termine $u_n^* - u_n$ rappresenta la propagazione da t_{n-1} a t_n dell'errore accumulato al livello temporale precedente. Il metodo converge se entrambi i termini tendono a 0 quando $h \rightarrow 0$. Supponendo che la derivata seconda di y esista e sia continua, grazie alla (4.6), si trova

$$y_n - u_n^* = \frac{h^2}{2} y''(\xi_n), \text{ per un qualche } \xi_n \in (t_{n-1}, t_n). \quad (7.10)$$

La quantità

$$\tau_n(h) = (y_n - u_n^*)/h$$

è chiamata *errore di troncamento locale* per il metodo di Eulero. Più in generale, l'errore di troncamento locale rappresenta l'errore che si sarebbe generato forzando la soluzione esatta a soddisfare lo schema numerico, mentre l'*errore di troncamento globale* è definito come

$$\tau(h) = \max_{n=0,\dots,N_h} |\tau_n(h)|.$$

Alla luce della (7.10) si deduce che per il metodo di Eulero in avanti l'errore di troncamento globale assume la forma seguente

$$\tau(h) = Mh/2, \quad (7.11)$$

dove $M = \max_{t \in [t_0, T]} |y''(t)|$.

Dalla (7.10) si deduce inoltre che $\lim_{h \rightarrow 0} \tau_n(h) = 0$. In generale, un metodo per il quale l'errore di troncamento locale tende a 0 per h che tende a 0 verrà detto *consistente*. Diremo inoltre che è consistente con ordine p se $\tau(h) = \mathcal{O}(h^p)$ per un opportuno intero $p \geq 1$.

Consideriamo ora il secondo addendo della (7.9). Abbiamo

$$u_n^* - u_n = e_{n-1} + h [f(t_{n-1}, y_{n-1}) - f(t_{n-1}, u_{n-1})]. \quad (7.12)$$

Di conseguenza, essendo f lipschitziana rispetto al suo secondo argomento, si trova

$$|u_n^* - u_n| \leq (1 + hL)|e_{n-1}|.$$

Se $e_0 = 0$, la relazione precedente diventa

$$\begin{aligned} |e_n| &\leq |y_n - u_n^*| + |u_n^* - u_n| \\ &\leq h|\tau_n(h)| + (1 + hL)|e_{n-1}| \\ &\leq [1 + (1 + hL) + \dots + (1 + hL)^{n-1}] h\tau(h) \\ &= \frac{(1 + hL)^n - 1}{L} \tau(h) \leq \frac{e^{L(t_n - t_0)} - 1}{L} \tau(h), \end{aligned}$$

dove abbiamo usato l'identità

$$\sum_{k=0}^{n-1} (1 + hL)^k = [(1 + hL)^n - 1]/hL,$$

la diseguaglianza $1 + hL \leq e^{hL}$ ed abbiamo osservato che $nh = t_n - t_0$. Troviamo quindi

$$|e_n| \leq \frac{e^{L(t_n - t_0)} - 1}{L} \frac{M}{2} h, \forall n = 0, \dots, N_h, \quad (7.13)$$

pertanto *il metodo di Eulero in avanti converge con ordine 1*. Come si nota l'ordine del metodo è uguale all'ordine dell'errore di troncamento locale: questa è una proprietà comune a molti schemi per la risoluzione delle equazioni differenziali ordinarie.

La stima di convergenza (7.13) è stata ottenuta richiedendo semplicemente che f sia continua e Lipschitziana. Una stima migliore, precisamente

$$|e_n| \leq Mh(t_n - t_0)/2 \quad (7.14)$$

vale se $\partial f(t, y)/\partial y$ esiste ed è ≤ 0 per ogni $t \in [t_0, T]$ e per ogni $-\infty < y < \infty$. Infatti, in tal caso, usando lo sviluppo in serie di Taylor, dalla (7.12) troviamo

$$u_n^* - u_n = (1 + h \partial f / \partial y(t_{n-1}, \eta_n)) e_{n-1},$$

essendo η_n un punto appartenente all'intervallo di estremi y_{n-1} e u_{n-1} , per cui $|u_n^* - u_n| \leq |e_{n-1}|$, purché valga la seguente restrizione (di stabilità, come vedremo nel seguito)

$$h < 2 / \max_{t \in [t_0, T]} |\partial f / \partial y(t, y(t))|. \quad (7.15)$$

Di conseguenza $|e_n| \leq |y_n - u_n^*| + |e_{n-1}| \leq nh\tau(h) + |e_0|$ e quindi la (7.14) grazie alla (7.11) ed al fatto che $e_0 = 0$.

Osservazione 7.1 (Consistenza) La proprietà di consistenza è necessaria per poter avere la convergenza. Se infatti essa non fosse soddisfatta, il metodo introdurrebbe ad ogni passo un errore non infinitesimo rispetto a h che, sommandosi con gli errori pregressi, pregiudicherebbe in modo irrimediabile la possibilità che l'errore globale tenda a 0 quando $h \rightarrow 0$.

Per il metodo di Eulero all'indietro l'errore di troncamento locale vale

$$\tau_n(h) = \frac{1}{h} [y_n - y_{n-1} - hf(t_n, y_n)].$$

Usando nuovamente lo sviluppo in serie di Taylor si trova

$$\tau_n(h) = -\frac{h^2}{2} y''(\xi_n)$$

per un opportuno $\xi_n \in (t_{n-1}, t_n)$, purché $y \in C^2$. Di conseguenza, anche il metodo di Eulero all'indietro converge con ordine 1 rispetto a h .

Esempio 7.1 Consideriamo il problema di Cauchy

$$\begin{cases} y'(t) = \cos(2y(t)) & t \in (0, 1], \\ y(0) = 0, \end{cases} \quad (7.16)$$

la cui soluzione è $y(t) = \frac{1}{2} \arcsin((e^{4t} - 1)/(e^{4t} + 1))$. Risolviamolo con il metodo di Eulero in avanti (Programma 17) e con il metodo di Eulero all'indietro (Programma 18). Tramite i comandi che seguono usiamo diversi valori di h : $1/2, 1/4, 1/8, \dots, 1/512$:

```
>> tspan=[0,1]; y0=0; f=inline('cos(2*y)', 't', 'y');
>> u=inline('0.5*asin((exp(4*t)-1)./(exp(4*t)+1))', 't');
>> Nh=2;
for k=1:10
    [t,ufe]=feuler(f,tspan,y0,Nh); fe(k)=abs(ufe(end)-feval(u,t(end)));
    [t,ube]=beuler(f,tspan,y0,Nh); be(k)=abs(ube(end)-feval(u,t(end)));
    Nh = 2*Nh;
end
```

Gli errori valutati per $t = 1$ sono memorizzati nelle variabili **fe** (per Eulero in avanti) e **be** (per Eulero all'indietro). Per stimare l'ordine di convergenza usiamo la formula (1.11). Tramite i comandi seguenti

```
>> p=log(abs(fe(1:end-1)./fe(2:end)))/log(2); p(1:2:end)
    1.2898  1.0349  1.0080  1.0019  1.0005
>> p=log(abs(be(1:end-1)./be(2:end)))/log(2); p(1:2:end)
    0.9070  0.9720  0.9959  0.9898  0.9975
```

possiamo verificare che entrambi i metodi convergono con ordine 1.

Osservazione 7.2 (Effetto degli errori di arrotondamento) La stima dell'errore (7.13) è stata derivata supponendo che la soluzione numerica $\{u_n\}$ sia stata calcolata in aritmetica esatta. Se si dovesse tener conto degli (inevitabili) errore di arrotondamento, l'errore esploderebbe quando h tende a 0 come $\mathcal{O}(1/h)$ (si veda, ad esempio, [Atk89]). Questa osservazione suggerisce che non è ragionevole prendere h al di sotto di un valore di soglia h^* (che è generalmente piccolissimo) nei calcoli.

Si vedano gli Esercizi 7.1-7.3.



7.3 Il metodo di Crank-Nicolson

Sommando membro a membro il generico passo dei metodi di Eulero in avanti e di Eulero all'indietro si ottiene un altro metodo ad un passo隐式的, il cosiddetto *metodo di Crank-Nicolson*

$$u_{n+1} = u_n + \frac{h}{2}[f_n + f_{n+1}], \quad n = 0, \dots, N_h - 1 \quad (7.17)$$

Esso può essere anche derivato applicando il teorema fondamentale del calcolo integrale (richiamato nel paragrafo 1.4.3) al problema di Cauchy (7.5), ottenendo

$$y_{n+1} = y_n + \int_{t_n}^{t_{n+1}} f(t, y(t)) dt, \quad (7.18)$$

per poi approssimare l'integrale su $[t_n, t_{n+1}]$ con la formula del trapezio (4.19).

L'errore di troncamento locale del metodo di Crank-Nicolson è dato da:

$$\begin{aligned} \tau_n(h) &= \frac{1}{h}[y(t_n) - y(t_{n-1})] - \frac{1}{2}[f(t_n, y(t_n)) + f(t_{n-1}, y(t_{n-1}))] \\ &= \frac{1}{h} \int_{t_{n-1}}^{t_n} f(t, y(t)) dt - \frac{1}{2}[f(t_n, y(t_n)) + f(t_{n-1}, y(t_{n-1}))]. \end{aligned}$$

L'ultima uguaglianza segue dalla (7.18) ed esprime, a meno di un fattore h , l'errore associato all'uso della formula del trapezio (4.19) per l'integrazione numerica. Se supponiamo che $y \in C^3$, dalla (4.20) si ricava che

$$\tau_n(h) = -\frac{h^2}{12}y'''(\xi_n) \text{ per un opportuno } \xi_n \in (t_{n-1}, t_n). \quad (7.19)$$

Il metodo di Crank-Nicolson è dunque consistente con ordine 2, cioè presenta un errore di troncamento locale che tende a 0 come h^2 . Con calcoli analoghi a quelli mostrati per il metodo di Eulero in avanti, si verifica inoltre che è anche convergente con ordine 2 rispetto a h .

Il metodo di Crank-Nicolson è implementato nel Programma 19. I parametri di ingresso e di uscita in questo programma sono gli stessi di quelli impiegati per i metodi di Eulero.



Programma 19 - cranknic : il metodo di Crank-Nicolson

```
function [t,u]=cranknic(odefun,tspan,y,Nh,varargin)
%CRANKNIC Risolve equazioni differenziali usando il metodo di Crank-Nicolson.
% [T,Y] = CRANKNIC(ODEFUN,TSPAN,Y0,NH) with TSPAN = [T0 TFINAL]
% integra il sistema di equazioni differenziali y' = f(t,y) dal tempo T0 a
% TFINAL con condizione iniziale Y0 usando il metodo di Crank-Nicolson su una
% griglia di NH intervalli equispaziati.
% La funzione ODEFUN(T,Y) deve ritornare un vettore colonna corrispondente
% a f(t,y). Ogni riga dell'array soluzione Y corrisponde alla soluzione calcolata
% per il corrispondente vettore colonna T.
% [T,Y] = CRANKNIC(ODEFUN,TSPAN,Y0,NH,P1,P2,...) passa i parametri
% addizionali P1,P2,... alla funzione ODEFUN come ODEFUN(T,Y,P1,P2...).
h=(tspan(2)-tspan(1))/Nh;
tt=linspace(tspan(1),tspan(2),Nh+1);
u(:,1)=y;

fid=fopen('myfun.m','w');
fprintf(fid,'function z=myfun(w,t,y,h,odefun)\n');
fprintf(fid,'z=w-y-0.5*h*(feval(odefun,t,w)+feval(odefun,t,y));\n');
fprintf(fid,'return\n');
fclose(fid);
options=optimset;
options.TolFun=1.e-06;
options.MaxFunEvals=10000;
for t=tt(2:end)
    w = fsolve(@(w) myfun(w,t,y,h,odefun),y,options);
    u = [u; w];
    y = w;
end
t=tt;
return
```

Esempio 7.2 Risolviamo con il metodo di Crank-Nicolson il problema (7.16) con gli stessi valori di h usati nell'Esempio 7.1. Come si vede, i risultati confermano che l'errore tende a zero con ordine 2

```
>> y0=0; tspan=[0 1]; N=2; f=inline('cos(2*y)','t','y');
>> y='0.5*asin((exp(4*t)-1)./(exp(4*t)+1))';
>> for k=1:10
    [tt,u]=cranknic(f,tspan,y0,N);
    t=tt(end); e(k)=abs(u(end)-eval(y)); N=2*N; end
>> p=log(abs(e(1:end-1)./e(2:end)))/log(2); p(1:2:end)
1.7940 1.9944 1.9997 2.0000 2.0000
```

7.4 Zero-stabilità

In generale, intendiamo per stabilità di un metodo numerico la possibilità di controllare l'effetto sulla soluzione di eventuali perturbazioni sui dati. Tra i possibili tipi di stabilità che si possono considerare per la risoluzione numerica di un problema di Cauchy, ve ne è uno, la cosiddetta zero-stabilità, che garantisce, se soddisfatta, che, in un intervallo limitato e fissato, piccole perturbazioni sui dati producano perturbazioni limitate sulla soluzione quando $h \rightarrow 0$.

Precisamente, un metodo numerico per l'approssimazione del problema (7.5), con $I = [t_0, T]$, è detto *zero-stabile* se esiste $C > 0$ tale che per ogni $\delta > 0$ e per ogni h sufficientemente piccolo, cioè $0 < h \leq h_0$ per un opportuno $h_0 > 0$, vale

$$|z_n - u_n| \leq C\delta, \quad 0 \leq n \leq N_h, \quad (7.20)$$

dove C è una costante che può dipendere dalla lunghezza dell'intervallo di integrazione I , z_n è la soluzione che si otterebbe applicando il metodo numerico al problema *perturbato* e δ indica la massima grandezza della perturbazione. Naturalmente, δ deve essere sufficientemente piccolo da garantire che il problema perturbato ammetta comunque un'unica soluzione sull'intervallo di integrazione.

Ad esempio, nel caso del metodo di Eulero in avanti, u_n soddisfa il problema

$$\begin{cases} u_{n+1} = u_n + hf(t_n, u_n), \\ u_0 = y_0, \end{cases} \quad (7.21)$$

mentre z_n soddisfa il problema perturbato

$$\begin{cases} z_{n+1} = z_n + h [f(t_n, z_n) + \rho_{n+1}], \\ z_0 = y_0 + \rho_0, \end{cases} \quad (7.22)$$

per $0 \leq n \leq N_h - 1$, supponendo $|\rho_n| \leq \delta$, $0 \leq n \leq N_h$.

Per un metodo ad un passo consistente si può dimostrare che la zero-stabilità è una conseguenza del fatto che f è continua e Lipschitziana rispetto al suo secondo argomento (si veda, ad esempio, [QSS04]). In tal caso la costante C nella (7.20) dipende da $\exp((T - t_0)L)$, dove L è la costante di Lipschitz.

D'altra parte la Lipschitzianità di f può non essere sufficiente per altre famiglie di metodi. Supponiamo ad esempio che il metodo numerico possa essere scritto nella forma generale

$$u_{n+1} = \sum_{j=0}^p a_j u_{n-j} + h \sum_{j=0}^p b_j f_{n-j} + hb_{-1} f_{n+1}, \quad n = p, p+1, \dots \quad (7.23)$$

per opportuni coefficienti $\{a_k\}$ e $\{b_k\}$ e per un opportuno intero $p \geq 0$. Un metodo di questo tipo è noto come *metodo multistep lineare* e $p+1$ denota il numero di passi (o *step*). Mostreremo alcuni esempi di metodi multistep nel paragrafo 7.6. Il polinomio di grado $p+1$ rispetto a r

$$\pi(r) = r^{p+1} - \sum_{j=0}^p a_j r^{p-j}$$

è detto il *primo polinomio caratteristico* associato con al metodo numerico (7.23); denotiamo le sue radici con r_j , $j = 0, \dots, p$. Si può allora provare che il metodo (7.23) è zero-stabile se e solo se vale la seguente *condizione delle radici*

$$\begin{cases} |r_j| \leq 1 \text{ per ogni } j = 0, \dots, p, \\ \text{inoltre } \pi'(r_j) \neq 0 \text{ per quei } j \text{ tali che } |r_j| = 1. \end{cases} \quad (7.24)$$

Ad esempio, per il metodo di Eulero in avanti abbiamo $p = 0$, $a_0 = 1$, $b_{-1} = 0$, $b_0 = 1$. Per il metodo di Eulero all'indietro abbiamo $p = 0$, $a_0 = 1$, $b_{-1} = 1$, $b_0 = 0$ e per il metodo di Crank-Nicolson $p = 0$, $a_0 = 1$, $b_{-1} = 1/2$, $b_0 = 1/2$. In tutti questi casi c'è una sola radice di $\pi(r)$ che vale 1 e, di conseguenza, tutti questi metodi sono zero-stabili.

La seguente proprietà, nota come teorema di equivalenza di Lax-Ritchmyer, fondamentale nella teoria dei metodi numerici (si veda, ad esempio, [IK66]) illustra il ruolo decisivo giocato dalla proprietà di zero-stabilità:

Ogni metodo consistente è convergente se e solo se è zero-stabile.

(7.25)

Coerentemente con quanto fatto in precedenza, l'errore di troncamento locale per un metodo multistep (7.23) è definito come

$$\tau_n(h) = \frac{1}{h} \left\{ y_{n+1} - \sum_{j=0}^p a_j y_{n-j} - h \sum_{j=0}^p b_j f(t_{n-j}, y_{n-j}) - h b_{-1} f(t_{n+1}, y_{n+1}) \right\}. \quad (7.26)$$

Il metodo è detto consistente se $\tau(h) = \max |\tau_n(h)|$ tende a zero per h che tende a zero. Questa condizione equivale a richiedere che

$$\sum_{j=0}^p a_j = 1, \quad -\sum_{j=0}^p j a_j + \sum_{j=-1}^p b_j = 1, \quad (7.27)$$

che corrispondere ad affermare che $r = 1$ è una radice del polinomio $\pi(r)$ (per la dimostrazione si veda ad esempio [QSS04, Capitolo 11]).

Si vedano gli Esercizi 7.4-7.5.



7.5 Stabilità su intervalli illimitati

Nel precedente paragrafo ci siamo occupati della risoluzione di problemi di Cauchy su intervalli limitati. In quel contesto il numero N_h di sottointervalli tende all'infinito soltanto se h tende a 0.

D'altra parte, esistono numerose situazioni nelle quali si è interessati a determinare la soluzione di un problema di Cauchy per tempi grandi, virtualmente infiniti. In questi casi, anche per h fissato, N_h tende comunque all'infinito e risultati quali (7.13) perdono di significato in quanto a secondo membro compaiono quantità illimitate. Si è pertanto interessati a caratterizzare metodi che, pur in corrispondenza di h sufficientemente grandi, consentano di ottenere un valore comunque accurato della soluzione $y(t)$ anche per t che tende all'infinito.

Sfortunatamente il metodo di Eulero in avanti, di così semplice implementazione, non gode di questa proprietà. Introduciamo il seguente *problema modello*

$$\begin{cases} y'(t) = \lambda y(t), & t \in (0, \infty), \\ y(0) = 1, \end{cases} \quad (7.28)$$

dove λ è un numero reale negativo. La soluzione esatta, $y(t) = e^{\lambda t}$, tende a 0 per t che tende all'infinito. Se applichiamo a (7.28) il metodo di Eulero in avanti, troviamo

$$u_0 = 1, u_{n+1} = u_n(1 + \lambda h) = (1 + \lambda h)^{n+1}, \quad n \geq 0. \quad (7.29)$$

Avremo $\lim_{n \rightarrow \infty} u_n = 0$ se e solo se

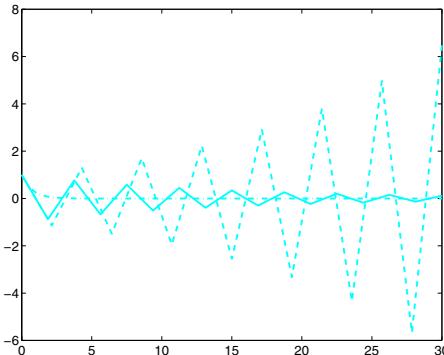


Figura 7.4. Soluzioni del problema (7.28), con $\lambda = -1$, ottenute con il metodo di Eulero in avanti, corrispondenti a $h = 30/14 (> 2)$ (linea tratteggiata), $h = 30/16 (< 2)$ (linea continua) e $h = 1/2$ (linea tratto-punto)

$$-1 < 1 + h\lambda < 1, \text{ ovvero } h < 2/|\lambda| \quad (7.30)$$

Questa condizione esprime la richiesta che, per h fissato, la soluzione numerica sia in grado di riprodurre l'andamento della soluzione esatta quando t_n tende all'infinito. Se $h > 2/|\lambda|$, allora $\lim_{n \rightarrow \infty} |u_n| = +\infty$; quindi (7.30) è una condizione di stabilità. Precisamente la proprietà che $\lim_{n \rightarrow \infty} u_n = 0$ è detta *assoluta stabilità*.

Esempio 7.3 Risolviamo con il metodo di Eulero in avanti il problema (7.28) con $\lambda = -1$. In tal caso dobbiamo avere $h < 2$ per garantire l'assoluta stabilità. In Figura 7.4 vengono riportate le soluzioni ottenute sull'intervallo $[0, 30]$ per tre diversi valori di h : $h = 30/14$ (che viola la condizione di stabilità), $h = 30/16$ (che soddisfa, seppur di poco, la condizione di stabilità), e $h = 1/2$. Si osserva che nel primo caso il valore assoluto della soluzione numerica non tende a zero per n che tende all'infinito (anzi, addirittura diverge).

Conclusioni analoghe valgono quando λ è un numero complesso (si veda il paragrafo 7.5.1) o quando $\lambda = \lambda(t)$ in (7.28) è una funzione negativa di t . In quest'ultimo caso, nella condizione di stabilità (7.30), $|\lambda|$ dovrà essere sostituito da $\max_{t \in [0, \infty)} |\lambda(t)|$. Questa condizione può essere indebolita se si utilizza un *passo variabile* h_n che tenga conto dell'andamento locale di $|\lambda(t)|$ in ciascun intervallo (t_n, t_{n+1}) .

In particolare, si può ricorrere al seguente metodo di Eulero in avanti *adattivo*:

posto $u_0 = y_0$ e $h_0 = 2\alpha/|\lambda(t_0)|$, allora

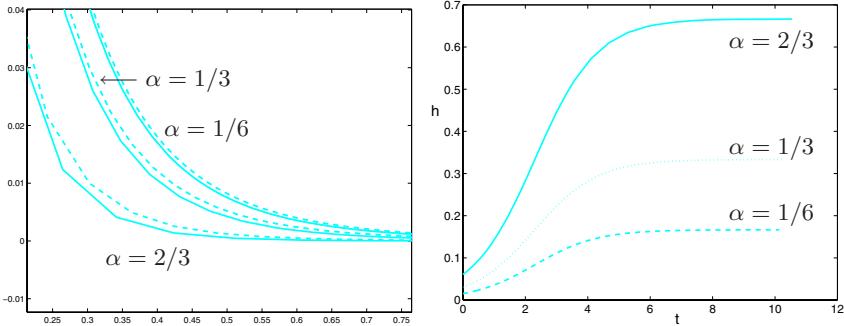


Figura 7.5. A sinistra: la soluzione numerica sull’intervallo temporale $(0, 0.8)$ ottenuta con il metodo di Eulero in avanti per $h = \alpha h_0$ (linea tratteggiata) e con il metodo adattivo (7.31) (linea continua) per tre diversi valori di α . A destra: l’andamento del passo variabile h_n per il metodo adattivo (7.31)

per $n = 0, 1, \dots$, calcolare

$$\begin{aligned} t_{n+1} &= t_n + h_n, \\ u_{n+1} &= u_n + h_n \lambda(t_n) u_n, \\ h_{n+1} &= 2\alpha / |\lambda(t_{n+1})|, \end{aligned} \tag{7.31}$$

dove α è una costante che deve essere minore di 1 in modo da garantire che il metodo sia assolutamente stabile.

Ad esempio, consideriamo il problema

$$y'(t) = -(10e^{-t} + 1)y(t), \quad t \in (0, 10),$$

con $y(0) = 1$. Essendo $|\lambda(t)|$ decrescente, la condizione più restrittiva per avere assoluta stabilità del metodo di Eulero in avanti è $h < h_0 = 2/|\lambda(0)| = 2/11$. In Figura 7.5, a sinistra, confrontiamo la soluzione ottenuta con il metodo di Eulero in avanti con quella ottenuta con il metodo adattivo (7.31) per tre diversi valori di α . Si noti che, anche se ogni valore $\alpha < 1$ è sufficiente a garantire la stabilità, per ottenere soluzioni accurate è necessario scegliere α sufficientemente piccolo. In Figura 7.5, a destra riportiamo i valori di h_n sull’intervallo $(0, 10]$ per i tre valori di α . Da questo grafico si deduce che la successione $\{h_n\}$ è monotona crescente.

Contrariamente al metodo di Eulero in avanti, i metodi di Eulero all’indietro e di Crank-Nicolson non richiedono limitazioni su h per garantire l’assoluta stabilità. Applicando il metodo di Eulero all’indietro al problema modello (7.28) infatti si trova $u_{n+1} = u_n + \lambda h u_{n+1}$ e

$$u_{n+1} = \left(\frac{1}{1 - \lambda h} \right)^{n+1}, \quad n \geq 0,$$

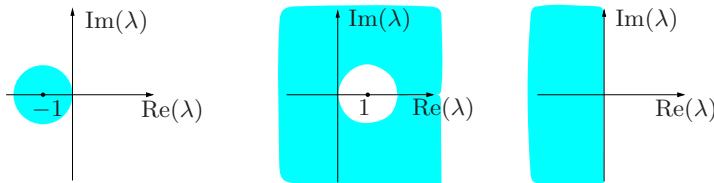


Figura 7.6. Le regioni di assoluta stabilità (in colore) per i metodi di Eulero in avanti (a sinistra), di Eulero all'indietro (al centro) e di Crank-Nicolson (a destra)

che tende a zero per $n \rightarrow \infty$ per tutti i valori $h > 0$. Analogamente applicando il metodo di Crank-Nicolson si trova

$$u_{n+1} = \left[\left(1 + \frac{h\lambda}{2} \right) / \left(1 - \frac{h\lambda}{2} \right) \right]^{n+1}, \quad n \geq 0,$$

che tende nuovamente a zero per $n \rightarrow \infty$ per tutti i possibili valori di $h > 0$. Deduciamo quindi che il metodo di Eulero in avanti è *condizionatamente assolutamente stabile*, mentre i metodi di Eulero all'indietro e di Crank-Nicolson sono *incondizionatamente assolutamente stabili*.

7.5.1 La regione di assoluta stabilità

Supponiamo ora che nel problema (7.28) λ sia un numero complesso con parte reale negativa. In tal caso la soluzione $u(t) = e^{\lambda t}$ tende ancora a 0 quando t tende all'infinito. Chiamiamo *regione di assoluta stabilità* \mathcal{A} di un metodo numerico l'insieme dei valori del piano complesso $z = h\lambda$ in corrispondenza dei quali il metodo è assolutamente stabile (ovvero si abbia $\lim_{n \rightarrow \infty} u_n = 0$).

Ad esempio, per il metodo di Eulero in avanti la regione di assoluta stabilità è costituita dai valori $z = h\lambda \in \mathbb{C}$ tali che $|1 + h\lambda| < 1$, ovvero dal cerchio di raggio unitario e centro $(-1, 0)$. Per il metodo di Eulero all'indietro la proprietà di assoluta stabilità è invece soddisfatta per ogni valore di $h\lambda$ che non appartiene al cerchio del piano complesso di raggio unitario e centro $(1, 0)$ (si veda la Figura 7.6). Infine, il metodo di Crank-Nicolson ha una regione di assoluta stabilità che coincide con il semipiano complesso dei numeri con parte reale strettamente negativa.

Metodi che risultino incondizionatamente assolutamente stabili per tutti i numeri complessi λ in (7.28) sono detti *A-stabili*. I metodi di Eulero all'indietro e di Crank-Nicolson sono dunque *A-stabili*, così come molti altri metodi impliciti. Questa proprietà rende i metodi impliciti interessanti, nonostante richiedano in generale costi computazionali decisamente più elevati dei metodi espliciti.

Esempio 7.4 Calcoliamo la restrizione cui deve soddisfare h qualora si utilizzi il metodo di Eulero esplicito per la risoluzione del problema di Cauchy $y'(t) = \lambda y$ con $\lambda = -1 + i$. Il valore di λ dato appartiene alla frontiera della regione di assoluta stabilità. Un qualunque $h \in (0, 1)$ sarà sufficiente ad assicurare che $h\lambda \in \mathcal{A}$. Se fosse $\lambda = -2 + 2i$ la diseguaglianza $|1 + h\lambda| < 1$ comporterebbe una restrizione più severa, $h \in (0, 1/2)$.

7.5.2 L'assoluta stabilità controlla le perturbazioni

Consideriamo il seguente *problema modello generalizzato*

$$\begin{cases} y'(t) = \lambda(t)y(t) + r(t), & t \in (0, +\infty), \\ y(0) = 1, \end{cases} \quad (7.32)$$

dove λ e r sono due funzioni continue e $-\lambda_{max} \leq \lambda(t) \leq -\lambda_{min}$ con $0 < \lambda_{min} \leq \lambda_{max} < +\infty$. In tal caso la soluzione esatta non tende necessariamente a zero quando t tende all'infinito. Ad esempio, se entrambi r e λ sono costanti abbiamo

$$y(t) = \left(1 + \frac{r}{\lambda}\right) e^{\lambda t} - \frac{r}{\lambda}$$

il cui limite per t che tende all'infinito è $-r/\lambda$. Dunque, in generale, non appare sensato richiedere che un metodo numerico sia assolutamente stabile quando applicato al problema (7.32). D'altra parte, mostreremo che un metodo numerico che sia assolutamente stabile per il problema modello (7.28), quando applicato al problema generalizzato (7.32) garantisce che le eventuali perturbazioni restino sotto controllo quando t tende all'infinito (accettando al più un'opportuna condizione sul passo di integrazione h).

Per semplicità limitiamo la nostra analisi al metodo di Eulero in avanti per il problema (7.32):

$$\begin{cases} u_{n+1} = u_n + h(\lambda_n u_n + r_n), & n \geq 0, \\ u_0 = 1. \end{cases}$$

La soluzione è (si veda l'Esercizio 7.9)

$$u_n = u_0 \prod_{k=0}^{n-1} (1 + h\lambda_k) + h \sum_{k=0}^{n-1} r_k \prod_{j=k+1}^{n-1} (1 + h\lambda_j), \quad (7.33)$$

avendo posto $\lambda_k = \lambda(t_k)$ e $r_k = r(t_k)$ ed avendo adottato la convenzione che la produttoria sia uguale a 1 se $k+1 > n-1$. Consideriamo ora il problema “perturbato”

$$\begin{cases} z_{n+1} = z_n + h(\lambda_n z_n + r_n + \rho_{n+1}), & n \geq 0, \\ z_0 = u_0 + \rho_0, \end{cases} \quad (7.34)$$

dove ρ_0, ρ_1, \dots sono perturbazioni note. Possiamo immaginare che in questo semplice modello ρ_0 e ρ_{n+1} tengano conto del fatto che né u_0 , né r_n possono essere determinati esattamente. (Se si dovesse tener precisamente conto di *tutti* gli errori di arrotondamento che vengono introdotti ad ogni passo temporale $n \geq 0$, il problema perturbato sarebbe molto più complesso e difficile da analizzare.) La soluzione di (7.34) assume una forma simile a (7.33) purché u_k venga sostituito da z_k e r_k da $r_k + \rho_{k+1}$, per ogni $k = 0, \dots, n - 1$. Si ha allora

$$z_n - u_n = \rho_0 \prod_{k=0}^{n-1} (1 + h\lambda_k) + h \sum_{k=0}^{n-1} \rho_{k+1} \prod_{j=k+1}^{n-1} (1 + h\lambda_j). \quad (7.35)$$

La quantità $|z_n - u_n|$ è detta *errore di perturbazione* al passo n ; questa quantità non dipende dalla funzione $r(t)$.

i. Consideriamo dapprima il caso speciale in cui λ_k e ρ_k sono due costanti, pari a λ e ρ , rispettivamente. Supponiamo che $h < h_0(\lambda) = 2/|\lambda|$, che è la condizione che assicura l'assoluta stabilità per il metodo di Eulero in avanti quando applicato al problema modello (7.28). Utilizzando la seguente identità

$$\sum_{k=0}^{n-1} a^k = \frac{1 - a^n}{1 - a}, \quad \text{se } |a| \neq 1, \quad (7.36)$$

troviamo

$$z_n - u_n = \rho \left\{ (1 + h\lambda)^n \left(1 + \frac{1}{\lambda} \right) - \frac{1}{\lambda} \right\}. \quad (7.37)$$

Segue che l'errore di perturbazione soddisfa (si veda l'Esercizio 7.10)

$$|z_n - u_n| \leq \varphi(\lambda)|\rho|, \quad (7.38)$$

con $\varphi(\lambda) = 1$ se $\lambda \leq -1$, mentre $\varphi(\lambda) = |1 + 2/\lambda|$ se $-1 \leq \lambda < 0$. Possiamo quindi concludere che l'errore di perturbazione è limitato da $|\rho|$ per una costante che non dipende da n e h . Inoltre,

$$\lim_{n \rightarrow \infty} |z_n - u_n| = \frac{\rho}{|\lambda|}.$$

La Figura 7.7 corrisponde al caso in cui $\rho = 0.1$, $\lambda = -2$ (sinistra) e $\lambda = -0.5$ (destra). In entrambi i casi abbiamo preso $h = h_0(\lambda) - 0.01$. Naturalmente, l'errore di perturbazione esplode al crescere di n se si viola la limitazione $h < h_0(\lambda)$.

ii. Nel caso generale quando λ e r non sono costanti, richiediamo che h soddisfi la restrizione $h < h_0(\lambda)$, dove stavolta $h_0(\lambda) = 2/\lambda_{max}$. Allora,

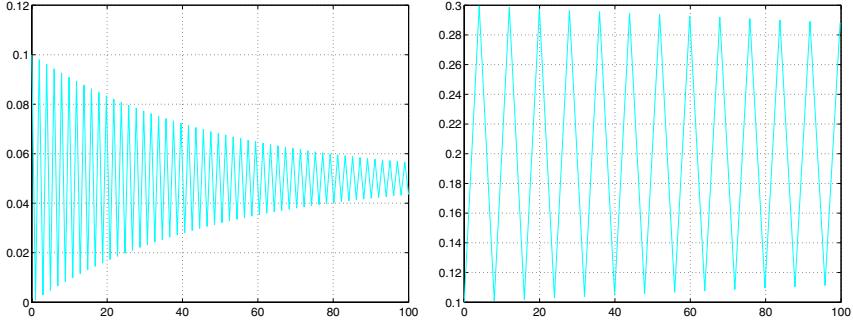


Figura 7.7. L'errore di perturbazione quando $\rho = 0.1$: $\lambda = -2$ (sinistra) e $\lambda = -0.5$ (destra). In entrambi i casi si è usato $h = h_0(\lambda) - 0.01$

$$|1 + h\lambda_k| \leq a(h) = \max\{|1 - h\lambda_{\min}|, |1 - h\lambda_{\max}|\}.$$

Essendo $a(h) < 1$, possiamo usare ancora l'identità (7.36) in (7.35) ricavando

$$|z_n - u_n| \leq \rho_{\max} \left([a(h)]^n + h \frac{1 - [a(h)]^n}{1 - a(h)} \right), \quad (7.39)$$

dove $\rho_{\max} = \max |\rho_k|$. Si noti che $a(h) = |1 - h\lambda_{\min}|$ se $h \leq h^*$ mentre $a(h) = |1 - h\lambda_{\max}|$ se $h^* \leq h < 2/h_{\max}(\lambda)$, avendo posto $h^* = 2/(\lambda_{\min} + \lambda_{\max})$. Quando $h \leq h^*$, si ha $a(h) > 0$ e si deduce che

$$|z_n - u_n| \leq \frac{\rho_{\max}}{\lambda_{\min}} [1 - [a(h)]^n (1 - \lambda_{\min})], \quad (7.40)$$

quindi

$$\lim_{n \rightarrow \infty} \sup |z_n - u_n| \leq \frac{\rho_{\max}}{\lambda_{\min}}, \quad (7.41)$$

dal che si conclude ancora che l'errore di perturbazione è limitato da ρ_{\max} per una costante che non dipende da n e h (anche se ora le oscillazioni non vengono più smorzate come nel caso precedente).

Di fatto, una conclusione analoga vale anche quando $h^* \leq h \leq 2/h_0(\lambda)$, anche se ciò non segue dalla precedente diseguaglianza (7.40) che in questo caso appare troppo pessimistica.

In Figura 7.8 riportiamo gli errori di perturbazione calcolati per il problema (7.32), dove $\lambda_k = \lambda(t_k) = -2 - \sin(t_k)$, $\rho_k = \rho(t_k) = 0.1 \sin(t_k)$ con $h < h^*$ (a sinistra) e con $h^* \leq h < h_0(\lambda)$ (a destra).

iii. Consideriamo ora il caso generale. La soluzione numerica del problema di Cauchy (7.5) può essere messa in relazione con quella del problema modello generalizzato (7.32) nei casi in cui si abbia

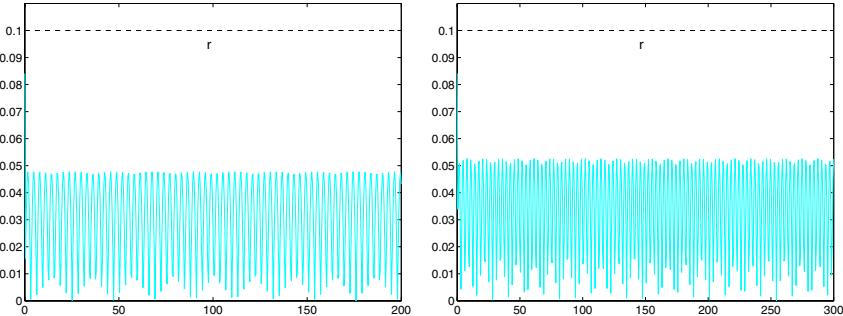


Figura 7.8. L'errore di perturbazione quando $\rho(t) = 0.1 \sin(t)$ e $\lambda(t) = -2 - \sin(t)$ per $t \in (0, nh)$ con $n = 500$: il passo di discretizzazione è $h = h^* - 0.1 = 0.4$ (a sinistra) e $h = h^* + 0.1 = 0.6$ (a destra)

$$-\lambda_{\max} < \partial f / \partial y(t, y) < -\lambda_{\min}, \forall t \geq 0, \forall y \in (-\infty, \infty),$$

per opportuni valori $\lambda_{\min}, \lambda_{\max} \in (0, +\infty)$. A questo scopo, per ogni t nel generico intervallo (t_n, t_{n+1}) , sottraiamo (7.6) da (7.22) in modo da ottenere la seguente equazione per l'errore di perturbazione

$$z_n - u_n = (z_{n-1} - u_{n-1}) + h \{f(t_{n-1}, z_{n-1}) - f(t_{n-1}, u_{n-1})\} + h\rho_n.$$

Applicando il teorema del valor medio, ricaviamo

$$f(t_{n-1}, z_{n-1}) - f(t_{n-1}, u_{n-1}) = \lambda_{n-1}(z_{n-1} - u_{n-1}),$$

dove $\lambda_{n-1} = f_y(t_{n-1}, \xi_{n-1})$, avendo posto $f_y = \partial f / \partial y$ ed essendo ξ_{n-1} un punto opportuno nell'intervallo di estremi u_{n-1} e z_{n-1} . Allora

$$z_n - u_n = (1 + h\lambda_{n-1})(z_{n-1} - u_{n-1}) + h\rho_n.$$

Applicando ricorsivamente questa formula troviamo l'identità (7.35), a partire dalla quale giungiamo alle stesse conclusioni ottenute al punto *ii.*, purché valga la condizione di stabilità $0 < h < 2/\lambda_{\max}$.

Esempio 7.5 Consideriamo ad esempio il problema di Cauchy

$$y'(t) = \arctan(3y) - 3y + t, \quad t > 0, \quad y(0) = 1. \quad (7.42)$$

Essendo $f_y = 3/(1+9y^2) - 3$ negativa, possiamo scegliere $\lambda_{\max} = \max |f_y| = 3$ e porre $h < 2/3$. Possiamo quindi aspettarci che le perturbazioni nel metodo di Eulero in avanti restino controllate purché $h < 2/3$. Questa conclusione è confermata dai risultati sperimentali riportati in Figura 7.9. Si noti che in questo esempio, prendendo $h = 2/3 + 0.01$ (che viola la precedente condizione di stabilità) l'errore di perturbazione esplode al crescere di t .

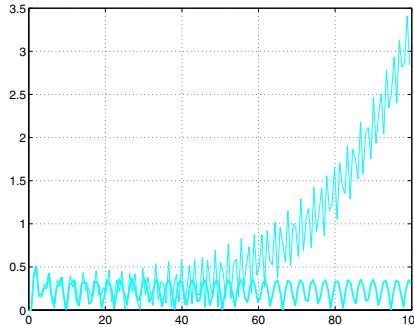


Figura 7.9. L'errore di perturbazione quando $\rho(t) = \sin(t)$ con $h = 2/\lambda_{max} - 0.01$ (linea spessa) e $h = 2/\lambda_{max} + 0.01$ (linea sottile) per il problema di Cauchy (7.42)

Esempio 7.6 Cerchiamo un limite su h che garantisca la stabilità per il metodo di Eulero in avanti applicato al problema di Cauchy

$$y' = 1 - y^2, \quad t > 0, \quad (7.43)$$

con $y(0) = (e-1)/(e+1)$. La soluzione esatta è $y(t) = (e^{2t+1}-1)/(e^{2t+1}+1)$ e $f_y = -2y$. Poiché $f_y \in (-2, -0.9)$ per ogni $t > 0$, possiamo prendere h minore di $h_0 = 1$. In Figura 7.10, a sinistra, riportiamo le soluzioni ottenute sull'intervallo $(0, 35)$ con $h = 20/21$ (linea spessa) e $h = 20/19$ (linea sottile). In entrambi i casi le soluzioni oscillano, ma restano limitate. Inoltre, nel primo caso nel quale la condizione di stabilità è soddisfatta, le oscillazioni vengono smorzate e la soluzione numerica tende a quella esatta al crescere di t . In Figura 7.10, a destra, riportiamo gli errori di perturbazione corrispondenti a $\rho(t) = \sin(t)$ con $h = 20/21$ (linea spessa) e $h = 20/19$ (linea sottile). In entrambi i casi gli errori di perturbazione si mantengono limitati. Inoltre, nel primo caso il limite superiore (7.41) è soddisfatto.

In tutti quei casi in cui non sono disponibili informazioni su y , il calcolo di $\lambda_{max} = \max|f_y|$ può non essere agevole. Si può in questa circostanza seguire un approccio euristico adottando una procedura di adattività del passo di integrazione. Precisamente, si potrebbe prendere $t_{n+1} = t_n + h_n$, dove

$$h_n < 2 \frac{\alpha}{|f_y(t_n, u_n)|},$$

per opportuni valori di α strettamente minori di 1. Si noti che il denominatore dipende dal valore u_n che è noto. In Figura 7.11 riportiamo gli errori di perturbazione corrispondenti all'Esempio 7.6 per due diversi valori di α .

L'analisi precedente può essere condotta anche per altri tipi di metodi, in particolare per i metodi di Eulero all'indietro e di Crank-Nicolson. Per questi metodi, che sono A-stabili, si ricavano le stesse conclusioni sul

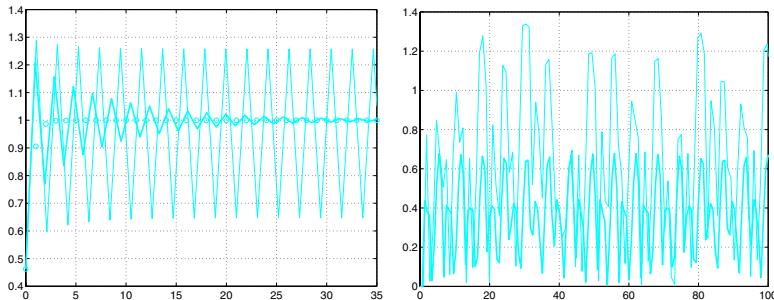


Figura 7.10. A sinistra, le soluzioni numeriche del problema (7.43) calcolate con il metodo di Eulero in avanti per $h = 20/19$ (linea sottile) e per $h = 20/21$ (linea spessa). I valori della soluzione esatta sono stati indicati con dei cerchietti. A destra, gli errori di perturbazione corrispondenti a $\rho(t) = \sin(t)$ per $h = 20/21$ (linea spessa) e $h = 20/19$ (linea sottile)

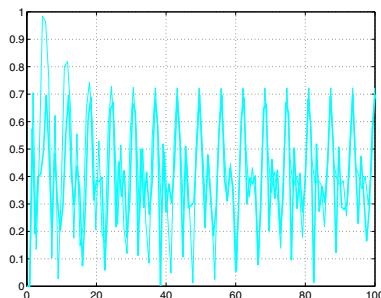


Figura 7.11. Gli errori di perturbazione corrispondenti a $\rho(t) = \sin(t)$ con $\alpha = 0.8$ (linea spessa) e $\alpha = 0.9$ (linea sottile) per l’Esempio 7.6, usando la strategia adattiva

controllo dell’errore di perturbazione, senza tuttavia richiedere alcuna limitazione sul passo di integrazione. Di fatto, basta sostituire nell’analisi precedente il termine $1 + h\lambda_n$ con $(1 - h\lambda_n)^{-1}$ nel caso del metodo di Eulero all’indietro e con $(1 + h\lambda_n/2)/(1 - h\lambda_n/2)$ nel caso del metodo di Crank-Nicolson.



Riassumendo

1. Un metodo è assolutamente stabile quando, applicato al problema modello (7.28), genera una soluzione u_n che tende a zero quando t_n tende all’infinito;
2. un metodo è detto *A-stabile* quando è assolutamente stabile per ogni possibile scelta del passo di integrazione h (in caso contrario si dice

- che il metodo è condizionatamente assolutamente stabile e h dovrà essere minore di una costante che dipende da $|\lambda|$);
3. quando un metodo assolutamente stabile viene applicato ad un problema modello generalizzato, come (7.32), l'errore di perturbazione, che è il valore assoluto della differenza fra la soluzione perturbata e quella imperturbata, è limitato uniformemente rispetto a h . Per tale ragione, diciamo che un metodo assolutamente stabile *controlla le perturbazioni*;
 4. l'analisi di assoluta stabilità per il problema modello può essere sfruttata per trovare condizioni di stabilità sul passo di integrazione anche per il generico problema di Cauchy non lineare (7.5) quando f sia tale che $\partial f / \partial y < 0$. In tal caso la condizione di stabilità richiede che il passo di integrazione sia scelto come una funzione di $\partial f / \partial y$. Precisamente, il nuovo intervallo di integrazione $[t_n, t_{n+1}]$ verrà scelto in modo tale che $h_n = t_{n+1} - t_n$ soddisfi $h_n < 2\alpha / |\partial f(t_n, u_n) / \partial y|$ per un opportuno α minore di 1.

Si vedano gli Esercizi 7.6-7.13.



7.6 Metodi di ordine elevato

Tutti i metodi presentati finora sono esempi elementari di metodi ad un passo. Accenniamo ora ad altri metodi che consentono il raggiungimento di un maggior ordine di accuratezza, i *metodi Runge-Kutta* ed i *metodi multistep*. I metodi Runge-Kutta (in breve, RK) sono ancora metodi ad un passo che tuttavia comportano diverse valutazioni della funzione $f(t, y)$ in ciascun intervallo $[t_n, t_{n+1}]$. Nella sua forma più generale, un metodo RK può essere scritto come

$$u_{n+1} = u_n + h \sum_{i=1}^s b_i K_i, \quad n \geq 0,$$

dove

$$K_i = f(t_n + c_i h, u_n + h \sum_{j=1}^s a_{ij} K_j), \quad i = 1, 2, \dots, s$$

e s denota il numero di *stadi* del metodo. I coefficienti $\{a_{ij}\}$, $\{c_i\}$ e $\{b_i\}$ caratterizzano completamente un metodo RK e sono generalmente raccolti nel cosiddetto *array di Butcher*

$$\begin{array}{c|cc} \mathbf{c} & \mathbf{A} \\ \hline & \mathbf{b}^T \end{array},$$

essendo $A = (a_{ij}) \in \mathbb{R}^{s \times s}$, $\mathbf{b} = (b_1, \dots, b_s)^T \in \mathbb{R}^s$ e $\mathbf{c} = (c_1, \dots, c_s)^T \in \mathbb{R}^s$. Se i coefficienti a_{ij} di A sono uguali a zero per $j \geq i$, con $i = 1, 2, \dots, s$, allora ciascun K_i può essere esplicitamente calcolato usando gli $i-1$ coefficienti K_1, \dots, K_{i-1} che sono già stati calcolati. In tal caso il metodo RK si dice *esplicito*. In caso contrario, il metodo è detto *implicito* e per calcolare i coefficienti K_i si deve risolvere un sistema non lineare di dimensione s .

Uno tra i più noti metodi RK assume la seguente forma

$$u_{n+1} = u_n + \frac{h}{6}(K_1 + 2K_2 + 2K_3 + K_4) \quad (7.44)$$

dove

$$\begin{aligned} K_1 &= f_n, & 0 \\ K_2 &= f(t_n + \frac{h}{2}, u_n + \frac{h}{2}K_1), & \frac{1}{2} \Big| \frac{1}{2} \\ K_3 &= f(t_n + \frac{h}{2}, u_n + \frac{h}{2}K_2), & \frac{1}{2} \Big| 0 \quad \frac{1}{2} \\ K_4 &= f(t_{n+1}, u_n + hK_3), & 1 \Big| 0 \quad 0 \quad 1 \\ && \hline \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{aligned}.$$

Esso metodo si può derivare dalla (7.18) usando la formula di quadratura di Simpson (4.23) per approssimare l'integrale fra t_n e t_{n+1} . Si tratta di un metodo esplicito di ordine 4 rispetto a h ; ad ogni passo, richiede 4 nuove valutazioni della funzione f . Naturalmente si possono costruire molti altri metodi RK, sia esplicativi che impliciti di ordine arbitrario. Ad esempio, un metodo RK隐式 di ordine 4 a 2 stadi è definito dal seguente array di Butcher

$$\begin{array}{c|cc} \frac{3-\sqrt{3}}{6} & \frac{1}{4} & \frac{3-2\sqrt{3}}{12} \\ \frac{3+\sqrt{3}}{6} & \frac{3+2\sqrt{3}}{12} & \frac{1}{4} \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}$$

La regione di assoluta stabilità \mathcal{A} dei metodi RK, anche esplicativi, può crescere in estensione al crescere dell'ordine: un esempio è fornito dal grafico in Figura 7.13 a sinistra, dove è stato riportata \mathcal{A} per alcuni metodi RK esplicativi di ordine crescente: RK1 è il metodo di Eulero in avanti, RK2 il metodo di Eulero migliorato (7.51), RK3 presenta il seguente *array* di Butcher

$$\begin{array}{c|cc} 0 & \\ \frac{1}{2} & \frac{1}{2} \\ 1 & -1 \quad 2 \\ \hline & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \end{array} \quad (7.45)$$

e RK4 il metodo (7.44) appena presentato.

I metodi RK sono alla base di alcuni programmi MATLAB il cui nome contiene la radice `ode` seguita da numeri o lettere. In particolare, `ode45` è basato su una coppia di metodi RK esplicativi (la cosiddetta coppia di Dormand-Prince) di ordine 4 e 5, rispettivamente. `ode23` implementa un'altra coppia di metodi RK esplicativi (la coppia di Bogacki e Shampine). In questi metodi il passo di integrazione varia in modo da garantire che l'errore si mantenga al di sotto di una tolleranza fissata (in tutti questi programmi la tolleranza di default `RelTol` per l'errore relativo è fissata pari a 10^{-3}). Il programma `ode23tb` implementa invece un metodo Runge-Kutta implicito il cui primo stadio (relativo alla valutazione di K_1) è basato sulla formula del trapezio, mentre il secondo stadio è basato su una formula di differenziazione all'indietro di ordine 2 (si veda (7.48)).

`ode45``ode23``ode23tb`

I metodi multistep (o multipasso) (si veda la (7.23)) consentono di ottenere un ordine di accuratezza elevato coinvolgendo i valori $u_n, u_{n-1}, \dots, u_{n-p}$ nella determinazione di u_{n+1} . Essi possono essere derivati ad esempio applicando prima la formula (7.18), quindi approssimando l'integrale che vi compare con una formula di quadratura che utilizza il polinomio interpolatore di f su un opportuno insieme di nodi. Un esempio notevole di metodo multistep è la formula (esplicita) a tre passi ($p = 2$), del terz'ordine di Adams-Basforth (AB3)

$$u_{n+1} = u_n + \frac{h}{12} (23f_n - 16f_{n-1} + 5f_{n-2}) \quad (7.46)$$

ottenuta sostituendo f nella (7.18) con il suo polinomio interpolatore di grado 2 nei nodi t_{n-2}, t_{n-1}, t_n . Un altro importante esempio di metodo a tre passi del quart'ordine (implicito) è dato dalla formula di Adams-Moulton (AM4)

$$u_{n+1} = u_n + \frac{h}{24} (9f_{n+1} + 19f_n - 5f_{n-1} + f_{n-2}) \quad (7.47)$$

ottenuta approssimando f in (7.18) con il suo polinomio interpolatore di grado 3 nei nodi $t_{n-2}, t_{n-1}, t_n, t_{n+1}$.

Un'altra famiglia di metodi multistep si ottiene scrivendo l'equazione differenziale al tempo t_{n+1} ed approssimando $y'(t_{n+1})$ con un rapporto incrementale all'indietro di ordine elevato. Un esempio è costituito dalla formula di differenziazione all'indietro del second'ordine (*backward difference formula*, in breve BDF2) a due passi (implicito)

$$u_{n+1} = \frac{4}{3}u_n - \frac{1}{3}u_{n-1} + \frac{2h}{3}f_{n+1} \quad (7.48)$$

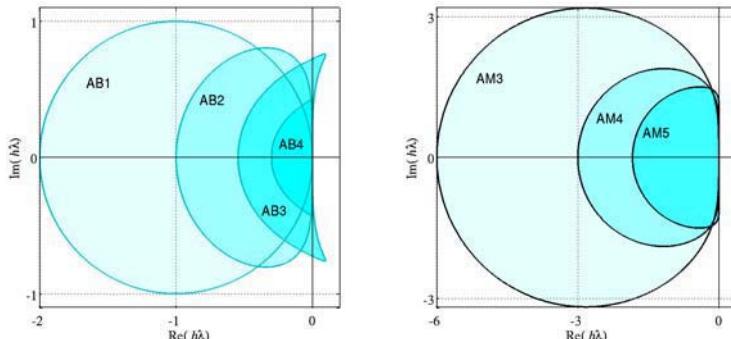


Figura 7.12. Le regioni di assoluta stabilità di alcuni metodi di Adams-Basforth (a sinistra) e di Adams-Moulton (a destra)

o di quella a tre passi (BDF3), del terz'ordine (implicita)

$$u_{n+1} = \frac{18}{11}u_n - \frac{9}{11}u_{n-1} + \frac{2}{11}u_{n-2} + \frac{6h}{11}f_{n+1} \quad (7.49)$$

Tutti questi metodi possono essere scritti nella forma generale (7.23). È facile verificare che per tutti le condizioni (7.27) sono soddisfatte e, di conseguenza, tutti questi metodi sono consistenti. Inoltre, tutti questi metodi sono zero-stabili. In effetti, per entrambe le formule (7.46) e (7.47) il primo polinomio caratteristico è $\pi(r) = r^3 - r^2$ le cui radici sono $r_0 = 1$, $r_1 = r_2 = 0$, il primo polinomio caratteristico di (7.48) è $\pi(r) = r^2 - (4/3)r + 1/3$ ed ha radici $r_0 = 1$ e $r_1 = 1/3$, mentre quello di (7.49) è $\pi(r) = r^3 - 18/11r^2 + 9/11r - 2/11$ che ha radici $r_0 = 1$, $r_1 = 0.3182 + 0.2839i$, $r_2 = 0.3182 - 0.2839i$, dove i è l'unità immaginaria. In tutti i casi, la condizione delle radici (7.24) è soddisfatta.

Inoltre, quando applicati al problema modello (7.28), AB3 è assolutamente stabile se $h < 0.545/|\lambda|$, mentre AM4 è assolutamente stabile se $h < 3/|\lambda|$. Il metodo BDF2 è A-stabile, mentre BDF3 è incondizionatamente assolutamente stabile per tutti i numeri reali negativi λ , ma non per tutti i $\lambda \in \mathbb{C}$ con parte reale negativa. In altre parole, il metodo BDF3 non è A-stabile. Più in generale, in accordo con la seconda barriera di Dahlquist, non ci sono metodi multistep *A-stabili* di ordine strettamente maggiore di 2. In Figura 7.12 riportiamo le regioni di assoluta stabilità di alcuni metodi di Adams-Basforth e di Adams-Moulton. Si noti come l'estensione della regione decresca al crescere dell'ordine di convergenza. Nei grafici di Figura 7.13 a destra riportiamo le regioni (ilimitate) di assoluta stabilità di alcuni metodi BDF: anch'esse ricoprono un'area del piano complesso che si riduce al crescere dell'ordine, contrariamente a quelle dei metodi Runge-Kutta (riportate a sinistra) che al crescere dell'ordine aumentano di estensione.

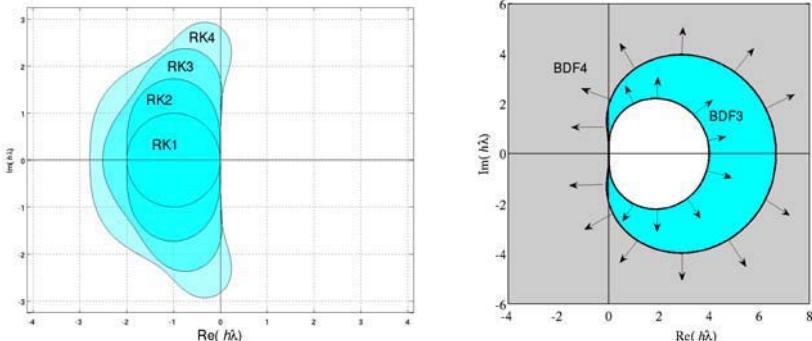


Figura 7.13. Le regioni di assoluta stabilità di alcuni metodi RK esplicativi (a sinistra) e BDF (a destra). In quest'ultimo caso le regioni sono illimitate e si estendono nella direzione indicata dalle frecce

Osservazione 7.3 È possibile calcolare il bordo $\partial\mathcal{A}$ della regione di assoluta stabilità \mathcal{A} di un metodo multistep con un semplice artificio. Il bordo è infatti costituito dai numeri complessi $h\lambda$ tali

$$h\lambda = r^{p+1} - \sum_{j=0}^p a_j r^{p-j} \Bigg/ \sum_{j=-1}^p b_j r^{p-j}, \quad (7.50)$$

con r numero complesso di modulo unitario. Di conseguenza, per ottenere in MATLAB una rappresentazione approssimata di $\partial\mathcal{A}$ è sufficiente valutare il secondo membro di (7.50) al variare di r sulla circonferenza unitaria (ad esempio, ponendo $r = \exp(i*\pi*(0:2000)/1000)$, dove i è l'unità immaginaria). I grafici delle Figure 7.12 e 7.13 sono stati ottenuti in questo modo.

In accordo con la prima barriera di Dahlquist l'ordine massimo q di un metodo multistep a $p + 1$ passi è $q = p + 1$ se il metodo è esplicito, mentre per un metodo implicito si ha $q = p + 2$ se $p + 1$ è dispari, $q = p + 3$ se $p + 1$ è pari.

Osservazione 7.4 (Metodi compositi ciclici) È possibile superare le barriere di Dahlquist combinando opportunamente più metodi multipasso. Ad esempio, i seguenti due metodi

$$\begin{aligned} u_{n+1} &= -\frac{8}{11}u_n + \frac{19}{11}u_{n-1} + \frac{h}{33}(30f_{n+1} + 57f_n + 24f_{n-1} - f_{n-2}), \\ u_{n+1} &= \frac{449}{240}u_n + \frac{19}{30}u_{n-1} - \frac{361}{240}u_{n-2} \\ &\quad + \frac{h}{720}(251f_{n+1} + 456f_n - 1347f_{n-1} - 350f_{n-2}), \end{aligned}$$

hanno ordine 5, ma sono instabili. Utilizzandoli però in modo combinato (il primo se n è pari, il secondo se n è dispari) producono un metodo A-stabile di ordine 5 a 3 passi.

Anche i metodi multistep sono ampiamente implementati in MATLAB, ad esempio nel programma `ode15s`.

7.7 I metodi predictor-corrector

Come abbiamo osservato nel paragrafo 7.2, se la funzione f del problema di Cauchy è non lineare rispetto a y , i metodi impliciti generano ad ogni passo un problema non lineare nell'incognita u_{n+1} . Per risolverlo si può ricorrere ad uno fra i metodi che abbiamo presentato nel capitolo 2 oppure, come abbiamo fatto nei Programmi 18 e 19, utilizzare la funzione `fzero`.

Un'ulteriore alternativa consiste nell'eseguire delle iterazioni di punto fisso ad ogni passo temporale. Ad esempio, per il metodo di Crank-Nicolson (7.17), per $k = 0, 1, \dots$, fino a convergenza si calcola

$$u_{n+1}^{(k+1)} = u_n + \frac{h}{2} \left[f_n + f(t_{n+1}, u_{n+1}^{(k)}) \right].$$

Si può dimostrare che se il dato iniziale $u_{n+1}^{(0)}$ viene scelto opportunamente basta una sola iterazione di punto fisso per ottenere una soluzione numerica $u_{n+1}^{(1)}$ la cui accuratezza sia dello stesso ordine della soluzione u_{n+1} calcolata dal metodo隐式的 originale. Precisamente, se il metodo隐式的 ha ordine p (≥ 2), il dato iniziale $u_{n+1}^{(0)}$ dovrà essere generato da un metodo esplicito per lo meno accurato di ordine $p - 1$.

Ad esempio, se si usa il metodo di Crank-Nicolson e lo si inizializza con il metodo (del prim'ordine) di Eulero in avanti, si ottiene il *metodo di Heun* (anche chiamato *metodo di Eulero migliorato*) che è un metodo Runge-Kutta esplicito di ordine 2

$$\begin{aligned} u_{n+1}^* &= u_n + hf_n, \\ u_{n+1} &= u_n + \frac{h}{2} [f_n + f(t_{n+1}, u_{n+1}^*)] \end{aligned} \tag{7.51}$$

In generale, il metodo esplicito viene denominato *predictor*, mentre il metodo implicito è detto *corrector*. Un altro esempio di questa famiglia di metodi è dato dalla combinazione del metodo (AB3) (7.46), usato come predictor, con il metodo (AM4) (7.47), usato come corrector. Schemi di questo genere vengono quindi chiamati metodi *predictor-corrector*. Essi garantiscono l'ordine di accuratezza del metodo corrector. D'altra parte, essendo esplicativi, presentano una regione di assoluta stabilità che è ridotta rispetto a quella del puro metodo corrector, ma più estesa di quella del puro predictor (si vedano ad esempio le regioni di assoluta stabilità riportate nei grafici di Figura 7.14). Questi schemi non sono

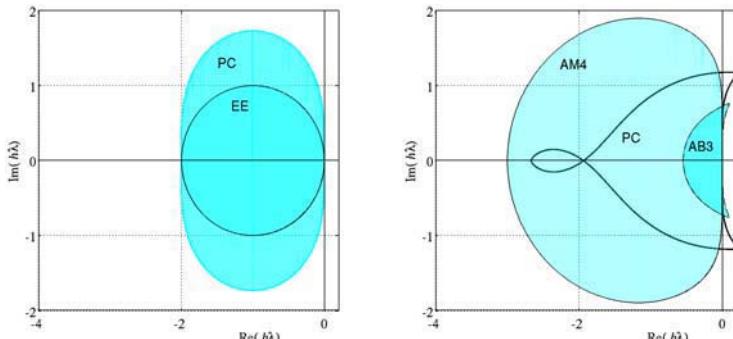


Figura 7.14. Le regioni di assoluta stabilità dei metodi predictor-corrector ottenuti combinando i metodi di Eulero esplicito (EE) a sinistra e AB3 e AM4 a destra. Si noti la riduzione dell'estensione della regione rispetto a quella dei corrispondenti metodi impliciti (nel primo caso la regione del metodo di Crank-Nicolson non è stata riportata in quanto coincide con tutto il semipiano complesso $\text{Re}(h\lambda) < 0$)

pertanto adeguati per la risoluzione di problemi di Cauchy su intervalli illimitati.

Nel Programma 20 implementiamo un generico metodo predictor-corrector. Le stringhe `predictor` e `corrector` identificano il tipo di metodo scelto. Ad esempio, se si usano le funzioni `eeonestep` e `cnonestep`, definite nel Programma 21, richiamiamo `predcor` come segue

```
>> [t,u]=predcor(f,[t0,T],y0,N,'feonestep','cnonestep');
```

ottenendo così il metodo di Heun.

Programma 20 - `predcor` : un generico metodo predictor-corrector

```
function [t,u]=predcor(odefun,tspan,y,Nh,pred,corr,varargin)
%PREDCOR Risolve un'equazione differenziale con un metodo predictor-corrector.
% [T,Y]=PREDCOR(ODEFUN,TSPAN,Y0,NH,PRED,CORR) con TSPAN =
% [T0 TFINAL] integra il sistema di equazioni differenziali ordinarie
% y' = f(t,y) dal tempo T0 a TFINAL con condizione iniziale Y0
% usando un metodo predictor-corrector su una griglia di NH intervalli equispaziati.
% La funzione ODEFUN(T,Y) restituisce un vettore corrispondente a f(t,y).
% Ogni riga nella matrice delle soluzioni Y corrisponde alla soluzione calcolata al
% corrispondente tempo nel vettore T. Le function PRED e CORR
% identificano il tipo di metodo predictor-corrector scelto.
% [T,Y] = PREDCOR(ODEFUN,TSPAN,Y0,NH,PRED,CORR,P1,P2,...) passa
% i parametri addizionali P1,P2,... alle funzioni ODEFUN, PRED e
% CORR come ODEFUN(T,Y,P1,P2...), PRED(T,Y,P1,P2...), CORR(T,Y,P1,P2...).
h=(tspan(2)-tspan(1))/Nh; tt=[tspan(1):h:tspan(2)];
u=y; [n,m]=size(u); if n ensuremathj m, u=u'; end
for t=tt(1:end-1)
```



```

y = u(:,end); fn = feval(odefun,t,y,varargin{:});
upre = feval(pred,t,y,h,fn);
ucor = feval(corr,t+h,y,upre,h,odefun,fn,varargin{:});
u = [u, ucor];
end
t = tt;
return

```



Programma 21 - onestep : un passo del metodo di Eulero in avanti (**feonestep**), un passo del metodo di Eulero all'indietro (**beonestep**), uno di Crank-Nicolson (**cnonestep**)

```

function [u]=feonestep(t,y,h,f)
u = y + h*f;
return

function [u]=beonestep(t,u,y,h,f,fn,varargin)
u = u + h*feval(f,t,y,varargin{:});
return

function [u]=cnonestep(t,u,y,h,f,fn,varargin)
u = u + 0.5*h*(feval(f,t,y,varargin{:})+fn);
return

```

ode113

Il programma MATLAB **ode113** implementa un metodo predictor-
corrector di Adams Moulton/Bashforth con passo di discretizzazione
variabile.



Si vedano gli Esercizi 7.14-7.17.

7.8 Sistemi di equazioni differenziali

Consideriamo il seguente sistema di equazioni differenziali di ordine uno nelle incognite $y_1(t), \dots, y_m(t)$

$$\begin{cases} y'_1(t) = f_1(t, y_1, \dots, y_m), \\ \vdots \\ y'_m(t) = f_m(t, y_1, \dots, y_m), \end{cases}$$

dove $t \in (t_0, T]$, con condizioni iniziali

$$y_1(t_0) = y_{0,1}, \dots, y_m(t_0) = y_{0,m}.$$

Per la sua risoluzione si potrebbe applicare a ciascuna delle equazioni che compongono il sistema, uno dei metodi introdotti precedentemente per un problema scalare. Ad esempio, il passo n -esimo del metodo di Eulero in avanti diverrebbe

$$\left\{ \begin{array}{l} u_{n+1,1} = u_{n,1} + h f_1(t_n, u_{n,1}, \dots, u_{n,m}), \\ \vdots \\ u_{n+1,m} = u_{n,m} + h f_m(t_n, u_{n,1}, \dots, u_{n,m}). \end{array} \right.$$

Scrivendo il sistema in forma vettoriale $\mathbf{y}'(t) = \mathbf{F}(t, \mathbf{y}(t))$, l'estensione dei metodi precedentemente sviluppati nel caso di una singola equazione appare immediata. Ad esempio, il metodo

$$\mathbf{u}_{n+1} = \mathbf{u}_n + h(\vartheta \mathbf{F}(t_{n+1}, \mathbf{u}_{n+1}) + (1 - \vartheta) \mathbf{F}(t_n, \mathbf{u}_n)), \quad n \geq 0,$$

con $\mathbf{u}_0 = \mathbf{y}_0$, $0 \leq \vartheta \leq 1$, rappresenta la forma vettoriale del metodo di Eulero in avanti se $\vartheta = 0$, di Eulero all'indietro se $\vartheta = 1$ e di Crank-Nicolson se $\vartheta = 1/2$.

Esempio 7.7 (Biologia) Risolviamo il sistema delle equazioni di Lotka-Volterra (7.3) con il metodo di Eulero in avanti quando $C_1 = C_2 = 1$, $b_1 = b_2 = 0$ e $d_1 = d_2 = 1$. Per poter utilizzare il Programma 17 nel caso di un *sistema*, costruiamo una *function* **f** che precisi le componenti del vettore **F** e che salveremo nel file **f.m**. Per il sistema in esame abbiamo

```
function y = f(t,y)
C1=1; C2=1; d1=1; d2=1; b1=0; b2=0;
yy(1)=C1*y(1)*(1-b1*y(1)-d2*y(2)); % prima equazione
y(2)=-C2*y(2)*(1-b2*y(2)-d1*y(1)); % seconda equazione
y(1)=yy(1);
return
```

A questo punto, eseguiamo il Programma 17 con le seguenti istruzioni

```
[t,u]=feuler('f',[0,10],[2 2],2000);
plot(t,u); figure(2); plot(u(:,1),u(:,2));
```

Abbiamo quindi risolto il sistema di Lotka-Volterra sull'intervallo temporale $[0, 10]$ con un passo di integrazione $h = 0.005$.

Il grafico di sinistra di Figura 7.15 rappresenta l'evoluzione nel tempo delle due componenti della soluzione. Come si vede, esse mostrano un andamento periodico di periodo 2π . Il grafico di destra mostra invece la traiettoria uscente dal dato iniziale nel cosiddetto *piano delle fasi*, cioè in un piano cartesiano che ha come coordinate y_1 e y_2 . Appare evidente che la traiettoria si mantiene in una regione limitata di questo piano. Se provassimo a partire dal dato iniziale $(1.2, 1.2)$ troveremmo una traiettoria ancor più confinata che sembra mantenersi chiusa attorno al punto $(1, 1)$. Ciò è dovuto al fatto che il sistema ammette 2 punti di equilibrio (ovvero punti nei quali $y'_1 = 0$ e $y'_2 = 0$) e uno di essi è proprio $(1, 1)$.

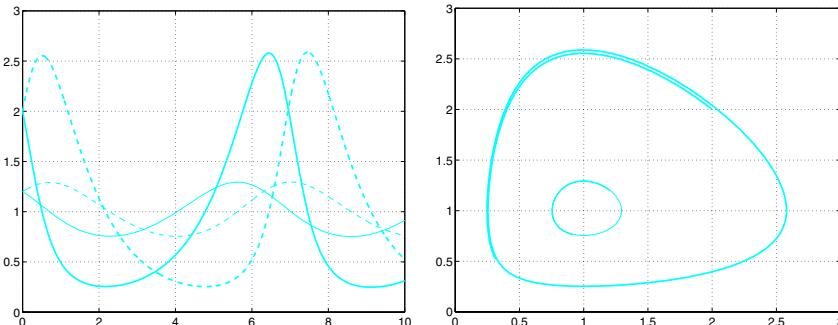


Figura 7.15. Soluzioni numeriche del sistema (7.3). A sinistra, rappresentazione in funzione del tempo dell'evoluzione delle due componenti della soluzione (y_1 in linea piena, y_2 in linea tratteggiata) per due diversi valori del dato iniziale ((2, 2) in linea più marcata, (1.2, 1.2) in linea sottile). A destra, le corrispondenti traiettorie nel piano delle fasi

Tali punti sono le soluzioni del sistema non lineare seguente

$$\begin{cases} y'_1 = y_1 - y_1 y_2 = 0, \\ y'_2 = -y_2 + y_2 y_1 = 0, \end{cases}$$

e sono pertanto $(0, 0)$ e $(1, 1)$. Per un dato iniziale pari ad una di queste copie di valori la soluzione si mantiene costante nel tempo. Il punto $(0, 0)$ è un punto di equilibrio instabile, mentre il punto $(1, 1)$ è stabile, ciò significa che traiettorie che partono da un dato iniziale vicino a tale punto si mantengono limitate nel piano delle fasi.

Qualora si utilizzi uno schema esplicito il passo di integrazione deve soddisfare una condizione di stabilità, analoga a quella incontrata nel paragrafo 7.5. Quando le parti reali degli autovalori λ_k della matrice Jacobiana $A(t) = [\partial \mathbf{F}/\partial \mathbf{y}](t, \mathbf{y})$ di \mathbf{F} sono tutte negative, possiamo porre $\lambda = -\max_t \rho(A(t))$, dove $\rho(A(t))$ è il raggio spettrale di $A(t)$. Questo λ è il naturale candidato a rimpiazzare quello che interviene nelle condizioni di stabilità (come ad esempio (7.30)) derivate per il problema di Cauchy scalare.

Osservazione 7.5 I programma MATLAB (`ode23`, `ode45`, ...) precedentemente ricordati, possono essere facilmente usati anche per risolvere sistemi di equazioni differenziali ordinarie. La sintassi da usare è `odeXX('f', [t0 tf], y0)`, dove y_0 è il vettore delle condizioni iniziali, f è una funzione specificata dall'utente e `odeXX` è uno dei metodi disponibili in MATLAB.

Consideriamo ora il caso di un'equazione differenziale di ordine m

$$y^{(m)}(t) = f(t, y, y', \dots, y^{(m-1)}) \quad (7.52)$$

con $t \in (t_0, T]$, la cui soluzione (quando esiste) è una famiglia di funzioni definite a meno di m costanti arbitrarie. Queste ultime possono essere

determinate imponendo m condizioni iniziali

$$y(t_0) = y_0, y'(t_0) = y_1, \dots y^{(m-1)}(t_0) = y_{m-1}.$$

Ponendo

$$w_1(t) = y(t), w_2(t) = y'(t), \dots w_m(t) = y^{(m-1)}(t),$$

la (7.52) può essere trasformata nel seguente sistema di m equazioni differenziali di ordine 1

$$w'_1(t) = w_2(t),$$

$$w'_2(t) = w_3(t),$$

$$\vdots$$

$$w'_{m-1}(t) = w_m(t),$$

$$w'_m(t) = f(t, w_1, \dots, w_m),$$

con condizioni iniziali

$$w_1(t_0) = y_0, w_2(t_0) = y_1, \dots w_m(t_0) = y_{m-1}.$$

La risoluzione numerica di un'equazione differenziale di ordine $m > 1$ è pertanto riconducibile alla risoluzione numerica di un sistema di m equazioni del prim'ordine.

Esempio 7.8 (Elettrotecnica) Consideriamo il circuito del Problema 7.4 con $L(i_1) = L$ costante e $R_1 = R_2 = R$. In tal caso v può essere determinato risolvendo il seguente sistema di due equazioni differenziali

$$\begin{cases} v'(t) = w(t), \\ w'(t) = -\frac{1}{LC} \left(\frac{L}{R} + RC \right) w(t) - \frac{2}{LC} v(t) + \frac{e}{LC}, \end{cases} \quad (7.53)$$

con condizioni iniziali $v(0) = 0$, $w(0) = 0$. Esso è stato ricavato a partire dalla seguente equazione differenziale di ordine 2

$$LC \frac{d^2v}{dt^2} + \left(\frac{L}{R_2} + R_1 C \right) \frac{dv}{dt} + \left(\frac{R_1}{R_2} + 1 \right) v = e. \quad (7.54)$$

Poniamo $L = 0.1$ Henry, $C = 10^{-3}$ Farad, $R = 10$ Ohm ed $e = 5$ Volt, dove Henry, Farad, Ohm e Volt sono rispettivamente le unità di misura di induttanza, capacità, resistenza e voltaggio. Applichiamo il metodo di Eulero in avanti con $h = 0.001$ secondi nell'intervallo temporale $[0, 0.1]$ tramite il Programma 17

```
>> [t,u]=feuler('fsys',[0,0.1],[0 0],100);
```

dove `fsys` è precisata nel file `fsys.m` come

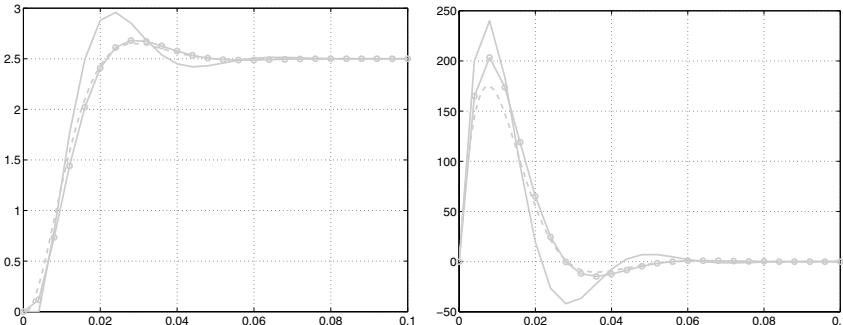


Figura 7.16. Soluzioni numeriche del sistema (7.53). A sinistra, riportiamo la differenza di potenziale $v(t)$ in funzione del tempo, a destra, la sua derivata: in tratteggio la soluzione ottenuta per $h = 0.001$ con il metodo di Eulero in avanti, in linea continua quella generata con lo stesso metodo con $h = 0.004$, con i pallini con quella prodotta con il metodo di Newmark (con $\theta = 1/2$ and $\zeta = 1/4$) per $h = 0.004$

```
function y=fsys(t,y)
L=0.1; C=1.e-03; R=10; e=5;
yy(1)=y(2);
y(2)=-(L/R+R*C)/(L*C)*y(2)-2/(L*C)*y(1)+e/(L*C);
y(1)=yy(1);
return
```

In Figura 7.16 vengono riportati i valori approssimati di $v(t)$ e $w(t)$. Come ci si aspetta, per t grandi $v(t)$ tende a $e/2 = 2.5$ Volt. Per questo problema la parte reale degli autovalori di $A(t) = [\partial \mathbf{F} / \partial \mathbf{y}](t, \mathbf{y})$ è negativa e λ può essere scelto pari a -141.4214 . Questo comporta la restrizione $h < 2/|\lambda| = 0.0282$ per garantire l'assoluta stabilità.

Si possono ottenere alcuni metodi di approssimazione per equazioni differenziali di ordine superiore al primo senza passare attraverso un sistema equivalente del prim'ordine. Consideriamo ad esempio il caso di un problema di Cauchy di ordine 2

$$\begin{cases} y''(t) = f(t, y(t), y'(t)) & t \in (t_0, T], \\ y(t_0) = \alpha_0, \quad y'(t_0) = \beta_0. \end{cases} \quad (7.55)$$

Si può costruire un metodo di approssimazione elementare nel modo seguente: trovare u_n per $1 \leq n \leq N_h$ tale che

$$\frac{u_{n+1} - 2u_n + u_{n-1}}{h^2} = f(t_n, u_n, v_n) \quad (7.56)$$

con $u_0 = \alpha_0$ e $v_0 = \beta_0$. Poiché $(y_{n+1} - 2y_n + y_{n-1})/h^2$ è un'approssimazione di ordine due di $y''(t_n)$, consideriamo una approssimazione di ordine 2 anche per $y'(t_k)$, ovvero

$$v_n = \frac{u_{n+1} - u_{n-1}}{2h}, \text{ with } v_0 = \beta_0. \quad (7.57)$$

Si ottiene il cosiddetto *metodo leap frog*, (7.56)-(7.57) che è un metodo accurato di ordine 2 rispetto a h .

Un metodo più generale è quello *di Newmark* in cui si costruiscono le due successioni

$$u_{n+1} = u_n + hv_n + h^2 [\zeta f(t_{n+1}, u_{n+1}, v_{n+1}) + (1/2 - \zeta) f(t_n, u_n, v_n)],$$

$$v_{n+1} = v_n + h [(1 - \theta) f(t_n, u_n, v_n) + \theta f(t_{n+1}, u_{n+1}, v_{n+1})],$$

con $u_0 = \alpha_0$ e $v_0 = \beta_0$; ζ e θ sono due numeri reali non negativi. Questo metodo è esplicito solo se $\zeta = \theta = 0$ ed è di ordine 2 se $\theta = 1/2$ (in caso contrario è di ordine 1).

La condizione $\theta \geq 1/2$ è necessaria per la stabilità. Inoltre, se $\theta = 1/2$ e $\zeta = 1/4$ si trova uno schema alquanto popolare, essendo incondizionatamente stabile. Tuttavia, questa scelta particolare di θ e ζ non è adatta per simulazioni di problemi definiti su grandi intervalli temporali in quanto introduce delle soluzioni spurie oscillanti. Per tali problemi è preferibile utilizzare $\theta > 1/2$ e $\zeta \geq (\theta + 1/2)^2/4$, sebbene l'ordine di accuratezza degradi a 1.

Nel Programma 22 implementiamo il metodo di Newmark. Il vettore **parameters** serve a precisare nelle sue due componenti i valori dei coefficienti del metodo: **parameters(1)= ζ** e **parameters(2)= θ** .

Programma 22 - newmark : il metodo di Newmark



```

function [tt,u]=newmark(odefun,tspan,y,Nh,parameters,varargin)
%NEWMARK Risolve un'equazione differenziale del second'ordine
% [T,Y]=NEWMARK(ODEFUN,TSPAN,Y0,NH,PARAMETERS) con TSPAN =
% [T0 TFINAL] integra l'equazione y'' = f(t,y,y') dal tempo
% T0 a TFINAL con condizioni iniziali Y0=(y(t0),y'(t0)) usando il metodo
% di Newmark
% su una griglia equispaziata di NH intervalli. La funzione ODEFUN(T,Y)
% deve restituire una quantita' scalare corrispondente a f(t,y,y').
%
zeta = parameters(1);
theta = parameters(2);
h=(tspan(2)-tspan(1))/Nh;
tt=linspace(tspan(1),tspan(2),Nh+1);
u(1,:)=y;
fid=fopen('myfun.m','w');
fprintf(fid,'function z=myfun(w,t,y,h,fn,zeta,theta,odefun,varargin)\n');
fprintf(fid,'fn1 = feval(odefun,t,w,varargin{:});\n');
fprintf(fid,'z = w-y-h*[y(1,2), (1-theta)*fn+theta*fn1]-h^2*[zeta*fn1+(0.5-zeta)*fn,0];\n');
fprintf(fid,'return\n');
fclose(fid);
options=optimset;
```

```

options.TolFun=1.e-12;
options.MaxFunEvals=10000;

fn =feval(odefun,tt(1),u(1,:),varargin{:});
for t=tt(2:end)
    w = fsolve(@(w) myfun(w,t,y,h,fn,zeta,theta,odefun,varargin{:}),y,options);
    fn =feval(odefun,t,w,varargin{:});
    u = [u; w];
    y = w;
end
t=tt;
return

```

Esempio 7.9 (Elettrotecnica) Consideriamo nuovamente il circuito del Problema 7.4 e risolviamo l'equazione del second'ordine (7.54) con lo schema di Newmark. In Figura 7.16 confrontiamo l'approssimazione della funzione $v(T)$ calcolata usando lo schema di Eulero (per $h = 0.001$ in linea tratteggiata e per $h = 0.04$ in linea continua) ed il metodo di Newmark per $\theta = 1/2$ e $\zeta = 1/4$ (linea con i cerchietti), e passo di discretizzazione $h = 0.04$. La miglior accuratezza di quest'ultima approssimazione è dovuta al fatto che il metodo (7.56)-(7.57) è accurato di ordine 2 rispetto a h .



Si vedano gli Esercizi 7.18-7.20.

7.9 Alcuni esempi

7.9.1 Il pendolo sferico

Il moto di un punto $\mathbf{x}(t) = (x_1(t), x_2(t), x_3(t))^T$ di massa m soggetto alla forza di gravità $\mathbf{F} = (0, 0, -gm)^T$ (con $g = 9.8 \text{ m/s}^2$) e vincolato a muoversi sulla superficie sferica di equazione $\Phi(\mathbf{x}) = x_1^2 + x_2^2 + x_3^2 - 1 = 0$ è descritto dal seguente sistema di equazioni differenziali ordinarie

$$\ddot{\mathbf{x}} = \frac{1}{m} \left(\mathbf{F} - \frac{m \dot{\mathbf{x}}^T H \dot{\mathbf{x}} + \nabla \Phi^T \mathbf{F}}{|\nabla \Phi|^2} \nabla \Phi \right) \text{ per } t > 0. \quad (7.58)$$

Abbiamo indicato con $\dot{\mathbf{x}}$ la derivata prima rispetto a t , con $\ddot{\mathbf{x}}$ la derivata seconda, con $\nabla \Phi$ il gradiente di Φ , pari a $2\mathbf{x}^T$, con H la matrice Hessiana di Φ le cui componenti sono $H_{ij} = \partial^2 \Phi / \partial x_i \partial x_j$ per $i, j = 1, 2, 3$. Nel nostro caso H è una matrice diagonale di coefficienti pari a 2. Al sistema (7.58) dobbiamo aggiungere le condizioni iniziali $\mathbf{x}(0) = \mathbf{x}_0$ e $\dot{\mathbf{x}}(0) = \mathbf{v}_0$.

Al fine di risolvere numericamente (7.58) trasformiamolo in un sistema di equazioni differenziali di ordine 1 nella nuova incognita \mathbf{y} , un vettore di 6 componenti. Posto $y_i = x_i$ e $y_{i+3} = \dot{x}_i$ per $i = 1, 2, 3$, e

$$\lambda = \frac{m(y_4, y_5, y_6)^T H(y_4, y_5, y_6) + \nabla \Phi^T \mathbf{F}}{|\nabla \Phi|^2}.$$

otteniamo

$$\begin{aligned}\dot{y}_i &= y_{3+i}, \quad i = 1, 2, 3, \\ \dot{y}_{3+i} &= \frac{1}{m} \left(F_i - \lambda \frac{\partial \Phi}{\partial y_i} \right), \quad i = 1, 2, 3.\end{aligned}\tag{7.59}$$

Applichiamo i metodi di Eulero e di Crank-Nicolson. Dapprima è necessario definire una *function* MATLAB (la `fvinc` del Programma 23) che restituisca le espressioni dei termini di destra delle equazioni del sistema (7.59). Supponiamo inoltre che le condizioni iniziali siano date dal vettore `y0=[0,1,0,.8,0,1.2]` e che l'intervallo di integrazione sia `tspan=[0,25]`. Richiamiamo il metodo di Eulero esplicito nel modo seguente

```
[t,y]=feuler(@fvinc,tspan,y0,nt);
```

(ed analogamente i metodi di Eulero all'indietro `beuler` e di Crank-Nicolson `cranknic`), dove `nt` è il numero di intervalli (di ampiezza costante) impiegati per la discretizzazione dell'intervallo `[tspan(1), tspan(2)]`. Nei grafici di Figura 7.17 riportiamo le traiettorie ottenute con 10000 e 100000 nodi di discretizzazione: come si vede solo nel secondo caso la soluzione è ragionevolmente accurata. In effetti, pur non conoscendo la soluzione esatta del problema, possiamo avere un'idea dell'accuratezza osservando che la soluzione esatta soddisfa $r(\mathbf{y}) \equiv y_1^2 + y_2^2 + y_3^2 - 1 = 0$ e misurando quindi il massimo valore del residuo $r(\mathbf{y}_n)$ al variare di n , essendo \mathbf{y}_n l'approssimazione della soluzione esatta generata al tempo t_n . Usando 10000 nodi di discretizzazione troviamo $r = 1.0578$, mentre con 100000 nodi si ha $r = 0.1111$, in accordo con la teoria che vuole il metodo di Eulero esplicito convergente di ordine 1.

Utilizzando il metodo di Eulero implicito con 20000 passi troviamo la soluzione riportata in Figura 7.18, mentre il metodo di Crank-Nicolson (di ordine 2) con soli 2000 passi fornisce la soluzione riportata nella stessa figura a destra, decisamente più accurata. Troviamo infatti $r = 0.5816$ per il metodo di Eulero隐式 e $r = 0.0966$ per quello di Crank-Nicolson.

Per confronto, risolviamo lo stesso problema con i metodi esplicativi adattivi di tipo Runge-Kutta `ode23` e `ode45`, presenti in MATLAB. Essi (a meno di indicazioni diverse) modificano il passo di integrazione in modo da garantire che l'errore relativo sulla soluzione sia minore di 10^{-3} e quello assoluto minore di 10^{-6} . Li lanciamo con i seguenti comandi

```
[t,y]=ode23(@fvinc,tspan,y0);
[t,y]=ode45(@fvinc,tspan,y0);
```

ed otteniamo le soluzioni presentate in Figura 7.19.

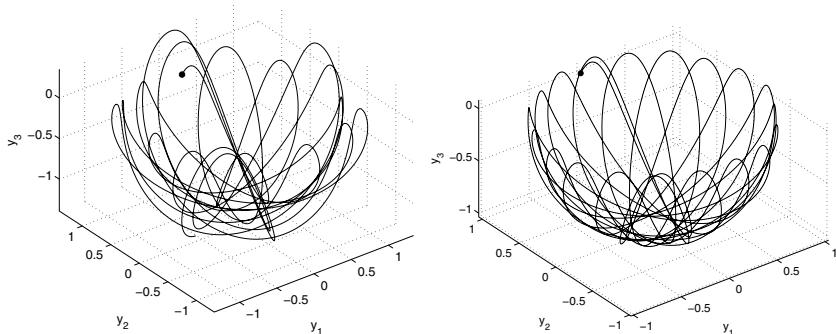


Figura 7.17. Le traiettorie ottenute con il metodo di Eulero esplicito con $h = 0.0025$ a sinistra e $h = 0.00025$ a destra. Il punto annerito indica il dato iniziale

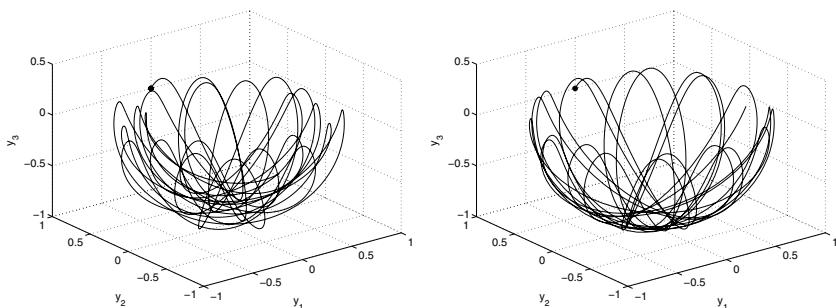


Figura 7.18. Le traiettorie ottenute con il metodo di Eulero implicito con $h = 0.00125$ a sinistra e con il metodo di Crank-Nicolson con $h = 0.025$ a destra

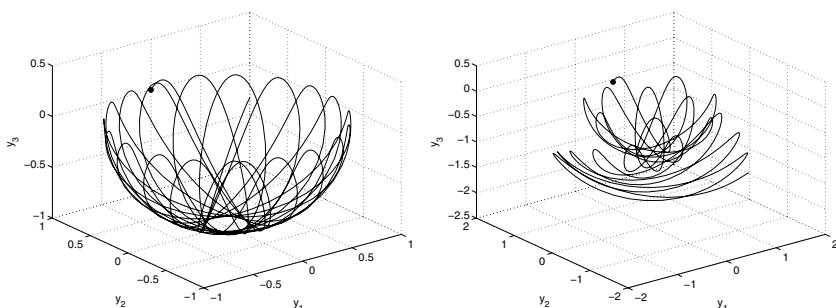


Figura 7.19. Le traiettorie ottenute con i metodi `ode23` (a sinistra) e `ode45` (a destra) con gli stessi criteri di accuratezza. Nel secondo caso il controllo sull'errore fallisce e la soluzione che si trova è meno accurata

I due metodi hanno usato 783 e 537 nodi di discretizzazione, rispettivamente, distribuiti in modo non uniforme. Il residuo r vale 0.0238 per `ode23` e 3.2563 per `ode45`. Sorprendentemente, il risultato ottenuto con il metodo di ordine più elevato è dunque meno accurato e questo ci deve mettere in guardia quando facciamo uso dei programmi della famiglia `ode` disponibili in MATLAB. Una spiegazione di questo comportamento risiede nel fatto che lo stimatore dell'errore implementato in `ode45` è meno stringente di quello presente in `ode23`. Diminuendo di poco la tolleranza relativa (basta porre `options=odeset('RelTol',1.e-04)`) e richiamando il programma come `[t,y]=ode45(@fvinc,tspan,y0,options);` si trovano infatti risultati confrontabili.

Programma 23 - `fvinc` : termine forzante per il problema del pendolo sferico



```
function [f]=fvinc(t,y)
[n,m]=size(y); phix='2*y(1)'; phiy='2*y(2)'; phiz='2*y(3)';
H=2*eye(3);
%massa
massa=1;
% F=forza peso
F1='0*y(1)'; F2='0*y(2)'; F3=-massa*9.8';
%
f=zeros(n,m);
xpunto=zeros(3,1);
xpunto(1:3)=y(4:6);
F=[eval(F1);eval(F2);eval(F3)];
G=[eval(phix);eval(phiy);eval(phiz)];
lambda=(m*xpunto'*H*xpunto+F'*G)/(G'*G);
f(1:3)=y(4:6);
f(4)=(F(1)-lambda*G(1))/massa;
f(5)=(F(2)-lambda*G(2))/massa;
f(6)=(F(3)-lambda*G(3))/massa;
return
```

7.9.2 Il problema dei tre corpi

Vogliamo calcolare l'evoluzione di un sistema costituito da tre oggetti, note le loro posizioni e velocità iniziali e le loro masse sotto l'influenza della loro reciproca attrazione gravitazionale. Il problema si formula utilizzando le leggi del moto di Newton. Tuttavia, a differenza del caso di due corpi, non si conoscono soluzioni in forma chiusa. Noi supponiamo che uno dei tre corpi abbia massa decisamente superiore a quella degli altri due, in particolare studiamo il caso del sistema Sole-Terra-Marte, un problema studiato da matematici illustri quali Lagrange nel XVIII secolo, Poincaré verso la fine del XIX secolo e dall'italiano Tullio Levi-Civita nel secolo XX.

Denotiamo allora con M_s la massa del Sole, M_t quella della Terra e M_m quella di Marte. Essendo la massa del Sole circa 330000 volte quella della Terra e quella di Marte circa un decimo di quella terrestre, possiamo immaginare che il baricentro dei tre corpi sia pressoché coincidente con il centro del Sole (che resterà pertanto immobile in questo modello) e che i tre oggetti si mantengano nel piano determinato dalle loro posizioni iniziali. In tal caso la forza complessiva agente ad esempio sulla Terra sarà pari a

$$\mathbf{F}_t = \mathbf{F}_{ts} + \mathbf{F}_{tm} = M_t \frac{d^2 \mathbf{x}_t}{dt^2}, \quad (7.60)$$

dove $\mathbf{x}_t = (x_t, y_t)^T$ denota la posizione della Terra, mentre \mathbf{F}_{ts} e \mathbf{F}_{tm} denotano rispettivamente la forza esercitata dal Sole e da Marte sulla Terra. Utilizzando la legge di gravitazione universale la (7.60) diventa (\mathbf{x}_m denota la posizione di Marte)

$$M_t \frac{d^2 \mathbf{x}_t}{dt^2} = -GM_t M_s \frac{\mathbf{x}_t}{|\mathbf{x}_t|^3} + GM_t M_m \frac{\mathbf{x}_m - \mathbf{x}_t}{|\mathbf{x}_m - \mathbf{x}_t|^3}.$$

Adimensionalizzando le equazioni e scalando le lunghezze rispetto alla lunghezza del semi-asse maggiore dell'orbita della Terra, si perviene alla seguente equazione

$$M_t \frac{d^2 \mathbf{x}_t}{dt^2} = 4\pi^2 \left(\frac{M_m}{M_s} \frac{\mathbf{x}_m - \mathbf{x}_t}{|\mathbf{x}_m - \mathbf{x}_t|^3} - \frac{\mathbf{x}_t}{|\mathbf{x}_t|^3} \right). \quad (7.61)$$

Con calcoli simili si perviene all'equazione analoga per il pianeta Marte

$$M_m \frac{d^2 \mathbf{x}_m}{dt^2} = 4\pi^2 \left(\frac{M_t}{M_s} \frac{\mathbf{x}_t - \mathbf{x}_m}{|\mathbf{x}_t - \mathbf{x}_m|^3} - \frac{\mathbf{x}_m}{|\mathbf{x}_m|^3} \right). \quad (7.62)$$

Il sistema del second'ordine (7.61)-(7.62) si riduce immediatamente ad un sistema di 8 equazioni di ordine 1. Il Programma 24 consente la valutazione di una *function* contenente i termini di destra del sistema (7.61)-(7.62).



Programma 24 - threebody : termine forzante per il problema dei tre corpi semplificato

```
function f=threebody(t,y)
Ms=330000;
Me=1;
Mm=0.1;
D1 = ((y(5)-y(1))^2+(y(7)-y(3))^2)^(3/2);
D2 = (y(1)^2+y(3)^2)^(3/2);
f(1)=y(2);
f(2)=4*pi^2*(Me/Ms*(y(5)-y(1))/D1-y(1)/D2);
f(3)=y(4);
```

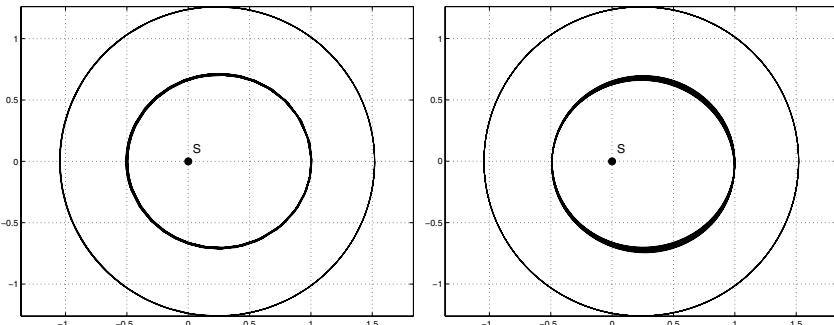


Figura 7.20. L'orbita della Terra (più interna) e quella di Marte rispetto al Sole calcolate con il metodo adattivo `ode23` a sinistra (con 564 passi) e con quello di Crank-Nicolson, a destra (con 2000 passi)

```
f(4)=4*pi^2*(Me/Ms*(y(7)-y(3))/D1-y(3)/D2);
D2 = (y(5)^2+y(7)^2)^(3/2);
f(5)=y(6);
f(6)=4*pi^2*(Mm/Ms*(y(1)-y(5))/D1-y(5)/D2);
f(7)=y(8);
f(8)=4*pi^2*(Mm/Ms*(y(3)-y(7))/D1-y(7)/D2);
return
```

Confrontiamo fra loro il metodo di Crank-Nicolson (implicito) ed il metodo adattivo Runge-Kutta implementato in `ode23` (esplicito). Avendo posto che la Terra stia a 1 unità dal Sole, Marte si troverà a circa 1.52 unità: la posizione iniziale sarà dunque $(1, 0)$ per la Terra e $(1.52, 0)$ per Marte. Supponiamo inoltre che i due pianeti abbiano al tempo iniziale velocità orizzontale nulla e velocità verticale pari rispettivamente a -5.1 unità per la Terra e -4.6 unità per Marte: in tal modo dovrebbero percorrere orbite ragionevolmente stabili attorno al Sole. Per il metodo di Crank-Nicolson sceglieremo 2000 passi di discretizzazione.

```
[t23,u23]=ode23(@threebody,[0 10],[1.52 0 0 -4.6 1 0 0 -5.1]);
[tcn,ucn]=cranknic(@threebody,[0 10],[1.52 0 0 -4.6 1 0 0 -5.1],2000);
```

I grafici di Figura 7.20 mostrano che i due metodi sono entrambi in grado di riprodurre le orbite ellittiche dei due pianeti attorno al Sole. Il metodo `ode23` ha richiesto solo 564 passi (non uniformi) per generare una soluzione più accurata di quella generata da un metodo implicito dello stesso ordine di accuratezza, ma che non usa adattività del passo.

7.9.3 Alcuni problemi stiff

Consideriamo il seguente problema differenziale, proposto da [Gea71], come variante del problema modello (7.28):

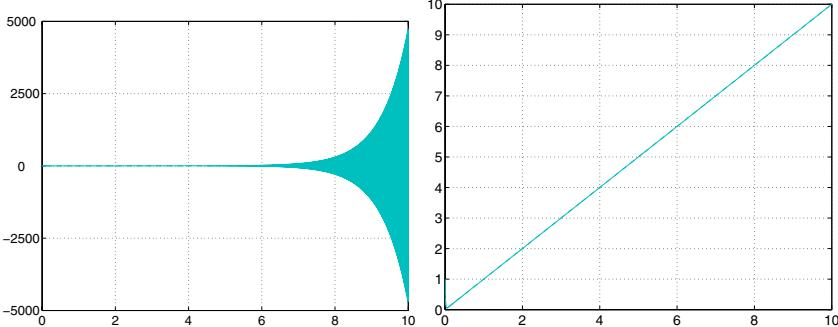


Figura 7.21. Le soluzioni ottenute con il metodo (7.46) per il problema (7.63) violando la condizione di stabilità ($h = 0.0055$, a sinistra) e rispettandola ($h = 0.0054$, a destra)

$$\begin{cases} y'(t) = \lambda(y(t) - g(t)) + g'(t), & t > 0, \\ y(0) = y_0, \end{cases} \quad (7.63)$$

dove g è una funzione regolare e $\lambda \ll 0$, la cui soluzione è

$$y(t) = (y_0 - g(0))e^{\lambda t} + g(t), \quad t \geq 0. \quad (7.64)$$

Essa presenta due componenti, $(y_0 - g(0))e^{\lambda t}$ e $g(t)$, la prima trascurabile rispetto alla seconda per t sufficientemente grande. Poniamo in particolare $g(t) = t$, $\lambda = -100$ e risolviamo il problema (7.63) sull'intervallo $(0, 100)$ con il metodo di Eulero esplicito: essendo in questo caso $f(t, y) = \lambda(y(t) - g(t)) + g'(t)$ abbiamo $\partial f / \partial y = \lambda$, e l'analisi di stabilità condotta nel paragrafo 7.4 suggerisce di scegliere $h < 2/100$. Questa restrizione è dettata dalla presenza della componente che si comporta come e^{-100t} ed appare del tutto ingiustificata se si pensa al peso che essa ha rispetto all'intera soluzione (per avere un'idea, per $t = 1$ abbiamo $e^{-100} \approx 10^{-44}$). La situazione peggiora usando un metodo esplicito di ordine superiore, come ad esempio il metodo di Adams-Bashforth (7.46) di ordine 3: la regione di assoluta stabilità si riduce (si veda la Figura 7.12) e, conseguentemente, la restrizione su h diventa ancora più severa, $h < 0.00545$. Violare anche di poco questa restrizione produce soluzioni del tutto inaccettabili (come mostrato in Figura 7.21 a sinistra).

Ci troviamo dunque di fronte ad un problema apparentemente semplice, ma che risulta impegnativo da risolvere con un metodo esplicito (e più in generale con un metodo che non sia A-stabile) per la presenza di due componenti nella soluzione dal comportamento radicalmente diverso per t che tende all'infinito: un problema di questo genere si dice *stiff*.

Più precisamente, diciamo che un sistema di equazioni differenziali lineari della forma

$$\mathbf{y}'(t) = A\mathbf{y}(t) + \boldsymbol{\varphi}(t), \quad A \in \mathbb{R}^{n \times n}, \quad \boldsymbol{\varphi}(t) \in \mathbb{R}^n, \quad (7.65)$$

in cui A abbia n autovalori distinti λ_j , $j = 1, \dots, n$ con $\operatorname{Re}(\lambda_j) < 0$, $j = 1, \dots, n$ è stiff se

$$r_s = \frac{\max_j |\operatorname{Re}(\lambda_j)|}{\min_j |\operatorname{Re}(\lambda_j)|} \gg 1.$$

La soluzione esatta di (7.65) è

$$\mathbf{y}(t) = \sum_{j=1}^n C_j e^{\lambda_j t} \mathbf{v}_j + \boldsymbol{\psi}(t), \quad (7.66)$$

dove C_1, \dots, C_n sono n costanti e $\{\mathbf{v}_j\}$ è una base formata dagli autovettori di A , mentre $\boldsymbol{\psi}(t)$ è una soluzione particolare dell'equazione differenziale. Se $r_s \gg 1$ assistiamo nuovamente all'insorgere di componenti della soluzione \mathbf{y} che tendono a zero con velocità differenti. La componente che tende a zero più rapidamente per t che tende all'infinito (quella relativa all'autovalore di modulo massimo) sarà quella che comporterà la restrizione più pesante sul passo di integrazione, a meno naturalmente di impiegare un metodo incondizionatamente assolutamente stabile.

Esempio 7.10 Consideriamo il sistema $\mathbf{y}' = Ay$ per $t \in (0, 100)$ con condizione iniziale $\mathbf{y}(0) = \mathbf{y}_0$, dove $\mathbf{y} = (y_1, y_2)^T$, $\mathbf{y}_0 = (y_{1,0}, y_{2,0})^T$ e

$$A = \begin{bmatrix} 0 & 1 \\ -\lambda_1 \lambda_2 & \lambda_1 + \lambda_2 \end{bmatrix},$$

dove λ_1 e λ_2 sono due numeri negativi distinti con $|\lambda_1| \gg |\lambda_2|$. La matrice A ha come autovalori λ_1 e λ_2 ed autovettori $\mathbf{v}_1 = (1, \lambda_1)^T$, $\mathbf{v}_2 = (1, \lambda_2)^T$. Grazie alla (7.66) la soluzione del sistema è pari a

$$\mathbf{y}(t) = \begin{pmatrix} C_1 e^{\lambda_1 t} + C_2 e^{\lambda_2 t} \\ C_1 \lambda_1 e^{\lambda_1 t} + C_2 \lambda_2 e^{\lambda_2 t} \end{pmatrix}^T. \quad (7.67)$$

Le costanti C_1 e C_2 si ottengono imponendo la condizione iniziale:

$$C_1 = \frac{\lambda_2 y_{1,0} - y_{2,0}}{\lambda_2 - \lambda_1}, \quad C_2 = \frac{y_{2,0} - \lambda_1 y_{1,0}}{\lambda_2 - \lambda_1}.$$

Per le considerazioni svolte in precedenza il passo di integrazione di un metodo esplicito usato per la risoluzione di tale sistema dipenderà esclusivamente dall'autovalore di modulo massimo, λ_1 . Rendiamocene conto sperimentalmente usando il metodo di Eulero esplicito e scegliendo $\lambda_1 = -100$, $\lambda_2 = -1$, $y_{1,0} = y_{2,0} = 1$. In Figura 7.22 riportiamo le soluzioni calcolate violando (a sinistra) o rispettando (a destra) la condizione di stabilità $h < 1/50$.

La definizione di problema stiff può essere estesa, seppur con qualche attenzione, al caso non lineare (si veda ad esempio [QSS04, Capitolo 11]). Uno dei problemi *stiff* non lineari più studiati è dato dall'equazione di Van der Pol

$$\frac{d^2x}{dt^2} = \mu(1 - x^2) \frac{dx}{dt} - x, \quad (7.68)$$

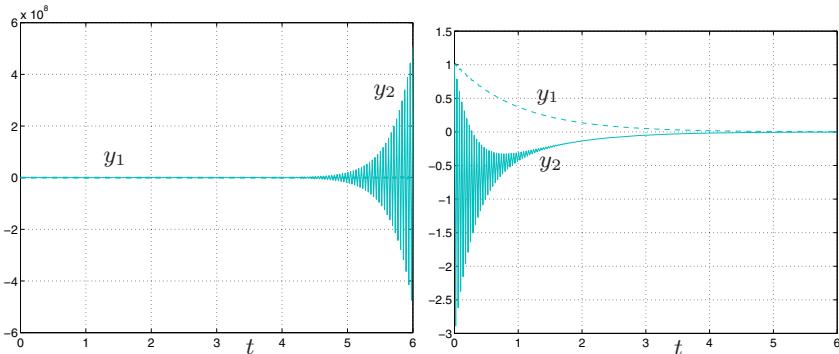


Figura 7.22. Le soluzioni calcolate per il problema dell’Esempio 7.10 per $h = 0.0207$ (a sinistra) e $h = 0.0194$ (a destra). Nel primo caso la condizione $h < 2/|\lambda_1| = 0.02$ è violata ed il metodo è instabile. Si tenga conto della scala completamente diversa dei due grafici

proposta nel 1920 ed utilizzata nello studio di circuiti che contengano valvole termoioniche, i cosiddetti tubi a vuoto, come il tubo catodico del televisore o il magnetron nei forni a micro-onde.

Se si pone $\mathbf{y} = (x, y)^T$, la (7.68) è equivalente al seguente sistema non lineare del prim’ordine

$$\mathbf{y}' = \begin{bmatrix} 0 & 1 \\ -1 & \mu(1-x^2) \end{bmatrix} \mathbf{y}. \quad (7.69)$$

Tale sistema diventa sempre più stiff quanto più cresce il parametro μ . Nella soluzione compaiono infatti due componenti che al crescere di μ presentano dinamiche completamente diverse. Quella con la dinamica più veloce detta una limitazione tanto più proibitiva sul passo di integrazione tanto maggiore è il valore assunto da μ .

Se risolviamo (7.68) usando `ode23` e `ode45`, ci rendiamo conto che essi sono troppo onerosi quando μ è grande. Con $\mu = 100$ e condizione iniziale $\mathbf{y} = (1, 1)^T$, `ode23` richiede 7835 passi e `ode45` 23473 passi per integrare fra $t = 0$ e $t = 100$. Leggendo l’*help* di MATLAB scopriamo che questi metodi non sono consigliati per problemi *stiff*: per essi vengono indicate altre procedure, come ad esempio i metodi impliciti `ode23s` o `ode15s`. La differenza in termini di numero di passi è notevole, come indicato in Tabella 7.1. Si osservi tuttavia che il numero di passi di `ode23s` è inferiore a quello di `ode23` solo per valori di μ sufficientemente grandi (e dunque per problemi molto stiff).

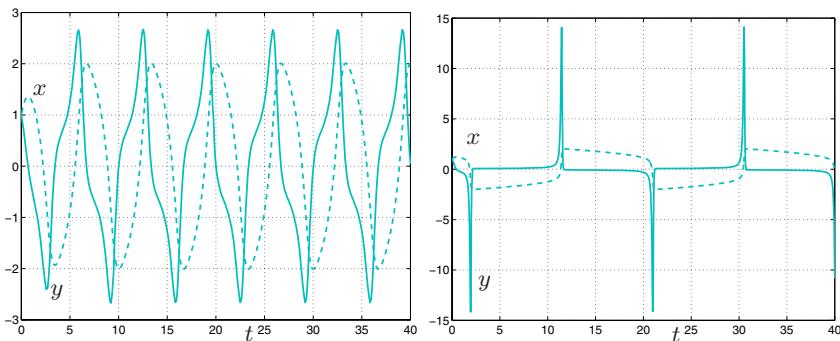


Figura 7.23. Andamento delle componenti della soluzione \mathbf{y} del sistema (7.69) per $\mu = 1$ (a sinistra) e $\mu = 10$ (a destra)

μ	ode23	ode45	ode23s	ode15s
0.1	471	509	614	586
1	775	1065	838	975
10	1220	2809	1005	1077
100	7835	23473	299	305
1000	112823	342265	183	220

Tabella 7.1. Andamento del numero di passi di integrazione per vari metodi di approssimazione al crescere del parametro μ

7.10 Cosa non vi abbiamo detto

Per una completa derivazione dell'intera famiglia dei metodi Runge-Kutta rimandiamo a [But87], [Lam91] e [QSS04, Capitolo 11].

Per la derivazione e l'analisi dei metodi multistep rimandiamo a [Arn73] e [Lam91].

7.11 Esercizi

Esercizio 7.1 Si applichino il metodo di Eulero in avanti e di Eulero all'indietro per la risoluzione del seguente problema di Cauchy

$$y' = \sin(t) + y, \quad t \in (0, 1], \quad \text{con } y(0) = 0, \quad (7.70)$$

e se ne verifichi la convergenza lineare.

Esercizio 7.2 Si consideri il problema di Cauchy

$$y' = -te^{-y}, \quad t \in (0, 1], \quad \text{con } y(0) = 0. \quad (7.71)$$

Lo si risolva con il metodo di Eulero esplicito con $h = 1/100$ e si stimi il numero di cifre significative corrette della soluzione approssimata per $t = 1$, sapendo che la soluzione esatta si mantiene limitata fra -1 e 0 .

Esercizio 7.3 Il metodo di Eulero implicito applicato al problema (7.71) richiede, ad ogni passo, la risoluzione dell'equazione non lineare: $u_{n+1} = u_n - ht_{n+1}e^{-u_{n+1}} = \phi(u_{n+1})$. La soluzione u_{n+1} può essere calcolata attraverso la seguente procedura di punto fisso:

per $k = 0, 1, \dots$, calcolare $u_{n+1}^{(k+1)} = \phi(u_{n+1}^{(k)})$, con $u_{n+1}^{(0)} = u_n$.

Si determinino eventuali restrizioni su h affinché tale metodo converga.

Esercizio 7.4 Si applichi il metodo di Crank-Nicolson per la risoluzione di (7.70) e si verifichi la convergenza di ordine 2.

Esercizio 7.5 Si verifichi che il metodo di Crank-Nicolson può essere ottenuto a partire dalla seguente forma integrale del problema di Cauchy (7.5)

$$y(t) - y_0 = \int_{t_0}^t f(\tau, y(\tau))d\tau,$$

approssimando l'integrale con la formula del trapezio (4.19).

Esercizio 7.6 Si risolva il problema modello (7.28) con $\lambda = -1 + i$ con il metodo di Eulero in avanti. Per quali valori di h il metodo è assolutamente stabile?

Esercizio 7.7 Si mostri che il metodo di Heun definito in (7.51) è consistente. Si scriva un programma MATLAB che lo implementa e si verifichi sperimentalmente l'ordine 2 di accuratezza rispetto a h , risolvendo il problema di Cauchy (7.70).

Esercizio 7.8 Si mostri che il metodo (7.51) è assolutamente stabile se $-2 \leq h\lambda \leq 0$ per λ reale e negativo.

Esercizio 7.9 Si dimostra la formula (7.33).

Esercizio 7.10 Si dimostri la disuguaglianza (7.38).

Esercizio 7.11 Si dimostri la disuguaglianza (7.39).

Esercizio 7.12 Si verifichi che il metodo Runge-Kutta esplicito di ordine 3 (7.45) è consistente. Si scriva un programma MATLAB che lo implementa e si verifichi sperimentalmente l'ordine 3 di accuratezza rispetto a h , risolvendo il problema di Cauchy (7.70). I metodi (7.51) e (7.45) sono alla base del programma MATLAB `ode23` per la risoluzione di equazioni differenziali ordinarie.

Esercizio 7.13 Si mostri che il metodo (7.45) è assolutamente stabile se $-2.5 \leq h\lambda \leq 0$ per λ reale e negativo.

Esercizio 7.14 Sotto quali condizioni su h il seguente metodo (detto di *Eulero modificato*)

$$u_{n+1}^* = u_n + hf(t_n, u_n), \quad u_{n+1} = u_n + hf(t_{n+1}, u_{n+1}^*) \quad (7.72)$$

è assolutamente stabile?

Esercizio 7.15 (Termodinamica) Si risolva l'equazione (7.1) con i metodi di Crank-Nicolson e con il metodo di Heun quando il corpo in esame è un cubo di lato 1 m e massa pari a 1 Kg. Si supponga $T_0 = 180K$, $T_e = 200K$, $\gamma = 0.5$ e $C = 100J/(Kg/K)$. Si confrontino i risultati ottenuti usando $h = 20$ e $h = 10$ per $t \in [0, 200]$ secondi.

Esercizio 7.16 Si individui, tramite MATLAB, la regione di assoluta stabilità del metodo di Heun.

Esercizio 7.17 Si risolva il problema di Cauchy (7.16) con il metodo di Heun e se ne verifichi l'ordine.

Esercizio 7.18 Lo spostamento $x(t)$ di un sistema vibrante costituito da un corpo di un certo peso, da una molla e soggetto ad una forza resistiva proporzionale alla velocità è descritto dall'equazione differenziale $x'' + 5x' + 6x = 0$. La si risolva con il metodo di Heun, supponendo $x(0) = 1$ e $x'(0) = 0$ e $t \in [0, 5]$.

Esercizio 7.19 Le equazioni che descrivono il moto di un pendolo di Foucault in assenza di attrito sono

$$x'' - 2\omega \sin(\Psi)y' + k^2x = 0, \quad y'' + 2\omega \cos(\Psi)x' + k^2y = 0,$$

dove Ψ è la latitudine del luogo in cui si trova il pendolo, $\omega = 7.29 \cdot 10^{-5}$ sec $^{-1}$ è la velocità angolare della Terra, $k = \sqrt{g/l}$ con $g = 9.8$ m/sec 2 e l la lunghezza del pendolo. Si calcolino numericamente $x = x(t)$ e $y = y(t)$, per $t \in [0, 300]$ secondi, con il metodo di Eulero esplicito per $\Psi = \pi/4$.

Esercizio 7.20 (Sport) Si risolva con `ode23` il Problema 7.3 supponendo che la velocità iniziale della palla sia $\mathbf{v}(0) = v_0(\cos(\theta), 0, \sin(\theta))^T$ con $v_0 = 38$ m/s e θ uguale ad 1 grado e con una velocità angolare $180 \cdot 1.047198$ radianti al secondo. Se $\mathbf{x}(0) = \mathbf{0}^T$ approssimativamente dopo quanti secondi la palla raggiunge terra (la quota $z = 0$)?

Metodi numerici per problemi ai limiti

I problemi ai limiti (o ai valori al contorno) sono problemi differenziali definiti su un intervallo (a, b) della retta reale o in un aperto multidimensionale $\Omega \subset \mathbb{R}^d$ ($d = 2, 3$) per i quali il valore della soluzione incognita (o delle sue derivate) è assegnato agli estremi a e b dell'intervallo o sul bordo $\partial\Omega$ della regione multidimensionale.

Nel caso multidimensionale l'equazione differenziale coinvolge *derivate parziali* della soluzione esatta rispetto alle coordinate spaziali.

Si parla di problemi ai limiti ed ai valori iniziali quando, oltre alle derivate parziali rispetto alle variabili spaziali, compaiono anche derivate rispetto alla variabile temporale (indicata con t). In tali casi bisogna assegnare una (o più) condizioni iniziali al tempo $t = 0$.

Nel seguito, riportiamo alcuni esempi di problemi ai limiti ed ai valori iniziali:

1. l'*equazione di Poisson*:

$$-u''(x) = f(x), \quad x \in (a, b), \quad (8.1)$$

o (in più dimensioni)

$$-\Delta u(\mathbf{x}) = f(\mathbf{x}), \quad \mathbf{x} = (x_1, \dots, x_d)^T \in \Omega, \quad (8.2)$$

dove f è una funzione assegnata e Δ è il cosiddetto *operatore di Laplace*:

$$\Delta u = \sum_{i=1}^d \frac{\partial^2 u}{\partial x_i^2}.$$

Il simbolo $\partial \cdot / \partial x_i$ denota la derivata parziale rispetto alla variabile x_i , cioè per ogni punto \mathbf{x}^0

$$\frac{\partial u}{\partial x_i}(\mathbf{x}^0) = \lim_{h \rightarrow 0} \frac{u(\mathbf{x}^0 + h\mathbf{e}_i) - u(\mathbf{x}^0)}{h}, \quad (8.3)$$

dove \mathbf{e}_i è l' i -esimo vettore unitario di \mathbb{R}^d ;

2. l'*equazione del calore*:

$$\frac{\partial u(x, t)}{\partial t} - \mu \frac{\partial^2 u(x, t)}{\partial x^2} = f(x, t), \quad x \in (a, b), \quad t > 0,$$

o (in più dimensioni)

$$\frac{\partial u(\mathbf{x}, t)}{\partial t} - \mu \Delta u(\mathbf{x}, t) = f(\mathbf{x}, t), \quad \mathbf{x} \in \Omega, \quad t > 0,$$

dove $\mu > 0$ è un coefficiente assegnato che rappresenta la conducibilità termica e f è nuovamente una funzione assegnata;

3. l'*equazione delle onde*:

$$\frac{\partial^2 u(x, t)}{\partial t^2} - c \frac{\partial^2 u(x, t)}{\partial x^2} = 0, \quad x \in (a, b), \quad t > 0,$$

o (in più dimensioni)

$$\frac{\partial^2 u(\mathbf{x}, t)}{\partial t^2} - c \Delta u(\mathbf{x}, t) = 0, \quad \mathbf{x} \in \Omega, \quad t > 0,$$

dove c è una costante positiva assegnata.

Per problemi ai limiti ed ai valori iniziali e per equazioni differenziali più generali rimandiamo ad esempio a [Qua06], [EEHJ96] o [Lan03]. Noi ci limiteremo a considerare equazioni come (8.1) e (8.2).

Problema 8.1 (Termodinamica) Se siamo interessati a calcolare la distribuzione della temperatura in un quadrato Ω di lato di lunghezza L , possiamo calcolare la variazione netta d'energia in ciascuna direzione in un quadrato di lato infinitesimo di lunghezza $l \ll L$. Abbiamo

$$J(\mathbf{x}) - J(\mathbf{x} + l\mathbf{e}_i) = -l\mathbf{e}_i \cdot \frac{\partial J}{\partial x_i}(\mathbf{x}),$$

dove $J(\mathbf{x})$ rappresenta il trasferimento di energia per unità di tempo. La legge di Fourier stabilisce che J sia proporzionale alla variazione di temperatura T e quindi

$$J(\mathbf{x}) - J(\mathbf{x} + l\mathbf{e}_i) = -l\mathbf{e}_i \cdot \frac{\partial}{\partial x_i} \left(k l \frac{\partial T}{\partial x_i} \right) = k l^2 \frac{\partial^2 T}{\partial x_i^2},$$

dove k è una costante positiva associata al coefficiente di condutività. All'equilibrio, la somma delle variazioni di energia deve annullarsi e quindi

$$\Delta T(\mathbf{x}) = \sum_{i=1}^d \frac{\partial^2 T}{\partial x_i^2}(\mathbf{x}) = 0 \quad \mathbf{x} \in \Omega,$$

che è un problema di Poisson con $f = 0$.

Problema 8.2 (Idrogeologia) In alcuni casi, lo studio della filtrazione delle acque nel sottosuolo può essere ricondotto alla risoluzione dell'equazione (8.2). Consideriamo una regione tridimensionale Ω occupata da un mezzo poroso (come la terra o l'argilla) la cui conduttività idraulica K sia costante. Allora, per la legge di Darcy si ha che la velocità di filtrazione media dell'acqua $\mathbf{q} = (q_1, q_2, q_3)^T$ è proporzionale alla variazione del livello dell'acqua ϕ nel mezzo, cioè

$$\mathbf{q} = -K(\partial\phi/\partial x_1, \partial\phi/\partial x_2, \partial\phi/\partial x_3)^T. \quad (8.4)$$

Se la densità del fluido è costante, allora il principio di conservazione della massa porta all'equazione $\operatorname{div}\mathbf{q} = 0$, dove

$$\operatorname{div}\mathbf{q} = \sum_{i=1}^3 \frac{\partial q_i}{\partial x_i}$$

è la *divergenza* del vettore \mathbf{q} . Allora, per la (8.4) si ha che ϕ soddisfa al problema di Poisson $\Delta\phi = 0$ (si veda l'Esercizio 8.9).

L'equazione di Poisson (8.2) ammette infinite soluzioni. Al fine di ottenere un'unica soluzione devono essere imposte opportune condizioni su tutto il bordo $\partial\Omega$ di Ω . Una possibilità è assegnare il valore di u su $\partial\Omega$, cioè

$$u(\mathbf{x}) = g(\mathbf{x}) \text{ per } \mathbf{x} \in \partial\Omega, \quad (8.5)$$

dove g è una funzione assegnata.

Il problema (8.2), con le condizioni al contorno (8.5), è detto *problema ai limiti di Dirichlet*, ed è precisamente questo il problema che risolveremo nel prossimo paragrafo. In alternativa a (8.5) si può imporre una condizione sulla derivata parziale di u nella direzione normale a $\partial\Omega$ (ottenendo il cosiddetto *problema ai limiti di Neumann*).

Si può dimostrare che se f e g sono due funzioni continue e la regione Ω è sufficientemente regolare, allora esiste un'unica soluzione u del problema ai limiti di Dirichlet (mentre la soluzione del problema ai limiti di Neumann è unica a meno di una costante additiva).

Nel caso monodimensionale, il problema ai limiti di Dirichlet assume la forma seguente: date due costanti α e β ed una funzione $f = f(x)$, trovare una funzione $u = u(x)$ tale che

$$\begin{aligned} -u''(x) &= f(x) \text{ per } x \in (a, b), \\ u(a) &= \alpha, \quad u(b) = \beta \end{aligned} \quad (8.6)$$

Integrando due volte, si può facilmente mostrare che se $f \in C^0([a, b])$, la soluzione u esiste ed è unica; inoltre, essa appartiene a $C^2([a, b])$.

Anche se governato da un'equazione differenziale ordinaria, il problema (8.6) non può essere messo nella forma di un problema di Cauchy in quanto il valore di u è assegnato in due punti differenti.

I metodi numerici adatti per risolvere i problemi alle derivate parziali in 2 (o più) dimensioni si basano sugli stessi presupposti usati per risolvere il problema (8.6). È per questa ragione che nel paragrafo 8.1 faremo una digressione sulla risoluzione numerica del problema monodimensionale (8.6).

8.1 Approssimazione di problemi ai limiti

Introduciamo su $[a, b]$ una decomposizione in intervalli $I_j = [x_j, x_{j+1}]$ per $j = 0, \dots, N$ con $x_0 = a$ e $x_{N+1} = b$. Supponiamo per semplicità che gli intervalli abbiano tutti la stessa ampiezza h .

8.1.1 Approssimazione con differenze finite

L'equazione differenziale deve essere soddisfatta, in particolare per ogni punto x_j (che chiameremo d'ora in poi *nodo*) interno ad (a, b) , ovvero

$$-u''(x_j) = f(x_j), \quad j = 1, \dots, N.$$

Per ottenere una approssimazione di questo insieme di N equazioni, sostituiamo alla derivata seconda un opportuno rapporto incrementale (come abbiamo fatto nel caso delle derivate prime del Capitolo 4). In particolare, osserviamo che, data una funzione $u : [a, b] \rightarrow \mathbb{R}$ sufficientemente regolare in un intorno di un generico punto $\bar{x} \in (a, b)$, la quantità

$$\delta^2 u(\bar{x}) = \frac{u(\bar{x} + h) - 2u(\bar{x}) + u(\bar{x} - h)}{h^2} \quad (8.7)$$

fornisce una approssimazione di $u''(\bar{x})$ di ordine 2 rispetto a h (si veda l'Esercizio 8.3). Questo suggerisce di usare la seguente approssimazione del problema (8.6): trovare $\{u_j\}_{j=1}^N$ tale che

$$-\frac{u_{j+1} - 2u_j + u_{j-1}}{h^2} = f(x_j), \quad j = 1, \dots, N, \quad (8.8)$$

con $u_0 = \alpha$ e $u_{N+1} = \beta$. Le equazioni (8.8) formano un sistema lineare

$$\mathbf{A}\mathbf{u}_h = h^2\mathbf{f}, \quad (8.9)$$

dove $\mathbf{u}_h = (u_1, \dots, u_N)^T$ è il vettore delle incognite, $\mathbf{f} = (f(x_1) + \alpha/h^2, f(x_2), \dots, f(x_{N-1}), f(x_N) + \beta/h^2)^T$, ed \mathbf{A} è la matrice tridiagonale

$$\begin{bmatrix} 2 & -1 & 0 & \dots & 0 \\ -1 & 2 & \ddots & & \vdots \\ 0 & \ddots & \ddots & -1 & 0 \\ \vdots & & -1 & 2 & -1 \\ 0 & \dots & 0 & -1 & 2 \end{bmatrix}. \quad (8.10)$$

Tale sistema ammette un'unica soluzione in quanto A è simmetrica e definita positiva (si veda l'Esercizio 8.1). Essendo A anche tridiagonale, il sistema potrà essere risolto utilizzando il metodo di Thomas (presentato nel paragrafo 5.4). Si tenga comunque conto che la matrice A è, per h piccolo (e quindi per grandi valori di N), malcondizionata. Infatti, $\forall h > 0$, $K(A) = \lambda_{\max}(A)/\lambda_{\min}(A) = Ch^{-2}$, per un'opportuna costante C indipendente da h (si veda l'Esercizio 8.2). Di conseguenza, la risoluzione numerica del sistema (8.9) richiede una cura particolare sia nel caso in cui si usi un metodo diretto sia in quello in cui si usi un metodo iterativo (in questo secondo caso converrà ricorrere ad un precondizionatore).

Si può dimostrare (si veda, ad esempio, [QSS04, Capitolo 12]) che, se $f \in C^2([a, b])$, allora

$$\max_{j=0, \dots, N+1} |u(x_j) - u_j| \leq \frac{h^2}{96} \max_{x \in [a, b]} |f''(x)| \quad (8.11)$$

cioè il metodo alle differenze finite (8.8) converge con ordine 2 rispetto a h .

Nel Programma 25 viene risolto il problema ai limiti

$$\begin{cases} -u''(x) + \delta u'(x) + \gamma u(x) = f(x) & \text{per } x \in (a, b), \\ u(a) = \alpha, & \\ u(b) = \beta, & \end{cases} \quad (8.12)$$

con γ e δ costanti, che è un'estensione del problema (8.6). Lo schema alle differenze finite utilizzato, che generalizza (8.8), è il seguente

$$\begin{cases} -\frac{u_{j+1} - 2u_j + u_{j-1}}{h^2} + \delta \frac{u_{j+1} - u_{j-1}}{2h} + \gamma u_j = f(x_j), & j = 1, \dots, N, \\ u_0 = \alpha, & \\ u_{N+1} = \beta. & \end{cases}$$

I parametri d'ingresso del Programma 25 sono gli estremi a e b dell'intervallo di definizione, il numero di nodi interni N , i coefficienti costanti δ e γ e la *function* **bvpfun** che dovrà precisare l'espressione di f . Infine, ua e ub sono i valori che deve assumere la soluzione in $x=a$ ed in $x=b$. In uscita, vengono restituiti il vettore dei nodi di discretizzazione x e la soluzione calcolata uh . Si noti che le soluzioni generate con questo programma possono essere affette da oscillazioni spurie se $h < 2/|\delta|$ (si veda l'Esercizio 8.6).

Programma 25 - bvp : approssimazione di un problema ai limiti con il metodo delle differenze finite

```
function [x,uh]=bvp(a,b,N,delta,gamma,bvpfun,ua,ub,varargin)
%BVP Risolve un problema ai limiti
% [X,UH]=BVP(A,B,N,DELTA,GAMMA,BVPFUN,UA,UB) risolve con il
% metodo delle differenze finite centrate il problema
```



```
% -D(DU/DX)/DX+DELTA*DU/DX+GAMMA*U=BVPFUN
% sull'intervallo (A,B) con condizioni al bordo U(A)=UA
% e U(B)=UB.
% BVPFUN puo' essere una funzione inline.
h = (b-a)/(N+1);
z = linspace(a,b,N+2);
e = ones(N,1);
A = spdiags([-e-0.5*h*delta 2*e+gamma*h^2 -e+0.5*h*delta], -1:1, N, N);
x = z(2:end-1);
f = h^2*feval(bvpfun,x,varargin{:});
f=f'; f(1) = f(1) + ua; f(end) = f(end) + ub;
uh = A\f;
uh=[ua; uh; ub];
x = z;
```

8.1.2 Approssimazione con elementi finiti

Il *metodo degli elementi finiti* rappresenta un'alternativa al metodo delle differenze finite appena introdotto. Viene derivato da un'opportuna riformulazione del problema (8.6).

Moltiplichiamo ambo i membri della (8.6) per una generica funzione v , definita su $[a, b]$; integrando la corrispondente uguaglianza sull'intervallo (a, b) , otteniamo

$$-\int_a^b u''(x)v(x) \, dx = \int_a^b f(x)v(x) \, dx. \quad (8.13)$$

Se assumiamo $v \in C^1([a, b])$ ed usiamo la formula di integrazione per parti, otteniamo

$$\int_a^b u'(x)v'(x) \, dx - [u'(x)v(x)]_a^b = \int_a^b f(x)v(x) \, dx.$$

Supponendo inoltre che v si annulli negli estremi $x = a$ e $x = b$, il problema (8.6) diventa: trovare u tale che $u(a) = \alpha$, $u(b) = \beta$ e

$$\int_a^b u'(x)v'(x) \, dx = \int_a^b f(x)v(x) \, dx \quad (8.14)$$

per ogni $v \in C^1([a, b])$ tale che $v(a) = v(b) = 0$. La (8.14) viene chiamata *formulazione debole* del problema (8.6) (di fatto, le funzioni test v possono essere meno regolari di $C^1([a, b])$, si veda, ad esempio [QSS04] o [Qua06]).

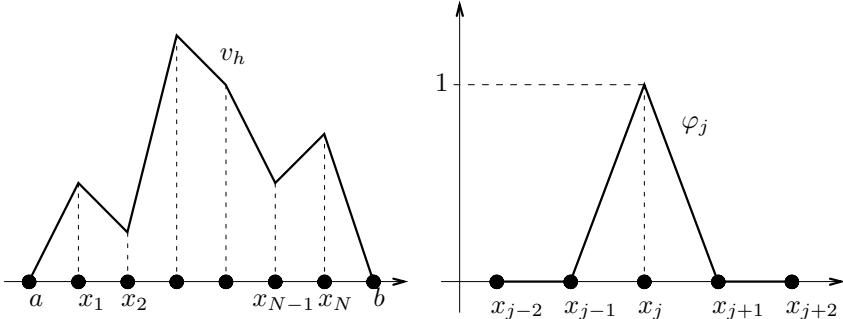


Figura 8.1. A sinistra, una generica funzione $v_h \in V_h^0$. A destra, la funzione di base per V_h^0 associata al nodo k -esimo

La sua approssimazione ad elementi finiti è definita come segue

trovare $u_h \in V_h$ tale che $u_h(a) = \alpha, u_h(b) = \beta$ e

$$\sum_{j=0}^{N-1} \int_{x_j}^{x_{j+1}} u'_h(x) v'_h(x) \, dx = \int_a^b f(x) v_h(x) \, dx, \quad \forall v_h \in V_h^0 \quad (8.15)$$

dove

$$V_h = \{v_h \in C^0([a, b]) : v_h|_{I_j} \in \mathbb{P}_1, j = 0, \dots, N\},$$

cioè V_h è lo spazio delle funzioni continue in (a, b) le cui restrizioni in ogni sotto intervallo I_j sono polinomi di grado uno, mentre V_h^0 è il sottospazio di V_h delle funzioni che si annullano agli estremi a e b . V_h è detto lo spazio degli elementi finiti di grado 1.

Le funzioni di V_h^0 sono polinomi composti di grado 1 (si veda la Figura 8.1 a sinistra). Di conseguenza, ogni funzione v_h di V_h^0 ammette la seguente rappresentazione

$$v_h(x) = \sum_{j=1}^N v_h(x_j) \varphi_j(x),$$

dove per $j = 1, \dots, N$

$$\varphi_j(x) = \begin{cases} \frac{x - x_{j-1}}{x_j - x_{j-1}} & \text{se } x \in I_{j-1}, \\ \frac{x - x_{j+1}}{x_j - x_{j+1}} & \text{se } x \in I_j, \\ 0 & \text{altrimenti.} \end{cases}$$

La generica φ_j è dunque nulla in tutti i nodi x_i fuorché in x_j dove vale 1 (si veda la Figura 8.1 a destra). Le funzioni φ_j , $j = 1, \dots, N$ sono dette *funzioni di forma* e formano una base per lo spazio V_h^0 .

Di conseguenza, verificare la (8.15) per ogni funzione di V_h equivale a verificarla per le sole funzioni di forma φ_j , $j = 1, \dots, N$. Sfruttando la proprietà che φ_j si annulla al di fuori degli intervalli I_{j-1} e I_j , dalla (8.15) otteniamo

$$\int_{I_{j-1} \cup I_j} u'_h(x) \varphi'_j(x) \, dx = \int_{I_{j-1} \cup I_j} f(x) \varphi_j(x) \, dx, \quad j = 1, \dots, N. \quad (8.16)$$

D'altra parte possiamo scrivere $u_h(x) = \sum_{j=1}^N u_j \varphi_j(x) + \alpha \varphi_0(x) + \beta \varphi_{N+1}(x)$, dove $u_j = u_h(x_j)$, $\varphi_0(x) = (a + h - x)/h$ per $a \leq x \leq a + h$, e $\varphi_{N+1}(x) = (x - b + h)/h$ per $b - h \leq x \leq b$, mentre $\varphi_0(x)$ e $\varphi_{N+1}(x)$ sono nulle altrove. Sostituendo questa espressione in (8.16), per ogni $j = 1, \dots, N$ troviamo

$$\begin{aligned} & u_{j-1} \int_{I_{j-1}} \varphi'_{j-1}(x) \varphi'_j(x) \, dx + u_j \int_{I_{j-1} \cup I_j} \varphi'_j(x) \varphi'_j(x) \, dx \\ & + u_{j+1} \int_{I_j} \varphi'_{j+1}(x) \varphi'_j(x) \, dx = \int_{I_{j-1} \cup I_j} f(x) \varphi_j(x) \, dx + B_{1,j} + B_{N,j}, \end{aligned}$$

dove

$$B_{1,j} = \begin{cases} -\alpha \int_{I_0} \varphi'_0(x) \varphi'_1(x) \, dx = -\frac{\alpha}{x_1 - a} & \text{se } j = 1, \\ 0 \text{ altrimenti,} & \end{cases}$$

mentre

$$B_{N,j} = \begin{cases} -\beta \int_{I_N} \varphi'_{N+1}(x) \varphi'_j(x) \, dx = -\frac{\beta}{b - x_N} & \text{se } j = N, \\ 0 \text{ altrimenti.} & \end{cases}$$

Avendo supposto all'inizio del paragrafo che tutti gli intervalli abbiano la stessa ampiezza h , abbiamo $\varphi'_{j-1} = -1/h$ in I_{j-1} , $\varphi'_j = 1/h$ in I_{j-1} e $\varphi'_j = -1/h$ in I_j , $\varphi'_{j+1} = 1/h$ in I_j . Di conseguenza, otteniamo per $j = 1, \dots, N$

$$-u_{j-1} + 2u_j - u_{j+1} = h \int_{I_{j-1} \cup I_j} f(x) \varphi_j(x) \, dx + B_{1,j} + B_{N,j}.$$

Si trova dunque un sistema lineare nelle incognite $\{u_1, \dots, u_N\}$ con matrice uguale a quella ottenuta nel caso delle differenze finite e termine noto diverso (e, naturalmente, anche una diversa soluzione a dispetto delle notazioni coincidenti). Elementi finiti lineari e differenze finite condividono invece la stessa accuratezza rispetto a h quando si calcoli l'errore massimo nei soli nodi.

L'approccio degli elementi finiti può essere generalizzato a problemi come (8.12) (anche nel caso in cui δ e γ dipendano da x). Un'ulteriore generalizzazione consiste nell'usare polinomi compositi di grado maggiore di uno. In tal caso si incrementa l'ordine di accuratezza dello schema. Facciamo notare che la matrice associata ad approssimazioni di ordine elevato non coincide più con la matrice delle differenze finite.

Si vedano gli Esercizi 8.1-8.8.



8.2 Le differenze finite in 2 dimensioni

Consideriamo un'equazione alle derivate parziali, ad esempio la (8.2), in una regione Ω del piano.

L'idea alla base delle differenze finite consiste nell'approssimare le derivate parziali ancora con rapporti incrementalii su una opportuna griglia (detta griglia computazionale) costituita da un insieme finito di nodi. In questo modo u verrà approssimata solo in questi nodi.

Il primo problema è quindi quello di costruire una griglia computazionale. Supponiamo per semplicità che Ω sia il rettangolo $(a, b) \times (c, d)$. Introduciamo una decomposizione di $[a, b]$ in intervalli (x_k, x_{k+1}) per $k = 0, \dots, N_x$, con $x_0 = a$ e $x_{N_x+1} = b$. Indichiamo con $\Delta_x = \{x_0, \dots, x_{N_x+1}\}$ l'insieme degli estremi di tali intervalli e con $h_x = \max_{k=0, \dots, N_x} (x_{k+1} - x_k)$ la loro massima lunghezza.

In modo del tutto analogo introduciamo una discretizzazione $\Delta_y = \{y_0, \dots, y_{N_y+1}\}$, con $y_0 = c$ e $y_{N_y+1} = d$, dell'asse delle y in intervallini la cui massima lunghezza sia h_y . Il prodotto cartesiano $\Delta_h = \Delta_x \times \Delta_y$ definisce la griglia computazionale su Ω (come mostrato in Figura 8.2), avendo posto $h = \max\{h_x, h_y\}$. Siamo interessati a trovare i valori $u_{i,j}$ che approssimano $u(x_i, y_j)$. Supponiamo per semplicità che i nodi siano equispaziati ossia che $x_i = x_0 + ih_x$ per $i = 0, \dots, N_x + 1$ e $y_j = y_0 + jh_y$ per $j = 0, \dots, N_y + 1$.

Le derivate seconde parziali di una funzione possono essere approssimate con un opportuno rapporto incrementale, esattamente come fatto per le derivate ordinarie. Nel caso di una funzione di 2 variabili definiamo i seguenti rapporti incrementalii

$$\delta_x^2 u_{i,j} = \frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{h_x^2}, \quad \delta_y^2 u_{i,j} = \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{h_y^2}. \quad (8.17)$$

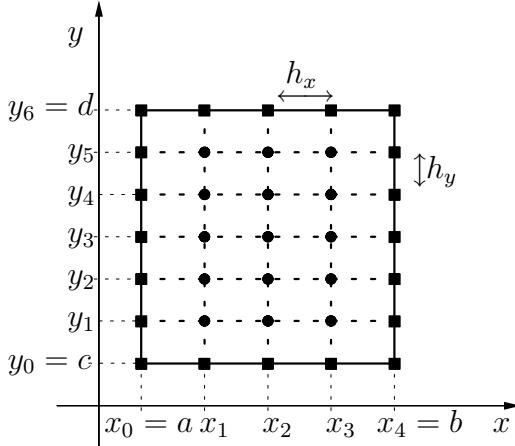


Figura 8.2. Griglia di calcolo Δ_h di soli 15 nodi interni su un dominio rettangolare

Essi sono accurati al second'ordine rispetto a h_x ed a h_y , rispettivamente, per l'approssimazione di $\partial^2 u / \partial x^2$ e $\partial^2 u / \partial y^2$ nel nodo (x_i, y_j) . Se sostituiamo le derivate parziali seconde di u con le formule (8.17), richiedendo che l'equazione alle derivate parziali venga soddisfatta in tutti i nodi interni di Δ_h , perveniamo alle seguenti equazioni

$$-(\delta_x^2 u_{i,j} + \delta_y^2 u_{i,j}) = f_{i,j}, \quad i = 1, \dots, N_x, \quad j = 1, \dots, N_y. \quad (8.18)$$

Abbiamo posto $f_{i,j} = f(x_i, y_j)$. Ad esse vanno aggiunte le equazioni che impongono il dato di Dirichlet sul bordo

$$u_{i,j} = g_{i,j} \quad \forall i, j \text{ tale che } (x_i, y_j) \in \partial \Delta_h, \quad (8.19)$$

dove $\partial \Delta_h$ denota l'insieme dei punti di Δ_h che appartengono al bordo del rettangolo. Tali punti sono indicati in Figura 8.2 con dei quadratini. Si tenga conto che, se supponiamo che la griglia sia uniforme in entrambe le direzioni, cioè che $h_x = h_y = h$, invece di (8.18) otteniamo

$$\boxed{-\frac{1}{h^2}(u_{i-1,j} + u_{i,j-1} - 4u_{i,j} + u_{i,j+1} + u_{i+1,j}) = f_{i,j}, \quad i = 1, \dots, N_x, \quad j = 1, \dots, N_y} \quad (8.20)$$

Il sistema formato dalle equazioni (8.18) (o (8.20)) e (8.19) consente di calcolare i valori nodali $u_{i,j}$ in tutti i nodi di Δ_h . Per ogni coppia fissata di indici i e j , l'equazione (8.20) coinvolge 5 valori nodali, come mostrato in Figura 8.3. Per questo motivo questo schema è noto come *schema a 5 punti* per l'operatore di Laplace.

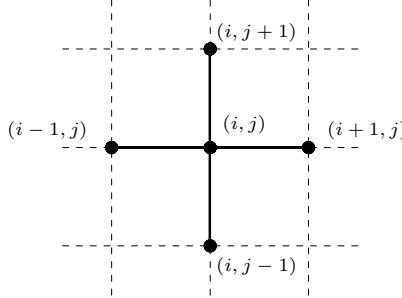


Figura 8.3. Supporto dello schema a 5 punti per il problema di Poisson

Le incognite associate ai nodi di bordo, possono essere eliminate usando (8.19) (o (8.18)), e quindi (8.20) coinvolge solo $N = N_x N_y$ incognite.

Il sistema risultante può essere scritto in modo più significativo se ordiniamo opportunamente i nodi interni della griglia: a partire dal nodo 1 individuato da (x_1, y_1) e proseguendo da sinistra verso destra, dal basso verso l'alto, numeriamo progressivamente tutti i nodi interni. Con tale ordinamento, detto *lessicografico*, il sistema lineare associato ai soli nodi interni prende ancora la forma (8.9). Tuttavia stavolta la matrice $A \in \mathbb{R}^{N \times N}$ ha la seguente forma (tridiagonale a blocchi)

$$A = \begin{bmatrix} T & D & 0 & \dots & 0 \\ D & T & \ddots & & \vdots \\ 0 & \ddots & \ddots & D & 0 \\ \vdots & & D & T & D \\ 0 & \dots & 0 & D & T \end{bmatrix}.$$

Essa ha N_y righe e N_y colonne ed ogni ingresso (denotato con una lettera maiuscola) è una matrice $N_x \times N_x$. In particolare, $D \in \mathbb{R}^{N_x \times N_x}$ è la matrice diagonale i cui coefficienti diagonali valgono $-1/h_y^2$, mentre $T \in \mathbb{R}^{N_x \times N_x}$ è la seguente matrice simmetrica tridiagonale

$$T = \begin{bmatrix} \frac{2}{h_x^2} + \frac{2}{h_y^2} & -\frac{1}{h_x^2} & 0 & \dots & 0 \\ -\frac{1}{h_x^2} & \frac{2}{h_x^2} + \frac{2}{h_y^2} & \ddots & & \vdots \\ 0 & \ddots & \ddots & -\frac{1}{h_x^2} & 0 \\ \vdots & & -\frac{1}{h_x^2} & \frac{2}{h_x^2} + \frac{2}{h_y^2} & -\frac{1}{h_x^2} \\ 0 & \dots & 0 & -\frac{1}{h_x^2} & \frac{2}{h_x^2} + \frac{2}{h_y^2} \end{bmatrix}.$$

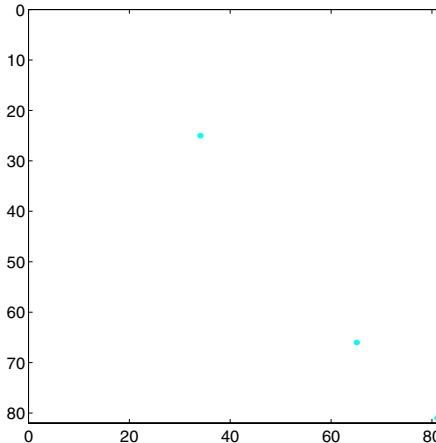


Figura 8.4. Struttura della matrice associata allo schema a 5 punti usando un ordinamento lessicografico delle incognite

La matrice A è simmetrica in quanto tutti i blocchi diagonali sono simmetrici. Verifichiamo che è anche definita positiva dimostrando che $\mathbf{v}^T A \mathbf{v} > 0$ per ogni $\mathbf{v} \in \mathbb{R}^N$, $\mathbf{v} \neq \mathbf{0}$. Partizionando \mathbf{v} in N_y vettori \mathbf{v}_i di lunghezza N_x , otteniamo

$$\mathbf{v}^T A \mathbf{v} = \sum_{k=1}^{N_y} \mathbf{v}_k^T T \mathbf{v}_k - \frac{2}{h_y^2} \sum_{k=1}^{N_y-1} \mathbf{v}_k^T \mathbf{v}_{k+1}. \quad (8.21)$$

Possiamo scrivere $T = 2/h_y^2 I + 1/h_x^2 K$, dove K è la matrice (simmetrica e definita positiva) data nella (8.10) e I è la matrice identità. Di conseguenza, (8.21) diventa

$$(\mathbf{v}_1^T K \mathbf{v}_1 + \mathbf{v}_2^T K \mathbf{v}_2 + \dots + \mathbf{v}_{N_y}^T K \mathbf{v}_{N_y})/h_x^2$$

che è un numero reale strettamente positivo in quanto K è definita positiva ed almeno uno dei vettori \mathbf{v}_i è non nullo.

Avendo provato che A è non singolare, il sistema ammette un'unica soluzione \mathbf{u}_h .

Osserviamo che A è una matrice *sparsa* e come tale verrà memorizzata nel formato **sparse** in MATLAB (si veda il paragrafo 5.4). In Figura 8.4 riportiamo la struttura della matrice (ottenuta con il comando **spy(A)**) per una griglia uniforme di 11×11 nodi, dopo aver eliminato le righe e le colonne associate ai nodi di $\partial\Delta_h$. Come si nota, i soli elementi non nulli (indicati con dei pallini) sono tutti disposti su 5 diagonali.

Inoltre, essendo A simmetrica e definita positiva il sistema può essere risolto sia con metodi iterativi che diretti, come illustrato nel Capitolo 5. Infine, notiamo che, come per la corrispondente matrice del

caso monodimensionale, A è mal condizionata in quanto il suo numero di condizionamento cresce come h^{-2} al decrescere di h .

Nel Programma 26 costruiamo e risolviamo (con il metodo richiamato dal comando \ di MATLAB, si veda il paragrafo 5.6) il sistema (8.18)-(8.19). I parametri d'ingresso **a**, **b**, **c** e **d** servono per precisare gli estremi del dominio rettangolare $\Omega = (a, b) \times (c, d)$, mentre **nx** e **ny** indicano i valori assunti da N_x e N_y (si può avere $N_x \neq N_y$). Infine, **fun** e **bound** sono le stringhe che precisano la funzione $f = f(x, y)$ ed il dato di Dirichlet $g = g(x, y)$. In uscita, **u** è una matrice avente come elemento i, j -esimo il valore u_{ij} . La soluzione numerica può essere visualizzata con il comando **mesh(x,y,u)**. La stringa **uem** (opzionale) precisa invece la soluzione esatta del problema, nel caso (di interesse accademico) in cui essa sia nota. In tal caso viene calcolato l'errore relativo (contenuto nel parametro di *output error*)

$$\text{error} = \max_{i,j} |u(x_i, y_j) - u_{i,j}| / \max_{i,j} |u(x_i, y_j)|.$$

mesh

Programma 26 - poissonfd : approssimazione del problema di Poisson con condizioni di Dirichlet usando il metodo a 5 punti delle differenze finite



```
function [u,x,y,error]=poissonfd(a,b,c,d,nx,ny,fun,bound,uex,varargin)
%POISSONFD approssimazione del problema di Poisson in due dimensioni
% [U,X,Y]=POISSONFD(A,B,C,D,NX,XY,FUN,BOUND) risolve con lo schema
% alle differenze finite a 5 punti il problema -LAPL(U) = FUN nel
% rettangolo (A,B)X(C,D) con condizioni al bordo di Dirichlet
% U(X,Y)=BOUND(X,Y) per ogni (X,Y) sul bordo del rettangolo.
% [U,X,Y,ERROR]=POISSONFD(A,B,C,D,NX,XY,FUN,BOUND,UEX)
% calcola anche l'errore nodale massimo ERROR rispetto alla soluzione
% esatta UEX. FUN, BOUND e UEX possono essere funzioni inline.
if nargin == 8
    uex = inline('0','x','y');
end
nx=nx+1;          ny=ny+1;
hx = (b-a)/nx;    hy = (d-c)/ny;
nx1 = nx+1;        hx2 = hx^2;      hy2 = hy^2;
kii = 2/hx2+2/hy2; kix = -1/hx2;   kiy = -1/hy2;
dim = (nx+1)*(ny+1); K = speye(dim,dim); rhs = zeros(dim,1);
y = c;
for m = 2:ny
    x = a; y = y + hy;
    for n = 2:nx
        i = n+(m-1)*(nx+1);
        x = x + hx;
        rhs(i) = feval(fun,x,y,varargin{:});
        K(i,i) = kii;    K(i,i-1) = kix;    K(i,i+1) = kix;
        K(i,i+nx1) = kiy; K(i,i-nx1) = kiy;
    end
end
```

```

    end
end
rhs1 = zeros(dim,1);
x = [a:hx:b];
rhs1(1:nx1) = feval(bound,x,c,varargin{:});
rhs1(dim-nx:dim) = feval(bound,x,d,varargin{:});
y = [c:hy:d];
rhs1(1:nx1:dim-nx) = feval(bound,a,y,varargin{:});
rhs1(nx1:nx1:dim) = feval(bound,b,y,varargin{:});
rhs = rhs - K*rhs1;
nbound = [[1:nx1],[dim-nx:dim],[1:nx1:dim-nx],[nx1:nx1:dim]];
ninternal = setdiff([1:dim],nbound);
K = K(ninternal,ninternal);
rhs = rhs(ninternal);
utemp = K\rhs;
uh = rhs1;
uh (ninternal) = utemp;
k = 1; y = c;
for j = 1:ny+1
    x = a;
    for i = 1:nx1
        u(i,j) = uh(k);           k = k + 1;
        ue(i,j) = feval(uex,x,y,varargin{:});
        x = x + hx;
    end
    y = y + hy;
end
x = [a:hx:b]; y = [c:hy:d];
if nargout == 4 & nargin == 8
    warning('Soluzione esatta non disponibile');
    error = [ ];
else
    error = max(max(abs(u-ue)))/max(max(abs(ue)));
end, end
return

```

Esempio 8.1 Lo spostamento trasversale u di una membrana, rispetto al piano di riferimento $z = 0$ nel dominio $\Omega = (0, 1)^2$, soggetto all'azione di una forza di intensità pari a $f(x, y) = 8\pi^2 \sin(2\pi x) \cos(2\pi y)$ soddisfa il problema di Poisson (8.2) in Ω . Le condizioni al bordo di Dirichlet sullo spostamento sono: $g = 0$ sui lati $x = 0$ e $x = 1$, e $g(x, 0) = g(x, 1) = \sin(2\pi x)$, $0 < x < 1$. Questo problema ammette come soluzione esatta la funzione $u(x, y) = \sin(2\pi x) \cos(2\pi y)$. In Figura 8.5 viene riportata la soluzione numerica ottenuta con lo schema a 5 punti. Sono stati usati due differenti valori di h : $h = 1/10$ (a sinistra) e $h = 1/20$ (a destra). Al decrescere di h la soluzione numerica migliora e, in effetti, l'errore nodale relativo passa da 0.0292 per $h = 1/10$ a 0.0081 per $h = 1/20$.

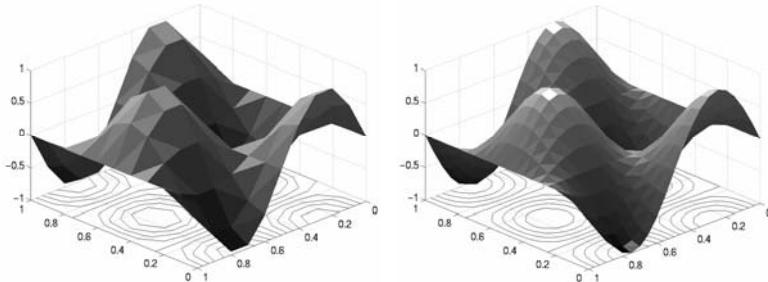


Figura 8.5. Spostamento trasversale di una membrana calcolato con una griglia più rada (a sinistra) e con una più fitta (a destra). Sul piano orizzontale vengono riportate le isoline della soluzione. La decomposizione in triangoli che compare nelle figure tridimensionali serve per la sola visualizzazione

Anche il metodo degli elementi finiti può essere facilmente esteso al caso bidimensionale. Si dovrà scrivere una opportuna formulazione integrale del problema (8.2) e sostituire alla decomposizione dell'intervallo (a, b) in sottointervalli una decomposizione in poligoni (tipicamente, triangoli) detti *elementi*. La generica funzione φ_k sarà ancora una funzione polinomiale di grado 1 su ogni elemento, globalmente continua, e pari ad 1 nel vertice k -esimo e 0 nei restanti vertici degli elementi della griglia di calcolo. Per una implementazione di tale metodo in 2 dimensioni si può usare il toolbox [pde](#).

8.2.1 Consistenza e convergenza



Nel precedente paragrafo abbiamo stabilito che la soluzione del problema approssimato esiste ed è unica. Siamo ora interessati a studiare l'errore di approssimazione. Supponiamo nuovamente che $h_x = h_y = h$. Se

$$\max_{i,j} |u(x_i, y_j) - u_{i,j}| \rightarrow 0 \quad \text{quando } h \rightarrow 0,$$

diremo che il metodo usato per calcolare $u_{i,j}$ è convergente.

Come abbiamo già osservato, condizione necessaria per la convergenza è che il metodo sia consistente. Ciò significa che, forzando la soluzione esatta a soddisfare lo schema numerico, l'errore che si ottiene, detto *di truncamento*, deve tendere a 0 per h che tende a 0. Nel caso del metodo a 5 punti in ogni nodo (x_i, y_j) interno a Δ_h poniamo

$$\tau_h(x_i, y_j) = -f(x_i, y_j) - \frac{u(x_{i-1}, y_j) + u(x_i, y_{j-1}) - 4u(x_i, y_j) + u(x_i, y_{j+1}) + u(x_{i+1}, y_j)}{h^2}.$$

Questo valore è detto *errore di truncamento locale* nel nodo (x_i, y_j) . Grazie alla (8.2) si ottiene

$$\begin{aligned}\tau_h(x_i, y_j) = & \left\{ \frac{\partial^2 u}{\partial x^2}(x_i, y_j) - \frac{u(x_{i-1}, y_j) - 2u(x_i, y_j) + u(x_{i+1}, y_j)}{h^2} \right\} \\ & + \left\{ \frac{\partial^2 u}{\partial y^2}(x_i, y_j) - \frac{u(x_i, y_{j-1}) - 2u(x_i, y_j) + u(x_i, y_{j+1})}{h^2} \right\}.\end{aligned}$$

Grazie all'analisi svolta nel paragrafo 8.2, possiamo concludere che entrambi i termini tendono a 0 quando h tende a 0. Dunque

$$\lim_{h \rightarrow 0} \tau_h(x_i, y_j) = 0, \quad \forall (x_i, y_j) \in \Delta_h \setminus \partial \Delta_h,$$

ossia il metodo a 5 punti è consistente. In effetti si può dimostrare che esso è anche convergente in quanto vale il seguente risultato (per la cui dimostrazione si veda, ad esempio, [IK66]):

Proposizione 8.1 *Supponiamo che la soluzione esatta $u \in C^4(\bar{\Omega})$, cioè che abbia tutte le sue derivate parziali fino al quart'ordine continue sulla chiusura $\bar{\Omega}$ del dominio. Allora, esiste una costante $C > 0$ tale che*

$$\max_{i,j} |u(x_i, y_j) - u_{i,j}| \leq CMh^2 \quad (8.22)$$

dove M è il massimo valore assoluto assunto dalle derivate quarte di u in $\bar{\Omega}$.

Esempio 8.2 Verifichiamo l'ordine del metodo a 5 punti sul problema di Poisson dell'Esempio 8.1. Risolviamo tale problema a partire da $h = 1/4$ e, per dimezzamenti successivi, fino a $h = 1/64$ con le seguenti istruzioni

```
>> a=0;b=1;c=0;d=1;
>> f=inline('8*pi^2*sin(2*pi*x).*cos(2*pi*y)','x','y');
>> g=inline('sin(2*pi*x).*cos(2*pi*y)','x','y'); uex=g; nx=4; ny=4;
>> for n=1:5
    [u,x,y,error(n)]=poissonfd(a,c,b,d,nx,ny,f,g,uex); nx = 2*nx; ny = 2*ny;
end
```

Il vettore contenente l'errore è

```
>> format short e; error
1.3565e-01 4.3393e-02 1.2308e-02 3.2775e-03 8.4557e-04
```

Come si può facilmente verificare con i comandi

```
>> log(abs(error(1:end-1)./error(2:end)))/log(2)
1.6443e+00 1.8179e+00 1.9089e+00 1.9546e+00
```

tale errore tende a decresce con ordine 2 rispetto a h^2 quando $h \rightarrow 0$.

Riassumendo



1. I problemi ai valori ai limiti sono equazioni differenziali in un dominio spaziale $\Omega \subset \mathbb{R}^d$ (un intervallo se $d = 1$) che richiedono informazioni sulla soluzione sul bordo di Ω ;
2. il metodo delle differenze finite è basato sulla discretizzazione di una data equazione differenziale in punti selezionati (chiamati nodi) nei quali le derivate vengono approssimate da formule alle differenze finite;
3. esso produce un vettore le cui componenti convergono ai corrispondenti valori della soluzione esatta quadraticamente rispetto al passo di discretizzazione;
4. il metodo degli elementi finiti è basato su una opportuna riformulazione integrale dell'equazione differenziale originaria e sull'assunzione che la soluzione approssimata sia un polinomio lineare composito;
5. le matrici associate a discretizzazioni agli elementi finiti ed alle differenze finite sono sparse e mal condizionate.

Si vedano gli Esercizi 8.9-8.10.



8.3 Che cosa non vi abbiamo detto

Potremmo semplicemente dire che vi abbiamo detto quasi nulla in quanto il settore dell'Analisi Numerica che si occupa dell'approssimazione delle equazioni alle derivate parziali è così vasto e sfaccettato da meritare un libro intero solo per fornire un'idea di massima (si vedano a questo proposito [Qua06], [QV94], [EEHJ96] e [TW98]).

Tuttavia, è bene segnalare che il metodo agli elementi finiti è il metodo oggi probabilmente più diffuso per la risoluzione di problemi differenziali (si vedano, ad esempio, [QV94], [Bra97], [BS01]). Come già notato il toolbox MATLAB `pde` consente di risolvere una famiglia abbastanza grande di equazioni alle derivate parziali con elementi finiti lineari.

Altre tecniche molto diffuse sono i metodi spettrali (si vedano [CHQZ06], [Fun92], [BM92], [KS99]) ed il metodo dei volumi finiti (si vedano [Krö98], [Hir88] e [LeV02]).

8.4 Esercizi

Esercizio 8.1 Si verifichi che la matrice (8.10) è definita positiva.

Esercizio 8.2 Si verifichi che la matrice (8.10) ha autovalori pari a

$$\lambda_j = 2(1 - \cos(j\theta)), \quad j = 1, \dots, N,$$

e corrispondenti autovettori dati da

$$\mathbf{q}_j = (\sin(j\theta), \sin(2j\theta), \dots, \sin(nj\theta))^T,$$

dove $\theta = \pi/(n+1)$. Si concluda che $K(A)$ è proporzionale a h^{-2} .

Esercizio 8.3 Si dimostri che la quantità (8.7) fornisce una approssimazione di $u''(\bar{x})$ di ordine 2 rispetto a h .

Esercizio 8.4 Si ricavino matrice e termine noto del sistema relativo allo schema proposto per la risoluzione del problema (8.12).

Esercizio 8.5 Si risolva il seguente problema ai limiti con il metodo delle differenze finite

$$\begin{cases} -u'' + \frac{k}{T}u = \frac{w}{T} & \text{in } (0, 1), \\ u(0) = u(1) = 0, \end{cases}$$

dove $u = u(x)$ rappresenta lo spostamento verticale di una fune lunga 1 metro, soggetta ad un carico trasversale di intensità $w(x)$ per unità di lunghezza. T è la tensione e k il coefficiente di elasticità della fune. Si prendano $w(x) = 1 + \sin(4\pi x)$, $T = 1$ e $k = 0.1$ e si confrontino i risultati ottenuti con $h = 1/i$ con $i = 10, 20, 40$. Si verifichi sperimentalmente l'ordine di convergenza del metodo.

Esercizio 8.6 Si consideri il problema (8.12) sull'intervallo $(0, 1)$ con $\gamma = 0$, $f = 0$, $\alpha = 0$ e $\beta = 1$. Tramite il Programma 25 si individui il massimo valore di h , h_{crit} , per il quale la soluzione numerica ha un andamento monotono per $\delta = 100$, analogo cioè a quello della soluzione esatta. Cosa accade se $\delta = 1000$? Si proponga una formula empirica per $h_{crit}(\delta)$ e la si verifichi per altri valori di δ .

Esercizio 8.7 Si risolva con le differenze finite il problema (8.12) nel caso in cui si considerino le seguenti condizioni ai limiti (dette *di Neumann*)

$$u'(a) = \alpha, \quad u'(b) = \beta.$$

Si usino le formule (4.11) per approssimare $u'(a)$ e $u'(b)$.

Esercizio 8.8 Si verifichi che per una griglia uniforme, il termine noto del sistema associato allo schema alle differenze finite coincide con quello dello schema agli elementi finiti quando in quest'ultimo si usi la formula composita del trapezio per approssimare gli integrali sugli elementi I_{k-1} ed I_k .

Esercizio 8.9 Si verifichi che $\operatorname{div} \nabla \phi = \Delta \phi$, dove ∇ è l'operatore gradiente che associa ad una funzione ϕ il vettore che ha come componenti le derivate parziali prime della ϕ stessa.

Esercizio 8.10 Si consideri una piastra bidimensionale quadrata di lato 20 cm e di conduttività termica $k = 0.2 \text{ cal}/(\text{sec}\cdot\text{cm}\cdot\text{C})$. Denotiamo con $Q = 5 \text{ cal}/(\text{cm}^3\cdot\text{sec})$ il tasso di generazione di calore per unità d'area. In tal caso la temperatura $T = T(x, y)$ della piastra soddisfa l'equazione $-\Delta T = Q/k$. Supponendo che T sia nulla su tre lati del bordo della piastra e valga 1 sul quarto lato, si determini la temperatura al centro della piastra.

Soluzione degli esercizi proposti

9.1 Capitolo 1

Soluzione 1.1 Stanno in $\mathbb{F}(2, 2, -2, 2)$ tutti i numeri della forma $\pm 0.1a_2 \cdot 2^e$ con $a_2 = 0, 1$ ed e intero compreso fra -2 e 2 . Fissato l'esponente, si possono rappresentare i soli numeri 0.10 e 0.11 , a meno del segno; di conseguenza, in $\mathbb{F}(2, 2, -2, 2)$ sono contenuti 20 numeri. Inoltre, $\epsilon_M = 1/2$.

Soluzione 1.2 Fissato l'esponente, abbiamo a disposizione β posizioni per le cifre a_2, \dots, a_t e $\beta - 1$ per la cifra a_1 (che non può assumere il valore 0). In tutto, avremo perciò $(\beta - 1)\beta^{t-1}$ numeri rappresentabili a meno del segno e per esponente fissato. D'altra parte, l'esponente può assumere $U - L + 1$ valori e quindi, complessivamente, l'insieme $\mathbb{F}(\beta, t, L, U)$ è costituito da $2(\beta - 1)\beta^{t-1}(U - L + 1)$ elementi.

Soluzione 1.3 Per la formula di Eulero si ha $i = e^{i\pi/2}$ e quindi $i^i = e^{-\pi/2}$ che è un numero reale. In MATLAB

```
>> exp(-pi/2)
ans =
    0.2079
>> i^i
ans =
    0.2079
```

Soluzione 1.4 Si devono dare i comandi `L=2*eye(10)-3*diag(ones(8,1),-2)` e `U=2*eye(10)-3*diag(ones(8,1),2)`.

Soluzione 1.5 Per scambiare la terza con la settima riga della matrice triangolare inferiore costruita in precedenza, basta porre `r=[1:10]; r(3)=7; r(7)=3; Lr=L(r,:)`. Si noti l'uso dei due punti in `L(r,:)` che indica che tutte le colonne di `L` devono essere percorse nell'ordine usuale. Per scambiare l'ottava colonna con la quarta, basta invece porre: `c=[1:10]; c(8)=4; c(4)=8; Lc=L(:,c)`. Un analogo discorso vale per la matrice triangolare superiore.

`L(r,:)`

Soluzione 1.6 Si costruisce la matrice $A = [v1; v2; v3; v4]$ dove $v1, v2, v3$ e $v4$ sono i vettori riga MATLAB corrispondenti ai 4 vettori dati. Siccome $\det(A)=0$ i 4 vettori sono linearmente dipendenti.

Soluzione 1.7 Utilizziamo i seguenti comandi per introdurre l'espressione simbolica delle funzioni f e g :

```
>> syms x
>> f=sqrt(x^2+1); pretty(f)
(x^2+1)^1/2
>> g=sin(x^3)+cosh(x); pretty(g)
sin(x^3) + cosh(x)
```

pretty Si noti il comando **pretty** con il quale è possibile avere una versione più leggibile delle funzioni introdotte. A questo punto per calcolare le derivate prima e seconda, nonché l'integrale indefinito di f basta scrivere:

```
>> diff(f,x)
ans =
1/(x^2+1)^(1/2)*x
>> diff(f,x,2)
ans =
-1/(x^2+1)^(3/2)*x^2+1/(x^2+1)^(1/2)
>> int(f,x)
ans =
1/2*x*(x^2+1)^(1/2)+1/2*asinh(x)
```

Analoghe istruzioni valgono per la funzione g .

Soluzione 1.8 L'accuratezza delle radici calcolate degrada rapidamente al crescere del grado del polinomio. Questo risultato ci deve mettere in guardia sull'accuratezza del calcolo delle radici di un polinomio di grado elevato.

Soluzione 1.9 Una possibile implementazione è la seguente:

```
function l=successione(n)
l = zeros(n+2,1); l(1) = (exp(1)-1)/exp(1);
for i = 0:n, l(i+2) = 1 - (i+1)*l(i+1); end
return
```

Eseguendo questo programma con $n = 20$, si trova una successione di valori che diverge con segno alterno. Questo comportamento è legato all'accumulo degli errori di arrotondamento.

Soluzione 1.10 L'andamento anomalo è causato dagli errori di arrotondamento nel calcolo della sottrazione più interna. Si osservi inoltre che nel momento in cui $4^{1-n} z_n^2$ sarà minore di $\epsilon_M/2$, l'intera successione fornirà tutti elementi nulli. Ciò accade da $n = 29$ in poi.

Soluzione 1.11 Il metodo in esame è un esempio di metodi di tipo Monte Carlo. Un programma che lo implementa è il seguente:

```
function pig=pimontecarlo(n)
x=rand(n,1); y = rand(n,1);
m=sum(x.*x+y.*y <= 1);
pig=4*m/n;
return
```

Abbiamo fatto uso del comando `rand` per generare i numeri casuali. L'istruzione `sum(x.*x+y.*y <= 1)` significa: nella parentesi tonda viene fatto un controllo di verità (0 per falso, 1 per vero) sul fatto che le componenti dei vettori `x` e `y`, prese a coppie, stiano nel cerchio di centro l'origine e raggio 1. In questo modo si genera un vettore di 0 e 1, con gli 1 relativi a quelle copie di punti per i quali la proprietà precedente è vera. Il comando `sum` esegue quindi la somma di tutte le componenti del vettore generato: in questo modo contiamo il numero di punti che sono interni al quarto di cerchio considerato.

`sum`

Se si lancia il programma come `pig=pigreco(n)` per vari `n` si vedrà, al crescere di `n`, un lento miglioramento nell'approssimazione di π . Ad esempio, per `n=1000` la nostra simulazione fornisce `pig=3.1120`, mentre per `n=300000` otteniamo `pig=3.1406` (ovviamente essendo la generazione di numeri casuale, la ripetizione dell'esecuzione con lo stesso valore di `n` non fornirà necessariamente lo stesso risultato).

Soluzione 1.12 La seguente *function* risponde al quesito:

```
function pig=bbpalgorith(n)
pig = 0;
for m=0:n
    m8 = 8*m; pig = pig + (1/16)^m*(4/(m8+1)-(2/(m8+4)+1/(m8+5)+1/(m8+6)));
end
return
```

Per `n=10` si trova che il valore `pig` così calcolato coincide (nella precisione di MATLAB) con `pi`. In effetti, è estremamente efficiente e permette di calcolare rapidamente centinaia di cifre significative per π .

Soluzione 1.13 Il calcolo del coefficiente binomiale può essere fatto con il seguente programma (si veda anche la funzione `nchoosek`):

`nchoosek`

```
function bc=bincoeff(n,k)
k = fix(k); n = fix(n);
if k > n, disp('k deve essere compreso fra 0 e n'); break; end
if k > n/2, k = n-k; end
if k <= 1, bc = n^k; else
    num = (n-k+1):n; den = 1:k; el = num./den; bc = prod(el);
end
```

Il comando `fix(k)` elimina l'eventuale parte decimale della variabile `k`. Il comando `disp(messaggio)` visualizza il contenuto della stringa `messaggio` ed è un modo per far comparire messaggi durante l'esecuzione di un programma. Il comando `break` provoca l'interruzione dell'operazione in corso e quindi, nel

`fix`
`disp`

nostro caso, fa terminare il programma. Infine, il comando `prod(el)` calcola il prodotto di tutte le componenti del vettore `el`.

`break`

`prod`

Soluzione 1.14 Le *function* che seguono implementano il calcolo di f_n per ricorsione (`fibrec`) o usando la formula (1.13) (`fibmat`):

```
function f=fibrec(n)
if n == 0
    f = 0;
elseif n == 1
    f = 1;
else
    f = fibrec(n-1)+fibrec(n-2);
end
return

function f=fibmat(n)
f = [0;1];
A = [1 1; 1 0];
f = A^n*f;
f = f(1);
return
```

Per $n=20$ troviamo i seguenti risultati e tempi di calcolo:

```
>> t=cputime; fn=fibrec(20), cpu=cputime-t
fn =
       6765
cpu =
      1.3400
>> t=cputime; fn=fibonacci3(20), cpu=cputime-t
fn =
       6765
cpu =
      0
```

La *function* per ricorsione è quindi decisamente inefficiente rispetto all'altra che si limita a calcolare la potenza di una matrice (e che è, di conseguenza, più semplice da manipolare in MATLAB).

9.2 Capitolo 2

Soluzione 2.1 Utilizzando il comando `fplot` studiamo l'andamento della funzione data al variare di γ . Per $\gamma = 1$ essa non ammette zeri reali. Per $\gamma = 2$, si ha il solo zero $\alpha = 0$ con molteplicità 4 (cioè tale che $f(\alpha) = f'(\alpha) = f''(\alpha) = f'''(\alpha) = 0$, ma $f^{(4)}(\alpha) \neq 0$). Per $\gamma = 3$, f presenta due zeri semplici, uno nell'intervallo $(-3, -1)$, uno in $(1, 3)$. Nel caso $\gamma = 2$ il metodo di bisezione non può essere impiegato non essendo possibile trovare un intervallo per il quale $f(a)f(b) < 0$. Per $\gamma = 3$ a partire da $[a, b] = [-3, -1]$, il metodo di bisezione

(Programma 1) converge in 34 iterazioni allo zero $\alpha = -1.85792082914850$ (con $f(\alpha) \simeq -3.6 \cdot 10^{-12}$), se richiamato con le seguenti istruzioni:

```
>> f=inline('cosh(x)+cos(x)-3'); a=-3; b=-1; tol=1.e-10; nmax=200;
>> [zero,res,niter]=bisection(f,a,b,tol,nmax)
zero =
-1.8579
res =
-3.6877e-12
niter =
34
```

Analogamente per $\gamma = 3$ si ha convergenza in 34 iterazioni allo zero $\alpha = 1.8579$ con $f(\alpha) \simeq -3.68 \cdot 10^{-12}$.

Soluzione 2.2 Si tratta di calcolare gli zeri della funzione $f(V) = pV + aN^2/V - abN^3/V^2 - pNb - kNT$. Uno studio grafico rivela che questa funzione presenta un solo zero semplice fra 0.01 e 0.06 con $f(0.01) < 0$ e $f(0.06) > 0$. Calcoliamo tale zero come richiesto con il metodo di bisezione tramite le seguenti istruzioni:

```
>> f=inline('35000000*x+401000./x-17122.7./x.^2-1494500');
>> [zero,res,niter]=bisection(f,0.01,0.06,1.e-12,100)
zero =
0.0427
res =
-6.3814e-05
niter =
35
```

Soluzione 2.3 La funzione è data da $f(\omega) = s(1,\omega) - 1 = 9.8[\sinh(\omega) - \sin(\omega)]/(2\omega^2) - 1$. Dal grafico si deduce che ha uno zero tra 0.5 e 1. A partire da questo intervallo, il metodo di bisezione converge con l'accuratezza desiderata in 15 iterazioni alla radice $\omega = 0.61214447021484$.

Soluzione 2.4 La diseguaglianza (2.7) si ricava osservando che $|e^{(k)}| < |I^{(k)}|/2$ con $|I^{(k)}| < \frac{1}{2}|I^{(k-1)}| < 2^{-k-1}(b-a)$. Di conseguenza, l'errore è certamente minore di ε a partire da quel k_{min} tale che $2^{-k_{min}-1}(b-a) < \varepsilon$ cioè se $2^{-k_{min}-1} < \varepsilon/(b-a)$, da cui si ricava la (2.7).

Soluzione 2.5 La formula implementata è meno soggetta agli errori di cancellazione.

Soluzione 2.6 Rimandiamo alla Soluzione 2.1 per quanto riguarda l'analisi degli zeri della funzione. Iniziamo dal caso $\gamma = 2$: a partire da $x^{(0)} = 1$, il metodo di Newton (richiamato con il Programma 2) converge al valore $\bar{\alpha} = 0.0056$ in 18 iterazioni. Tale valore, evidentemente inaccurato (la radice di f è in tal caso $\alpha = 0$), può essere spiegato solo tenendo conto dell'andamento estremamente appiattito della funzione in un intorno della radice stessa. Anche prendendo

una tolleranza pari a 10^{-16} , si trova in 29 iterazioni $\bar{\alpha} = 2.21 \cdot 10^{-4}$. In effetti, si nota che $f(\bar{\alpha}) = 0$ in MATLAB. Per $\gamma = 3$, mantenendo la tolleranza pari a 10^{-16} , il metodo converge in 9 iterazioni al valore 1.85792082915020 a partire da $x^{(0)} = 1$, mentre a partire da $x^{(0)} = -1$ si ha convergenza in 10 iterazioni al valore -1.85792082915020 (in entrambi i casi il residuo è nullo in MATLAB).

Soluzione 2.7 Bisogna risolvere le equazioni $x^2 = a$ e $x^3 = a$, rispettivamente. Gli algoritmi cercati sono pertanto: dato $x^{(0)}$ calcolare,

$$x^{(k+1)} = \frac{1}{2} \left(x^{(k)} + \frac{a}{x^{(k)}} \right), \quad k \geq 0 \quad \text{per la radice quadrata,}$$

$$x^{(k+1)} = \frac{1}{3} \left(2x^{(k)} + \frac{a}{(x^{(k)})^2} \right), \quad k \geq 0 \quad \text{per la radice cubica.}$$

Soluzione 2.8 Poniamo $\delta x^{(k)} = x^{(k)} - \alpha$. Allora, sviluppando f in serie di Taylor in un intorno del punto $x^{(0)}$ si trova:

$$0 = f(\alpha) = f(x^{(k)}) - \delta x^{(k)} f'(x^{(k)}) + \frac{1}{2} (\delta x^{(k)})^2 f''(x^{(k)}) + \mathcal{O}((\delta x^{(k)})^3).$$

Sostituendo tale espressione nel metodo di Newton e sottraendo α da ambo i membri si ottiene

$$\delta x^{(k+1)} = \frac{1}{2} (\delta x^{(k)})^2 \frac{f''(x^{(k)})}{f'(x^{(k)})} + \mathcal{O}((\delta x^{(k)})^3).$$

Dopo aver diviso per $(\delta x^{(k)})^2$, passando al limite si trova la relazione cercata.

Soluzione 2.9 Si devono calcolare le radici della funzione (2.3) al variare di β . Iniziamo con l'osservare che tale equazione può ammettere due soluzioni per certi valori di β corrispondenti a due configurazioni possibili del sistema di aste. I due valori iniziali suggeriti nel testo dell'esercizio servono appunto per consentire al metodo di Newton di approssimare entrambe queste radici. Con i comandi riportati di seguito possiamo ottenere il risultato cercato, mostrato in Figura 9.1. Si suppone di risolvere il problema per i soli valori di β pari a $k\pi/100$ per $k = 0, \dots, 80$ (se β è maggiore di 2.6389 il metodo di Newton non converge in quanto il sistema non ammette configurazioni possibili).

```
>> a1=10; a2=13; a3=8; a4=10;
>> ss = num2str((a1^2 + a2^2 - a3^2 + a4^2)/(2*a2*a4),15);
>> n=100; x01=-0.1; x02=2*pi/3; nmax=100;
>> for i=0:80
    w = i*pi/n; k=i+1; beta(k) = w;
    ws = num2str(w,15);
    f = inline(['10/13*cos(',ws,')-cos(x)-cos(',ws,'-x)+',ss'],'x');
    df = inline(['sin(x)-sin(',ws,'-x)'],'x');
    [zero,res,niter]=newton(f,df,x01,1e-12,nmax);
    alpha1(k) = zero; niter1(k) = niter;
    [zero,res,niter]=newton(f,df,x02,1e-12,nmax);
    alpha2(k) = zero; niter2(k) = niter;
end
```

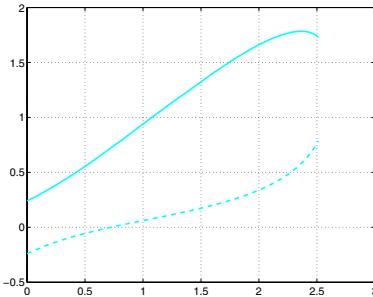


Figura 9.1. Le due curve rappresentano le due possibili configurazioni in corrispondenza al variare di β fra 0 e $4\pi/5$

Le componenti dei vettori `alpha1` e `alpha2` sono gli angoli calcolati in corrispondenza dei diversi valori di β , mentre quelle dei vettori `niter1` e `niter2` sono le iterazioni richieste dal metodo di Newton per soddisfare la tolleranza richiesta (tra le 5 e le 7).

Soluzione 2.10 La funzione data ammette due zeri positivi (α_2 e α_3 pari a circa 1.5 e 2.5) ed uno negativo (α_1 pari a circa -0.5). Il metodo di Newton a partire da $x^{(0)} = -0.5$ converge a α_1 in 4 iterazioni (`tol = 1.e-10`):

```
>> f=inline('exp(x)-2*x^2'); df=inline('exp(x)-4*x');
>> x0=-0.5; tol=1.e-10; nmax=100;
>> format long; [zero,res,niter]=newton(f,df,x0,tol,nmax)
zero =
-0.53983527690282
res =
0
niter =
4
```

La funzione in esame ammette un punto di massimo in $\bar{x} \simeq 0.3574$ (calcolato usando nuovamente il metodo di Newton per la funzione f'): per $x^{(0)} < \bar{x}$ il metodo converge sempre allo zero negativo. Se $x^{(0)} = \bar{x}$ il metodo non può essere applicato in quanto $f'(\bar{x}) = 0$. Per $x^{(0)} > \bar{x}$ il metodo converge alla radice $\alpha_3 = 2.61786661306681$.

Soluzione 2.11 Poniamo $x^{(0)} = 0$ e $\text{tol} = 10^{-17}$. Il metodo di Newton converge in 39 iterazioni al valore 0.64118239763649. Se utilizziamo questa approssimazione di α per studiare l'andamento dell'errore scopriamo che esso decresce solo linearmente al crescere delle iterazioni. Il motivo è legato al fatto che lo zero cercato è uno zero multiplo per f (si veda la Figura 9.2). Si potrebbe in tal caso usare il metodo di Newton modificato.

Soluzione 2.12 Il problema consiste nel trovare lo zero della funzione $f(x) = \sin(x) - \sqrt{2gh/v_0^2}$. Per questioni di simmetria possiamo restringere il nostro

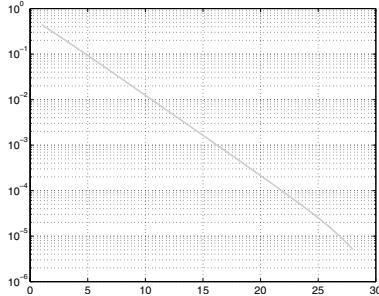


Figura 9.2. Andamento dell'errore nel metodo di Newton per il calcolo dello zero di $f(x) = x^3 - 3x^22^{-x} + 3x4^{-x} - 8^{-x}$

studio all'intervallo $(0, \pi/2)$. Il metodo di Newton con $x^{(0)} = \pi/4$, $\text{tol} = 10^{-10}$ converge in 5 iterazioni alla radice 0.45862863227859.

Soluzione 2.13 Con i dati indicati, la soluzione dell'esercizio passa attraverso le seguenti istruzioni:

```
>> f=inline('6000-1000*(1+x).*((1+x).^5 - 1)./x');
>> df=inline('1000*((1+x).^5*(1-5*x) - 1)./(x.^2)');
>> [zero,res,niter]=bisection(f,0.01,0.1,1.e-12,4);
>> [zero,res,niter]=newton(f,df,zero,1.e-12,100);
```

Il metodo di Newton converge al risultato cercato in 3 iterazioni.

Soluzione 2.14 Il grafico della funzione mostra che la (2.33) ha uno zero in $(\pi/6, \pi/4)$. Il metodo di Newton, richiamato con le seguenti istruzioni:

```
>> f=inline('-l2*cos(g+a)/sin(g+a)^2-l1*cos(a)/sin(a)^2','a','g','l1','l2');
>> df=inline('l2/sin(g+a)+2*l2*cos(g+a)^2/sin(g+a)^3+...
l1/sin(a)+2*l1*cos(a)^2/sin(a)^3','a','g','l1','l2');
>> [zero,res,niter]=newton(f,df,pi/4,1.e-15,100,3*pi/5,8,10);
```

converge al valore cercato, 0.596279927465474, in 6 iterazioni a partire da $x^{(0)} = \pi/4$. La lunghezza massima sarà allora pari a $L = 30.84$.

Soluzione 2.15 Se α è uno zero di molteplicità m per f , allora esiste una funzione h tale che $h(\alpha) \neq 0$ e $f(x) = h(x)(x - \alpha)^m$. Calcolando la derivata prima della funzione di iterazione del metodo di Newton, ϕ_N , si ha

$$\phi'_N(x) = 1 - \frac{[f'(x)]^2 - f(x)f''(x)}{[f'(x)]^2} = \frac{f(x)f''(x)}{[f'(x)]^2}.$$

Sostituendo a f , f' e f'' le corrispondenti espressioni in funzione di $h(x)$ e di $(x - \alpha)^m$, si ottiene $\lim_{x \rightarrow \alpha} \phi'_N(x) = 1 - 1/m$, da cui $\phi'_N(\alpha) = 0$ se e soltanto se $m = 1$. Di conseguenza, se $m = 1$ il metodo converge almeno quadraticamente per la (2.10). Se invece $m > 1$ il metodo converge con ordine 1 per la Proposizione 2.1.

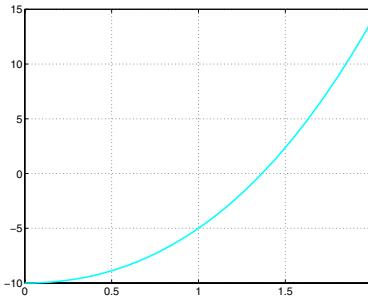


Figura 9.3. Grafico di $f(x) = x^3 + 4x^2 - 10$ per $x \in [0, 2]$

Soluzione 2.16 Da uno studio grafico, effettuato con i comandi:

```
>> f= 'x^3+4*x^2-10'; fplot(f,[-10,10]); grid on;
>> fplot(f,[-5,5]); grid on;
>> fplot(f,[0,5]); grid on
```

si ricava che f ammette un solo zero reale pari a circa 1.36 (in Figura 9.3) viene riportato l'ultimo grafico ottenuto). La funzione di iterazione e la sua derivata sono:

$$\begin{aligned}\phi(x) &= \frac{2x^3 + 4x^2 + 10}{3x^2 + 8x} = -\frac{f(x)}{3x^2 + 8x} + x, \\ \phi'(x) &= \frac{(6x^2 + 8x)(3x^2 + 8x) - (6x + 8)(2x^3 + 4x^2 + 10)}{(3x^2 + 8x)^2},\end{aligned}$$

e $\phi(\alpha) = \alpha$. Sostituendo il valore di α , si ricava $\phi'(\alpha) = 0$ in quanto $\phi'(x) = (6x + 8)f(x)/(3x^2 + 8x)^2$. Di conseguenza, il metodo proposto è convergente con ordine 2.

Soluzione 2.17 Il metodo in esame è almeno del second'ordine in quanto $\phi'(\alpha) = 0$.

Soluzione 2.18 Mantenendo invariati i restanti parametri, si trova che il metodo ora converge in 3 sole iterazioni allo zero 0.64118573649623 con una discrepanza dell'ordine di 10^{-9} sul risultato calcolato nella Soluzione 2.11. L'andamento della funzione, estremamente schiacciato in prossimità dello zero, ci autorizza però a ritenere maggiormente accurato il valore calcolato in precedenza. In Figura 9.4, riportiamo il grafico di f in $(0.5, 0.7)$ ottenuto con i seguenti comandi:

```
>> f='x^3-3*x^2*2^(-x) + 3*x*4^(-x) - 8^(-x)';
>> fplot(f,[0.5 0.7]); grid on
```

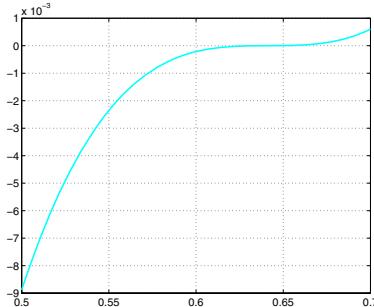


Figura 9.4. Grafico di $f(x) = x^3 - 3x^2 2^{-x} + 3x 4^{-x} - 8^{-x}$ per $x \in [0.5, 0.7]$

9.3 Capitolo 3

Soluzione 3.1 Osserviamo che se $x \in (x_0, x_n)$ allora deve esistere un intervallo $I_i = (x_{i-1}, x_i)$ tale che $x \in I_i$. Si ricava facilmente che il $\max_{x \in I_i} |(x - x_{i-1})(x - x_i)| = h^2/4$. Se ora maggioriamo $|x - x_{i+1}|$ con $2h$, $|x - x_{i+2}|$ con $3h$ e così via, troviamo la stima (3.6).

Soluzione 3.2 Essendo $n = 4$ in tutti i casi, dovremo maggiorare la derivata quinta di ciascuna funzione sugli intervalli dati. Si trova: $\max_{x \in [-1, 1]} |f_1^{(5)}| = 1.18$; $\max_{x \in [-1, 1]} |f_2^{(5)}| = 1.54$; $\max_{x \in [-\pi/2, \pi/2]} |f_3^{(5)}| = 1.41$. Gli errori corrispondenti sono allora di 0.0018, 0.0024 e 0.0211, rispettivamente.

Soluzione 3.3 Tramite il comando `polyfit` si calcolano i polinomi interpolatori di grado 3 nei due casi:

```
>> anni=[1975 1980 1985 1990];
>> eor=[70.2 70.2 70.3 71.2];
>> eoc=[72.8 74.2 75.2 76.4];
>> cor=polyfit(anni, eor, 3);
>> coc=polyfit(anni, eoc, 3);
>> stimaor=polyval(cor, [1970 1983 1988 1995])
stimaor =
    69.6000 70.2032 70.6992 73.6000
>> stimaoc=polyval(coc, [1970 1983 1988 1995])
stimaoc =
    70.4000 74.8096 75.8576 78.4000
```

Per l'Europa occidentale l'aspettativa di vita stimata per il 1970 risulta dunque pari a 70.4 anni (`stimaoc(1)`), con una sottostima di 1.4 anni rispetto al valore noto di 71.8 anni. Vista la simmetria del grafico del polinomio interpolatore ci si può aspettare che la previsione per il 1995, pari a 78.4 anni, sia sovrastimata della medesima quantità (l'aspettativa reale è in realtà leggermente inferiore e pari a 77.5 anni). Non vale lo stesso ragionamento per l'altro insieme di dati: in tal caso infatti l'estrapolato relativo al 1970 coincide addirittura con il valore reale, mentre il valore previsto per il 1995 è ampiamente sovrastimato (73.6 anni contro 71.2 anni).

Soluzione 3.4 Supponiamo di utilizzare come unità di tempo il mese a partire da $t_0 = 1$ in corrispondenza del mese di novembre del 1986, fino a $t_7 = 157$, corrispondente al mese di novembre del 1999. Dando i seguenti comandi:

```
>> tempo = [1 14 37 63 87 99 109 157];
>> prezzo = [4.5 5 6 6.5 7 7.5 8 8];
>> [c] = polyfit(tempo,prezzo,7);
```

calcoliamo i coefficienti del polinomio interpolatore. Ponendo `[prezzo2000]=polyval(c,180)` si trova che il prezzo previsto a novembre del 2001 è di circa 10.8 euro.

Soluzione 3.5 La spline cubica interpolatoria calcolata con il comando `spline` coincide con il polinomio interpolatore: esso in tal caso è infatti la spline stessa in quanto interpola i dati, ha derivate prima e seconda continua, ha derivata terza continua nei nodi x_1 e x_2 (per la condizione *not-a-knot*). Si sarebbe trovato un risultato diverso utilizzando una spline cubica interpolatoria naturale.

Soluzione 3.6 Basta scrivere le seguenti istruzioni:

```
>> T = [4:4:20];
>> rho=[1000.7794,1000.6427,1000.2805,999.7165,998.9700];
>> Tnew = [6:4:18]; format long e;
>> rhonew = spline(T,rho,Tnew)
rhonew =
Columns 1 through 3
    1.000740787500000e+03    1.000488237500000e+03    1.000022450000000e+03
Column 4
    9.993649250000000e+02
```

Il confronto con le nuove misure consente di affermare che l'approssimazione considerata è estremamente accurata. Si noti che l'equazione di stato internazionale per l'acqua marina (UNESCO, 1980) postula una dipendenza del quart'ordine fra densità e temperatura; tuttavia, approssimando la densità con una spline cubica, si ottiene una buona corrispondenza con i valori reali in quanto il coefficiente relativo alla potenza quarta di T è dell'ordine di 10^{-9} .

Soluzione 3.7 Confrontiamo tra loro i risultati ottenuti usando la spline cubica interpolatoria generata dal comando `spline` di MATLAB (che indicheremo con `s3`), quella naturale (`s3n`) e quella con derivata prima nulla agli estremi (`s3d`) (ottenuta con il Programma 7). Basta scrivere i seguenti comandi:

```
>> anno=[1965 1970 1980 1985 1990 1991];
>> produzione=[17769 24001 25961 34336 29036 33417];
>> z=[1962:0.1:1992];
>> s3 = spline(anno,produzione,z);
>> s3n = cubicpline(anno,produzione,z);
>> s3d = cubicpline(anno,produzione,z,0,[0 0]);
```

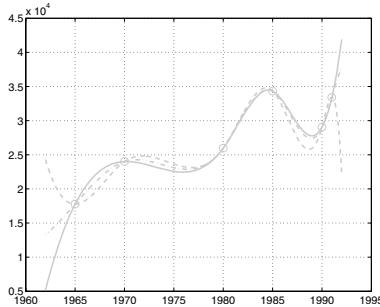


Figura 9.5. Confronto fra i grafici delle *splines* cubiche generate nel corso della Soluzione 3.7: s_3 in linea continua, s_{3d} in linea tratteggiata, s_{3n} in linea tratto-punto. I cerchietti rappresentano i valori interpolati

Nella tabella seguente riportiamo i valori ottenuti (in migliaia di quintali di agrumi):

anno	1962	1977	1992
$s_3 (\times 10^3 \text{ Kg})$	5146.1	22641.8	41894.4
$s_{3n} (\times 10^3 \text{ Kg})$	13285.3	22934.2	37798.0
$s_{3d} (\times 10^3 \text{ Kg})$	24313.0	23126.0	22165.8

Il confronto con i dati effettivamente misurati (12380, 27403 e 32059) mostra come la spline naturale sia in questo caso la più affidabile al di fuori degli estremi dell'intervallo di interpolazione (si veda anche la Figura 9.5). Il polinomio interpolatore di Lagrange si dimostra decisamente meno affidabile: presenta un andamento molto oscillante e fornisce per il 1962 una previsione di produzione pari a -77685 migliaia di quintali di agrumi.

Soluzione 3.8 Per ricavare il polinomio interpolatore p e la spline s_3 , basta scrivere le seguenti istruzioni:

```
>> pert = 1.e-04;
>> x=[-1:2/20:1]; y=sin(2*pi*x)+(-1).^(1:21)*pert; z=[-1:0.01:1];
>> c=polyfit(x,y,20); p=polyval(c,z); s3=spline(x,y,z);
```

avendo cura di porre $\text{pert}=0$ nel caso in cui si vogliano usare i dati non perturbati (abbiamo valutato il polinomio e la spline in 101 punti equispaziati). Con le valutazioni esatte $P_{20}f$ e s_3 non sono distinguibili, per lo meno a livello grafico, dalla funzione f . La situazione cambia drasticamente se si usano i dati perturbati; mentre infatti la spline si mantiene sostanzialmente immutata, il polinomio interpolatore presenta delle forti oscillazioni agli estremi dell'intervallo (si veda la Figura 9.6). L'approssimazione con funzioni spline è dunque più stabile, ovvero meno sensibile alle piccole perturbazioni, di quanto non lo sia l'interpolazione polinomiale di Lagrange.

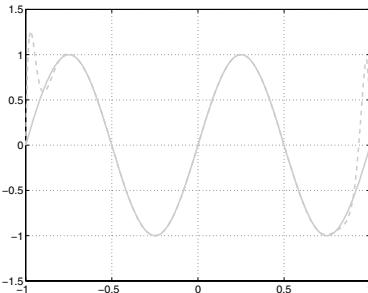


Figura 9.6. Il polinomio interpolatore (in linea tratteggiata) e la spline cubica interpolatoria (in linea piena) a confronto nel caso in cui si usino dati perturbati. Si noti lo scollamento fra i due grafici agli estremi dell'intervallo

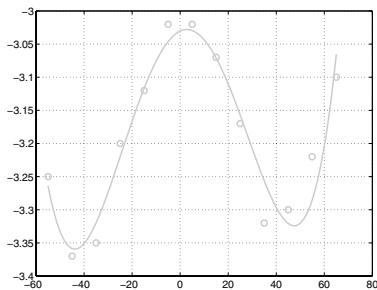


Figura 9.7. Polinomio di grado 4 dei minimi quadrati (in linea piena) a confronto con i dati della colonna di Tabella 3.1 per $K = 0.67$

Soluzione 3.9 Se $n = m$, ponendo $\tilde{f} = \Pi_n f$ si ha addirittura 0 a primo membro della (3.20) e quindi $\Pi_n f$ è soluzione del problema dei minimi quadrati. Essendo il polinomio interpolatore unico, si deduce che questa è l'unica soluzione del problema dei minimi quadrati.

Soluzione 3.10 I polinomi hanno i seguenti coefficienti (riportati con le sole prime 4 cifre significative ed ottenuti con il comando `polyfit`):

$$\begin{aligned} K &= 0.67, a_4 = 6.301 \cdot 10^{-8}, a_3 = -8.320 \cdot 10^{-8}, a_2 = -2.850 \cdot 10^{-4}, a_1 = \\ &\quad 9.718 \cdot 10^{-4}, a_0 = -3.032; \\ K &= 1.5, a_4 = -4.225 \cdot 10^{-8}, a_3 = -2.066 \cdot 10^{-6}, a_2 = 3.444 \cdot 10^{-4}, a_1 = \\ &\quad 3.364 \cdot 10^{-3}, a_0 = 3.364; \\ K &= 2, a_4 = -1.012 \cdot 10^{-7}, a_3 = -1.431 \cdot 10^{-7}, a_2 = 6.988 \cdot 10^{-4}, a_1 = \\ &\quad -1.060 \cdot 10^{-4}, a_0 = 4.927; \\ K &= 3, a_4 = -2.323 \cdot 10^{-7}, a_3 = 7.980 \cdot 10^{-7}, a_2 = 1.420 \cdot 10^{-3}, a_1 = \\ &\quad -2.605 \cdot 10^{-3}, a_0 = 7.315. \end{aligned}$$

In Figura 9.7 riportiamo il polinomio ottenuto per i dati della colonna relativa a $K = 0.67$ nella Tabella 3.1.

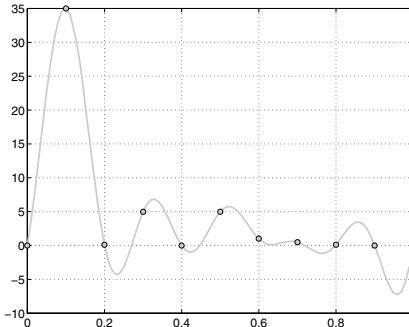


Figura 3.8. L'approssimante trigonometrico ottenuto con le istruzioni della Soluzione 3.14. I pallini si riferiscono ai dati sperimentali disponibili

Soluzione 3.11 Ripetendo le prime 3 istruzioni della Soluzione 3.7 e richiamando il comando `polyfit`, si trovano i seguenti valori (in migliaia di quintali di arance): 1962, 15280.12; 1977, 27407.10; 1992, 32019.01. Essi sono delle ottime approssimazioni dei valori effettivamente misurati (12380, 27403 e 32059 rispettivamente).

Soluzione 3.12 La varianza può essere riscritta come $v = \frac{1}{n+1} \sum_{i=0}^n x_i^2 - M^2$. Ed è proprio utilizzando questa espressione che è possibile scrivere i coefficienti del sistema (3.22) in termini di media e di varianza.

Soluzione 3.13 La proprietà cercata si ricava immediatamente dalla prima equazione del sistema (3.22).

Soluzione 3.14 È sufficiente utilizzare il comando `interpft` come segue:

```
>> discharge = [0 35 0.125 5 0 5 1 0.5 0.125 0];
>> y = interpft(discharge,100);
```

Il grafico della soluzione ottenuta è riportato in Figura 3.14.

9.4 Capitolo 4

Soluzione 4.1 Verifichiamo l'ordine della formula relativa a x_0 (per quella relativa a x_n si eseguono calcoli del tutto analoghi). Sviluppando in serie di Taylor $f(x_1)$ e $f(x_2)$ rispetto a x_0 , troviamo

$$\begin{aligned} f(x_1) &= f(x_0) + hf'(x_0) + \frac{h^2}{2}f''(x_0) + \frac{h^3}{6}f'''(\xi_1), \\ f(x_2) &= f(x_0) + 2hf'(x_0) + 2h^2f''(x_0) + \frac{4h^3}{3}f'''(\xi_2), \end{aligned}$$

dove $\xi_1 \in (x_0, x_1)$ e $\xi_2 \in (x_0, x_2)$. Sostituendo queste espressioni nella prima delle (4.11), si trova:

$$\frac{1}{2h} [-3f(x_0) + 4f(x_1) - f(x_2)] = f'(x_0) + \frac{h^2}{3} [f'''(\xi_1) - 2f'''(\xi_2)],$$

da cui il risultato cercato per un opportuno ξ_0 .

Soluzione 4.2 Sviluppiamo $f(\bar{x} \pm h)$ in serie di Taylor in avanti e all'indietro rispetto al punto \bar{x} , troncandone lo sviluppo al terz'ordine. Avremo:

$$f(\bar{x} \pm h) = f(\bar{x}) \pm hf'(\bar{x}) + \frac{h^2}{2}f''(\bar{x}) \pm \frac{h^3}{6}f'''(\xi_{\pm}),$$

con $\xi_- \in (\bar{x} - h, \bar{x})$ e $\xi_+ \in (\bar{x}, \bar{x} + h)$. Se sottraiamo $f(\bar{x} - h)$ da $f(\bar{x} + h)$, troviamo

$$f(\bar{x} + h) - f(\bar{x} - h) = 2hf'(\bar{x}) + \frac{h^3}{6}(f'''(\xi_+) + f'''(\xi_-)),$$

da cui, dividendo per h ed isolando $f'(\bar{x})$, si conclude che la (4.9) è una approssimazione di ordine 2 della derivata prima e si ritrova la formula (4.10).

Soluzione 4.3 Eseguendo operazioni analoghe a quelle indicate nella Soluzione 4.1, si trovano i seguenti errori (supponendo $f \in C^4$):

$$a. -\frac{1}{4}f^{(4)}(\xi)h^3, b. -\frac{1}{12}f^{(4)}(\xi)h^3, c. \frac{1}{6}f^{(4)}(\xi)h^3.$$

Soluzione 4.4 Usiamo l'approssimazione (4.9). Si trovano i seguenti valori

t (mesi)	0	0.5	1	1.5	2	2.5	3
δn	--	78	45	19	7	3	--
n'	--	77.91	39.16	15.36	5.91	1.99	--

che, come risulta dal confronto con i valori esatti di $n'(t)$ calcolati negli stessi istanti, sono abbastanza accurati.

Soluzione 4.5 L'errore di quadratura commesso con la formula composita del punto medio può essere maggiorato con

$$(b-a)^3/(24M^2) \max_{x \in [a,b]} |f''(x)|,$$

essendo $[a, b]$ l'intervallo di integrazione e M il numero (incognito) di intervalli.

La funzione f_1 è derivabile con continuità per ogni ordine. Con uno studio grafico, si deduce che $|f_1''(x)| \leq 2$ nell'intervallo considerato. Dunque affinché l'errore sia minore di 10^{-4} si dovrà avere $25^3/(24M^2) < 10^{-4}$ cioè $M > 322$.

Anche la funzione f_2 è derivabile per ogni ordine. Con un semplice studio si ricava che $\max_{x \in [0, \pi]} |f_2''(x)| = \sqrt{2}e^{3/4\pi}$; di conseguenza, affinché l'errore sia minore di 10^{-4} dovrà essere $M > 439$. Si noti che le stime ottenute maggiorano ampiamente l'errore e , di conseguenza, il numero minimo di intervalli che garantisce un errore inferiore alla tolleranza fissata è assai minore (ad esempio, per f_1 bastano soltanto 51 intervalli). La funzione f_3 non ha derivata prima definita in $x = 0$ e $x = 1$: non si può quindi applicare la stima dell'errore riportata in quanto $f_3 \notin C^2([0, 1])$.

Soluzione 4.6 Su ciascun intervallo I_k , $k = 1, \dots, M$, si commette un errore pari a $H^3/24f''(\xi_k)$ con $\xi_k \in [x_{k-1}, x_k]$. Di conseguenza, l'errore totale sarà dato da $H^3/24 \sum_{k=1}^M f''(\xi_k)$. Essendo f'' continua in $[a, b]$ esiste un punto $\xi \in [a, b]$ tale che $f''(\xi) = \frac{1}{M} \sum_{k=1}^M f''(\xi_k)$. Usando tale risultato e ricordando che $MH = b - a$ si ricava immediatamente la (4.14).

Soluzione 4.7 È legata all'accumulo degli errori che si commettono su ciascun sottointervallo.

Soluzione 4.8 La formula del punto medio integra per definizione in modo esatto le costanti. Per controllare che integri esattamente anche i polinomi di grado 1, basta verificare che $I(x) = I_{PM}(x)$. Abbiamo in effetti:

$$I(x) = \int_a^b x \, dx = \frac{b^2 - a^2}{2}, \quad I_{PM}(x) = (b - a) \frac{b + a}{2}.$$

Soluzione 4.9 Per la funzione f_1 si trova $M = 71$ se si usa la formula del trapezio e $M = 7$ per la formula di Gauss-Legendre con $n = 1$ (per questa formula si può usare il Programma 27). Come si vede il vantaggio nell'uso di quest'ultima formula è estremamente rilevante.



Programma 27 - gausslegendre : formula di quadratura di Gauss-Legendre con $n = 1$

```
function intGL=gausslegendre(a,b,f,N,varargin)
y = [-1/sqrt(3),1/sqrt(3)];
H2 = (b-a)/(2*N);
z = [a:2*H2:b];
zM = (z(1:end-1)+z(2:end))*0.5;
x = [zM+H2*y(1), zM+H2*y(2)];
f = feval(f,x,varargin{:});
intGL = H2*sum(f);
return
```

Soluzione 4.10 Dalla (4.18) sappiamo che l'errore di quadratura per la formula composita del trapezio con $H = H_1$ è pari a CH_1^2 con $C = -\frac{b-a}{12}f''(\xi)$. Se f'' non varia molto, possiamo pensare che anche l'errore per $H = H_2$ sia ancora della forma CH_2^2 . Allora, sottraendo le espressioni

$$I(f) = I_1 + CH_1^2, \quad I(f) = I_2 + CH_2^2, \tag{9.1}$$

possiamo calcolare la costante C come

$$C = \frac{I_1 - I_2}{H_2^2 - H_1^2}$$

e sostituendo tale valore in una qualsiasi delle (9.1) troviamo la (4.32).

Soluzione 4.11 Imponiamo che $I_{approx}(x^p) = I(x^p)$ per $p \geq 0$. Troviamo il seguente sistema di equazioni non lineari nelle incognite α , β , \bar{x} e \bar{z} :

$$\begin{aligned} p = 0 &\rightarrow \alpha + \beta = b - a, \\ p = 1 &\rightarrow \alpha\bar{x} + \beta\bar{z} = \frac{b^2 - a^2}{2}, \\ p = 2 &\rightarrow \alpha\bar{x}^2 + \beta\bar{z}^2 = \frac{b^3 - a^3}{3}, \\ p = 3 &\rightarrow \alpha\bar{x}^3 + \beta\bar{z}^3 = \frac{b^4 - a^4}{4}. \end{aligned}$$

Ci siamo arrestati a $p = 3$ avendo ottenuto un sistema a 4 incognite e 4 equazioni. Se si ricavano dalle prime due equazioni α e \bar{z} e si sostituiscono nelle ultime due, si trova un sistema non lineare nelle sole β e \bar{x} . A questo punto, risolvendo un'equazione di secondo grado in β , si ricava β in funzione di \bar{x} e si perviene ad un'equazione non lineare nella sola \bar{x} . Utilizzando ad esempio il metodo di Newton per la sua risoluzione, si trovano per \bar{x} due possibili valori che coincidono proprio con le ascisse dei nodi di quadratura di Gauss-Legendre per $n = 1$.

Soluzione 4.12 Calcoliamo il massimo (M_1 e M_2) del valore assoluto della derivata quarta delle funzioni date sugli intervalli assegnati. Abbiamo:

$$\begin{aligned} f_1^{(4)}(x) &= \frac{24}{(1 + (x - \pi)^2)^5(2x - 2\pi)^4} - \frac{72}{(1 + (x - \pi)^2)^4(2x - 2\pi)^2} \\ &\quad + \frac{24}{(1 + (x - \pi)^2)^3}, & M_1 &\simeq 25, \\ f_2^{(4)}(x) &= -4e^x \cos(x), & M_2 &\simeq 93. \end{aligned}$$

Di conseguenza, per la (4.22) si trova nel primo caso $H < 0.21$ e nel secondo $H < 0.16$.

Soluzione 4.13 Utilizzando MATLAB con i comandi `int('exp(-x^2/2)', 0, 2)` ricaviamo che l'integrale in questione vale circa 1.19628801332261. Il calcolo con la formula di Gauss-Legendre implementata nel Programma 27 fornisce il valore 1.20278027622354 (con un errore assoluto pari a 6.4923e-03), mentre per la formula di Simpson si ha 1.18715264069572 con un errore assoluto pari a -9.1354e-03.

Soluzione 4.14 Si noti che, essendo la funzione integranda non negativa, allora $I_k > 0 \forall k$. La formula ricorsiva proposta risulta però instabile a causa degli errori di arrotondamento come si vede dando i seguenti comandi MATLAB:

```
>> I(1)=1/exp(1); for k=2:20, I(k)=1-k*I(k-1); end
```

In effetti, $I(20) = -30.1924$. Utilizzando la formula di Simpson con $H < 0.25$ si ottiene l'accuratezza richiesta.

Soluzione 4.15 L'idea dell'estrapolazione di Richardson è generale e può dunque essere applicata ad una qualunque formula di quadratura. Basta prestare attenzione all'ordine di accuratezza della formula. In particolare, per la formula di Simpson e per quella di Gauss (entrambe accurate con ordine 4) la (4.32) diventerà:

$$I_R = I_1 + (I_1 - I_2)/(H_2^4/H_1^4 - 1).$$

Per la formula di Simpson si trovano i seguenti valori:

$$I_1 = 1.19616568040561, \quad I_2 = 1.19628173356793, \quad I_R = 1.19628947044542,$$

con un errore $I(f) - I_R = -1.4571e - 06$ inferiore di due ordini di grandezza rispetto a I_1 e di un fattore 1/4 rispetto ad I_2 . Per la formula di Gauss-Legendre si trovano invece i seguenti valori (tra parentesi vengono riportati gli errori commessi):

$$\begin{aligned} I_1 &= 1.19637085545393 \quad (-8.2842e - 05), \\ I_2 &= 1.19629221796844 \quad (-4.2046e - 06), \\ I_R &= 1.19628697546941 \quad (1.0379e - 06). \end{aligned}$$

Anche in questo caso è evidente il vantaggio dell'estrapolazione di Richardson.

Soluzione 4.16 Per stimare l'errore dobbiamo calcolare la derivata quarta della funzione integranda. Si trova $f^{(4)}(x) = e^x(x^2 + 8x + 12)$. Essendo una funzione monotona crescente nell'intervallo $[0, 1]$, assumerà il massimo sempre nel secondo estremo di integrazione. Affinché l'errore sia minore di 10^{-10} si dovrà allora richiedere che $H^4 < 10^{-10}2880/(r f^{(4)}(r))$. Il numero di sottointervalli M necessari per verificare tale diseguaglianza è allora dato, al variare di $r = k/10$ con $k = 1, \dots, 10$, da:

```
>> x=[0.1:0.1:1]; f4=exp(x).*(x.^2+8*x+12);
>> H=(10^(-10)*2880./(.^*f4)).^(1/4); M=fix(x./H)
M =
    4    11    20    30    41    53    67    83   100   118
```

I valori di $j(r)$ sono allora:

```
>> sigma=0.36; epsilon0 = 8.859e-12;
for k = 1:10
    r = k/10;
    j(k)=simpsonc(0,r,M(k),'exp(x).*x.^2');
    j(k) = j(k)*sigma/r^2*epsilon0;
end
```

Soluzione 4.17 Calcoliamo $E(213)$ con la formula di Simpson composita facendo crescere il numero di intervalli finché la differenza fra due approssimazioni successive (divisa per l'ultimo valore calcolato) non è inferiore a 10^{-11} :

```

>> f='2.39e-11./((x.^5).*(exp(1.432./(213*x))-1))';
>> a=3.e-04; b=14.e-04;
>> i=1; err = 1; lold = 0; while err >= 1.e-11
l=simpsonc(a,b,i,f);
err = abs(l-lold)/abs(l);
lold=l;
i=i+1;
end

```

Il ciclo si conclude per $i = 59$. Servono perciò 58 intervalli equispaziati per ottenere l'integrale $E(213)$ accurato fino alla decima cifra significativa. Qualora si usi la formula di Gauss-Legendre serviranno invece 53 intervalli.

Soluzione 4.18 Globalmente la funzione data non ha la regolarità richiesta per poter controllare l'errore con nessuna delle formule proposte. L'idea risolutiva consiste nell'applicare la formula di Simpson composita in ciascuno dei due sottointervalli $[0, 0.5]$ e $[0.5, 1]$ al cui interno la funzione data viene addirittura integrata esattamente (essendo un polinomio di grado 3).

9.5 Capitolo 5

Soluzione 5.1 Indichiamo con x_n il numero di operazioni (somme, sottrazioni e moltiplicazioni) richiesto per il calcolo di un determinante di una matrice $n \times n$ con la regola di Laplace. Allora

$$\begin{aligned} x_n &= nx_{n-1} + 2n - 1, \quad n \geq 3, \\ x_2 &= 3. \end{aligned}$$

Con qualche manipolazione algebrica si trova che

$$x_n = \frac{n!}{(n-s)!} x_{n-s} + \sum_{j=1}^s \frac{n!}{(n-j)!} + \frac{n!}{(n-s)!} - 1.$$

Valutando questa espressione per $s = n - 2$, si trova il risultato cercato:

$$x_n = 2n! + \sum_{j=1}^{n-2} \frac{n!}{(n-j)!} - 1.$$

Soluzione 5.2 Utilizziamo i seguenti comandi MATLAB per calcolare i determinanti ed i tempi di CPU necessari:

```

>> t = [];
for i = 3:500
    A = magic(i); tt = cputime; d=det(A); t=[t, cputime-tt];
end

```

Calcoliamo i coefficienti del polinomio dei minimi quadrati di grado 3 che approssima i dati $n=[3:500]$ e t . Troviamo:

```
>> format long; c=polyfit(n,t,3)
c =
0.00000002102187 0.00000171915661 -0.00039318949610 0.01055682398911
```

Il primo coefficiente è piccolo (quello relativo a n^3), ma non trascurabile rispetto al secondo. Se calcoliamo i coefficienti del polinomio di grado 4, otteniamo i seguenti valori:

```
>> c=polyfit(i,t,4)
c =
Columns 1 through 4
-0.00000000000051 0.00000002153039 0.00000155418071 -0.00037453657810
Column 5
0.01006704351509
```

Stavolta, il primo coefficiente è trascurabile rispetto al secondo e quindi possiamo concludere che il tempo di calcolo si comporta approssimativamente come n^3 .

Soluzione 5.3 Si trova: $\det A_1 = 1$, $\det A_2 = \varepsilon$, $\det A_3 = \det A = 2\varepsilon + 12$. Di conseguenza, se $\varepsilon = 0$ la seconda sottomatrice principale è singolare e la Proposizione 5.1 non è più valida. La matrice è singolare se $\varepsilon = -6$: in tal caso la fattorizzazione di Gauss può comunque essere portata a termine e si trova

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 1.25 & 1 \end{bmatrix}, U = \begin{bmatrix} 1 & 7 & 3 \\ 0 & -12 & -4 \\ 0 & 0 & 0 \end{bmatrix}.$$

Tuttavia, poiché U è singolare (com'era del resto da attendersi essendo A singolare), il sistema triangolare superiore $Ux = y$ ammette infinite soluzioni. Si osservi anche che il metodo delle sostituzioni all'indietro (5.8) non può essere utilizzato.

Soluzione 5.4 Dall'analisi dell'algoritmo (5.11) si ottiene che il numero di operazioni per il calcolo di L ed U è:

$$\begin{aligned} & \sum_{k=1}^{n-1} \sum_{i=k+1}^n \left(1 + \sum_{j=k+1}^n 2 \right) = \sum_{k=1}^{n-1} (n-k)(1+2(n-k)) \\ &= \sum_{j=1}^{n-1} j + 2 \sum_{j=1}^{n-1} j^2 = \frac{(n-1)n}{2} + \frac{(n-1)n(2n-1)}{6} = \frac{2}{3}n^3 - \frac{n^2}{2} - \frac{n}{6}. \end{aligned}$$

Soluzione 5.5 Per definizione, l'inversa X di una matrice $A \in \mathbb{R}^{n \times n}$ è tale che $XA = AX = I$. Di conseguenza, per ogni $j = 1, \dots, n$ il vettore colonna y_j di X risolve il sistema lineare $Ay_j = e_j$ il cui termine noto è il j -esimo vettore della base canonica di \mathbb{R}^n con componenti tutte nulle fuorché la j -esima che vale 1. Nota perciò una fattorizzazione LU di A , si tratterà di risolvere n sistemi lineari con la stessa matrice e termine noto variabile.

Soluzione 5.6 Utilizzando il Programma 11 si trovano i seguenti fattori:

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 - 3.38 \cdot 10^{15} & 1 \end{bmatrix}, U = \begin{bmatrix} 1 & 1 & 3 \\ 0 - 8.88 \cdot 10^{-16} & 14 \\ 0 & 0 & 4.73 \cdot 10^{16} \end{bmatrix},$$

il cui prodotto produce la matrice

```
>> L*U
ans =
1.0000 1.0000 3.0000
2.0000 2.0000 20.0000
3.0000 6.0000 0.0000
```

Si noti che l'elemento (3,3) di tale matrice vale 0, mentre il corrispondente elemento di A è pari a 4. Il calcolo accurato delle matrici L e U può essere ottenuto operando una pivotazione parziale per righe: con il comando `[L,U,P]=lu(A)` si ottengono infatti i fattori corretti.

Soluzione 5.7 Tipicamente, di una matrice simmetrica, si memorizza la sola parte triangolare superiore od inferiore. Di conseguenza, poiché il *pivoting* per righe non conserva in generale la simmetria di una matrice, esso risulta particolarmente penalizzante dal punto di vista dell'occupazione di memoria. Un rimedio consiste nello scambiare fra loro contemporaneamente righe e colonne con gli stessi indici, ovvero limitare la scelta dei *pivot* ai soli elementi diagonali.

Soluzione 5.8 I fattori L ed U sono:

$$L = \begin{bmatrix} 1 & 0 & 0 \\ (\varepsilon - 2)/2 & 1 & 0 \\ 0 & -1/\varepsilon & 1 \end{bmatrix}, U = \begin{bmatrix} 2 & -2 & 0 \\ 0 & \varepsilon & 0 \\ 0 & 0 & 3 \end{bmatrix}.$$

Ovviamente $l_{32} \rightarrow \infty$ quando $\varepsilon \rightarrow 0$. Ciò non implica però che la soluzione calcolata del sistema divenga inaccurata al rimpicciolirsi di ε come si verifica con le seguenti istruzioni:

```
>> e=1; for k=1:10
b=[0; e; 2];
L=[1 0 0; (e-2)*0.5 1 0; 0 -1/e 1]; U=[2 -2 0; 0 e 0; 0 0 3];
y=L\b; x=U\y; err(k)=max(abs(x-ones(3,1))); e=e*0.1;
end
>> err
err =
0 0 0 0 0 0 0 0 0 0
```

Soluzione 5.9 La soluzione calcolata diventa sempre meno accurata al crescere del pedice i . Gli errori in norma sono infatti pari a $2.63 \cdot 10^{-14}$ per $i = 1$, $9.89 \cdot 10^{-10}$ per $i = 2$ e $2.10 \cdot 10^{-6}$ per $i = 3$. Il responsabile di questo comportamento è il numero di condizionamento di A_i che cresce al crescere di i . Utilizzando il comando `cond` si trova infatti che esso è dell'ordine di 10^3 per $i = 1$, di 10^7 per $i = 2$ e di 10^{11} per $i = 3$.

Soluzione 5.10 Se λ è un autovalore di A associato ad un autovettore \mathbf{v} , allora λ^2 è un autovalore di A^2 associato allo stesso autovettore. Infatti, da $A\mathbf{v} = \lambda\mathbf{v}$ segue $A^2\mathbf{v} = \lambda A\mathbf{v} = \lambda^2\mathbf{v}$. Di conseguenza, $K(A^2) = (K(A))^2$.

Soluzione 5.11 La matrice di iterazione del metodo di Jacobi è:

$$B_J = \begin{bmatrix} 0 & 0 & -\alpha^{-1} \\ 0 & 0 & 0 \\ -\alpha^{-1} & 0 & 0 \end{bmatrix}$$

ed ha autovalori $\{0, \alpha^{-1}, -\alpha^{-1}\}$. Il metodo converge pertanto se $|\alpha| > 1$.

Nel caso del metodo di Gauss-Seidel si ha invece

$$B_{GS} = \begin{bmatrix} 0 & 0 & -\alpha^{-1} \\ 0 & 0 & 0 \\ 0 & 0 & \alpha^{-2} \end{bmatrix}$$

con autovalori dati da $\{0, 0, \alpha^{-2}\}$. Il metodo è quindi convergente se $|\alpha| > 1$. Si noti che, avendosi $\rho(B_{GS}) = [\rho(B_J)]^2$, il metodo di Gauss-Seidel convergerà 2 volte più rapidamente del metodo di Jacobi.

Soluzione 5.12 Condizione sufficiente per la convergenza dei metodi di Jacobi e di Gauss-Seidel è che A sia a dominanza diagonale stretta. Essendo la prima riga di A già a dominanza diagonale stretta, affinché lo sia A basterà imporre che $|\beta| < 5$. Si noti che il calcolo diretto dei raggi spettrali delle matrici di iterazione porterebbe alle limitazioni (necessaria e sufficiente) $|\beta| < 25$ per entrambi gli schemi.

Soluzione 5.13 Il metodo del rilassamento può essere scritto nella seguente forma vettoriale

$$(I - \omega D^{-1}E)\mathbf{x}^{(k+1)} = [(1 - \omega)I + \omega D^{-1}F]\mathbf{x}^{(k)} + \omega D^{-1}\mathbf{b}$$

dove $A = D - (E + F)$, essendo D la diagonale di A , $-E$ e $-F$ la parte triangolare inferiore e superiore di A , rispettivamente. Si ricava allora che la matrice di iterazione è:

$$B(\omega) = (I - \omega D^{-1}E)^{-1}[(1 - \omega)I + \omega D^{-1}F].$$

Possiamo a questo punto osservare che, se denotiamo con λ_i gli autovalori di $B(\omega)$, allora

$$\left| \prod_{i=1}^n \lambda_i \right| = |\det [(1 - \omega)I + \omega D^{-1}F]| = |1 - \omega|^n$$

e, di conseguenza, ci deve essere almeno un autovalore tale che $|\lambda_i| \geq |1 - \omega|$. Quindi, condizione necessaria per avere convergenza è che $|1 - \omega| < 1$, cioè $0 < \omega < 2$.

Soluzione 5.14 La matrice in esame è simmetrica. Dobbiamo chiederci se è definita positiva ossia se $\mathbf{z}^T A \mathbf{z} > 0$ per ogni $\mathbf{z} \neq \mathbf{0}$ di \mathbb{R}^2 . Eseguiamo i calcoli con MATLAB nel modo seguente:

```
>> syms z1 z2 real
>> z=[z1;z2]; A=[3 2; 2 6];
>> pos=z'*A*z; simple(pos)
ans =
3*z1^2+4*z1*z2+6*z2^2
```

dove il comando `syms z1 z2 real` ci è servito per dichiarare reali le variabili simboliche `z1` e `z2`. Il comando `simple` ha messo nella forma più semplice il contenuto della variabile `pos`. È evidente che la quantità ottenuta è sempre positiva, in quanto può essere riscritta come $2*(z1+z2)^2 + z1^2 + 4*z2^2$. La matrice è dunque simmetrica definita positiva ed il metodo di Gauss-Seidel converge.

`syms`
`simple`

Soluzione 5.15 Si trova:

$$\text{per il metodo di Jacobi: } \begin{cases} x_1^{(1)} = \frac{1}{2}(1 - x_2^{(0)}) \\ x_2^{(1)} = -\frac{1}{3}(x_1^{(0)}) \end{cases} \Rightarrow \begin{cases} x_1^{(1)} = \frac{1}{4} \\ x_2^{(1)} = -\frac{1}{3} \end{cases}$$

$$\text{per il metodo di Gauss-Seidel: } \begin{cases} x_1^{(1)} = \frac{1}{2}(1 - x_2^{(0)}) \\ x_2^{(1)} = -\frac{1}{3}x_1^{(1)} \end{cases} \Rightarrow \begin{cases} x_1^{(1)} = \frac{1}{4} \\ x_2^{(1)} = -\frac{1}{12} \end{cases}$$

Per quanto riguarda il metodo del gradiente, determiniamo prima il residuo pari a

$$\mathbf{r}^{(0)} = \mathbf{b} - A\mathbf{x}^{(0)} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} - \begin{bmatrix} 2 & 1 \\ 1 & 3 \end{bmatrix} \mathbf{x}^{(0)} = \begin{bmatrix} -3/2 \\ -5/2 \end{bmatrix}.$$

A questo punto, avendosi

$$\mathbf{P}^{-1} = \begin{bmatrix} 1/2 & 0 \\ 0 & 1/3 \end{bmatrix},$$

si può calcolare $\mathbf{z}^{(0)} = \mathbf{P}^{-1}\mathbf{r}^{(0)} = (-3/4, -5/6)^T$. Di conseguenza,

$$\alpha_0 = \frac{(\mathbf{z}^{(0)})^T \mathbf{r}^{(0)}}{(\mathbf{z}^{(0)})^T A \mathbf{z}^{(0)}} = \frac{77}{107},$$

e

$$\mathbf{x}^{(1)} = \mathbf{x}^{(0)} + \alpha_0 \mathbf{z}^{(0)} = (197/428, -32/321)^T.$$

Soluzione 5.16 Gli autovalori della matrice $B_\alpha = I - \alpha P^{-1}A$ sono $\lambda_i(\alpha) = 1 - \alpha \mu_i$, essendo μ_i l' i -esimo autovalore di $P^{-1}A$. Allora

$$\rho(B_\alpha) = \max_{i=1,\dots,n} |1 - \alpha \mu_i| = \max(|1 - \alpha \mu_{min}|, |1 - \alpha \mu_{max}|).$$

Di conseguenza, il valore ottimale di α (ossia quello che rende minimo il raggio spettrale della matrice di iterazione) si trova come soluzione dell'equazione

$$1 - \alpha \mu_{min} = \alpha \mu_{max} - 1$$

e cioè la (5.43). A questo punto la (5.54) si trova calcolando $\rho(B_{\alpha_{opt}})$.

Soluzione 5.17 La matrice A del modello di Leontieff non è in questo caso definita positiva, come si può osservare con le seguenti istruzioni:

```
>> for i=1:20; for j=1:20; C(i,j)=i+j; end; end; A=eye(20)-C;
>> min(eig(A))
ans =
-448.5830
```

e pertanto non si ha la garanzia che il metodo del gradiente converga. D'altra parte, essendo A non singolare, il sistema dato è equivalente al sistema $A^T A \mathbf{x} = A^T \mathbf{b}$ che ha matrice simmetrica e definita positiva. Risolviamo tale sistema richiedendo una tolleranza sul residuo relativo pari a 10^{-10} e partendo dal dato iniziale $\mathbf{x}^{(0)} = \mathbf{0}^T$:

```
>> b = [1:20]'; AA=A'*A; b=A'*b; x0 = zeros(20,1);
>> [x,iter]=itermeth(AA,b,x0,100,1.e-10);
```

Il metodo converge in 15 iterazioni. Facciamo notare che un problema di questo approccio risiede nel fatto che la matrice $A^T A$ ha, in generale, un numero di condizionamento molto maggiore della matrice di partenza A.

9.6 Capitolo 6

Soluzione 6.1 A_1 : il metodo converge in 34 passi al valore 2.00000000004989. A_2 : a partire dallo stesso vettore iniziale servono ora 457 iterazioni per ottenere il valore 1.99999999990611. Il peggioramento nella velocità di convergenza è dovuto al fatto che i due autovalori più grandi in modulo sono molto vicini tra loro. Infine, per A_3 il metodo non converge in quanto questa matrice ha come autovalori di modulo massimo i e $-i$.

Soluzione 6.2 La matrice di Leslie associata ai valori riportati in tabella è

$$A = \begin{bmatrix} 0 & 0.5 & 0.8 & 0.3 \\ 0.2 & 0 & 0 & 0 \\ 0 & 0.4 & 0 & 0 \\ 0 & 0 & 0.8 & 0 \end{bmatrix}.$$

Con il metodo delle potenze si trova che $\lambda_1 \simeq 0.5353$ e la distribuzione per fasce d'età è data dalle componenti dell'autovettore di norma unitaria associato cioè $\mathbf{x}_1 \simeq (0.8477, 0.3167, 0.2367, 0.3537)^T$.

Soluzione 6.3 Riscriviamo il generico vettore iniziale come

$$\mathbf{y}^{(0)} = \beta^{(0)} - \alpha_1 \mathbf{x}_1 + \alpha_2 \mathbf{x}_2 + \sum_{i=3}^n \alpha_i \mathbf{x}_i \right),$$

con $\beta^{(0)} = 1/\|\mathbf{x}^{(0)}\|$. Ripetendo i calcoli svolti nel paragrafo 6.1, al generico passo k avremo:

$$\mathbf{y}^{(k)} = \gamma^k \beta^{(k)} - \alpha_1 \mathbf{x}_1 e^{ik\vartheta} + \alpha_2 \mathbf{x}_2 e^{-ik\vartheta} + \sum_{i=3}^n \alpha_i \frac{\lambda_i^k}{\gamma^k} \mathbf{x}_i \right).$$

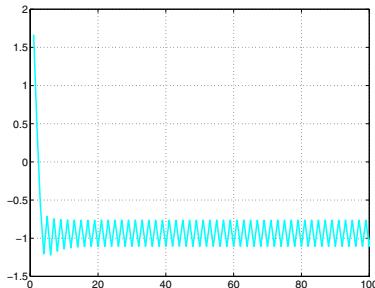


Figura 9.9. Approssimazioni dell'autovalore di modulo massimo calcolate dal metodo delle potenze al variare del numero di iterazioni per la matrice della Soluzione 6.5

Di conseguenza, per $k \rightarrow \infty$ i primi due termini della somma sopravvivono, a causa degli esponenti di segno opposto, ed impediscono alla successione degli $\mathbf{y}^{(k)}$ di convergere, conferendole un andamento oscillante.

Soluzione 6.4 Se A è non singolare, da $A\mathbf{x} = \lambda\mathbf{x}$, si ha $A^{-1}A\mathbf{x} = \lambda A^{-1}\mathbf{x}$, e quindi: $A^{-1}\mathbf{x} = (1/\lambda)\mathbf{x}$.

Soluzione 6.5 Il metodo delle potenze applicato ad A produce una successione oscillante di approssimazioni dell'autovalore di modulo massimo (si veda la Figura 9.9). Questo perché esistono due autovalori distinti aventi modulo massimo uguale a 1.

Soluzione 6.6 Sappiamo che gli autovalori di una matrice simmetrica sono tutti reali e quindi appartengono ad un intervallo chiuso e limitato $[\lambda_a, \lambda_b]$. Il nostro obiettivo è proprio calcolare λ_a e λ_b . Richiamiamo il Programma 13 per calcolare l'autovalore di modulo massimo di A :

```
>> A=wilkinson(7);
>> x0=ones(7,1); tol=1.e-15; nmax=100;
>> [lambdab,x,iter]=eigpower(A,tol,nmax,x0);
```

Si trova, in 35 iterazioni, $\text{lambdab}=3.76155718183189$, che coincide con il valore λ_b . A questo punto, per calcolare l'autovalore λ_a , applichiamo il metodo delle potenze alla matrice $A_b = A - \lambda_b I$. In tal modo troveremo l'autovalore λ di modulo massimo di A_b e porremo $\lambda_a = \lambda + \lambda_b$. Si trova:

```
>> [lambda,x,iter]=eigpower(A-lambda*eye(7),tol,nmax,x0);
>> lambda+lambdab
ans =
-1.12488541976457
```

con $\text{iter} = 33$. I risultati trovati sono delle ottime approssimazioni degli autovalori massimi, positivo e negativo, di A .

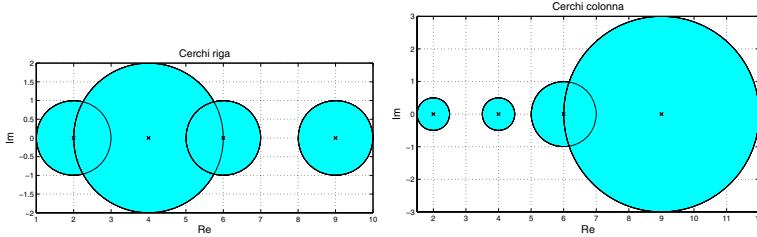


Figura 9.10. Cerchi colonna per la matrice B della Soluzione 6.7

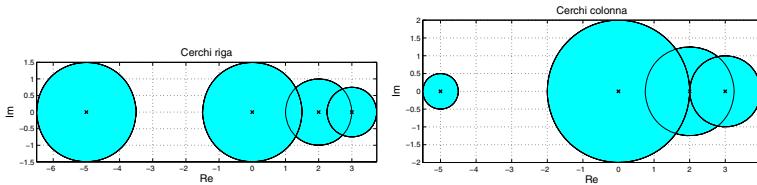


Figura 9.11. Cerchi colonna per la matrice B della Soluzione 6.7

Soluzione 6.7 Consideriamo la matrice A. Dall'esame dei cerchi riga vediamo che c'è un cerchio isolato di centro 9 e raggio 1 che, per la Proposizione 6.1, potrà contenere un solo autovalore λ_1 che dovrà essere reale (la matrice è a coefficienti reali, se $\lambda_1 \in \mathbb{C}$ allora anche $\bar{\lambda}_1$ dovrebbe essere un autovalore, ma, per come sono disposti i cerchi, questo non è possibile). Avremo quindi $\lambda_1 \in (8, 10)$. Dall'esame dei cerchi colonna vediamo che ci sono altri due cerchi isolati di raggio $1/2$ e centro 2 e 4, rispettivamente (si veda la Figura 9.10). Avremo quindi altri due autovalori reali, $\lambda_2 \in (1.5, 2.5)$ e $\lambda_3 \in (3.4, 4.5)$. Essendo la matrice a coefficienti reali anche l'autovalore restante dovrà essere reale.

Consideriamo ora la matrice B. Dall'analisi dei suoi cerchi riga e colonna (si veda la Figura 9.11) deduciamo che c'è un solo cerchio isolato di centro -5 e raggio $1/2$. Per come sono distribuiti i cerchi esiste quindi un autovalore reale in $(-5.5, -4.5)$. Gli altri tre cerchi hanno invece intersezione non vuota e quindi i restanti tre autovalori di B potranno essere o tutti reali o uno reale e due complessi coniugati.

Soluzione 6.8 Dall'analisi dei cerchi riga di A vediamo che c'è un cerchio isolato di centro 5 e raggio 2 che, per come sono fatti i cerchi restanti, deve contenere l'autovalore di modulo massimo. Poniamo dunque lo shift pari a 5. Il confronto si effettua con le seguenti istruzioni:

```
>> A=[5 0 1 -1; 0 2 0 -1/2; 0 1 -1 1; -1 -1 0 0]; tol=1.e-14; x0=ones(4,1); nmax=100;
>> t=cputime; [lambda,x,iter]=eigpower(A,tol,nmax,x0); cputime-t, iter
ans =
    0.0500
iter =
    34
```

```
>> t=cputime; [lambda,x,iter]=invshift(A,5,tol,nmax,x0); cputime-t, iter
ans =
    0.050
iter =
    12
```

Come si vede i due metodi, pur presentando un numero di iterazioni molto diverso, richiedono per la loro esecuzione lo stesso tempo di CPU. Questo è dovuto al fatto che il metodo delle potenze inverse con *shift* richiede il calcolo della fattorizzazione LU di A e, ad ogni iterazione, la soluzione di due sistemi triangolari.

Soluzione 6.9 Abbiamo

$$A^{(k)} = Q^{(k+1)} R^{(k+1)} \text{ e } A^{(k+1)} = R^{(k+1)} Q^{(k+1)}$$

e quindi

$$(Q^{(k+1)})^T A^{(k)} Q^{(k+1)} = R^{(k+1)} Q^{(k+1)} = A^{(k+1)}.$$

Si conclude che, essendo $(Q^{(k+1)})^T = (Q^{(k+1)})^{-1}$ la matrice $A^{(k)}$ è simile alla matrice $A^{(k+1)}$ per ogni $k \geq 0$.

Soluzione 6.10 Basta dare i comandi

```
>> A=[-5 0 1/2 1/2; 1/2 2 1/2 0; 0 1 0 1/2; 0 1/4 1/2 3];
>> D=qrbasis(A,1.e-14,1000)
```

Il metodo e' andato a convergenza in 91 iterazioni

```
D =
-4.9921
 3.1292
 2.1666
 -0.3038
```

```
>> B=[-5 0 1/1 1/2; 1/2 2 1/2 0; 0 1 0 1/2; 0 1/4 1/2 3];
>> D=qrbasis(B,1.e-14,1000)
```

Il metodo e' andato a convergenza in 91 iterazioni

```
D =
-4.9849
 3.1316
 2.1769
 -0.3237
```

I risultati confermano quanto dedotto nella Soluzione 6.7.

9.7 Capitolo 7

Soluzione 7.1 Per verificare l'ordine osserviamo che la soluzione analitica della (7.70) è $y(t) = \frac{1}{2}[e^t - \sin(t) - \cos(t)]$. Risolviamo allora il problema (7.70) con il metodo di Eulero esplicito con h che va da $1/2$ fino a $1/512$ per dimezzamenti successivi:

```

>> t0=0; y0=0;
>> f=inline('sin(t)+y','t','y'); y=inline('0.5*(exp(t)-sin(t)-cos(t))','t');
>> T=1; N=2; for k=1:10;
[tt,u]=feuler(f,[t0,T],y0,N);t=tt(end);e(k)=abs(u(end)-feval(y,t));N=2*N;end
>> e
e =
Columns 1 through 7
    0.4285    0.2514    0.1379    0.0725    0.0372    0.0189    0.0095
Columns 8 through 10
    0.0048    0.0024    0.0012

```

Per la (1.11), con il comando

```

>> p=log(abs(e(1:end-1)./e(2:end)))/log(2); p(1:2:end)
p =
    0.7696    0.9273    0.9806    0.9951    0.9988

```

si verifica che il metodo è di ordine 1. Facendo uso dei medesimi comandi appena impiegati e sostituendo la chiamata al Programma 17 con la corrispondente chiamata al Programma 18 si ottengono le seguenti stime per l'ordine di Eulero implicito

```

>> p=log(abs(e(1:end-1)./e(2:end)))/log(2); p(1:2:end)
p =
    1.5199    1.0881    1.0204    1.0050    1.0012

```

in buon accordo con quanto previsto dalla teoria.

Soluzione 7.2 Risolviamo il problema di Cauchy con il metodo di Eulero esplicito con le seguenti istruzioni:

```

>> t0=0; T=1; N=100; f=inline('-t*exp(-y)','t','y');
>> y0=0;[t,u]=feuler(f,[t0,T],y0,N);

```

Per calcolare il numero di cifre corrette, vogliamo usare la (7.13) e, di conseguenza, dobbiamo stimare L e M . Osserviamo che, essendo $f(t, y(t)) < 0$ nell'intervallo dato, $y(t)$ sarà una funzione monotona decrescente e, valendo 0 in $t = 0$, dovrà essere necessariamente negativa. Essendo sicuramente compresa fra -1 e 0, possiamo supporre che in $t = 1$ valga al più -1.

A questo punto possiamo determinare L . Essendo f derivabile con continuità rispetto a y , possiamo prendere $L = \max_{0 \leq t \leq 1} |L(t)|$ con $L(t) = \partial f / \partial y = -te^{-y}$. Osserviamo che $L(0) = 0$ e $L'(t) > 0$ per ogni $t \in (0, 1]$. Dunque, essa assumerà massimo in $t = 1$ e, per l'assunzione fatta su $y(1)$, varrà e .

Per quanto riguarda $M = \max_{0 \leq t \leq 1} |y''(t)|$ con $y'' = -e^{-y} - t^2 e^{-2y}$, si ha che $|y''|$ è massima per $t = 1$ e quindi $M = e + e^2$. Dalla (7.13), per $h = 0.01$ si ricava allora

$$|u_{100} - y(1)| \leq \frac{e^L - 1}{L} \frac{M}{200} = 0.26$$

e quindi il numero di cifre significative corrette della soluzione approssimata in $t = 1$ è al più uno. In effetti, l'ultima componente della soluzione numerica è $u(\text{end})=-0.6785$, mentre la soluzione esatta $y(t) = \log(1 - t^2/2)$ in $t = 1$ vale -0.6931.

Soluzione 7.3 La funzione di iterazione è $\phi(u) = u_n - ht_{n+1}e^{-u}$. Il metodo di punto fisso è convergente se $|\phi'(u)| < 1$. Dobbiamo quindi imporre $h(t_0 + (n+1)h) < e^u$. Consideriamo u uguale alla soluzione esatta. In tal caso la situazione più restrittiva si ha quando $u = -1$ (si veda la Soluzione 7.2). Si tratta pertanto di risolvere la disequazione $(n+1)h^2 < e^{-1}$, essendo $t_0 = 0$. La restrizione su h affinché si abbia convergenza è allora $h < \sqrt{e^{-1}/(n+1)}$.

Soluzione 7.4 Basta ripetere le istruzioni date nella Soluzione 7.1, utilizzando il Programma 19. Si trova la seguente stima dell'ordine:

```
>> p=log(abs(e(1:end-1)./e(2:end)))/log(2); p(1:2:end)
p =
    2.0379  2.0023  2.0001  2.0000  2.0000
```

in ottimo accordo con quanto previsto dalla teoria.

Soluzione 7.5 Consideriamo la formulazione integrale del problema di Cauchy sull'intervallo $[t_n, t_{n+1}]$:

$$y(t_{n+1}) - y(t_n) = \int_{t_n}^{t_{n+1}} f(\tau, y(\tau)) d\tau,$$

ed approssimiamo l'integrale con la formula del trapezio, ottenendo:

$$y(t_{n+1}) - y(t_n) \simeq \frac{h}{2} [f(t_n, y(t_n)) + f(t_{n+1}, y(t_{n+1}))].$$

Se ora definiamo $u_0 = y(t_0)$ e u_{n+1} tale che

$$u_{n+1} = u_n + \frac{h}{2} [f(t_n, u_n) + f(t_{n+1}, u_{n+1})], \quad \forall n \geq 0,$$

otteniamo proprio il metodo di Crank-Nicolson.

Soluzione 7.6 Sappiamo che la regione di assoluta stabilità per il metodo di Eulero in avanti è il cerchio di centro $(-1, 0)$ e raggio 1 o, equivalentemente, l'insieme degli $z \in \mathbb{C}$ tali che $|1 + h\lambda| < 1$. Sostituendo in questa espressione $\lambda = -1 + i$ otteniamo la limitazione su h : $h^2 - h < 0$, ovvero $h \in [0, 1)$.

Soluzione 7.7 Per comodità di notazioni riscriviamo il metodo di Heun nel seguente modo (comune ai metodi Runge-Kutta):

$$u_{n+1} = u_n + \frac{1}{2}(k_1 + k_2), \quad k_1 = hf(t_n, u_n), \quad k_2 = hf(t_{n+1}, u_n + k_1). \quad (9.2)$$

Abbiamo $h\tau_{n+1}(h) = y(t_{n+1}) - y(t_n) - (\hat{k}_1 + \hat{k}_2)/2$, con $\hat{k}_1 = hf(t_n, y(t_n))$ e $\hat{k}_2 = hf(t_{n+1}, y(t_n) + \hat{k}_1)$. Poiché f è continua rispetto ad entrambi gli argomenti si ha

$$\lim_{h \rightarrow 0} \tau_{n+1} = y'(t_n) - \frac{1}{2}[f(t_n, y(t_n)) + f(t_n, y(t_n))] = 0.$$

Il metodo di Heun è dunque consistente ed è implementato nel Programma 28. Utilizzando comandi del tutto analoghi a quelli usati nella Soluzione 7.1 si trovano le seguenti stime per l'ordine:

```
>> p=log(abs(e(1:end-1)./e(2:end)))/log(2); p(1:2:end)
ans =
1.7642 1.9398 1.9851 1.9963 1.9991
```

che sono in accordo con l'ordine previsto teoricamente.



Programma 28 - rk2 : metodo Runge-Kutta esplicito di ordine 2

```
function [t,u]=rk2(t0,T,N,f,y0)
h=(T-t0)/N; tt=[t0:h:T]; u(1)=y0;
for s=tt(1:end-1)
    t = s; y = u(end); k1=h*eval(f); t = t + h; y = y + k1; k2=h*eval(f);
    u = [u, y + 0.5*(k1+k2)];
end
t=tt;
```

Soluzione 7.8 Applicando il metodo (9.2) al problema (7.28) si trova $k_1 = h\lambda u_n$ e $k_2 = h\lambda u_n(1 + h\lambda)$. Di conseguenza, $u_{n+1} = u_n[1 + h\lambda + (h\lambda)^2/2] = u_n p_2(h\lambda)$. Per avere assoluta stabilità dobbiamo imporre $|p_2(h\lambda)| < 1$, ma essendo $p_2(h\lambda)$ sempre positivo, questa condizione equivale a chiedere che $0 < p_2(h\lambda) < 1$. Risolvendo quest'ultima disequazione si trova la restrizione cercata, $-2 < h\lambda < 0$ con λ con parte reale negativa.

Soluzione 7.9 Si noti che

$$u_n = u_{n-1}(1 + h\lambda_{n-1}) + hr_{n-1}$$

e si proceda quindi ricorsivamente su n .

Soluzione 7.10 La diseguaglianza (7.38) segue dalla (7.37) in quanto

$$|z_n - u_n| \leq |\rho| \left(\left| 1 + \frac{1}{\lambda} \right| + \left| \frac{1}{\lambda} \right| \right),$$

essendo $|1 + h\lambda| < 1$.

Soluzione 7.11 Dalla (7.35) abbiamo

$$|z_n - u_n| \leq \rho_{max} [a(h)]^n + h \rho_{max} \sum_{k=0}^{n-1} [a(h)]^{n-k-1}$$

ed il risultato segue per la (7.36).

Soluzione 7.12 Abbiamo:

$$h\tau_{n+1}(h) = y(t_{n+1}) - y(t_n) - \frac{1}{6}(\hat{k}_1 + 4\hat{k}_2 + \hat{k}_3),$$

$$\hat{k}_1 = hf(t_n, y(t_n)), \hat{k}_2 = hf(t_n + \frac{h}{2}, y(t_n) + \frac{\hat{k}_1}{2}), \hat{k}_3 = hf(t_{n+1}, y(t_n) + 2\hat{k}_2 - \hat{k}_1).$$

Essendo f continua rispetto ad entrambi gli argomenti, si ha

$$\lim_{h \rightarrow 0} \tau_{n+1} = y'(t_n) - \frac{1}{6}[f(t_n, y(t_n)) + 4f(t_n, y(t_n)) + f(t_n, y(t_n))] = 0,$$

ovvero il metodo consistente. Esso è implementato nel Programma 29. Utilizzando comandi del tutto analoghi a quelli usati nella Soluzione 7.7 si trovano le seguenti stime per l'ordine:

```
>> p=log(abs(e(1:end-1)./e(2:end)))/log(2); p(1:2:end)
ans =
    2.7306   2.9330   2.9833   2.9958   2.9990
```

che verificano la stima teorica.

Soluzione 7.13 Utilizzando passaggi del tutto analoghi a quelli della Soluzione 7.8 si trova la relazione

$$u_{n+1} = u_n [1 + h\lambda + \frac{1}{2}(h\lambda)^2 + \frac{1}{6}(h\lambda)^3] = u_n p_3(h\lambda).$$

Da uno studio grafico, effettuato con i seguenti comandi

```
>> c=[1/6 1/2 1 1]; z=[-3:0.01:1]; p=polyval(c,z); plot(z,abs(p))
```

si deduce che $|p_3(h\lambda)| < 1$ per $-2.5 < h\lambda < 0$.

Programma 29 - rk3 : metodo Runge-Kutta esplicito di ordine 3



```
function [t,u]=rk3(t0,T,N,f,y0)
h=(T-t0)/N; tt=[t0:h:T]; u(1)=y0;
for s=tt(1:end-1)
    t = s;      y = u(end);      k1=h*eval(f);
    t = t + h*0.5; y = y + 0.5*k1;      k2=h*eval(f);
    t = s + h;    y = u(end) + 2*k2-k1; k3=h*eval(f);
    u = [u, u(end) + (k1+4*k2+k3)/6];
end
t=tt;
```

Soluzione 7.14 Il metodo (7.72) quando applicato al problema modello (7.28) con $\lambda \in \mathbb{R}^-$ fornisce l'equazione $u_{n+1} = u_n(1 + h\lambda + (h\lambda)^2)$. Uno studio grafico mostra che la condizione di assoluta stabilità è soddisfatta se $-1 < h\lambda < 0$.

Soluzione 7.15 Per risolvere il Problema 7.1 con i valori indicati, basta ripetere le seguenti istruzioni prima con $N=10$ e poi con $N=20$:

```
>> f=inline('-1.68*10^(-9)*y^4+2.6880','t','y');
>> [tc,uc]=cranknic(f,[0,200],180,N);
>> [tp,up]=predcor(f,[0,200],180,N,'eonestep','cnonestep');
```

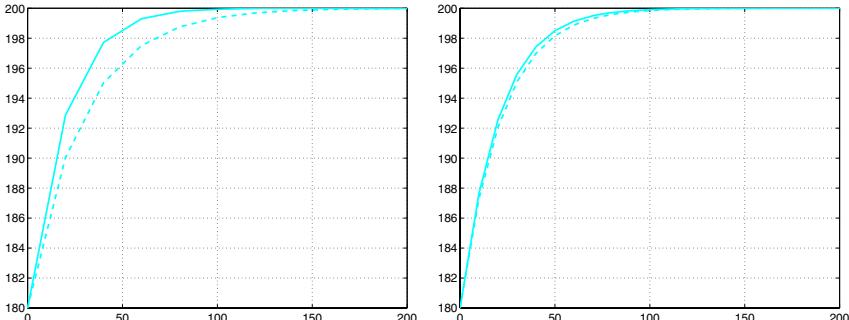


Figura 9.12. Soluzioni calcolate con $h = 20$ (a sinistra) e $h = 10$ (a destra) per il problema di Cauchy della Soluzione 7.15: in linea continua le soluzioni ottenute con il metodo di Crank-Nicolson, in linea tratteggiata quelle ricavate con il metodo di Heun

Le corrispondenti soluzioni vengono riportate nei grafici di Figura 9.12. Come si nota le soluzioni prodotte dal metodo di Crank-Nicolson sono più accurate di quelle ottenute con il metodo di Heun.

Soluzione 7.16 La soluzione numerica del metodo di Heun, applicato al problema modello (7.28), soddisfa

$$u_{n+1} = u_n \left(1 + h\lambda + \frac{1}{2}h^2\lambda^2 \right).$$

Il bordo della regione di assoluta stabilità è allora individuato dai valori di $h\lambda = x + iy$ tali che $|1 + h\lambda + h^2\lambda^2/2|^2 = 1$. Sviluppando questa espressione troviamo che essa è soddisfatta dalle coppie di valori (x, y) tali che $f(x, y) = x^4 + y^4 + 2x^2y^2 + 4x^3 + 4xy^2 + 8x^2 + 8x = 0$. Possiamo rappresentare questa funzione in MATLAB, disegnando la curva di livello corrispondente al valore $z = 0$ della funzione $f(x, y) = z$ con i seguenti comandi:

```
>> f='x.^4+y.^4+2*(x.^2).*(y.^2)+4*x.*y.^2+4*x.^3+8*x.^2+8*x';
>> [x,y]=meshgrid([-2.1:0.1:0.1],[-2:0.1:2]);
>> contour(x,y,eval(f),[0 0])
```

Con il comando `meshgrid` abbiamo introdotto nel rettangolo $[-2.1, 0.1] \times [-2, 2]$ una griglia formata da 23 nodi equispaziati lungo l'asse delle x e da 41 nodi equispaziati lungo l'asse delle y . Su di essi è stata valutata la funzione `f` con il comando `eval(f)` e, tramite la funzione `contour`, è stata individuata la linea di livello relativa al valore $z = 0$ (precisata nel vettore `[0 0]` nella chiamata a `contour`). In Figura 9.13 viene riportato in linea continua il risultato ottenuto. La regione di assoluta stabilità del metodo di Heun si trova all'interno di tale linea. Come si vede essa è più estesa della corrispondente regione del metodo di Eulero esplicito (delimitata dal cerchio in linea tratteggiata) ed è anche essa tangente nell'origine all'asse immaginario.

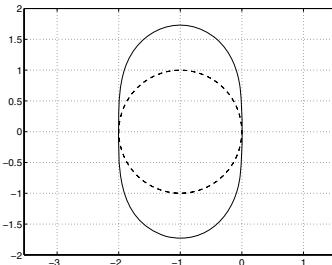


Figura 9.13. Bordo delle regioni di assoluta stabilità per i metodi di Eulero esplicito (in tratteggio) e di Heun (in linea continua). Le regioni si estendono all'interno delle aree delimitate dalle rispettive curve

Soluzione 7.17 Basta dare le seguenti istruzioni:

```
>> t0=0; y0=0;
>> f=inline('cos(2*y)','t','y');
>> y=inline('0.5*asin((exp(4*t)-1)./(exp(4*t)+1))','t');
>> T=1; N=2; for k=1:10;
[tt,u]=predcor(f,[t0,T],y0,N,'feonestep','cnonestep');
e(k)=abs(u(end)-feval(y,tt(end))); N=2*N; end
>> p=log(abs(e(1:end-1)./e(2:end)))/log(2); p(1:2:end)
2.4733 2.1223 2.0298 2.0074 2.0018
```

Al solito, si è stampato il valore presunto dell'ordine p , che converge a 2 a conferma dell'ordine atteso del metodo.

Soluzione 7.18 L'equazione data è equivalente al sistema seguente:

$$x'(t) = z(t), \quad z'(t) = -5z(t) - 6x(t),$$

con $x(0) = 1$, $z(0) = 0$. Richiamiamo Heun con le seguenti istruzioni:

```
>> t0=0; y0=[1 0]; T=5;
>> [t,u]=predcor(@fmolle,[t0,T],y0,N,'feonestep','cnonestep');
```

dove N è il numero di nodi che utilizzeremo, mentre `fmolle.m` è la seguente funzione:

```
function y=fmolley(t,y)
b=5; k=6;
yy=y; y(1)=yy(2); y(2)=-b*yy(2)-k*yy(1);
```

In Figura 9.14 riportiamo le 2 componenti della soluzione, calcolate con $N=20$ e $N=40$ e confrontate con la soluzione analitica $x(t) = 3e^{-2t} - 2e^{-3t}$ e con la sua derivata.

Soluzione 7.19 Riduciamo il sistema di equazioni di ordine 2 ad un sistema di equazioni del prim'ordine dato da:

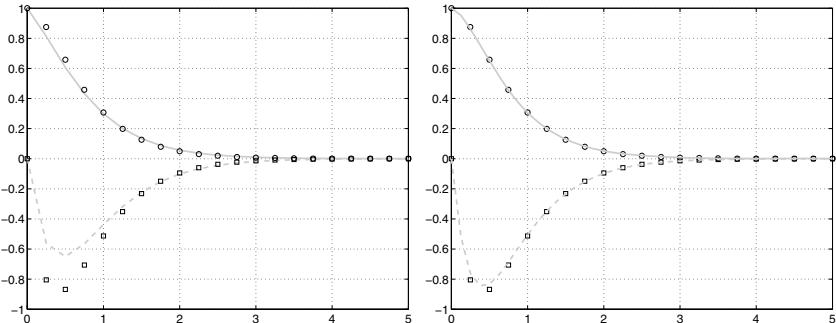


Figura 9.14. Approssimazioni di $x(t)$ (in linea continua) e $x'(t)$ (in linea tratteggiata) calcolate in corrispondenza di $N=20$ (a sinistra) e $N=40$ (a destra). I cerchietti ed i quadratini si riferiscono alle quantità esatte $x(t)$ e $x'(t)$, rispettivamente

$$\begin{cases} x'(t) = z(t), \\ y'(t) = v(t), \\ z'(t) = 2\omega \sin(\Psi) - k^2 x(t), \\ v'(t) = -2\omega \sin(\Psi)z(t) - k^2 y(t). \end{cases} \quad (9.3)$$

Se supponiamo che il pendolo all'istante iniziale $t_0 = 0$ sia fermo nella posizione $(1, 0)$, il sistema (9.3) viene completato dalle seguenti condizioni iniziali:

$$x(0) = 1, y(0) = 0, z(0) = 0, v(0) = 0.$$

Scegliamo $\Psi = \pi/4$ vale a dire pari alla latitudine media dell'Italia settentrionale. Richiamiamo il metodo di Eulero esplicito con le seguenti istruzioni:

`>> [t,u]=feuler(@ffocault,[0,300],[1 0 0 0],N);`

dove N è il numero di passi e `ffocault.m` la funzione seguente:

```
function y=ffocault(t,y)
l=20; k2=9.8/l; psi=pi/4; omega=7.29*1.e-05;
yy=y; y(1)=yy(3); y(2)=yy(4);
y(3)=2*omega*sin(psi)*yy(4)-k2*yy(1);
y(4)=-2*omega*sin(psi)*yy(3)-k2*yy(2);
```

Con pochi esperimenti si giunge alla conclusione che il metodo di Eulero esplicito non fornisce per questo problema soluzioni fisicamente plausibili, neppure per h molto piccolo. Ad esempio, in Figura 9.15 a sinistra viene riportato il grafico, nel piano delle fasi (x, y) , dei movimenti del pendolo calcolati prendendo $N=30000$ cioè $h = 1/100$. Come ci si aspetta il piano di rotazione del pendolo cambia al passare del tempo, ma, nonostante il passo di discretizzazione piccolo, aumenta inaspettatamente l'ampiezza delle oscillazioni. Risultati analoghi si trovano anche per valori molto più piccoli di h od utilizzando il metodo di Heun. Ciò accade perché problemi come questo, che presentano soluzioni limitate per t che tende all'infinito, ma non smorzate, hanno un comportamento analogo a quello del problema lineare (7.28) con valori di λ puramente

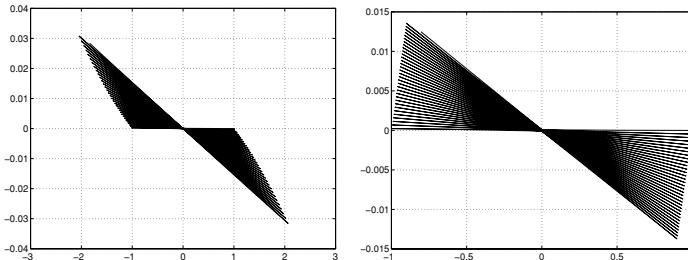


Figura 9.15. Traiettorie nel piano delle fasi per il pendolo di Foucault della Soluzione 7.19, ottenute con il metodo di Eulero esplicito (a sinistra) e con un metodo Runge-Kutta adattivo (a destra)

immaginari. In tal caso infatti la soluzione esatta è una funzione sinusoidale in t .

D'altra parte tanto il metodo di Eulero esplicito, quanto quello di Heun, hanno regioni di assoluta stabilità tangenti all'asse immaginario. Di conseguenza, il solo valore $h = 0$ garantirebbe assoluta stabilità.

Per confronto, abbiamo rappresentato in Figura 9.15, a destra, la soluzione ottenuta con la funzione MATLAB `ode23`. Essa corrisponde ad un metodo Runge-Kutta adattivo che presenta una regione di assoluta stabilità che interseca l'asse immaginario. In effetti, se richiamata con le seguenti istruzioni:

```
>> [t,u]=ode23('ffocault',[0 300],[1 0 0 0]);
```

fornisce una soluzione ragionevole, pur usando solo 1022 passi di integrazione.

`ode23`

Soluzione 7.20 Impostiamo il termine noto del problema nella seguente *function*

```
function y=baseball(t,y)
phi = 0; omega = 1800*1.047198e-01;
B = 4.1*1.e-4; yy=y;
g = 9.8;
vmodulo = sqrt(y(4)^2+y(5)^2+y(6)^2);
Fv = 0.0039+0.0058/(1+exp((vmodulo-35)/5));
y(1)=yy(4);
y(2)=yy(5);
y(3)=yy(6);
y(4)=-Fv*vmodulo*y(4)+B*omega*(yy(6)*sin(phi)-yy(5)*cos(phi));
y(5)=-Fv*vmodulo*y(5)+B*omega*yy(4)*cos(phi);
y(6)=-g-Fv*vmodulo*y(6)-B*omega*yy(4)*sin(phi);
return
```

A questo punto basta richiamare `ode23` nel modo seguente:

```
>> [t,u]=ode23(@baseball,[0 0.4],[0 0 0 38*cos(1*pi/180) 0 38*sin(1*pi/180)]);
```

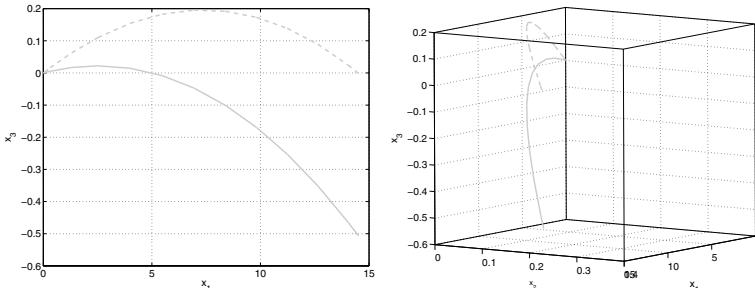


Figura 9.16. Le traiettorie seguite da una palla da baseball lanciata con angolo iniziale pari a 1 grado (in linea continua) e 3 gradi (in linea tratteggiata)

Con il comando `find` troviamo approssimativamente l'istante temporale nel quale la quota diventa negativa che corrisponde al momento d'impatto della palla con il suolo:

```
>> n=max(find(u(:,3))<=0)); t(n)
ans =
    0.1066
```

In Figura 7.1 riportiamo le traiettorie della palla da baseball con un'inclinazione di 1 grado e di 3 gradi in una rappresentazione sul piano x_1x_3 ed in una rappresentazione tridimensionale.

9.8 Capitolo 8

Soluzione 8.1 Lo verifichiamo direttamente mostrando che $\mathbf{x}^T \mathbf{A} \mathbf{x} > 0$ per ogni $\mathbf{x} \neq \mathbf{0}$. Abbiamo

$$\begin{aligned} & [x_1 \ x_2 \ \dots \ x_{N-1} \ x_N] \begin{bmatrix} 2 & -1 & 0 & \dots & 0 \\ -1 & 2 & \ddots & & \vdots \\ 0 & \ddots & \ddots & -1 & 0 \\ \vdots & & -1 & 2 & -1 \\ 0 & \dots & 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{N-1} \\ x_N \end{bmatrix} \\ & = 2x_1^2 - 2x_1x_2 + 2x_2^2 - 2x_2x_3 + \dots - 2x_{N-1}x_N + 2x_N^2. \end{aligned}$$

A questo punto basta raccogliere opportunamente i termini per concludere che l'ultima espressione trovata è equivalente a $(x_1 - x_2)^2 + \dots + (x_{N-1} - x_N)^2 + x_1^2 + x_N^2$, che è evidentemente positiva.

Soluzione 8.2 Verifichiamo che $\mathbf{A}\mathbf{q}_j = \lambda_j\mathbf{q}_j$. Eseguiamo il prodotto matrice-vettore ed imponiamo l'uguaglianza precedente componente per componente. Troviamo le seguenti equazioni:

$$\begin{cases} 2\sin(j\theta) - \sin(2j\theta) = 2(1 - \cos(j\theta))\sin(j\theta), \\ -\sin(jk\theta) + 2\sin(j(k+1)\theta) - \sin(j(k+2)\theta) = 2(1 - \cos(j\theta))\sin(2j\theta), \\ k = 1, \dots, N-2 \\ 2\sin(Nj\theta) - \sin((N-1)j\theta) = 2(1 - \cos(j\theta))\sin(Nj\theta). \end{cases}$$

La prima relazione è un'identità in quanto $\sin(2j\theta) = 2\sin(j\theta)\cos(j\theta)$. Per quanto riguarda le restanti relazioni, basta osservare che

$$\sin(jk\theta) = \sin((k+1)j\theta)\cos(j\theta) - \cos((k+1)j\theta)\sin(j\theta),$$

$$\sin(j(k+2)\theta) = \sin((k+1)j\theta)\cos(j\theta) + \cos((k+1)j\theta)\sin(j\theta).$$

Essendo A simmetrica e definita positiva, $K(A) = \lambda_{max}/\lambda_{min}$ ovvero $K(A) = \lambda_1/\lambda_N = (1 - \cos(N\pi/(N+1)))/(1 - \cos(\pi/(N+1)))$. Se si sviluppa in serie la funzione coseno e si arresta lo sviluppo al second'ordine, si trova allora $K(A) \simeq N^2$ cioè $K(A) \simeq h^{-2}$.

Soluzione 8.3 Basta osservare che:

$$\begin{aligned} u(\bar{x} + h) &= u(\bar{x}) + hu'(\bar{x}) + \frac{h^2}{2}u''(\bar{x}) + \frac{h^3}{6}u'''(\bar{x}) + \frac{h^4}{24}u^{(4)}(\xi_+), \\ u(\bar{x} - h) &= u(\bar{x}) - hu'(\bar{x}) + \frac{h^2}{2}u''(\bar{x}) - \frac{h^3}{6}u'''(\bar{x}) + \frac{h^4}{24}u^{(4)}(\xi_-), \end{aligned}$$

dove $\xi_+ \in (x, x+h)$ e $\xi_- \in (x-h, x)$. Sommando membro a membro le due espressioni si trova

$$u(\bar{x} + h) + u(\bar{x} - h) = 2u(\bar{x}) + h^2u''(\bar{x}) + \frac{h^4}{24}(u^{(4)}(\xi_+) + u^{(4)}(\xi_-)),$$

da cui la proprietà desiderata.

Soluzione 8.4 La matrice è ancora tridiagonale ed ha elementi $a_{i,i-1} = -1 - h\frac{\delta}{2}$, $a_{ii} = 2 + h^2\gamma$, $a_{i,i+1} = -1 + h\frac{\delta}{2}$. Il termine noto, una volta incorporate le condizioni al contorno, diventa conseguentemente $\mathbf{f} = (f(x_1) + \alpha(1 + h\delta/2)/h^2, f(x_2), \dots, f(x_{N-1}), f(x_N) + \beta(1 - h\delta/2)/h^2)^T$.

Soluzione 8.5 Con le seguenti istruzioni calcoliamo le soluzioni relative ai 3 valori di h indicati nel testo:

```
>> f=inline('1+sin(4*pi*x)', 'x');
>> [z,uh11]=bvp(0,1,9,0,0.1,f,0,0);
>> [z,uh21]=bvp(0,1,19,0,0.1,f,0,0);
>> [z,uh41]=bvp(0,1,39,0,0.1,f,0,0);
```

Si ricordi che $h = (b-a)/(N+1)$. Per stimare l'ordine di convergenza, non avendo a disposizione la soluzione analitica, calcoliamo una soluzione approssimata relativa ad una griglia estremamente fitta (ponendo ad esempio $h = 1/1000$). A questo punto utilizziamo la soluzione cosicalcolata invece della soluzione esatta. Troviamo:

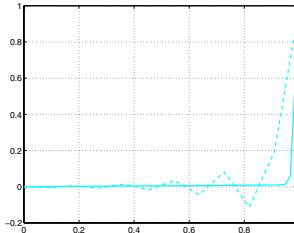


Figura 9.17. Soluzioni calcolate per il problema della Soluzione 8.6 con $h = 1/10$ (linea tratteggiata) e $h = 1/60$ (linea continua)

```
>> [z,uhex]=bvp(0,1,999,0,0.1,f,0,0);
>> max(abs(uh11-uhex(1:100:end)))
ans =
8.6782e-04
>> max(abs(uh21-uhex(1:50:end)))
ans =
2.0422e-04
>> max(abs(uh41-uhex(1:25:end)))
ans =
5.2789e-05
```

Dimezzando h l'errore si divide dunque per 4, a conferma dell'ordine 2 rispetto a h .

Soluzione 8.6 Per individuare il più grande h_{crit} che restituisce una soluzione monotona (come la soluzione analitica) eseguiamo il seguente ciclo:

```
>> f=inline('1+0.*x','x');
for k=3:1000
[z,uh]=bvp(0,1,k,100,0,f,0,1);
if sum(diff(uh)>0)==length(uh)-1,
break, end, end
```

Facciamo cioè variare $h (= 1/(k+1))$ finché i rapporti incrementali in avanti della soluzione numerica uh non sono tutti positivi ($\text{diff}(uh)>0$) restituisce un vettore con componenti uguale a 1 se il rapporto incrementale corrispondente è positivo, 0 in caso contrario; quando la somma delle sue componenti egualia la lunghezza del vettore uh diminuita di 1, significa che tutti i rapporti incrementali sono positivi).

Eseguendo il ciclo troviamo che esso si arresta per $k=49$ cioè per $h = 1/50$. Ripetendo lo stesso calcolo per $\delta = 1000$ e $\delta = 2000$, troviamo che il ciclo si arresta per $h = 1/500$ e $h = 1/1000$. Possiamo allora pensare che affinché la soluzione sia monotona, si debba richiedere $h < 2/\delta = h_{crit}(\delta)$. In effetti questa è anche la condizione che si ricava attraverso un'indagine teorica. Per una spiegazione di questo comportamento, si veda ad esempio [Qua06, Capitolo 5]. In Figura 9.17 riportiamo le soluzioni trovate quando $\delta = 100$ per vari valori di h .

Soluzione 8.7 Si tratta di modificare il Programma 25 in modo da incorporare le condizioni di Neumann. Un esempio è fornito nel Programma 30.

Programma 30 - neumann : approssimazione di un problema ai limiti di Neumann

```
function [x,uh]=neumann(a,b,N,delta,gamma,f,ua,ub)
h = (b-a)/(N+1); x = [a:h:b]; e = ones(N+2,1);
A = spdiags([-e-0.5*h*delta 2*e+gamma*h^2 -e+0.5*h*delta], -1:1, N+2, N+2);
f = h^2*eval(f); f=f';
A(1,1)=-3/2*h; A(1,2)=2*h; A(1,3)=-1/2*h; f(1)=h^2*ua;
A(N+2,N+2)=3/2*h; A(N+2,N+1)=-2*h; A(N+2,N)=1/2*h; f(N+2)=h^2*ub;
uh = A\f;
```

Soluzione 8.8 La formula di integrazione del trapezio, applicata su ciascun intervallo I_{k-1} e I_k , fornisce il seguente valore:

$$\int_{I_{k-1} \cup I_k} f(x)\varphi_k(x) dx = \frac{h}{2} f(x_k) + \frac{h}{2} f(x_{k-1}) = hf(x_k),$$

essendo $\varphi_k(x_j) = \delta_{jk}$. Si ottiene quindi lo stesso termine noto del metodo delle differenze finite.

Soluzione 8.9 Abbiamo $\nabla\phi = (\partial\phi/\partial x, \partial\phi/\partial y)^T$ e, di conseguenza, $\operatorname{div}\nabla\phi = \partial^2\phi/\partial x^2 + \partial^2\phi/\partial y^2$ che è proprio il laplaciano di ϕ .

Soluzione 8.10 Per calcolare la temperatura al centro della piastra, risolviamo il corrispondente problema di Poisson per vari valori di $\Delta_x = \Delta_y$ dando le seguenti istruzioni:

```
>> k=0; for N = [10,20,40,80], [u,x,y]=poisondf(0,1,0,1,N,N,'25+0.*x','(x==1)');
k=k+1; uc(k) = u(N/2+1,N/2+1); end
```

In uc sono stati memorizzati i valori della temperatura, calcolati al centro della piastra al decrescere del passo di griglia. Troviamo

```
>> uc
2.0775 2.0882 2.0909 2.0916
```

E quindi possiamo ritenere che la temperatura della piastra al centro sia di circa 2.09 C. In Figura 9.18 riportiamo le linee di livello della soluzione calcolata per due diversi valori del passo di griglia.

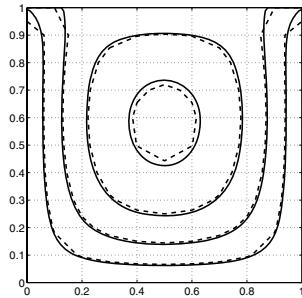


Figura 9.18. In linea tratteggiata le isoline della temperatura calcolata per $\Delta_x = \Delta_y = 1/10$, in linea piena quelle relative a $\Delta_x = \Delta_y = 1/80$

Riferimenti bibliografici

- [ABB⁺99] Anderson E., Bai Z., Bischof C., Blackford S., Demmel J., Dongarra J., Croz J. D., Greenbaum A., Hammarling S., McKenney A., and Sorensen D. (1999) *LAPACK User's Guide*. SIAM, Philadelphia, third edition.
- [Ada90] Adair R. (1990) *The physics of baseball*. Harper and Row, New York NY.
- [Arn73] Arnold V. (1973) *Ordinary Differential Equations*. The MIT Press, Cambridge, Massachusetts.
- [Atk89] Atkinson K. (1989) *An Introduction to Numerical Analysis*. John Wiley, New York.
- [Axe94] Axelsson O. (1994) *Iterative Solution Methods*. Cambridge University Press, New York.
- [BB96] Brassard G. and Bratley P. (1996) *Fundamentals of Algorithms*, 1/e. Prentice Hall, New York.
- [BC98] Bernasconi A. and Codenotti B. (1998) *Introduzione alla complessità computazionale*. Springer-Verlag Italia, Milano.
- [BM92] Bernardi C. and Maday Y. (1992) *Approximations Spectrales des Problèmes aux Limites Elliptiques*. Springer-Verlag, Paris.
- [Bra97] Braess D. (1997) *Finite Elements: Theory, Fast Solvers and Applications in Solid Mechanics*. Cambridge University Press, Cambridge.
- [BS01] Babuska I. and Strouboulis T. (2001) *The Finite Element Method and its Reliability*. Oxford University Press.
- [But87] Butcher J. (1987) *The Numerical Analysis of Ordinary Differential Equations: Runge-Kutta and General Linear Methods*. Wiley, Chichester.
- [CHQZ06] Canuto C., Hussaini M. Y., Quarteroni A., and Zang T. A. (2006) *Spectral Methods: Fundamentals in Single Domains*. Springer, Heidelberg.
- [CLW69] Carnahan B., Luther H., and Wilkes J. (1969) *Applied Numerical Methods*. John Wiley and Sons, Inc.
- [Com95] Comincioli V. (1995) *Analisi Numerica Metodi Modelli Applicazioni*. McGraw-Hill Libri Italia, Milano.

- [Dav63] Davis P. (1963) *Interpolation and Approximation*. Blaisdell Pub., New York.
- [DD95] Davis T. and Duff I. (1995) *A Combined Unifrontal/Multifrontal Method for Unsymmetric Sparse Matrices*. TR-95-020. University of Florida.
- [Dem97] Demmel J. (1997) *Applied Numerical Linear Algebra*. SIAM, Philadelphia.
- [Die93] Dierckx P. (1993) *Curve and Surface Fitting with Splines*. Clarendon Press, New York.
- [DL92] DeVore R. and Lucier J. (1992) Wavelets. *Acta Numerica* pages 1–56.
- [DR75] Davis P. and Rabinowitz P. (1975) *Methods of Numerical Integration*. Academic Press, New York.
- [DS83] Dennis J. and Schnabel R. (1983) *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Prentice-Hall, Englewood Cliffs, New York.
- [dV89] der Vorst H. V. (1989) High Performance Preconditioning. *SIAM J. Sci. Stat. Comput.* 10: 1174–1185.
- [EEHJ96] Eriksson K., Estep D., Hansbo P., and Johnson C. (1996) *Computational Differential Equations*. Cambridge Univ. Press, Cambridge.
- [EKH02] Etter D., Kuncicky D., and Hull D. (2002) *Introduction to MATLAB 6*. Prentice Hall.
- [Fun92] Funaro D. (1992) *Polynomial Approximation of Differential Equations*. Springer-Verlag, Berlin.
- [Gau97] Gautschi W. (1997) *Numerical Analysis. An Introduction*. Birkhäuser, Berlin.
- [Gea71] Gear C. (1971) *Numerical Initial Value Problems in Ordinary Differential Equations*. Prentice-Hall, Upper Saddle River NJ.
- [Gio97] Giordano N. (1997) *Computational physics*. Prentice-Hall, Upper Saddle River NJ.
- [GL89] Golub G. and Loan C. V. (1989) *Matrix Computations*. The John Hopkins Univ. Press, Baltimore and London.
- [Hac85] Hackbusch W. (1985) *Multigrid Methods and Applications*. Springer-Verlag, Berlin.
- [Hac94] Hackbusch W. (1994) *Iterative Solution of Large Sparse Systems of Equations*. Springer-Verlag, New York.
- [HH00] Higham D. and Higham N. (2000) *MATLAB Guide*. SIAM, Philadelphia.
- [Hir88] Hirsh C. (1988) *Numerical Computation of Internal and External Flows*, volume 1. John Wiley and Sons, Chichester.
- [HLR01] Hunt B., Lipsman R., and Rosenberg J. (2001) *A guide to MATLAB: for Beginners and Experienced Users*. Cambridge University Press.
- [IK66] Isaacson E. and Keller H. (1966) *Analysis of Numerical Methods*. Wiley, New York.
- [Krö98] Kröner D. (1998) *Finite volume schemes in multidimensions*. Pitman Res. Notes Math. Ser., 380, Longman, Harlow.
- [KS99] Karniadakis G. and Sherwin S. (1999) *Spectral/hp Element Methods for CFD*. Oxford University Press.

- [Lam91] Lambert J. (1991) *Numerical Methods for Ordinary Differential Systems*. John Wiley and Sons, Chichester.
- [Lan03] Langtangen H. (2003) *Computational Partial Differential Equations: Numerical Methods and Diffpack Programming*. Springer-Verlag, Heidelberg.
- [LeV02] LeVeque R. (2002) *Finite Volume Methods for Hyperbolic Problems*. Cambridge University Press, Cambridge.
- [Mat94] The MathWorks Inc., 24 Prime Park Way, Natick, Mass. 01760 (1994) *MATLAB User's Guide*.
- [Mei67] Meinardus G. (1967) *Approximation of Functions: Theory and Numerical Methods*. Springer-Verlag, New York.
- [MH03] Marchand P. and Holland O. (2003) *Graphics and Guis With Matlab*. CRC Press.
- [Pal04] Palm W. (2004) *Introduction to Matlab 7 for Engineers*. McGraw-Hill.
- [Pan92] Pan V. (1992) Complexity of Computations with Matrices and Polynomials. *SIAM Review* 34: 225–262.
- [PBP02] Prautzsch H., Boehm W., and Paluszny M. (2002) *Bzier and B-Spline Techniques*. Springer-Verlag.
- [PdDKÜK83] Piessens R., de Doncker-Kapenga E., Üeberhuber C., and Kahaner D. (1983) *QUADPACK: A Subroutine Package for Automatic Integration*. Springer-Verlag, Berlin and Heidelberg.
- [QSS02] Quarteroni A., Sacco R., and Saleri F. (2002) *Matematica Numerica*. Springer-Verlag Italia, Milano, 2 edizione.
- [QSS04] Quarteroni A., Sacco R., and Saleri F. (2004) *Numerical Mathematics*, volume 37 of *Texts in Applied Mathematics*. Springer-Verlag, New York, 2 edition.
- [Qua06] Quarteroni A. (2006) *Modellistica Numerica per Problemi Differenziali*. Springer-Verlag Italia, Milano, terza edizione.
- [QV94] Quarteroni A. and Valli A. (1994) *Numerical Approximation of Partial Differential Equations*. Springer, Berlin and Heidelberg.
- [RR85] Ralston A. and Rabinowitz P. (1985) *A First Course in Numerical Analysis*. McGraw-Hill, Singapore. 7h printing.
- [Saa92] Saad Y. (1992) *Numerical Methods for Large Eigenvalue Problems*. Halstead Press, New York.
- [Saa96] Saad Y. (1996) *Iterative Methods for Sparse Linear Systems*. PWS Publishing Company, Boston.
- [TW98] Tveito A. and Winther R. (1998) *Introduction to Partial Differential Equations. A Computational Approach*. Springer Verlag.
- [Üeb97] Üeberhuber C. (1997) *Numerical Computation: Methods, Software, and Analysis*. Springer-Verlag, Berlin Heidelberg.
- [Urb02] Urban K. (2002) *Wavelets in Numerical Simulation*. Springer Verlag.
- [vdV03] van der Vorst H. (2003) *Iterative Krylov Methods for Large Linear systems*. Cambridge University Press, Cambridge.
- [Wes04] Wesseling P. (2004) *An Introduction to Multigrid Methods*. R.T. Edwards, Inc.
- [Wil65] Wilkinson J. (1965) *The Algebraic Eigenvalue Problem*. Clarendon Press, Oxford.

Indice dei programmi MATLAB

bisection	il metodo di bisezione	41
newton	il metodo di Newton	46
newtonsys	il metodo di Newton per un sistema non lineare.....	48
aitken	il metodo di Aitken	57
horner	il metodo di divisione sintetica.....	60
newtonhorner	il metodo di Newton-Hörner	62
cubicspline	spline cubica interpolante	87
midpointc	formula composita del punto medio	105
simpsonc	formula composita di Simpson	108
simpadpt	formula di Simpson adattiva.....	116
lu_gauss	la fattorizzazione di Gauss	129
itermeth	un metodo iterativo generale	145
eigpower	il metodo delle potenze	169
invshift	il metodo delle potenze inverse con <i>shift</i> ..	173
gershcircles	i cerchi di Gershgorin.....	175
basisQR	il metodo delle iterazioni QR	178
feuler	il metodo di Eulero in avanti	190
beuler	il metodo di Eulero all'indietro	190
cranknic	il metodo di Crank-Nicolson	196
predcor	un generico metodo predictor-corrector ..	215
onestep	un passo del metodo di Eulero in avanti (feonestep), un passo del metodo di Eulero all'indietro (beonestep), uno di Crank-Nicolson (cnonestep)	216
newmark	il metodo di Newmark	221
fvinc	termine forzante per il problema del pendolo sferico	225
threebody	termine forzante per il problema dei tre corpi semplificato	226

bvp	approssimazione di un problema ai limiti con il metodo delle differenze finite	239
poissonfd	approssimazione del problema di Poisson con condizioni di Dirichlet usando il metodo a 5 punti delle differenze finite	247
gausslegendre	formula di quadratura di Gauss-Legendre con $n = 1$	270
rk2	metodo Runge-Kutta esplicito di ordine 2 . .	284
rk3	metodo Runge-Kutta esplicito di ordine 3 . .	285
neumann	approssimazione di un problema ai limiti di Neumann	293

Indice analitico

- `...`, 32
- `=`, 28
- `abs`, 7
- adattività, 86, 113
- Aitken
 - estrapolazione di, 55
 - metodo di, 56
- algoritmo, 24
 - delle sostituzioni all'indietro, 125
 - delle sostituzioni in avanti, 125
 - di divisione sintetica, 60
 - di Gauss, 127
 - di Strassen, 25
 - di Thomas, 138
- `aliasing`, 83
- `angle`, 7
- `ans`, 27
- `arpackc`, 181
- array di Butcher, 209
- arrotondamento, 2
- assoluta stabilità, 200
 - regione di, 202
 - assoluta stabilità, regione di, 212
- autovalore, 14, 165
- autovettore, 14, 165
- `axis`, 175
- base, 3
- `bisezione`, 41
- bisezione, il metodo di, 39
- `break`, 257
- cancellazione, 5
- cerchi di Gershgorin, 174
- `char`, 77
- Chebyshev
 - interpolazione di, 78
- `chol`, 131, 141
- `cholinc`, 156
- cifre significative, 3
- `clear`, 28
- `compass`, 7
- complessità, 24, 25
- `complex`, 6
- compressione di immagini, 168
- `cond`, 136
- `condest`, 136
- condizione
 - di Dirichlet, 237
 - di Neumann, 252
- `conj`, 7
- consistenza, 142
- `contour`, 286
- `conv`, 18
- convergenza, 23
 - lineare, 52
 - metodo di Eulero, 191
 - metodo di Richardson, 148
 - metodo iterativo, 143
 - quadratica, 44
- `cos`, 28
- costo computazionale, 24
- `cputime`, 26
- Cramer, regola di, 123

Crank-Nicolson, metodo di, 195
`cross`, 13
`cumtrapz`, 107

`dblquad`, 117
 decomposizione
 in valori singolari, 166
`deconv`, 18
 deflazione, 61, 181
 demografia, 167
 derivata
 di una funzione, 20
 parziale, 47, 235
`det`, 162
`det`, 10, 129
 determinante, 10, 129
`diag`, 11
 diagonale principale, 9, 11
`diff`, 20
 differenze finite
 in dimensione 1, 238
 in dimensione 2, 243
 schema a 5 punti, 244
 differenziazione numerica, 101
`disp`, 29, 257
`dot`, 13

`eig`, 177
`eigs`, 179
 elementi
 pivot, 127
 elementi finiti, 240
`end`, 26
`eps`, 3, 4
 ϵ_M , 3
 equazione
 alle derivate parziali, 185
 del calore, 236
 delle onde, 236
 di Poisson, 235
 differenziale ordinaria, 185
 equazioni
 di Lotka-Volterra, 186
 normali, 139
 errore
 assoluto, 3
 computazionale, 22
 di arrotondamento, 6, 22, 133, 171
 di perturbazione, 204
 di roundoff, 3
 di troncamento, 22, 249
 locale, 249
 relativo, 3
 stimatore, 24, 113
 esponente, 3
`etime`, 26
 Eulero migliorato
 metodo di, 214
`eulesp`, 190
`eval`, 15
`exit`, 27
`exp`, 28
`eye`, 9

 \mathbb{F} , 3
 fattorizzazione
 di Cholesky, 131, 173
 LU, 124, 173
 QR, 140, 178
`feval`, 15, 34
`fft`, 81
`figure`, 175
`find`, 41
`fix`, 257
`flops`, 24
`flops`, 128
`for`, 30
`format`, 2
 formula
 di Adams-Bashforth, 211
 di Adams-Moulton, 211
 di Eulero, 6
 di Simpson, 107
 formulazione debole, 240
 formule
 di Newton-Cotes, 117
`fplot`, 14
`fsolve`, 64
`function`, 31
`funtool`, 21
 funzione
 derivabile, 20
 di forma, 242
 di iterazione, 50
 grafico di una, 14
 lipschitziana, 188
 reale, 14
`fzero`, 16, 64

zerof, 190
gallery, 158
 Gauss
 algoritmo di, 127
 fattorizzazione di, 127
 Gauss-Seidel, metodo di, 146
gmres, 153
grid, 15
griddata, 96
griddata3, 96
griddatan, 96

help, 28, 34
 Heun, metodo di, 214
 Hilbert matrice di, 134
hold off, 175
hold on, 175

if, 26
ifft, 81
imag, 7
 incremento, 154
Inf, 4
inline, 34
int, 20
 integrazione numerica, 103
interp1, 85
interp1q, 85
interp2, 95
interp3, 95
interpft, 82
 interpolatore, 72
 di Hermite, 90
 polinomiale, 72
 razionale, 73
 trigonometrico, 72, 79
 interpolazione
 con funzioni spline, 86
 di Chebyshev, 78
 lineare composita, 84
 nodi di, 72
 polinomiale di Lagrange, 73
inv, 10
 inversa di una matrice, 274

 Jacobi, metodo di, 144

 Lagrange, forma di, 74

load, 28
loglog, 23
lu, 129, 141
luinc, 160

 m-file, 30
magic, 161
 mantissa, 3
 matrice, 8
 a dominanza diagonale, 131
 a dominanza diagonale stretta, 144, 147
 bidiagonale, 138
 definita positiva, 131, 147
 di Hilbert, 134
 di iterazione, 142
 di Leslie, 167
 di precondizionamento, 149
 di Wilkinson, 182
 diagonale, 11
 diagonalizzabile, 165
 hermitiana, 169
 identità, 9
 inversa, 10
 jacobiana, 47
 quadrata, 9
 simmetrica, 12, 131
 sparsa, 139, 141, 156, 159, 246
 trasposta, 12
 triangolare inferiore, 11
 triangolare superiore, 11
 tridiagonale, 138
 unitaria, 165
 Vandermonde, 128
 media, 98
 membrana, 248
mesh, 247
meshgrid, 96
 metodi
 di Krylov, 153, 161
 iterativi, 142
 multipasso, 211
 metodo
 A-stabile, 202
 ad un passo, 189
 BiCGstab, 153
 CGS, 153
 degli elementi finiti, 240
 dei minimi quadrati, 90

- del gradiente coniugato, 150
- delle iterazioni QR, 177
- delle potenze inverse, 172
- delle potenze inverse con shift, 173
- di Adams-Bashforth, 211
- di Adams-Moulton, 211
- di Aitken, 56
- di Bairstow, 64
- di bisezione, 39
- di Broyden, 64
- di Dekker-Brent, 64
- di Eulero in avanti adattivo, 200
- di Gauss-Seidel, 146
- di Hörner, 60
- di Jacobi, 144
- di Müller, 64
- di Newton, 43
- di Newton-Hörner, 61
- di Richardson dinamico, 148
- di Richardson stazionario, 148
- di Steffensen, 56
- di Thomas, 239
- esplicito, 189
- GMRES, 153
- implicito, 189
- leap frog, 221
- multifrontale, 161
- multistep, 198
- predictor-corrector, 214
- quasi-Newton, 64
- SOR, 163
- spettrale, 251
- metodo iterativo
 - convergenza di un, 142
- minimi quadrati, 91
- mkpp**, 87
- modello
 - di Lotka e Leslie, 167
- multigrid, 161
- multistep, 198
- NaN**, 5
- nargin**, 33
- nargout**, 33
- nchoosek**, 257
- newton**, 45
- Newton, metodo di, 43
- nodi, 109
- norm**, 13
- not-a-knot condition*, 87
- numeri
 - complessi, 6
 - floating-point, 1, 3
 - macchina, 1
 - reali, 1
- numero di condizionamento, 136
- ode113**, 216
- ode15s**, 214
- ode23**, 211, 232, 289
- ode23tb**, 211
- ode45**, 211
- ones**, 12
- operatore
 - di Laplace, 235
 - gradiente, 252
- ordinamento lessicografico, 245
- oscillazioni forzate, 166
- overflow*, 4, 6, 50
- passo di discretizzazione, 189
- patch**, 175
- path**, 30
- pcg**, 152
- pchip**, 89
- pde**, 249, 251
- pdetool**, 96
- pesi, 109
- piano
 - delle fasi, 217
 - di Gauss, 8
- piastra, 253
- pivoting, 132
 - per righe, 132
- \mathbb{P}_n , 16
- polinomi
 - divisione di, 18, 60
 - radici di, 17
- polinomio, 17
 - caratteristico, 74, 165
 - di Legendre, 110
 - di Taylor, 20
 - nodale, 110
- polinomio caratteristico
 - primo, 198
- poly**, 35, 77
- polyder**, 18, 77
- polyfit**, 19, 93

`polyint`, 18
`polyval`, 74
`polyval`, 17
`potenzeshift`, 173
`power`, 169
`ppval`, 87
 precondizionatore, 143, 148
`pretty`, 256
 primitiva, 19
 problema
 ai limiti, 235
 di Cauchy, 188
 di Dirichlet, 237
 di Neumann, 237
 modello, 199
`prod`, 258
 prodotto
 di matrici, 9
 scalare, 13
 vettore, 13
`ptomedioc`, 105
 punti di equilibrio, 217
 punto fisso, 50
 iterazione di, 50

 QR, fattorizzazione, 140
`quadl`, 112
 quadratura
 di Gauss-Legendre-Lobatto, 111
 formula interpolatoria, 109
`quit`, 27
`quiver`, 13
`quiver3`, 13
 quoquiente di Rayleigh, 165

 radice
 multipla, 18
 raggio spettrale, 143
`rand`, 26
 rango
 pieno, 139
`rcond`, 136
`real`, 7
`realmx`, 3
`realmn`, 3
 regola
 di Cartesio, 59
 di Cramer, 123
 di Laplace, 10

 residuo, 45, 137, 153
 precondizionato, 144
 retta di regressione, 92
`return`, 32
 Richardson
 estrapolazione di, 119
`roots`, 17, 64
`rpmak`, 96
`rsmak`, 96
 Runge
 funzione di, 76

`save`, 28
 scala logaritmica, 23
 serie
 discreta di Fourier, 80
 shift, 173
 simbolo di Kronecker, 73
`simpadpt`, 116
`simple`, 277
`simple`, 21
 Simpson
 adattiva
 formula di, 116
 formula di, 107
`sin`, 28
 sistema
 lineare, 121
 sottodeterminato, 126, 139
 sovradeterminato, 139
 triangolare, 125
 somma
 di matrici, 9
`sparse`, 139, 156
`spdemos`, 96
`spdiags`, 138
 spettro di una matrice, 168
`spline`, 86
 cubica naturale, 86
`spline`, 87
`spy`, 156, 246
`sqrt`, 28
 stabilità assoluta
 condizionata, 202
 incondizionata, 202
 stadi, 209
 Steffensen, metodo di, 56
 stimatore dell'errore, 24, 45
 Sturm, successioni di, 64, 181

successione di Fibonacci, 30, 36
sum, 257
svd, 180
svd, 180
syms, 277
syms, 20

taylor, 20
taylortool, 71
tempo di CPU, 26
teorema
 del valor medio, 20
 della media integrale, 19
 di Abel, 59
 di Cauchy, 59
 di Lax-Ritchmyer, 198
 fondamentale del calcolo integrale,
 19
Thomas, algoritmo di, 138
title, 175
toolbox, 17
trapz, 107
trasformata
 rapida di Fourier, 81
tril, 12
triu, 12

underflow, 4

valori singolari, 166, 179
vander, 128
varargin, 41
varianza, 98
vettore
 colonna, 9
 riga, 9
vettori
 linearmente indipendenti, 13
viabilità, 166

wavelet, 96
wavelet, 96
while, 30
wilkinson, 182

xlabel, 175

ylabel, 175

zero
 di una funzione, 15
 multiplo, 15
 semplice, 15, 44
zero-stabilità, 197
zeros, 9, 12

Springer - Collana Unitext

a cura di

Franco Brezzi
Ciro Ciliberto
Bruno Codenotti
Mario Pulvirenti
Alfio Quarteroni

Volumi pubblicati

A. Bernasconi, B. Codenotti
Introduzione alla complessità computazionale
1998, X+260 pp. ISBN 88-470-0020-3

A. Bernasconi, B. Codenotti, G. Resta
Metodi matematici in complessità computazionale
1999, X+364 pp, ISBN 88-470-0060-2

E. Salinelli, F. Tomarelli
Modelli dinamici discreti
2002, XII+354 pp, ISBN 88-470-0187-0

A. Quarteroni
Modellistica numerica per problemi differenziali (2a Ed.)
2003, XII+334 pp, ISBN 88-470-0203-6
(1a edizione 2000, ISBN 88-470-0108-0)

S. Bosch
Algebra
2003, VIII+380 pp, ISBN 88-470-0221-4

S. Graffi, M. Degli Esposti
Fisica matematica discreta
2003, X+248 pp, ISBN 88-470-0212-5

S. Margarita, E. Salinelli
MultiMath - Matematica Multimediale per l'Università
2004, XX+270 pp, ISBN 88-470-0228-1

A. Quarteroni, R. Sacco, F. Saleri
Matematica numerica (2a Ed.)
2000, XIV+448 pp, ISBN 88-470-0077-7
2002, 2004 ristampa riveduta e corretta
(1a edizione 1998, ISBN 88-470-0010-6)

A partire dal 2004, i volumi della serie sono contrassegnati da un numero di identificazione

13. A. Quarteroni, F. Saleri
Introduzione al Calcolo Scientifico (2a Ed.)
2004, X+262 pp, ISBN 88-470-0256-7
(1a edizione 2002, ISBN 88-470-0149-8)
14. S. Salsa
Equazioni a derivate parziali - Metodi, modelli e applicazioni
2004, XII+426 pp, ISBN 88-470-0259-1
15. G. Riccardi
Calcolo differenziale ed integrale
2004, XII+314 pp, ISBN 88-470-0285-0
16. M. Impedovo
Matematica generale con il calcolatore
2005, X+526 pp, ISBN 88-470-0258-3
17. L. Formaggia, F. Saleri, A. Veneziani
Applicazioni ed esercizi di modellistica numerica
per problemi differenziali
2005, VIII+396 pp, ISBN 88-470-0257-5
18. S. Salsa, G. Verzini
Equazioni a derivate parziali - Complementi ed esercizi
2005, VIII+406 pp, ISBN 88-470-0260-5
19. C. Canuto, A. Tabacco
Analisi Matematica I (2a Ed.)
2005, XII+448 pp, ISBN 88-470-0337-7
(1a edizione, 2003, XII+376 pp, ISBN 88-470-0220-6)
20. F. Biagini, M. Campanino
Elementi di Probabilità e Statistica
2006, XII+236 pp, ISBN 88-470-0330-X

21. S. Leonesi, C. Toffalori
Numeri e Crittografia
2006, VIII+178 pp, ISBN 88-470-0331-8

22. A. Quarteroni, F. Saleri
Introduzione al Calcolo Scientifico (3a Ed.)
2006, X+306 pp, ISBN 88-470-0480-2