



Alfio Quarteroni · Fausto Saleri
Paola Gervasio

Calcolo Scientifico

Esercizi e problemi risolti
con MATLAB e Octave

6a edizione

 Springer

UNITEXT – La Matematica per il 3+2

Volume 105

Editor-in-Chief

A. Quarteroni

Series Editors

L. Ambrosio

P. Biscari

C. Ciliberto

C. De Lellis

M. Ledoux

V. Panaretos

W.J. Runggaldier

www.springer.com/series/5418

Alfio Quarteroni • Fausto Saleri • Paola Gervasio

Calcolo Scientifico

Esercizi e problemi risolti
con MATLAB e Octave

6a edizione



Alfio Quarteroni
École Polytechnique Fédérale (EPFL)
Lausanne, Switzerland
and Politecnico di Milano
Milan, Italy

Paola Gervasio
DICATAM
Università degli Studi di Brescia
Brescia, Italy

Fausto Saleri (1965–2007)
MOX
Politecnico di Milano
Milan, Italy

ISSN versione cartacea: 2038-5722
UNITEXT – La Matematica per il 3+2
ISBN 978-88-470-3952-0
DOI 10.1007/978-88-470-3953-7

ISSN versione elettronica: 2038-5757
ISBN 978-88-470-3953-7 (eBook)

Springer Milan Heidelberg New York Dordrecht London
© Springer-Verlag Italia Srl. 2002, 2004, 2006, 2008, 2012, 2017

Quest'opera è protetta dalla legge sul diritto d'autore e la sua riproduzione è ammessa solo ed esclusivamente nei limiti stabiliti dalla stessa. Le fotocopie per uso personale possono essere effettuate nei limiti del 15% di ciascun volume dietro pagamento alla SIAE del compenso previsto dall'art. 68. Le riproduzioni per uso non personale e/o oltre il limite del 15% potranno avvenire solo a seguito di specifica autorizzazione rilasciata da AIDRO, Corso di Porta Romana n. 108, Milano 20122, e-mail segreteria@aidro.org e sito web www.aidro.org.

Tutti i diritti, in particolare quelli relativi alla traduzione, alla ristampa, all'utilizzo di illustrazioni e tavole, alla citazione orale, alla trasmissione radiofonica o televisiva, alla registrazione su microfilm o in database, o alla riproduzione in qualsiasi altra forma (stampata o elettronica) rimangono riservati anche nel caso di utilizzo parziale. La violazione delle norme comporta le sanzioni previste dalla legge.

L'utilizzo in questa pubblicazione di denominazioni generiche, nomi commerciali, marchi registrati, ecc. anche se non specificatamente identificati, non implica che tali denominazioni o marchi non siano protetti dalle relative leggi e regolamenti.

Immagine di copertina: La figura di copertina rappresenta l'approssimazione di un'autofunzione dell'operatore di Laplace ottenuta con il metodo degli elementi spettrali.

Questa edizione è pubblicata da SpringerNature
La società registrata è Springer-Verlag Italia Srl

A Fausto

Prefazione alle precedenti edizioni

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 simbolo per segnalare un programma ed il simbolo per attirare

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>.

VIII Prefazione alle precedenti edizioni

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 [QSS07] e [QSSG14] per i rimandi di carattere teorico o per gli approfondimenti, mentre per una descrizione completa di MATLAB rimandiamo a [HH17]. Tutti i programmi presenti nel volume possono essere trovati all'indirizzo:

mox.polimi.it/qs.

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 .

La 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 espansiva.

Losanna e Milano
febbraio 2006

*Alfio Quarteroni
Fausto Saleri*

La quarta edizione di questo testo si caratterizza per numerose e significative novità.

L'ambiente MATLAB è stato affiancato da Octave, una reimplementazione di MATLAB distribuita gratuitamente secondo le condizioni d'uso della GNU General Public License. Tutti gli esercizi e i problemi sono risolti con programmi che possono essere eseguiti in entrambi gli ambienti.

I capitoli relativi all'approssimazione di problemi alle derivate parziali, ellittici, parabolici ed iperbolici, sono stati notevolmente arricchiti da nuovi tipi di equazioni (fra cui quelle di trasporto e di diffusione-trasporto) nonché da nuovi metodi di discretizzazione alle differenze finite ed agli elementi finiti.

Sono stati eliminati alcuni accorgimenti tipografici a margine del testo, al fine di rendere più fruibile ed autonomo l'approccio agli argomenti trattati da parte del lettore, sia esso docente o studente.

Infine sono stati aggiunti nuovi problemi di interesse applicativo e numerosi esercizi con relative tracce di soluzioni.

Tutto questo è stato reso possibile grazie al contributo straordinario (per quantità e qualità) di Paola Gervasio. A lei va il mio ringraziamento e la mia stima.

Losanna e Milano
giugno 2008

Alfio Quarteroni

La quinta edizione si caratterizza per l'aggiunta di un nuovo capitolo sull'ottimizzazione numerica. In esso vengono presentate, discusse ed analizzate diverse famiglie di metodi per la minimizzazione di funzioni di una o più variabili. Per problemi di minimizzazione non vincolata sono presentati i metodi *derivative free*, quelli di discesa (o di tipo *line search*) e quelli di tipo *trust region*. Per quanto riguarda la minimizzazione vincolata abbiamo limitato la scelta a due metodi, quello della penalizzazione e quello della Lagrangiana aumentata.

Coerentemente con lo stile del libro, anche questo capitolo è corredata di esempi, esercizi e programmi eseguibili negli ambienti MATLAB ed Octave.

L'introduzione di questo argomento ha reso necessario rinumerare alcuni capitoli rispetto alle edizioni precedenti. Inoltre alcuni capitoli sono stati arricchiti con nuove sezioni di carattare sia teorico che pratico.

Ricordiamo infine ai lettori che tutti i programmi presentati in questo volume possono essere scaricati dalla pagina web

<http://mox.polimi.it/qs>

Losanna, Milano e Brescia
luglio 2012

Alfio Quarteroni
Paola Gervasio

Prefazione alla sesta edizione

In questa sesta edizione abbiamo rivisitato gli ambienti di programmazione MATLAB e Octave, aggiornandoli e unificandoli, laddove possibile, sotto un unico *logo* a cui è stato dato il nome di fantasia **MAT&OCT**. Questo stratagemma ci ha consentito di operare numerose semplificazioni a livello espositivo.

Diversi capitoli sono stati integrati con nuovi sviluppi. In particolare segnaliamo: l'introduzione della formula di interpolazione baricentrica nel Capitolo 3 e del Metodo Monte Carlo per l'integrazione numerica nel Capitolo 4; una nuova e più efficace presentazione dei metodi iterativi per sistemi lineari nel Capitolo 5; una più approfondita analisi di stabilità per il problema di Cauchy nel Capitolo 8; un approfondimento dell'analisi del metodo degli elementi finiti nel Capitolo 9. Inoltre sono stati introdotti nuovi esempi, con particolare riferimento ad applicazioni di interesse reale e svariati esercizi con relative soluzioni.

Come sempre, i lettori possono scaricare tutti i programmi presentati in questo volume dalla pagina web

<http://mox.polimi.it/qs>

Losanna, Milano e Brescia
giugno 2017

*Alfio Quarteroni
Paola Gervasio*

Indice

1	Quel che non si può non sapere	1
1.1	Gli ambienti MATLAB e Octave	1
1.2	I numeri reali	3
1.2.1	Come si rappresentano	4
1.2.2	Come si opera con i numeri floating-point	6
1.3	I numeri complessi	9
1.4	Le matrici	11
1.4.1	I vettori	15
1.4.2	Autovalori e autovettori di una matrice	17
1.5	Le strutture e i cell-array	17
1.6	Le funzioni	19
1.6.1	Gli zeri	22
1.6.2	I polinomi	23
1.6.3	L'integrale e la derivata	25
1.7	Errare non è solo umano	28
1.7.1	Parliamo di costi	32
1.8	Qualche parola in più su MATLAB e Octave	35
1.9	Programmare in MATLAB e Octave	39
1.10	Cosa non vi abbiamo detto	43
1.11	Esercizi	44
2	Equazioni non lineari	47
2.1	Alcuni problemi	47
2.2	Il metodo di bisezione	50
2.3	Il metodo di Newton	54
2.3.1	Come arrestare il metodo di Newton	56
2.4	Il metodo delle secanti	58
2.5	I sistemi di equazioni non lineari	59
2.6	Iterazioni di punto fisso	64
2.6.1	Come arrestare un'iterazione di punto fisso	69

2.7	Accelerazione con il metodo di Aitken	70
2.8	Polinomi algebrici	74
2.8.1	Il metodo di Hörner	75
2.8.2	Il metodo di Newton-Hörner	77
2.9	Cosa non vi abbiamo detto	79
2.10	Esercizi	81
3	Approssimazione di funzioni e di dati	85
3.1	Alcuni problemi	85
3.2	Approssimazione con i polinomi di Taylor	87
3.3	Interpolazione	89
3.3.1	Interpolazione polinomiale di Lagrange	90
3.3.2	Stabilità dell'interpolazione polinomiale	95
3.3.3	Interpolazione rispetto ai nodi di Chebyshev	96
3.3.4	Formula di interpolazione baricentrica	98
3.3.5	Interpolazione trigonometrica e FFT	101
3.4	Interpolazione lineare composita	107
3.5	Approssimazione con funzioni <i>spline</i>	108
3.6	Il metodo dei minimi quadrati	113
3.7	Cosa non vi abbiamo detto	118
3.8	Esercizi	119
4	Differenziazione ed integrazione numerica	123
4.1	Alcuni problemi	123
4.2	Approssimazione delle derivate	126
4.3	Integrazione numerica	130
4.3.1	La formula del punto medio	130
4.3.2	La formula del trapezio	133
4.3.3	La formula di Simpson	134
4.4	Formule di quadratura interpolatorie	135
4.5	La formula di Simpson adattiva	140
4.6	Metodi Monte Carlo per l'integrazione numerica	143
4.7	Cosa non vi abbiamo detto	145
4.8	Esercizi	146
5	Sistemi lineari	151
5.1	Alcuni problemi	151
5.2	Sistemi e complessità	156
5.3	Il metodo di fattorizzazione LU	158
5.4	La tecnica del pivoting	169
5.4.1	Il <i>fill-in</i> di una matrice	172
5.5	Quanto è accurata la risoluzione di un sistema lineare? .	174
5.6	Come risolvere un sistema tridiagonale	178
5.7	Sistemi sovradeterminati	179
5.8	Cosa si nasconde dietro al comando \	182

5.9	Metodi iterativi	183
5.9.1	Come costruire un metodo iterativo	184
5.10	I metodi del Gradiente e del Gradiente Coniugato	190
5.10.1	Precondizionatori e metodi iterativi precondizionati	197
5.10.2	Il caso non simmetrico	204
5.11	Quando conviene arrestare un metodo iterativo	208
5.12	Ed ora: metodi diretti o iterativi?	211
5.13	Cosa non vi abbiamo detto	216
5.14	Esercizi	217
6	Autovalori ed autovettori	221
6.1	Alcuni problemi	222
6.2	Il metodo delle potenze	226
6.2.1	Analisi di convergenza	229
6.3	Generalizzazione del metodo delle potenze	231
6.4	Come calcolare lo shift	234
6.5	Calcolo di tutti gli autovalori	237
6.6	Cosa non vi abbiamo detto	240
6.7	Esercizi	241
7	Ottimizzazione numerica	243
7.1	Alcuni problemi	244
7.2	Ottimizzazione non vincolata	247
7.3	Metodi <i>derivative free</i>	249
7.3.1	I metodi della sezione aurea e dell'interpolazione quadratica	249
7.3.2	Il metodo di Nelder e Mead	254
7.4	Il metodo di Newton	257
7.5	Metodi di discesa o <i>line-search</i>	258
7.5.1	Direzioni di discesa	259
7.5.2	Strategie per il calcolo del passo α_k	261
7.5.3	Il metodo di discesa con direzioni di Newton	267
7.5.4	Metodi di discesa con direzioni quasi-Newton	268
7.5.5	Metodi di discesa del gradiente e del gradiente coniugato	270
7.6	Metodi di tipo <i>trust region</i>	272
7.7	Il metodo dei minimi quadrati non lineari	280
7.7.1	Il metodo di Gauss-Newton	281
7.7.2	Il metodo di Levenberg-Marquardt	284
7.8	Ottimizzazione vincolata	285
7.8.1	Il metodo di penalizzazione	291
7.8.2	Il metodo della Lagrangiana aumentata	296
7.9	Cosa non vi abbiamo detto	300
7.10	Esercizi	300

8 Equazioni differenziali ordinarie	303
8.1 Alcuni problemi	303
8.2 Il problema di Cauchy	306
8.3 I metodi di Eulero e il metodo di Crank-Nicolson	308
8.4 Convergenza, Consistenza, Stabilità	311
8.4.1 Consistenza	311
8.4.2 Stabilità	313
8.4.3 Analisi di convergenza per il metodo di Eulero in avanti	315
8.4.4 Stimatori dell'errore per il metodo di Eulero in avanti	318
8.4.5 Analisi di convergenza per metodi ad un passo ..	320
8.5 Stabilità su intervalli illimitati	322
8.5.1 La regione di assoluta stabilità	325
8.6 L-stabilità	326
8.7 L'assoluta stabilità controlla le perturbazioni	328
8.8 Adattività del passo per il metodo di Eulero in avanti ..	334
8.9 Metodi di ordine elevato	339
8.9.1 I metodi Runge-Kutta	339
8.9.2 I metodi multistep	342
8.9.3 I metodi predictor-corrector	347
8.10 Sistemi di equazioni differenziali	350
8.10.1 Equazioni differenziali di ordine superiore a uno ..	352
8.11 Alcuni problemi <i>stiff</i>	356
8.12 Alcuni esempi	362
8.12.1 Il pendolo sferico	362
8.12.2 Il problema dei tre corpi	365
8.12.3 Cinetica chimica	368
8.13 Cosa non vi abbiamo detto	369
8.14 Esercizi	370
9 Metodi numerici per problemi ai limiti stazionari ed evolutivi	373
9.1 Alcuni problemi	374
9.2 Il problema di Poisson con condizioni di Dirichlet e di Neumann	376
9.3 Approssimazione alle differenze finite del problema di Poisson monodimensionale	378
9.3.1 Analisi dell'approssimazione con differenze finite del problema di Poisson monodimensionale	380
9.4 Approssimazione alle differenze finite di un problema di diffusione-trasporto a trasporto dominante	383
9.5 Approssimazione agli elementi finiti del problema di Poisson monodimensionale	385
9.5.1 Cenni all'analisi del metodo agli elementi finiti ..	388

9.6	Approssimazione agli elementi finiti di un problema di diffusione-trasporto a trasporto dominante	391
9.7	Approssimazione alle differenze finite del problema di Poisson in 2 dimensioni	393
9.7.1	Analisi dell'approssimazione con differenze finite del problema di Poisson in 2 dimensioni	399
9.8	Approssimazione dell'equazione del calore monodimensionale	401
9.8.1	Approssimazione alle differenze finite dell'equazione del calore monodimensionale	401
9.8.2	Approssimazione ad elementi finiti dell'equazione del calore monodimensionale	407
9.9	Equazioni iperboliche: un problema di trasporto scalare ..	411
9.9.1	Metodi alle differenze finite per la discretizzazione dell'equazione scalare iperbolica ..	413
9.9.2	Analisi dei metodi alle differenze finite per l'equazione scalare iperbolica	416
9.9.3	Discretizzazione in spazio dell'equazione scalare iperbolica con elementi finiti	422
9.10	L'equazione delle onde	423
9.10.1	Discretizzazione dell'equazione delle onde	425
9.11	Cosa non vi abbiamo detto	430
9.12	Esercizi	430
10	Soluzione degli esercizi proposti	435
10.1	Capitolo 1	435
10.2	Capitolo 2	439
10.3	Capitolo 3	451
10.4	Capitolo 4	457
10.5	Capitolo 5	464
10.6	Capitolo 6	474
10.7	Capitolo 7	477
10.8	Capitolo 8	485
10.9	Capitolo 9	495
	Riferimenti bibliografici	507
	Indice analitico	515

Indice dei programmi MATLAB ed Octave

Tutti i programmi presenti in questo volume sono eseguibili in entrambi gli ambienti MATLAB ed Octave e possono essere scaricati dalla pagina

<http://mox.polimi.it/qs>

2.1	bisection : il metodo di bisezione	52
2.2	newton : il metodo di Newton	57
2.3	newtonsys : il metodo di Newton per un sistema non lineare	60
2.4	broyden : il metodo di Broyden	62
2.5	aitken : il metodo di Aitken	72
2.6	horner : il metodo di divisione sintetica.....	76
2.7	newtonhorner : il metodo di Newton-Hörner	78
3.1	barycentric : interpolazione baricentrica	100
3.2	cubicspline : spline cubica interpolante.....	110
4.1	midpointc : formula composita del punto medio	132
4.2	simpsonc : formula composita di Simpson	135
4.3	simpadpt : formula di Simpson adattiva	142
5.1	lugauss : la fattorizzazione LU di Gauss	164
5.2	itermeth : metodo iterativo generico	187
5.3	gradient : il metodo del gradiente (precondizionato) ..	200
5.4	cg : il metodo del gradiente coniugato (precondizionato) .	202
6.1	eigpower : il metodo delle potenze	227
6.2	invshift : il metodo delle potenze inverse con shift	232
6.3	gershcircles : i cerchi di Gershgorin.....	234
6.4	qrbasic : il metodo delle iterazioni QR	238
7.1	golden : il metodo della sezione aurea	251
7.2	backtrack : la strategia di backtracking	264

7.3	descent : il metodo di discesa	265
7.4	trustregion : il metodo trust region	277
7.5	gaussnewton : il metodo di Gauss-Newton	282
7.6	penalty : il metodo di penalizzazione	293
7.7	auglagrange : il metodo della Lagrangiana aumentata...	297
8.1	feuler : il metodo di Eulero in avanti	309
8.2	beuler : il metodo di Eulero all'indietro	309
8.3	cranknic : il metodo di Crank-Nicolson	310
8.4	predcor : un generico metodo predictor-corrector	349
8.5	feonestep : un passo del metodo di Eulero in avanti ..	350
8.6	beonestep : un passo del metodo di Eulero all'indietro ..	350
8.7	cnonestep : un passo del metodo di Crank-Nicolson ..	350
8.8	newmark : il metodo di Newmark	355
8.9	fvinc : termine forzante per il problema del pendolo sferico	363
8.10	threebody : termine forzante per il problema semplificato dei tre corpi	367
9.1	bvp_fd_dir_1d : approssimazione di un problema ai limiti di diffusione, trasporto e reazione con il metodo delle differenze finite	380
9.2	bvp_fe_dir_1d : approssimazione di un problema ai limiti di diffusione, trasporto e reazione con il metodo degli elementi finiti di grado 1	392
9.3	poisson_fd_2d : approssimazione del problema di Poisson con condizioni di Dirichlet usando il metodo delle differenze finite a 5 punti	397
9.4	heat_fd_1d : θ -metodo e differenze finite per l'equazione del calore monodimensionale	404
9.5	heat_fe_1d : θ -metodo ed elementi finiti di grado 1 per l'equazione del calore monodimensionale	410
9.6	newmarkwave : metodo di Newmark per l'equazione delle onde	426
10.1	fixedpoint : il metodo di punto fisso	450
10.2	glcomp1 : formula composita di quadratura di Gauss-Legendre con $n = 1$	459
10.3	itermethb : metodo iterativo per sistemi lineari	472
10.4	rk2 : metodo Runge-Kutta esplicito di ordine 2 (o di Heun) ..	489
10.5	rk3 : metodo Runge-Kutta esplicito di ordine 3	489
10.6	bvp_fd_neu_1d : approssimazione di un problema ai limiti di Neumann con differenze finite	498
10.7	bvp_fd_upwind_1d : approssimazione del problema di diffusione-trasporto con differenze finite upwind	499
10.8	hyper : gli schemi Lax-Friedrichs, Lax-Wendroff e upwind ..	505

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. In particolare, considereremo nozioni proprie di Analisi, Algebra e Geometria, rivisitate in funzione del loro utilizzo nel Calcolo Scientifico. Non solo, introdurremo anche nuovi concetti tipici del Calcolo Scientifico ed inizieremo ad esplorarne il significato e l'utilità avvalendoci di:

- MATLAB, un ambiente integrato per la programmazione e la visualizzazione,
- GNU Octave (Octave in breve), un interprete per linguaggio di alto livello largamente compatibile con MATLAB che è distribuito gratuitamente secondo le condizioni d'uso della GNU General Public License e che riproduce una larga parte dei comandi MATLAB.

Introdurremo in modo preliminare gli ambienti MATLAB ed Octave nella Sezione 1.1, mentre presenteremo gli elementi di base relativi alla sintassi e alla programmazione nella Sezione 1.8. Rimandiamo comunque il lettore interessato al manuale [Att16] per una descrizione di MATLAB e ad [EBHW15] per una descrizione di Octave.

1.1 Gli ambienti MATLAB e Octave

MATLAB ed Octave sono ambienti integrati per il Calcolo Scientifico e la visualizzazione grafica, scritti soprattutto in linguaggio C e C++.

MATLAB è distribuito da The MathWorks (si veda www.mathworks.com) e sta per *MATrix LABoratory*, in quanto fu originariamente sviluppato per consentire un accesso immediato a pacchetti di software appositamente sviluppati per il calcolo matriciale.

Octave, anche noto come GNU Octave (si veda www.octave.org), è distribuito gratuitamente. È possibile redistribuirlo e/o modificarlo a patto di aderire ai termini della GNU General Public License (GPL), come espresso dalla Free Software Foundation.

Gli ambienti MATLAB ed Octave presentano delle differenze, sia nel linguaggio sia nei pacchetti aggiuntivi, noti come *toolbox*¹ in MATLAB e *packages* in Octave. Tuttavia essi sono sufficientemente compatibili da averci permesso di scrivere la maggior parte dei programmi di questo libro in modo che siano eseguibili senza modifiche passando da un ambiente all'altro.

MAT&OCT

In tutto il libro indicheremo con il nome di fantasia **MAT&OCT** comandi, funzioni, programmi e, più in generale blocchi di istruzioni, comuni in entrambi gli ambienti MATLAB ed Octave.

MAT||OCT

Quando ciò non è possibile – perché alcuni comandi hanno una sintassi differente, perché essi operano in modo diverso o semplicemente perché non sono implementati in Octave o in MATLAB – abbiamo aggiunto una nota (identificata dalla sigla **MAT||OCT**) per spiegare come sostituire un comando con istruzioni equivalenti o per descrivere le specificità dei due diversi ambienti.

Così come MATLAB ha i suoi *toolbox*, Octave ha un ricco insieme di programmi (detti *packages*) sviluppati all'interno di un progetto chiamato Octave-forge (si veda il sito web octave.sourceforge.io). Questo *repository* si arricchisce costantemente di nuovi programmi per la risoluzione di problemi di diversa natura. Alcune *function* che richiameremo in questo libro non appartengono al pacchetto originale di Octave, ma possono essere scaricate dalla pagina octave.sourceforge.io/packages.php.

Dopo aver installato MATLAB sul proprio computer, ogni qualvolta si lancia l'eseguibile si apre sul desktop un ambiente di lavoro come quello riportato in Figura 1.1.

Procedendo in modo analogo con Octave, si apre un ambiente di lavoro come quello riportato in Figura 1.2.

In entrambi i casi, il desktop di lavoro è suddiviso in sottofinestre in cui è possibile eseguire comandi e vedere l'output, gestire i file, visualizzare il contenuto delle variabili e della memoria di lavoro. La finestra principale è la *command window*: caratterizzata dalla presenza del **>>** *prompt >>*, essa è la finestra in cui si digitano i comandi e le istruzioni.

A partire dal Capitolo 2 ometteremo l'uso del *prompt >>* al fine di alleggerire le notazioni.

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

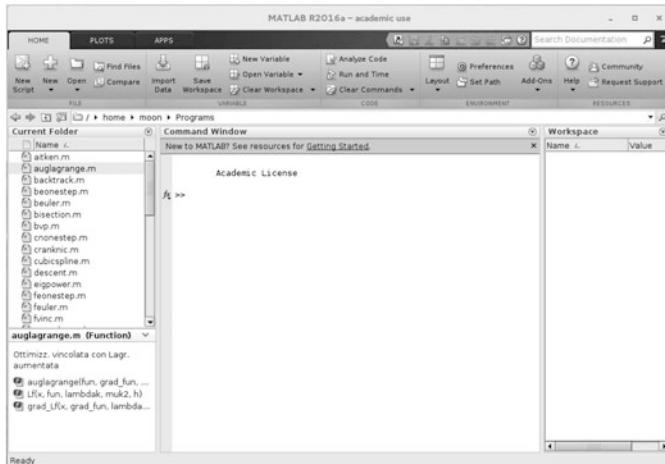


Figura 1.1. L'ambiente di lavoro di MATLAB

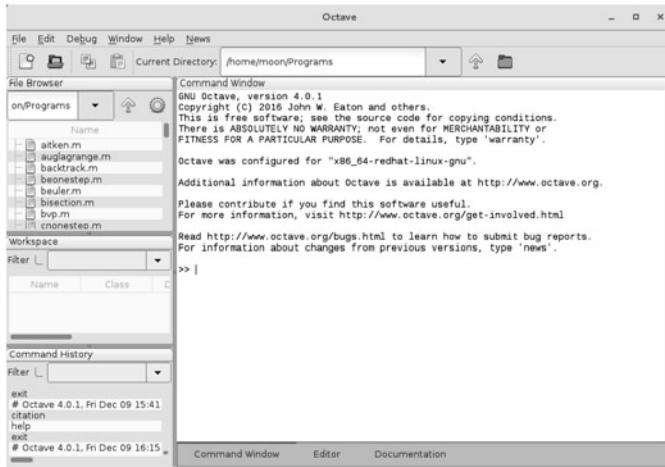


Figura 1.2. L'ambiente di lavoro di Octave

1.2 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*. Peraltrò \mathbb{F} è caratterizzato da proprietà diverse rispetto a quelle dell'insieme \mathbb{R} , come vedremo nella Sezione 1.2.2. Ciò è dovuto al fatto che ogni singolo numero reale x viene rappresentato dalla macchina con un numero ar-

rotondato, che si indica con $f(x)$ e viene detto *numero macchina*, e che non coincide necessariamente con il numero x di partenza.

1.2.1 Come si rappresentano

Per renderci conto delle differenze fra \mathbb{R} e \mathbb{F} diamo uno sguardo, tramite alcuni semplici esperimenti, al modo con il quale un calcolatore (un PC ad esempio) tratta i numeri reali. Utilizzare MATLAB o Octave 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* 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. Consideriamo il numero $1/7$. In **MAT&OCT** sono disponibili diversi formati di *output*, che producono i seguenti risultati

	MATLAB	Octave
<code>format short</code>	0.1429	0.14286
<code>format short e</code>	$1.4286\text{e-}01$	$1.4286\text{e-}01$
<code>format short g</code>	0.14286	0.14286
<code>format long</code>	0.142857142857143	0.142857142857143
<code>format long e</code>	$1.428571428571428\text{e-}01$	$1.42857142857143\text{e-}01$
<code>format long g</code>	0.142857142857143	0.142857142857143

Le leggere discrepanze in alcuni valori dell'output sono riconducibili ai diversi processi di calcolo usati dai due ambienti MATLAB e Octave e, talvolta, al diverso formato di rappresentazione dei risultati.

Talune di queste rappresentazioni sono più fedeli di altre al formato interno dell'elaboratore. Quest'ultimo memorizza generalmente i numeri nel modo seguente

$$x = (-1)^s \cdot (0.a_1 a_2 \dots a_t) \cdot \beta^e = (-1)^s \cdot m \cdot \beta^{e-t}, \quad a_1 \neq 0 \quad (1.1)$$

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` è 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* normalizzati essendo variabile la posizione del punto decimale. Le cifre $a_1 a_2 \dots a_p$ (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 l'insieme dei numeri *floating point* nel formato (1.1) ed è 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 `MAT&OCT` 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 `MAT&OCT` con il `format long`).

I numeri *floating point* di $\mathbb{F}(2, 53, -1021, 1024)$ sono memorizzati in registri di 8 Byte, più precisamente il segno s occupa 1 bit, l'esponente e 11 bit, e la mantissa m 52 bit. Quando la base di rappresentazione è pari a 2, a fronte di 52 bit occupati da m , il numero di cifre significative della mantissa è $t = 53$. Infatti in questo caso, poiché la prima cifra a_1 di ogni numero *floating point* deve essere diversa da 0, essa è necessariamente uguale ad 1 ed è superfluo memorizzarla. Nei 52 bit associati a m vengono quindi memorizzate le cifre a_2, \dots, a_{53} .

Fortunatamente l'inevitabile *errore di arrotondamento* 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 $\epsilon_M = \beta^{1-t}$, detta *epsilon macchina*, rappresenta la distanza fra 1 ed il più piccolo numero *floating-point* maggiore di 1. Si osservi che ϵ_M dipende da β e da t . Ad esempio, in **MAT_EOCT**, dove ϵ_M è calcolabile con **eps** 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 $u = \frac{1}{2}\epsilon_M$ rappresenta dunque il massimo errore relativo che la macchina può commettere nella rappresentazione di un numero reale. Per questa ragione viene talvolta chiamato *unità di arrotondamento*.

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

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

realmin Usando i comandi **realmin** e **realmax** in **MAT_EOCT** è possibile determinare tali valori:

$$\begin{aligned} x_{\min} &= 2.225073858507201 \cdot 10^{-308}, \\ x_{\max} &= 1.797693134862316 \cdot 10^{+308}. \end{aligned}$$

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 [QSSG14], Capitolo 2). Un numero positivo maggiore di x_{\max} produce invece una segnalazione di *overflow* e viene memorizzato nella

Inf 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 a 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

$$\begin{aligned} x_{\max}^- &= 1.797693134862315 \cdot 10^{+308} \\ x_{\min}^+ &= 2.225073858507202 \cdot 10^{-308}, \end{aligned}$$

dunque $x_{\min}^+ - x_{\min}^- \simeq 10^{-323}$, mentre $x_{\max} - x_{\max}^- \simeq 10^{293}$ (!). La distanza relativa resta comunque piccola, come si deduce dalla (1.2).

1.2.2 Come si opera con i numeri floating-point

Veniamo alle operazioni elementari fra numeri di \mathbb{F} : essendo \mathbb{F} soltanto un sottoinsieme finito e discreto 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 $fl(x + y) = fl(y + x)$) o fra fattori ($fl(xy) = fl(yx)$), ma vengono violate l'unicità dello zero, la proprietà associativa e quella distributiva.

Per renderci conto della non unicità dello zero, assegnamo ad una variabile **a** un valore qualunque, ad esempio 1, ed eseguiamo le seguenti istruzioni:

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

Osserviamo che la variabile **b** viene dimezzata ad ogni passo finché la somma di **a** e di **b** si mantiene diversa (non uguale, \sim) da **a**. Evidentemente, se stessimo utilizzando i numeri reali, il programma non si arresterebbe mai; invece nel nostro caso il ciclo di istruzioni 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 **MAT&OCT**, per un numero positivo **a** qualsiasi, il numero **a+eps(a)** è il più piccolo numero di \mathbb{F} successivo ad **a**. Di conseguenza, comandando 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:

```
a + ( b + c ) = 1.0990e+308,      (a + b) + c = Inf
```

Più in generale, quando si sommano tra loro numeri che hanno all'incirca lo stesso modulo, ma segno opposto, 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 la seguente operazione $((1+x)-1)/x$ con $x \neq 0$, il cui risultato esatto è ovviamente 1 per ogni $x \neq 0$. Scegliendo $x = 10^{-15}$, in **MAT&OCT** troviamo invece:

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

```
ans =
1.1102
```

Questo risultato è decisamente poco accurato, essendo l'errore relativo maggiore dell'11%.

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 1.3 (l'andamento reale è quello di $(x - 1)^7$ cioè una funzione sostanzialmente nulla e costante in

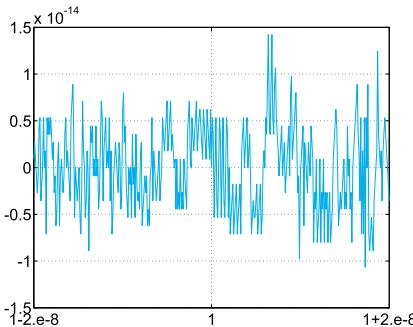


Figura 1.3. Andamento oscillante della funzione (1.3) causato dagli errori di cancellazione di cifre significative

questo minuscolo intorno di $x = 1$). Vedremo nella Sezione 1.6 i comandi `MAT&OCT` 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 `MAT&OCT`) per i quali non possono più valere le usuali regole di calcolo.

Osservazione 1.1 (Aritmetica esatta e aritmetica floating-point)

L’aritmetica del calcolatore è talvolta chiamata *aritmetica floating-point*, in contrapposizione alla cosiddetta *aritmetica esatta* che si basa sulla effettuazione esatta delle operazioni elementari (pertanto senza tener conto degli errori di arrotondamento) su operandi noti esattamente (e non attraverso la loro rappresentazione *floating-point*). ■

Osservazione 1.2 È vero che gli errori di arrotondamento sono generalmente piccoli, tuttavia, se ripetuti all’interno di algoritmi lunghi e complessi, possono avere effetti catastrofici. Due casi eclatanti riguardano l’esplosione del missile Ariane 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.

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

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

la quale converge a π quando n tende all’infinito. (Questa successione è una riscrittura della più nota *formula di François Viète* (matematico francese del XVI secolo) per l’approssimazione di π [Bec71].) Se calcoliamo numericamente z_n , troviamo che l’errore relativo fra π e z_n decresce fino a $n = 16$ per poi cominciare a crescere a causa degli errori di arrotondamento (come mostrato in Figura 1.4). ■



Si vedano gli Esercizi 1.1–1.2.

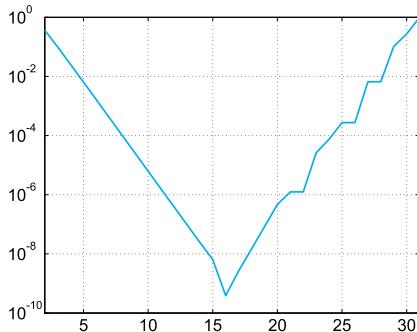


Figura 1.4. Errore relativo $|\pi - z_n|/\pi$ al variare di n

1.3 I numeri complessi

I numeri complessi, il cui insieme viene indicato con il simbolo \mathbb{C} , hanno la forma $z = x + iy$, dove i è l'unità immaginaria (cioè $i^2 = -1$), mentre $x = \text{Re}(z)$ e $y = \text{Im}(z)$ 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 i e j di MATLAB 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

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

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

```
>> abs(z)*(cos(angle(z))+i*sin(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 1.5.

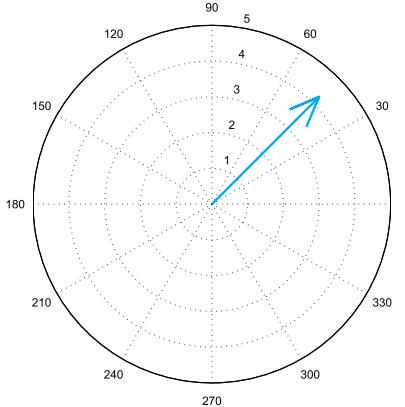


Figura 1.5. Output del comando `compass` di **MAT&OCT**

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 **MAT&OCT** tutte le operazioni vengono eseguite supponendo implicitamente che il risultato e gli operandi possano essere complessi. Così non deve stupire se calcolando la radice cubica di -5 come $(-5)^{(1/3)}$, anziché il numero reale $-1.7100\dots$ 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 le radici terze complesse di z troviamo $\sqrt[3]{\rho}e^{i(\theta/3+2k\pi/3)}$, vale a dire le tre radici distinte

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

MAT&OCT 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

$$z_1 = \sqrt[3]{5}(\cos(\pi/3) + i \sin(\pi/3)) \simeq 0.8550 + 1.4809i,$$

$$z_2 = \sqrt[3]{5}(\cos(\pi) + i \sin(\pi)) \simeq -1.7100,$$

$$z_3 = \sqrt[3]{5}(\cos(-\pi/3) + i \sin(-\pi/3)) \simeq 0.8550 - 1.4809i.$$

La prima è la radice prescelta (si veda la Figura 1.6 per la rappresentazione di z_1 , z_2 e z_3 nel piano di Gauss).

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

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

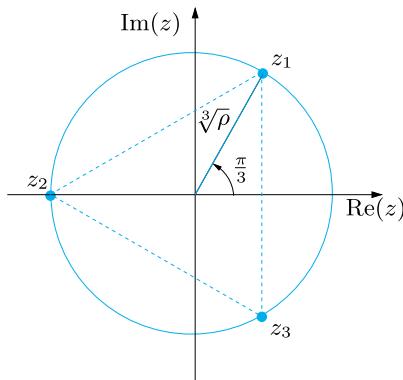


Figura 1.6. Rappresentazione nel piano di Gauss delle radici cubiche complesse di -5

1.4 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 digitare 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 `eye(m,n)` di MATLAB genera invece una matrice rettangolare i cui elementi sono tutti nulli ad eccezione di quelli

`eye`

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.

[]

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:

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

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, MATLAB risponde:

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

ed Octave

```
>> A+C
error: operator +: nonconformant arguments
(op1 is 2x3, op2 is 3x2)
>> A*B
error: operator *: nonconformant arguments
(op1 is 2x3, op2 is 2x3)
```

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 $A \in \mathbb{R}^{2 \times 2}$ 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 $\det(A) = \det(B)\det(C)$.

Data una matrice quadrata A di dimensione n , diciamo che essa è *invertibile* o *non singolare* se esiste unica la matrice $X = A^{-1}$ tale che $AX = XA = I$. A^{-1} è detta *matrice inversa* di A. Ricordiamo che la matrice inversa A^{-1} esiste se e solo se il *determinante* di A è non nullo, cioè se i vettori colonna di A sono linearmente indipendenti. Nel caso in cui $\det(A) = 0$ diciamo che A è *singolare*.

Il calcolo dell'inversa può essere realizzato attraverso il comando `inv(A)`, mentre per il calcolo del determinante si può usare il comando `inv`
`det` `det(A)`.

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 **MAT&OCT** 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 (ovvero la diagonale i cui elementi hanno indici $i, i + m$) con elementi uguali a quelli contenuti nel vettore **v**. Questo comando può essere richiamato anche con **m** negativo; in tal caso saranno interessate le sottodiagonali della matrice.

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.

tril 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. Le varianti **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 =
 3     0     0
 -1    3     0
 -2   -1     3
```

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

```
L2 =
 3     1     0
 -1    3     4
 -2   -1     3
```

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 . Quando $n = m$, se $A = A^T$, allora A è detta *simmetrica*. In **MAT&OCT**, se A è una matrice reale, A' denota la sua trasposta. Se invece A è una matrice complessa A' è la sua trasposta coniugata (ovvero A^H). Una matrice quadrata complessa A che coincide con la sua trasposta coniugata A^H è detta *hermitiana*.

1.4.1 I vettori

I vettori sono 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 $\mathbf{v} \in \mathbb{R}^n$.

I vettori in **MAT&OCT** sono trattati 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 virgole. Così ad esempio, le istruzioni $\mathbf{v} = [1; 2; 3]$ e $\mathbf{w} = [1 2 3]$ 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 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*. Dati due vettori $\mathbf{v}, \mathbf{w} \in \mathbb{R}^n$, il primo è definito come

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

essendo (v_k) e (w_k) le componenti di \mathbf{v} e \mathbf{w} , rispettivamente. Il comando **dot** corrispondente è $\mathbf{w}' * \mathbf{v}$ o **dot(v,w)**, dove l'apice esegue l'operazione di trasposizione del vettore \mathbf{w} . Per un vettore \mathbf{v} con componenti complesse, \mathbf{v}' denota il suo traspusto coniugato \mathbf{v}^H ovvero un vettore riga le cui componenti sono i complessi coniugati \bar{v}_k di v_k . Il modulo di un vettore \mathbf{v} è allora dato da

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

norm viene calcolato con il comando **norm(v)**. $\|\mathbf{v}\|$ è anche detta *norma euclidea* del vettore \mathbf{v} .

Il prodotto vettore fra due vettori $\mathbf{v}, \mathbf{w} \in \mathbb{R}^3$, è invece dato dal vettore $\mathbf{u} \in \mathbb{R}^3$ (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 il più piccolo dei due angoli individuati dalle direzioni di \mathbf{v} e \mathbf{w} . Il comando corrispondente **cross** è **cross(v,w)**. La visualizzazione di vettori in **MAT&OCT** 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 **MAT&OCT** che proporremo compariranno delle operazioni fra vettori precedute da un punto, come ad esempio **x.*y**, **.* . / . ^** **x./y** o **x.^2**. In questo modo si segnala all'elaboratore che l'operazione non va eseguita nel senso usuale, bensì componente per componente. Così **x.*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:

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

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

```
>> y'*x
ans =
32
>> x.*y
ans =
    4
   10
   18
```

Si noti che il prodotto **y*x** non è neppure definito, non avendo i vettori le dimensioni corrette.

1.4.2 Autovalori e autovettori di una matrice

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

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

Il numero complesso λ viene detto *autovalore* di A .

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 $\mathbf{y} = P^{-1}\mathbf{x}$.

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 della diagonale principale. L'approssimazione di autovalori e autovettori verrà affrontata nel Capitolo 6.

Si vedano gli Esercizi 1.3–1.6.



1.5 Le strutture e i cell-array

Le strutture e i cell-array in MATLAB possono contenere variabili di tipo diverso. Ad esempio, se vogliamo memorizzare una matrice quadrata, il suo determinante ed il numero di elementi non nulli in una sola variabile A , possiamo utilizzare una *struttura* con tre campi: il campo `mat` che conterrà la matrice, il campo `d` che conterrà il determinante, e il campo `nz` che conterrà il numero di elementi non nulli. I campi sono separati dal nome della variabile con un punto.

Possiamo inizializzare la struttura A , come segue:

```
>> A.mat=[1 0 3; 2 -1 0; 0 1 2];
>> A.d=det(A.mat);
>> A.nz=nnz(A.mat);
```

Il comando `nnz(A)` restituisce il numero di elementi non nulli della `nnz` matrice A . In MATLAB la semplice richiesta di visualizzare il contenuto di A produce l'output:

```
>> A
A =
    mat: [3x3 double]
    d: 4
    nz: 6
```

mentre il contenuto del campo `mat` viene visualizzato con il comando

```
>> A.mat
ans =
    1     0     3
    2    -1     0
    0     1     2
```

In Octave invece con il comando

```
>> A
```

viene visualizzato direttamente anche il contenuto del campo `.mat`.

Una struttura può essere definita in MATLAB anche con il comando `struct struct('field1',value1,...,'fieldn',valuuen)` come segue:

```
>> A=struct('mat',[1 0 3;2 -1 0;0 1 2],'d',4,'nz',6);
```

Un campo di una struttura potrebbe essere a sua volta un'altra struttura e possono essere aggiunti campi in ogni momento. Un campo può essere cancellato con il comando `struct=rmfield(struct,'field')`. Ad esempio, il campo `nz` viene cancellato dalla struttura `A` con l'istruzione

```
>> A=rmfield(A,'nz');
```

Le strutture sono utilizzate da varie *function* MATLAB per definire i parametri utilizzati da un algoritmo, si vedano ad esempio i comandi `optimset` e `fsolve`.

I *cell-array* sono array che contengono dati di tipo e dimensione differente. Ad esempio, se vogliamo memorizzare tre vettori di lunghezza diversa $\mathbf{v}_1 \in \mathbb{R}^4$, $\mathbf{v}_2 \in \mathbb{R}^2$ e $\mathbf{v}_3 \in \mathbb{R}^3$ nella stessa variabile `v`, possiamo inizializzare la variabile di tipo *cell-array* con il comando `cell(n,m)` (nel nostro caso `n=3` e `m=1`)

```
>> v=cell(3,1)
```

quindi memorizziamo i tre vettori come segue:

```
>> v{1}=[1; 3; 2; -1];
>> v{2}=[-2; 3];
>> v{3}=[0; 1; 5];
```

Sottolineiamo il fatto che le parentesi graffe sono utilizzate in MATLAB solo per referenziare le componenti di un *cell-array*, ma non le componenti di un array classico (in tal caso bisogna utilizzare le parentesi tonde). Quindi per leggere il contenuto di un *cell-array* daremo le istruzioni:

```
>> v
v =
[4x1 double]
[2x1 double]
[3x1 double]
```

per vedere tutta la variabile, o

```
>> v{2}
ans =
-2
3
```

per visualizzare una componente particolare della variabile.

1.6 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.

Prendiamo ad esempio la funzione $f(x) = 1/(1+x^2)$ e vediamo quali sono le istruzioni per definirla, valutarla in un punto o in un insieme di punti e rappresentarla graficamente.

Il modo più semplice per definire una funzione matematica in MATLAB è mediante una *anonymous function* con l'ausilio di un *function handle* `@` (ovvero di un puntatore alla funzione) come segue:

```
>> fun=@(x) 1/(1+x^2)
```

e per valutarla ad esempio in $x = 3$ il comando è:

```
>> y=fun(3)
y =
0.1000
```

Una *anonymous function* è una funzione che non viene memorizzata in un file, ma è definita facendo uso di una variabile di tipo *function handle* a cui la *anonymous function* è associata. Una *anonymous function* può contenere solo una istruzione eseguibile, può accettare variabili in input (di fatto le variabili indipendenti su cui essa agisce) e può contenere anche dei parametri. Le variabili di tipo *function handle* possono essere passate come variabili all'interno di altre *function*.

La sintassi generale per definire una *anonymous function* con un *function handle* `fun` è:

```
fun=@(arg1, arg2, ..., argn)expr
```

dove `arg1, arg2, ..., argn` sono le variabili indipendenti, mentre `expr` è l'espressione della funzione che vogliamo definire e, volendo, può essere racchiusa tra parentesi tonde o quadre.

L'espressione `expr` può contenere dei parametri che non rientrano nell'elenco delle variabili, ma che devono essere in ogni caso assegnati

prima di definire la funzione. Ad esempio, per definire $f(x) = a/(1+x^2)$ con $a = 3$ e valutarla in $x = 2$, scriveremo:

```
>> a=3; fun=@(x) a/(1+x^2); y=fun(2)
```

e il risultato sarà

```
y =
0.6000
```

Se modifichiamo il valore del parametro a , dobbiamo ridefinire il *function handle* `fun`. Ad esempio:

```
>> a=8; fun=@(x) a/(1+x^2); y=fun(2)
y =
1.6000
```

Per visualizzare il grafico di $f(x)$ in un intervallo possiamo utilizzare **fplot** il comando `fplot(fun,lims)` dove `fun` è un *function handle* e `lims` è un array di due elementi i cui valori sono gli estremi dell'intervallo in questione. Volendo rappresentare $f(x) = 1/(1+x^2)$ su $[-5, 5]$, basterà scrivere:

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

In alternativa, si può scrivere direttamente:

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

invocando l'*anonymous function* senza l'uso del *function handle* `fun`.

I grafici prodotti da **MAT&OCT** sono una rappresentazione approssimata, a meno dello 0.2%, del grafico di f e sono ottenuti campionando la funzione su un opportuno insieme non equispaziato di ascisse. Per aumentare l'accuratezza 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 (`[]`). Per introdurre una griglia di riferimento come quella che compare nella

grid Figura 1.3 basta dare, dopo il comando `fplot`, il comando `grid on`.

Una funzione matematica può essere definita, oltre che con una *anonymous function*, anche mediante la creazione di una *function*, detta anche *user-defined function*. Ad esempio, possiamo scrivere le seguenti istruzioni

```
function y=myfun(x)
y=1/(1+x^2);
end
```

e memorizzarle nel file `myfun.m`. Suggeriamo, anche se non è obbligatorio, di assegnare al file `.m` il nome della *function* in esso contenuta. Infatti, quando invochiamo la function `myfun`, MATLAB controlla se esiste in memoria una *anonymous function* di nome `myfun` e, qualora non la trovi, cerca nel proprio *path* solo il file `myfun.m`.

Se vogliamo rappresentare graficamente sull'intervallo $[-\pi, \pi]$ la funzione memorizzata nel file `myfun.m`, la sintassi corretta da utilizzare è:

```
>> fplot(@myfun, [-pi, pi])
```

oppure

```
>> fplot('myfun', [-pi, pi])
```

(Il carattere speciale `@` genera il *function handle* associato alla *user-defined function* `myfun`.)

Se la variabile `x` è un array, le operazioni `/`, `*` e `^` utilizzate per definire la funzione devono essere sostituite dalle corrispondenti *operazioni punto* `./`, `.*` e `.^` che lavorano elemento per elemento.

Ad esempio, l'istruzione `fun=@(x) 1/(1+x^2)` deve essere sostituita da `fun=@(x) 1./(1+x.^2)`.

Il comando `plot` può essere usato in alternativa a `fplot`, a patto che la funzione matematica sia stata prima valutata su un insieme di ascisse. Le seguenti istruzioni:

```
>> x=linspace(-2,3,100);
>> y=exp(x).*(sin(x).^2)-0.4;
>> plot(x,y,'c','LineWidth',2); grid on
```

producono il grafico di $f(x) = e^x \sin^2(x) - 0.4$, più precisamente il comando `linspace(a,b,n)` genera un vettore riga di n punti equispaziati da a a b , mentre il comando `plot(x,y,'c','LineWidth',2)` crea una spezzata che congiunge i punti (x_i, y_i) (per $i = 1, \dots, n$) di colore cyan e di due punti di spessore.

Osservazione 1.3 Con i *function handle* si possono definire anche funzioni vettoriali. In questo caso si segue la sintassi comune per definire vettori e matrici, ovvero spazi o virgole per separare gli elementi di una riga, e punto e virgola per separare le righe. Per esempio, per definire la funzione vettoriale $\mathbf{g} : \mathbb{R}^2 \rightarrow \mathbb{R}^2$, con $\mathbf{g}(x, y) = [e^x \sin(y), x^2 - y]^t$ possiamo scrivere

```
>> g=@(x,y) [exp(x).*sin(y); x.^2-y]
```

Per prevenire definizioni errate di funzioni, conviene evitare spazi bianchi quando non necessari. Per esempio, scrivendo $f(x) = 2x - \sin(x)$ e introducendo uno spazio bianco tra `2*x` e `-sin(x)`

```
>> f=@(x) [2*x -sin(x)],
```

l'istruzione

```
>> y=f(pi/2)
```

produce il vettore

```
y=
 3.1416 -1.0000
```

invece del valore scalare $y=3.1416$. In effetti, lo spazio tra $2*x$ e $-\sin(x)$ è interpretato da **MAT&OCT** come elemento separatore; ciò comporta che l'istruzione data sopra in realtà definisca la funzione vettoriale $f : \mathbb{R} \rightarrow \mathbb{R}^2$ $f(x) = [2x, \sin(x)]$ invece di quella scalare $f(x) = 2x - \sin(x)$. ■

1.6.1 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 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 reali.

Il calcolo diretto di tutti gli 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 abbia la forma

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

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

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.

Nel seguito indicheremo con \mathbb{P}_n lo spazio dei polinomi di grado minore o uguale a n ,

$$p_n(x) = \sum_{k=0}^n a_k x^k \tag{1.9}$$

dove gli a_k sono coefficienti assegnati, reali o complessi.

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 eseguito in **MAT&OCT** tramite il comando **fzero(fun,x0)**. 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 tra `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) x^2-1+exp(x);
>> fzero(fun,-1)
ans =
-0.7146

>> fzero(fun,1)
ans =
5.4422e-18
```

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,[-1 -0.2])
ans =
-0.7146

>> fzero(fun,[-0.2 1])
ans =
-5.2609e-17
```

Come si vede il risultato ottenuto per la seconda 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'algoritmo implementato in `fzero`. Vedremo nel capitolo 2 alcuni metodi per il calcolo approssimato degli zeri di una funzione generica.

Se la *function* `fun` è definita esternamente attraverso un M-file, possiamo utilizzare a scelta una delle seguenti forme di chiamata a `fzero`:

```
>> fzero(@fun,1)

o

>> fzero('fun',1)
```

Osservazione 1.4 Octave consiglia l'uso della *function* `fsolve` invece di `fzero` qualora non si assegni un intervallo in cui cercare la radice, ma si cerchi una radice prossima a `x0`. ■

1.6.2 I polinomi

Come abbiamo già avuto modo di dire i polinomi sono funzioni abbastanza particolari e per essi sono stati approntati comandi `MATGOCT` specifici. 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];
>> y = polyval(p,x)
y =
    Columns 1 through 5
    1.0000    0.5540   -0.2578   -0.8126   -1.0000
    Columns 6 through 9
   -0.8124   -0.2422    0.8210    3.0000
```

In alternativa, per valutare un polinomio si potrebbe usare anche una *anonymous function*; essa è però generalmente scomoda perché obbligherebbe a riportare nella stringa che definisce la funzione da disegnare l'espressione completa del polinomio e non solo i suoi coefficienti.

roots 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
```

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 =
 -1.0094
 -1.0059 + 0.0073i
 -1.0059 - 0.0073i
 -0.9979 + 0.0092i
 -0.9979 - 0.0092i
 -0.9915 + 0.0041i
 -0.9915 - 0.0041i
```

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 la Sezione 2.8.2).

Tabella 1.1. Principali comandi MATLAB relativi ai polinomi

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>	$q = \text{coefficienti di } q, r = \text{coefficienti di } r$ tali che $p_1 = qp_2 + r$
<code>y=polyder(p)</code>	$y = \text{coefficienti di } p'(x)$
<code>y=polyint(p)</code>	$y = \text{coefficienti di } \int_0^x p(t) dt$

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 $p1 = conv(p2,q) + r$.

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 $p_1(x) = q(x)p_2(x) + r(x)$.

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`, `p1`, `p2` sono i vettori contenenti i coefficienti dei polinomi p , p_1 e p_2 , rispettivamente.

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 la Sezione 3.3.1).

1.6.3 L'integrale e la derivata

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

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

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

è una *primitiva* di f , è derivabile e si ha

$$F'(x) = f(x), \quad \forall x \in [a, b];$$

2. 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

$$f'(\bar{x}) = \lim_{h \rightarrow 0} \frac{f(\bar{x} + h) - f(\bar{x})}{h}. \quad (1.10)$$

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, $C^0([a, b])$ è lo spazio delle funzioni continue in $[a, b]$.

Un risultato dell'Analisi che utilizzeremo spesso è il *teorema del valore medio* (o di Lagrange) secondo il quale, se $f \in C^0([a, b])$ ed è derivabile in (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 derivabile n volte nel 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

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

Il *toolbox symbolic* di MATLAB ed il pacchetto *symbolic* di Octave-Forge consentono, attraverso i comandi *diff*, *int* e *taylor*, *diff int taylor* di calcolare analiticamente la derivata, l'integrale indefinito (ovvero la primitiva) ed il polinomio di Taylor di semplici funzioni. Anzitutto bisogna dichiarare la variabile *simbolica* *x* che compare nella definizione della funzione *f* con il comando *syms x*. In tal modo, essa potrà essere manipolata algebricamente senza dover essere necessariamente valutata. Quindi, una volta definita l'espressione *f* della funzione sulla quale si intende operare, *diff(f)* ne calcola la derivata prima, *diff(f,n)* la derivata di ordine *n*, *int(f)* l'integrale in senso indefinito e *taylor(f,'expansionPoint',x0,'Order',n+1)* il polinomio di Taylor di grado *n* in un intorno del punto *x0*.

Ad esempio, per calcolare la derivata, l'integrale indefinito ed il polinomio di Taylor del quint'ordine della funzione $f(x) = (x^2 + 2x + 2)/(x^2 + 1)$ in un intorno del punto $x_0 = 1$, basta digitare i comandi:

```
>> syms x
>> f = (x^2+2*x+2)/(x^2+1) ;
>> diff(f)
ans =
  (2*x+2)/(x^2+1)-(2*x*(x^2+2*x+2))/(x^2+1)^2
>> int(f)
ans =
  x+log(x^2+1)+atan(x)
>> t= taylor(f,'expansionPoint',1,'Order',6)
t =
  (x-1)^3/2-(x-1)^2/4-x/2-(3*(x-1)^4)/8+(x-1)^5/8+3
>> simplify(t)
ans =
  x^5/8-x^4+(13*x^3)/4-(21*x^2)/4+(29*x)/8+7/4
```

Con il comando *simplify* è possibile semplificare le espressioni generate da *diff*, *int* e *taylor* in modo da renderle più semplici possibile.

Infine, se f è una funzione di $d \geq 2$ variabili, cioè $f = f(x_1, \dots, x_d)$, ed $\mathbf{e}_i \in \mathbb{R}^d$ è l' i -simo vettore della base canonica in \mathbb{R}^d , denotiamo con

$$\frac{\partial f}{\partial x_i}(\bar{\mathbf{x}}) = \lim_{h \rightarrow 0} \frac{f(\bar{\mathbf{x}} + h\mathbf{e}_i) - f(\bar{\mathbf{x}})}{h} \quad (1.11)$$

la *derivata parziale* di f rispetto alla variabile x_i valutata nel punto $\bar{\mathbf{x}}$.

Osservazione 1.5 Dopo aver accertato che i programmi *Python* e *Sympy* *MAT||OCT* sono presenti nel proprio sistema (essi sono programmi esterni ad Octave) e che la connessione ad internet è attiva, il pacchetto Octave-Forge *symbolic*

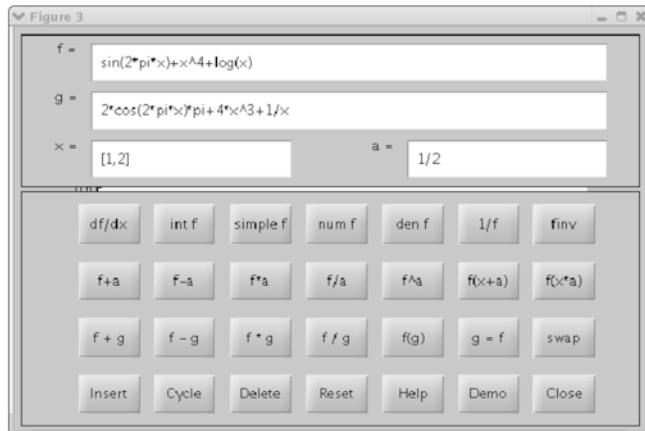


Figura 1.7. Interfaccia grafica del comando `funtool` di MATLAB

può essere installato con i seguenti comandi (da digitare all'interno di una sessione di Octave):

```
>> pkg install -forge symbolic
>> pkg load symbolic
```

L'istruzione di caricamento del pacchetto `symbolic` deve essere digitata ad ogni nuova sessione di Octave in cui si voglia eseguire del calcolo simbolico.

funtool Il comando `funtool` di MATLAB consente, attraverso l'interfaccia grafica riportata in Figura 1.7, di manipolare simbolicamente delle funzioni e di studiarne le principali caratteristiche.

Il comando `funtool` non è disponibile in Octave. ■



Si vedano gli Esercizi 1.7–1.8.

1.7 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 degli errori. L'importante perciò non è annullare gli errori, ma imparare a controllarne l'effetto.

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 1.8).

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).

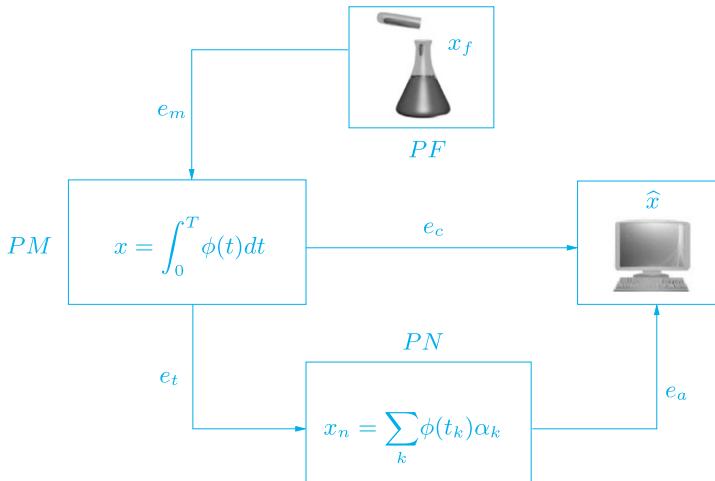


Figura 1.8. I vari tipi di errore nel processo computazionale

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 1.8, un'equazione algebrica o differenziale, un sistema lineare o non lineare) non è in generale risolvibile 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 truncamento*. 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 restituisca 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

$$e_c \leq Ch^p \quad (1.12)$$

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 (1.12), 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 (1.10). 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 nella Sezione 4.2 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. (La pendenza sarà $p = 1$ per metodi di ordine uno, $p = 2$ per metodi di ordine due, e così via.) 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 1.9 a sinistra, vengono riportate le rette relative all'andamento degli errori per due diversi metodi. Quello in linea continua risulta del prim'ordine, mentre quello in linea tratteggiata è del second'ordine. In Figura 1.9, a destra, sono mostrati gli stessi dati riportati a sinistra, ma ora rappresentati con il comando `plot`, cioè in scala lineare sia per l'asse delle ascisse che per l'asse delle ordinate. È evidente che la rappresentazione lineare di questi dati non è ottimale, poiché la curva tratteggiata risulta molto schiacciata sull'asse delle ascisse quando $x \in [10^{-6}, 10^{-2}]$, nonostante le corrispondenti ordinate vadano da 10^{-12} a 10^{-4} e sono quindi distribuite su otto ordini di grandezza.

`loglog`

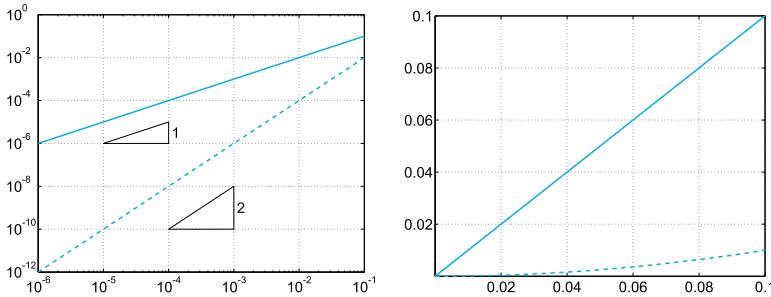


Figura 1.9. Grafici dei medesimi dati in scala log-log (*a sinistra*) e lineare-lineare (*a destra*)

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

$$p_i = \log(e_i/e_{i-1}) / \log(h_i/h_{i-1}), \quad i = 2, \dots, N. \quad (1.13)$$

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 possano essere utilizzate per stimare l'errore stesso, i cosiddetti *stimatori dell'errore*. Ne vedremo degli esempi nei paragrafi 2.3.1, 2.6 e 4.5.

Talvolta invece di utilizzare la scala log-log, faremo uso della scala semilogaritmica, cioè della scala logaritmica lungo l'asse delle ordinate e di quella lineare lungo l'asse delle ascisse. Questo tipo di rappresentazione è da preferirsi ad esempio quando dobbiamo plottare l'errore di un metodo iterativo al variare delle iterazioni, come abbiamo fatto nella Figura 1.4, o più in generale, quando le ordinate spaziano su un intervallo molto più vasto di quello delle ascisse.

A titolo di esempio consideriamo tre successioni tutte convergenti a $\sqrt{2}$:

$$\begin{aligned} x_0 &= 1, & x_{n+1} &= \frac{3}{4}x_n + \frac{1}{2x_n}, & n &= 0, 1, \dots, \\ y_0 &= 1, & y_{n+1} &= \frac{1}{2}y_n + \frac{1}{y_n}, & n &= 0, 1, \dots, \\ z_0 &= 1, & z_{n+1} &= \frac{3}{8}z_n + \frac{3}{2z_n} - \frac{1}{2z_n^3}, & n &= 0, 1, \dots. \end{aligned}$$

In Figura 1.10 sono rappresentati in scala semilogaritmica gli errori $e_{x,n} = |x_n - \sqrt{2}|/\sqrt{2}$ (linea continua), $e_{y,n} = |y_n - \sqrt{2}|/\sqrt{2}$ (linea

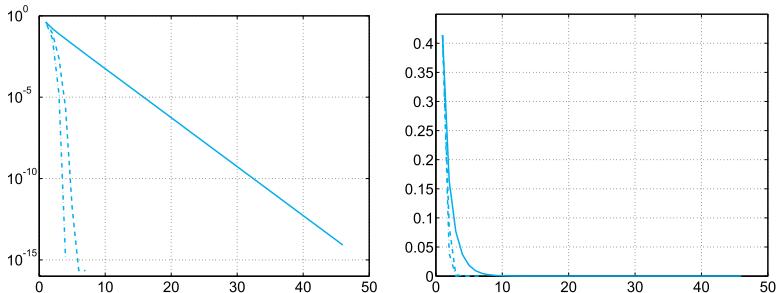


Figura 1.10. Errori $e_{x,n}$ (linea continua), $e_{y,n}$ (linea tratteggiata) e $e_{z,n}$ (linea tratto-punto) in scala semilogaritmica (a sinistra) e lineare-lineare (a destra)

tratteggiata) $e_{z,n} = |z_n - \sqrt{2}|/\sqrt{2}$ (linea tratto-punto) al variare delle iterazioni. È possibile dimostrare che

$$e_{x,n} \simeq \rho_x^n e_{x,0}, \quad e_{y,n} \simeq \rho_y^{n^2} e_{y,0}, \quad e_{z,n} \simeq \rho_z^{n^3} e_{z,0},$$

dove $\rho_x, \rho_y, \rho_z \in (0, 1)$, cosicché, applicando il logaritmo solo alle ordinate, si ha

$$\begin{aligned} \log(e_{x,n}) &\simeq C_1 + \log(\rho_x)n, & \log(e_{y,n}) &\simeq C_2 + \log(\rho_y)n^2, \\ \log(e_{z,n}) &\simeq C_3 + \log(\rho_z)n^3, \end{aligned}$$

cioè una linea retta, una parabola ed una cubica, rispettivamente, esattamente come possiamo vedere in Figura 1.10, a sinistra.

Il comando `MAT&OCT` per la rappresentazione grafica in scala `semilogy` semilogaritmica è `semilogy(x,y)`, dove `x` e `y` sono vettori della stessa lunghezza.

In Figura 1.10, a destra, sono mostrati gli errori $e_{x,n}$, $e_{y,n}$ e $e_{z,n}$ al variare delle iterazioni, in scala lineare-lineare utilizzando il comando `plot`. È evidente che la scala semilogaritmica è più appropriata di quella lineare-lineare.

1.7.1 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. Uno degli indicatori della velocità di un elaboratore è il massimo numero di operazioni *floating-point* che l'elaboratore stesso esegue in un secondo

(*flops*). In particolare sono utilizzate le seguenti sigle: *Mega-flops* pari a 10^6 *flops*, *Giga-flops* pari a 10^9 *flops*, *Tera-flops* pari a 10^{12} *flops*, *Peta-flops* pari a 10^{15} *flops*. Il calcolatore più potente che esista alla data in cui scriviamo (primo della *top500 supercomputer list* a giugno 2017) può effettuare circa 93 *Peta-flops* ed è il Sunway TaihuLight del National Supercomputing Center in Wuxi, China.

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 (Prodotto matrice-vettore) Si consideri una matrice quadrata A di dimensione n ed un vettore $\mathbf{v} \in \mathbb{R}^n$: vogliamo quantificare il costo computazionale dell’usuale algoritmo per eseguire il prodotto $A\mathbf{v}$. 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 fra 2 matrici quadrate di dimensione 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 (Determinante di una matrice quadrata) Come abbiamo ricordato, il determinante di una matrice quadrata 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 *floating-point* 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 da tenere in considerazione nell'analisi dell'algoritmo stesso. Esso tuttavia non è il solo. Quando un algoritmo viene codificato, altri fattori possono condizionarne l'efficacia, ad esempio l'accesso alle memorie. Una misura delle prestazioni di un programma (o di una sequenza di istruzioni) è il cosiddetto *tempo di CPU* (CPU sta per *central processing unit*) ovvero il tempo impiegato dall'unità centrale del calcolatore per eseguire quel determinato programma. Il tempo di CPU non tiene conto del tempo utilizzato da altri processi (siano essi di sistema o lanciati dall'utente) che sono in esecuzione in contemporanea e che sono gestiti in *multitasking* dal processore stesso, ed 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* e, su calcolatori monoprocesso, esso è maggiore o uguale al tempo di CPU. Quando però una CPU è dotata di più *core* ed il carico di lavoro del processo è suddiviso tra di essi (si hanno cioè più *thread*), il tempo totale utilizzato dalla CPU risulta essere la somma dei tempi misurati sui singoli *core* e può essere ben maggiore dell'*elapsed time*.

I comandi `MAT&OCT` per misurare (in secondi) il tempo di CPU e `cputime` l'*elapsed time* sono rispettivamente `cputime` ed `etime`.

`etime`

Esempio 1.4 Misuriamo sia il tempo di CPU sia l'*elapsed time* necessari per calcolare il prodotto fra una matrice quadrata ed un vettore su un pc dotato di un processore Intel® i7-4790, 3.60 GHz con 4 *core*. A tale scopo eseguiamo le seguenti istruzioni:

```
>> n=10000; step=100;
>> A=rand(n,n);
>> v=rand(n,1);
>> CPUtime=[ ];
>> Etime=[ ];
>> sizeA=[ ];
>> for k = 100:step:n
    AA = A(1:k,1:k);
    vv = v(1:k);
    t = cputime;
    for i=1:20
        b = AA*vv;
    end
    ttc = (cputime - t)/20;
    t = clock;
    for i=1:20
        b = AA*vv;
    end
    tte = etime(clock,t)/20;
    Etime = [Etime, tte];
    CPUtime = [CPUtime, ttc];
    sizeA = [sizeA,k];
end
```

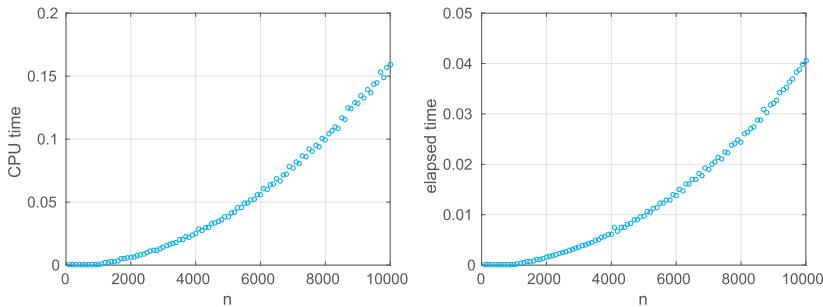


Figura 1.11. Tempo di CPU (*a sinistra*) ed *elapsed time* (*a destra*) misurati su un processore Intel® i7-4790, 3.60 GHz con 4 *core* per eseguire un prodotto matrice-vettore in funzione della dimensione n della matrice. I tempi sono misurati in secondi

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=100`, `b=10000` e `step=100`). Il comando `rand(n,m)` inizializza una matrice `n×m` i cui elementi sono numeri casuali. Nelle componenti del vettore `CPUTime` vengono memorizzati i tempi di CPU misurati per eseguire ogni prodotto matrice-vettore (in realtà abbiamo calcolato un tempo medio su 20 esecuzioni in quanto la singola esecuzione sarebbe durata meno del tempo minimo misurabile pari a un centesimo di secondo). `cputime` restituisce il tempo complessivo impiegato (da `MAT&OCT`) per eseguire tutti i processi dal momento in cui la sessione di lavoro è iniziata. Il tempo richiesto per eseguire un singolo processo è dunque la differenza tra il tempo di CPU attuale e quello calcolato prima dell'inizio delle operazioni, quest'ultimo qui memorizzato nella variabile `t`. Nelle componenti del vettore `Etime` sono stati invece memorizzati gli *elapsed time*, misurati con il comando `clock`, il quale restituisce il tempo di `clock` del computer su cui stiamo lavorando.

I grafici in Figura 1.11 (ottenuti con il comando `plot(sizeA,T,'o')` e `T=CPUTime` a sinistra, o `T=Etime` a destra) mostrano il tempo di CPU (a sinistra) e l'*elapsed time* (a destra). Entrambi crescono proporzionalmente al quadrato della dimensione `n` della matrice, ma il tempo di CPU è pari a quattro volte il corrispondente *elapsed time*. MATLAB ed Octave sono infatti in grado di distribuire automaticamente sui quattro *core* del processore il carico di lavoro necessario a svolgere l'operazione di prodotto matrice-vettore, senza bisogno di dare specifiche direttive di parallelizzazione. ■

1.8 Qualche parola in più su MATLAB e Octave

Dopo le note introduttive viste sino ad ora, siamo pronti a lavorare sia in ambiente MATLAB sia in Octave, in una sola parola in `MAT&OCT`.

Premuto il tasto *enter* (o *return*), tutto ciò che scriveremo dopo il *prompt* verrà interpretato,² ovvero **MAT&OCT** si domanderà se ciò che abbiamo scritto rientri fra le variabili definite e, qualora questo non accada, se è il nome di uno dei programmi o dei comandi presenti in **MAT&OCT**. Nel caso in cui anche questo controllo fallisca, **MAT&OCT** segnala un messaggio d'errore. In caso contrario il comando viene eseguito producendo eventualmente un *output*. In entrambi i casi il sistema ripropone il *prompt* al termine dell'esecuzione e rimane in attesa di un nuovo comando.

Per chiudere una sessione **MAT&OCT** è sufficiente scrivere il comando **quit** (**quit**) (o **exit**) e battere *enter*. D'ora in poi sottintenderemo che per **exit** 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 coincide con una delle strutture elementari definite in **MAT&OCT** (come i numeri o le stringhe di caratteri che si precisano tra apici), quest'ultima viene restituita in **ans** (**put**) nella variabile di *default ans* (che sta per *answer*, cioè *risposta*). Ad esempio, se digitiamo la stringa di caratteri '*casa*' abbiamo:

```
>> '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 il comando. Così l'esecuzione di '*casa*'; si limita a riproporre il *prompt*, assegnando comunque il valore '*casa*' alla variabile **ans**.

- = Il comando **=** serve per assegnare ad una data variabile un valore numerico o una stringa di caratteri. Ad esempio, volendo assegnare la stringa '*Benvvenuto a Milano*' alla variabile **a** basterà scrivere:

```
>> a='Benvenuto a Milano';
```

Come si vede non è necessario dichiarare il tipo di una variabile; sarà **MAT&OCT** ad allocare automaticamente e dinamicamente le variabili che di volta in volta utilizzeremo. Ad esempio, se decidessimo di voler utilizzare la variabile **a** che prima abbiamo 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 definire una variabile **quit** e di assegnarle il valore 5. Abbiamo quindi creato una variabile che ha lo stesso nome del comando **quit** di **MAT&OCT**; così facendo non possiamo più eseguire il comando **quit** in quanto **MAT&OCT** 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

² Di conseguenza, un programma **MAT&OCT** non deve essere compilato come succede, ad esempio, in Fortran o in C; tuttavia per aumentare la velocità di esecuzione dei codici, in MATLAB si può ricorrere ad un compilatore con il comando **mcc**.

di assegnare a variabili, o a programmi, nomi di variabili o programmi già definiti in **MAT&OCT**. In ogni caso, con il comando **clear** seguito dal nome della variabile, ad esempio **quit**, è possibile cancellare una variabile dal sistema e riaccedere quindi, nel nostro caso, al comando **quit**.

Utilizzando il comando **save nomefile** tutte le variabili della sessione (che sono memorizzate nel cosiddetto *base workspace*) vengono salvate in un file binario di nome **nomefile.mat** in MATLAB e **nomefile** in Octave. Analogamente, il comando **load nomefile** ripristina nella sessione corrente tutte le variabili precedentemente memorizzate nel file binario di nome **nomefile.mat** in MATLAB e **nomefile** in Octave. Se poi si volessero salvare solo alcune variabili, diciamo **v1**, **v2** e **v3**, in uno specifico file, di nome ad esempio **area.mat** in MATLAB o **area** in Octave, 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 *toolbox*) di comandi disponibili durante l'esecuzione. Tra i moltissimi ricordiamo quelli che definiscono le funzioni elementari seno (**sin(x)**), coseno (**cos(x)**), radice quadrata (**sqrt(x)**) ed esponenziale (**exp(x)**).

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 (o booleani) *and* (&), *or* (|), *not* (^), gli operatori relazionali di maggiore (>), maggiore o uguale (>=),

minore (<), minore o uguale (<=), uguale (==). Infine, nessun nome può cominciare con un numero, nè contenere una parentesi o un carattere della punteggiatura.

Lo speciale linguaggio di programmazione fornito in **MAT&OCT** 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 **MAT&OCT** presenta diverse istruzioni o costrutti (chiamati *statement*) fra i quali citiamo i *condizionali* ed i *cicli*.

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

```
if <condition 1>
  <statement 1.1>
  <statement 1.2>
  ...
elseif <condition 2>
  <statement 2.1>
  <statement 2.2>
  ...
else
```

```
<statement n.1>
<statement n.2>
...
end
```

dove `<condition 1>`, `<condition 2>`, ... rappresentano espressioni logiche che possono assumere i valori 0 o 1 (falso o vero). L'intera costruzione permette l'esecuzione degli *statement* corrispondenti alla prima condizione che assume valore uguale a 1. Se tutte le condizioni fossero false sarebbero eseguiti gli *statement* `<statement n.1>`, `<statement n.2>`, ... Infatti se `<condition k>` è falsa gli *statement* `<statement k.1>`, `<statement k.2>`, ... non vengono eseguiti ed il controllo passa oltre.

Ad esempio, per calcolare le radici del polinomio $ax^2 + bx + c = 0$ si `disp` 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');
else
    disp('L''equazione data e'' un''identita''')
end
(1.14)
```

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

Le espressioni logiche che compaiono in una condizione possono essere ottenute combinando espressioni logiche elementari mediante gli operatori booleani `&`, `|`, `&&`, e `||`. Gli ultimi due operatori `&&` e `||` implementano la forma *short-circuiting* invece di quella *element-by-element* degli operatori `&` and `|`. La forma *element-by-element* è spesso sufficiente a realizzare le operazioni logiche più comuni, tuttavia, a volte è più conveniente bloccare la valutazione dell'espressione logica non appena il suo valore di verità può essere determinato. Ciò è realizzabile solo con la forma *short-circuiting*.

Ad esempio: mentre con l'istruzione

```
>> (nit <= nmax) & (err > tol)
```

è calcolato il valore logico di entrambe le espressioni nelle parentesi e poi su di esse agisce l'operatore *and*, con l'istruzione

```
>> (nit <= nmax) && (err > tol)
```

è calcolato il valore logico della prima parentesi e, solo se questo è 1 (cioè vero), viene calcolato il valore della seconda.

In **MAT&OCT** sono disponibili due tipi di ciclo: **for** (simile al ciclo **do for** 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 $\{f_i = f_{i-1} + f_{i-2}, i \geq 3\}$ 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 che il punto e virgola può essere usato per separare istruzioni scritte sulla stessa riga. 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 intanto che 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
```

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

1.9 Programmare in MATLAB e Octave

In questa Sezione vogliamo fornire qualche indicazione per la stesura di programmi **MAT&OCT**. Un nuovo programma deve essere salvato in un file, chiamato genericamente *m-file* ed avente estensione **.m**. Questo file potrà essere richiamato durante la sessione di lavoro purché si trovi in una delle cartelle (o *directory*) nelle quali **MAT&OCT** 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 path** corrispondente per vedere come aggiungere una cartella a tale lista.

A livello di programmi, è importante distinguere tra *script* e *user-defined function*. I primi sono solo una semplice raccolta di istruzioni o comandi **MAT&OCT**, senza un'interfaccia di input/output. Ad esempio, l'insieme delle istruzioni (1.14), 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
```

```
>> x
x =
-0.5000 + 0.8660i  -0.5000 - 0.8660i

>> a = 0; b = 1; c = 1;
>> equation
>> x
x =
-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 *user-defined function*.

Una *user-defined function*, in breve *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 *user-defined function*: in esso viene definita una nuova *function*, chiamata `det23`, che calcola, secondo la formula data nella Sezione 1.4, 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
%dì 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
else
    disp(' Solo matrici quadrate ');
end
```

... 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,l])` estrae da `A` una sottomatrice 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 della sezione aurea $\alpha = 1.6180339887\dots$, 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]=fibonacci0
% FIBONACCIO: Approssimazione della sezione aurea
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
```

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:

```
>> [alpha,niter]=fibonacci0
alpha =
1.61805555555556
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 *user-defined function* può variare. Per esempio, la *function* `fibonacci0` appena vista potrebbe modificarsi come segue:

```
function [golden,k]=fibonacci1(tol,kmax)
% FIBONACC1: Approssimazione della sezione aurea
% La tolleranza ed il num. max di iterazioni
% possono essere assegnati in input
```

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.

```

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

```

nargin La *function* `nargin` conta il numero di parametri di input (in modo analogo, con `nargout` si contano i parametri di output). In questa nuova *function* `fibonacci1` possiamo prescrivere una specifica tolleranza `tol` ed il massimo numero di iterazioni consentite `kmax` oppure, non passando tali variabili in input, accettare i valori di *default* predisposti all'interno della *function* stessa (nel nostro caso `tol=1.e-04` e `kmax=100`). Un uso possibile allora è il seguente:

```

>> [alpha,niter]=fibonacci1(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. Ad esempio:

```

>> nargin('fibonacci1')
ans =
    2

```

Dopo questa breve introduzione, l'invito è di esplorare **MATLAB** utilizzandone l'*help* e di acquisire dimestichezza nella codifica degli algoritmi, mediante la lettura dei programmi descritti e proposti in questo libro.

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

MAT||OCT **Osservazione 1.6** Come abbiamo già accennato, praticamente tutto quanto scritto nei paragrafi precedenti è eseguibile sia in ambiente MATLAB sia in ambiente Octave senza modifiche. Esistono tuttavia alcune differenze legate ai linguaggi stessi, cosicché dei programmi scritti in Octave possono non essere

eseguibili in MATLAB e viceversa. Ad esempio, Octave accetta che vengano definite variabili di tipo stringa sia con apice singolo sia con apice doppio:

```
octave:1> a="home"
a = home
octave:2> a='home'
a = home
```

mentre MATLAB accetta solo apici singoli, in quanto gli apici doppi producono errori.

Di seguito riportiamo una lista con alcune delle incompatibilità fra i due linguaggi:

- MATLAB non accetta caratteri bianchi prima dell'operazione di trasposizione. Ad esempio, `[0 1]'` è corretto in MATLAB, mentre `[0 1]'` non lo è. Octave al contrario accetta entrambe le situazioni;
- MATLAB richiede sempre ...

```
rand (1, ...
      2)
```

mentre in Octave, oltre a ... funziona anche la forma

```
rand (1,
      2)
```

- per implementare la potenza, Octave consente l'uso sia di `^` che di `**`; MATLAB richiede `^`;
- per implementare l'operatore logico *not*, Octave consente l'uso sia di `~=` sia di `!=`; mentre MATLAB richiede `~=`;
- per la chiusura di cicli e blocchi di selezione, Octave consente l'uso sia di `end` che di `endif`, `endfor`, ...; MATLAB richiede sempre `end`;
- per convertire una funzione simbolica in *function handle* si utilizza il comando `matlabFunction` in MATLAB e `function_handle` in Octave.



Si vedano gli Esercizi 1.9–1.15.

1.10 Cosa non vi abbiamo detto

Una trattazione più sistematica dei numeri *floating-point* può essere trovata in [Übe97], [Hig02] e in [QSSG14].

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 [HH17], ma anche monografie quali [Att16], [HLR14], [Pra16], [Pal08] o [MH03].

Per Octave raccomandiamo di consultare la documentazione *on-line* disponibile sul sito <http://www.octave.org> ed il volume [EBHW15].

1.11 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 **MAT&OCT**.

Esercizio 1.4 Si costruiscano in **MAT&OCT** 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 **MAT&OCT** 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

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

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

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

poly Esercizio 1.8 Dato un vettore \mathbf{v} di dimensione n , scrivendo $\mathbf{c}=\text{poly}(\mathbf{v})$ è possibile costruire gli $n+1$ coefficienti del polinomio $p(x) = \sum_{k=1}^{n+1} \mathbf{c}(k)x^{n+1-k}$ che coincide con $\prod_{k=1}^n (x - \mathbf{v}(k))$. In aritmetica esatta si ha $\mathbf{v} = \text{roots}(\text{poly}(\mathbf{v}))$, tuttavia ciò potrebbe non verificarsi a causa degli errori di arrotondamento, come si può constatare richiamando il comando $\text{roots}(\text{poly}([1:n]))$, dove n varia da 2 fino a 25.

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

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

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

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 essa viene calcolata con **MAT&OCT**.

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 `MAT&OCT` 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* `MAT&OCT` 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

$$\begin{bmatrix} f_i \\ f_{i-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} f_{i-1} \\ f_{i-2} \end{bmatrix} \quad (1.15)$$

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

Esercizio 1.15 Sappiamo che $\lim_{n \rightarrow \infty} (1 + \frac{1}{n})^n = e$. È sensato approssimare numericamente il numero di Nepero e con un valore molto alto di n ?

Equazioni non lineari

Il calcolo degli *zeri* di una funzione f reale di variabile reale o, equivalentemente, 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 nella Sezione 1.6.1 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.

Inizieremo il capitolo formulando alcuni semplici problemi di interesse applicativo che conducono ad equazioni non lineari. La risoluzione di tali problemi verrà poi svolta nel seguito, dopo aver introdotto ed analizzato i diversi metodi numerici. Questa impostazione verrà poi riproposta in tutti i Capitoli che seguono.

2.1 Alcuni problemi

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

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

deduciamo che r è la radice dell'equazione non lineare

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

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. ■

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

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

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 $x^+ =$

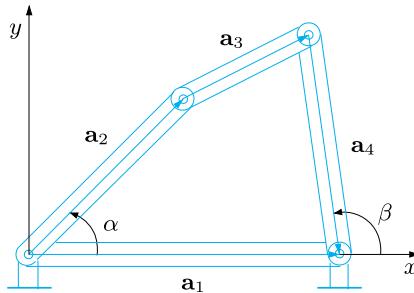


Figura 2.1. Il sistema di quattro aste del Problema 2.3

$\phi(x) = xR(x)$ stabilisce un legame fra il numero x di individui di una generazione ed il numero x^+ di individui della generazione successiva. 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, (noto come modello di Beverton-Holt o modello discreto di Verhulst)

$$R(x) = R_V(x) = \frac{r}{1 + xK}, \quad r > 0, \quad K > 0, \quad (2.3)$$

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

$$R(x) = R_P = \frac{rx}{1 + (x/K)^2}, \quad (2.4)$$

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

La dinamica di una popolazione è quindi descritta dal processo iterativo

$$x^{(k)} = \phi(x^{(k-1)}), \quad k \geq 1, \quad (2.5)$$

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 (2.5) è un esempio di iterazione di punto fisso (si veda la Sezione 2.6). ■

2.2 Il metodo di bisezione

Sia f una funzione continua in $[a, b]$ tale che $f(a)f(b) < 0$. Sotto tali ipotesi f ammette almeno uno zero in (a, b) . (Questo risultato è noto come *Teorema degli zeri di una funzione continua*.) 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

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

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:

$$\text{dato } x^{(k-1)} = (a^{(k-1)} + b^{(k-1)})/2,$$

$$\text{se } f(x^{(k-1)}) = 0,$$

$$\text{allora } \alpha = x^{(k-1)}$$

ed il metodo si arresta;

altrimenti,

$$\text{se } f(a^{(k-1)})f(x^{(k-1)}) < 0$$

$$\text{si pone } a^{(k)} = a^{(k-1)}, \quad b^{(k)} = x^{(k-1)};$$

$$\text{se } f(x^{(k-1)})f(b^{(k-1)}) < 0$$

$$\text{si pone } a^{(k)} = x^{(k-1)}, \quad b^{(k)} = b^{(k-1)},$$

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

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

$$I^{(0)} = (-0.25, 1.25), \quad x^{(0)} = 0.5,$$

$$I^{(1)} = (0.5, 1.25), \quad x^{(1)} = 0.875,$$

$$I^{(2)} = (0.875, 1.25), \quad x^{(2)} = 1.0625,$$

$$I^{(3)} = (0.875, 1.0625), \quad x^{(3)} = 0.96875.$$

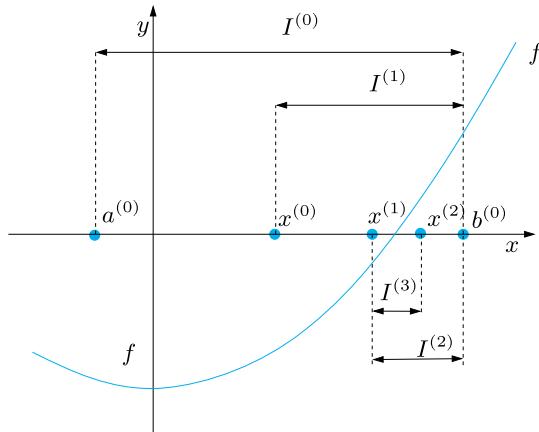


Figura 2.2. Alcune iterazioni del metodo di bisezione

Si noti che ognuno degli intervalli $I^{(k)}$ contiene lo zero α . Inoltre, la successione $\{x^{(k)}\}$ converge necessariamente allo zero α in quanto ad ogni passo l'ampiezza $|I^{(k)}| = b^{(k)} - a^{(k)}$ dell'intervalllo $I^{(k)}$ 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

$$k_{\min} > \log_2 \left(\frac{b - a}{\varepsilon} \right) - 1 \quad (2.6)$$

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

Il metodo di bisezione è implementato nel Programma 2.1: `fun` è un *function handle* associato alla funzione f , `a` e `b` sono gli estremi dell'intervallo di ricerca, `tol` la tolleranza ε e `kmax` 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.

`find`
`varargin`

Programma 2.1. bisection: il metodo di bisezione

```

function [zero,res,niter]=bisection(fun,a,b,tol,%
                                    kmax,varargin)
%BISECTION Trova uno zero di una funzione.
%  ZERO=BISECTION(FUN,A,B,TOL,KMAX) approssima uno
%  zero della funzione FUN nell'intervallo [A,B] con
%  il metodo di bisezione. FUN deve essere definita
%  su variabile di tipo array e puo' essere una anonymous
%  function od una function definita in un M-file.
%  Se la ricerca dello zero di FUN fallisce, il
%  programma restituisce un messaggio d'errore.
%
%  ZERO=BISECTION(FUN,A,B,TOL,KMAX,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 = 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 < kmax
    niter = niter + 1;
    if fx(1)*fx(2) < 0
        x(3) = x(2);
        x(2) = x(1)+(x(3)-x(1))*0.5;
        fx = 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 = fun(x,varargin{:});
        I = (x(3)-x(1))*0.5;
    else
        x(2) = x(find(fx==0)); I = 0;
    end
end
if (niter==kmax && I > tol)
    fprintf(['Il metodo di bisezione si e'' arrestato',...
             ' senza soddisfare la tolleranza richiesta\n',...
             ' avendo raggiunto il numero massimo di iterazioni\n']);
end
zero = x(2); x = x(2);
res = fun(x,varargin{:});

```

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

```
M=6000; v=1000; f=@(r) (M-v*(1+r).*((1+r).^5-1)./r);
fplot(f,[0.01,0.3]);
```

(si ricorda che stiamo deliberatamente omettendo il *prompt* allo scopo di alleggerire le notazioni) si ricava che f presenta un'unica radice nell'intervallo $(0.01, 0.1)$, pari a circa 0.06. Eseguiamo quindi il Programma 2.1 con $a = 0.01$, $b = 0.1$, $tol = 10^{-12}$ e $kmax=1000$ ed il comando

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

Il metodo converge dopo 36 iterazioni al valore 0.061402411536183, in perfetto accordo con la stima (2.6) per la quale $k_{\min} = 36$. Si può quindi concludere che il tasso di interesse r è pari a circa il 6.14%.

Invece di lavorare con una *anonymous function*, avremmo potuto generare la *function* Rfuncv.m

```
function y=Rfuncv(r,M,v)
% RFUNCV function per l'Esempio 2.1
y=M - v*(1+r)./r.*((1+r).^5 - 1);
end
```

ed eseguire le seguenti istruzioni:

```
M=6000; v=1000;
[zero,res,niter]=bisection(@Rfuncv,0.01,0.1,%
1.e-12,1000,M,v)
```

Osserviamo che nel primo caso abbiamo richiamato la *function* bisection.m con 5 parametri di input, in quanto i valori di M e v sono incorporati nel *function handle* f al momento della sua definizione. Al contrario, nel secondo caso dobbiamo passare a bisection.m anche M e v , essi saranno memorizzati nella variabile varargin e poi passati a Rfuncv al momento della sua valutazione. ■

Il metodo di bisezione non garantisce una riduzione monotona 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.



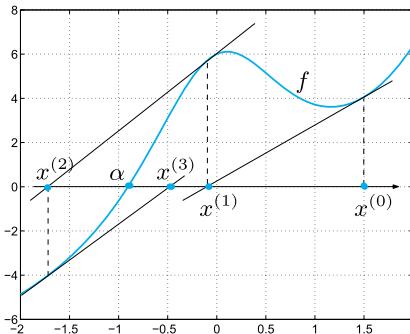


Figura 2.3. Prime iterate generate dal metodo di Newton a partire dal dato iniziale $x^{(0)}$ per la funzione $f(x) = x + e^x + 10/(1+x^2) - 5$

2.3 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 sfrutti maggiori informazioni su f , precisamente 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

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{f'(x^{(k)})}, \quad k \geq 0 \quad (2.7)$$

purché $f'(x^{(k)}) \neq 0$. La (2.7) 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

$$f(x^{(k+1)}) = f(x^{(k)}) + \delta^{(k)} f'(x^{(k)}) + \mathcal{O}((\delta^{(k)})^2), \quad (2.8)$$

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 (2.7). In questo senso la (2.7) può essere vista come una approssimazione della (2.8).

Evidentemente, (2.7) 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 , ovvero appartenenti ad un intorno $I(\alpha)$ sufficientemente piccolo di α . Questa richiesta a prima vista sembra insensata: per trovare l'incognita α abbiamo bisogno di scegliere $x^{(0)}$ sufficientemente vicino ad α , quando α è proprio il valore sconosciuto!

In pratica, un possibile valore di $x^{(0)}$ può essere ottenuto 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)

$$\lim_{k \rightarrow \infty} \frac{x^{(k+1)} - \alpha}{(x^{(k)} - \alpha)^2} = \frac{f''(\alpha)}{2f'(\alpha)} \quad (2.9)$$

La (2.9) 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, ovvero $f'(\alpha) = 0, \dots, f^{(m-1)}(\alpha) = 0$, il metodo di Newton è ancora convergente, purché $x^{(0)}$ sia scelto opportunamente e $f'(x) \neq 0 \forall x \in I(\alpha) \setminus \{\alpha\}$. Tuttavia in questo caso l'ordine di convergenza è pari a 1 (si veda l'Esercizio 2.17). In tal caso, l'ordine 2 può essere ancora recuperato usando anziché (2.7) la relazione

$$x^{(k+1)} = x^{(k)} - m \frac{f(x^{(k)})}{f'(x^{(k)})}, \quad k \geq 0 \quad (2.10)$$

purché $f'(x^{(k)}) \neq 0$. 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 [QSSG14, Sez. 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 (2.7) e con la sua versione modificata (2.10). 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 (2.7) l'errore decresce solo linearmente. ■

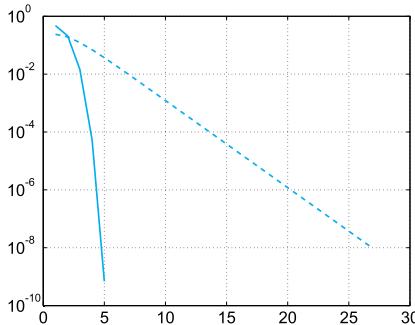


Figura 2.4. Errore in scala semi-logaritmica in funzione del numero di iterazioni per la funzione dell’Esempio 2.3. La curva tratteggiata corrisponde al metodo di Newton (2.7), quella continua al metodo di Newton modificato (2.10) (con $m = 2$)

2.3.1 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

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

Si tratta di un test sull’errore. Sfortunatamente essendo l’errore incondito, 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 della Sezione 2.6, 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

$$|x^{(k_{\min})} - x^{(k_{\min}-1)}| < \varepsilon \quad (2.11)$$

Si tratta di un test sull’incremento. Come vedremo nella Sezione 2.6.1, 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* al passo k 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

$$|r^{(k_{\min})}| = |f(x^{(k_{\min})})| < \varepsilon \quad (2.12)$$

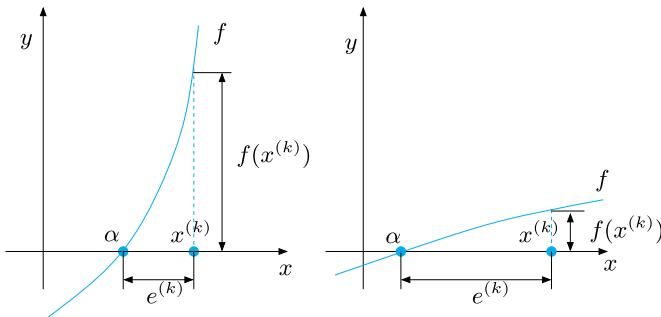


Figura 2.5. Le due possibili situazioni nelle quali il residuo non è un buon stimatore dell'errore: $|f'(x)| \gg 1$ (a sinistra), $|f'(x)| \ll 1$ (a destra), con x appartenente ad un intervallo contenente α

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 vedano gli Esercizi 2.6 e 2.20).

Nel Programma 2.2 viene riportata una implementazione del metodo di Newton nella sua forma (2.7) (per utilizzare la forma modificata è sufficiente inserire, invece di f' , la funzione f'/m). I parametri `fun` e `dfun` sono *function handle* associati alla funzione f ed alla sua derivata prima, mentre `x0` è il dato iniziale. Il metodo viene arrestato se il valore assoluto della differenza fra due iterate consecutive è minore della tolleranza `tol` o se è stato oltrepassato il massimo numero di iterazioni consentito, `kmax`. I parametri di *output* contengono: `zero` l'approssimazione calcolata $\hat{\alpha}$ della radice α , `res` il valore $f(\hat{\alpha})$ del residuo, `niter` il numero minimo di iterazioni necessarie a soddisfare il test d'arresto e `difv` il vettore dei valori assoluti $|x^{(k+1)} - x^{(k)}|$ delle differenze tra due iterate successive (utilizzati per il test d'arresto).

Programma 2.2. `newton`: il metodo di Newton

```
function [zero,res,niter,difv]=newton(fun,dfun,x0,%
                                         tol,kmax,varargin)
%NEWTON Trova uno zero di una funzione.
% ZERO=NEWTON(FUN,DFUN,X0,TOL,KMAX) approssima lo
% zero ZERO della funzione definita nella function
% FUN, continua e derivabile, usando il metodo di
% Newton e partendo da X0. Se la ricerca
% dello zero fallisce in KMAX iterazioni, il pro-
% grammma restituisce un messaggio d'errore.
% FUN e DFUN possono essere anonymous
% function o function definite in M-file.
% ZERO=NEWTON(FUN,DFUN,X0,TOL,KMAX,P1,P2,...) passa
```

```
% i parametri P1,P2,... alle funzioni
% FUN(X,P1,P2,...) e DFUN(X,P1,P2,...).
% [ZERO,RES,NITER,DIFV]= NEWTON(FUN,...) restituisce
% il valore del residuo RES in ZERO, il numero di
% iterazioni NITER necessario per calcolare ZERO ed
% il vettore DIFV degli incrementi |x^(k+1)-x^k|
%
x = x0;
fx = fun(x,varargin{:});
dfx = dfun(x,varargin{:});
k = 0; diff = tol+1; difv=[ ];
while diff >= tol && k < kmax
    k = k + 1;
    diff = - fx/dfx;
    x = x + diff;
    diff = abs(diff); difv=[difv; diff];
    fx = fun(x,varargin{:});
    dfx = dfun(x,varargin{:});
end
if (k==kmax && diff > tol)
    printf(['Newton si e' arrestato senza aver',...
        'soddisfatto l''accuratezza richiesta, avendo\n',...
        'raggiunto il massimo numero di iterazioni\n']);
end
zero = x; res = fx; niter=k;
```

2.4 Il metodo delle secanti

In molte applicazioni è possibile che la funzione f di cui vogliamo calcolare gli zeri non sia nota in forma esatta, ma che sia solo valutabile in un punto x assegnato attraverso un programma. Di conseguenza, ci risulta impossibile poter valutare la sua derivata in maniera esatta ed applicare il metodo di Newton.

Per ovviare a questo inconveniente, la valutazione di $f'(x^{(k)})$ può essere sostituita da un rapporto incrementale calcolato su valori di f già noti. Una possibile implementazione di questa strategia è quella del metodo delle secanti: assegnati due punti $x^{(0)}$ e $x^{(1)}$, per $k \geq 1$ si calcola

$$x^{(k+1)} = x^{(k)} - \left(\frac{f(x^{(k)}) - f(x^{(k-1)})}{x^{(k)} - x^{(k-1)}} \right)^{-1} f(x^{(k)}) \quad (2.13)$$

Rinunciando alla conoscenza esatta della derivata prima, in caso di convergenza, la velocità sarà inferiore a quella del metodo di Newton. In effetti si può dimostrare che, se α è radice semplice e $I(\alpha)$ un suo opportuno intorno, se $x^{(0)}$ e $x^{(1)}$ sono sufficientemente vicini ad α e $f'(x) \neq 0 \forall x \in I(\alpha) \setminus \{\alpha\}$, allora il metodo delle secanti (2.13) converge ad α . Inoltre, se $f \in C^2(I(\alpha))$ e $f'(\alpha) \neq 0$, allora esiste una costante

$c > 0$ tale che

$$|x^{(k+1)} - \alpha| \leq c|x^{(k)} - \alpha|^p, \quad \text{con } p = \frac{1 + \sqrt{5}}{2} \simeq 1.618\dots \quad (2.14)$$

cioè il metodo delle secanti converge con ordine p *super-lineare*. Se invece la radice α è multipla, allora la convergenza è soltanto lineare come succederebbe usando il metodo di Newton.

Esempio 2.4 Risolviamo con il metodo delle secanti lo stesso caso dell'Esempio 2.1 a partire dai dati iniziali $x^{(0)} = 0.3$ e $x^{(1)} = -0.3$. Il metodo converge allo zero cercato in 8 iterazioni, contro le 6 iterazioni necessarie al metodo di Newton partendo da $x^{(0)} = 0.3$.

Scegliendo $x^{(0)} = 0.3$ e $x^{(1)} = 0.1$ il metodo delle secanti convergerebbe in 6 iterazioni, al pari di Newton. ■

2.5 I sistemi di equazioni non lineari

Consideriamo il seguente sistema di equazioni non lineari

$$\begin{cases} f_1(x_1, x_2, \dots, x_n) = 0, \\ f_2(x_1, x_2, \dots, x_n) = 0, \\ \vdots \\ f_n(x_1, x_2, \dots, x_n) = 0, \end{cases} \quad (2.15)$$

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 (2.15) nella forma

$$\mathbf{f}(\mathbf{x}) = \mathbf{0}. \quad (2.16)$$

Un semplice esempio di sistema non lineare è il seguente

$$\begin{cases} f_1(x_1, x_2) = x_1^2 + x_2^2 - 1 = 0, \\ f_2(x_1, x_2) = \sin(\pi x_1/2) + x_2^3 = 0. \end{cases} \quad (2.17)$$

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 (1.11)). Con questa notazione, il metodo di Newton

per (2.16) diventa: dato $\mathbf{x}^{(0)} \in \mathbb{R}^n$, per $k = 0, 1, \dots$, fino a convergenza

$$\begin{aligned} & \text{risolvere } J_{\mathbf{f}}(\mathbf{x}^{(k)})\delta\mathbf{x}^{(k)} = -\mathbf{f}(\mathbf{x}^{(k)}); \\ & \text{porre } \mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \delta\mathbf{x}^{(k)} \end{aligned} \quad (2.18)$$

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

Il Programma 2.3 implementa il metodo di Newton per un sistema non lineare usando il comando `\` di MATLAB (si veda la Sezione 5.8) per risolvere il sistema lineare sulla matrice Jacobiana. In ingresso, è necessario definire un vettore che rappresenta il dato iniziale e due *function*, `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`. `Ffun` e `Jfun` possono essere *function handles* o *user-defined functions*. 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 `kmax`. Le variabili in output assumono lo stesso significato che hanno nel Programma `newton` 2.2, ad eccezione di `difv` che ora contiene i valori $\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\|$ al variare di $k = 0, \dots$, fino a convergenza.

Programma 2.3. newtonsys: il metodo di Newton per un sistema non lineare

```
function [x,res,niter,difv] = newtonsys(Ffun,Jfun,...  
x0,tol, kmax, varargin)  
%NEWTONSYS calcola una radice di un sistema non lineare  
% [ZERO,RES,NITER,DIFV]=NEWTONSYS(FFUN,JFUN,X0,...  
% TOL, KMAX)  
% calcola il vettore ZERO, radice di un sistema non  
% lineare definito nella function FFUN con matrice  
% Jacobiana definita nella function JFUN a partire  
% dal vettore X0. RES contiene il valore del residuo  
% in ZERO e NITER il numero di iterazioni necessarie  
% per calcolare ZERO. Il vettore DIFV contiene le  
% norme ||x^(k+1)-x^(k)||. FFUN e JFUN possono essere  
% anonymous functions o user-defined functions.  
  
k = 0;  
err = tol + 1; difv=[ ];  
x = x0;  
while err >= tol && k < kmax  
    J = Jfun(x,varargin{:});  
    F = Ffun(x,varargin{:});  
    delta = - J\F;  
    x = x + delta;  
    err = norm(delta); difv=[difv; err];  
    k = k + 1;  
end  
res = norm(Ffun(x,varargin{:}));  
if (k==kmax && err> tol)
```

```

fprintf(['Il metodo non converge nel massimo ',...
         'numero di iterazioni. L''ultima iterata\n',...
         'calcolata ha residuo relativo pari a %e\n'],F);
end
niter=k;

```

Esempio 2.5 Consideriamo il sistema non lineare (2.17) 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).

Partendo dal dato iniziale $\mathbf{x}_0=[1; 1]$ e usando le seguenti istruzioni:

```

Ffun=@(x)[x(1)^2+x(2)^2-1;
           sin(pi*x(1)/2)+x(2)^3];
pi2 = 0.5*pi;
Jfun=@(x)[2*x(1), 2*x(2);
           pi2*cos(pi2*x(1)), 3*x(2)^2];
x0=[1;1]; tol=1e-5; kmax=10;
[x,F,niter,difv] = newtonsys(Ffun,Jfun,x0,tol,kmax);

```

il metodo di Newton converge in 8 iterazioni al vettore

```

4.760958225338114e-01
-8.793934089897496e-01

```

Si noti che per far convergere il metodo all'altra radice basta scegliere come dato iniziale $\mathbf{x}_0=[-1; -1]$. 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 $\det(\mathbf{J}_f(\mathbf{x}^{(0)})) \neq 0$. ■

Il metodo delle secanti può essere adattato alla risoluzione di sistemi di equazioni non lineari, mantenendo l'ordine di convergenza super-lineare. L'idea di base è quella di sostituire le matrici Jacobiane $\mathbf{J}_f(\mathbf{x}^{(k)})$ (per $k \geq 0$) con delle matrici B_k definite ricorsivamente a partire da una matrice B_0 che sia una approssimazione di $\mathbf{J}_f(\mathbf{x}^{(0)})$. (Approssimazioni alternative verranno considerate nella Sezione 4.2 e nel Capitolo 9.) Il metodo più noto che si basa su questa idea è quello di Broyden. Utilizzando le stesse notazioni della sezione precedente, esso si formula così: dato $\mathbf{x}^{(0)} \in \mathbb{R}^n$, data $B_0 \in \mathbb{R}^{n \times n}$, per $k = 0, 1, \dots$, fino a convergenza

$$\begin{aligned}
&\text{risolvere } B_k \delta \mathbf{x}^{(k)} = -\mathbf{f}(\mathbf{x}^{(k)}) \\
&\text{porre } \mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \delta \mathbf{x}^{(k)} \\
&\text{porre } \delta \mathbf{f}^{(k)} = \mathbf{f}(\mathbf{x}^{(k+1)}) - \mathbf{f}(\mathbf{x}^{(k)}) \\
&\text{calcolare } B_{k+1} = B_k + \frac{(\delta \mathbf{f}^{(k)} - B_k \delta \mathbf{x}^{(k)}) \delta \mathbf{x}^{(k)T}}{\delta \mathbf{x}^{(k)T} \delta \mathbf{x}^{(k)}} \tag{2.19}
\end{aligned}$$

Facciamo osservare che non si chiede alla successione $\{B_k\}$ così costruita di convergere alla vera matrice Jacobiana $\mathbf{J}_f(\boldsymbol{\alpha})$ ($\boldsymbol{\alpha}$ è la radice del sistema); questo risultato in effetti non è nemmeno garantito. Piuttosto, si ha

$$\lim_{k \rightarrow \infty} \frac{\|(B_k - J_f(\alpha))(x^{(k)} - \alpha)\|}{\|x^{(k)} - \alpha\|} = 0.$$

Ciò esprime il fatto che B_k approssima bene $J_f(\alpha)$ lungo la direzione dell'errore $x^{(k)} - \alpha$, garantendo una convergenza super-lineare.

Ad ogni passo, il costo $\mathcal{O}(n^3)$ per il calcolo di $\delta x^{(k)}$ può essere ridotto ad $\mathcal{O}(n^2)$, utilizzando ricorsivamente fattorizzazioni QR sulle matrici B_k (si veda, ad esempio, [GM72]) e, grazie all'uguaglianza $(\delta f^{(k)} - B_k \delta x^{(k)}) = f(x^{(k+1)})$, non serve implementare prodotti matrice-vettore nel calcolo di B_{k+1} .

Per una descrizione più completa del metodo di Broyden e di altri metodi di tipo secanti (detti anche metodi *quasi-Newton*) rimandiamo a [DS96], [Deu04], [SM03] e [QSSG14, Cap. 6].

Il metodo di Broyden (2.19) è implementato nel Programma 2.4. Il parametro di *input* $B0$ contiene la matrice B_0 , tutti gli altri parametri sono definiti come per il Programma 2.3.

Programma 2.4. broyden: il metodo di Broyden

```

function [zero,res,niter,difv]=broyden(fun,B0,x0,%
                                         tol, kmax,varargin)
%BROYDEN Trova uno zero di un sistema di funzioni.
% ZERO=BROYDEN(FUN,B0,X0,TOL,KMAX) approssima lo
% zero ZERO del sistema di funzioni definite nella
% function FUN, usando il metodo di Broyden
% partendo da X0, dove B0 e' l'approssimazione
% dello Jacobiano al passo 0. FUN accetta in ingresso
% un vettore x e restituisce un vettore della stessa
% dimensione. Se la ricerca dello zero fallisce in
% KMAX iterazioni, il programma restituisce un mes-
% saggio d'errore. FUN puo' essere una anonymous
% function o una function definita in M-file.
% ZERO=BROYDEN(FUN,B0,X0,TOL,KMAX,P1,P2,...)
% passa i parametri P1,P2,... alla funzione
% FUN(X,P1,P2,...).
% [ZERO,RES,NITER,DIFV]= BROYDEN(FUN,...) restituisce
% il valore del residuo RES in ZERO, il numero di
% iterazioni NITER necessario per calcolare ZERO ed
% il vettore DIFV delle norme ||x^(k+1)-x^(k)||

fx0 = fun(x0,varargin{:});
k = 0;
diff = tol+1; difv= [ ];
while diff >= tol && k < kmax
    k = k + 1;
    deltax=-B0\fx0;
    x1=x0+deltax;
    fx1=fun(x1,varargin{:});
    B0=B0+(fx1*deltax')/(deltax'*deltax);
    diff = norm(deltax);
    difv=[difv;diff];
    x0=x1; fx0=fx1;
end
zero = x1; res = norm(fx1);
if (k==kmax && diff > tol)

```

```

fprintf(['Broyden si e\' arrestato senza aver ',...
    'soddisfatto l\'accuratezza richiesta, avendo\n',...
    'raggiunto il massimo numero di iterazioni\n']);
end
niter=k;

```

Esempio 2.6 Consideriamo il problema dell’Esempio 2.5 e risolviamolo con il metodo di Broyden (2.19). Prendendo $B_0 = I$, tolleranza $\varepsilon = 10^{-5}$ per il test d’arresto sull’incremento e $\mathbf{x}^{(0)} = (1, 1)^T$ otteniamo convergenza in 10 iterazioni al punto $(0.476095825652119, -0.879393405072448)^T$ con un residuo in norma pari a $1.324932e-08$, contro le 8 iterazioni del metodo di Newton ed un residuo in norma pari a $2.235421e-11$. Scegliendo $\mathbf{x}^{(0)} = (-1, -1)^T$, sempre con $B_0 = I$, otteniamo convergenza alla seconda radice in 17 iterazioni con residuo in norma uguale a $5.744382e-08$ contro 8 iterazioni di Newton e residuo in norma uguale a $2.235421e-11$. Scegliendo invece $B_0 = 2I$, il numero delle iterazioni di Broyden si riduce a 12, evidenziando quanto sia importante scegliere bene la matrice iniziale al fine di velocizzare la convergenza.

Per quanto riguarda l’accuratezza delle soluzioni calcolate, osserviamo che il residuo “di Newton” è di 3 ordini di grandezza inferiore al residuo “di Broyden” inducendoci a concludere che le soluzioni ottenute con il metodo di Newton siano comunque più accurate di quelle ottenute con quello di Broyden.

Si vedano gli Esercizi 2.6–2.16.



Riassumendo

- Il calcolo degli zeri di una funzione f viene condotto attraverso metodi iterativi;
- il metodo di bisezione consente di approssimare uno zero di una funzione “incapsulandolo” in intervalli la cui ampiezza viene dimezzata ad ogni iterazione. Esso converge sempre 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 α di f viene condotta utilizzando i valori assunti da f e dalla sua derivata prima. Esso generalmente converge solo per valori del dato iniziale sufficientemente vicini ad α ;
- quando converge, il metodo di Newton converge quadraticamente se α è uno zero semplice, linearmente altrimenti;
- il metodo di Newton può essere esteso al caso del calcolo degli zeri di un sistema di equazioni non lineari;
- il metodo delle secanti è una approssimazione di quello di Newton in cui la derivata prima sia sostituita da un rapporto incrementale. Se α è semplice, esso converge più che linearmente, ma meno che quadraticamente; se α è multipla esso converge linearmente. Come per il metodo di Newton, i punti iniziali devono essere scelti in prossimità della radice.

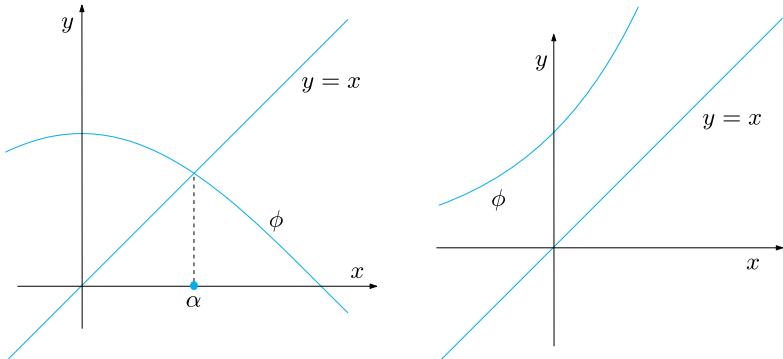


Figura 2.6. La funzione $\phi(x) = \cos x$ (a sinistra) ammette un solo punto fisso, mentre la funzione $\phi(x) = e^x$ (a destra) non ne ammette alcuno

2.6 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 di numeri reali

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

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 potrebbe essere usato per il calcolo degli zeri di f (uno solo, in questo caso). D'altra parte non tutte le funzioni ammettono punti fissi; se ad esempio si ripete l'esperimento precedente con la funzione esponenziale a partire da $x^{(0)} = 1$, dopo soli 4 passi si giunge ad una situazione di *overflow* (si veda la Figura 2.6). Precisiamo 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

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

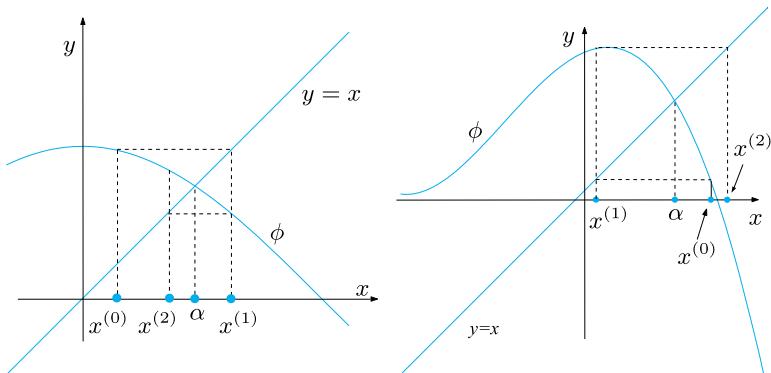


Figura 2.7. Rappresentazione delle prime iterazioni di punto fisso per due funzioni di iterazione. Le iterazioni convergono verso il punto fisso α (*a sinistra*), mentre si allontanano da α (*a destra*)

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

$$x^{(k+1)} = \phi(x^{(k)}), \quad k \geq 0 \quad (2.20)$$

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 per la funzione $\phi(x) = \cos(x)$.

Un'interpretazione geometrica della (2.20) 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 2.7 Il metodo di Newton (2.7) può essere riletto come un algoritmo di iterazioni di punto fisso per la funzione

$$\phi(x) = x - \frac{f(x)}{f'(x)}. \quad (2.21)$$

Tale funzione verrà d'ora in poi indicata con il simbolo ϕ_N , dove N sta per Newton. I metodi di bisezione e di secanti non sono invece iterazioni 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 le iterazioni di punto fisso convergano. Vale infatti il seguente risultato:

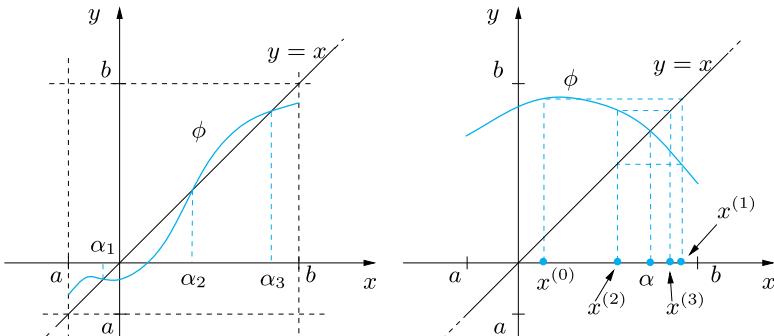


Figura 2.8. Una funzione di punto fisso che ammette 3 punti fissi (*a sinistra*), una funzione di punto fisso che soddisfa l’ipotesi (2.22) ed i primi elementi della successione (2.24) convergente all’unico punto fisso α (*a destra*)

Proposizione 2.1 Consideriamo la successione (2.20).

1. Supponiamo che $\phi(x)$ sia continua in $[a, b]$ e che $\phi(x) \in [a, b]$ per ogni $x \in [a, b]$; allora esiste almeno un punto fisso $\alpha \in [a, b]$.
2. Se supponiamo inoltre che

$$\exists L < 1 \text{ t.c. } |\phi(x_1) - \phi(x_2)| \leq L|x_1 - x_2| \quad \forall x_1, x_2 \in [a, b], \quad (2.22)$$

allora ϕ ha un unico punto fisso $\alpha \in [a, b]$ e la successione definita nella (2.20) converge a α , qualunque sia il dato iniziale $x^{(0)}$ in $[a, b]$.

Dimostrazione. 1. Dimostriamo dapprima l’esistenza di punti fissi per ϕ . Definiamo la funzione $g(x) = \phi(x) - x$, essa è continua per costruzione su $[a, b]$ e, per l’ipotesi sull’immagine di ϕ , si ha $g(a) = \phi(a) - a \geq 0$ e $g(b) = \phi(b) - b \leq 0$. Applicando il teorema degli zeri di una funzione continua, concludiamo che g ammette almeno uno zero in $[a, b]$, ovvero ϕ ammette almeno un punto fisso in $[a, b]$. (Per un esempio si veda la Figura 2.8.)

2. Supponiamo ora che valga l’ipotesi (2.22).

Se esistessero due punti fissi distinti α_1 e α_2 avremmo

$$|\alpha_1 - \alpha_2| = |\phi(\alpha_1) - \phi(\alpha_2)| \leq L|\alpha_1 - \alpha_2| < |\alpha_1 - \alpha_2|,$$

il che è assurdo.

Dimostriamo ora che la successione $x^{(k)}$ definita in (2.20) converge per $k \rightarrow \infty$ all’unico punto fisso α , per ogni scelta del dato iniziale $x^{(0)} \in [a, b]$. Abbiamo

$$\begin{aligned} 0 &\leq |x^{(k+1)} - \alpha| = |\phi(x^{(k)}) - \phi(\alpha)| \\ &\leq L|x^{(k)} - \alpha| \leq \dots \leq L^{k+1}|x^{(0)} - \alpha|, \end{aligned}$$

ovvero, $\forall k \geq 0$,

$$\frac{|x^{(k)} - \alpha|}{|x^{(0)} - \alpha|} \leq L^k. \quad (2.23)$$

Passando al limite per $k \rightarrow \infty$, otteniamo $\lim_{k \rightarrow \infty} |x^{(k)} - \alpha| = 0$, che è il risultato cercato. \blacksquare

Nella pratica è però spesso difficile delimitare *a priori* l'ampiezza dell'intervallo $[a, b]$; in tal caso è utile il seguente risultato di convergenza *locale*, per la cui dimostrazione si rimanda a [OR70].

Teorema 2.1 (di Ostrowski) *Sia α un punto fisso di una funzione ϕ continua e derivabile con continuità in un opportuno intorno \mathcal{J} di α . Se risulta $|\phi'(\alpha)| < 1$, allora esiste $\delta > 0$ in corrispondenza del quale la successione $\{x^{(k)}\}$ converge ad α , per ogni $x^{(0)}$ tale che $|x^{(0)} - \alpha| < \delta$. Inoltre si ha*

$$\lim_{k \rightarrow \infty} \frac{x^{(k+1)} - \alpha}{x^{(k)} - \alpha} = \phi'(\alpha) \quad (2.24)$$

Dimostrazione. Limitiamoci a verificare la proprietà (2.24). Per il teorema di Lagrange, per ogni $k \geq 0$, esiste un punto ξ_k compreso tra $x^{(k)}$ e α tale che $x^{(k+1)} - \alpha = \phi(x^{(k)}) - \phi(\alpha) = \phi'(\xi_k)(x^{(k)} - \alpha)$, ovvero

$$(x^{(k+1)} - \alpha)/(x^{(k)} - \alpha) = \phi'(\xi_k). \quad (2.25)$$

Poiché ξ_k è compreso tra $x^{(k)}$ ed α , si ha $\lim_{k \rightarrow \infty} \xi_k = \alpha$ e, passando al limite in entrambi i termini di (2.25) e ricordando che ϕ' è continua in un intorno di α , si ottiene (2.24). \blacksquare

Dalla (2.23) e dalla (2.24) 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 (L in (2.23), $\phi'(\alpha)$ in (2.24)) indipendente da k ed il cui valore assoluto è minore di 1.

Per questo motivo tale costante viene detta *fattore di convergenza asintotico*. Va infine osservato che la convergenza sarà tanto più rapida quanto più piccola è tale costante.

Osservazione 2.1 Nel caso in cui $|\phi'(\alpha)| > 1$, dalla (2.25) segue che se $x^{(k)}$ è sufficientemente vicino ad α , in modo tale che $|\phi'(x^{(k)})| > 1$, allora $|\alpha - x^{(k+1)}| > |\alpha - x^{(k)}|$, e non è possibile che la successione converga al punto fisso. Quando invece $|\phi'(\alpha)| = 1$ non si può trarre alcuna conclusione poiché potrebbero verificarsi sia la convergenza sia la divergenza, a seconda delle caratteristiche della funzione di punto fisso. \blacksquare

Esempio 2.8 La funzione $\phi(x) = \cos(x)$ soddisfa le ipotesi del Teorema 2.1 in quanto $|\phi'(\alpha)| = |\sin(\alpha)| \simeq 0.67 < 1$ e, di conseguenza per continuità, esiste un

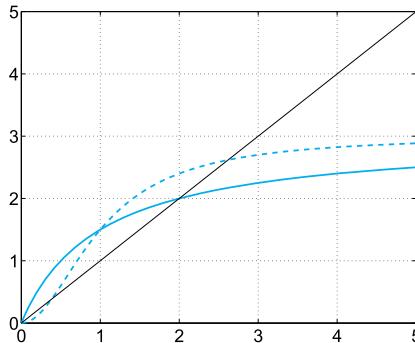


Figura 2.9. I punti fissi per due diversi modelli di dinamica delle popolazioni: il modello discreto di Verhulst (in linea continua) e quello predatore/preda (in linea tratteggiata)

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 2.9 (Dinamica delle popolazioni) Applichiamo le iterazioni di punto fisso alla funzione $\phi_V(x) = rx/(1+xK)$ del modello discreto di Verhulst (2.3) ed alla funzione $\phi_P(x) = rx^2/(1+(x/K)^2)$ del modello predatore/preda (2.4) 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 2.9). Il punto fisso $\alpha = 0$ comune a ϕ_V e ϕ_P può essere calcolato solo come punto fisso di ϕ_P , ma non di ϕ_V . Infatti $\phi'_P(\alpha) = 0$, mentre $|\phi'_V(\alpha)| = r > 1$. Analogamente il punto fisso $\alpha = 0.3820\dots$ di ϕ_P non può essere calcolato in quanto $|\phi'_P(\alpha)| > 1$. Per lo svolgimento di questo esercizio abbiamo utilizzato il Programma 10.1 `fixedpoint.m` (si veda il Capitolo 10). ■

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 del Teorema 2.1. Se, inoltre, ϕ è derivabile con continuità due volte e se*

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

allora il metodo di punto fisso (2.20) è convergente di ordine 2 e si ha

$$\lim_{k \rightarrow \infty} \frac{x^{(k+1)} - \alpha}{(x^{(k)} - \alpha)^2} = \frac{1}{2} \phi''(\alpha) \quad (2.26)$$

Dimostrazione. Basta osservare che, in questo caso,

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

per un opportuno $\eta^{(k)}$ compreso tra $x^{(k)}$ e α . ■

L'Esempio 2.7 mostra che le iterazioni di punto fisso (2.20) possono servire anche per il calcolo degli zeri di funzioni. Naturalmente, data una funzione f , la ϕ definita in (2.21) 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 (2.21) 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 funzione 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 α .

2.6.1 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 nella Sezione 1.6.3) 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

$$\alpha - x^{(k)} = \frac{1}{1 - \phi'(\xi^{(k)})}(x^{(k+1)} - x^{(k)}). \quad (2.27)$$

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 2.10 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

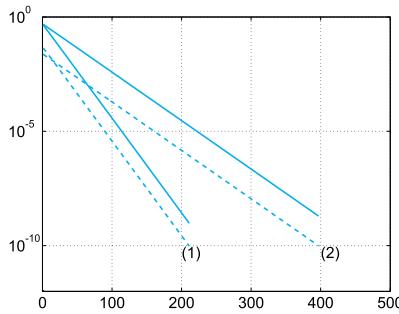


Figura 2.10. Valori assoluti degli errori (*in linea continua*) e delle differenze fra iterate (*in linea tratteggiata*) al variare delle iterazioni per il caso dell’Esempio 2.10: le curve (1) si riferiscono a $m = 11$, mentre le (2) a $m = 21$

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 2.17) 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 2.10 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 $m = 21$. ■

2.7 Accelerazione con il metodo di Aitken

In questa Sezione 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 (2.24) si ricava che, per k fissato, dovrà esistere un valore λ (da determinare) tale che

$$\phi(x^{(k)}) - \alpha = \lambda(x^{(k)} - \alpha), \quad (2.28)$$

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 (2.28) ricaviamo

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

ovvero

$$\boxed{\alpha = x^{(k)} + (\phi(x^{(k)}) - x^{(k)})/(1 - \lambda)} \quad (2.29)$$

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

$$\lambda^{(k)} = \frac{\phi(\phi(x^{(k)})) - \phi(x^{(k)})}{\phi(x^{(k)}) - x^{(k)}} \quad (2.30)$$

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. Se $x^{(k+1)} = \phi(x^{(k)})$, allora $x^{(k+2)} = \phi(\phi(x^{(k)}))$ e quindi, dalla (2.30), si ricava $\lambda^{(k)} = (x^{(k+2)} - x^{(k+1)})/(x^{(k+1)} - x^{(k)})$, cioè

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

da cui, passando al limite e ricordando la (2.24), si perviene alla tesi

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

■

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 (2.30) nella (2.29) e definiamo un nuovo $x^{(k+1)}$ nel modo seguente

$$x^{(k+1)} = x^{(k)} - \frac{(\phi(x^{(k)}) - x^{(k)})^2}{\phi(\phi(x^{(k)})) - 2\phi(x^{(k)}) + x^{(k)}}, \quad k \geq 0 \quad (2.31)$$

Questa espressione è nota come *formula di estrapolazione di Aitken* e 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 2.2 *Siano $x^{(k+1)} = \phi(x^{(k)})$ le iterazioni di punto fisso (2.20), 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 estrapolazione di Aitken converge anche se le corrispondenti iterazioni di punto fisso divergono.

Nel Programma 2.5 riportiamo un'implementazione del metodo di Aitken. In esso `phi` è un *function handle* associato alla funzione ϕ 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 2.5. 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 puo' essere una anonymous
% function, o una function definita in un M-file.
% [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;
k = 0;
while k < kmax && diff >= tol
    gx = phi(x,varargin{:});
    ggx = phi(gx,varargin{:});
    xnew = (x*ggx-gx^2)/(ggx-2*gx+x);
    diff = abs(x-xnew);
    x = xnew;
    k = k + 1;
end
niter=k;
if (k==kmax && diff>tol)
    fprintf([' Il metodo non converge nel numero',...
              ' massimo di iterazioni\n']);
end
```

Esempio 2.11 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 2.5 con `tol=1.e-10`, `kmax=100`, `x0=2` e definiamo le due funzioni di iterazione come segue

```
phi0 = @(x) log(x*exp(x));
phi1 = @(x)(exp(x)+x)/(exp(x)+1);
```

A questo punto eseguiamo il Programma 2.5 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 $|\phi'_0(1)| = 2$). ■



Si vedano gli Esercizi 2.17–2.21.

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.

2.8 Polinomi algebrici

In questa Sezione consideriamo il caso in cui f sia un polinomio di grado $n \geq 0$ della forma (1.9). Come già osservato, lo spazio di tutti i polinomi (1.9) viene indicato con il simbolo \mathbb{P}_n . Si ricorda che se $p_n \in \mathbb{P}_n$, per $n \geq 2$, e $a_k \in \mathbb{R}$, se $\alpha \in \mathbb{C}$ con $\text{Im}(\alpha) \neq 0$ 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 2.3 (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 2.12 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 una 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. \blacksquare

Teorema 2.4 (di Cauchy) Tutti gli zeri di p_n sono inclusi nel cerchio Γ del piano complesso

$$\Gamma = \{z \in \mathbb{C} : |z| \leq 1 + \eta\}, \quad \text{dove } \eta = \max_{0 \leq k \leq n-1} |a_k/a_n|. \quad (2.32)$$

Questa proprietà è di scarsa utilità quando $\eta \gg 1$ (per il polinomio p_6 dell'Esempio 2.12 si ha ad esempio $\eta = 8$, mentre le radici stanno tutte all'interno di cerchi di raggio decisamente minore).

2.8.1 Il metodo di Hörner

Illustriamo in questa Sezione 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 (1.9) è equivalente alla seguente rappresentazione

$$p_n(x) = a_0 + x(a_1 + x(a_2 + \dots + x(a_{n-1} + a_n x) \dots)). \quad (2.33)$$

Tuttavia, mentre la (1.9) richiede n addizioni e $2n - 1$ moltiplicazioni per valutare $p_n(x)$ (per x fissato), la (2.33) richiede solo n addizioni più n moltiplicazioni. L'espressione (2.33), nota anche come algoritmo delle moltiplicazioni annidate, sta alla base del metodo di Hörner. Quest'ultimo consente la valutazione efficiente del polinomio p_n in un punto z mediante il seguente algoritmo di *divisione sintetica*:

$$\begin{aligned} b_n &= a_n, \\ b_k &= a_k + b_{k+1}z, \quad k = n-1, n-2, \dots, 0 \end{aligned} \quad (2.34)$$

Nella (2.34) tutti i coefficienti b_k con $k \leq n-1$ dipendono da z e possiamo verificare che $b_0 = p_n(z)$. Il polinomio

$$q_{n-1}(x; z) = b_1 + b_2 x + \dots + b_n x^{n-1} = \sum_{k=1}^n b_k x^{k-1}, \quad (2.35)$$

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 (2.34)

è stato implementato nel Programma 2.6. I coefficienti a_j del polinomio da valutare sono memorizzati nel vettore \mathbf{a} a partire da a_n fino ad a_0 .

Programma 2.6. 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);
```

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*

$$h_n(x) = g_m(x)\delta(x) + \rho(x). \quad (2.36)$$

Dividendo allora un polinomio $p_n \in \mathbb{P}_n$ per $x - z$, grazie alla (2.36) 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 $m = n, n - 1, \dots, 1$:

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 (2.34)–(2.35) (posto $z = r_m$);
3. si pone $p_{m-1} = q_{m-1}$.

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

2.8.2 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 (2.34). Infatti, se q_{n-1} è il polinomio associato a p_n definito nella (2.35), 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

$$r_j^{(k+1)} = r_j^{(k)} - \frac{p_n(r_j^{(k)})}{p'_n(r_j^{(k)})} = r_j^{(k)} - \frac{p_n(r_j^{(k)})}{q_{n-1}(r_j^{(k)}; r_j^{(k)})} \quad (2.37)$$

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 2.7. 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 2.7. newtonhorner: il metodo di Newton-Hörner

```

function [radici,iter]=newtonhorner(a,x0,tol,kmax)
%NEWTONHORNER Metodo di Newton-Hörner
% [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-Hörner a partire dal dato
% iniziale X0. Il metodo si arresta per ogni radice
% al massimo dopo 100 iterazioni o dopo che il valore
% assoluto della differenza fra due iterate conse-
% cutive è 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

    if (niter==kmax && diff> tol)
        fprintf([' Il metodo non converge nel numero',...
            ' massimo di iterazioni\n']);
    end
    % Deflazione
    [pz,a] = horner(a,x);
    radici(k) = x;
    iter(k) = niter;
end

```

Osservazione 2.2 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 [QSSG14]). ■

Esempio 2.13 Richiamiamo il Programma 2.7 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 2.2 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 4) si trovano i seguenti risultati

```
a=[1 -7 15 -13 4]; format long;
[x,niter]=newtonhorner(a,0,1.e-15,100)

x =
    1.000006935337374
    0.999972452635761
    1.000020612232168
    3.999999999794697

niter =
    61
    100
    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. Più in generale, si può dimostrare (si veda [QSSG14, Cap. 6]) che il problema del calcolo di radici di una funzione f diventa mal condizionato (ovvero, molto sensibile a perturbazioni sui dati) qualora la derivata f' sia prossima a zero nelle radici. Per un esempio, si veda l’Esercizio 2.6. ■

2.9 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 a questo proposito il comando `fzero` (già introdotto nella Sezione 1.6.1) che utilizza il metodo di Dekker-Brent (si veda [QSS07, Cap. 6]). Nella sua forma più semplice a partire da un dato `x0`, `fzero(fun,x0)` calcola lo zero di una funzione `fun` più vicino a `x0`.

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

```
M=6000; v=1000; f=@(r) M-v*(1+r)/r*((1+r)^5-1);
x0=0.3;
[alpha,res,flag,info]=fzero(f,x0);
```

per trovare la radice **alpha=0.06140241153653** ed un residuo pari a **res=-1.8190e-12** dopo 7 iterazioni e con 29 valutazioni funzionali. La variabile **info** è una cosiddetta struttura, che si compone di 5 sotto-campi. In particolare nei campi **iterations** e **funcCount** della struttura **info** (ovvero in **info.iterations** ed in **info.funcCount**) vengono riportati rispettivamente il numero delle iterazioni svolte ed il numero delle valutazioni funzionali effettuate. 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 **alpha=0.06140241153653** con un residuo pari a **res=9.0949e-13** richiedendo tuttavia anche la conoscenza della derivata prima di *f* per un totale di 12 valutazioni funzionali.

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 [QSSG14]), ed il metodo di Bairstow ([RR01], pag. 371 e seguenti). Un altro approccio consiste nel caratterizzare gli zeri di un polinomio come gli autovalori di una particolare matrice, detta *companion matrix*, e nel calcolare quest'ultimi con tecniche opportune. Questo è l'approccio adottato dalla funzione **roots** di **MAT&OCT**, introdotta nella Sezione 1.6.2.

Nella Sezione 2.5 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.

fsolve L'istruzione **MAT&OCT**

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

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

Ad esempio, per il sistema non lineare (2.17) la corrispondente *function* **MAT&OCT**, che noi chiamiamo **systemnl**, 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;
```

Le istruzioni **MAT&OCT** per risolvere il sistema dato sono allora

```
x0 = [1 1];
alpha=fsolve(@system1,x0)

alpha =
    0.4761    -0.8794
```

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

Osservazione 2.3 I comandi `fzero` e `fsolve` hanno la stessa funzionalità in MATLAB e in Octave, tuttavia le loro interfacce differiscono leggermente per quanto concerne gli input opzionali. Consigliamo al lettore di consultare la documentazione attraverso l'`help` per entrambi i comandi sia in ambiente MATLAB sia in Octave. ■

MAT||OCT

2.10 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 di f 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 pari a 10^{-5} .

Esercizio 2.4 Si dimostri la diseguaglianza (2.6).

Esercizio 2.5 Nel Programma 2.1 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 (2.9) 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 $\beta \in [0, 2\pi/3]$ e con una tolleranza pari a 10^{-5} , 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 a $x^{(0)} = -0.1$ e a $x^{(0)} = 2\pi/3$.

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 $g = 9.8$ m/s² è l'accelerazione di gravità. Si calcoli α con il metodo di Newton, quando $v_0 = 10$ m/s e $h = 1$ 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 2.11. La lunghezza massima L di un'asta che possa passare da un estremo all'altro strisciando per terra è data da

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

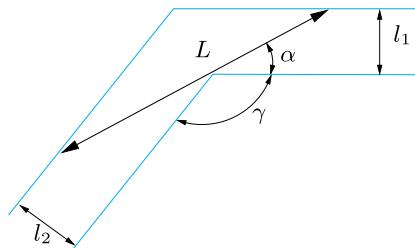


Figura 2.11. Problema dello scorrimento di un'asta in un corridoio

dove α è la soluzione della seguente equazione non lineare

$$l_2 \frac{\cos(\pi - \gamma - \alpha)}{\sin^2(\pi - \gamma - \alpha)} - l_1 \frac{\cos(\alpha)}{\sin^2(\alpha)} = 0. \quad (2.38)$$

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

Esercizio 2.15 Si considerino i metodi di Newton e delle secanti per approssimare le radici dell'equazione $(1-x)/(100x^2 - 100x + 26) = x/3$ nell'intervallo $[-2, 2]$. Localizzare le radici per via grafica e dire se esse sono semplici o multiple. Dopo aver posto la tolleranza per il test d'arresto pari a `tol=1.e-8` ed il numero massimo di iterazioni pari a `kmax=200`, si richiamino i metodi di Newton, prima con $x^{(0)} = 0.45$ e poi con $x^{(0)} = 0.55$, e quindi il metodo delle secanti, prima con $x^{(0)} = 0.45$, $x^{(1)} = 0.75$ e poi con $x^{(0)} = 0.5$, $x^{(1)} = 0.96$. I risultati numerici concordano con quanto afferma la teoria?

Esercizio 2.16 Si consideri il sistema di equazioni non lineari

$$\begin{cases} x^3 + y - 2x^2 - 2 = 0 \\ (x - 0.5)^2 - y^2 + x + \gamma = 0 \end{cases} \quad (2.39)$$

essendo γ un parametro reale assegnato. Per $\gamma = 2$ si localizzino le radici e le si classifichi in base alla molteplicità, quindi si calcolino tutte le radici del sistema dato con il metodo di Newton. Ripetere le stesse operazioni per $\gamma = 3.75$.

Esercizio 2.17 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 2.18 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 $x^{(0)}$, si definisce $x^{(k+1)}$ come

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

Se ne studi la convergenza a α .

Esercizio 2.19 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 2.20 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?

Esercizio 2.21 Si vogliono approssimare le intersezioni tra le funzioni $f_1(x) = \frac{1}{3} \log\left(\frac{x}{\pi}\right) + x^2$ e $f_2(x) = x^2 + x - 2$. Localizzare per via grafica le intersezioni tra le due curve, quindi approssimare i punti di intersezione con il metodo di Newton, scegliendo opportunamente il dato iniziale. In un secondo momento proporre una o più funzioni di punto fisso $\phi(x)$, i cui punti fissi coincidono con le intersezioni cercate, ed analizzarne la convergenza.

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. I problemi che seguono danno un'idea di questo approccio.

3.1 Alcuni problemi

Problema 3.1 (Climatologia) La temperatura dell'aria in prossimità del suolo dipende dalla concentrazione K dell'acido carbonico (H_2CO_3). 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 $\delta_K = \theta_K - \theta_{\bar{K}}$ della temperatura media che si avrebbero nel globo rispetto alla temperatura media corrispondente ad un valore di riferimento \bar{K} di K . Qui \bar{K} rappresenta il valore misurato nel 1896 ed è normalizzato a 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 e per altri valori di K (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 Zurigo su due anni. La curva è stata ottenuta semplicemente congiungendo con un segmento i prezzi fissati ogni giorno alla chiusura del mercato. Questa semplice

Tabella 3.1. Variazioni della temperatura media annua del globo terrestre per quattro diversi valori della concentrazione K di acido carbonico a diverse latitudini

Latitudine	δ_K			
	$K = 0.67$	$K = 1.5$	$K = 2.0$	$K = 3.0$
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

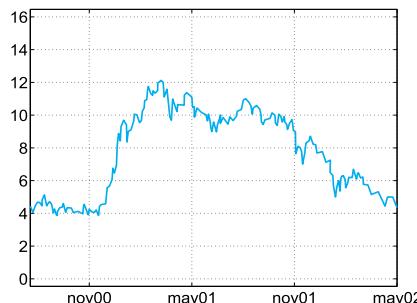


Figura 3.1. Andamento del prezzo di un’azione nell’arco di due anni

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 nella Sezione 3.6, 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 3.12). ■

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

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

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

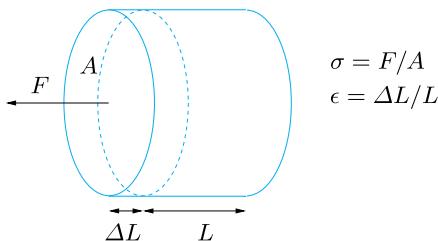


Figura 3.2. Una rappresentazione schematica di un disco intervertebrale

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 (MPa = 100 N/cm²). Si veda per la risoluzione l’Esempio 3.13. ■

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. Nell’Esempio 3.10 risolveremo questo problema con l’ausilio delle funzioni *spline*. ■

3.2 Approssimazione con i polinomi di Taylor

Come è noto, una funzione f può essere approssimata in un intervallo dal suo polinomio di Taylor di un certo grado n , introdotto nella Sezione 1.6.3. 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 nei punti

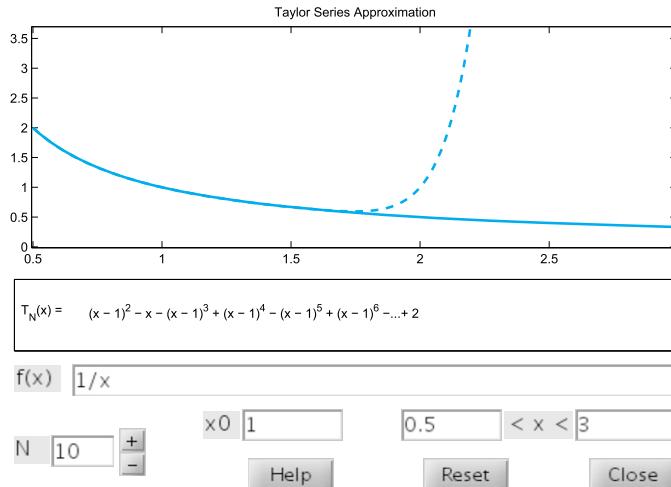


Figura 3.3. Confronto tra la funzione $f(x) = 1/x$ (in linea continua) ed il suo polinomio di Taylor di grado 10 riferito al punto $a = 1$ (in linea tratteggiata)

x relativamente lontani 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 costruito attorno al punto $x_0 = 1$. Questa figura mostra anche l’interfaccia grafica del programma MATLAB **taylortool** (disponibile nel *symbolic toolbox* di MATLAB) che consente di calcolare il polinomio di Taylor di grado arbitrario di una data funzione f . Come si vede più ci si allontana da $x_0 = 1$ 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.

MAT||OCT **Osservazione 3.1** **taylortool** non è disponibile in Octave, tuttavia dopo aver installato il pacchetto *symbolic* (si veda l’Osservazione 1.5), possiamo visualizzare il grafico di Figura 3.3 con i seguenti comandi:

```
pkg load symbolic
syms x
f=1/x; a=1; n=10;
Tn=taylor(f,'expansionPoint',a,'Order',n+1)
tn=function_handle(Tn);
ff=function_handle(f);
figure(1); clf
fplot(ff,[1,3], 'c'); hold on
fplot(tn,[1,3], 'c--')
axis([1,3,0,2])
```

Il comando `function_handle` di Octave converte una *symbolic function* in una *anonymous function* al fine, ad esempio, di poterne disegnare il grafico con il comando `fplot`. In MATLAB, invece di `function_handle`, possiamo utilizzare il comando `matlabFunction`.

■ `matlab Function`

3.3 Interpolazione

Come abbiamo potuto notare dai problemi 3.1, 3.2 e 3.3, in molte applicazioni concrete si conosce una funzione solo attraverso i suoi valori in 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 variazione di 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* dell'insieme di dati $\{y_i\}$ 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.

3.3.1 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 , procediamo come segue. Per ogni k compreso tra 0 e n costruiamo un polinomio di grado n , denotato $\varphi_k(x)$, che interpola i valori y_i tutti nulli fuorché quello per $i = k$ per il quale $y_k = 1$, ovvero

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

dove δ_{jk} è il simbolo di Kronecker (si veda la Figura 3.4).

Le funzioni φ_k hanno la seguente espressione

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

Mettiamoci ora nel caso generale in cui $\{y_i, i = 0, \dots, n\}$ sia un insieme di valori arbitrari. Per il principio di sovrapposizione degli effetti abbiamo

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

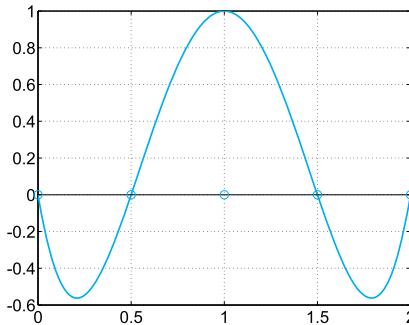


Figura 3.4. Il polinomio $\varphi_2 \in \mathbb{P}_4$ associato ai 5 nodi equispaziati in $[0, 2]$

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 loro ruolo peculiare, le funzioni φ_k sono dette *polinomi caratteristici di Lagrange* e la (3.4) è nota come *forma di Lagrange* del polinomio interpolatore.

Possiamo memorizzare le coordinate degli $n+1$ punti $\{(x_i, y_i)\}$ 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}, \dots e `c(n+1)` il valore di $\Pi_n(0)$. (Maggiori dettagli su questo comando sono contenuti nella Sezione 3.6.)

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 $y_i = f(x_i)$ e l'espressione della funzione f sia nota in forma esplicita, denotiamo con $\Pi_n f$ il *polinomio di interpolazione di Lagrange* di f di f . Dopo aver memorizzato i nodi x_i nel vettore `x`, possiamo costruire il vettore `y` mediante l'istruzione `y=f(x)`.

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
```

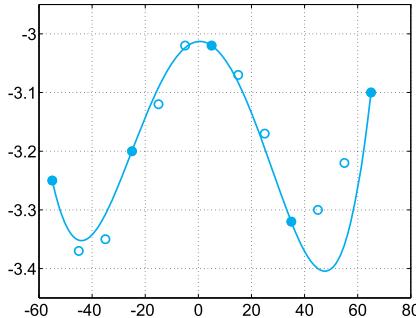


Figura 3.5. Il polinomio interpolatore di grado 4 introdotto nell’Esempio 3.1

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');grid on;
```

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(end)` consente di accedere direttamente all’ultima componente del vettore `x`, senza bisogno di conoscerne la lunghezza. In Figura 3.5 i cerchietti pieni corrispondono ai dati utilizzati per costruire il polinomio di interpolazione, mentre quelli vuoti corrispondono ai dati che non sono stati utilizzati. 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 $\forall x \in I$ $\exists \xi_x \in I$ tale che*

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

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

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,

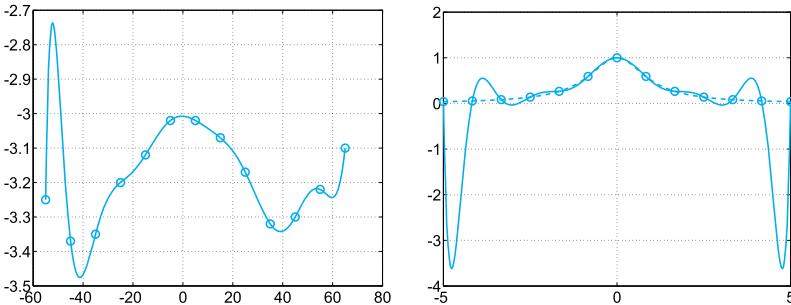


Figura 3.6. Due esemplificazioni del fenomeno di Runge: $\Pi_{12}f$ calcolato per l'insieme di dati della Tabella 3.1, colonna $K = 0.67$ (a sinistra); $\Pi_{12}f$ (in linea continua) calcolato su 13 nodi equispaziati nel caso della funzione di Runge $f(x) = 1/(1+x^2)$ (in linea tratteggiata) (a destra)

$\forall x \in (x_0, x_n)$ 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 (a sinistra), il cui comportamento, nelle vicinanze dell'estremo sinistro 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 MATLAB:

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

I massimi dei valori assoluti delle funzioni $f^{(n)}$, $n = 1, \dots, 21$, sono stati memorizzati nel vettore M . Si noti che il comando `matlabFunction` di MATLAB (sostituibile in Octave con `function_handle`, come descritto nell'Osservazione 3.1) converte la variabile simbolica `df` in un *function handle* che poi viene passato alla 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 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+1]);
end
r([3,9,15,21])
```

```
ans =
    1.1574e+01    5.1814e-02    1.3739e-05    4.7247e-10
```

poly 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

```
5.4034e+01    1.6801e+04    1.6706e+07    2.2877e+10
```

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*. ■

Oltre alla (3.7) si può anche dimostrare che vale la seguente disegualanza

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

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 . Il polinomio $(\Pi_n f)'$

può essere calcolato tramite il comando `MAT&OCT [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 la Sezione 1.6.2).

3.3.2 Stabilità dell'interpolazione polinomiale

Cosa succede al polinomio di interpolazione se, anziché partire da dati esatti $f(x_i)$ relativi ai nodi x_i , con $i = 0, \dots, n$, in un intervallo I , si considera una loro approssimazione, diciamo $\hat{f}(x_i)$? La perturbazione $f(x_i) - \hat{f}(x_i)$ potrebbe essere dovuta ad esempio all'effetto degli errori di arrotondamento oppure essere causata da un errore nella misurazione dei dati stessi.

Sia $\Pi_n \hat{f}$ il polinomio interpolatore corrispondente ai valori $\hat{f}(x_i)$. Indicando con \mathbf{x} il vettore le cui componenti sono i nodi di interpolazione $\{x_i\}$, si ha

$$\begin{aligned} \max_{x \in I} |\Pi_n f(x) - \Pi_n \hat{f}(x)| &= \max_{x \in I} \left| \sum_{i=0}^n (f(x_i) - \hat{f}(x_i)) \varphi_i(x) \right| \\ &\leq A_n(\mathbf{x}) \max_{0 \leq i \leq n} |f(x_i) - \hat{f}(x_i)| \end{aligned} \quad (3.8)$$

dove

$$A_n(\mathbf{x}) = \max_{x \in I} \sum_{i=0}^n |\varphi_i(x)| \quad (3.9)$$

indica la cosiddetta *costante di Lebesgue* che dipende dalla scelta dei nodi di interpolazione. Di conseguenza, a piccole perturbazioni sui dati corrisponderanno piccole variazioni sul polinomio interpolatore purché la costante di Lebesgue sia piccola. Quest'ultima assume il significato di *numero di condizionamento* del problema dell'interpolazione. Nel caso dell'interpolazione polinomiale di Lagrange su nodi equispaziati, si trova

$$A_n(\mathbf{x}) \simeq \frac{2^{n+1}}{en(\log n + \gamma)}, \quad (3.10)$$

dove $e \simeq 2.71834$ è il numero di Nepero e $\gamma \simeq 0.57721$ rappresenta la costante di Eulero (si veda [Hes98] e [Nat65]).

Ciò suggerisce che per n grande questo tipo di interpolazione potrebbe essere instabile, come si evince dal seguente esempio. (Si veda anche l'Esercizio 3.8.)

Esempio 3.3 Sull'intervallo $[-1, 1]$ interpoliamo la funzione $f(x) = \sin(2\pi x)$ su 22 nodi equispaziati x_i , $i = 0, \dots, 21$. Generiamo un insieme di valori $\hat{f}(x_i)$

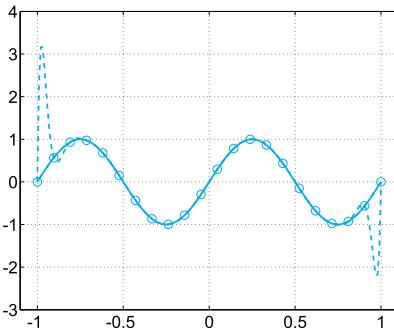


Figura 3.7. Esempio 3.3. Effetti dell'instabilità nell'interpolazione semplice di Lagrange. $\Pi_{21}f$ è relativo ai dati imperturbati (*in linea continua*), $\Pi_{21}\hat{f}$ è relativo ai dati perturbati (*in linea tratteggiata*)

ottenuti perturbando in maniera casuale i valori $f(x_i)$, in modo che

$$\max_{i=0,\dots,21} |f(x_i) - \hat{f}(x_i)| \simeq 9.5 \cdot 10^{-4}.$$

In Figura 3.7 vengono confrontati i due polinomi di interpolazione $\Pi_{21}f$ e $\Pi_{21}\hat{f}$: come si vede agli estremi dell'intervallo di interpolazione la differenza è molto più grande della perturbazione operata, essendo $\max_{x \in I} |\Pi_n f(x) - \Pi_n \hat{f}(x)| \simeq 3.1342$. Si noti che in questo caso la costante di Lebesgue è molto grande, si ha $\Lambda_{21}(\mathbf{x}) \simeq 20454$. ■



Si vedano gli Esercizi 3.1–3.4.

3.3.3 Interpolazione rispetto ai nodi 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-Gauss-Lobatto*

$$x_i = \frac{a+b}{2} + \frac{b-a}{2} \hat{x}_i, \quad \text{dove } \hat{x}_i = -\cos(\pi i/n), \quad i = 0, \dots, n \quad (3.11)$$

Naturalmente $x_i = \hat{x}_i$, $i = 0, \dots, n$ 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-Gauss-Lobatto, 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 3.8 a destra).

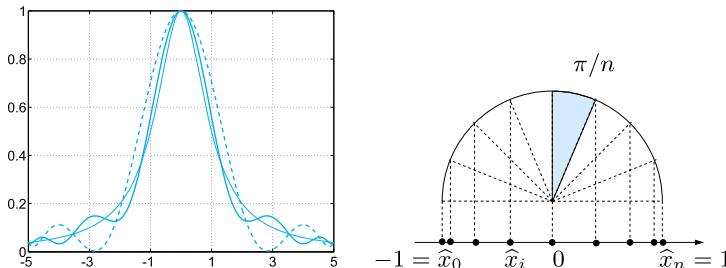


Figura 3.8. La funzione di Runge $f(x) = 1/(1+x^2)$ (in linea continua sottile) a confronto con i polinomi interpolatori sui nodi di Chebyshev-Gauss-Lobatto di grado 8 (linea tratteggiata) e 12 (linea continua più marcata) (a sinistra). Si noti come ora, al crescere del grado, le oscillazioni si smorzino e l'approssimazione divenga sempre più accurata. La distribuzione dei nodi di Chebyshev-Gauss-Lobatto nell'intervallo $[-1, 1]$ (a destra)

Un'altra distribuzione di nodi per la quale si hanno le stesse proprietà di convergenza è data dai *nodi di Chebyshev-Gauss*

$$x_i = \frac{a+b}{2} - \frac{b-a}{2} \cos\left(\frac{2i+1}{n+1} \frac{\pi}{2}\right), \quad i = 0, \dots, n \quad (3.12)$$

Esempio 3.4 Riprendiamo la funzione di Runge ed interpoliamola nei nodi di Chebyshev-Gauss-Lobatto. Per generarli possiamo usare i seguenti comandi

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

dove $n+1$ è il numero di nodi, mentre a e b sono gli estremi dell'intervallo di interpolazione (nel nostro caso porremo $a=-5$ e $b=5$). Quindi, il polinomio interpolatore si genererà con le seguenti istruzioni

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

Valutiamo a questo punto il valore assoluto delle differenze fra f ed il suo polinomio interpolatore rispetto ai nodi di Chebyshev-Gauss-Lobatto in 1000 punti equispaziati nell'intervallo $[-5, 5]$ e prendiamone il massimo

```
x1 = linspace(-5,5,1000); p=polyval(c,x1);
f1 = f(x1); err = max(abs(p-f1));
```

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

Tabella 3.3. L'errore di interpolazione per la funzione di Runge $f(x) = 1/(1+x^2)$ qualora si utilizzino i nodi di Chebyshev-Gauss-Lobatto (3.11)

n	5	10	20	40
E_n	0.6386	0.1322	0.0177	0.0003

È interessante osservare che, qualora si considerino i nodi di Chebyshev-Gauss-Lobatto (3.11), la costante di Lebesgue si può maggiorare come segue [Hes98]

$$\Lambda_n(\mathbf{x}) < \frac{2}{\pi} \left(\log n + \gamma + \log \frac{8}{\pi} \right) + \frac{\pi}{72 n^2}, \quad (3.13)$$

mentre qualora si considerino i nodi di Chebyshev-Gauss (3.12) si ha

$$\Lambda_n(\mathbf{x}) < \frac{2}{\pi} \left(\log(n+1) + \gamma + \log \frac{8}{\pi} \right) + \frac{\pi}{72(n+1)^2}, \quad (3.14)$$

dove $\gamma \simeq 0.57721$ denota sempre la costante di Eulero.

Confrontando le maggiorazioni (3.13) e (3.14) con la stima (3.10) valida per nodi equispaziati, possiamo dedurre che l'interpolazione su nodi di Chebyshev è molto meno sensibile alla propagazione degli errori di arrotondamento di quanto non lo sia l'interpolazione su nodi equispaziati.

Esempio 3.5 Riprendiamo i dati dell'esempio 3.3 operando stavolta l'interpolazione sui nodi di Chebyshev (3.11) e (3.12). Partendo dalle stesse perturbazioni sui dati utilizzate per l'esempio 3.3 (inferiori a $9.5 \cdot 10^{-4}$), con $n = 21$ otteniamo $\max_{x \in I} |\Pi_n f(x) - \Pi_n \hat{f}(x)| \simeq 1.0977 \cdot 10^{-3}$ per i nodi (3.11) e $\max_{x \in I} |\Pi_n f(x) - \Pi_n \hat{f}(x)| \simeq 1.1052 \cdot 10^{-3}$ per i nodi (3.12). Questo è in accordo con le stime (3.13) e (3.14) le quali, per $n = 21$, fornirebbero rispettivamente $\Lambda_n(\mathbf{x}) \lesssim 2.9008$ e $\Lambda_n(\mathbf{x}) \lesssim 2.9304$. ■

3.3.4 Formula di interpolazione baricentrica

Il polinomio interpolatore di Lagrange $\Pi_n(x)$ introdotto nella Proposizione 3.1 può essere calcolato anche con la seguente *formula baricentrica* [BT04]

$$\Pi_n(x) = \frac{\sum_{k=0}^n \frac{w_k}{x-x_k} y_k}{\sum_{k=0}^n \frac{w_k}{x-x_k}} \quad (3.15)$$

dove i coefficienti

$$w_k = \left(\prod_{\substack{j=0 \\ j \neq k}}^n (x_k - x_j) \right)^{-1}, \quad k = 0, \dots, n, \quad (3.16)$$

sono chiamati *pesi baricentrici*.

Per ricavare (3.15) da (3.4) riscriviamo i polinomi caratteristici (3.3) come

$$\varphi_k(x) = \prod_{j=0}^n \frac{x - x_j}{x_k - x_j} = \underbrace{\left(\prod_{j=0}^n (x - x_j) \right)}_{\ell(x)} \frac{w_k}{x - x_k},$$

cosicché

$$\Pi_n(x) = \ell(x) \sum_{k=0}^n \frac{w_k}{x - x_k} y_k. \quad (3.17)$$

Notando che

$$\ell(x) \sum_{k=0}^n \frac{w_k}{x - x_k} = 1,$$

(questo risultato segue da (3.17) ponendo $y_k = 1$ per $k = 0, \dots, n$ e notando che in questo caso $\Pi_n(x) \equiv 1$) si ha

$$\Pi_n(x) = \frac{\ell(x) \sum_{k=0}^n \frac{w_k}{x - x_k} y_k}{\ell(x) \sum_{k=0}^n \frac{w_k}{x - x_k}},$$

ovvero (3.15).

Prendiamo un insieme di nodi equispaziati e una funzione $f(x)$ tale che in aritmetica esatta si abbia $\max_{x \in [a,b]} |f(x) - \Pi_n f(x)| \rightarrow 0$ per $n \rightarrow \infty$. Consideriamo il polinomio interpolatore di Lagrange espresso nelle tre forme seguenti: quella di Lagrange (3.4), quella baricentrica (3.15), infine nello sviluppo rispetto alla base dei monomi

$$\Pi_n f(x) = \sum_{k=0}^n c_{k+1} x^{n-k}. \quad (3.18)$$

In tutti e tre i casi, l'errore di interpolazione decresce finché n è minore di un certo valore e poi comincia a crescere per effetto degli errori di arrotondamento dovuti all'aritmetica *floating-point*. Esso diverge per la forma di Lagrange (3.4) e per quella dei monomi (3.18), mentre rimane limitato (all'ordine di grandezza di 1) per la forma baricentrica (3.15). Si veda la Figura 3.9, a sinistra, in cui abbiamo considerato $f(x) = \sin(x)$ sull'intervallo $[-1, 1]$.

Qui i nodi sono uniformemente distribuiti e i pesi della forma baricentrica sono $w_k = (-1)^k \binom{n}{k}$. Ciò può potenzialmente generare forti oscillazioni del polinomio interpolatore vicino al bordo dell'intervallo contenente i nodi di interpolazione. Ad ogni modo, questo fenomeno, noto come fenomeno di Runge (si veda l'Esempio 3.2) non è da imputarsi alla formula baricentrica, anzi è intrinseco all'interpolazione di Lagrange su nodi equispaziati, in quanto è legato al cattivo condizionamento del problema dell'interpolazione quando n diventa molto grande (si veda la Sezione 3.3.2 dove è riportata la costante di Lebesgue Λ_n per nodi equispaziati), succede che a piccole variazioni sui dati possono corrispondere forti errori nella costruzione del polinomio interpolatore.

Al contrario, quando utilizziamo i nodi di Chebyshev per costruire il polinomio di Lagrange, il problema dell'interpolazione è ben condizionato (si veda la Sezione 3.3.3) e i pesi della formula baricentrica sono limitati. Ora, mentre le forme di Lagrange (3.4) e quella dei monomi

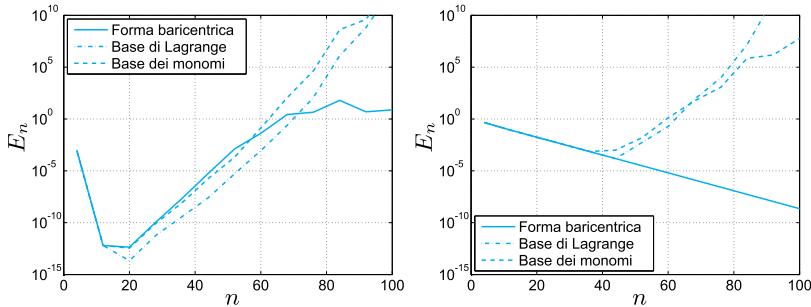


Figura 3.9. Errori di interpolazione per la funzione $f(x) = \sin(x)$ usando nodi equispaziati in $[-1, 1]$ (a sinistra). Errori di interpolazione per la funzione $f(x) = 1/(1+x^2)$ usando nodi di Chebyshev in $[-5, 5]$ (a destra)

(3.18) risultano instabili rispetto alla propagazione degli errori di arrotondamento, la forma baricentrica è stabile anche per valori di n molto grandi (si veda la Fig. 3.9, a destra), a patto che siano osservati due piccoli accorgimenti (si veda [BT04, Hig04]).

Il primo riguarda l'*underflow* e l'*overflow*: quando $n \rightarrow \infty$, la scala dei pesi w_j (3.16) crescerà o decrescerà in modo proporzionale a $(4/(b-a))^n$, essendo $[a, b]$ l'intervallo in cui cerchiamo il polinomio interpolatore. *Underflow* e *overflow* possono essere evitati moltiplicando ogni fattore $(x_k - x_j)$ in (3.16) per $4/(b-a)$.

Il secondo accorgimento riguarda la valutazione di $\Pi_n(x)$ quando $x = x_k$. Una implementazione non accorta di (3.15) produrrebbe un output di tipo *Nan*: tuttavia in questo caso è sufficiente sostituire il calcolo (3.15) di $\Pi_n(x_k)$ con il valore assegnato y_k . Quando x è molto vicino a x_k la formula baricentrica risulta comunque stabile, come è mostrato in [Hen79].

Il Programma 3.1 implementa la formula baricentrica (3.15) tenendo conto degli accorgimenti sopra descritti. Le variabili in input \mathbf{x} , \mathbf{y} , e $\mathbf{x1}$ hanno lo stesso significato delle omonime variabili utilizzate dalla *function polyfit* di MATLAB. La variabile di output $\mathbf{y1}$ contiene i valori di $\Pi_n(x)$ nei nodi memorizzati in $\mathbf{x1}$.

Programma 3.1. `barycentric`: interpolazione baricentrica

```
function [y1]=barycentric(x,y,x1)
%BARYCENTRIC Calcola l'interpolatore di Lagrange nella
% forma baricentrica.
% Y1=BARYCENTRIC(X,Y,X1) calcola il valore del poli-
% nomio interpolatore di Lagrange che interpola i dati
% (X,Y) nei punti di ascissa memorizzati in X1,
% utilizzando la formula baricentrica

np=length(x);
a=min(x); b=max(x);
```

```
w=ones(np,1);
C=4/(b-a);
for j=1:np
    for k=1:j-1
        w(j)=w(j)*(x(j)-x(k))*C;
    end
    for k=j+1:np
        w(j)=w(j)*(x(j)-x(k))*C;
    end
end
w=1./w;
num=zeros(size(x1));den=num; exa=num;
for j=1:np
    xdiff=x1-x(j); wx=w(j)./xdiff;
    den=den+wx; num=num+wx*y(j);
    exa(xdiff==0)=j;
end
y1=num./den;
for i=1:length(x1)
    if exa(i)>0, y1(i)=y(exa(i)); end
end
```

Esempio 3.6 Interpoliamo la funzione $f(x) = 1/(1+x^2)$ sull'intervallo $[-5, 5]$ negli $(n + 1)$ nodi di Chebyshev-Gauss-Lobatto (3.11). Richiamiamo il Programma 3.1 per la formula baricentrica (3.15), quindi un semplice programma che calcola $\Pi_n f$ utilizzando la forma di Lagrange (3.4), e infine il comando `polyfit` di MATLAB che calcola i coefficienti di $\Pi_n f$ rispetto alla base dei monomi (3.18).

In accordo con la teoria, l'errore di interpolazione $E_n = \max_{[-5,5]} |f(x) - \Pi_n f(x)|$ dovrebbe convergere esponenzialmente a zero per $n \rightarrow \infty$, grazie alle buone proprietà dei nodi di Chebyshev. In pratica invece, come possiamo vedere in Figura 3.9, quando $\Pi_n f(x)$ è calcolato con la formula di Lagrange (3.4) o l'espansione (3.18), l'errore E_n decresce esponenzialmente fino a $n \simeq 40$, ma poi esso comincia a crescere a causa della propagazione degli errori di arrotondamento (per $n \geq 20$ la function MATLAB `polyfit` produce un messaggio di warning che evidenzia il mal condizionamento della matrice di Vandermonde associata alla base dei monomi, si veda l'Esempio 5.4).

Al contrario, quando si usa la formula baricentrica (3.15), l'errore continua a decrescere esponenzialmente fino alla precisione di macchina. Abbiamo considerato $n = 4 : 8 : 128$. ■

3.3.5 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 interpoli negli $n + 1$ nodi equispaziati $x_j = 2\pi j / (n + 1)$, $j = 0, \dots, n$, ovvero tale che

$$\tilde{f}(x_j) = f(x_j), \quad \text{per } j = 0, \dots, n. \quad (3.19)$$

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

Consideriamo dapprima il caso in cui n è pari. In particolare cerchiamo una funzione

$$\tilde{f}(x) = \frac{a_0}{2} + \sum_{k=1}^M [a_k \cos(kx) + b_k \sin(kx)], \quad (3.20)$$

con $M = n/2$, i cui coefficienti complessi a_k , per $k = 0, \dots, M$ e b_k , per $k = 1, \dots, M$ sono incogniti. Utilizzando la formula di Eulero $e^{ikx} = \cos(kx) + i \sin(kx)$, il polinomio trigonometrico (3.20) può essere riscritto come

$$\tilde{f}(x) = \sum_{k=-M}^M c_k e^{ikx}, \quad (3.21)$$

dove i è l'unità immaginaria e i coefficienti c_k , per $k = 0, \dots, M$, sono legati ai coefficienti a_k e b_k dalle relazioni

$$a_k = c_k + c_{-k}, \quad b_k = i(c_k - c_{-k}). \quad (3.22)$$

Infatti, grazie alle proprietà di simmetria delle funzioni seno e coseno, si ha:

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

Nel caso in cui n sia dispari, il polinomio trigonometrico \tilde{f} può essere definito come

$$\tilde{f}(x) = \sum_{k=-(M+1)}^{M+1} c_k e^{ikx}, \quad (3.23)$$

con $M = (n-1)/2$. Si osservi che i coefficienti incogniti in (3.23) sono $n+2$, mentre le condizioni di interpolazione (3.19) sono $n+1$. Il problema di determinare il polinomio interpolatore \tilde{f} diventa ben posto solo dopo aver ridotto a $n+1$ il numero delle incognite. Una possibile scelta, che coincide con quanto fa la *function interpft* di **MATLAB**, consiste nell'imporre che $c_{-(M+1)} = c_{(M+1)}$.

Anche per n dispari possiamo scrivere \tilde{f} come somma di seni e coseni, ottenendo una formula analoga a (3.20) in cui l'indice k della sommatoria va ora da 1 a $M+1$. I coefficienti c_k di (3.23) sono legati ai coefficienti a_k e b_k ancora mediante le formule (3.22), questa volta per $k = 0, \dots, M+1$. In particolare, poiché $c_{-(M+1)} = c_{(M+1)}$, otteniamo $a_{(M+1)} = 2c_{(M+1)}$ e $b_{(M+1)} = 0$.

Per poter trattare contemporaneamente i casi con n pari ed n dispari, introduciamo un parametro μ e poniamo $\mu = 0$ se n è pari, e $\mu = 1$, se n è dispari. Il polinomio interpolatore può essere riscritto nella forma più generale come

$$\tilde{f}(x) = \sum_{k=-(M+\mu)}^{M+\mu} c_k e^{ikx} = \sum_{k=-M}^M c_k e^{ikx} + 2\mu c_{(M+1)} \cos((M+1)x),$$

dove $M = (n-\mu)/2$ e $c_{(M+1)} = c_{-(M+1)}$ è trascurato per n pari ($\mu = 0$).

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 per $j = 0, \dots, n$,

$$\sum_{k=-M}^M c_k e^{ikjh} + 2\mu c_{(M+1)} \cos((M+1)jh) = f(x_j). \quad (3.24)$$

Per calcolare i coefficienti $\{c_k, k = -M, \dots, M+\mu\}$ moltiplichiamo ambo i membri della (3.24) per $e^{-imx_j} = e^{-imjh}$ con m intero fra $-M$ and $M+\mu$, e poi sommiamo su j

$$\begin{aligned} & \sum_{j=0}^n \sum_{k=-M}^M c_k e^{ikjh} e^{-imjh} + 2\mu c_{(M+1)} \sum_{j=0}^n \cos((M+1)jh) e^{-imjh} \\ &= \sum_{j=0}^n f(x_j) e^{-imjh}. \end{aligned} \quad (3.25)$$

Anzitutto osserviamo che quando n è dispari si ha $\sin((M+1)jh) = \sin(\pi j)$ e

$$\cos((M+1)jh) = e^{i(M+1)jh}.$$

Quindi consideriamo l'identità

$$\sum_{j=0}^n e^{ijh(k-m)} = (n+1)\delta_{km}, \quad k, m = -M, \dots, M;$$

per $k \neq m$ è una conseguenza della proprietà

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

e del fatto che il numeratore a destra è nullo in quanto

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

Utilizzando in (3.25) l'identità appena dimostrata otteniamo

$$\sum_{j=0}^n \cos((M+1)jh) e^{-imjh} = (n+1)\delta_{(M+1)m}$$

e la seguente un'espressione esplicita per i coefficienti di \tilde{f} :

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

$$c_{(M+1)} = c_{-(M+1)} = \frac{1}{2(n+1)} \sum_{j=0}^n (-1)^j f(x_j), \quad \text{per } n \text{ dispari}$$

(3.26)

Qualora f sia una funzione a valori reali, dalla (3.26) deduciamo che $c_{(M+1)} = c_{-(M+1)}$ sono reali e che $c_{-k} = \overline{c_k}$ per $k = -M, \dots, M$ (quest'ultima identità segue da $e^{ikjh} = \overline{e^{-ikjh}}$). Grazie a (3.22) abbiamo $a_k, b_k \in \mathbb{R}$ (per $k = 0, \dots, M + \mu$), e quindi anche \tilde{f} è una funzione a valori reali.

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* (FFT), implementata in **MAT&OCT** nel **fft** programma **fft** (si veda l'Esempio 3.7). 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 **ifft** programma **ifft**.

Esempio 3.7 Consideriamo la funzione $f(x) = x(x - 2\pi)e^{-x}$ per $x \in [0, 2\pi]$. Vogliamo ottenere i coefficienti del suo interpolatore trigonometrico con $n = 9$ (da cui $M = 4$). Per usare il comando **fft** di **MAT&OCT**, campioniamo la funzione nei nodi $x_j = j\pi/5$ per $j = 0, \dots, 9$ con i seguenti comandi (ricordiamo che $.*$ è il prodotto fra vettori, componente per componente):

```
n=9; x=2*pi/(n+1)*[0:n]; y=x.*(x-2*pi).*exp(-x);
```

A questo punto, calcoliamo il vettore dei coefficienti di Fourier con la FFT tramite i comandi:

```
Y=fft(y);
C=fftshift(Y)/(n+1)
```

```
C =
Columns 1 through 2
 0.0870      0.0926 - 0.0214i
```

```

Columns 3 through 4
 0.1098 - 0.0601i  0.1268 - 0.1621i
Columns 5 through 6
-0.0467 - 0.4200i -0.6520
Columns 7 through 8
-0.0467 + 0.4200i  0.1268 + 0.1621i
Columns 9 through 10
 0.1098 + 0.0601i  0.0926 + 0.0214i

```

Le componenti del vettore \mathbf{Y} sono legate ai coefficienti c_k definiti in (3.26) mediante la seguente relazione: $\mathbf{Y} = (n+1)[c_0, \dots, c_M, c_{-(M+\mu)}, \dots, c_{-1}]$ e, nel caso in cui n sia dispari, il coefficiente $c_{(M+1)}$ (che, ricordiamo, coincide con $c_{-(M+1)}$) non viene riportato tra le componenti del vettore \mathbf{Y} . Il comando `fftshift` riordina le componenti del vettore in input, cosicché `fftshift C=[c_{-(M+\mu)}, \dots, c_{-1}, c_0, \dots, c_M]`.

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. ■

Il comando `interpft` calcola l'interpolatore trigonometrico di un insieme di dati reali. 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 reali 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 3.7 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 $t_i = 2i\pi/100$, $i = 0, \dots, 99$, si possono ottenere nel modo seguente (si veda la Figura 3.10)

```

n=9; x=2*pi/(n+1)*[0:n]; 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.

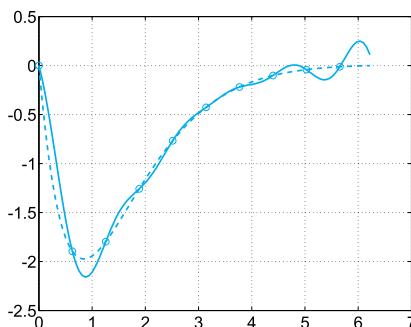


Figura 3.10. La funzione $f(x) = x(x - 2\pi)e^{-x}$ (in linea tratteggiata) ed il corrispondente interpolatore trigonometrico $\tilde{f}(x)$ (in linea continua) relativo a 10 nodi equispaziati

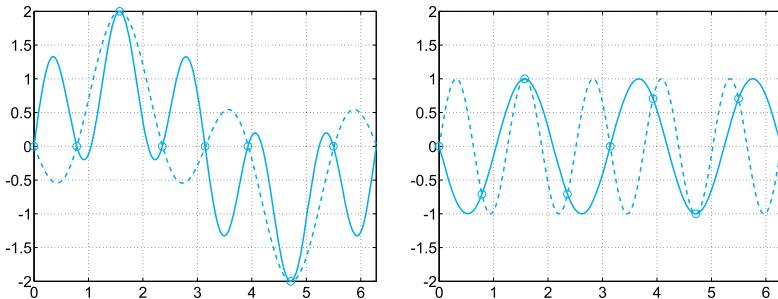


Figura 3.11. Gli effetti dell’*aliasing*. Confronto tra la funzione $f(x) = \sin(x) + \sin(5x)$ (in linea continua) ed il suo interpolatore trigonometrico $\tilde{f}(x)$ con $M = 3$ (linea tratteggiata) (a sinistra). Le funzioni $\sin(5x)$ (in linea tratteggiata) e $-\sin(3x)$ (in linea continua) (a destra) assumono gli stessi valori nei nodi di interpolazione. Questo spiega la forte perdita di accuratezza mostrata nella figura di sinistra

Esempio 3.8 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 ottenuto con `MAT&OCT` viene riportato in Figura 3.11 a sinistra. 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 3.11 a destra). La funzione che viene approssimata è quindi in realtà $F(x) = f_1(x) + f_3(x)$ e non $f(x)$ (in effetti, il grafico in tratteggio della Figura 3.11 a sinistra 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 delle ruote di una bicicletta 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 genera 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;
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 quelle 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 efficiente per il calcolo dei coefficienti di Fourier dell'interpolatore trigonometrico a partire dai suoi valori nodali; la trasformazione inversa è realizzata dall'algoritmo IFFT.

3.4 Interpolazione lineare composita

Se f è una funzione di cui si conosce l'espressione analitica, l'interpolazione rispetto ai nodi 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) o f è poco regolare, si può ricorrere ad un metodo di interpolazione differente, detto interpolazione composita lineare.

Precisamente, data una distribuzione di nodi $x_0 < x_1 < \dots < x_n$, 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 $(x_i, f(x_i))$ e $(x_{i+1}, f(x_{i+1}))$ (si veda la Figura 3.12). Tale funzione, denotata con $\Pi_1^H f$, è detta *polinomio interpolatore composito lineare* di f ed assume la seguente espressione

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

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

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

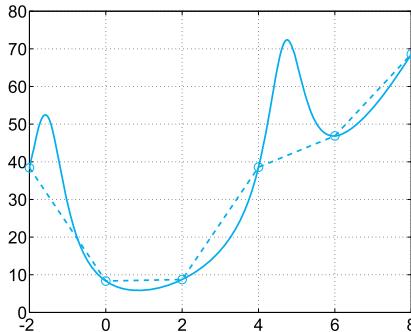


Figura 3.12. La funzione $f(x) = x^2 + 10/(\sin(x) + 1.2)$ (in linea continua) ed il suo interpolatore lineare composito $\Pi_1^H f$ (in linea tratteggiata)

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.

interp1 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 $i = 1, \dots, n+1$, osservando che `z` può assumere dimensione arbitraria. Quando i nodi di interpolazione sono dati in ordine crescente (cioè $x(i+1) > x(i)$, per $i=1, \dots, n$) allora si può usare la versione **interp1q** computazionalmente più economica `interp1q` (in inglese `q` sta per *quickly*). La function `interp1q` è più veloce di `interp1` qualora i nodi `x(i)` non siano equispaziati in quanto non esegue controllo sui dati in input. Tuttavia si rammenta che le variabili di input `x`, `y` e `z` devono essere, per questa function, vettori colonna.

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

3.5 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 $f - \Pi_2^H f$ valutato nella norma del massimo decresce ora come H^3 quando H tende a zero. Tuttavia la principale controindicazione dell'interpolazione composita (lineare o di grado $k \geq 1$) è che la funzione $\Pi_k^H f$ è solo globalmente 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 $j = i, i + 1$ (in particolare s_3 sarà continua su tutto l'intervallo);
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} ;
- $2(n - 1)$ equazioni addizionali sono ottenute imponendo anche la continuità della derivata prima e della derivata seconda nei nodi interni.

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

$$s_3''(x_0) = 0, \quad s_3''(x_n) = 0. \quad (3.27)$$

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

Scegliendo opportunamente le incognite per rappresentare s_3 (si veda [QSSG14, Sez. 7.6]), si può determinare s_3 risolvendo un sistema lineare quadrato 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 nella Sezione 5.6) attraverso il Programma 3.2 i cui parametri d'ingresso obbligatori sono i vettori \mathbf{x} e \mathbf{y} dei dati da interpolare ed il vettore \mathbf{zi} delle ascisse nelle quali si vuole che venga valutata s_3 . La scelta (3.27) non è l'unica possibile per completare il sistema di equazioni. Un'alternativa a (3.27) 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 3.2 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) in x_0 e a **der**(2) in x_n . Se **type**=1 viene invece calcolata la *spline* cubica interpolatoria con derivata seconda assegnata agli estremi e pari a **der**(1) in x_0 e a **der**(2) in x_n .

spline Diversamente, nel comando **spline** di **MAT&OCT** 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 **x** e **y** dei dati da interpolare ed il vettore **zi** delle ascisse nelle quali si vuole che venga valutata s_3 . I comandi **mkpp** e **ppval** servono per costruire e valutare efficientemente in **MAT&OCT** un polinomio composito.

Programma 3.2. **cubicspline**: spline cubica interpolante

```
function s=cubicspline(x,y,zi,type,der)
%CUBICSPLINE calcola una spline cubica
%   S=CUBICSPLINE(X,Y,XI) calcola le valutazioni
%   nei nodi XI 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 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));
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 3.9 Riprendiamo i dati della Tabella 3.1, della 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];
zi = [-55:1:65];
s = spline(x,y,zi);

```

Il grafico di s_3 , riportato in Figura 3.13, appare più plausibile di quello generato dall'interpolatore di Lagrange negli stessi nodi. ■

Esempio 3.10 (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 $x = x(t)$ e $y = y(t)$ interpolanti 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 3.2, per ottenere il risultato desiderato basta scrivere le seguenti istruzioni

```

x1 = [0 1 4]; 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];

```

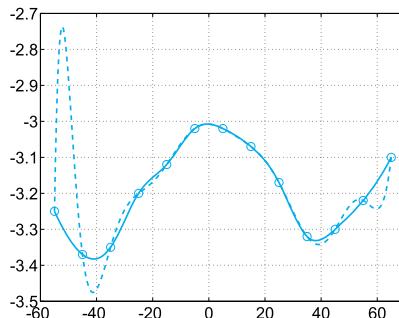


Figura 3.13. Confronto fra la *spline* cubica (in linea continua) ed il polinomio interpolatore di Lagrange (in linea tratteggiata) per il caso discusso nell'Esempio 3.9

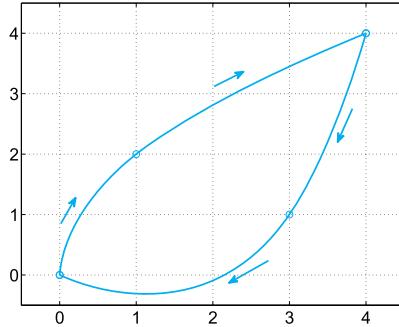


Figura 3.14. La traiettoria nel piano xy del robot descritto nel Problema 3.4. I pallini rappresentano le posizioni dei punti attraverso cui deve transitare il robot durante il suo movimento

```
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 3.14. ■

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 diseguaglianze [dB01]

$$\max_{x \in I} |f^{(r)}(x) - s_3^{(r)}(x)| \leq C_r H^{4-r} \max_{x \in I} |f^{(4)}(x)|, \quad r = 0, 1, 2 \quad (3.28)$$

e

$$\max_{x \in I \setminus \{x_0, \dots, x_n\}} |f^{(3)}(x) - s_3^{(3)}(x)| \leq C_3 H \max_{x \in I} |f^{(4)}(x)|,$$

dove $I = [x_0, x_n]$ e $H = \max_{i=0, \dots, n-1} (x_{i+1} - x_i)$, mentre C_r (per $r = 0, \dots, 3$) 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 unitaria del primo quadrante usando le coppie di punti $(x_k = \sin(k\pi/6), y_k = \cos(k\pi/6))$, per $k = 0, \dots, 3$, otterremmo la *spline* oscillante di Figura 3.15. In casi come questo conviene utilizzare altre tecniche di approssimazione. Ad esempio, il comando **pchip** di **MATLAB** genera un interpolatore cubico composito (detto di Hermite, si veda ad esempio [Atk89]) che, oltre alla funzione f , interpola anche la sua derivata prima nei nodi $\{x_i, i = 1, \dots, n-1\}$ e, soprattutto, garantisce la monotonia locale dell'interpolatore stesso (si veda la Figura 3.15). Tale interpolatore si ricava attraverso i seguenti comandi:

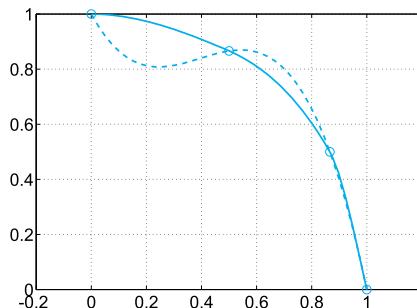


Figura 3.15. Approssimazione del primo quarto di circonferenza del cerchio unitario usando solo 4 nodi. La linea tratteggiata è il grafico della *spline* cubica interpolante, mentre la linea continua è il corrispondente interpolatore cubico composito di Hermite

```
t = linspace(0,pi/2,4)
x = sin(t); y = cos(t);
xx = linspace(0,1,40);
plot(xx,pchip(x,y,xx), 'c', 'Linewidth',2); hold on
plot(xx,spline(x,y,xx), 'c--', 'Linewidth',2)
plot(x,y,'co','MarkerSize',10);
grid on
```

Si vedano gli Esercizi 3.5–3.10.



3.6 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 3.11 (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 . Dato $m \geq 1$ (in genere, m sarà decisamente minore di n), cerchiamo un polinomio $\tilde{f} \in \mathbb{P}_m$ che soddisfi la seguente diseguaglianza

$$\sum_{i=0}^n [y_i - \tilde{f}(x_i)]^2 \leq \sum_{i=0}^n [y_i - p_m(x_i)]^2 \quad (3.29)$$

per ogni polinomio $p_m \in \mathbb{P}_m$. Quando esiste, diremo che \tilde{f} è l'*approssimazione nel senso dei minimi quadrati di grado m* (ovvero in \mathbb{P}_m) dei dati $\{(x_i, y_i), i = 0, \dots, n\}$. Se $m < n$ non sarà ora più possibile garantire che $\tilde{f}(x_i) = y_i$ per $i = 0, \dots, n$.

Ponendo

$$\tilde{f}(x) = a_0 + a_1 x + \dots + a_m x^m, \quad (3.30)$$

dove i coefficienti a_0, \dots, a_m sono incogniti, il problema (3.29) 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 (1.11)).

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

$$\begin{aligned} a_0(n+1) + a_1 \sum_{i=0}^n x_i &= \sum_{i=0}^n y_i, \\ a_0 \sum_{i=0}^n x_i + a_1 \sum_{i=0}^n x_i^2 &= \sum_{i=0}^n y_i x_i. \end{aligned} \quad (3.31)$$

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 in vari modi. La prima generalizzazione è al caso in cui m sia un intero arbitrario. Il sistema lineare quadrato di dimensione $m+1$ cui si perviene, che è simmetrico, avrà la forma seguente

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

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 3.11).

Il comando `c=polyfit(x,y,m)` di **MATLAB** calcola 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.

Esempio 3.12 (Finanza) In Figura 3.16 a sinistra 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. ■

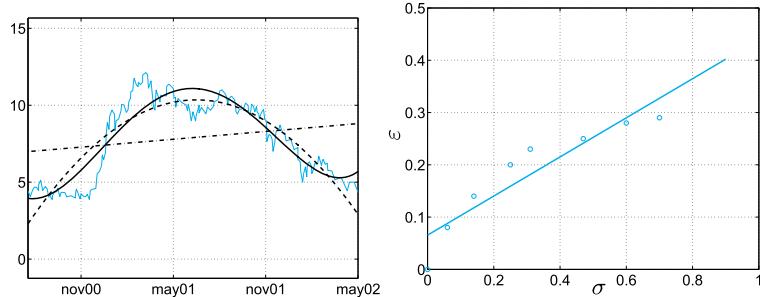


Figura 3.16. Approssimazioni nel senso dei minimi quadrati dei dati del Problema 3.2: con polinomi di grado 1 (linea tratto-punto), di grado 2 (linea tratteggiata) e di grado 4 (linea continua nera) (a sinistra). I dati esatti del problema sono rappresentati in linea sottile. L'approssimazione ai minimi quadrati con polinomi di grado 1 per i dati del Problema 3.3 (a destra)

Esempio 3.13 (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 a destra). 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 (3.29). Precisamente, nel problema di minimizzazione (3.29) 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 si cela dietro l'insieme 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 $\psi_j(x) = \cos(\gamma_j x)$, $j = 0, \dots, 4$, con $\gamma = \pi/60$.

Lasciamo al lettore di verificare che i coefficienti incogniti a_j che compaiono nell'espressione

$$\tilde{f}(x) = \sum_{j=0}^m a_j \psi_j(x),$$

sono le soluzioni del seguente sistema (delle equazioni normali)

$$\mathbf{B}^T \mathbf{B} \mathbf{a} = \mathbf{B}^T \mathbf{y} \quad (3.32)$$

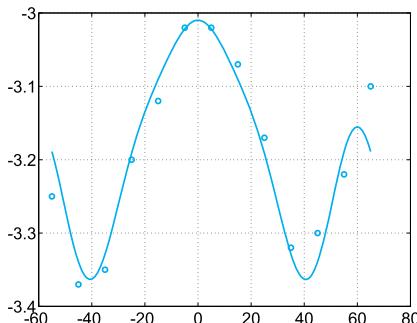


Figura 3.17. L'approssimazione nel senso dei minimi quadrati dei dati dell'Esempio 3.1 usando una base di coseni. I valori esatti sono rappresentati dai cerchietti

dove 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.

Il sistema (3.32) è un sistema di equazioni lineari che può essere risolto in maniera efficiente mediante la fattorizzazione QR o, in alternativa, attraverso una decomposizione in valori singolari della matrice B (si veda la Sezione 5.7).

Rimandiamo al Capitolo 7 per la risoluzione di problemi ai minimi quadrati non lineari, ovvero problemi in cui la funzione \tilde{f} dipende in maniera non lineare dai coefficienti incogniti a_j .

Si vedano gli Esercizi 3.11–3.17.



Riassumendo

1. L'interpolatore lineare composito di una funzione f è una funzione continua e lineare a pezzi 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. Esso è alla base dell'approssimazione di problemi differenziali con il metodo degli elementi finiti (si veda il Capitolo 9);
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 (solitamente $m \ll n$) 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 applicare ad una classe di funzioni \tilde{f} non necessariamente di tipo polinomiale.

3.7 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 in poliedri (tetraedri o prismi).

Una situazione particolarmente semplice è quella in cui Ω sia di forma rettangolare o parallelepipedo. In tal caso in **MAT&OCT** 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.

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 6×6 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 con il comando **interp2** 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 **MAT&OCT**. 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**).

Se Ω è una regione bidimensionale di forma generica, in MATLAB se ne può ottenere una decomposizione in triangoli utilizzando l'interfaccia **pdetool**. In Octave invece possiamo utilizzare il pacchetto **msh** di Octave-Forge che contiene *function* per la creazione e la manipolazione di mesh di triangoli o tetraedri compatibili con il **pdetool** di MATLAB. Il pacchetto **msh** si basa su GMSH (<http://gmsh.info>), un generatore di

mesh, distribuito secondo le condizioni d'uso della GNU General Public License.

Per una presentazione generale delle funzioni *spline* si veda, ad esempio, [Die93] e [PBP02]. Tramite i comandi `rpmak` e `rsmak` di MATLAB si possono generare 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*).

`rpmak`
`rsmak`

Il toolbox *Curve Fitting* di MATLAB consente di esplorare svariate applicazioni delle funzioni *spline*. In Octave-Forge sono presenti i pacchetti `splines` e `nurbs`, in particolare il secondo contiene un insieme di *function* per la creazione e la manipolazione di superfici e volumi NURBS.

Nell'ambito 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 `wavelet` di MATLAB e nel pacchetto `ltfat` di Octave-Forge.

3.8 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

Anno	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 1977, 1983 e 1988.

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 2002 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 con quelli ottenuti dallo svolgimento dell’Esercizio 3.3.

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 s_3 sull’intervallo di temperatura $[4, 20]$ suddiviso in 4 sottointervalli di uguale ampiezza. 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 si confronti con la produzione reale che è stata, rispettivamente, pari a 12380×10^5 kg, 27403×10^5 kg e 32059×10^5 kg. 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 $\{f(x_i) + (-1)^{i+1} 10^{-4}\}$, per $i = 0, \dots, n$, 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 Per interpolare la funzione $f(x) = 0.0039 + \frac{0.0058}{1+e^x}$ sull’intervallo $[-5, 5]$ si usi l’interpolazione globale di Lagrange su $(n+1)$ nodi di Chebyshev-Gauss-Lobatto (3.11), l’interpolazione composita lineare su N nodi equispaziati e l’interpolazione con spline cubica sempre su $(n+1)$ nodi equispaziati. Si analizzi l’errore di interpolazione al crescere di n .

Esercizio 3.10 Si approssimi la funzione $f(x) = e^{-x^2}$ sull’intervallo $[-2, 2]$ con l’interpolatore composito lineare definito su $(n+1)$ nodi equispaziati. Determinare il minimo numero di nodi che garantisca che l’errore di interpolazione sia minore o uguale di $\varepsilon = 10^{-6}$ e verificare numericamente quanto trovato.

Esercizio 3.11 Si verifichi che se $m = n$ e se $y_i = f(x_i)$ (per una opportuna funzione f) allora il polinomio dei minimi quadrati approssimante f nei nodi x_0, \dots, x_n coincide con $\Pi_n f$ interpolante f negli stessi nodi.

Esercizio 3.12 Si calcoli il polinomio di grado 4 che approssima nel senso dei minimi quadrati i valori di K riportati nelle colonne della Tabella 3.1.

Esercizio 3.13 Si ripetano i calcoli eseguiti nell'Esercizio 3.7 usando il polinomio dei minimi quadrati di grado 3.

Esercizio 3.14 Si esprimano i coefficienti del sistema (3.31) in funzione della media $M = \frac{1}{(n+1)} \sum_{i=0}^n x_i$ e della varianza $v = \frac{1}{(n+1)} \sum_{i=0}^n (x_i - M)^2$ relative all'insieme di dati $\{x_i, i = 0, \dots, n\}$.

Esercizio 3.15 Si verifichi che la retta di regressione passa per il punto la cui ascissa è la media dei valori $\{x_i\}$ e la cui ordinata è la media dei valori $\{y_i\}$.

Esercizio 3.16 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 .

Esercizio 3.17 Si stimi la complessità asintotica di un algoritmo di ordinamento di un vettore di lunghezza N , conoscendo i tempi di CPU, in secondi, richiesti per la sua esecuzione e riportati in Tabella:

$N(\times 10^3)$	10	20	30	40	50	60	70	80	90	100
CPUTime	0.31	0.99	2.29	4.10	6.41	9.20	12.60	16.32	20.83	25.47

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 difficile valutarla. Ad esempio nel caso in cui $f(x) = \cos(4x) \cos(3 \sin(x))$, si ha

$$\int_0^\pi f(x)dx = \pi \left(\frac{3}{2}\right)^4 \sum_{k=0}^{\infty} \frac{(-9/4)^k}{k!(k+4)!};$$

come si vede, il problema del calcolo di un integrale si è trasformato in quello (altrettanto 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.

4.1 Alcuni problemi

Problema 4.1 (Idraulica) Ad intervalli di 5 secondi è stata misurata la quota $q(t)$ in metri 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 (espressi in metri):

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 (Riconoscimento dei contorni) Avendo disponibile un'immagine in bianco e nero in formato jpeg (png, o altro formato) di $m \times n$ pixel, attraverso l'uso del comando `A=imread('filename.jpg')` di MATLAB possiamo generare una matrice A di n righe e m colonne i cui elementi contengono il valore dell'intensità luminosa nei pixel corrispondenti. Quest'ultima varia su una scala di interi dal valore 0, corrispondente al nero, al valore 255, corrispondente al bianco.

Il pixel (i, j) può essere associato al punto di \mathbb{R}^2 di coordinate

$$x = j \quad \text{e} \quad y = n + 1 - i, \quad \text{con } i = 1, \dots, n, \quad j = 1, \dots, m. \quad (4.1)$$

(Si noti che lungo l'asse verticale abbiamo scelto un ordinamento dei pixel dall'alto verso il basso per poter utilizzare l'ordinamento naturale degli indici di matrice.)

L'intensità luminosa è pertanto rappresentata da una funzione discreta $b : \{0, \dots, m\} \times \{0, \dots, n\} \rightarrow \{0, \dots, 255\}$, e $b(x, y) = A_{ij}$.¹

Al fine di individuare i contorni delle forme che costituiscono l'immagine, possiamo determinare i punti dell'immagine stessa in cui la variazione dell'intensità luminosa è particolarmente significativa. Introduciamo a tale scopo una funzione

$$g(x, y) = \sqrt{\left(\frac{\partial b}{\partial x}(x, y)\right)^2 + \left(\frac{\partial b}{\partial y}(x, y)\right)^2}, \quad (4.2)$$

supponendo che la funzione di intensità luminosa b in questo caso sia definita sui reali e non solo sugli interi e risulti essere differenziabile. Essendo tuttavia b disponibile soltanto sui valori interi, un'idea naturale consiste nel sostituire le derivate parziali in (4.2) con formule alle differenze finite che presuppongano solo la conoscenza dei valori di b nelle suddette variabili intere.

A valle di questo calcolo possiamo denotare con \bar{g} il valore medio di g sull'insieme dei pixel dell'immagine. I pixel che individuano i contorni saranno quelli in corrispondenza dei quali la funzione g assume un

¹ Qualora l'immagine sia a colori, A risulta essere una matrice di n righe, m colonne e 3 pagine, queste ultime corrispondenti ai tre colori fondamentali del formato RGB (red-green-blue); in tal caso l'intensità luminosa del pixel (i, j) può essere definita come $b(x, y) = \sum_{c=1}^3 A_{ijc}/3$.

valore significativamente più grande del suo valor medio. Risolveremo questo problema nell'Esempio 4.2, utilizzando le formule di derivazione alle differenze finite descritte in Sezione 4.2. ■

Problema 4.3 (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

$$E(T) = 2.39 \cdot 10^{-11} \int_{3 \cdot 10^{-4}}^{14 \cdot 10^{-4}} \frac{dx}{x^5(e^{1.432/(Tx)} - 1)}, \quad (4.3)$$

che è l'equazione di Planck per l'energia $E(T)$, 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 (4.3) si veda l'Esercizio 4.17. ■

Problema 4.4 (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 carica $\rho(r)$. 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

$$j(r, t) = \gamma(r)e^{-\sigma t/\varepsilon_0}, \quad \gamma(r) = \frac{\sigma}{\varepsilon_0 r^2} \int_0^r \rho(\xi)\xi^2 d\xi, \quad (4.4)$$

dove $\varepsilon_0 = 8.859 \cdot 10^{-12}$ farad/m è la costante dielettrica del vuoto.

Per il calcolo di questo integrale si veda l'Esercizio 4.16. ■

Problema 4.5 (Demografia) Consideriamo una popolazione formata da un numero M grande di individui. La distribuzione $n(s)$ 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 ,

$$n(s) = \frac{M}{\sigma\sqrt{2\pi}} e^{-(s-\bar{h})^2/(2\sigma^2)}.$$

Allora

$$N_{[h, h+\Delta h]} = \int_h^{h+\Delta h} n(s) ds \quad (4.5)$$

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

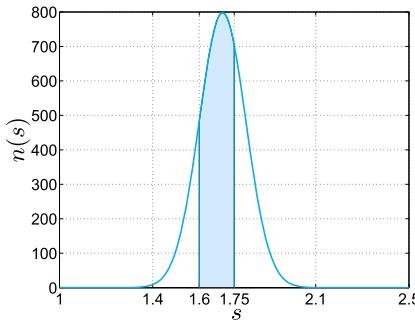


Figura 4.1. Distribuzione dell'altezza per una popolazione formata da $M = 200$ individui

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 1.6 e 1.75 m. Per la risoluzione di questo problema si veda l'Esempio 4.3. ■

4.2 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 (1.10), si può ritenere che, per h sufficientemente piccolo e positivo, la quantità

$$(\delta_+ f)(\bar{x}) = \frac{f(\bar{x} + h) - f(\bar{x})}{h} \quad (4.6)$$

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

$$f(\bar{x} + h) = f(\bar{x}) + h f'(\bar{x}) + \frac{h^2}{2} f''(\xi), \quad (4.7)$$

dove ξ è un punto opportuno in $(\bar{x}, \bar{x} + h)$. Pertanto

$$(\delta_+ f)(\bar{x}) = f'(\bar{x}) + \frac{h}{2} f''(\xi), \quad (4.8)$$

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). Supponendo ancora $f \in C^2((a, b))$, in maniera del tutto analoga, dal seguente sviluppo

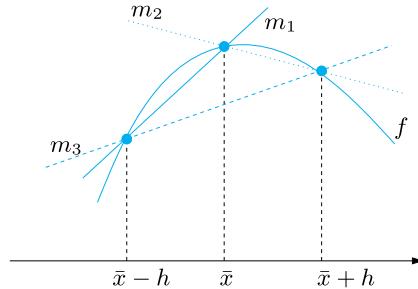


Figura 4.2. Approssimazione alle differenze finite di $f'(\bar{x})$: all'indietro (linea continua), in avanti (linea punteggiata) e centrata (linea tratteggiata). $m_1 = (\delta_+ f)(\bar{x})$, $m_2 = (\delta_- f)(\bar{x})$ e $m_3 = (\delta f)(\bar{x})$ rappresentano le pendenze delle rette indicate

$$f(\bar{x} - h) = f(\bar{x}) - h f'(\bar{x}) + \frac{h^2}{2} f''(\eta) \quad (4.9)$$

con $\eta \in (\bar{x} - h, \bar{x})$, possiamo ottenere la seguente formula, detta *differenza finita all'indietro*

$$(\delta_- f)(\bar{x}) = \frac{f(\bar{x}) - f(\bar{x} - h)}{h} \quad (4.10)$$

sempre accurata di ordine 1. Si noti che le formule (4.6) e (4.10) 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 centrata*

$$(\delta f)(\bar{x}) = \frac{f(\bar{x} + h) - f(\bar{x} - h)}{2h} \quad (4.11)$$

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

$$f'(\bar{x}) - (\delta f)(\bar{x}) = -\frac{h^2}{12}[f'''(\xi_-) + f'''(\xi_+)], \quad (4.12)$$

dove ξ_- e ξ_+ sono punti opportuni negli intervalli $(\bar{x} - h, \bar{x})$ e $(\bar{x}, \bar{x} + h)$, rispettivamente (si veda l'Esercizio 4.2).

Quando si usa la (4.11), 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 (4.6), (4.10) e (4.11) 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 (4.11) rispetto alle (4.10) o (4.6). ■

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 $h > 0$. Quando si vuole approssimare $f'(x_i)$, la si può sostituire con una qualunque delle formule (4.6), (4.10) o (4.11) in corrispondenza di $\bar{x} = x_i$.

Va osservato che la formula centrata (4.11) 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

$$\begin{aligned} \frac{1}{2h} [-3f(x_0) + 4f(x_1) - f(x_2)] &\quad \text{in } x_0, \\ \frac{1}{2h} [3f(x_n) - 4f(x_{n-1}) + f(x_{n-2})] &\quad \text{in } x_n, \end{aligned} \tag{4.13}$$

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).

Esempio 4.2 (Riconoscimento di contorni) Risolviamo il Problema 4.2 sull'immagine precedentemente memorizzata nel file `svalbard.png`. I comandi

```
A=imread('svalbard.png');
image(A); colormap(gray(256));
```

ci restituiscono l'immagine riportata in Figura 4.3, a sinistra. Come anticipato nel Problema 4.2 approssimiamo il valore della funzione g definita in (4.2) nei punti $(x, y) = (j, n + 1 - i)$ (si veda (4.1)) attraverso la seguente formula

$$g_{ij} = \sqrt{\text{dbx}_{ij}^2 + \text{dby}_{ij}^2}, \tag{4.14}$$

in cui dbx_{ij} rappresenta l'approssimazione alle differenze finite di $\frac{\partial b}{\partial x}$ nel punto (x, y) calcolata con (4.11), ovvero

$$\text{dbx}_{ij} = \frac{\mathbf{b}_{i,j+1} - \mathbf{b}_{i,j-1}}{2} \quad \text{per } j = 2, \dots, m - 1 \tag{4.15}$$

e, analogamente dby_{ij} rappresenta l'approssimazione alle differenze finite di $\frac{\partial b}{\partial y}$ nel punto (x, y) (sempre calcolata con (4.11)), ovvero

$$\text{dby}_{ij} = \frac{\mathbf{b}_{i+1,j} - \mathbf{b}_{i-1,j}}{2} \quad \text{per } i = 2, \dots, n-1. \quad (4.16)$$

In \mathbf{b}_{ij} è memorizzato il valore dell'intensità luminosa nel pixel (i, j) .

Osserviamo che, poiché la griglia è cartesiana, la derivata parziale lungo la direzione orizzontale, valutata in un pixel (i, j) coinvolge solo i pixel che stanno sulla riga di indice i e, analogamente la derivata lungo la direzione verticale coinvolge solo i pixel che stanno sulla colonna di indice j . Inoltre abbiamo ristretto il calcolo delle derivate ai soli pixel interni all'immagine, con esclusione di quelli corrispondenti alle prime e ultime righe e colonne.

Le istruzioni **MAT&OCT** per approssimare le derivate lungo entrambe le direzioni nei pixel interni all'immagine sono le seguenti:

```
[n, m]=size(A);
b=double(A);
e=1/2*ones(m, 1);
Dx=spdiags([-e, e], [-1, 1], m, m);
Dx(1, 2)=0; Dx(m, m-1)=0;
dbx=(Dx*b)';
Dy=spdiags([-e, e], [-1, 1], n, n);
Dy(1, 2)=0; Dy(n, n-1)=0;
dby=Dy*b;
```

Poiché il comando **imread** genera una variabile di interi a 8 bit (*uint8*), con il comando **double** l'abbiamo convertita in una variabile di tipo *double* al fine di poter svolgere le operazioni nel sistema *floating point*.

A questo punto valutiamo il suo valore medio di \bar{g} sull'insieme dei pixel dell'immagine

$$\bar{g} = \frac{1}{n \cdot m} \sum_{\substack{1 \leq i \leq n \\ 1 \leq j \leq m}} g_{ij} \quad (4.17)$$

mediante le seguenti istruzioni

```
g=sqrt(dbx.^2+dby.^2);
gm=sum(sum(g))/(n*m);
```

Infine definiamo l'insieme

$$\mathcal{C} = \left\{ (i, j) : \frac{|g_{ij} - \bar{g}|}{\bar{g}} > 1, \text{ con } 1 \leq i \leq n, 1 \leq j \leq m \right\} \quad (4.18)$$

ovvero l'insieme dei pixel in corrispondenza dei quali la variazione dell'intensità luminosa si discosta sensibilmente dal suo valore medio. I pixel di \mathcal{C} individueranno i contorni.

Per definire \mathcal{C} e visualizzare un'immagine con i soli contorni, inizializziamo una matrice $m \times n$ di nome **edge** e di tipo *uint8* con valori tutti uguali a 1 (cioè corrispondenti al bianco); selezioniamo i pixel (i, j) che soddisfano la diseguaglianza $|g_{ij} - \bar{g}|/\bar{g} > 1$ e poniamo uguale a 0 gli elementi edge_{ij} (al fine di ottenere pixel di colore nero). Le corrispondenti istruzioni **MAT&OCT** sono:

```
edge=255*uint8(ones(size(g)));
for j=1:m
    c_j=find(abs((g(:,j)-gm)/gm)>1);
```



Figura 4.3. L’immagine originale dell’Esempio 4.2 (*a sinistra*) e l’immagine dei contorni (*a destra*)

```

edge(cj,j)=0;
end
figure
image(edge); colormap(gray(256));
imwrite(edge, 'edge.jpg', 'jpg')

```

find Si noti che il comando `find(x)` di MATLAB restituisce un vettore contenente gli indici degli elementi non nulli di x .

L’immagine dei contorni è riportata a destra di Figura 4.3. ■



Si vedano gli Esercizi 4.1–4.4.

4.3 Integrazione numerica

In questa Sezione 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 .

4.3.1 La formula del punto medio

Una semplice procedura per approssimare $I(f)$ consiste nel suddividere l’intervallo $[a, b]$ in M sottointervalli di ampiezza $H = (b - a)/M$, $I_k = [x_{k-1}, x_k]$, $k = 1, \dots, M$, con $x_k = a + kH$, $k = 0, \dots, M$. Poiché

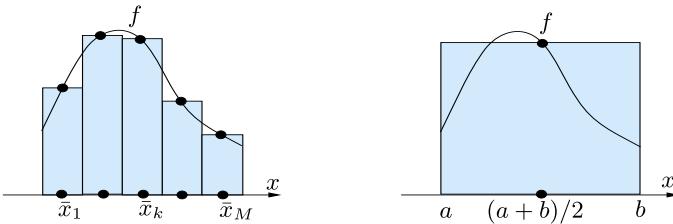


Figura 4.4. Formule del punto medio composito (a sinistra) e del punto medio (a destra)

$$I(f) = \sum_{k=1}^M \int_{I_k} f(x) dx, \quad (4.19)$$

su ogni sotto-intervallo I_k si sostituisce l'integrale di f con l'integrale di un polinomio \tilde{f} che approssimi f su I_k . La soluzione più semplice consiste nello scegliere 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*

$$I_{pm}^c(f) = H \sum_{k=1}^M f(\bar{x}_k) \quad (4.20)$$

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

$$I(f) - I_{pm}^c(f) = \frac{b-a}{24} H^2 f''(\xi), \quad (4.21)$$

dove ξ è un opportuno punto in $[a, b]$ (si veda l'Esercizio 4.6).

Più in generale una formula di quadratura composita ha *ordine di accuratezza p* rispetto ad H se l'errore $|I(f) - I_{app}(f)|$ tende a zero per H che tende a zero come H^p .

La formula (4.20) è anche nota come formula di quadratura *composita del rettangolo* per la sua interpretazione geometrica, che è evidente in Figura 4.4.

La *formula del punto medio classica* (nota anche come *formula del rettangolo*) si ottiene prendendo $M = 1$ nella (4.20), ovvero usando la formula del punto medio direttamente sull'intervallo (a, b)

$$I_{pm}(f) = (b-a)f[(a+b)/2] \quad (4.22)$$

L'errore ora è dato da

$$I(f) - I_{pm}(f) = \frac{(b-a)^3}{24} f''(\xi), \quad (4.23)$$

dove ξ è un opportuno punto in $[a, b]$.

La (4.23) segue come caso particolare della (4.21), 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 (4.23) segue in quanto $\int_a^b (x - \bar{x}) dx = 0$ e poiché, 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 qualsiasi polinomio di grado r è uguale all'integrale esatto. Come si deduce dalle (4.21) e (4.23), 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 4.1. 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 4.1. midpointc: formula composita del punto medio

```
function Imp=midpointc(fun,a,b,M,varargin)
% MIDPOINTC Formula composita del punto medio.
%   IMP = MIDPOINTC(FUN,A,B,M) calcola una
%   approssimazione dell'integrale della funzione
%   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 anonymous function o una function definita
%   in un M-file.
%   IMP = MIDPOINTC(FUN,A,B,M,P1,P2,...) passa alla
%   function FUN i parametri opzionali
%   P1,P2,... come FUN(X,P1,P2,...).
H=(b-a)/M;
x = linspace(a+H/2,b-H/2,M);
```

```
fmp=fun(x,varargin{:}).*ones(1,M);
Imp=H*sum(fmp);
return
```



Si vedano gli Esercizi 4.5–4.8.

4.3.2 La formula del trapezio

Si può ottenere un'altra formula di quadratura sostituendo f su ogni I_k con il suo interpolatore lineare nei nodi x_{k-1} e x_k (equivalentemente, sostituendo f in $[a, b]$ con l'interpolatore lineare composito $\Pi_1^H f$, si veda la Sezione 3.4). Si perviene alla formula seguente, detta *formula del trapezio composta*

$$\begin{aligned} I_t^c(f) &= \frac{H}{2} \sum_{k=1}^M [f(x_{k-1}) + f(x_k)] \\ &= \frac{H}{2} [f(a) + f(b)] + H \sum_{k=1}^{M-1} f(x_k) \end{aligned} \quad (4.24)$$

Essa è accurata al second'ordine rispetto a H , più precisamente

$$I(f) - I_t^c(f) = -\frac{b-a}{12} H^2 f''(\xi) \quad (4.25)$$

per un opportuno $\xi \in [a, b]$, purché $f \in C^2([a, b])$. Utilizzando (4.24) con $M = 1$, si trova la formula

$$I_t(f) = \frac{b-a}{2} [f(a) + f(b)] \quad (4.26)$$

detta *formula del trapezio* per via della sua interpretazione geometrica (si veda la Figura 4.5). L'errore che si commette vale

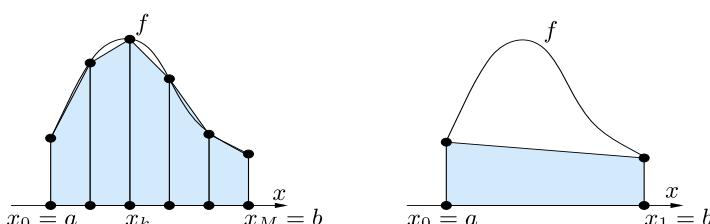


Figura 4.5. Formule del trapezio composta (a sinistra) e del trapezio (a destra)

$$I(f) - I_t(f) = -\frac{(b-a)^3}{12} f''(\xi), \quad (4.27)$$

con ξ opportuno in $[a, b]$. Si deduce che (4.26) ha grado di esattezza uguale ad 1, come la formula del punto medio.

La formula composita del trapezio (4.24) è implementata nei programmi `trapz` e `cumtrapz` di MATLAB. In particolare, se indichiamo `x` il vettore con componenti gli x_k e con `y` il vettore delle $f(x_k)$, `z=cumtrapz(x,y)` restituisce un vettore `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, `z(M+1)` contiene un'approssimazione dell'integrale di f su (a, b) .

4.3.3 La formula di Simpson

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

$$\begin{aligned} \Pi_2 f(x) &= \frac{2(x - \bar{x}_k)(x - x_k)}{H^2} f(x_{k-1}) \\ &+ \frac{4(x_{k-1} - x)(x - x_k)}{H^2} f(\bar{x}_k) + \frac{2(x - \bar{x}_k)(x - x_{k-1})}{H^2} f(x_k), \end{aligned}$$

si ottiene la *formula di quadratura composita di Simpson*

$$I_s^c(f) = \frac{H}{6} \sum_{k=1}^M [f(x_{k-1}) + 4f(\bar{x}_k) + f(x_k)] \quad (4.28)$$

Si può dimostrare che essa introduce un errore pari a

$$I(f) - I_s^c(f) = -\frac{b-a}{180} \frac{H^4}{16} f^{(4)}(\xi), \quad (4.29)$$

dove ξ è un punto opportuno in $[a, b]$, purché $f \in C^4([a, b])$. Essa ha pertanto ordine di accuratezza 4 rispetto a H . Se applicata ad un solo intervallo $[a, b]$ otteniamo la *formula di quadratura di Simpson*

$$I_s(f) = \frac{b-a}{6} [f(a) + 4f((a+b)/2) + f(b)] \quad (4.30)$$

L'errore ora vale

$$I(f) - I_s(f) = -\frac{1}{16} \frac{(b-a)^5}{180} f^{(4)}(\xi), \quad (4.31)$$

per un opportuno $\xi \in [a, b]$. Il grado di esattezza è quindi uguale a 3.

La formula (4.28) è implementata nel Programma 4.2.

Programma 4.2. simpsonc: formula composita di Simpson

```
function [Isic]=simpsonc(fun,a,b,M,varargin)
%SIMPSONC Formula composita di Simpson
% ISIC = SIMPSONC(FUN,A,B,M) calcola una
% 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 reale x
% e restituisce un vettore reale.
% FUN puo' essere una anonymous function
% o una function definita in un M-file.
% ISIC = SIMPSONC(FUN,A,B,M,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=fun(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=fun(x,varargin{:}).*ones(1,M);
Isic = Isic+2*H*sum(fpm)/3;
return
```

Esempio 4.3 (Demografia) Consideriamo il Problema 4.5. Per determinare il numero di individui la cui altezza è compresa fra 1.6 e 1.75 m, dobbiamo calcolare l'integrale (4.5) per $h = 1.6$ e $\Delta h = 0.15$. Usiamo la formula composita di Simpson con 100 sotto-intervalli:

```
M = 200; hbar = 1.7; sigma = 0.1;
N = @(h)M/(sigma*sqrt(2*pi))*exp(-(h-hbar).^...
    2./(2.*sigma.^2));
int = simpsonc(N, 1.6, 1.75, 100)

int =
106.56
```

Si stima quindi che il numero di individui con altezza nell'intervallo indicato è 106.56, corrispondente al 53.28% della popolazione. ■

Esempio 4.4 Vogliamo confrontare le approssimazioni dell'integrale $I(f) = \int_0^{2\pi} xe^{-x} \cos(2x)dx = -(10\pi - 3 + 3e^{2\pi})/(25e^{2\pi}) \simeq -0.122122604618968$ ottenute usando le formule composite del punto medio, del trapezio e di Simpson. In Figura 4.6 riportiamo in scala logaritmica l'andamento degli errori in funzione di H . Come osservato nella Sezione 1.7, 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. ■

4.4 Formule di quadratura interpolatorie

Le formule di quadratura come la (4.22), la (4.26) o la (4.30) corrispondenti ad un solo intervallo, ovvero alla scelta $M = 1$, si dicono *semplici*

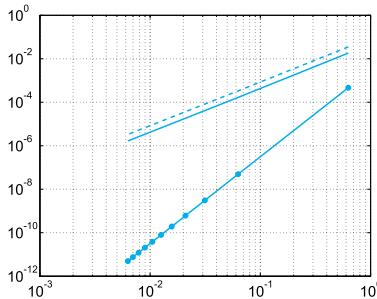


Figura 4.6. Rappresentazione in scala logaritmica degli errori (rispetto a H) per le formule composite di Simpson (linea continua con cerchietti), del punto medio (linea continua) e del trapezio (linea tratteggiata)

(o *non composite*). Esse rappresentano casi particolari della seguente formula generale

$$I_{appr}(f) = \sum_{j=0}^n \alpha_j f(y_j) \quad (4.32)$$

I numeri reali $\{\alpha_j\}$ sono detti *pesi*, mentre i punti $\{y_j\}$ sono detti *nodi*. Una formula di quadratura è detta *aperta* se i nodi di quadratura sono tutti interni all'intervallo (a, b) , *chiusa* se gli estremi a e b dell'intervallo di integrazione rientrano nell'insieme dei nodi di quadratura.

In generale, si richiede che la formula (4.32) 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 definito nella (3.3), tale che $\varphi_i(y_j) = \delta_{ij}$, per $i, j = 0, \dots, n$.

Esempio 4.5 Per la formula del trapezio (4.26) abbiamo $n = 1$, $y_0 = a$, $y_1 = b$ e

$$\begin{aligned} \alpha_0 &= \int_a^b \varphi_0(x) dx = \int_a^b \frac{x - b}{a - b} dx = \frac{b - a}{2}, \\ \alpha_1 &= \int_a^b \varphi_1(x) dx = \int_a^b \frac{x - a}{b - a} dx = \frac{b - a}{2}. \end{aligned}$$

■

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 (3.11) troveremo immediatamente i corrispondenti nodi e pesi su un intervallo $[a, b]$ generico,

$$y_j = \frac{a+b}{2} + \frac{b-a}{2}\bar{y}_j, \quad \alpha_j = \frac{b-a}{2}\bar{\alpha}_j. \quad (4.33)$$

La risposta al nostro quesito è contenuta nel risultato che segue (per la cui dimostrazione rimandiamo a [QSSG14, Cap. 9]):

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 ed il polinomio nodale $\omega_{n+1}(x) = \prod_{i=0}^n (x - \bar{y}_i)$ associato ai nodi $\{\bar{y}_i\}$ è tale che*

$$\int_{-1}^1 \omega_{n+1}(x)p(x)dx = 0, \quad \forall p \in \mathbb{P}_{m-1}. \quad (4.34)$$

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

$$L_0(x) = 1, \quad L_1(x) = x,$$

$$L_{k+1}(x) = \frac{2k+1}{k+1}xL_k(x) - \frac{k}{k+1}L_{k-1}(x), \quad k = 1, 2, \dots,$$

Si può dimostrare che un qualsiasi polinomio $p_n \in \mathbb{P}_n$ 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 Legendre 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 (4.34) 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

$$\begin{cases} \bar{y}_j = \text{zeri di } L_{n+1}(x), \\ \bar{\alpha}_j = \frac{2}{(1 - \bar{y}_j^2)[L'_{n+1}(\bar{y}_j)]^2}, \quad j = 0, \dots, n. \end{cases} \quad (4.35)$$

Tabella 4.1. Nodi e pesi di alcune formule di quadratura di Gauss-Legendre interni all’intervallo $[-1, 1]$. I pesi corrispondenti a coppie di nodi simmetrici rispetto allo 0 vengono riportati una volta sola

n	$\{\bar{y}_j\}$	$\{\bar{\alpha}_j\}$
1	$\{\pm 1/\sqrt{3}\}$	$\{1\}$
2	$\{\pm \sqrt{15}/5, 0\}$	$\{5/9, 8/9\}$
3	$\left\{ \pm(1/35)\sqrt{525 - 70\sqrt{30}}, \right.$ $\left. \pm(1/35)\sqrt{525 + 70\sqrt{30}} \right\}$	$\{(1/36)(18 + \sqrt{30}),$ $(1/36)(18 - \sqrt{30})\}$
4	$\left\{ 0, \pm(1/21)\sqrt{245 - 14\sqrt{70}} \right.$ $\left. \pm(1/21)\sqrt{245 + 14\sqrt{70}} \right\}$	$\{128/225, (1/900)(322 + 13\sqrt{70})$ $(1/900)(322 - 13\sqrt{70})\}$

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 $n \geq 1$

$$\begin{cases} \bar{y}_0 = -1, \bar{y}_n = 1, \bar{y}_j = \text{zeri di } L'_n(x), & j = 1, \dots, n-1, \\ \bar{\alpha}_j = \frac{2}{n(n+1)} \frac{1}{[L_n(\bar{y}_j)]^2}, & j = 0, \dots, n. \end{cases} \quad (4.36)$$

Se $f \in C^{(2n)}([-1, 1])$, l’errore corrispondente è pari a

$$I(f) - I_{GLL}(f) = -\frac{(n+1)n^3 2^{2n+1} ((n-1)!)^4}{(2n+1)((2n)!)^3} f^{(2n)}(\xi),$$

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 `quadl(fun,a,b)` di **MAT&OCT** è possibile approssimare un integrale con una formula di quadratura *composita* di Gauss-Legendre-Lobatto. La funzione da integrare deve essere precisata in input come *anonymous function*. Ad esempio, per integrare $f(x) = 1/x$ in

Tabella 4.2. Nodi e pesi di alcune formule di quadratura di Gauss-Legendre-Lobatto nell'intervallo $[-1, 1]$. I pesi corrispondenti a coppie di nodi simmetrici rispetto allo 0 vengono riportati una volta sola

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	$\{\pm 1, \pm \sqrt{21}/7, 0\}$	$\{1/10, 49/90, 32/45\}$

$[1, 2]$, definiamo prima la seguente *function*

```
fun=@(x) 1./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 `MAT&OCT` valuterà questa espressione componente per componente sul vettore dei 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 specifica tolleranza `tol`. Nella Sezione 4.5 introdurremo un metodo per stimare l'errore di quadratura e , conseguentemente, per cambiare H in modo adattivo.



Si vedano gli Esercizi 4.9–4.21.

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.

4.5 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 (4.28), grazie alla (4.29) basterebbe richiedere che

$$\frac{b-a}{180} \frac{H^4}{16} \max_{x \in [a,b]} |f^{(4)}(x)| < \varepsilon, \quad (4.37)$$

dove $f^{(4)}$ denota 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 (4.37) 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 passi di integrazione nell'intervallo $[a, b]$. In tal modo si garantisce la stessa accuratezza della formula di Simpson composita, ma con un numero inferiore di nodi di quadratura 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 approssima l'integrale $\int_a^{a+H} f(x)dx$ con l'accuratezza desiderata. Quando questo test è soddisfatto, 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 $A = [a, b]$, $N = \emptyset$ e $S = \emptyset$, mentre ad un passo intermedio avremo una situazione analoga a quella descritta nella Figura 4.7. 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

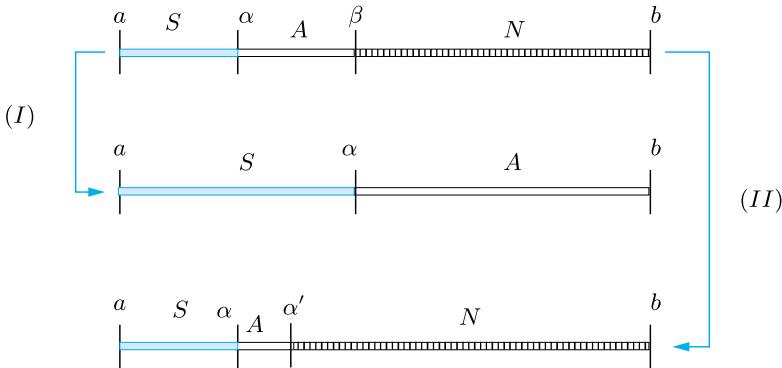


Figura 4.7. Distribuzione degli intervalli di integrazione ad un passo intermedio del processo di integrazione adattiva

f sull’intervallo attivo $[\alpha, \beta]$, rappresentato in bianco in Figura 4.7). 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 $S \leftarrow S \cup A, A = N, N = \emptyset$ (corrispondente al cammino (I) in Figura 4.7) 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 4.7);
 - (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 $[\alpha, \beta] \subset [a, b]$: 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 (4.31) segue che

$$E_s(f; \alpha, \beta) = \int_{\alpha}^{\beta} f(x) dx - I_s(f) = -\frac{(\beta - \alpha)^5}{2880} f^{(4)}(\xi),$$

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 (4.29) con $a = \alpha$ e $b = \beta$, troviamo che

$$\int_{\alpha}^{\beta} f(x)dx - I_s^c(f) = -\frac{(\beta - \alpha)^5}{46080} f^{(4)}(\eta), \quad (4.38)$$

per un opportuno η diverso da ξ . Sottraendo membro a membro le due ultime equazioni, si trova allora

$$\Delta I = I_s^c(f) - I_s(f) = -\frac{(\beta - \alpha)^5}{2880} f^{(4)}(\xi) + \frac{(\beta - \alpha)^5}{46080} f^{(4)}(\eta). \quad (4.39)$$

Assumiamo ora che $f^{(4)}(x)$ sia approssimativamente costante sull'intervallo $[\alpha, \beta]$. In tal caso, $f^{(4)}(\xi) \simeq f^{(4)}(\eta)$. Ricavando $f^{(4)}(\eta)$ dalla (4.39) e sostituendolo nella (4.38), si trova la seguente stima dell'errore:

$$\int_{\alpha}^{\beta} f(x)dx - I_s^c(f) \simeq \frac{1}{15} \Delta I.$$

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 4.3 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 4.3. simpadpt: formula di Simpson adattiva

```
function [JSf , nodes]=simpadpt(fun,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 ed utilizzando un passo variabile
%   H >= HMIN. FUN e' una function che riceve
%   in ingresso un vettore x e restituisce un vettore
%   reale. FUN puo' essere una anonymous function o
%   una function definita in un M-file.
%   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 distri-
% buzione di nodi usati nel processo di quadratura.
A=[a,b]; N=[]; S=[]; JSf = 0; ba = 2*(b - a); nodes=[];
while ~isempty(A),
    [deltaI,ISc]=caldeltai(A,fun,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,fun,varargin)
L=A(2)-A(1);
t=[0; 0.25; 0.5; 0.75; 1];
x=L*t+A(1);
L=L/6;
w=[1; 4; 1]; wp=[1;4;2;4;1];
fx=fun(x,varargin{:}).*ones(5,1);
IS=L*sum(fx([1 3 5]).*w);
ISc=0.5*L*sum(fx.*wp);
deltaI=IS-ISc;
return
```

Esempio 4.6 Calcoliamo $I(f) = \int_{-1}^1 20(1-x^2)^3 dx$ con la formula di Simpson adattiva. Eseguendo il Programma 4.3 con:

```
fun=@(x)(1-x.^2).^3*20;
tol = 1.e-04; hmin = 1.e-03; a=-1;b=1;
[JSf,nodes]=simpadpt(fun,a,b,tol,hmin)
```

troviamo il valore approssimato 18.2857116732797, mentre il valore esatto è 18.2857142857143. L'errore risulta pari a $2.6124 \cdot 10^{-6}$, minore della tolleranza richiesta di 10^{-4} . Si noti che per ottenere questo risultato servono 41 valutazioni funzionali. La formula di Simpson composita con passo uniforme avrebbe richiesto circa 90 valutazioni funzionali per ottenere un errore pari a $2.5989 \cdot 10^{-6}$. ■

4.6 Metodi Monte Carlo per l'integrazione numerica

I metodi Monte Carlo sono utilizzati in una innumerevole varietà di contesti, fra i quali la risoluzione di equazioni differenziali stocastiche e la quantificazione dell'incertezza.

Quando usati per l'integrazione numerica su regioni $\Omega \subset \mathbb{R}^n$, essi risultano competitivi quando n è grande. Ciò dipende dal fatto che la scelta dei nodi di quadratura è fatta *statisticamente*, in funzione dei valori ottenuti da variabili casuali di cui è nota la distribuzione di probabilità.

Una *variabile casuale* (o *stocastica*) $\mathbf{X} = \mathbf{X}(\zeta) = (X_1(\zeta), \dots, X_n(\zeta)) \in \mathbb{R}^n$ (o, più propriamente, un *vettore casuale*) è una funzione definita per ogni valore ζ , ed i suoi valori sono ottenuti in risposta ad un esperimento casuale. Alla variabile casuale è sempre associata una *funzione di densità di probabilità* $p(\mathbf{X})$ (si veda [Pap87], Capitolo 4), cioè una funzione a valori reali con le proprietà

$$p(X_1, \dots, X_n) \geq 0, \quad \int_{\mathbb{R}^n} p(X_1, \dots, X_n) d\mathbf{X} = 1.$$

Dato un vettore $\mathbf{x} = (x_1, \dots, x_n)^T \in \mathbb{R}^n$, la *probabilità* $\mathcal{P}\{\mathbf{X} \leq \mathbf{x}\}$ che si verifichi l'evento casuale $\{X_1 \leq x_1, \dots, X_n \leq x_n\}$ è data da

$$\mathcal{P}\{\mathbf{X} \leq \mathbf{x}\} = \int_{-\infty}^{x_n} \dots \int_{-\infty}^{x_1} p(X_1, \dots, X_n) dX_1 \dots dX_n.$$

Infine, data una funzione f definita sulla variabile casuale \mathbf{X} con funzione di densità di probabilità associata $p(\mathbf{X})$, la *media statistica* (o *attesa*) di f è

$$\mu(f) = \int_{\mathbb{R}^n} f(\mathbf{X}) p(\mathbf{X}) d\mathbf{X}. \quad (4.40)$$

L'idea alla base del metodo Monte Carlo consiste nell'interpretare l'integrale della funzione f in termini della *media statistica* della funzione stessa. Identificando un generico punto $\mathbf{x} = (x_1, x_2, \dots, x_n)^T \in \mathbb{R}^n$ con la variabile casuale $\mathbf{X} = (X_1, X_2, \dots, X_n)^T$ abbiamo

$$\int_{\Omega} f(\mathbf{x}) d\mathbf{x} = |\Omega| \int_{\mathbb{R}^n} |\Omega|^{-1} \chi_{\Omega}(\mathbf{X}) f(\mathbf{X}) d\mathbf{X} = |\Omega| \mu(f),$$

dove:

- $|\Omega|$ denota la misura n -dimensionale di Ω ,
- $\chi_{\Omega}(\mathbf{X})$ è la funzione caratteristica dell'insieme Ω , uguale a 1 per $\mathbf{X} \in \Omega$ e 0 altrimenti,
- $p(\mathbf{X}) = |\Omega|^{-1} \chi_{\Omega}(\mathbf{X})$ è la funzione di densità di probabilità associata alla variabile casuale \mathbf{X} .

Il calcolo numerico della media statistica $\mu(f)$ è svolto prendendo N campioni $\mathbf{X}_1, \dots, \mathbf{X}_N \in \mathbb{R}^n$ della variabile casuale \mathbf{X} , fra di loro mutuamente indipendenti e valutando la *media aritmetica*

$$I_N(f) = \frac{1}{N} \sum_{i=1}^N f(\mathbf{X}_i). \quad (4.41)$$

La legge dei grandi numeri assicura con probabilità 1 che la successione delle medie aritmetiche $I_N(f)$ converge alla media statistica $\mu(f)$ quando $N \rightarrow \infty$. Nella pratica, l'insieme dei campioni $\mathbf{X}_1, \dots, \mathbf{X}_N$ è costruito deterministicamente con un generatore di numeri casuali e la formula così ottenuta è detta *formula di integrazione pseudo-casuale*.

L'errore di quadratura $E_N(f) = \mu(f) - I_N(f)$ può essere espresso in termini della *varianza*

$$\sigma^2(I_N(f)) = \mu((\mu(f) - I_N(f))^2)$$

e inoltre si ha

$$\sigma^2(I_N(f)) = \frac{\sigma^2(f)}{N}, \quad (4.42)$$

per cui, quando $N \rightarrow \infty$, l'ordine di convergenza $\mathcal{O}(N^{-1/2})$ è una conseguenza della stima statistica dell'errore $E_N(f)$, poiché $E_N(f) \propto \sqrt{\sigma^2(I_N(f))}$ (si veda ad esempio [KW08]).

La velocità di convergenza del metodo Monte Carlo *non* dipende dalla dimensione n del dominio di integrazione. Tuttavia, è importante osservare che essa è indipendente dalla *regolarità* di f e, a differenza di quanto succede per le formula di quadratura interpolatorie, i metodi Monte Carlo non guadagnano in accuratezza quando la funzione integranda è molto regolare.

La stima (4.42) è molto debole ed in pratica si ottengono spesso risultati poco accurati. Una implementazione più efficiente dei metodi Monte Carlo è basata su un approccio composito o su metodi semi-analitici; un esempio di queste tecniche è fornito nella Libreria NAG (Numerical Algorithms Group) dove è impiegato un metodo Monte Carlo composito per il calcolo di integrali su ipercubi in \mathbb{R}^n .

Esempio 4.7 Calcoliamo gli integrali $I_{2D} = \int_{[0,1]^2} \sin(\pi x) \cos(\pi y) dx dy$ e $I_{3D} = \int_{[0,1]^3} \sin(\pi x) \cos(\pi y) \sin(\pi z) dx dy dz$ con la formula Monte Carlo (4.41) con $N = 10^{k/2}$ e $k = 0, \dots, 15$. I valori assoluti degli errori $E_N(f)$ al variare di N sono riportati in Figura 4.8 (a sinistra il caso 2D, a destra il caso 3D). La velocità di convergenza $\mathcal{O}(N^{-1/2})$ dell'errore è rispettata in entrambi i casi. ■

4.7 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 [QSSG14, Cap. 8]. Analogamente, le formule di Gauss-Legendre e di Gauss-Legendre-Lobatto appartengono alla famiglia delle formule di quadratura Gaussiane: esse sono *ottimali* nel senso che massimizzano il grado di esattezza, a parità di numero di nodi. Rimandiamo a [QSSG14,

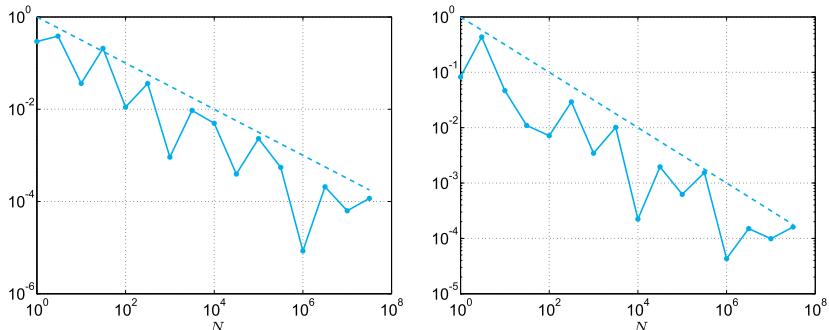


Figura 4.8. Valori assoluti degli errori $E_N(f)$ del metodo Monte Carlo (linea continua) e $N^{-1/2}$ (linea tratteggiata) per le funzioni dell’Esempio 4.7. A sinistra, il caso 2D. A destra, il caso 3D

Cap. 9] o a [RR01] 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 $\int_0^\infty f(x)dx$. Se per un opportuno punto α , il valore di $\int_\alpha^\infty f(x)dx$ può essere trascurato rispetto a quello di $\int_0^\infty f(x)dx$, ci si può limitare a calcolare quest’ultimo con una formula di quadratura. Alternativamente si può ricorrere a formule di quadratura di Gauss per intervalli illimitati (si veda [QSSG14, Cap. 9]).

dblquad

Infine, l’integrazione numerica può essere estesa ad integrali su domini multidimensionali. Il comando `dblquad(fun,xmin,xmax,ymin,ymax)` di MATLAB consente ad esempio di approssimare l’integrale di una data funzione $f(x, y)$ sul rettangolo $[xmin, xmax] \times [ymin, ymax]$. La funzione integranda è precisata attraverso un *function handle* `fun` che deve avere come parametri d’ingresso almeno le due variabili x e y rispetto alle quali si calcola l’integrale doppio. Qualora la funzione f sia valutata attraverso un M-file `fun.m`, l’istruzione di chiamata sarà:

`dblquad(@fun,xmin,xmax,ymin,ymax).`

4.8 Esercizi

Esercizio 4.1 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 (4.13) 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 4.2 Si verifichi che se $f \in C^3$ in un intorno di \bar{x} l’errore della formula (4.11) è dato dalla (4.12).

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 (Demografia) 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:

$$\begin{aligned} f_1(x) &= \frac{1}{1 + (x - \pi)^2} \quad \text{in } [0, 5], \\ f_2(x) &= e^x \cos(x) \quad \text{in } [0, \pi], \\ f_3(x) &= \sqrt{x(1 - x)} \quad \text{in } [0, 1], \end{aligned}$$

utilizzando la formula composita del punto medio. Si verifichino sperimentalmente i risultati ottenuti tramite il Programma 4.1.

Esercizio 4.6 Si dimostri la (4.21) a partire dalla (4.23).

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 la formula composita del trapezio e la formula composita di Gauss-Legendre con $n = 1$.

Esercizio 4.10 Siano I_1 e I_2 due approssimazioni di $I(f) = \int_a^b f(x)dx$, ottenute utilizzando la formula composita del trapezio con due passi di quadratura diversi, H_1 e H_2 . Se $f^{(2)}$ non varia molto in (a, b) , il seguente valore

$$I_R = I_1 + (I_1 - I_2)/(H_2^2/H_1^2 - 1) \quad (4.43)$$

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 (4.25) si ricavi la (4.43).

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 di Gauss-Legendre 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 (4.30) 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 derivi la formula di estrapolazione di Richardson per le formule di Simpson (4.30) e di Gauss-Legendre per $n = 1$ di Tabella 4.1. Quindi si applichi all'approssimazione dell'integrale $I(f) = \int_0^2 e^{-x^2/2} dx$ con $H_1 = 1$ e $H_2 = 0.5$. 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 (4.4) per $r = k/10$ m con $k = 1, \dots, 10$, $\rho(\xi) = \exp(\xi)$ 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 (4.3) 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} .

Esercizio 4.19 Data un'asta di sezione quadrata di area $A = 1 \text{ cm}^2$, di lunghezza $L = 1 \text{ m}$ e densità lineare variabile $\lambda(x) = \sin(x)/x \text{ kg/m}$, si calcolino la massa $m = A \int_0^L \lambda(x)dx$ dell'asta e la posizione (lungo l'asse x) del suo centro di massa

$$\bar{x} = \frac{\int_0^L x\lambda(x)dx}{\int_0^L \lambda(x)dx},$$

utilizzando un metodo opportuno di quadratura composita, garantendo che nell'approssimare la massa si commetta un errore al più pari ad $\epsilon = 10^{-8}$.

Esercizio 4.20 Si vuole calcolare, con precisione dell'ordine di 10^4 km , la lunghezza dell'orbita terrestre attorno al sole. I semiassi maggiore e minore dell'orbita ellittica misurano rispettivamente $a = 149.60 \cdot 10^6 \text{ km}$ e $b = a\sqrt{1 - e^2} \text{ km}$, con $e = 0.0167086$ (eccentricità) e la lunghezza di una ellisse è

$$L = \int_0^{2\pi} \sqrt{a^2 \cos^2(t) + b^2 \sin^2(t)} dt.$$

Utilizzando la formula composita dei trapezi, dire qual è il minimo numero di intervalli (di uguale ampiezza H) che permette di approssimare l'integrale dato con l'accuratezza richiesta. Fornire infine il valore calcolato dell'orbita terrestre.

Esercizio 4.21 Sia $f : [a, b] \rightarrow \mathbb{R}$. Dimostrare che la formula di quadratura

$$\tilde{I}(f) = \frac{b-a}{8} \left[f(a) + 3f\left(\frac{2a+b}{3}\right) + 3f\left(\frac{a+2b}{3}\right) + f(b) \right] \quad (4.44)$$

ha grado di esattezza pari a 3. Riscrivere la formula in forma composita su M intervalli di uguale ampiezza H e determinare numericamente il suo ordine di accuratezza rispetto ad H .

Sistemi lineari

Nelle scienze applicate la risoluzione di problemi, anche complessi, viene spesso ricondotta alla risoluzione di uno o più sistemi lineari della forma

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \quad (5.1)$$

dove \mathbf{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

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n &= b_1, \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n &= b_2, \\ &\vdots && \vdots && \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \dots + a_{nn}x_n &= b_n. \end{aligned}$$

Presentiamo quattro problemi che danno luogo a sistemi lineari.

5.1 Alcuni problemi

Problema 5.1 (Rete Idrica) 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 $p_0 = 10$ 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 alle estremità della condotta

$$Q_j = \frac{1}{R_j L_j} \Delta p_j, \quad (5.2)$$

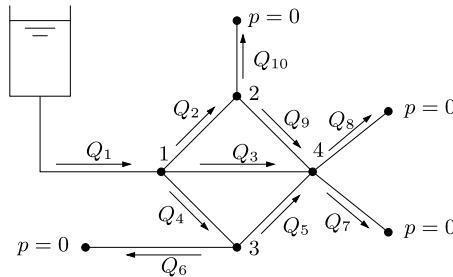


Figura 5.1. La rete di condotte del Problema 5.1

dove R_j è la resistenza idraulica per unità di lunghezza (misurata in $(\text{bar s})/\text{m}^4$) e L_j la lunghezza (in m) della condotta j -sima. 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.

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 $\mathbf{Ap} = \mathbf{b}$.

Nella seguente tabella riassumiamo le caratteristiche principali delle diverse condotte specificate dall'indice j .

j	R_j	L_j	j	R_j	L_j	j	R_j	L_j
1	0.2500	20	2	2.0000	10	3	1.0204	14
4	2.0000	10	5	2.0000	10	6	7.8125	8
7	7.8125	8	8	7.8125	8	9	2.0000	10
10	7.8125	8						

Corrispondentemente, \mathbf{A} e \mathbf{b} assumeranno i seguenti valori (abbiamo riportato le sole prime 4 cifre significative):

$$\mathbf{A} = \begin{bmatrix} -0.370 & 0.050 & 0.050 & 0.070 \\ 0.050 & -0.116 & 0 & 0.050 \\ 0.050 & 0 & -0.116 & 0.050 \\ 0.070 & 0.050 & 0.050 & -0.202 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} -2 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

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. Leontief propose nel 1930¹ un modello di produzione di tipo lineare per il quale 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, 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: un coefficiente a_{ij} positivo (risp. negativo) indica la quantità netta del bene i -esimo prodotto (risp. consumato) dall'impresa j -esima. 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 $\mathbf{b} = (b_i)$ di \mathbb{R}^n (il vettore della *domanda finale*). La componente b_i rappresenta dunque la quantità del bene i -esimo richiesta dal mercato. L'equilibrio è raggiunto quando il vettore $\mathbf{x} = (x_i) \in \mathbb{R}^n$ della produzione egualia la domanda totale ovvero

$$A\mathbf{x} = \mathbf{b}, \quad \text{dove } A = P - C. \quad (5.4)$$

In questo modello 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 5.20. ■

¹ Nel 1973 Wassily Leontief fu insignito del premio Nobel in economia per i suoi studi.

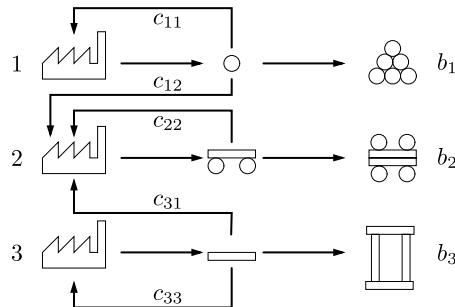


Figura 5.2. Lo schema di interazione fra 3 industrie ed il mercato descritto nel Problema 5.3



Figura 5.3. Un letto capillare

Problema 5.4 (Rete di capillari) I capillari sono piccolissimi vasi sanguigni, la più piccola unità del sistema circolatorio. Si raggruppano in reti dette letti capillari costituite da un numero variabile di elementi, indicativamente fra 10 e 100, a seconda dell'organo o del tipo di tessuto. Il sangue ossigenato vi ci arriva attraverso piccole arterie dette arteriole e, nel letto capillare, avviene la cessione di sangue ossigenato ai tessuti attraverso le pareti dei globuli rossi e l'eliminazione delle scorie metaboliche (i cataboliti). Il sangue che fluisce nel letto capillare viene infine raccolto da piccole vene e quindi da vene che lo riconducono al cuore.

Un letto capillare (si veda un esempio in Figura 5.3) può essere descritto da un modello di rete, simile a quella idrica considerata nel Problema 5.1, in cui ogni capillare è assimilato ad una condotta i cui estremi vengono detti nodi. Nella rappresentazione schematica di Figura 5.4 essi sono raffigurati da piccoli cerchi vuoti. L'arteriola che alimenta il letto capillare può essere considerata funzionalmente equivalente ad un serbatoio di pressione uniforme (di circa 50 mmHg, dove mmHg sta per millimetro di mercurio e 760 mmHg equivalgono a 1 atmosfera). In que-

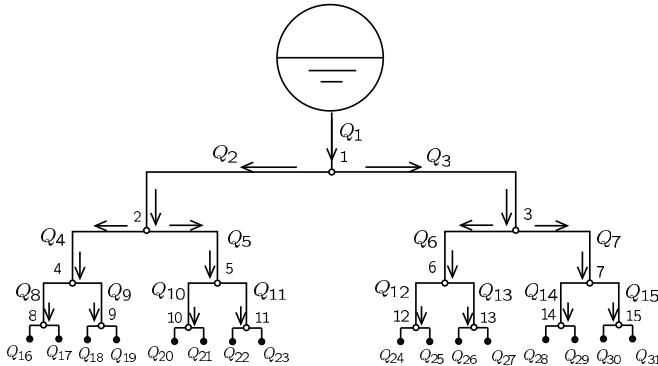


Figura 5.4. Schematizzazione di un letto capillare

sto modello possiamo supporre che ai nodi di uscita del letto capillare (indicati in figura con dei piccoli cerchi neri) la pressione abbia un valore costante (pari alla pressione venosa), che per semplicità normalizzeremo a zero. Lo scorrimento del sangue dall'arteriola fin verso i nodi di uscita è dovuto alla differenza di pressione fra un nodo e quello gerarchicamente inferiore.

Con riferimento alla Figura 5.4, indichiamo con p_j , $j = 1, \dots, 15$ (misurata in mmHg) la pressione nel nodo j -simo e con Q_m , $m = 1, \dots, 31$ (misurata in mm^3/s) la portata nel capillare m -simo. Per ogni m , indicando con i e j i nodi che delimitano il capillare m -simo, adotteremo la seguente relazione costitutiva

$$Q_m = \frac{1}{R_m L_m} (p_i - p_j), \quad (5.5)$$

dove R_m indica la resistenza idraulica per unità di lunghezza (in $(\text{mmHg s})/\text{mm}^4$) e L_m la lunghezza (in mm) del capillare in esame. Naturalmente quando si tratta il nodo 1 si dovrà tenere conto che $p_0 = 50$; analogamente, quando si trattano i nodi estremi, dal n. 8 al n. 15, si dovranno porre uguali a zero le pressioni nei nodi di uscita (dal n. 16 al n. 31). Infine, in ogni nodo della rete imporremo un'equazione di bilancio fra le portate in ingresso e quelle in uscita, ovvero

$$\left(\sum_{m \text{ entranti}} Q_m \right) - \left(\sum_{m \text{ uscenti}} Q_m \right) = 0.$$

In questo modo si perviene al sistema lineare

$$\mathbf{A}\mathbf{p} = \mathbf{b}, \quad (5.6)$$

dove $\mathbf{p} = [p_1, p_2, \dots, p_{15}]^T$ è il vettore delle pressioni incognite nei 15 nodi della rete, \mathbf{A} è la matrice dei coefficienti, mentre \mathbf{b} è il vettore dei dati.

Supponendo per semplicità che i capillari abbiano tutti la stessa resistenza idraulica $R_m = 1$ e che la lunghezza del primo capillare sia $L_1 = 20$, mentre ad ogni biforcazione la lunghezza dei capillari si dimezzhi (pertanto $L_2 = L_3 = 10$, $L_4 = \dots = L_7 = 5$ etc.), si ottiene la seguente matrice

$$A = \begin{bmatrix} -\frac{1}{4} & \frac{1}{10} & \frac{1}{10} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{10} & -\frac{1}{2} & 0 & \frac{1}{5} & \frac{1}{5} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{10} & 0 & -\frac{1}{2} & 0 & 0 & \frac{1}{5} & \frac{1}{5} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{5} & 0 & -1 & 0 & 0 & 0 & 0.4 & 0.4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{5} & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0.4 & 0.4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{5} & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0.4 & 0.4 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{5} & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0.4 & 0.4 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{5} & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0.4 & 0.4 & 0 \\ 0 & 0 & 0 & 0.4 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.4 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.4 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.4 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.4 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.4 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.4 & 0 & 0 & 0 & 0 & 0 & -2 & 0 \end{bmatrix}$$

mentre $\mathbf{b} = [-5/2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]^T$.

La risoluzione di questo sistema verrà discussa nell'Esempio 5.7. ■

5.2 Sistemi e complessità

La soluzione del sistema (5.1) esiste ed è unica 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 $3(n + 1)!$ operazioni intendendo, come al solito, per operazione la singola somma, sottrazione, moltiplicazione o divisione. Ad esempio, su un calcolatore in grado di eseguire 10^{11} flops (cioè 100 Giga flops) servirebbero poco meno di 11 minuti per risolvere un sistema di dimensione $n = 15$, 48 anni se $n = 20$ e 10^{141} anni se $n = 100$. Si veda la Tabella 5.1.

Si osservi che 10^{11} flops è la velocità di un comune PC dotato di un processore Intel® Core™ i7-4790 3.60 GHz con 4 core mentre il computer Sunway TaihuLight con 10 649 600 core, primo della lista top500 dei supercomputer del mondo a giugno 2017 (si veda www.top500.org), ha una velocità di 93 Peta-flops (cioè circa $93 \cdot 10^{15}$ flops).

Tabella 5.1. Tempo richiesto per risolvere un sistema lineare di dimensione n mediante la regola di Cramer. “f.p.” sta per “fuori portata”

n	Flops				
	10^9 (Giga)	10^{10}	10^{11}	10^{12} (Tera)	10^{15} (Peta)
10	10^{-1} sec	10^{-2} sec	10^{-3} sec	10^{-4} sec	trascurabile
15	17 ore	1.74 ore	10.46 min	1 min	$0.6 \cdot 10^{-1}$ sec
20	4860 anni	486 anni	48.6 anni	4.86 anni	1.7 giorni
25	f.p.	f.p.	f.p.	f.p.	38365 anni

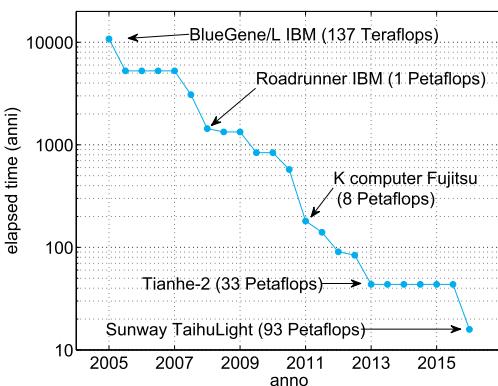


Figura 5.5. Tempo in anni per risolvere un sistema lineare di 24 equazioni in 24 incognite con il metodo di Cramer sui supercomputer al top della lista www.top500.org al variare dell’anno solare

A partire da giugno 2005 la lista sul sito www.top500.org viene aggiornata semestralmente, tracciando così l’evoluzione delle performance dei supercomputer.

Se avessimo voluto risolvere con il metodo di Cramer un sistema di dimensione 24 sul calcolatore più veloce al mondo nel giugno 2005 (il BlueGene/L di IBM, con un’velocity di picco di 137 Teraflops) avremmo dovuto attendere 10 800 anni; sfruttando invece il calcolatore più veloce al mondo nel giugno 2017 (il Sunway TaihuLight, con una velocity di picco di 93 Petaflops) sarebbero bastati (per modo di dire) 16 anni, un tempo sicuramente più ragionevole del precedente, ma comunque non accettabile! In Figura 5.5 abbiamo riportato alcuni dei nomi dei supercomputer al top della lista top500 tra il 2005 e il 2017 (con le relative performance di picco) ed il tempo che questi avrebbero impiegato per risolvere il nostro sistema lineare.

Il costo computazionale della risoluzione di un sistema lineare 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’E-

sempio 1.3. Tuttavia, tale costo risulterebbe ancora troppo elevato per le applicazioni pratiche.

Considereremo due possibili approcci alternativi: 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 dalla Sezione 5.9, 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 la Sezione 5.12 per maggiori dettagli).

Facciamo notare che in generale un sistema lineare con matrice piena 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.

Sebbene molti dei metodi che andremo a presentare in questo Capitolo siano validi per matrici a coefficienti complessi, per semplicità limiteremo la trattazione al caso di matrici a coefficienti reali. Osserviamo comunque che le *function* di MATLAB e di Octave per la risoluzione di sistemi lineari lavorano non solo con variabili reali, ma anche con variabili complesse senza bisogno di modificare le istruzioni di chiamata.

Faremo esplicito riferimento a matrici complesse solo laddove le ipotesi sulle matrici reali devono essere sostituite da specifiche condizioni in campo complesso, come per esempio nella definizione di matrici definite positive o nella presentazione della fattorizzazione di Cholesky.

5.3 Il metodo di fattorizzazione LU

Sia $A \in \mathbb{R}^{n \times n}$. Supponiamo che esistano due opportune matrici L ed U , triangolare inferiore e superiore, rispettivamente, tali che

$$A = LU \quad (5.7)$$

La (5.7) 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 nella Sezione 1.4).

In tal caso, risolvere $Ax = b$ conduce alla risoluzione dei due seguenti sistemi triangolari

$$Ly = b, \quad Ux = y \quad (5.8)$$

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*

$$\begin{aligned} y_1 &= \frac{1}{l_{11}}b_1, \\ y_i &= \frac{1}{l_{ii}} \left(b_i - \sum_{j=1}^{i-1} l_{ij}y_j \right), \quad i = 2, \dots, n \end{aligned} \tag{5.9}$$

Quantifichiamo il numero di operazioni richiesto da (5.9) 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

$$\begin{aligned} x_n &= \frac{1}{u_{nn}}y_n, \\ x_i &= \frac{1}{u_{ii}} \left(y_i - \sum_{j=i+1}^n u_{ij}x_j \right), \quad i = n-1, \dots, 1 \end{aligned} \tag{5.10}$$

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 (5.7) 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)

$$\begin{array}{ll} (e_1) & l_{11}u_{11} = a_{11}, \\ (e_3) & l_{21}u_{11} = a_{21}, \end{array} \quad \begin{array}{ll} (e_2) & l_{11}u_{12} = a_{12}, \\ (e_4) & l_{21}u_{12} + l_{22}u_{22} = a_{22}. \end{array} \quad (5.11)$$

Il sistema (5.11) 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 (5.11) può essere risolto procedendo nel modo seguente: dalle (e₁) ed (e₂) ricaviamo gli elementi u_{11} ed u_{12} della prima riga di U. 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), l'unico elemento non nullo u_{22} 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

$$\begin{array}{lll} (e_1) & l_{11}u_{11} = a_{11}, & (e_2) & l_{11}u_{12} = a_{12}, & (e_3) & l_{11}u_{13} = a_{13}, \\ (e_4) & l_{21}u_{11} = a_{21}, & (e_5) & l_{21}u_{12} + l_{22}u_{22} = a_{22}, & (e_6) & l_{21}u_{13} + l_{22}u_{23} = a_{23}, \\ (e_7) & l_{31}u_{11} = a_{31}, & (e_8) & l_{31}u_{12} + l_{32}u_{22} = a_{32}, & (e_9) & l_{31}u_{13} + l_{32}u_{23} \\ & & & & & & + l_{33}u_{33} = a_{33}. \end{array}$$

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 $A \in \mathbb{R}^{n \times n}$ con n arbitrario possiamo procedere nel modo seguente:

1. gli elementi di L e di U soddisfano il seguente sistema non lineare di equazioni

$$\sum_{r=1}^{\min(i,j)} l_{ir}u_{rj} = a_{ij}, \quad i, j = 1, \dots, n; \quad (5.12)$$

2. il sistema (5.12) 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 (5.12));
3. imponendo che gli n elementi diagonali di L siano pari a 1, (5.12) diviene un sistema quadrato determinato che può essere risolto con il seguente algoritmo, detto di Gauss: posto $A^{(1)} = A$ ovvero $a_{ij}^{(1)} = a_{ij}$ per $i, j = 1, \dots, n$, si calcoli

$$\begin{aligned}
 & \text{per } k = 1, \dots, n-1 \\
 & \quad \text{per } i = k+1, \dots, n \\
 & \quad l_{ik} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}, \\
 & \quad \text{per } j = k+1, \dots, n \\
 & \quad a_{ij}^{(k+1)} = a_{ij}^{(k)} - l_{ik} a_{kj}^{(k)}
 \end{aligned} \tag{5.13}$$

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 (5.13) non vengono calcolati esplicitamente gli elementi diagonali di L in quanto già sappiamo che sono pari a 1.

In virtù del fatto che gli elementi di L e U possono essere calcolati con l'algoritmo di Gauss (5.13), questa fattorizzazione LU è anche nota come fattorizzazione di Gauss; nel seguito ci riferiremo ad essa semplicemente come *fattorizzazione LU*. Il calcolo dei coefficienti dei fattori L ed U richiede circa $2n^3/3$ operazioni (si veda l'Esercizio 5.4).

Osservazione 5.1 Naturalmente non è necessario memorizzare tutte le matrici $A^{(k)}$ definite nell'algoritmo (5.13). 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 (i cosiddetti *moltiplicatori*), così come fatto nel Programma 5.1.

Di conseguenza, al k -esimo passo del processo gli elementi memorizzati al posto dei coefficienti originali della matrice A sono

$$\left[\begin{array}{cccccc|ccc}
 a_{11}^{(1)} & a_{12}^{(1)} & \cdots & \cdots & \cdots & a_{1n}^{(1)} & & & \\
 l_{21} & a_{22}^{(2)} & & & & a_{2n}^{(2)} & & & \\
 \vdots & \ddots & \ddots & & & \vdots & & & \\
 l_{k1} & \dots & l_{k,k-1} & a_{kk}^{(k)} & \dots & a_{kn}^{(k)} & & & \\
 \vdots & & \vdots & \vdots & & \vdots & & & \\
 l_{n1} & \dots & l_{n,k-1} & a_{nk}^{(k)} & \dots & a_{nn}^{(k)} & & &
 \end{array} \right],$$

dove la matrice nel riquadro è $A^{(k)}$.

Nella pratica questo algoritmo può essere realizzato usando una sola matrice in memoria, inizialmente posta uguale ad A e modificata ad ogni passo

$k \geq 2$ con i nuovi elementi $a_{ij}^{(k)}$, per $i, j \geq k + 1$ e con i moltiplicatori l_{ik} per $i \geq k + 1$. Si osservi che non è necessario memorizzare gli elementi diagonali l_{ii} , essendo sottinteso che essi sono uguali a 1. Si veda l'Osservazione 5.1. ■

Osservazione 5.2 (Il Metodo di Eliminazione di Gauss (MEG)) L'algoritmo di Gauss (5.13) può essere esteso al termine noto del sistema lineare $\mathbf{Ax} = \mathbf{b}$. Il metodo che ne deriva è: posto $\mathbf{A}^{(1)} = \mathbf{A}$ ovvero $a_{ij}^{(1)} = a_{ij}$ per $i, j = 1, \dots, n$; posto $\mathbf{b}^{(1)} = \mathbf{b}$ ovvero $b_i^{(1)} = b_i$ per $i = 1, \dots, n$,

$$\boxed{\begin{aligned} & \text{per } k = 1, \dots, n-1 \\ & \text{per } i = k+1, \dots, n \\ & l_{ik} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}, \\ & \text{per } j = k+1, \dots, n \\ & a_{ij}^{(k+1)} = a_{ij}^{(k)} - l_{ik} a_{kj}^{(k)} \\ & b_i^{(k+1)} = b_i^{(k)} - l_{ik} b_k^{(k)} \end{aligned}} \quad (5.14)$$

Esso è noto come *Metodo di Eliminazione di Gauss (MEG)*. Alla fine del processo, il sistema triangolare superiore $\mathbf{A}^{(n)}\mathbf{x} = \mathbf{b}^{(n)}$, che è equivalente a quello originale (ovvero ha la stessa soluzione), può essere risolto mediante l'algoritmo di sostituzioni all'indietro (5.10). ■

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 μm 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 5.2 (tratti da [CLW69, pag. 331]). Le pressioni parziali possono essere calcolate risolvendo il sistema (5.3) per $n = 7$ con la fattorizzazione LU.

Tabella 5.2. I coefficienti di sensitività per una particolare miscela gassosa

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 5.3. Tempo richiesto per risolvere un sistema lineare pieno di dimensione n con MEG. “f.p.” sta per “fuori portata”

n	Flops		
	10^9 (Giga)	10^{12} (Tera)	10^{15} (Peta)
10^2	$7 \cdot 10^{-4}$ sec	trascurabile	trascurabile
10^4	11 min	0.7 sec	$7 \cdot 10^{-4}$ sec
10^6	21 anni	7.7 mesi	11 min
10^8	f.p.	f.p.	21 anni

Otteniamo

`parzpress=`

```
0.6525
2.2038
0.3348
6.4344
2.9975
0.5505
25.6317
```

Tali valori conducono ad una stima della pressione totale (calcolabile con `sum(parzpress)`) che differisce dal valore misurato di 0.0252 μm di Hg. ■

Esempio 5.4 Consideriamo la seguente matrice di Vandermonde

$$A = (a_{ij}) \quad \text{con } a_{ij} = x_i^{n-j}, \quad i, j = 1, \dots, n, \quad (5.15)$$

dove gli x_i sono n valori distinti. Essa può essere costruita usando il comando `vander` di **MAT&OCT**. In Tabella 5.3 riportiamo il tempo richiesto per calcolare la fattorizzazione LU di A (che cresce come $2n^3/3$, si veda la Figura 5.6) su computer con una velocità nominale di 1 GigaFlops, 1 TeraFlops e 1 PetaFlops, rispettivamente. In Figura 5.6 riportiamo il numero di operazioni *floating-point* necessarie per realizzare la fattorizzazione LU della matrice di Vandermonde, come funzione della dimensione n della matrice. Questi valori sono stati ottenuti con il comando `flops` che era presente in vecchie versioni di MATLAB. ■

`vander`

La fattorizzazione LU è alla base dei seguenti comandi **MAT&OCT**:

- `[L,U]=lu(A)` le cui modalità di impiego verranno discusse nella Sezione 5.4;
- `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 $A\b$ (si veda la Sezione 5.8).

`inv`

Osservazione 5.3 (Calcolo del determinante) Tramite la fattorizzazione LU si può calcolare il determinante di una matrice A con un costo computazionale di $\mathcal{O}(n^3)$ operazioni, osservando che (si veda la Sezione 1.4)

$$\det(A) = \det(L)\det(U) = \prod_{k=1}^n u_{kk}.$$

Questa procedura è alla base del comando `det` di **MAT&OCT**. ■

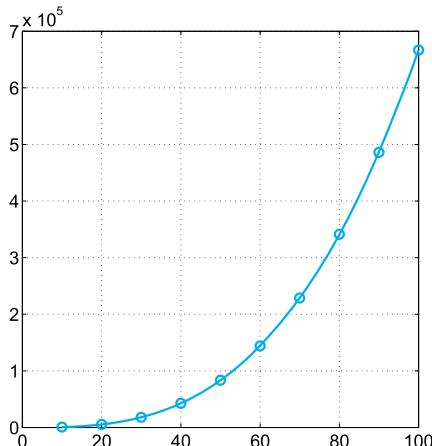


Figura 5.6. Il numero di operazioni *floating-point* necessarie per calcolare la fattorizzazione LU di A in corrispondenza di diversi valori della dimensione n della matrice, precisamente $n = 10, 20, \dots, 100$. La curva ottenuta è un polinomio in n di terzo grado che rappresenta l'approssimazione ai minimi quadrati di tali valori

Nel Programma 5.1 abbiamo implementato l'algoritmo (5.13). 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 5.1. lugauss: la fattorizzazione LU di Gauss

```
function A=lugauss(A)
%LUGAUSS Fattorizzazione LU senza pivoting
%   A = LUGAUSS(A) calcola la fattorizzazione
%   LU 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=lugauss(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

$$\mathbf{A} = \begin{bmatrix} 1 & 1-\varepsilon & 3 \\ 2 & 2 & 2 \\ 3 & 6 & 4 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 5-\varepsilon \\ 6 \\ 13 \end{bmatrix}, \quad \varepsilon \in \mathbb{R}, \quad (5.16)$$

è $\mathbf{x} = (1, 1, 1)^T$ (indipendente da ε).

Per $\varepsilon = 1$ la fattorizzazione LU di \mathbf{A} , ottenuta con il Programma 5.1, è

$$\mathbf{L} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 3 & 1 \end{bmatrix}, \quad \mathbf{U} = \begin{bmatrix} 1 & 0 & 3 \\ 0 & 2 & -4 \\ 0 & 0 & 7 \end{bmatrix}.$$

Se poniamo $\varepsilon = 0$, anche se \mathbf{A} è non singolare, la fattorizzazione LU non può essere calcolata in quanto l'algoritmo (5.13) comporta una divisione per 0. ■

L'esempio precedente mostra che, sfortunatamente, la fattorizzazione LU potrebbe non esistere anche se la matrice \mathbf{A} è non singolare. In tal senso si può dimostrare il seguente risultato:

Proposizione 5.1 *Data una matrice $\mathbf{A} \in \mathbb{R}^{n \times n}$, la sua fattorizzazione LU esiste ed è unica se e solo se le sottomatrici principali \mathbf{A}_i di \mathbf{A} di ordine $i = 1, \dots, n-1$ (cioè quelle ottenute limitando \mathbf{A} alle sole prime i righe e colonne) sono non singolari. (Tale risultato vale anche per $\mathbf{A} \in \mathbb{C}^{n \times n}$ [Zha99, Sez. 3.2]).*

Tornando all'Esempio 5.6, possiamo in effetti notare che quando $\varepsilon = 0$ la seconda sottomatrice \mathbf{A}_2 di \mathbf{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 a dominanza diagonale stretta. Una matrice è detta *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 disuguaglianze possiamo sostituire il segno \geq con quello $>$ diremo che A è a dominanza diagonale *stretta* (per righe o per colonne, rispettivamente). Questo risultato vale anche per matrici $A \in \mathbb{C}^{n \times n}$ (si veda [GI04]);

2. le matrici reali simmetriche e definite positive. Ricordiamo che una matrice $A \in \mathbb{R}^{n \times n}$ è *definita positiva* se

$$\forall \mathbf{x} \in \mathbb{R}^n \quad \text{con } \mathbf{x} \neq \mathbf{0}, \quad \mathbf{x}^T A \mathbf{x} > 0$$

e *semi definita positiva* se

$$\forall \mathbf{x} \in \mathbb{R}^n, \quad \mathbf{x}^T A \mathbf{x} \geq 0;$$

3. le matrici complesse $A \in \mathbb{C}^{n \times n}$ definite positive, ovvero tali che

$$\forall \mathbf{x} \in \mathbb{C}^n \quad \text{con } \mathbf{x} \neq \mathbf{0}, \quad \mathbf{x}^H A \mathbf{x} > 0;$$

osserviamo che tali matrici sono necessariamente hermitiane (si veda [Zha99, Sez. 3.2]).

Se $A \in \mathbb{R}^{n \times n}$ è simmetrica e definita positiva, è inoltre possibile trovarne una fattorizzazione speciale

$$A = R^T R \tag{5.17}$$

essendo R una matrice triangolare superiore con elementi positivi sulla diagonale. Tale fattorizzazione è unica.

La (5.17) è nota come *fattorizzazione di Cholesky*. Il calcolo di R richiede circa $n^3/3$ operazioni (cioè la metà di quelle richieste per calcolare le due matrici della fattorizzazione LU). Si noti inoltre che per la simmetria di A , se ne può memorizzare la sola parte triangolare superiore ed R potrà essere memorizzata nella stessa area di memoria.

Gli elementi di R possono essere calcolati tramite il seguente algoritmo: poniamo $r_{11} = \sqrt{a_{11}}$ e, per $j = 2, \dots, n$,

$$\boxed{\begin{aligned} r_{ij} &= \frac{1}{r_{ii}} \left(a_{ij} - \sum_{k=1}^{i-1} r_{ki} r_{kj} \right), \quad i = 1, \dots, j-1 \\ r_{jj} &= \sqrt{a_{jj} - \sum_{k=1}^{j-1} r_{kj}^2} \end{aligned}} \tag{5.18}$$

La fattorizzazione di Cholesky è richiamabile in `MAT&OCT` con la sintassi `R=chol(A)`. Nel caso in cui sia $A \in \mathbb{C}^{n \times n}$ definita positiva, la formula (5.17) diventa $A=R^H R$, essendo R^H la trasposta coniugata di R .

Esempio 5.7 (Rete di capillari) Consideriamo il problema 5.4. Poiché la matrice $-A$ del problema 5.4 è simmetrica e definita postiva, il sistema $-Ap = -b$ può essere risolto ricorrendo alla fattorizzazione di Cholesky. La soluzione ottenuta è data dal seguente vettore

$$\mathbf{p} = [12.46, 3.07, 3.07, 0.73, 0.73, 0.73, 0.15, 0.15, 0.15, 0.15, \\ 0.15, 0.15, 0.15, 0.15]^T.$$

Corrispondentemente, attraverso le relazioni (5.5), troviamo i seguenti valori di portata:

$$\begin{aligned} Q_1 &= 1.88 \\ Q_{2,3} &= 0.94 \\ Q_{4,\dots,7} &= 0.47 \\ Q_{8,\dots,15} &= 0.23 \\ Q_{16,\dots,31} &= 0.12. \end{aligned}$$

La matrice A ha una struttura speciale: si veda per un esempio la Figura 5.7 corrispondente ad un letto capillare con 8 livelli di biforcazione. I punti colorati sono quelli corrispondenti alla presenza di elementi non nulli. Su ogni riga esistono al più 3 elementi non nulli. Si tratta di una matrice a banda e sparsa (come definiremo alla fine di questo esempio), essendo solo 379 gli elementi non nulli su un totale di $(127)^2 = 16129$ elementi della matrice. Osserviamo che la fattorizzazione di Cholesky di $-A$ genera il riempimento della banda, (noto anche come fenomeno del *fill-in*, si veda la Sez. 5.4.1) come risulta dalla Figura 5.7 (a destra) in cui viene riportata la struttura del fattore triangolare superiore R . La riduzione di questo fenomeno è possibile grazie all'uso di algoritmi di riordinamento della matrice di partenza. Un esempio è

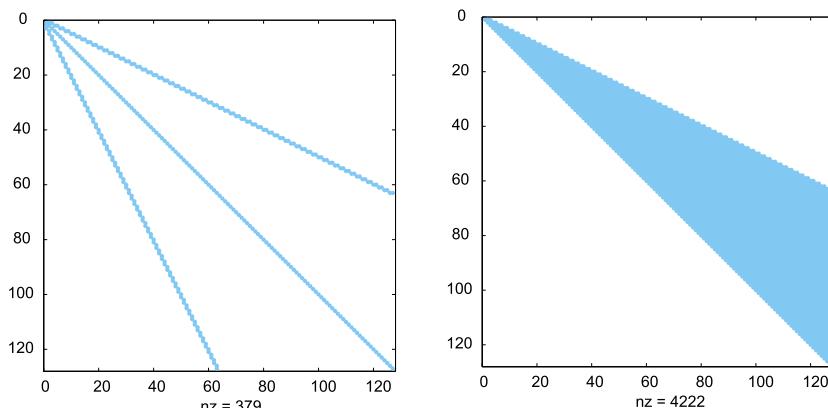


Figura 5.7. Pattern delle matrici A (a sinistra) e R (a destra) dell'Esempio 5.7

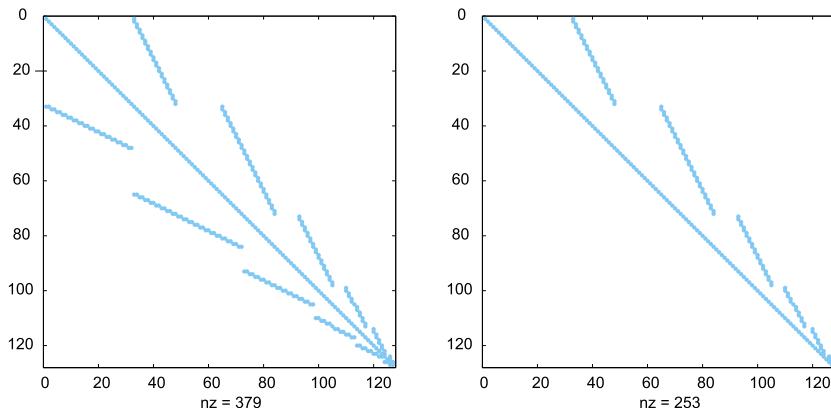


Figura 5.8. Pattern delle matrici A (a sinistra) e R (a destra) dell’Esempio 5.7 dopo il riordino

riportato in Figura 5.8, in cui si evidenzia a sinistra la matrice A riordinata, e a destra il nuovo fattore R che ne deriva. Per una discussione sulle principali tecniche di riordinamento rinviamo il lettore interessato a [QSS07, Sez. 3.9]. ■

Una matrice quadrata di dimensione n è detta *sparsa* se ha un numero di elementi non nulli dell’ordine di n (e non di n^2). Inoltre chiamiamo *pattern* di una matrice sparsa l’insieme dei suoi elementi non nulli. Ad esempio, in Figura 5.7 sono rappresentati i *pattern* delle matrici A e R dell’Esempio 5.7, ottenuti rispettivamente con le istruzioni `spy(A)` e `spy(R)`.

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.

Delle matrici sparse, o a banda, di grandi dimensioni, è naturale memorizzare i soli elementi non nulli. Ciò è possibile generando la matrice in questione con i comandi `sparse` o `spdiags` di **MAT&OCT**. Ad esempio, per inizializzare a zero uno *sparse-array* (è un tipo di variabile **MAT&OCT**) di n righe ed m colonne, basta digitare il comando

`A=sparse(n,m)`

mentre la matrice A di dimensione $n = 25$ di elementi

$$\begin{aligned} a_{ii} &= 1 && \text{per } i = 1, \dots, n, \\ a_{1j} &= 1 && \text{per } j = 1, \dots, n, \\ a_{i1} &= 1 && \text{per } i = 1, \dots, n, \\ a_{ij} &= 0 && \text{altrimenti} \end{aligned} \tag{5.19}$$

può essere definita con le seguenti istruzioni:

```
n=25; e=ones(n,1);
A=spdiags(e,0,n,n);
A(1,:)=e'; A(:,1)=e;
```

Il comando `spdiags` inizializza una matrice di n righe ed n colonne nel formato *sparse-array*, posizionando il vettore colonna `e` sulla diagonale principale (di indice 0), quindi le istruzioni successive (valide per tutti gli array `MAT&OCT`) aggiornano la prima riga e la prima colonna di `A`.

Quando un sistema lineare è risolto con il comando `\`, `MAT&OCT` è in grado di riconoscere il *pattern* della matrice `e`, e, in particolare, se essa è memorizzata nel formato sparse-array; di conseguenza `MAT&OCT` seleziona l'algoritmo risolutivo più appropriato come vedremo nella Sezione 5.8.

Si vedano gli Esercizi 5.1–5.5.



5.4 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

$$\left[\begin{array}{ccc|c} 1 & 1 & 3 & \\ \hline 2 & 0 & -4 & \\ 3 & 3 & -5 & \end{array} \right]. \quad (5.20)$$

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

$$\left[\begin{array}{ccc|c} 1 & 1 & 3 & \\ \hline 3 & 3 & -5 & \\ 2 & 0 & -4 & \end{array} \right]$$

e la fattorizzazione avrebbe potuto terminare senza incontrare divisioni per 0.

Quindi, *permuto* (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 (5.20) che presenta l'elemento pivot nullo (in posizione (2, 2)). Osservato che l'elemento in posizione (3, 2) è non nullo, scambiamo la terza 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 restituisce la matrice A di partenza a meno di una permutazione fra le righe. Precisamente,

$$\boxed{PA = LU} \quad (5.21)$$

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 r e s di P. Corrispondentemente, dovremo ora risolvere i seguenti sistemi triangolari

$$\boxed{Ly = Pb, \quad Ux = y.} \quad (5.22)$$

Dalla seconda equazione nella (5.13) 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 5.8 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 5.1 non si generano elementi *pivot* nulli. Nonostante ciò, i fattori L ed U calcolati sono assai inaccurati, come si verifica calcolando $A - LU$ (che in aritmetica esatta sarebbe uguale alla matrice nulla). Si ottiene

$$A - LU = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 4 \end{bmatrix}.$$

■

È pertanto consigliabile eseguire il *pivoting* ad ogni passo della procedura di fattorizzazione, cercando fra tutti i *pivot* disponibili $a_{ik}^{(k)}$ con

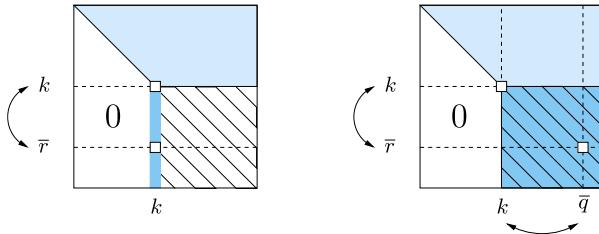


Figura 5.9. Pivotazione per righe (a sinistra) e totale (a destra). In azzurro più intenso la sottomatrice in cui cercare il *pivot* al passo k

$i = k, \dots, n$, quello di modulo massimo (si veda la Figura 5.9, a sinistra). L'algoritmo (5.13) con il *pivoting* per righe, eseguito ad ogni passo, diventa allora: posto $A^{(1)} = A$ e $P=I$, si calcoli

per $k = 1, \dots, n - 1$,
 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}
 sia in A che in P ,

per $i = k + 1, \dots, n$

$$l_{ik} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}},$$

per $j = k + 1, \dots, n$

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - l_{ik} a_{kj}^{(k)}$$

(5.23)

Come già osservato per l'algoritmo (5.13) senza permutazioni, un solo spazio di memoria è sufficiente a contenere gli elementi variabili $a_{ij}^{(k)}$ nonché i moltiplicatori l_{ik} . Di conseguenza, per ogni k , la stessa permutazione effettuata su A e P agisce anche sui moltiplicatori.

La funzione `lu` di **MAT&OCT**, cui abbiamo accennato in precedenza, calcola la fattorizzazione LU con *pivoting* per righe. La sua sintassi completa è infatti `[L,U,P]=lu(A)`, dove P è la matrice di permutazione. Quando richiamata con la sintassi abbreviata `[L,U]=lu(A)`, essa produce una matrice L che è uguale a $P*M$, 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. In particolare, quando la matrice è memorizzata in formato **sparse**, la pivotazione per righe è effettuata solo quando viene generato un elemento *pivot* nullo (o molto piccolo).

Diciamo che si opera una *pivotazione totale* quando la ricerca del *pivot* è estesa alla sottomatrice $A^{(k)}$ costituita dagli elementi $a_{ij}^{(k)}$ con $i, j = k, \dots, n$ (si veda la Figura 5.9, a destra). Essa coinvolge non solo

le righe, ma anche le colonne del sistema e conduce alla costruzione di due matrici di permutazione P e Q , una sulle righe, l'altra sulle colonne tali che

$$\boxed{PAQ = LU} \quad (5.24)$$

Poiché

$$Ax = b \Leftrightarrow \underbrace{PAQ}_{LU} \underbrace{Q^{-1}x^*}_{x^*} = Pb,$$

la soluzione del sistema $Ax = b$ è quindi ottenuta attraverso la risoluzione di due sistemi triangolari e di una permutazione come segue

$$\boxed{Ly = Pb \quad Ux^* = y \quad x = Qx^*} \quad (5.25)$$

Il comando `MATGOC lu` implementa la pivotazione totale quando la matrice A in input è in formato sparse-array (quindi generata con i comandi `sparse` o `spdiags`) e vengono specificati 4 parametri di output. Più precisamente il comando di chiamata è `[L,U,P,Q]=lu(A)`.

Dal punto di vista computazionale la pivotazione totale ha un costo superiore rispetto a quella parziale in quanto ad ogni passo della fattorizzazione devono essere svolti molti più confronti. Tuttavia essa può apportare dei vantaggi in termini di risparmio di memoria e di stabilità come vedremo nelle prossime sezioni.

5.4.1 Il *fill-in* di una matrice

Consideriamo ora una matrice A con molti elementi nulli. Non è detto che le matrici L ed U ottenute dalla fattorizzazione mantengano la struttura del corrispondente triangolo della matrice A iniziale, anzi il processo di fattorizzazione tende a riempire le matrici L ed U generando il cosiddetto fenomeno del *fill-in* (o riempimento) che dipende fortemente dalla struttura e dal valore dei singoli elementi non nulli della matrice A .

Abbiamo già incontrato un caso di *fill-in* nell'Esempio 5.7 (si veda la Figura 5.7), mentre in Figura 5.10 possiamo osservare il *pattern* delle matrici A , L ed U quando A è la matrice definita in (5.19). Un altro esempio di *fill-in* è mostrato in Figura 5.11, in questo caso gli elementi non nulli della prima riga e della prima colonna di A inducono un riempimento totale delle corrispondenti colonne in U e righe in L , rispettivamente, mentre gli elementi non nulli nelle sopra e sotto diagonali di A comportano un riempimento delle diagonali superiori di U ed inferiori di L comprese tra quella principale e quelle non nulle di A .

Per ovviare al *fill-in* della matrice si possono adottare tecniche di rordinamento (già citate nell'Esempio 5.7) che permutano righe e colonne della matrice prima di realizzare la fattorizzazione.

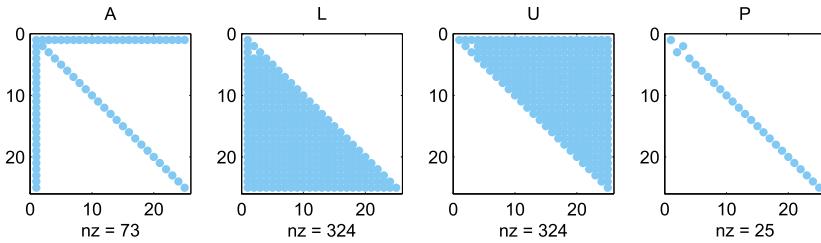


Figura 5.10. *Fill-in* della matrice A definita in (5.19)

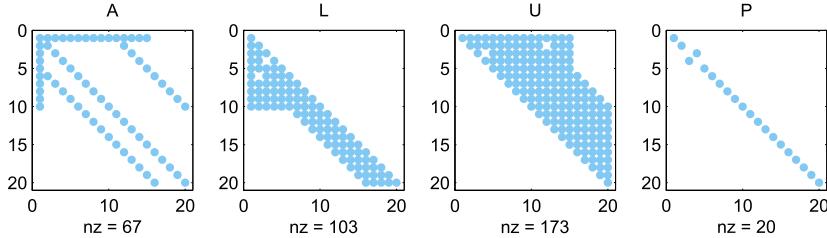


Figura 5.11. Esempio di *fill-in* per una matrice A il cui profilo è indicato nella prima immagine da sinistra

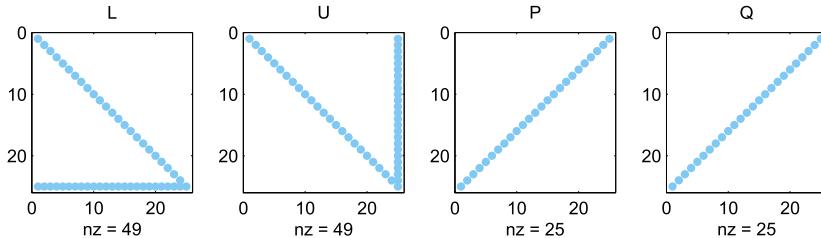


Figura 5.12. Le matrici L, U, P e Q della fattorizzazione con pivotazione totale della matrice A definita in (5.19)

In taluni casi tuttavia, la sola pivotazione totale permette di raggiungere lo stesso obiettivo. Ad esempio in Figura 5.12 sono mostrati i *pattern* delle matrici L, U, P e Q ottenute dalla fattorizzazione con pivotazione totale della matrice A definita in (5.19), ora non si è verificato il *fill-in* delle matrici, al costo però di invertire l'ordine di tutte le righe e di tutte le colonne della matrice, come deduciamo dal *pattern* di P e Q.

In Figura 5.13 riportiamo le matrici L, U, P e Q ottenute dalla fattorizzazione LU con pivotazione totale della matrice A di Figura 5.11. Anche in questo caso la pivotazione totale ha agito positivamente in quanto il *fill-in* è molto più contenuto di quello che si genererebbe con la pivotazione per righe.

Si vedano gli Esercizi 5.6–5.8.



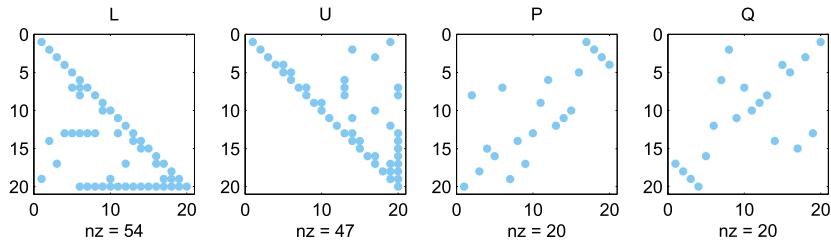


Figura 5.13. Le matrici L, U, P e Q della fattorizzazione con pivotazione totale della matrice A di Figura 5.11

5.5 Quanto è accurata la risoluzione di un sistema lineare?

Come abbiamo già avuto modo di notare nell’Esempio 5.8, a causa degli errori di arrotondamento il prodotto LU non ritorna 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. (In teoria il *pivoting* totale risulta migliore di quello parziale, ma l’esperienza pratica mostra che in genere anche solo quello parziale produce buoni risultati [Hig02, Sez. 9.3].

Purtroppo ciò non sempre è vero, come mostra l’esempio seguente.

Esempio 5.9 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 LU di A_n con *pivoting* per righe. Risolviamo quindi i sistemi lineari associati (5.22), indicando con $\hat{\mathbf{x}}_n$ la soluzione calcolata. Nella Figura 5.14 riportiamo (in scala logaritmica) l’andamento dell’errore relativo al variare di n ,

$$E_n = \|\mathbf{x}_n - \hat{\mathbf{x}}_n\| / \|\mathbf{x}_n\|, \quad (5.26)$$

avendo indicato con $\|\cdot\|$ la norma euclidea introdotta nella Sezione 1.4.1. 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. Lo stesso tipo di risultato si ottiene con il *pivoting* totale. ■

Sulla base della precedente osservazione, si può allora ipotizzare che quando si risolve numericamente il sistema lineare $A \mathbf{x} = \mathbf{b}$ si trovi la soluzione *esatta* $\hat{\mathbf{x}}$ di un sistema *perturbato* della forma

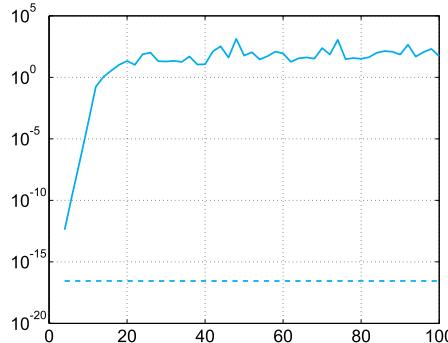


Figura 5.14. Andamento di E_n (in linea continua) e di $\max_{i,j=1,\dots,n} |r_{ij}|$ (in linea tratteggiata) in scala logaritmica, per il sistema di Hilbert dell’Esempio 5.9. Gli r_{ij} sono i coefficienti della matrice R_n

$$(A + \delta A)\hat{x} = b + \delta b, \quad (5.27)$$

dove δA e δb sono rispettivamente una matrice ed un vettore di perturbazione che dipendono dallo specifico metodo numerico impiegato nella risoluzione del sistema.

Incominciamo ad analizzare il caso in cui $\delta A = 0$ e $\delta b \neq \mathbf{0}$, in quanto più semplice del caso generale.

Supponiamo inoltre per semplicità che $A \in \mathbb{R}^{n \times n}$ sia simmetrica e definita positiva.

Confrontando (5.1) e (5.27) troviamo $x - \hat{x} = -A^{-1}\delta b$, e dunque

$$\|x - \hat{x}\| = \|A^{-1}\delta b\|. \quad (5.28)$$

Per trovare un limite superiore del termine di destra della (5.28) procediamo nel modo seguente. Essendo A simmetrica e definita positiva, esiste una base ortonormale di \mathbb{R}^n formata dagli autovettori $\{\mathbf{v}_i\}_{i=1}^n$ di A (si veda ad esempio [QSSG14, Cap. 1]). Questo comporta che

$$A\mathbf{v}_i = \lambda_i \mathbf{v}_i, \quad i = 1, \dots, n, \quad \mathbf{v}_i^T \mathbf{v}_j = \delta_{ij}, \quad i, j = 1, \dots, n,$$

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

$$\begin{aligned} \|Aw\|^2 &= (Aw)^T (Aw) \\ &= [w_1(A\mathbf{v}_1)^T + \dots + w_n(A\mathbf{v}_n)^T][w_1 A \mathbf{v}_1 + \dots + w_n A \mathbf{v}_n] \\ &= (\lambda_1 w_1 \mathbf{v}_1^T + \dots + \lambda_n w_n \mathbf{v}_n^T)(\lambda_1 w_1 \mathbf{v}_1 + \dots + \lambda_n w_n \mathbf{v}_n) \\ &= \sum_{i=1}^n \lambda_i^2 w_i^2. \end{aligned}$$

Denotiamo con λ_{\max} il più grande autovalore di A . Poiché $\|w\|^2 = \sum_{i=1}^n w_i^2$, concludiamo che

$$\|Aw\| \leq \lambda_{\max}\|w\|, \quad \forall w \in \mathbb{R}^n. \quad (5.29)$$

In modo analogo, otteniamo

$$\|A^{-1}w\| \leq \frac{1}{\lambda_{\min}}\|w\|,$$

in quanto gli autovalori di A^{-1} sono i reciproci di quelli di A . Questa diseguaglianza consente di dedurre dalla (5.28) che

$$\frac{\|\mathbf{x} - \hat{\mathbf{x}}\|}{\|\mathbf{x}\|} \leq \frac{1}{\lambda_{\min}} \frac{\|\delta\mathbf{b}\|}{\|\mathbf{x}\|}. \quad (5.30)$$

Usando nuovamente la (5.29) e ricordando che $Ax = \mathbf{b}$, otteniamo infine

$$\frac{\|\mathbf{x} - \hat{\mathbf{x}}\|}{\|\mathbf{x}\|} \leq \frac{\lambda_{\max}}{\lambda_{\min}} \frac{\|\delta\mathbf{b}\|}{\|\mathbf{b}\|} \quad (5.31)$$

Quindi l'errore relativo sulla soluzione dipende dall'errore relativo sui dati attraverso la seguente costante (≥ 1)

$$K(A) = \frac{\lambda_{\max}}{\lambda_{\min}} \quad (5.32)$$

Essa è detta *numero di condizionamento (spettrale) della matrice A*. Ricordiamo che quest'ultima deve essere simmetrica e definita positiva.

Una dimostrazione più elaborata avrebbe condotto ad un risultato più generale nel caso in cui A fosse stata una matrice simmetrica e definita positiva e δA una matrice non nulla simmetrica e definita positiva e tale che $\lambda_{\max}(\delta A) < \lambda_{\min}(A)$. In questo caso si può infatti dimostrare che

$$\frac{\|\mathbf{x} - \hat{\mathbf{x}}\|}{\|\mathbf{x}\|} \leq \frac{K(A)}{1 - \lambda_{\max}(\delta A)/\lambda_{\min}(A)} \left(\frac{\lambda_{\max}(\delta A)}{\lambda_{\max}(A)} + \frac{\|\delta\mathbf{b}\|}{\|\mathbf{b}\|} \right) \quad (5.33)$$

Infine, se le matrici A e δA non sono simmetriche e definite positive e δA è tale che $\|\delta A\|_2 \|A^{-1}\|_2 < 1$, vale la seguente stima:

$$\frac{\|\mathbf{x} - \hat{\mathbf{x}}\|}{\|\mathbf{x}\|} \leq \frac{K_2(A)}{1 - K_2(A)\|\delta A\|_2/\|A\|_2} \left(\frac{\|\delta A\|_2}{\|A\|_2} + \frac{\|\delta\mathbf{b}\|}{\|\mathbf{b}\|} \right) \quad (5.34)$$

essendo $\|A\|_2 = \sqrt{\lambda_{\max}(A^T A)}$ e

$$K_2(A) = \|A\|_2 \cdot \|A^{-1}\|_2 \quad (5.35)$$

il *numero di condizionamento in norma 2*. Osserviamo che $K_2(A)$ restituisce il numero di condizionamento spettrale (5.32) quando A è simmetrica e definita positiva.

Se $K_2(A)$ (o $K(A)$) è “piccolo”, cioè dell’ordine dell’unità, la matrice A viene detta *ben condizionata* ed a piccoli errori sui dati corrispondranno 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 sulla soluzione.

Osservazione 5.4 $K_2(A)$ può essere calcolato in **MAT&OCT** con il comando **cond(A)** (o **cond(A, 2)**). Quando A è una matrice sparsa, il comando **cond** non può essere richiamato, a meno di non trasformare prima la matrice da formato sparso a formato pieno con il comando **A1=full(A)**. In alternativa, possiamo richiamare il comando **condeest(A)** che fornisce una stima (dal basso) del numero di condizionamento $K_1(A) = \|A\|_1 \cdot \|A^{-1}\|_1$, essendo $\|A\|_1 = \max_j \sum_{i=1}^n |a_{ij}|$ la cosiddetta *norma 1* di A . È bene osservare che in genere $K_1(A) \neq K_2(A)$, quindi non dobbiamo aspettarci che **condeest(A)** sia una stima di **cond(full(A))**.

Ulteriori definizioni di numero di condizionamento sono disponibili per matrici non simmetriche, si veda ad esempio [QSSG14, Cap. 3]. ■

Esempio 5.10 Per la matrice di Hilbert introdotta nell’Esempio 5.9, $K(A_n)$ è una funzione rapidamente crescente di n . Abbiamo $K(A_4) > 15000$, mentre se $n > 13$, **MAT&OCT** segnala che la matrice ha un numero di condizionamento così elevato da essere ritenuta singolare. In realtà, $K(A_n)$ cresce esponenzialmente rispetto ad n , $K(A_n) \simeq e^{3.5n}$ (si veda [Hig02]). Non dobbiamo perciò sorprenderci dei cattivi risultati ottenuti nell’Esempio 5.9. ■

La diseguaglianza (5.31) può essere riformulata introducendo il *residuo* \mathbf{r} :

$$\mathbf{r} = \mathbf{b} - A\hat{\mathbf{x}}. \quad (5.36)$$

Evidentemente se $\hat{\mathbf{x}}$ fosse la soluzione esatta, il residuo sarebbe il vettore nullo. Di conseguenza, \mathbf{r} può essere ritenuto uno *stimatore* dell’errore commesso $\mathbf{x} - \hat{\mathbf{x}}$. La sua efficacia come stimatore dell’errore dipende dalla grandezza del numero di condizionamento di A . Infatti, osservando che $\delta\mathbf{b} = A(\hat{\mathbf{x}} - \mathbf{x}) = A\hat{\mathbf{x}} - \mathbf{b} = -\mathbf{r}$, dalla stima (5.31) si ricava

$$\frac{\|\mathbf{x} - \hat{\mathbf{x}}\|}{\|\mathbf{x}\|} \leq K_2(A) \frac{\|\mathbf{r}\|}{\|\mathbf{b}\|} \quad (5.37)$$

Quindi se $K_2(A)$ è “piccolo”, avremo la certezza che l’errore sarà piccolo quando lo è il residuo, mentre ciò non è necessariamente vero quando $K_2(A)$ è “grande”.

Esempio 5.11 Se calcoliamo la norma del residuo per i sistemi dell'Esempio 5.9, 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.

5.6 Come risolvere un sistema tridiagonale

In molte applicazioni (si veda ad esempio il Capitolo 9) 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.

Se la fattorizzazione LU di A esiste, i fattori L e U sono due matrici *bidiagonali* (inferiore e superiore, rispettivamente) della forma

$$L = \begin{bmatrix} 1 & & 0 \\ \beta_2 & 1 & \\ \ddots & \ddots & \\ 0 & \beta_n & 1 \end{bmatrix}, \quad U = \begin{bmatrix} \alpha_1 & c_1 & & 0 \\ & \alpha_2 & \ddots & \\ & & \ddots & c_{n-1} \\ 0 & & & \alpha_n \end{bmatrix}.$$

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 per il calcolo dei fattori L e U

$$\alpha_1 = a_1, \quad \beta_i = \frac{e_i}{\alpha_{i-1}}, \quad \alpha_i = a_i - \beta_i c_{i-1}, \quad i = 2, \dots, n. \quad (5.38)$$

Grazie ad esse, possiamo risolvere i due sistemi bidiagonali $Ly = b$ e $Ux = y$, ottenendo le seguenti formule

$$(Ly = b) \quad y_1 = b_1, \quad y_i = b_i - \beta_i y_{i-1}, \quad i = 2, \dots, n, \quad (5.39)$$

$$(Ux = y) \quad x_n = \frac{y_n}{\alpha_n}, \quad x_i = \frac{y_i - c_i x_{i+1}}{\alpha_i}, \quad i = n-1, \dots, 1. \quad (5.40)$$

Questa procedura è nota come *algoritmo di Thomas* e consente di risolvere un sistema tridiagonale con un costo dell'ordine di n operazioni.

Il comando `spdiags` di `MAT&OCT` 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.

Se A è una matrice tridiagonale generata in modalità sparsa, l'algoritmo risolutivo selezionato dal comando `\` di `MATLAB` sarà quello di Thomas. (Si veda anche la Sezione 5.8 per una presentazione più generale del comando `\`.)

5.7 Sistemi sovradeterminati

Un sistema lineare $A\mathbf{x} = \mathbf{b}$ 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 in senso classico, a meno che il termine noto \mathbf{b} non sia un elemento del $\text{range}(A)$, definito come

$$\text{range}(A) = \{\mathbf{z} \in \mathbb{R}^m : \mathbf{z} = A\mathbf{y} \text{ per } \mathbf{y} \in \mathbb{R}^n\}. \quad (5.41)$$

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

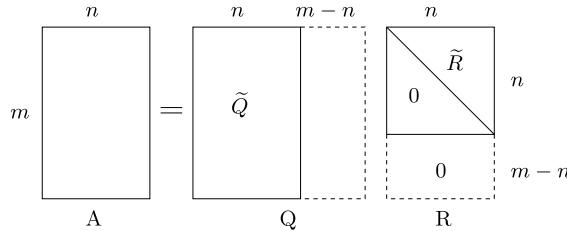
$$\varPhi(\mathbf{x}^*) = \|A\mathbf{x}^* - \mathbf{b}\|^2 \leq \|A\mathbf{y} - \mathbf{b}\|^2 = \varPhi(\mathbf{y}) \quad \forall \mathbf{y} \in \mathbb{R}^n. \quad (5.42)$$

Il vettore \mathbf{x}^* quando esiste è detto *soluzione nel senso dei minimi quadrati* del sistema sovradeterminato $A\mathbf{x} = \mathbf{b}$.

In modo del tutto analogo a quanto fatto nella Sezione 3.6, si può trovare la soluzione di (5.42) imponendo che il gradiente della funzione \varPhi sia nullo in \mathbf{x}^* . Con calcoli del tutto simili si trova che \mathbf{x}^* è di fatto la soluzione del sistema lineare $n \times n$

$$A^T A \mathbf{x}^* = A^T \mathbf{b} \quad (5.43)$$

detto sistema delle *equazioni normali*. Il sistema (5.43) è non singolare se A è a *rango pieno* (cioè se $\text{rank}(A) = \min(m,n)$, dove il rango di A $\text{rank}(A)$ 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 (5.17) 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 fattorizzazione QR oppure la decomposizione in valori singolari di A .

**Figura 5.15.** La fattorizzazione QR

Incominciamo dalla prima. Ogni matrice $A \in \mathbb{R}^{m \times n}$ a rango pieno, con $m \geq n$, ammette un'unica *fattorizzazione QR*

$$A = QR \quad (5.44)$$

nella quale la matrice $Q \in \mathbb{R}^{m \times m}$ è ortogonale (cioè tale che $Q^T Q = I$) mentre $R \in \mathbb{R}^{m \times n}$ è una matrice con tutti gli elementi sotto la diagonale principale nulli e tutti quelli diagonali non nulli. Si veda la Figura 5.15.

Si può verificare che vale anche $A = \tilde{Q}\tilde{R}$, essendo $\tilde{Q} = Q(1:m, 1:n)$ e $\tilde{R} = R(1:n, 1:n)$ le due sottomatrici evidenziate in Figura 5.15; \tilde{Q} ha vettori colonna ortonormali, mentre \tilde{R} è una matrice triangolare superiore (coincidente con il fattore R della fattorizzazione di Cholesky della matrice $A^T A$). Essendo \tilde{R} non singolare, l'unica soluzione di (5.43) è data da

$$\mathbf{x}^* = \tilde{R}^{-1} \tilde{Q}^T \mathbf{b}. \quad (5.45)$$

Introduciamo ora la decomposizione in valori singolari di una matrice. Si può dimostrare che per ogni matrice rettangolare $A \in \mathbb{C}^{m \times n}$ esistono due matrici $U \in \mathbb{C}^{m \times m}$ e $V \in \mathbb{C}^{n \times n}$ unitarie tali che

$$U^H A V = \Sigma = \text{diag}(\sigma_1, \dots, \sigma_p) \in \mathbb{R}^{m \times n} \quad (5.46)$$

dove $p = \min(m, n)$ e $\sigma_1 \geq \dots \geq \sigma_p \geq 0$. Una matrice U è detta *unitaria* se $U^H U = U U^H = I$. La (5.46) è detta *decomposizione in valori singolari* (o *singular value decomposition*, in breve *SVD*) di A ed i numeri σ_i sono detti i *valori singolari* di A . Si ha che $\sigma_i = \sqrt{\lambda_i(A^H A)}$, essendo $\lambda_i(A^H A)$ gli autovalori, reali e positivi, della matrice $A^H A$.

Osserviamo che, se A è una matrice a coefficienti reali, allora lo sono anche U e V . Inoltre U e V sono matrici *ortogonal*e e U^H non è altro che U^T .

Se operiamo la decomposizione in valori singolari (5.46) sulla matrice A del sistema (5.43), essendo U ortogonale, abbiamo $A^T A = V \Sigma^T \Sigma V^T$ e quindi il sistema delle equazioni normali (5.43) è equivalente al sistema

$$V \Sigma^T \Sigma V^T \mathbf{x}^* = V \Sigma^T U^T \mathbf{b}. \quad (5.47)$$

Osserviamo che V è ortogonale e che la matrice $\Sigma^T \Sigma$ è una matrice quadrata diagonale non singolare i cui elementi diagonali sono i quadrati dei valori singolari di A . Allora, moltiplicando a sinistra entrambi i termini del sistema (5.47) per $V(\Sigma^T \Sigma)^{-1}V^T$, otteniamo

$$\mathbf{x}^* = V\Sigma^{\dagger}U^T\mathbf{b} = A^{\dagger}\mathbf{b}, \quad (5.48)$$

dove $\Sigma^{\dagger} = (\Sigma^T \Sigma)^{-1} \Sigma^T = \text{diag}(1/\sigma_1, \dots, 1/\sigma_n, 0, \dots, 0)$, mentre $A^{\dagger} = (A^T A)^{-1} A^T = V\Sigma^{\dagger}U^T$. Quest'ultima è detta *pseudoinversa* di A .

Dalla formula (5.48) si evince quindi che il calcolo dei valori singolari di A e delle matrici U e V permette, con semplici operazioni aggiuntive, di trovare la soluzione del sistema delle equazioni normali (5.43).

MAT&OCT mette a disposizione due *function*: **svd** e **svds**. La prima **svd** calcola tutti i valori singolari di una matrice e la seconda soltanto i k più grandi, con k da precisare in ingresso (per default $k=6$). Rimandiamo a [ABB+99] per una presentazione esaustiva dell'algoritmo che viene utilizzato.

Esempio 5.12 Consideriamo un approccio alternativo al problema di trovare la retta di regressione $\epsilon(\sigma) = a_1\sigma + a_0$ (si veda la Sezione 3.6) per i dati del Problema 3.3. Usando i dati della Tabella 3.2 e imponendo le condizioni di interpolazione otteniamo il sistema sovradeterminato $A\mathbf{a} = \mathbf{b}$, dove $\mathbf{a} = (a_1, a_0)^T$ e

$$A = \begin{bmatrix} 0 & 1 \\ 0.06 & 1 \\ 0.14 & 1 \\ 0.25 & 1 \\ 0.31 & 1 \\ 0.47 & 1 \\ 0.60 & 1 \\ 0.70 & 1 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 0 \\ 0.08 \\ 0.14 \\ 0.20 \\ 0.23 \\ 0.25 \\ 0.28 \\ 0.29 \end{bmatrix}.$$

Per calcolarne la soluzione nel senso dei minimi quadrati usiamo le seguenti istruzioni **MAT&OCT**:

```
[Q,R]=qr(A);
Qt=Q(:,1:2);
Rt=R(1:2,:);
xstar = Rt \ (Qt'*b)

xstar =
    0.3741
    0.0654
```

Questi sono esattamente i coefficienti della retta di regressione calcolata nell'Esempio 3.13. Si noti che questa procedura è automaticamente implementata nel comando \: l'istruzione $xstar = A\b$ produce infatti il medesimo vettore $xstar$ calcolato con le formule (5.44) e (5.45). ■

5.8 Cosa si nasconde dietro al comando \

È importante sapere che il comando `\` richiama uno specifico algoritmo a seconda delle caratteristiche della matrice A del sistema lineare che si intende risolvere. Anzitutto, sia MATLAB che Octave controllano se il formato della matrice è *full* o *sparse*. Nel primo caso le *function* `MAT&OCT` si basano sulla libreria LAPACK (una libreria di algebra lineare largamente usata dalla comunità scientifica [ABB⁹⁹]), mentre nel secondo caso si appoggiano alla libreria UMFPACK e ad altri pacchetti come SuiteSparse [Dav06].

Se la matrice ha formato *full*, `MAT&OCT` segue a grandi linee la seguente procedura:

1. se la matrice è triangolare, superiore o inferiore (anche a meno di permutazioni), il sistema è risolto con il metodo di sostituzione all'indietro (5.10) o in avanti (5.9), rispettivamente;
2. se la matrice è simmetrica con elementi positivi sulla diagonale principale, in prima battuta è richiamata la fattorizzazione di Cholesky (5.17)–(5.18). Qualora questa fallisca MATLAB implementa la variante LDL della fattorizzazione LU (si cercano L triangolare inferiore e D diagonale tali che $A = LDL^T$), mentre Octave richiama la fattorizzazione LU;
3. se la matrice non è simmetrica, oppure è simmetrica ma non soddisfa le ipotesi del punto precedente, il sistema è risolto con la fattorizzazione LU con *pivoting* per righe (5.23);
4. se la matrice non è quadrata, viene calcolata una soluzione nel senso dei minimi quadrati mediante la fattorizzazione QR (5.43)–(5.44).

Se la matrice ha formato *sparse*, oltre ai controlli effettuati per le matrici di tipo *full*, `MAT&OCT` controlla se la matrice è a banda; in caso positivo vengono richiamate delle varianti della fattorizzazione LU ottimizzate per questo tipo di matrici (come l'algoritmo di Thomas introdotto in Sezione 5.6). Infine, nel caso in cui la matrice sia generale (non simmetrica e non a banda), viene richiamato il metodo *multifrontale* implementato in UMFPACK.

Riassumendo

1. La fattorizzazione LU di $A \in \mathbb{R}^{n \times n}$ consiste nel calcolare una matrice triangolare inferiore L ed una matrice triangolare superiore U , tali che $A = LU$;
2. se esiste, la fattorizzazione LU non è unica. Tuttavia può essere univocamente determinata fissando n condizioni addizionali, come, ad esempio, ponendo i coefficienti diagonali di L pari a uno; in tal caso si trova la cosiddetta fattorizzazione LU;
3. la fattorizzazione LU esiste unica se e solo se le sottomatrici principali di A di ordine da 1 fino a $n - 1$ sono tutte non singolari;

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 reali simmetriche e definite positive esiste ed è unica la cosiddetta fattorizzazione di Cholesky, $A = R^T R$, con R triangolare superiore. 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 sovrardeterminato può essere intesa nel senso dei minimi quadrati. Essa può essere calcolata attraverso la fattorizzazione QR oppure usando la decomposizione in valori singolari di A.

5.9 Metodi iterativi

Un metodo iterativo per la risoluzione del sistema lineare (5.1) con $A \in \mathbb{R}^{n \times n}$ e $\mathbf{b} \in \mathbb{R}^n$ consiste nel costruire una successione di vettori $\{\mathbf{x}^{(k)}, k \geq 0\}$ di \mathbb{R}^n che *converge* alla soluzione esatta \mathbf{x} , ossia tale che

$$\lim_{k \rightarrow \infty} \mathbf{x}^{(k)} = \mathbf{x} \quad (5.49)$$

per un qualunque vettore iniziale $\mathbf{x}^{(0)} \in \mathbb{R}^n$.

Per realizzare questo processo una possibile strategia è quella di definire ricorsivamente

$$\mathbf{x}^{(k+1)} = B\mathbf{x}^{(k)} + \mathbf{g}, \quad k \geq 0, \quad (5.50)$$

essendo B una matrice opportuna (dipendente da A) e g un vettore opportuno (dipendente da A e da b), scelti in modo tale da garantire la *condizione di consistenza*

$$\mathbf{x} = B\mathbf{x} + \mathbf{g}. \quad (5.51)$$

Essendo $\mathbf{x} = A^{-1}\mathbf{b}$, necessariamente dovrà avversi $\mathbf{g} = (I - B)A^{-1}\mathbf{b}$.

Detto $\mathbf{e}^{(k)} = \mathbf{x} - \mathbf{x}^{(k)}$ l'errore al passo k , sottraendo la (5.50) dalla (5.51), si ottiene

$$\mathbf{e}^{(k+1)} = B\mathbf{e}^{(k)}.$$

Per tale ragione B è detta *matrice di iterazione* del metodo (5.50). Se B è simmetrica e definita positiva, grazie alla (5.29) otteniamo

$$\|\mathbf{e}^{(k+1)}\| = \|B\mathbf{e}^{(k)}\| \leq \rho(B)\|\mathbf{e}^{(k)}\|, \quad \forall k \geq 0,$$

dove $\rho(B)$ è il *raggio spettrale* di B ed è il massimo modulo degli autovalori di B . Per matrici simmetriche e definite positive esso coincide con il massimo autovalore. Iterando a ritroso la stessa disegualanza, si trova

$$\|\mathbf{e}^{(k)}\| \leq [\rho(B)]^k \|\mathbf{e}^{(0)}\|, \quad k \geq 0. \quad (5.52)$$

Se $\rho(B) < 1$, allora $\mathbf{e}^{(k)} \rightarrow \mathbf{0}$ per $k \rightarrow \infty$ per ogni possibile $\mathbf{e}^{(0)}$ (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 $\rho(B)$, dalla (5.52) 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 $[\rho(B)]^{k_{\min}} \leq \varepsilon$.

In generale, per una generica matrice vale la seguente proprietà:

Proposizione 5.2 *Un metodo iterativo della forma (5.50) la cui matrice di iterazione B soddisfi la (5.51), è convergente per ogni $\mathbf{x}^{(0)}$ se e soltanto se $\rho(B) < 1$. Inoltre, minore è $\rho(B)$, minore è il numero di iterazioni necessario per ridurre l'errore iniziale di un dato fattore.*

5.9.1 Come costruire un metodo iterativo

Una tecnica generale per costruire un metodo iterativo è basata sulla seguente *decomposizione additiva* (o *splitting*) della matrice A :

$$A = P - (P - A),$$

dove P è una opportuna matrice non singolare che chiameremo *precondizionatore* di A . La ragione di tale nome sarà chiarita nella Sezione 5.10.1. Di conseguenza,

$$Px = (P - A)x + b$$

è un sistema della forma (5.51) purché si ponga $B = P^{-1}(P - A) = I - P^{-1}A$ e $\mathbf{g} = P^{-1}\mathbf{b}$. Questa identità suggerisce la definizione del seguente metodo iterativo

$$P(\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}) = \mathbf{r}^{(k)}, \quad k \geq 0,$$

dove

$$\mathbf{r}^{(k)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(k)} \quad (5.53)$$

denota il vettore residuo alla k -esima iterazione. Una generalizzazione di questo metodo iterativo è

$$P(\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}) = \alpha_k \mathbf{r}^{(k)}, \quad k \geq 0 \quad (5.54)$$

dove $\alpha_k \neq 0$ è un parametro che può cambiare ad ogni iterazione e che, a priori, servirà a migliorare le proprietà di convergenza della successione $\{\mathbf{x}^{(k)}\}$.

Il metodo (5.54) è noto anche come *metodo di Richardson*; se $\alpha_k = \alpha$ per ogni $k \geq 0$ esso si dice *stazionario*, altrimenti, si dirà *dinamico*. Esso richiede ad ogni passo di trovare il cosiddetto *residuo precondizionato* $\mathbf{z}^{(k)}$ dato dalla soluzione del sistema lineare

$$P\mathbf{z}^{(k)} = \mathbf{r}^{(k)}, \quad (5.55)$$

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 (5.55) sia modesto (ogni matrice P diagonale, tridiagonale o triangolare andrebbe bene a questo scopo). Introduciamo ora alcuni esempi particolari di metodi iterativi della forma (5.54).

Il metodo di Jacobi

Se i coefficienti diagonali di A sono non nulli, possiamo scegliere $P = D = \text{diag}(a_{11}, a_{22}, \dots, a_{nn})$, ovvero D è la matrice diagonale costruita a partire dagli elementi diagonali di A . Il metodo di Jacobi corrisponde a questa scelta supponendo $\alpha_k = 1$ per ogni k . Di conseguenza, dalla (5.54), otteniamo

$$D\mathbf{x}^{(k+1)} = \mathbf{b} - (A - D)\mathbf{x}^{(k)}, \quad k \geq 0,$$

che, per componenti, assume la forma

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1, j \neq i}^n a_{ij} x_j^{(k)} \right), \quad i = 1, \dots, n \quad (5.56)$$

dove $k \geq 0$ e $\mathbf{x}^{(0)} = (x_1^{(0)}, x_2^{(0)}, \dots, x_n^{(0)})^T$ è il vettore iniziale.

La matrice di iterazione è allora

$$B = D^{-1}(D - A) = \begin{bmatrix} 0 & -a_{12}/a_{11} & \dots & -a_{1n}/a_{11} \\ -a_{21}/a_{22} & 0 & & -a_{2n}/a_{22} \\ \vdots & & \ddots & \vdots \\ -a_{n1}/a_{nn} & -a_{n2}/a_{nn} & \dots & 0 \end{bmatrix}. \quad (5.57)$$

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 $A \in \mathbb{R}^{n \times n}$ 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 (5.57), 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 (si veda la Sezione 6.4). Siano λ un autovalore di B e \mathbf{x} un autovettore associato a λ . 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 5.2 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 , il massimo numero `kmax` di iterazioni consentite ed una tolleranza fissata `tol` per il test d'arresto. La procedura iterativa si arresta non appena il rapporto fra la norma del residuo corrente e quella del residuo iniziale sia inferiore od uguale a `tol` (si veda a questo riguardo la Sezione 5.11).

Programma 5.2. itermeth: metodo iterativo generico

```

function [x, iter, resv]= itermeth(A,b,x0,kmax,tol,P)
%ITERMETH Un metodo iterativo generale
% [X, ITER, RESV] = ITERMETH(A,B,X0,KMAX,TOL,P)
% risolve iterativamente il sistema di equazioni
% lineari A*X=B su X. La matrice A di N-per-N coef-
% ficienti 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 selezio-
% nato il metodo di Gauss-Seidel. Altrimenti, P e'
% una matrice N-per-N non singolare che gioca il ruo-
% lo di precondizionatore nel metodo del gradiente,
% che e' un metodo di Richardson a parametro
% dinamico. Il metodo si arresta quando il rapporto
% fra la norma del residuo corrente e quella del
% residuo iniziale e' minore di TOL e ITER e' il
% numero di iterazioni effettuate. KMAX prescrive
% il numero massimo di iterazioni consentite. Se P
% non viene precisata, viene usato il metodo del
% gradiente non precondizionato. In output, RESV
% e' il vettore contenente i residui relativi
[n,n]=size(A); resv=[ ];
if nargin == 6
    if ischar(P)==1
        if P=='J'
            L=diag(diag(A)); U=eye(n); beta=1; alpha=1;
        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; x = x0;
r = b - A * x0; r0 = norm(r); res = tol+1;
while res > tol & iter < kmax
    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;
    res = norm (r) / r0; resv = [resv; res];
end

```

Il metodo di Gauss-Seidel

Quando si applica il metodo di Jacobi ogni componente $x_i^{(k+1)}$ del nuovo vettore $\mathbf{x}^{(k+1)}$ viene calcolata indipendentemente 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 (5.56) come segue: per $k \geq 0$ (supponendo ancora $a_{ii} \neq 0$ per $i = 1, \dots, n$)

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^n a_{ij} x_j^{(k)} \right), \quad i = 1, \dots, n \quad (5.58)$$

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 (5.54), 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

$$B = (D - E)^{-1}(D - E - A).$$

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 reali simmetriche e definite positive.

Il metodo di Gauss-Seidel è richiamabile nel Programma 5.2 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 (si veda ad esempio [Saa03, Thm. 4.7]):

Proposizione 5.4 *Se $A \in \mathbb{R}^{n \times n}$ è una matrice tridiagonale 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 quella del metodo di Jacobi.*

Esempio 5.13 Consideriamo un sistema lineare $\mathbf{Ax} = \mathbf{b}$ 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 \mathbf{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 $\text{tol} = 10^{-12}$, $\text{kmax}=400$ e si è partiti da un dato iniziale nullo). Le istruzioni necessarie per ottenere questo risultato sono

```
n=10; e=ones(n,1); x0=zeros(n,1);
A=spdiags([-e,3*e,-2*e],[-1,0,1],n,n); b=A*e;
[x,iterJ]=itermeth(A,b,x0,400,1.e-12,'J');
[x,iterG]=itermeth(A,b,x0,400,1.e-12,'G');
```



Si vedano gli Esercizi 5.11–5.14.

Il metodo di Richardson

Ritorniamo ora a considerare metodi che si scrivono nella forma generale (5.54).

Per studiare la convergenza di tali metodi definiamo la cosiddetta norma dell'energia associata alla matrice A

$$\|\mathbf{v}\|_A = \sqrt{\mathbf{v}^T A \mathbf{v}} \quad \forall \mathbf{v} \in \mathbb{R}^n. \quad (5.59)$$

Valgono i seguenti risultati (si veda, ad esempio, [QV94, Cap. 2], [Axe94]).

Proposizione 5.5 *Sia $A \in \mathbb{R}^{n \times n}$. Per ogni matrice non singolare $P \in \mathbb{R}^{n \times n}$ il metodo di Richardson stazionario (5.54) converge se e solo se*

$$|\lambda_i|^2 < \frac{2}{\alpha} \operatorname{Re} \lambda_i \quad \forall i = 1, \dots, n,$$

dove λ_i sono gli autovalori di $P^{-1}A$. In particolare, se gli autovalori di $P^{-1}A$ sono reali, allora esso converge se e solo se

$$0 < \alpha \lambda_i < 2 \quad \forall i = 1, \dots, n.$$

Se A e P sono entrambe simmetriche e definite positive il metodo di Richardson stazionario converge per ogni possibile scelta di $\mathbf{x}^{(0)}$ se e solo se $0 < \alpha < 2/\lambda_{\max}(> 0)$ 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

$$\alpha_{opt} = \frac{2}{\lambda_{\min} + \lambda_{\max}} \quad (5.60)$$

essendo λ_{\min} l'autovalore minimo di $P^{-1}A$. Infine, se $\alpha = \alpha_{opt}$, vale la seguente stima di convergenza

$$\|\mathbf{e}^{(k)}\|_A \leq \left(\frac{K(P^{-1}A) - 1}{K(P^{-1}A) + 1} \right)^k \|\mathbf{e}^{(0)}\|_A, \quad k \geq 0 \quad (5.61)$$

Osservazione 5.5 Osserviamo che se A e P sono simmetriche e definite positive, non è detto che $P^{-1}A$ sia anch'essa simmetrica e definita positiva. Tuttavia essa risulta simile ad una matrice simmetrica e definita positiva, quindi i suoi autovalori sono tutti reali e strettamente positivi e possiamo valutarne il condizionamento spettrale (5.32) (si veda l'Esercizio 5.17). ■

Poiché la valutazione del parametro ottimale α_{opt} nel metodo di Richardson stazionario richiede la conoscenza degli autovalori estremi di $P^{-1}A$, quasi sempre si preferisce utilizzare un metodo dinamico in cui, per ogni $k \geq 0$, il parametro α_k venga calcolato a partire da quantità già note. La strategia più comune è quella fornita dal metodo del gradiente che illustreremo nella prossima sezione.

5.10 I metodi del Gradiente e del Gradiente Coniugato

Dato il sistema lineare (5.1) quadrato di dimensione n , possiamo definire la funzione $\Phi : \mathbb{R}^n \rightarrow \mathbb{R}$

$$\Phi(\mathbf{x}) = \frac{1}{2} \mathbf{x}^T A \mathbf{x} - \mathbf{x}^T \mathbf{b} \quad (5.62)$$

(in Figura 5.16 è riportato un esempio di funzione Φ quando $n = 2$).

Qualora A sia simmetrica e definita positiva, Φ è una funzione convessa, cioè tale che per ogni $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ e per ogni $\alpha \in [0, 1]$ vale

$$\Phi(\alpha \mathbf{x} + (1 - \alpha) \mathbf{y}) \leq \alpha \Phi(\mathbf{x}) + (1 - \alpha) \Phi(\mathbf{y}), \quad (5.63)$$

ed ammette un unico punto stazionario \mathbf{x}^* che è anche punto di minimo locale ed assoluto. Quindi

$$\mathbf{x}^* = \underset{\mathbf{x} \in \mathbb{R}^n}{\operatorname{argmin}} \Phi(\mathbf{x}) \quad (5.64)$$

è l'unica soluzione dell'equazione

$$\nabla \Phi(\mathbf{x}) = A \mathbf{x} - \mathbf{b} = \mathbf{0}. \quad (5.65)$$

Risolvere il problema di minimo (5.64) equivale pertanto a risolvere il sistema (5.1).

Partendo da un generico punto $\mathbf{x}^{(0)} \in \mathbb{R}^n$, i metodi del gradiente e del gradiente coniugato costruiscono una successione di vettori $\mathbf{x}^{(k)}$ convergenti a \mathbf{x}^* , sfruttando le informazioni fornite dal vettore gradiente di Φ . Infatti, per un generico $\bar{\mathbf{x}} \in \mathbb{R}^n$ diverso da \mathbf{x}^* , $\nabla \Phi(\bar{\mathbf{x}})$ è un vettore non nullo di \mathbb{R}^n che individua la direzione lungo cui avviene la massima crescita di Φ , di conseguenza $-\nabla \Phi(\bar{\mathbf{x}})$ individua la direzione di massima decrescita di Φ a partire da $\bar{\mathbf{x}}$ (si veda Figura 5.17).

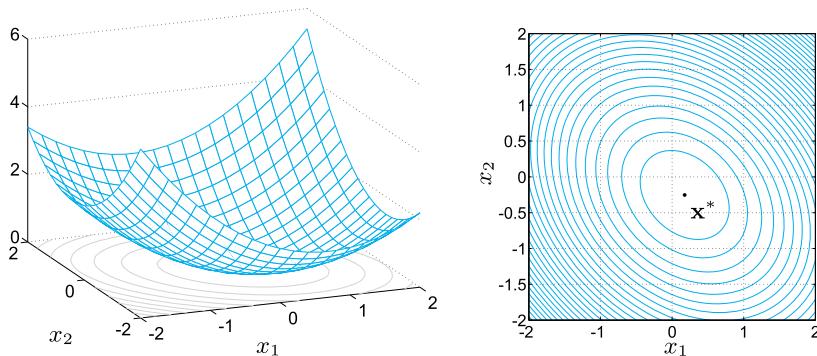


Figura 5.16. La funzione $\Phi(\mathbf{x})$ associata ad $A = [1, 0.3; 0.3, 1]$ e $\mathbf{b} = [0.1, -0.2]^T$. A sinistra il plot di $\Phi(\mathbf{x})$ in 3 dimensioni, a destra le sue curve di livello ed il punto di minimo assoluto \mathbf{x}^*

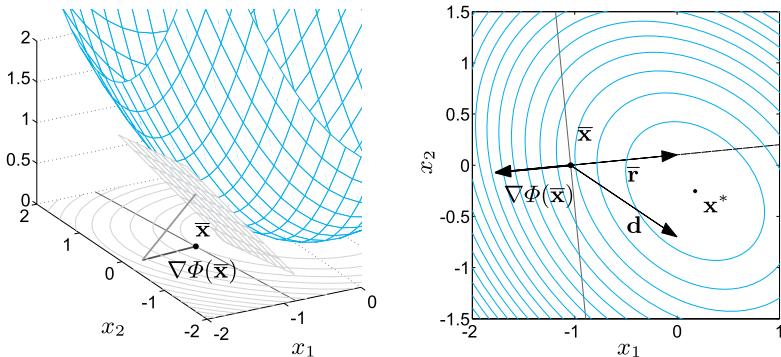


Figura 5.17. A sinistra sono mostrati il piano tangente a Φ nel punto $\bar{\mathbf{x}}$, il vettore normale al piano tangente e infine il vettore $\nabla\Phi(\bar{\mathbf{x}})$. A destra sono disegnati il vettore $\nabla\Phi(\bar{\mathbf{x}})$, il residuo $\bar{\mathbf{r}}$ e un'altra direzione di discesa \mathbf{d} in $\bar{\mathbf{x}}$

Ricordando che il vettore residuo nel punto $\bar{\mathbf{x}}$ è $\bar{\mathbf{r}} = \mathbf{b} - A\bar{\mathbf{x}}$, grazie alla (5.65) abbiamo

$$\bar{\mathbf{r}} = -\nabla\Phi(\bar{\mathbf{x}}), \quad (5.66)$$

cioè il residuo individua una possibile direzione lungo cui muoversi per avvicinarsi al punto di minimo \mathbf{x}^* .

Più in generale, diciamo che un vettore \mathbf{d} rappresenta una *direzione di discesa* per Φ nel punto $\bar{\mathbf{x}}$ se si verificano le seguenti condizioni (si veda la Figura 5.17):

$$\begin{aligned} \mathbf{d}^T \nabla\Phi(\bar{\mathbf{x}}) &< 0 & \text{se } \nabla\Phi(\bar{\mathbf{x}}) \neq \mathbf{0} \\ \mathbf{d} = \mathbf{0} & & \text{se } \nabla\Phi(\bar{\mathbf{x}}) = \mathbf{0}. \end{aligned} \quad (5.67)$$

Il residuo è pertanto una direzione di discesa.

Per avvicinarci al punto di minimo ci muoveremo lungo opportune direzioni di discesa.

A tale scopo, i *metodi di discesa* sono così definiti: assegnato un vettore $\mathbf{x}^{(0)} \in \mathbb{R}^n$, per $k = 0, 1, \dots$ fino a convergenza

$$\boxed{\begin{aligned} &\text{si determina una direzione di discesa } \mathbf{d}^{(k)} \in \mathbb{R}^n \\ &\text{si determina un passo } \alpha_k \in \mathbb{R} \\ &\text{si pone } \mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k \mathbf{d}^{(k)} \end{aligned}} \quad (5.68)$$

I metodi del gradiente e del gradiente coniugato sono metodi di discesa che si differenziano nella scelta delle direzioni di discesa, mentre la scelta dei passi α_k è comune ad entrambi. Cominciamo allora ad esaminare questo secondo punto, supponendo di aver già calcolato la nuova direzione $\mathbf{d}^{(k)}$.

Cerchiamo

$$\alpha_k = \underset{\alpha \in \mathbb{R}}{\operatorname{argmin}} \Phi(\mathbf{x}^{(k)} + \alpha \mathbf{d}^{(k)}), \quad (5.69)$$

ovvero un passo α_k tale che $\Phi(\mathbf{x}^{(k+1)})$ sia il minimo di Φ lungo la direzione $\mathbf{d}^{(k)}$ passante per $\mathbf{x}^{(k)}$. Svolgendo i conti (si veda l'Esercizio 5.18), si ottiene

$$\boxed{\alpha_k = \frac{(\mathbf{d}^{(k)})^T \mathbf{r}^{(k)}}{(\mathbf{d}^{(k)})^T A \mathbf{d}^{(k)}}} \quad (5.70)$$

Denotando con $\mathbf{e}^{(k+1)} = \mathbf{x} - \mathbf{x}^{(k+1)}$ il vettore errore al passo $(k+1)$, con $\|\cdot\|_A$ la norma dell'energia definita in (5.59) e con $\varphi_k(\alpha) = \|\mathbf{e}^{(k+1)}\|_A^2$, si dimostra che α_k definito in (5.70) è anche l'unica soluzione del problema

$$\alpha_k = \underset{\alpha \in \mathbb{R}}{\operatorname{argmin}} \varphi_k(\alpha), \quad (5.71)$$

ovvero garantisce che la norma (dell'energia) dell'errore al passo successivo sia minima. Con tale scelta il residuo al passo successivo è ortogonale all'ultima direzione di discesa utilizzata, ovvero

$$(\mathbf{d}^{(k)})^T \mathbf{r}^{(k+1)} = 0, \quad k = 0, 1, \dots \quad (5.72)$$

(si veda l'Esercizio 5.19). La relazione (5.72) equivale a dire che la proiezione del residuo al passo $(k+1)$ lungo la direzione $\mathbf{d}^{(k)}$ è nulla. Quando una tale condizione è verificata, diciamo che la nuova soluzione $\mathbf{x}^{(k+1)}$ è *ottimale* rispetto alla direzione $\mathbf{d}^{(k)}$. (Ricordiamo che, per ogni $\mathbf{v}, \mathbf{w} \in \mathbb{R}^n$, il valore reale $\mathbf{v}^T \mathbf{w}$ rappresenta la proiezione di \mathbf{v} lungo la direzione \mathbf{w} .)

Il *metodo del gradiente* è caratterizzato dalla scelta

$$\mathbf{d}^{(k)} = \mathbf{r}^{(k)} = -\nabla \Phi(\mathbf{x}^{(k)}), \quad k = 0, 1, \dots, \quad (5.73)$$

ovvero la direzione di discesa ad ogni passo è la direzione opposta a quella del gradiente della funzione Φ (da cui il nome del metodo).

L'algoritmo del metodo del gradiente è: dato $\mathbf{x}^{(0)} \in \mathbb{R}^n$, definiamo $\mathbf{r}^{(0)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(0)}$ e calcoliamo:

$$\begin{aligned} & \text{per } k = 0, 1, \dots \\ & \alpha_k = \frac{(\mathbf{r}^{(k)})^T \mathbf{r}^{(k)}}{(\mathbf{r}^{(k)})^T \mathbf{A} \mathbf{r}^{(k)}}, \\ & \mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k \mathbf{r}^{(k)}, \\ & \mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - \alpha_k \mathbf{A} \mathbf{r}^{(k)} \end{aligned} \quad (5.74)$$

Osserviamo che le direzioni di discesa sono calcolate alla fine del ciclo in k , e sono quindi pronte per essere utilizzate al passo successivo.

Vale il seguente risultato:

Proposizione 5.6 Se $\mathbf{A} \in \mathbb{R}^{n \times n}$ è simmetrica e definita positiva il metodo del gradiente converge alla soluzione del sistema lineare $\mathbf{A}\mathbf{x} = \mathbf{b}$ per ogni dato iniziale $\mathbf{x}^{(0)} \in \mathbb{R}^n$ e inoltre vale la seguente stima dell'errore (si veda [QSSG14] per una possibile dimostrazione)

$$\|\mathbf{e}^{(k)}\|_{\mathbf{A}} \leq \left(\frac{K(\mathbf{A}) - 1}{K(\mathbf{A}) + 1} \right)^k \|\mathbf{e}^{(0)}\|_{\mathbf{A}}, \quad k \geq 0 \quad (5.75)$$

dove $\|\cdot\|_{\mathbf{A}}$ è la norma dell'energia definita in (5.59) e $K(\mathbf{A})$ è il numero di condizionamento (spettrale) della matrice \mathbf{A} definito in (5.32).

Il metodo del gradiente è implementato nel Programma 5.3 che verrà presentato nella Sezione 5.10.1.

Esempio 5.14 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

$$2x_1 + x_2 = 1, \quad x_1 + 3x_2 = 0 \quad (5.76)$$

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 5.18 l'andamento del residuo relativo

$$E^{(k)} = \|\mathbf{r}^{(k)}\| / \|\mathbf{r}^{(0)}\|, \quad (5.77)$$

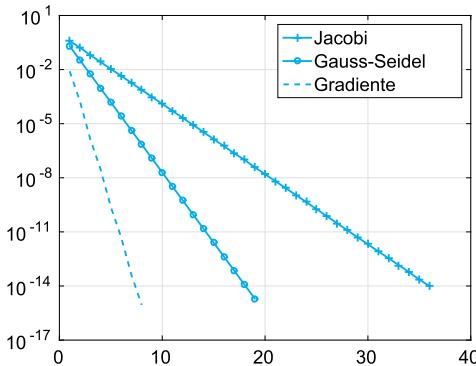


Figura 5.18. Storia di convergenza dei metodi di Jacobi, Gauss-Seidel e del gradiente applicati al sistema (5.76)

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 in 8 iterazioni contro le 19 iterazioni del metodo di Gauss-Seidel e le 37 del metodo di Jacobi. ■

Vogliamo ora capire come costruire delle direzioni di discesa “migliori” al fine di giungere a convergenza in un numero di iterazioni inferiore rispetto a quelle del metodo del gradiente.

Partendo dall’ultima riga dell’algoritmo (5.68) e sfruttando la definizione ricorsiva dei vettori $\mathbf{x}^{(k)}$, possiamo scrivere

$$\begin{aligned}
 \mathbf{x}^{(k+1)} &= \mathbf{x}^{(k)} + \alpha_k \mathbf{d}^{(k)} \\
 &= \mathbf{x}^{(k-1)} + \alpha_{k-1} \mathbf{d}^{(k-1)} + \alpha_k \mathbf{d}^{(k)} \\
 &= \dots \\
 &= \mathbf{x}^{(0)} + \sum_{j=0}^k \alpha_j \mathbf{d}^{(j)}. \tag{5.78}
 \end{aligned}$$

Deduciamo che il vettore $(\mathbf{x}^{(k+1)} - \mathbf{x}^{(0)}) \in \mathbb{R}^n$ è una combinazione lineare delle direzioni di discesa $\{\mathbf{d}^{(j)}\}_{j=0}^k$.

È importante osservare che le direzioni di discesa costruite dal metodo del gradiente ($\mathbf{d}^{(j)} = \mathbf{r}^{(j)}$ per $j = 0, 1, \dots$) non sono necessariamente tutte linearmente indipendenti fra di loro, ma grazie alla (5.72) è garantita l’ortogonalità solo fra due direzioni successive, ovvero di indice j e $(j+1)$.

Anche nel caso ottimistico in cui le n direzioni $\{\mathbf{r}^{(0)}, \dots, \mathbf{r}^{(n-1)}\}$ risultassero tutte linearmente indipendenti (e costituissero quindi una base di \mathbb{R}^n rispetto alla quale si potrebbe sviluppare anche il vettore soluzione \mathbf{x}^*), i passi α_k costruiti in (5.70) non coinciderebbero necessaria-

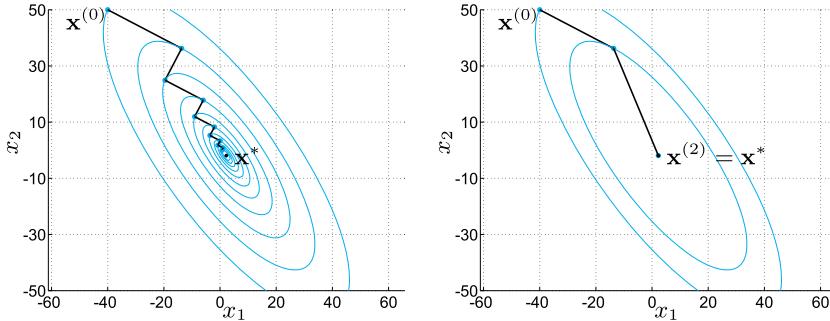


Figura 5.19. A sinistra è mostrata la convergenza del metodo del gradiente per il sistema $Ax = b$ con $A = [10, 6; 6, 6]$ e $b = [11, 2]^T$. A destra la convergenza (in soli due passi) del metodo del gradiente coniugato. In nero sono riportate le direzioni di discesa per i due metodi, mentre in azzurro sono disegnate le curve di livello di Φ alle quote $\Phi(\mathbf{x}^{(k)})$

mente con i coefficienti dello sviluppo di $(\mathbf{x}^* - \mathbf{x}^{(0)})$ rispetto al sistema $\{\mathbf{r}^{(0)}, \dots, \mathbf{r}^{(n-1)}\}$. Quindi, potrebbero servire molte (a priori anche infinite) direzioni di discesa per dare una rappresentazione sufficientemente accurata di $(\mathbf{x}^* - \mathbf{x}^{(0)})$. In Figura 5.19 mostriamo i primi termini della successione $\{\mathbf{x}^{(k)}\}$ e le relative direzioni di discesa $\mathbf{d}^{(k)} = \mathbf{r}^{(k)}$ per un sistema di dimensione $n = 2$. Osserviamo che, per ogni k , $\mathbf{r}^{(k+1)}$ e $\mathbf{r}^{(k)}$ individuano direzioni perpendicolari e quindi (5.72) è verificata. In questo esempio servono ben 57 iterazioni per giungere a convergenza, avendo posto la tolleranza del test d'arresto pari a $\varepsilon = 10^{-8}$.

Il *metodo del gradiente coniugato* costruisce un sistema di direzioni di discesa in \mathbb{R}^n che siano tutte linearmente indipendenti fra di loro (quindi $\{\mathbf{d}^{(k)}\}_{k=0}^{n-1}$ sarà una base di \mathbb{R}^n) e tali che i valori $\{\alpha_k\}_{k=0}^{n-1}$ calcolati con (5.70) siano proprio i coefficienti dello sviluppo di $(\mathbf{x}^* - \mathbf{x}^{(0)})$ rispetto alla base $\{\mathbf{d}^{(k)}\}_{k=0}^{n-1}$. Questo comporta che l' n -simo termine $\mathbf{x}^{(n)}$ della successione (5.78) coincide con la soluzione esatta \mathbf{x}^* (una proprietà notevole del metodo del gradiente coniugato). In Figura 5.19, a destra, sono riportate le $n = 2$ iterate del metodo del gradiente coniugato, insieme alle direzioni di discesa (in nero). In questo caso bastano 2 iterazioni per ottenere la soluzione esatta del sistema 2×2 .

Dopo aver anticipato l'idea del metodo, vediamo nel dettaglio come si costruiscono le sue direzioni di discesa e le proprietà che esse soddisfano.

Le direzioni di discesa del metodo del gradiente coniugato sono così definite:

$$\begin{aligned} \mathbf{d}^{(0)} &= \mathbf{r}^{(0)}, \\ \mathbf{d}^{(k+1)} &= \mathbf{r}^{(k+1)} - \beta_k \mathbf{d}^{(k)}, \quad k = 0, 1, \dots, \end{aligned} \tag{5.79}$$

con

$$\beta_k = \frac{(\mathbf{A}\mathbf{d}^{(k)})^T \mathbf{r}^{(k+1)}}{(\mathbf{A}\mathbf{d}^{(k)})^T \mathbf{d}^{(k)}}, \quad k = 0, 1, \dots \quad (5.80)$$

Esse sono A-*ortogonal* (o A-*coniugate*), cioè tali che per ogni $k \geq 0$,

$$(\mathbf{A}\mathbf{d}^{(j)})^T \mathbf{d}^{(k+1)} = 0, \quad j = 0, 1, \dots, k, \quad (5.81)$$

e tali da garantire che

$$(\mathbf{r}^{(k+1)})^T \mathbf{d}^{(j)} = 0, \quad j = 0, 1, \dots, k. \quad (5.82)$$

Le relazioni (5.81) garantiscono che i vettori $\mathbf{d}^{(k)}$ siano anche mutuamente linearmente indipendenti, in quanto la matrice A è non singolare.

Le relazioni (5.82) garantiscono che il residuo al passo $(k+1)$ sia ortogonale a tutte le direzioni di discesa precedentemente generate, cioè che le proiezioni del residuo $\mathbf{r}^{(k+1)}$ lungo le direzioni $\mathbf{d}^{(0)}, \dots, \mathbf{d}^{(k)}$ siano nulle e, quindi, che la soluzione $\mathbf{x}^{(k+1)}$ sia ottimale rispetto a tutte le direzioni di discesa finora costruite. (Per la dimostrazione si veda [QSSG14].)

In virtù della relazione (5.82) per $k = n - 1$, il residuo $\mathbf{r}^{(n)}$ (che è un vettore di \mathbb{R}^n) risulta quindi ortogonale alla base $\{\mathbf{d}^{(k)}\}_{k=0}^{n-1}$ (anch'essa di vettori di \mathbb{R}^n), cioè

$$\mathbf{r}^{(n)} = \mathbf{0} \quad \text{e} \quad \mathbf{x}^{(n)} = \mathbf{x}^*.$$

L'algoritmo del metodo del gradiente coniugato è quindi: dato $\mathbf{x}^{(0)}$, definiamo $\mathbf{d}^{(0)} = \mathbf{r}^{(0)} = \mathbf{b} - \mathbf{Ax}^{(0)}$ e calcoliamo:

per $k = 0, 1, \dots$

$$\begin{aligned} \alpha_k &= \frac{(\mathbf{d}^{(k)})^T \mathbf{r}^{(k)}}{(\mathbf{d}^{(k)})^T \mathbf{Ad}^{(k)}}, \\ \mathbf{x}^{(k+1)} &= \mathbf{x}^{(k)} + \alpha_k \mathbf{d}^{(k)}, \\ \mathbf{r}^{(k+1)} &= \mathbf{r}^{(k)} - \alpha_k \mathbf{Ad}^{(k)}, \end{aligned} \quad (5.83)$$

$$\beta_k = \frac{(\mathbf{Ad}^{(k)})^T \mathbf{r}^{(k+1)}}{(\mathbf{Ad}^{(k)})^T \mathbf{d}^{(k)}}$$

$$\mathbf{d}^{(k+1)} = \mathbf{r}^{(k+1)} - \beta_k \mathbf{d}^{(k)}$$

Vale il seguente risultato:

Proposizione 5.7 *Sia $A \in \mathbb{R}^{n \times n}$ una matrice simmetrica e definita positiva. Il metodo del gradiente coniugato per risolvere (5.1) converge al più in n iterazioni (in aritmetica esatta). Inoltre, il residuo $\mathbf{r}^{(k)}$ alla k -esima iterazione (con $k < n$) è ortogonale a $\mathbf{d}^{(j)}$, per $j = 0, \dots, k-1$ e*

$$\|\mathbf{e}^{(k)}\|_A \leq \frac{2c^k}{1+c^{2k}} \|\mathbf{e}^{(0)}\|_A, \quad (5.84)$$

$$\text{essendo } c = \frac{\sqrt{K(A)} - 1}{\sqrt{K(A)} + 1}.$$

In aritmetica esatta il metodo del gradiente coniugato è pertanto un metodo diretto in quanto termina dopo un numero finito di iterazioni.

D'altra parte, per matrici di grande dimensione esso viene usualmente impiegato come un metodo iterativo ed arrestato quando uno stimatore dell'errore, come ad esempio il residuo relativo (5.77), è minore di una tolleranza assegnata.

Grazie alla (5.84), 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 $K(A)$).

Il metodo del gradiente coniugato è implementato nel Programma 5.4 che verrà presentato nella prossima Sezione.

5.10.1 Precondizionatori e metodi iterativi precondizionati

Dalla stima (5.75) si evince che le iterazioni del metodo del gradiente, quando convergono alla soluzione esatta \mathbf{x}^* , sono caratterizzate da una velocità di convergenza dipendente dal numero di condizionamento della matrice A : tanto più grande è il numero di condizionamento di A e tanto più lenta sarà la convergenza.

Lo stesso avviene per il metodo del gradiente coniugato nel caso (assai frequente) in cui esso venga utilizzato come metodo iterativo.

Per accelerare la convergenza di questi metodi quando il numero di condizionamento (5.32) è grande, invece di risolvere il sistema lineare (5.1) si può risolvere il cosiddetto *sistema precondizionato* (equivalente)

$$\mathbf{P}_L^{-1} \mathbf{A} \mathbf{P}_R^{-1} \hat{\mathbf{x}} = \mathbf{P}_L^{-1} \mathbf{b}, \quad \text{con } \hat{\mathbf{x}} = \mathbf{P}_R \mathbf{x}, \quad (5.85)$$

dove \mathbf{P}_L e \mathbf{P}_R sono opportune matrici non singolari, tali che:

1. la matrice

$$\mathbf{P} = \mathbf{P}_L \mathbf{P}_R, \quad (5.86)$$

detta *precondizionatore*, è simmetrica e definita positiva;

2. $K(P^{-1}A) \ll K(A)$;²
3. la risoluzione di sistemi lineari della forma

$$P_L s = t, \quad P_R v = w, \quad (5.87)$$

è computazionalmente molto meno costosa della risoluzione del sistema originario $Ax = b$.

Se si sceglie P_R uguale alla matrice identità, la matrice $P = P_L$ è detta *precondizionatore sinistro*, ed il sistema precondizionato (5.85) assume la forma

$$P^{-1}Ax = P^{-1}b. \quad (5.88)$$

Analogamente, se P_L è la matrice identità, la matrice $P = P_R$ è detta *precondizionatore destro*, ed il sistema precondizionato (5.85) assume la forma

$$AP^{-1}\hat{x} = b, \quad \text{con } \hat{x} = Px. \quad (5.89)$$

In genere può risultare complesso trovare una matrice di precondizionamento che garantisca un bilanciamento ottimale tra abbattimento del numero di condizionamento e sforzo computazionale richiesto per risolvere ad ogni passo il sistema (5.55). La scelta dovrà essere fatta caso per caso, tenuto conto del tipo di matrice A in esame.

Nel caso in cui A sia anche sparsa, oltre ad essere simmetrica definita positiva, la matrice di precondizionamento P può essere costruita mediante una fattorizzazione incompleta di Cholesky della matrice A . Con questa strategia si costruisce una matrice R_P triangolare inferiore che approssima il fattore R^T della fattorizzazione di Cholesky (5.18) di A , quindi si definisce

$$P = R_P R_P^T. \quad (5.90)$$

In tal caso i precondizionatori sinistro e destro sono dati rispettivamente da $P_L = R_P$ e $P_R = R_P^T$.

Una tecnica assai diffusa per costruire R_P è nota come ICT (*incomplete Cholesky with threshold dropping*); per implementarla:

1. si sceglie una tolleranza $\varepsilon_d > 0$ (nota in inglese come *drop tolerance*);
2. si esegue l'algoritmo (5.18) della fattorizzazione di Cholesky (opportunamente modificato per generare una matrice triangolare inferiore) per costruire gli elementi r_{ij} di R_P , ignorando quelli extradiagonali che risultano minori di $c_j \varepsilon_d$ (dove c_j è la norma del vettore colonna j -simo del triangolo inferiore di A).

La tecnica *nofill* invece ignora tutti gli elementi r_{ij} di R_P per cui $a_{ij} = 0$, di fatto è una fattorizzazione senza *fill-in* (si veda la Sezione 5.4.1).

² La matrice $P^{-1}A$ non è necessariamente simmetrica, ma i suoi autovalori sono reali e positivi, si veda l'Osservazione 5.5.

La fattorizzazione incompleta di Cholesky è implementata nella *function ichol* di **MATLAB**. Ad esempio con l'istruzione

```
RP=ichol(A,struct('type','ict','droptol',1e-03));
```

costruiamo la matrice R_P con la tecnica ICT e *drop tolerance* $\varepsilon_d = 10^{-3}$.

Osserviamo che minore è ε_d , meglio R_P approssima R (e quindi minore è $K(P^{-1}A)$), ma allo stesso tempo maggiore è il *fill-in* della matrice R_P e quindi maggiore il costo computazionale per risolvere (5.88).

Per la descrizione di altri precondizionatori rimandiamo a [QSSG14, Axe94, Saa03, QV99, TW05, BGL05, Che04].

Il *metodo del gradiente precondizionato* si ottiene applicando il metodo del gradiente al sistema precondizionato (5.85) in cui si prenda $P_R = P_L^T$ e quindi $P_R^{-1} = P_L^{-T} \equiv (P_L^{-1})^T$. Più precisamente, riscriviamo

$$\underbrace{P_L^{-1}AP_L^{-T}}_{\hat{A}} \underbrace{P_L^T x}_{\hat{x}} = \underbrace{P_L^{-1}b}_{\hat{b}} \quad (5.91)$$

ed applichiamo l'algoritmo (5.74) al sistema $\hat{A}\hat{x} = \hat{b}$. In tal caso abbiamo

$$\hat{r}^{(k)} = \hat{b} - \hat{A}\hat{x}^{(k)} = P_L^{-1}(b - Ax^{(k)}) = P_L^{-1}r^{(k)}$$

e

$$\begin{aligned} \alpha_k &= \frac{(\hat{r}^{(k)})^T \hat{r}^{(k)}}{(\hat{r}^{(k)})^T \hat{A} \hat{d}^{(k)}} = \frac{(P_L^{-1}r^{(k)})^T P_L^{-1}r^{(k)}}{(P_L^{-1}r^{(k)})^T P_L^{-1} A P_L^{-T} P_L^{-1}r^{(k)}} \\ &= \frac{(P_L^{-1}r^{(k)})^T r^{(k)}}{(P_L^{-1}r^{(k)})^T A P_L^{-1}r^{(k)}} = \frac{(z^{(k)})^T r^{(k)}}{(z^{(k)})^T A z^{(k)}}. \end{aligned}$$

Abbiamo utilizzato l'identità $P_L^{-T}P_L^{-1} = P^{-1}$ e definito il *residuo precondizionato*

$$z^{(k)} = P^{-1}r^{(k)}. \quad (5.92)$$

(Naturalmente non è richiesta la costruzione di P^{-1} .) Infine osserviamo che la relazione $\hat{x}^{(k+1)} = \hat{x}^{(k)} + \alpha_k \hat{r}^{(k)}$ può essere riscritta come

$$P_L^T x^{(k+1)} = P_L^T x^{(k)} + \alpha_k P_L^{-1} r^{(k)}$$

e quindi, poiché P_L è non singolare,

$$x^{(k+1)} = x^{(k)} + \alpha_k P^{-1} r^{(k)} = x^{(k)} + \alpha_k z^{(k)}.$$

Analogamente, dalla relazione $\hat{r}^{(k+1)} = \hat{r}^{(k)} - \alpha_k \hat{A} \hat{r}^{(k)}$ otteniamo

$$r^{(k+1)} = r^{(k)} + \alpha_k A P^{-1} r^{(k)} = r^{(k)} - \alpha_k A z^{(k)}.$$

L'algoritmo del gradiente coniugato precondizionato si può pertanto formulare come segue: dato $x^{(0)}$, poniamo $r^{(0)} = b - Ax^{(0)}$ e calcoliamo

per $k = 0, 1, \dots$

$$P\mathbf{z}^{(k)} = \mathbf{r}^{(k)},$$

$$\alpha_k = \frac{(\mathbf{z}^{(k)})^T \mathbf{r}^{(k)}}{(\mathbf{z}^{(k)})^T A \mathbf{z}^{(k)}}, \quad (5.93)$$

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k \mathbf{z}^{(k)},$$

$$\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - \alpha_k A \mathbf{z}^{(k)}$$

Osserviamo che (5.93) differisce da (5.74) solo per la scelta della direzione di discesa: quest'ultima coincide con il residuo in (5.74) e con il residuo precondizionato (5.92) in (5.93).

Ogni iterazione del metodo del gradiente richiede il calcolo di un prodotto matrice-vettore (e la soluzione di (5.87) qualora si consideri la versione precondizionata), più altre operazioni di complessità inferiore, come la somma o il prodotto scalare tra vettori.

Nella consueta ipotesi che A e P siano simmetriche e definite positive, vale un risultato di convergenza analogo a quello di Proposizione 5.6, a patto che in (5.75) $K(A)$ sia sostituito da $K(P^{-1}A)$.

Il metodo del gradiente precondizionato è implementato nel programma 5.3, dove A e \mathbf{b} sono rispettivamente la matrice ed il termine noto del sistema da risolvere, \mathbf{x}_0 è il vettore iniziale, \mathbf{tol} e \mathbf{kmax} contengono la tolleranza ed il numero massimo di iterazioni per il test d'arresto; le iterazioni vengono arrestate al primo k per cui $\sqrt{((\mathbf{r}^{(k)})^T \mathbf{z}^{(k)}) / ((\mathbf{r}^{(0)})^T \mathbf{z}^{(0)})} \leq \mathbf{tol}$.

Assumeremo che il precondizionatore P abbia la forma (5.90) avendo scelto $P_L = R_P$, costruito a partire dalla fattorizzazione incompleta di Cholesky, e $P_R = R_P^T$. Qualora si voglia risolvere il sistema non precondizionato (ovvero con $P = I$) basta inizializzare al vuoto (con $RP=[]$) la variabile RP .

Programma 5.3. gradient: il metodo del gradiente (precondizionato)

```

function [x,relres,k,res]=gradient(A,b,tol,kmax,RP,x0)
% GRADIENT metodo del gradiente (precondizionato).
% [X,RELRES,K,RES]=GRADIENT(A,B,TOL,KMAX,RP,X0)
% risolve il sistema A*X=B con il metodo del gradiente
% (precondizionato). X0 e' il vettore iniziale,
% TOL specifica la tolleranza per il test d'arresto.
% KMAX indica il numero massimo di iterazioni ammesse.
% RP contiene una matrice triangolare superiore t.c.
% il precondizionatore e' definito come P = RP'*RP.
% [X,RELRES,K,RES]=GRADIENT(A,B,TOL,KMAX,[],X0) ri-
% chiama il metodo del gradiente non precondizionato.
% RELRES e' la norma del residuo corrispondente alla
% soluzione calcolata diviso la norma del vettore B.
% K e' il numero di iterazioni effettuate dal metodo,
% RES e' un vettore contenente la norma dei residui
% ad ogni passo del metodo

```

```

k = 0;
bnorm = norm(b);
res=[ ];
if bnorm==0
x=zeros(n,1); relres=0;
else
r=b-A*x0; x=x0;
% sistema sul precondizionatore
if ~isempty(RP); z=RP' \ (RP\r); else z=r; end
zr=z'*r; res0=sqrt(zr); relres=zr;
while relres > tol && k< kmax
    v=A*z;
    alpha=zr/(z'*v);
    x=x+alpha*z;
    r=r-alpha*v;
    if ~isempty(RP); z=RP' \ (RP\r); else z=r; end
    zr=z'*r;
    relres=sqrt(zr)/res0;
    res=[res; relres];
    k=k+1;
end
end

```

Nelle stesse ipotesi sulla matrice di precondizionamento e procedendo in maniera analoga a quanto fatto per il metodo del gradiente, *il metodo del gradiente coniugato precondizionato* è: dato $\mathbf{x}^{(0)}$, definiamo $\mathbf{r}^{(0)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(0)}$, $\mathbf{z}^{(0)} = \mathbf{P}^{-1}\mathbf{r}^{(0)}$ e $\mathbf{d}^{(0)} = \mathbf{z}^{(0)}$, e calcoliamo

$$\begin{aligned}
&\text{per } k = 0, 1, \dots \\
\alpha_k &= \frac{\mathbf{z}^{(k)T} \mathbf{r}^{(k)}}{(\mathbf{A}\mathbf{d}^{(k)})^T \mathbf{d}^{(k)}}, \\
\mathbf{x}^{(k+1)} &= \mathbf{x}^{(k)} + \alpha_k \mathbf{d}^{(k)}, \\
\mathbf{r}^{(k+1)} &= \mathbf{r}^{(k)} - \alpha_k \mathbf{A}\mathbf{d}^{(k)}, \\
\mathbf{P}\mathbf{z}^{(k+1)} &= \mathbf{r}^{(k+1)}, \\
\beta_k &= \frac{(\mathbf{A}\mathbf{d}^{(k)})^T \mathbf{z}^{(k+1)}}{(\mathbf{A}\mathbf{d}^{(k)})^T \mathbf{d}^{(k)}}, \\
\mathbf{d}^{(k+1)} &= \mathbf{z}^{(k+1)} - \beta_k \mathbf{d}^{(k)}
\end{aligned} \tag{5.94}$$

In questo caso la stima dell'errore (5.84) è ancora vera pur di sostituire $K(\mathbf{A})$ con il più favorevole $K(\mathbf{P}^{-1}\mathbf{A})$.

Il metodo del gradiente coniugato è implementato nel Programma 5.4 ed i parametri in input ed output sono definiti come per il Programma 5.3. In **MATGOCT** abbiamo a disposizione la function **pcg**. **pcg**

Programma 5.4. cg: il metodo del gradiente coniugato (precondizionato)

```

function [x,relres,k,res]=cg(A,b,tol,kmax,RP,x0)
% CG metodo del gradiente coniugato (precondizionato).
% [X,RELRES,K,RES]=CG(A,B,TOL,KMAX,RP,X0)
% risolve il sistema A*X=B con il metodo del gradiente
% coniugatyo (precondizionato). TOL specifica la
% tolleranza per il test d'arresto. KMAX indica il
% numero massimo di iterazioni ammesse.
% RP contiene una matrice triangolare superiore t.c.
% il precondizionatore e' definito come P = RP'*RP.
% X0 e' il vettore iniziale.
% [X,RELRES,K,RES]=CG(A,B,TOL,KMAX,[ ],X0) richiama il
% metodo del gradiente coniugato non precondizionato.
% RELRES e' la norma del residuo corrispondente alla
% soluzione calcolata diviso la norma del vettore B.
% K e' il numero di iterazioni effettuate dal metodo,
% RES e' un vettore contenente la norma dei residui
% ad ogni passo del metodo

k = 0;
bnorm = norm(b);
res=[ ];
if bnorm==0
x=zeros(n,1); relres=0;
else
r=b-A*x0; x=x0;
% sistema sul precondizionatore
if ~isempty(RP); z=RP'\ (RP\r); else z=r; end
zr=z'*r; res0=sqrt(zr); relres=zr; d=z;
while relres > tol && k< kmax
    v=A*d; vd=v'*d;
    alpha=zr/vd;
    x=x+alpha*d;
    r=r-alpha*v;
    if ~isempty(RP); z=RP'\ (RP\r); else z=r; end
    beta=(v'*z)/vd;
    d=z-beta*d;
    zr=z'*r;
    relres=sqrt(zr)/res0;
    res=[res; relres];
    k=k+1;
end
end

```

Esempio 5.15 Consideriamo il sistema $A\mathbf{x} = \mathbf{b}$ con A di dimensione $n = 3600$ così definita: gli elementi sulla diagonale principale sono uguali a $4h$, con $h = 1/n$, quelli sulle diagonali in posizione $1, -1, 60$ e -60 sono uguali a $-h$, mentre \mathbf{b} è un vettore di elementi tutti uguali a 1. A risulta simmetrica e definita positiva. Richiamiamo i metodi del gradiente e del gradiente coniugato ponendo tolleranza per il test d'arresto pari all'*epsilon* di macchina ϵ_M , numero massimo di iterazioni $kmax=2000$, ed un vettore iniziale $\mathbf{x}^{(0)} = \mathbf{0}$. Le istruzioni **MAT&OCT** sono:

```

n1=60; n=3600; e=ones(n,1);
A=spdiags([-e,-e,4*e,-e,-e],[-n1,-1,0,1,n1],n,n);

```

```

h=1/n; A=A*h; b=e;
tol=eps; kmax=2000; x0=zeros(n,1);
[xg,relresg,itg,resg]=gradient(A,b,tol,kmax,[ ],x0);
[xcg,relrescg,itcg,resxg]=cg(A,b,tol,kmax,[ ],x0);

```

Dopo 2000 iterazioni il metodo del gradiente non è ancora giunto a convergenza (il residuo finale è $\text{relresg}=0.3062$), mentre il metodo del gradiente coniugato si è arrestato dopo 276 iterazioni producendo un residuo pari a $\text{relrescg}=1.8172\text{e-}16$.

Il numero di condizionamento di A , calcolato con il comando `cond(full(A))`, risulta $K(A) \simeq 3009.80$, quindi è opportuno precondizionare il sistema.

Scegliamo il precondizionatore calcolato con la fattorizzazione incompleta di Cholesky di A , sia di tipo *no-fill* che di tipo ICT con *drop tolerance* pari a $\epsilon_d = 10^{-2}$, $\epsilon_d = 10^{-3}$ e $\epsilon_d = 10^{-4}$. I comandi `MATGOCT` sono:

```

RP=ichol(A);
[x,relresg,itg,resg]=gradient(A,b,tol,kmax,RP,x0);
[x,relrescg,itcg,rescg]=cg(A,b,tol,kmax,RP,x0);

```

e, per $\epsilon_d = 10^{-2}$,

```

drop=1.e-2;
RP=ichol(A,struct('type','ict','droptol',drop));
[x,relresg,itg,resg]=gradient(A,b,tol,kmax,RP,x0);
[x,relrescg,itcg,rescg]=cg(A,b,tol,kmax,RP,x0);

```

In Tabella 5.4 riportiamo il numero di iterazioni richieste dai due metodi (per diversi precondizionatori) per convergere nei limiti della tolleranza fissata ed il numero di condizionamento della matrice $P^{-1}A$ nei vari casi. Come previsto, il numero di iterazioni diminuisce al diminuire di $K(P^{-1}A)$ e il metodo del gradiente coniugato possiede una velocità di convergenza sempre maggiore di quella del metodo del gradiente. Inoltre, se P è di tipo ICT con *drop tolerance*, al diminuire di quest'ultima diminuisce anche il numero di condizionamento e, di conseguenza, decresce anche il numero di iterazioni necessarie a soddisfare il test d'arresto. ■

Esempio 5.16 Riprendiamo l'Esempio 5.9 sulla matrice di Hilbert A_n e risolviamo il sistema per diversi valori di n con i metodi del gradiente e del gradiente coniugato. Prendiamo $\mathbf{x}^{(0)} = \mathbf{0}$ ed arrestiamo il metodo quando il residuo relativo (5.77) è minore dell'*epsilon* di macchina. Nella Tabella 5.5 riportiamo gli errori relativi (rispetto alla soluzione esatta) ottenuti con i metodi del gradiente e del gradiente coniugato e l'errore che si ottiene utilizzando il comando \ di `MATGOCT`. L'errore prodotto da tutti e tre i metodi degenera al crescere

Tabella 5.4. I risultati dell'Esempio 5.15

	$P = I$	$P = \text{ICHOL}$ <i>no-fill</i>	$P = \text{ICT}$ $\epsilon_d = 10^{-2}$	$P = \text{ICT}$ $\epsilon_d = 10^{-3}$	$P = \text{ICT}$ $\epsilon_d = 10^{-4}$
$K(P^{-1}A)$	3009.80	266.97	65.22	10.35	2.44
it. gradiente	>2000	>2000	1162	179	33
it. CG	276	96	50	22	12

Tabella 5.5. Risultati relativi all’Esempio 5.16

n	$K(A_n)$	Gradiente		Gradiente Coniugato	
		\ Errore	Errore	N.ro iterazioni	Errore
4	1.55e+04	4.08e-13	8.72e-03	>2000	6.08e-14
6	1.50e+07	3.38e-10	6.63e-03	>2000	6.54e-11
8	1.53e+10	2.22e-07	6.87e-03	>2000	2.57e-07
10	1.60e+13	1.74e-04	8.49e-03	>2000	5.43e-06
12	1.68e+16	1.72e-01	8.84e-03	>2000	1.69e-06
14	3.08e+17	1.12e+00	8.38e-03	>2000	5.01e-06

di n , in maniera proporzionale a $K(A)$ per il metodo diretto ed in maniera più contenuta per i metodi iterativi. Osserviamo che il metodo del gradiente non giunge mai a convergenza entro le 2000 iterazioni; dalla rappresentazione grafica del vettore dei residui si evince anche che il metodo sta convergendo molto lentamente. Il metodo del gradiente coniugato invece converge in un numero ragionevole di iterazioni (ma comunque maggiore della dimensione del sistema per effetto dell’alto numero di condizionamento della matrice A) producendo un errore relativo sulla soluzione esatta al più dell’ordine di 10^{-6} . ■

5.10.2 Il caso non simmetrico

Il metodo del gradiente coniugato è un esempio notevole della famiglia dei cosiddetti metodi di *Krylov*; tali metodi possono essere usati per la risoluzione di sistemi non necessariamente simmetrici (si vedano ad esempio [Axe94], [Saa03] e [vdV03], [FGN92]).

Data una matrice A di dimensione n ed un vettore $\mathbf{v} \in \mathbb{R}^n$, lo spazio

$$\mathcal{K}_k(A, \mathbf{v}) = \text{span}\{\mathbf{A}^j \mathbf{v}, j = 0, \dots, k-1\}, \quad k \geq 1, \quad (5.95)$$

è detto *sottospazio di Krylov* di dimensione k in \mathbb{R}^n .

I metodi di Krylov sono metodi iterativi che, a partire da un dato vettore $\mathbf{x}^{(0)}$ e dal corrispondente residuo $\mathbf{r}^{(0)} = \mathbf{b} - \mathbf{Ax}^{(0)}$, costruiscono i vettori

$$\mathbf{x}^{(k)} \in \mathbf{x}^{(0)} + \mathcal{K}_k(A, \mathbf{r}^{(0)}), \quad k \geq 1, \quad (5.96)$$

secondo due possibili criteri:

1. *approccio Ritz-Galerkin (o del residuo ortogonale)*: $\mathbf{x}^{(k)}$ è calcolato in modo tale che

$$\mathbf{r}^{(k)} \perp (\mathbf{x}^{(0)} + \mathcal{K}_k(A, \mathbf{r}^{(0)})) \quad (5.97)$$

(o più in generale $\mathbf{r}^{(k)} \perp \mathcal{S}_k$ con \mathcal{S}_k opportuno sottospazio di \mathbb{R}^n);

2. *approccio del minimo residuo*: $\mathbf{x}^{(k)}$ è costruito in modo tale che

$$\|\mathbf{r}^{(k)}\|_2 = \underset{\mathbf{y} \in \mathbf{x}^{(0)} + \mathcal{K}_k(A, \mathbf{r}^{(0)})}{\operatorname{argmin}} \|\mathbf{b} - \mathbf{Ay}\|_2. \quad (5.98)$$

Esempi di metodi basati sul criterio (5.97) sono il metodo del gradiente coniugato se A è simmetrica definita positiva, il metodo Bi-CG [Fle76] nel caso in cui A sia generica, mentre l'esempio più noto ottenuto implementando il criterio (5.98) è il metodo GMRES (*Generalized Minimum Residual*) [SS86].

Se A è non simmetrica o simmetrica ma indefinita (ovvero ammette autovalori reali sia negativi che positivi), non è garantito che il problema (5.97) ammetta soluzione ad ogni passo k ; corrispondentemente si potrebbe verificare il cosiddetto *breakdown* del metodo. Per ovviare a questo inconveniente sono stati proposti metodi ibridi, le cui iterate (5.96) sono calcolate seguendo entrambi gli approcci (5.97) e (5.98). Un esempio di metodo ibrido è il Bi-CGSTab [vdV03].

Se nel caso di A simmetrica definita positiva non vi sono dubbi su quale sia il metodo più efficiente (esso sarà il gradiente coniugato), nel caso di matrici non simmetriche o simmetriche ma indefinite la scelta è meno ovvia.

Fra i metodi di Krylov, quello in genere più robusto è il GMRES. Come il metodo del gradiente coniugato esso gode della proprietà di terminazione finita, ossia in aritmetica esatta restituisce la soluzione esatta del sistema in un numero finito di iterazioni. Tuttavia, poichè ad ogni iterazione bisogna costruire una base ortogonale per il sottospazio $\mathcal{K}_k(A, \mathbf{r}^{(0)})$, quando la dimensione n del sistema è molto grande e anche k è grande, questo calcolo può diventare computazionalmente molto oneroso e richiedere molta memoria.

Per contenere le esigenze di memoria e di risorse di calcolo, il metodo GMRES è di solito fatto ripartire dopo m iterazioni, dando così luogo al cosiddetto metodo GMRES con *restart* o GMRES(m). Non esiste una regola precisa per determinare il valore ottimale del parametro m di *restart* per ogni matrice A ; inoltre il numero di iterazioni richieste dal metodo GMRES(m) per soddisfare il test d'arresto può variare molto sensibilmente anche cambiando di una sola unità il valore di m .

Nel metodo GMRES(m) si distinguono iterazioni interne ed iterazioni esterne; il contatore delle iterazioni esterne è incrementato ad ogni *restart*, quindi ogni m iterazioni interne. Si noti che ogni iterazione interna del metodo GMRES richiede il calcolo di un prodotto matrice-vettore.

Il metodo Bi-CGSTab, anch'esso a terminazione finita in aritmetica esatta, contiene sensibilmente i costi computazionali rispetto al metodo GMRES e, come già anticipato, si basa su un approccio ibrido, cioè sfrutta entrambi i criteri (5.97) e (5.98). Ogni iterazione del metodo Bi-CGSTab richiede il calcolo di due prodotti matrice-vettore e quindi costa all'incirca come due iterazioni del metodo del gradiente coniugato.

Il metodo Bi-CGSTab ed il metodo GMRES sono implementati in **MAT&OCT** nelle *function* **bicgstab** e **gmres**, rispettivamente. Anche questi due metodi hanno una corrispondente versione precondizionata.

bicgstab
gmres

Un precondizionatore per matrici non simmetriche simile al precondizionatore di Cholesky incompleto è ottenuto mediante la fattorizzazione LU incompleta della matrice A: si costruiscono due matrici triangolari L_P (inferiore) e U_P (superiore) utilizzando tecniche molto simili a quelle descritte nella Sezione 5.10.1 per la fattorizzazione incompleta di Cholesky senza *fill-in* o con *drop tolerance*, quindi si definisce $P = L_P U_P$. La **ilu** fattorizzazione incompleta LU di A è implementata nella *function* **ilu** di **MAT&OCT**. Ad esempio con l'istruzione

```
[LP,UP]=ilu(A,struct('type','ilutp','droptol',1e-03));
```

costruiamo L_P e U_P con le tecniche del *threshold dropping* e della pivotazione, e con *drop tolerance* $\varepsilon_d = 10^{-3}$, mentre con

```
[LP,UP]=ilu(A);
```

calcoliamo la fattorizzazione LU incompleta nella versione *no-fill*.

Le istruzioni per richiamare i metodi Bi-CGStab e GMRES in **MAT&OCT** sono:

```
[x,flag,relres,it,resvec]=bicgstab(A,b,tol,kmax,...  
LP,UP,x0)  
[x,flag,relres,it,resvec]=gmres(A,b,m,tol,...  
kmax,LP,UP,x0)
```

In input A e b indicano matrice e termine noto del sistema da risolvere, m (solo per il GMRES) contiene il parametro di *restart* del metodo, tol e kmax rappresentano tolleranza e numero massimo di iterazioni per il test d'arresto, LP e UP sono i fattori triangolare inferiore e superiore del precondizionatore P, e infine x0 è il vettore iniziale. In output, x è il vettore soluzione, flag sarà pari a 0 se il metodo è giunto a convergenza (si veda l'*help* del comando per gli altri valori), relres è il residuo relativo che compare in (5.99), it contiene informazioni sul numero di iterazioni effettuate, resvec è il vettore delle norme dei residui $\|\mathbf{r}^{(k)}\|$ in MATLAB e dei residui relativi $\|\mathbf{r}^{(k)}\|/\|\mathbf{r}^{(0)}\|$ in Octave. Qualora si richiami la *function* bicgstab, la variabile it contiene esattamente il numero di iterazioni effettuate, mentre per la *function* gmres, it è un vettore di due componenti ed il numero totale di iterazioni interne svolte è $it_interne=1+m*(it(1)-1)+it(2)$.

Un confronto computazionale tra i due metodi può essere realizzato contando il numero totale di prodotti matrice-vettore svolti: esso è pari al doppio del numero di iterazioni per il metodo Bi-CGStab e pari al numero di iterazioni interne per il metodo GMRES.

Esempio 5.17 Risolviamo il sistema $\mathbf{Ax} = \mathbf{b}$, in cui

$$A = \begin{pmatrix} 6 & 1 & -2 & 2 & 1 \\ 0 & -3 & 3 & -2 & 1 \\ 2 & 0.5 & 5 & -1 & -2 \\ 0 & 1 & 2 & -3 & 2 \\ 0.5 & -1 & 1 & 0.4 & 2 \end{pmatrix} \quad \mathbf{b} = \begin{pmatrix} 15 \\ 0 \\ 4 \\ 6 \\ 13.1 \end{pmatrix}$$

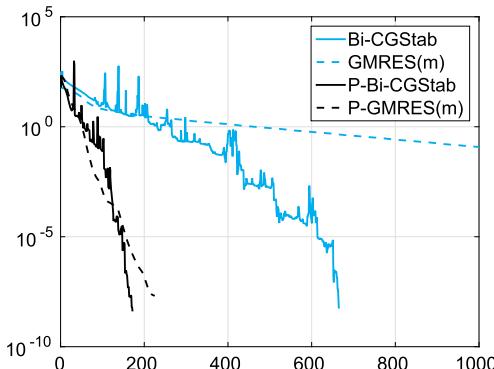


Figura 5.20. I residui relativi ottenuti durante la risoluzione del sistema dell’Esempio 5.18. In ascissa abbiamo un contatore k che si incrementa ogni qualvolta gli algoritmi eseguono un prodotto matrice-vettore, quindi due volte per iterazione per il metodo Bi-CGStab e una volta per iterazione per il metodo GMRES

con i metodi Bi-CGStab e GMRES non precondizionati. Abbiamo considerato un vettore iniziale di numeri casuali, tolleranza per il test d’arresto pari a $\text{tol}=1.e-14$ e numero massimo di iterazioni pari a $\text{kmax}=100$ (osservando che nel caso del metodo GMRES, kmax rappresenta il numero massimo di iterazioni interne). Abbiamo fatto variare il parametro m di *restart* del GMRES(m) tra 1 e 4, nella Tabella sottostante riportiamo il numero di prodotti matrice-vettore svolti dai due metodi fino al soddisfacimento del test d’arresto.

Bi-CGStab	GMRES(4)	GMRES(3)	GMRES(2)	GMRES(1)
12	16	126	>201	>101

Esempio 5.18 Consideriamo il sistema $\mathbf{Ax} = \mathbf{b}$ con \mathbf{A} di dimensione $n = 3600$ così definita: gli elementi sulla diagonale principale sono uguali a $4h^2 + 0.01h$, con $h = 1/\sqrt{n}$, quelli sulle diagonali in posizione 1, 60 e -60 sono uguali a $-h^2$, quelli sulla diagonale in posizione -1 sono uguali a $-h^2 - 0.01h$, mentre \mathbf{b} è un vettore di elementi tutti uguali a 1. Richiamiamo i metodi Bi-CGStab e GMRES(m) con $m = \sqrt{n}$, sia nella versione non precondizionata, sia con precondizionatore ottenuto con fattorizzazione incompleta LU della matrice \mathbf{A} senza *fill-in*. Fissiamo tolleranza per il test d’arresto pari a 10^{-10} , numero massimo di iterazioni $\text{kmax}=2000$ ed un vettore iniziale $\mathbf{x}^{(0)} = \mathbf{0}$. Il metodo Bi-CGStab converge in 332 iterazioni e la sua versione precondizionata in 85 iterazioni, mentre il metodo GMRES(m) converge in 5240 iterazioni e la sua versione precondizionata in 225 iterazioni. In Figura 5.20 è mostrata l’evoluzione dei residui relativi ai variare delle iterazioni. ■

Si vedano gli Esercizi 5.15–5.22.



5.11 Quando conviene arrestare un metodo iterativo

Teoricamente, per convergere alla soluzione esatta, i metodi iterativi necessitano di un numero infinito di iterazioni. Anche quando non è così (ad esempio nel caso del metodo del Gradiente Coniugato) il numero di iterazioni richieste è comunque molto elevato per sistemi lineari di dimensione considerevole.

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, si veda (5.53). Più precisamente, potremmo arrestare il nostro metodo iterativo al primo passo k_{\min} in corrispondenza del quale

$$\frac{\|\mathbf{r}^{(k_{\min})}\|}{\|\mathbf{b}\|} \leq \varepsilon. \quad (5.99)$$

Ponendo $\hat{\mathbf{x}} = \mathbf{x}^{(k_{\min})}$ e $\mathbf{r} = \mathbf{r}^{(k_{\min})}$ in (5.37) 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.

In alternativa a (5.99) si può utilizzare il seguente test d'arresto (come già abbiamo fatto nei Programmi 5.3 e 5.4)

$$\frac{\|\mathbf{r}^{(k_{\min})}\|}{\|\mathbf{r}^{(0)}\|} \leq \varepsilon. \quad (5.100)$$

Esso è solitamente implementato per i metodi del gradiente e del gradiente coniugato in quanto il vettore residuo è disponibile ad ogni iterazione; la sua determinazione non richiede pertanto calcoli aggiuntivi.

Esempio 5.19 Consideriamo il sistema lineare (5.1) in cui $\mathbf{A} = \mathbf{A}_{20}$ è la matrice di Hilbert di dimensione 20 introdotta nell'Esempio 5.9 e \mathbf{b} viene costruito in modo tale che la soluzione esatta sia $\mathbf{x} = (1, 1, \dots, 1)^T$. Essendo \mathbf{A} simmetrica e definita positiva, il metodo di Gauss-Seidel sicuramente converge. Utilizziamo il Programma 5.2 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 relativo calcolato è pari a 0.0586. Questo

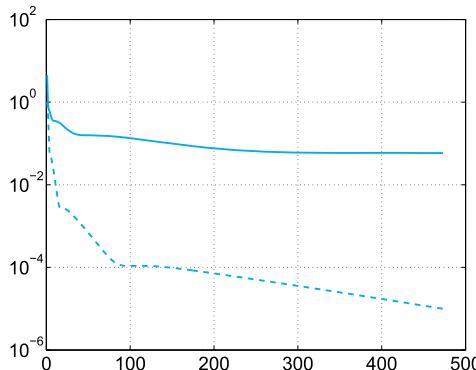


Figura 5.21. Andamento (al variare di k) del residuo relativo (5.77) (in linea tratteggiata) e dell'errore $\|\mathbf{x} - \mathbf{x}^{(k)}\|/\|\mathbf{x}\|$ (in linea continua) per il metodo di Gauss-Seidel applicato al sistema di Hilbert dell'Esempio 5.19

comportamento è dovuto al fatto che la matrice è estremamente mal condizionata, essendo $K(\mathbf{A}) \simeq 10^{17}$. In Figura 5.21 viene mostrata l'evoluzione della norma del residuo (diviso per la norma del residuo iniziale) e quella dell'errore relativo 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, un metodo iterativo del tipo (5.50) viene arrestato al primo passo k_{\min} per il quale

$$\|\delta^{(k_{\min})}\| \leq \varepsilon. \quad (5.101)$$

Nel caso particolare in cui \mathbf{B} sia simmetrica e definita positiva, avremo

$$\|\mathbf{e}^{(k)}\| = \|\mathbf{e}^{(k+1)} + \delta^{(k)}\| \leq \rho(\mathbf{B})\|\mathbf{e}^{(k)}\| + \|\delta^{(k)}\|$$

e, dovendo avere $\rho(\mathbf{B}) < 1$ per garantire la convergenza, possiamo scrivere

$$\|\mathbf{e}^{(k)}\| \leq \frac{1}{1 - \rho(\mathbf{B})} \|\delta^{(k)}\| \quad (5.102)$$

Si può quindi concludere che il controllo dell'incremento è significativo soltanto se $\rho(\mathbf{B})$ 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 \mathbf{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 (5.102).

Qualora si sia interessati a valutare l'errore relativo, il test (5.101) può essere sostituito da

$$\frac{\|\delta^{(k_{\min})}\|}{\|\mathbf{b}\|} \leq \varepsilon$$

e, a sua volta, (5.102) da

$$\frac{\|\mathbf{e}^{(k)}\|}{\|\mathbf{b}\|} \leq \frac{1}{1 - \rho(\mathbf{B})} \varepsilon.$$

Esempio 5.20 Consideriamo un sistema con matrice $\mathbf{A} \in \mathbb{R}^{50 \times 50}$ tridiagonale simmetrica avente elementi sulla diagonale principale pari a 2.001 e quelli sulla sopra e sottodiagonale 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 \mathbf{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 5.2 per risolvere il sistema con l'accortezza di sostituire al criterio d'arresto basato sul residuo quello basato sull'incremento, ovvero $\|\delta^{(k)}\| \leq \varepsilon$. Partendo da un vettore iniziale di componenti $(\mathbf{x}_0)_i = 10 \sin(100i)$ (per $i = 1, \dots, n$) e richiedendo una tolleranza $\text{tol} = 10^{-5}$, il programma restituisce dopo ben 859 iterazioni una soluzione affetta da un errore $\|\mathbf{e}^{(859)}\| \simeq 0.0021$. La convergenza è molto lenta e l'errore è piuttosto grande poiché 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 $\|\mathbf{e}^{(17)}\| \simeq 8.96 \cdot 10^{-6}$: in questo caso infatti il raggio spettrale della matrice di iterazione è pari a 0.4428 (si veda la Proposizione 5.2). ■

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. il metodo del gradiente coniugato per matrici simmetriche definite positive ha la proprietà di terminazione finita, ovvero in aritmetica esatta esso converge alla soluzione esatta in un numero di iterazioni minore o uguale alla dimensione del sistema. Esso appartiene alla famiglia dei metodi di Krylov, di cui fanno parte anche i metodi Bi-CGStab e GMRES per matrici generiche (anche questi godono della proprietà di terminazione finita);

6. i precondizionatori sono opportune matrici non singolari utilizzate in un metodo iterativo per accelerarne la convergenza, previa riduzione del numero di condizionamento della matrice del sistema;
7. 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 .

5.12 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, per le quali 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 dalle proprietà della matrice del sistema lineare in questione (quali la simmetria, la definita positività, la sparsità, il numero di condizionamento) 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® Core™ i7-4790 3.60 GHz con 4 core e 16 GB di RAM.

Test 1: un sistema lineare sparso con banda piccola

In questo primo caso test risolveremo sistemi sparsi generati dalla discretizzazione con il metodo delle differenze finite del problema di Poisson sul quadrato $(-1, 1)^2$ con condizioni al bordo di Dirichlet omogenee (si veda la Sezione 9.7). In particolare, le matrici verranno generate a partire da una decomposizione del quadrato in reticolati uniformi di passo $h = 2/(N + 1)$ lungo entrambe le direzioni, per diversi valori di N . Le corrispondenti matrici alle differenze finite, con $(N + 2)^2$ righe e colonne, sono generate attraverso il Programma 9.3. Il grafico a sinistra di Figura 5.22 riporta il *pattern* della matrice di dimensione $(N + 2)^2 = 256$ (ottenuta con il comando `spy`): come si vede si tratta di una matrice sparsa con 5 elementi non nulli per riga. Dopo aver eliminato le righe e le colonne associate ai nodi di bordo, denotiamo con $n = N^2$ la dimensione della matrice ridotta. Le matrici ottenute dopo questa riduzione sono tutte simmetriche e definite positive ed hanno un numero di condizionamento che si comporta come h^{-2} , sono quindi tanto più mal condizionate

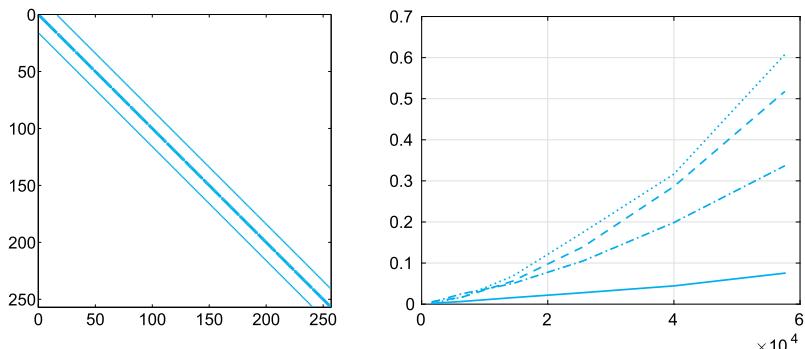


Figura 5.22. Caso Test 1. Il *pattern* della matrice (*a sinistra*). Il confronto fra tempi di CPU (in sec.) necessari per la risoluzione dei sistemi lineari (*a destra*): la *linea continua* corrisponde al comando `\`, la *linea punteggiata* alla fattorizzazione di Cholesky, la *linea tratteggiata* al metodo del gradiente coniugato precondizionato e la *linea tratto-punto* al metodo del gradiente coniugato non precondizionato. In ascissa è riportata la dimensione delle matrici

quanto più h è piccolo. Per risolvere i corrispondenti sistemi lineari confronteremo il metodo di fattorizzazione di Cholesky ed il metodo del gradiente coniugato precondizionato con una fattorizzazione incompleta di Cholesky di tipo ICT e *drop tolerance* $\varepsilon_d = 10^{-3}$ (si veda la Sezione 5.10.1), nonché il metodo implementato nel comando `\` di **MAT&OCT** che, in tal caso, si traduce in un metodo *ad hoc* per matrici a banda simmetriche.

Per il metodo del gradiente coniugato precondizionato richiederemo che a convergenza il residuo relativo (5.77) sia minore di 10^{-13} e congeggeremo nel tempo di calcolo anche il tempo necessario per costruire il precondizionatore.

Nel grafico di destra di Figura 5.22 confrontiamo i tempi di calcolo dei tre metodi in esame al crescere della dimensione della matrice. Come si 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.

La versione precondizionata del metodo del gradiente coniugato risulta meno efficiente della versione non precondizionata, anche se il numero di iterazioni richieste dalla versione precondizionata è inferiore. Ad esempio nel caso in cui si consideri $n = 4096$ (corrispondente a $N = 64$) la versione precondizionata richiede 20 iterazioni, mentre quella non precondizionata 150. Entrambe le versioni sono più convenienti della fattorizzazione di Cholesky. Avvertiamo il lettore che le conclusioni tratte da questo esempio devono essere prese in considerazione con raziocinio, in quanto esse dipendono fortemente dall'implementazione degli algoritmi (quindi dal *software*) e dal tipo di computer utilizzato.

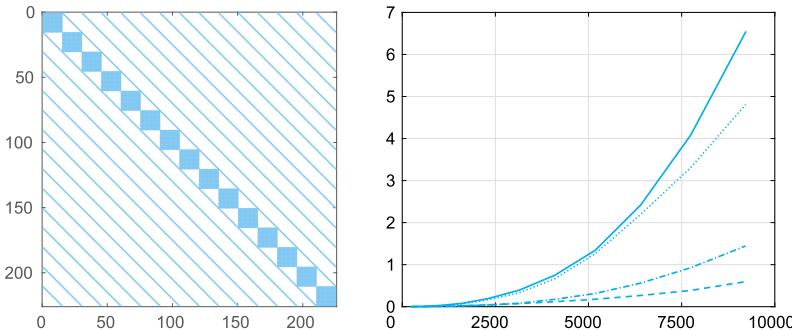


Figura 5.23. Caso Test 2. Il *pattern* di una delle matrici (*a sinistra*). Il confronto fra tempi di CPU (in sec.) necessari per la risoluzione dei sistemi lineari (*a destra*): la *linea continua* corrisponde al comando `\`, la *linea punteggiata* alla fattorizzazione di Cholesky, la *linea tratteggiata* al metodo del gradiente coniugato precondizionato e la *linea tratto-punto* al metodo del gradiente coniugato non precondizionato. In ascissa è riportata la dimensione delle matrici

Test 2: un sistema lineare con banda estesa

I sistemi lineari che consideriamo ora si riferiscono ancora alla discrinetizzazione del problema di Poisson sul quadrato $(-1, 1)^2$, ma sono stati generati impiegando una discrinetizzazione basata su metodi spettrali con formule di quadratura di tipo Gauss-Legendre-Lobatto (si vedano ad esempio [Qua16] e [CHQZ06]). Anche se il numero di punti che formano il reticolo è lo stesso utilizzato con le differenze finite, l'approssimazione delle derivate di una funzione mediante metodi spettrali coinvolge molti più nodi (infatti, la derivata lungo la direzione x in un dato nodo è approssimata usando tutti i nodi che giacciono sulla stessa riga, mentre i nodi che giacciono sulla stessa colonna sono utilizzati per calcolarne le derivate lungo la direzione y). Le matrici sono ancora sparse e strutturate, ma il numero di elementi non nulli è decisamente superiore rispetto al caso di approssimazione con differenze finite. Per dare un'idea abbiamo riportato in Figura 5.23, a sinistra la struttura di una matrice spettrale di dimensione $N^2 = 256$ in cui gli elementi non nulli sono 7936 contro i 1216 della matrice alle differenze finite riportata in Figura 5.22.

I tempi di calcolo riportati nel grafico di destra in Figura 5.23 mostrano come per questo sistema il metodo del gradiente coniugato precondizionato con una fattorizzazione incompleta di Cholesky (nella versione ICT con *drop tolerance* $\varepsilon_d = 10^{-4}$), 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 banda grande, il metodo del gradiente coniugato precondizionato risulti più efficiente del miglior metodo diretto implementato in **MATLAB**. Questo naturalmente a patto di disporre di un efficiente precondizionatore, requisito non sempre facilmente realizzabile.

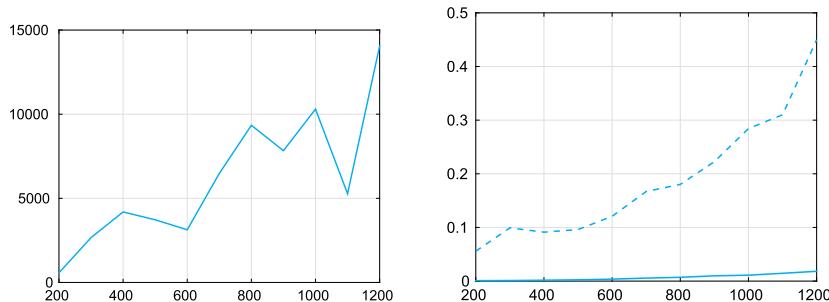


Figura 5.24. Caso Test 3. Il numero di condizionamento della matrice di Riemann (a sinistra). Il confronto tra i tempi di CPU (in sec.) (a destra): la linea continua corrisponde al comando `\` e la linea tratteggiata al metodo iterativo GMRES(m) non precondizionato. In ascissa è riportata la dimensione delle matrici

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.

Test 3: un sistema con matrice piena

In `MAT&OCT` 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 piena e non simmetrica $n \times n$ tale che $\det(A) = \mathcal{O}(n!n^{-1/2+\epsilon})$ per ogni $\epsilon > 0$. Risolviamo il sistema lineare associato con il metodo iterativo GMRES (si veda la Sezione 5.10.2). Per arrestare il metodo richiederemo che il residuo relativo (5.77) sia minore di 10^{-10} . In alternativa, come metodo diretto useremo il comando `\` di `MAT&OCT` che, in questo caso, implementa la fattorizzazione LU.

Al crescere di n risolviamo i sistemi lineari di matrice A e termine noto costruito in modo tale che la soluzione esatta del sistema sia $\mathbf{x} = (1, 1, \dots, 1)^T$. Abbiamo richiamato il metodo GMRES(m) non precondizionato con *restart* $m = n/10$. In Figura 5.24, a destra, riportiamo i tempi di CPU (in sec.) ottenuti per n compresi tra 200 e 1200. Nella stessa figura, a sinistra, riportiamo il numero di condizionamento di A calcolato con il comando `cond(A)`. Come si vede il metodo diretto è più favorevole del metodo GMRES(m) non precondizionato.

Test 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 riportato nella (9.22) per il caso monodimensionale. Come metodo di approssi-

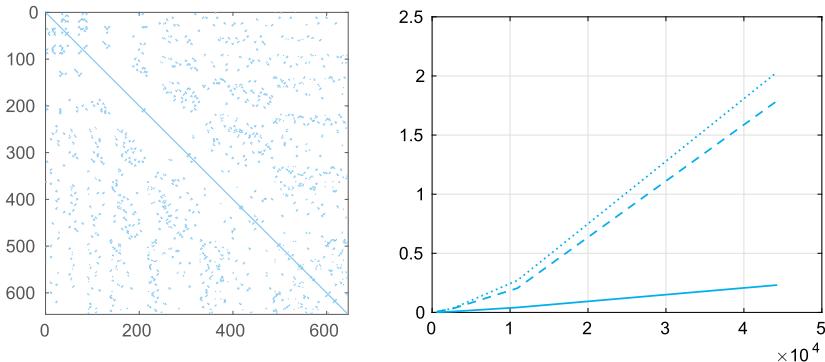


Figura 5.25. Caso Test 4. Il *pattern* della matrice per $h = 0.0125$ (*a sinistra*). Il confronto fra tempi di CPU (in sec.) (*a destra*): la *linea continua* corrisponde al comando `\`, la *linea tratteggiata* al metodo iterativo Bi-CGStab non precondizionato e la *linea punteggiata* al metodo iterativo Bi-CGStab precondizionato con fattorizzazione LU incompleta. In ascissa è riportata la dimensione delle matrici

simazione utilizziamo gli elementi finiti lineari (che verranno introdotti, sempre per il caso monodimensionale, nella Sezione 9.5). Essi utilizzano polinomi composti lineari per rappresentare la soluzione in ogni elemento triangolare di una griglia che partiziona la regione dove è assegnato il problema ai limiti. Le incognite del sistema lineare associato sono i valori della soluzione nei vertici dei triangoli. Per una descrizione del metodo e per la costruzione della matrice associata rimandiamo per esempio a [QV94, Qua16]. Osserviamo solo che la matrice è sparsa, ma non è a banda (il *pattern* dipende da come sono stati numerati i vertici ed è riportato in Figura 5.25 a sinistra) e non è simmetrica per la presenza del termine di trasporto. La mancanza di simmetria della matrice non è evidente dalla rappresentazione della sola struttura.

Minore sarà il *diametro* h dei triangoli (ossia la lunghezza del lato maggiore), maggiore sarà la dimensione del sistema lineare da risolvere. Stiamo utilizzando griglie non strutturate di triangoli generate, ad esempio, con il *toolbox* `pdetool` di MATLAB. Abbiamo confrontato i tempi di CPU necessari per risolvere i sistemi che si trovano per h pari a 0.1, 0.05, 0.025 e 0.0125. Abbiamo utilizzato il comando `\` (che in questo caso richiama la libreria UMFPACK) ed il metodo iterativo Bi-CGStab, sia nella versione non precondizionata che precondizionata con fattorizzazione LU incompleta di tipo ILUTP e *drop tolerance* $\varepsilon_d = 10^{-3}$ (si veda la Sezione 5.10.2).

Come si vede il metodo diretto è in questo caso decisamente più conveniente del metodo iterativo.

In Tabella 5.6 abbiamo riportato la dimensione del sistema lineare ed il numero di iterazioni richieste dal metodo Bi-CGStab non precondi-

Tabella 5.6. Caso Test 4. Il numero di iterazioni del metodo Bi-CGStab non precondizionato e precondizionato con fattorizzazione incompleta LU

h	n	it. Bi-CGStab	it. Bi-CGStab precondizionato
0.1	725	61	4
0.05	2849	128	7
0.025	11261	298	13
0.0125	44795	596	26

zionato e precondizionato al variare del diametro h della mesh. Usando come precondizionatore per il metodo Bi-CGStab una fattorizzazione LU incompleta il numero di iterazioni diminuisce, ma come si evince dai grafici di Figura 5.25 a destra, i tempi di calcolo del metodo iterativo crescono rispetto al caso non precondizionato.

In conclusione

Il confronto considerato, seppur specifico, mette in luce alcuni aspetti importanti: i metodi diretti nelle loro versioni più sofisticate (come quelle che si trovano nelle librerie PARDISO, SUPERLU, o in UMFPACK, quest'ultima richiamata dal comando \ di `MAT&OCT` quando la matrice è sparsa) in generale sono computazionalmente più efficienti dei metodi iterativi quando per questi ultimi non si usino precondizionatori efficaci. Soffrono tuttavia in modo maggiore il malcondizionamento delle matrici (si veda anche l'Esempio 5.16) 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 calcolare il prodotto matrice vettore su un vettore noto. Questo aspetto rende i metodi iterativi particolarmente interessanti in quei problemi in cui la matrice non si genera esplicitamente.

5.13 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 già accennato, dal comando \ di `MAT&OCT` in certe circostanze. Per approfondimenti si vedano [GL96, Dav06, DD99].

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 [Hac16]).

5.14 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 `magic(n)`, di `MAT&OCT` $n = 3, 4, \dots, 500$, `magic` 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 nella Sezione 1.4 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 (5.16) non soddisfa le ipotesi della Proposizione 5.1? Per quali valori di ε 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{x}_j di A^{-1} soddisfa il sistema lineare $A\mathbf{x}_j = \mathbf{e}_j$, 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 5.8 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

$$A = \begin{bmatrix} 2 & -2 & 0 \\ \varepsilon - 2 & 2 & 0 \\ 0 & -1 & 3 \end{bmatrix},$$

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 $\varepsilon \rightarrow 0$. Si verifichi che la soluzione numerica del sistema lineare ottenuta tramite il processo di fattorizzazione non è affetta da errori di arrotondamento qualora si considerino $\varepsilon = 10^{-k}$, con $k = 0, \dots, 9$, e $\mathbf{b} = (0, \varepsilon, 2)^T$. Inoltre si analizzi il comportamento dell'errore relativo sulla soluzione esatta al variare di $\varepsilon = 1/3 \cdot 10^{-k}$, con $k = 0, \dots, 9$, sapendo che la soluzione esatta è $\mathbf{x}_{ex} = (\log(5/2), 1, 1)^T$.

Esercizio 5.9 Si considerino i sistemi lineari $A_i \mathbf{x}_i = \mathbf{b}_i$, $i = 1, 2, 3$, con

$$A_1 = \begin{bmatrix} 15 & 6 & 8 & 11 \\ 6 & 6 & 5 & 3 \\ 8 & 5 & 7 & 6 \\ 11 & 3 & 6 & 9 \end{bmatrix}, \quad A_i = (A_1)^i, \quad i = 2, 3,$$

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

$$A = \begin{bmatrix} -10 & 2 \\ \beta & 5 \end{bmatrix}. \quad (5.103)$$

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 $\mathbf{Ax} = \mathbf{b}$ 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. Si ripeta l'esercizio per la matrice $A = \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix}$.

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 (5.76), posto $\mathbf{x}^{(0)} = (1, 1/2)^T$.

Esercizio 5.16 Si dimostri (5.60) e che

$$\rho(B_{\alpha_{opt}}) = \frac{\lambda_{\max} - \lambda_{\min}}{\lambda_{\max} + \lambda_{\min}} = \frac{K(P^{-1}A) - 1}{K(P^{-1}A) + 1}. \quad (5.104)$$

Esercizio 5.17 Provare che, se A e P sono matrici simmetriche e definite positive, allora $P^{-1}A$ è simile ad una matrice simmetrica e definita positiva.

Esercizio 5.18 Si osservi che, usando un parametro di accelerazione generico (invece di α_k), dalla (5.68) avremmo $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha \mathbf{d}^{(k)}$ e pertanto l'errore $\mathbf{e}^{(k+1)} = \mathbf{x} - \mathbf{x}^{(k+1)}$ dipenderebbe da α . Si provi che il valore α_k definito in (5.70) è quello che minimizza sia la funzione $\Phi(\mathbf{x}^{(k)} + \alpha \mathbf{d}^{(k)})$, sia la funzione $\varphi(\alpha) = \|\mathbf{e}^{(k+1)}\|_A^2$ al variare di $\alpha \in \mathbb{R}$.

Esercizio 5.19 Dimostrare che se in un metodo di discesa del tipo (5.68), α_k è scelto come in (5.70) allora vale (5.72).

Esercizio 5.20 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.

Esercizio 5.21 Dati

$$A = \begin{bmatrix} 7 & 1 & 2 & 4 \\ 1 & 10 & 4 & 3 \\ 2 & 4 & 6 & 2 \\ 4 & 3 & 2 & 7 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 14 \\ 18 \\ 14 \\ 17 \end{bmatrix},$$

ed $\alpha \in \mathbb{R}$, si consideri il seguente metodo iterativo per la risoluzione del sistema $\mathbf{Ax} = \mathbf{b}$:

$$\begin{cases} \mathbf{x}^{(0)} \\ \mathbf{x}^{(k+1)} = (I - \alpha A)\mathbf{x}^{(k)} + \alpha \mathbf{b}, \end{cases} \quad k \geq 0. \quad (5.105)$$

Dopo aver osservato che vale l'identità $\lambda_i(I - \alpha A) = 1 - \alpha \lambda_i(A)$ per ogni $\alpha \in \mathbb{R}$, per $i = 1, \dots, n$ ($\lambda_i(A)$ sono gli autovalori di A), dire per quali valori di $\alpha \in \mathbb{R}$ il metodo iterativo (5.105) converge.

Scrivere un programma `MAT&OCT` per la risoluzione di $\mathbf{Ax} = \mathbf{b}$ con il metodo (5.105). Prendendo un vettore iniziale di numeri casuali e una tolleranza per il test d'arresto sul residuo relativo pari a 10^{-12} , verificare numericamente quanto trovato al punto precedente.

Esercizio 5.22 Siano

$$A = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 4 \\ 500 & -1.5 & -0.002 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 9 \\ 5 \\ 500 \end{bmatrix} \quad (5.106)$$

e $\hat{\mathbf{b}} = \mathbf{b} + 10^{-3}[1, 1, 1]^T$ una perturbazione di \mathbf{b} . Denotate con \mathbf{x} e $\hat{\mathbf{x}}$, rispettivamente, le soluzioni dei sistemi $A\mathbf{x} = \mathbf{b}$ (originario) e $A\hat{\mathbf{x}} = \hat{\mathbf{b}}$ (perturbato), stimare a priori l'errore tra la soluzione perturbata $\hat{\mathbf{x}}$ e quella del problema originario \mathbf{x} . Verificare tale stima dopo aver risolto i due sistemi con un metodo diretto. Il sistema dato può essere risolto con una fattorizzazione LU senza pivotazione? Quali metodi iterativi possiamo utilizzare per risolvere il sistema?

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* $\mathbf{x}^H A \mathbf{x} / \|\mathbf{x}\|^2$, dove $\mathbf{x}^H = \bar{\mathbf{x}}^T$ 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.

Ricordiamo 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

$$U^{-1}AU = \Lambda = \text{diag}(\lambda_1, \dots, \lambda_n). \quad (6.2)$$

Le colonne di U sono gli autovettori di A e formano una base per \mathbb{C}^n .

Nel caso speciale in cui A sia una matrice diagonale o triangolare, gli autovalori sono dati direttamente dagli elementi diagonali. Qualora 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.

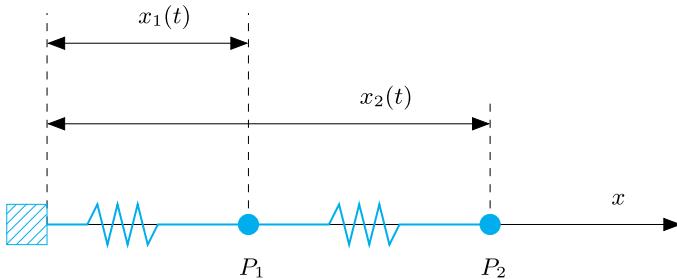


Figura 6.1. Un sistema di due corpi puntiformi di ugual massa collegati da molle

6.1 Alcuni problemi

Problema 6.1 (Molle elastiche) 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

$$m\ddot{x}_1 = K(x_2 - x_1) - Kx_1, \quad m\ddot{x}_2 = K(x_1 - x_2),$$

dove K è il coefficiente di elasticità di entrambe le molle. Siamo interessati alle oscillazioni libere 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

$$-ma_1\omega^2 = K(a_2 - a_1) - Ka_1, \quad -ma_2\omega^2 = K(a_1 - a_2). \quad (6.3)$$

Questo è 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 , (6.3) 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 (Dinamica delle popolazioni) 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

$$\begin{aligned}x_{i+1}^{(t+1)} &= x_i^{(t)} s_i, \quad i = 0, \dots, n-1, \\x_0^{(t+1)} &= \sum_{i=0}^n x_i^{(t)} m_i.\end{aligned}$$

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 nella Sezione 6.2 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 6.3 (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. ■

Problema 6.4 (Compressione di immagini) Il problema della compressione di un'immagine può essere affrontato utilizzando la decomposizione in valori singolari introdotta in (5.46). In effetti, un'immagine in bianco e nero può essere memorizzata in una matrice \mathbf{A} rettangolare $n \times m$ 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 in posizione ij ,

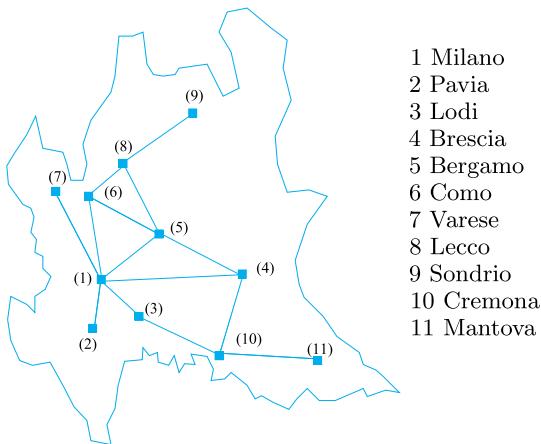


Figura 6.2. Una rappresentazione schematica delle connessioni ferroviarie in Lombardia fra i capoluoghi di provincia

(1 se bianco, 0 se nero). Grazie alla decomposizione in valori singolari (5.46) di A avremo allora che

$$A = \sigma_1 \mathbf{u}_1 \mathbf{v}_1^T + \sigma_2 \mathbf{u}_2 \mathbf{v}_2^T + \dots + \sigma_p \mathbf{u}_p \mathbf{v}_p^T, \quad (6.4)$$

avendo denotato 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 (6.4) 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 da un dispositivo ad un altro (computer, smartphone...) 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 6.10 vedremo in azione questa tecnica. ■

Problema 6.5 (Ricerca di informazioni sul web) I motori di ricerca scandiscono con regolare frequenza il web rilevando pagine nuove e aggiungendole ai propri archivi, in un secondo momento elaborano tutte le pagine censite al fine di compilare un enorme indice di tutte le parole individuate e delle relative posizioni su ciascuna pagina.

Quando digitiamo una o più parole chiave sul web utilizzando un motore di ricerca (come ad esempio Google, Yahoo! o Bing), quest'ultimo ci restituisce in un tempo brevissimo, spesso inferiore al secondo, un elenco di indirizzi di pagine web, detti URL (*Uniform Resource Locator*), ordinati in funzione della loro rilevanza e dell'attinenza con le parole chiave che abbiamo digitato. Questo ordine è stabilito tenendo in considerazione molti fattori.

Nel caso specifico di Google, uno dei fattori che vengono valutati (di fatto il più noto) è il *PageRank*: esso misura l'importanza di una pagina web in base ai *link* presenti in altre pagine che rimandano ad essa; più grande è il *PageRank* di una pagina web e più alta è la posizione di quella pagina nell'elenco fornito da Google in seguito alla ricerca effettuata.

Riprendiamo la definizione di *PageRank* data dai fondatori di Google, Brin e Page, nell'articolo [BP98]. Denotiamo con N il numero totale di pagine censite da Google, con P_i la generica pagina, con \mathcal{T}_i l'insieme degli indici di tutte le pagine del web che contengono un *link* alla pagina P_i e con c_i il numero di *link* dalla pagina P_i verso altre pagine. Il *PageRank* r_i della pagina P_i è così definito:

$$r_i = \frac{1 - \alpha}{N} + \alpha \sum_{j \in \mathcal{T}_i} \frac{r_j}{c_j}, \quad (6.5)$$

essendo $\alpha \in (0, 1)$ un fattore di smorzamento (un valore consigliato in [BP98] è 0.85). I *PageRank* di tutte le pagine censite devono soddisfare l'equazione

$$\sum_{j=1}^N r_j = 1. \quad (6.6)$$

Negli ultimi anni la definizione (6.5) è stata sostituita da altre più complesse che tengono conto anche dell'importanza dei *link* presenti sul web. Ad esempio se un *link* è ritenuto di tipo *spam*, allora gli viene attribuito un peso che influisce negativamente sui risultati di ricerca.

Come si evince dalla formula (6.5), il *PageRank* di una pagina dipende dal *PageRank* delle pagine collegate. I *PageRank* di tutte le pagine censite da Google vengono calcolati (e ordinati) periodicamente in seguito all'aggiornamento continuo delle pagine sul web, cosicché, quando si esegue una ricerca in Google, quest'ultimo non fa altro che estrarre, dalla propria lista ordinata, le pagine che contengono le parole chiave cercate.

Per capire come calcolare i *PageRank*, riscriviamo (6.5) in forma vettoriale. Definiamo: il vettore $\mathbf{r} = [r_1, \dots, r_N]^T$, la matrice $A \in \mathbb{R}^{N \times N}$ i cui elementi sono

$$A_{ij} = \begin{cases} \frac{1}{c_j} & \text{se } P_j \text{ ha un link verso } P_i \\ 0 & \text{altrimenti} \end{cases} \quad (6.7)$$

e la matrice $E \in \mathbb{R}^{N \times N}$ i cui elementi sono tutti uguali a 1. La formula (6.5) può essere riscritta come

$$r_i = \frac{1 - \alpha}{N} \sum_{j=1}^N r_j + \alpha A_{ij} r_j, \quad (6.8)$$

o più semplicemente come

$$\mathbf{r} = \mathbf{Gr}, \quad (6.9)$$

avendo posto

$$G = \frac{1-\alpha}{N}E + \alpha A. \quad (6.10)$$

La matrice G è nota come *matrice di Google del web*, essa è sparsa e di grandi dimensioni (nel momento in cui scriviamo questo testo, giugno 2017, la sua dimensione N è circa $4.61 \cdot 10^9$) e si può dimostrare (si veda ad esempio [LM06]) che il suo massimo autovalore è $\lambda = 1$. Di conseguenza, il vettore \mathbf{r} dei *PageRank* è l'autovettore associato all'autovalore di G di modulo massimo.

Si veda l'Esempio 6.3. ■

Sebbene molti dei metodi che andremo a presentare in questo Capitolo siano validi per matrici a coefficienti complessi, per semplicità limiteremo la trattazione al caso di matrici a coefficienti reali. Osserviamo comunque che le *function* di **MATLAB** per il calcolo di autovalori ed autovettori lavorano non solo con variabili reali, ma anche con variabili complesse senza bisogno di modificare le istruzioni di chiamata.

6.2 Il metodo delle potenze

Come abbiamo visto nei Problemi 6.3 e 6.2, 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 reale quadrata di dimensione n e assumiamo che i suoi autovalori siano così ordinati:

$$|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \dots \geq |\lambda_n|. \quad (6.11)$$

Si noti che stiamo in particolare supponendo 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 $\mathbf{y}^{(0)} = \mathbf{x}^{(0)} / \|\mathbf{x}^{(0)}\|$, si calcola

per $k = 1, 2, \dots$

$$\mathbf{x}^{(k)} = A\mathbf{y}^{(k-1)}, \quad \mathbf{y}^{(k)} = \frac{\mathbf{x}^{(k)}}{\|\mathbf{x}^{(k)}\|}, \quad \lambda^{(k)} = (\mathbf{y}^{(k)})^H A \mathbf{y}^{(k)}$$

(6.12)

Si noti che, procedendo in modo ricorsivo, si ha $\mathbf{y}^{(k)} = \beta^{(k)} A^k \mathbf{x}^{(0)}$, essendo $\beta^{(k)} = (\prod_{i=0}^k \|\mathbf{x}^{(i)}\|)^{-1}$ per $k \geq 1$. La presenza delle potenze di A giustifica il nome del metodo.

Come vedremo nella prossima Sezione 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 .

Si può inoltre dimostrare ([QSSG14] che, se $(\mathbf{x}^{(0)})^H \mathbf{x}_1 \neq 0$, le differenze $\|\mathbf{y}^{(k)} - (\mathbf{y}^{(k)})^H \mathbf{x}_1 \mathbf{x}_1\|$ e $|\lambda^{(k)} - \lambda_1|$ sono entrambe proporzionali al rapporto $|\lambda_2/\lambda_1|^k$ nel caso in cui A sia generica, ed a $|\lambda_2/\lambda_1|^{2k}$, quando la matrice A è hermitiana. In entrambi i casi, $\lambda^{(k)} \rightarrow \lambda_1$ per $k \rightarrow \infty$.

Un'implementazione del metodo delle potenze è fornita nel Programma 6.1. 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 A, il vettore iniziale x0, la tolleranza tol impiegata nel criterio d'arresto ed il numero massimo di iterazioni consentite kmax. I parametri in uscita sono l'autovalore di modulo massimo lambda, l'autovettore associato ed il numero di iterazioni che sono state effettuate.

Programma 6.1. eigpower: il metodo delle potenze

```
function [lambda,x,iter]=eigpower(A,tol,kmax,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,KMAX,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 KMAX (il
%   valore di default e' 100) e' stato raggiunto.
%   [LAMBDA,X,ITER] = EIGPOWER(A,TOL,KMAX,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);    kmax = 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<=kmax
    x = pro;           x = x/norm(x);
    pro = A*x;         lambdanew = x'*pro;
    err = abs(lambdanew - lambda);
    lambda = lambdanew; iter = iter + 1;
end
```

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 si ha $\lambda_1 = -30.643$, $\lambda_2 = 29.7359$, $\lambda_3 = -11.6806$ 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 $\text{tol}=1.e-12$, $\mathbf{x}0=\text{ones}(11,1)$, dopo 26 iterazioni il Programma 6.1 restituisce la seguente approssimazione dell'autovettore di modulo unitario associato all'autovalore di modulo massimo di A

```
x' =
Columns 1 through 8
0.5271  0.1590  0.2165  0.3580  0.4690  0.3861
0.1590  0.2837
Columns 9 through 11
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. ■

Esempio 6.3 (Calcolo dei PageRank) Consideriamo una rete (minuscola!) di $N = 8$ pagine connesse fra di loro come mostrato in Figura 6.3. Vogliamo ordinare le pagine in ordine decrescente di *PageRank*. La matrice A associata al grafo (si veda la definizione (6.7)) è

$$A = \begin{bmatrix} 0 & 1/2 & 1/2 & 1/2 & 0 & 0 & 0 & 0 \\ 1/2 & 0 & 0 & 0 & 0 & 0 & 0 & 1/2 \\ 1/2 & 1/2 & 0 & 1/2 & 1/3 & 1/3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1/2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1/3 & 0 & 0 \\ 0 & 0 & 1/2 & 0 & 0 & 0 & 0 & 1/2 \\ 0 & 0 & 0 & 0 & 1/3 & 1/3 & 0 & 0 \\ 0 & 1/2 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

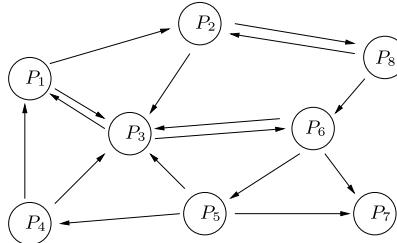


Figura 6.3. La rete dell’Esempio 6.3. Le frecce indicano i *link* fra le pagine

Dopo aver scelto $\alpha = 0.85$, definiamo la matrice G come in (6.10) e calcoliamo con il metodo delle potenze l’autovettore associato all’autovalore di modulo massimo. Con le seguenti istruzioni `MAT&OCT`

```

A=[ 0 1/2 1/2 1/2 0 0 0 0;
     1/2 0 0 0 0 0 0 1/2;
     1/2 1/2 0 1/2 1/3 1/3 0 0;
     0 0 0 0 1/3 0 0 0;
     0 0 0 0 0 1/3 0 0;
     0 0 1/2 0 0 0 0 1/2;
     0 0 0 0 1/3 1/3 0 0;
     0 1/2 0 0 0 0 0 0];
alpha=.85; N=8;
G=(1-alpha)/N*ones(N)+alpha*A;
tol=1.e-14; x0=rand(8,1);
[lambda,x,iter]=eigpower(G,tol,100,x0);
[r,inde]=sort(x,'descend')

```

otteniamo convergenza in 51 iterazioni all’autovalore `lambda=0.9887` ed all’auto vettore

```

x =
0.4959
0.3443
0.6064
0.0930
0.1595
0.3918
0.2052
0.1953

```

Infine, con il comando `[r,inde]=sort(x,'descend')`, possiamo ordinare in `sort` senso decrescente il vettore `x` (che contiene i *PageRank* delle pagine) e memorizzare nel vettore `inde` l’elenco delle pagine secondo valori decrescenti del *PageRank*. L’ordinamento cercato è:

3 1 6 2 7 8 5 4

■

6.2.1 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

$$\mathbf{x}^{(1)} = \mathbf{A}\mathbf{y}^{(0)} = \beta^{(0)} \mathbf{A} \sum_{i=1}^n \alpha_i \mathbf{x}_i = \beta^{(0)} \sum_{i=1}^n \alpha_i \lambda_i \mathbf{x}_i$$

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

$$\mathbf{y}^{(k)} = \beta^{(k)} \sum_{i=1}^n \alpha_i \lambda_i^k \mathbf{x}_i, \quad \beta^{(k)} = \frac{1}{\|\mathbf{x}^{(0)}\| \cdots \|\mathbf{x}^{(k)}\|}$$

e quindi

$$\mathbf{y}^{(k)} = \lambda_1^k \beta^{(k)} \left(\alpha_1 \mathbf{x}_1 + \sum_{i=2}^n \alpha_i \left(\frac{\lambda_i}{\lambda_1} \right)^k \mathbf{x}_i \right).$$

Poiché $|\lambda_i/\lambda_1| < 1$ per $i = 2, \dots, n$, il vettore $\mathbf{y}^{(k)}$ tende ad allinearsi lungo la stessa direzione dell'autovettore \mathbf{x}_1 quando k tende a $+\infty$ e $\alpha_1 \neq 0$.

Potrebbe sorgere il dubbio che i valori $|\lambda_1^k \beta^{(k)} \alpha_1|$ possano tendere a $+\infty$ se $|\lambda_1| > 1$ o a 0 se $|\lambda_1| < 1$. In realtà ciò non può succedere e, anzi, si dimostra che $|\lambda_1^k \beta^{(k)} \alpha_1| \rightarrow 1$ per $k \rightarrow \infty$. Infatti per induzione si ottiene

$$\beta^{(k)} = \frac{1}{\|\mathbf{A}^k \mathbf{x}^{(0)}\|} = \frac{1}{\|\lambda_1^k (\alpha_1 \mathbf{x}_1 + \mathbf{r}^{(k)})\|} \quad \text{per } k \geq 1,$$

dove $\mathbf{r}^{(k)} = \sum_{i=2}^n \alpha_i (\lambda_i/\lambda_1)^k \mathbf{x}_i$ e, poiché $\mathbf{r}^{(k)} \rightarrow \mathbf{0}$ per $k \rightarrow \infty$ e $\|\mathbf{x}_1\| = 1$, si ha

$$|\lambda_1^k \beta^{(k)} \alpha_1| = \frac{|\lambda_1^k \alpha_1|}{\|\lambda_1^k (\alpha_1 \mathbf{x}_1 + \mathbf{r}^{(k)})\|} \rightarrow 1.$$

Infine si ha $\mathbf{y}^{(k)H} \mathbf{x}_1 = \lambda_1^k \beta^{(k)} \alpha_1$, cosicché le quantità $\|\mathbf{y}^{(k)} - (\mathbf{y}^{(k)H} \mathbf{x}_1) \mathbf{x}_1\|$ convergono a zero come il termine dominante del vettore $\mathbf{r}^{(k)}$, ovvero come $|\lambda_2/\lambda_1|^k$ per $k \rightarrow \infty$.

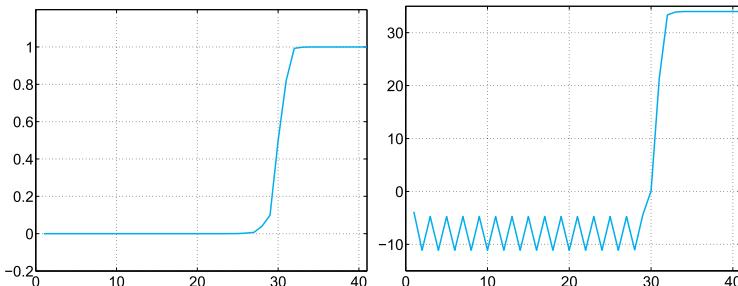


Figura 6.4. Le quantità $\cos(\theta^{(k)}) = (\mathbf{y}^{(k)})^H \mathbf{x}_1$ (a sinistra) e i valori $\lambda^{(k)}$ (a destra) dell’Esempio 6.4, per $k = 1, \dots, 41$

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 6.4 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 . In Figura 6.4 viene riportata la quantità $\cos(\theta^{(k)}) = (\mathbf{y}^{(k)})^H \mathbf{x}_1$ al variare di k . Come si nota dopo circa 30 iterazioni questo valore tende a 1 e l’angolo $\theta^{(k)}$ tra i due vettori tende rapidamente a 0, 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 . ■

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.



6.3 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 $\mathbf{y}^{(0)} = \mathbf{x}^{(0)}/\|\mathbf{x}^{(0)}\|$, si calcola

per $k = 1, 2, \dots$

$$\mathbf{x}^{(k)} = \mathbf{A}^{-1}\mathbf{y}^{(k-1)}, \quad \mathbf{y}^{(k)} = \frac{\mathbf{x}^{(k)}}{\|\mathbf{x}^{(k)}\|}, \quad \mu^{(k)} = (\mathbf{y}^{(k)})^H \mathbf{A}^{-1} \mathbf{y}^{(k)}$$
(6.13)

Se \mathbf{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 $\mathbf{A}\mathbf{x}^{(k)} = \mathbf{y}^{(k-1)}$. È pertanto conveniente generare la fattorizzazione LU di \mathbf{A} (o la sua fattorizzazione di Cholesky se \mathbf{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 `1u` di **MATGOCT** è in grado di calcolare la fattorizzazione LU anche di una matrice a coefficienti complessi.

Il metodo delle potenze può essere utilizzato anche per calcolare l'autovalore di \mathbf{A} più vicino ad un dato numero $\mu \in \mathbb{C}$. Denotiamo con λ_μ tale autovalore, ponendo $\mathbf{A}_\mu = \mathbf{A} - \mu\mathbf{I}$, osserviamo che $\lambda(\mathbf{A}_\mu) = \lambda(\mathbf{A}) - \mu$. Pertanto, per calcolare l'autovalore λ_μ di \mathbf{A} più prossimo a μ , basta approssimare l'autovalore $\lambda_{\min}(\mathbf{A}_\mu)$ di modulo minimo di \mathbf{A}_μ e quindi calcolare $\lambda_\mu = \lambda_{\min}(\mathbf{A}_\mu) + \mu$. Questo vuol dire applicare il metodo delle potenze inverse alla matrice \mathbf{A}_μ per ottenere un'approssimazione di $\lambda_{\min}(\mathbf{A}_\mu)$. Questo algoritmo è noto come il *metodo delle potenze inverse con shift* ed il numero μ è chiamato lo *shift*.

Nel Programma 6.2 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`, `kmax`, `x0` sono gli stessi del Programma 6.1, mentre `mu` denota il valore dello *shift*. I parametri in uscita sono l'approssimazione dell'autovalore λ_μ di \mathbf{A} , l'autovettore associato `x` ed il numero di iterazioni effettuate per calcolarlo.

Programma 6.2. `invshift`: il metodo delle potenze inverse con shift

```
function [lambda,x,iter]=invshift(A,mu,tol,kmax,x0)
% INVSHIFT    Approssima l'autovalore di modulo minimo.
%   LAMBDA = INVSHIFT(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 = INVSHIFT(A,MU) calcola l'autovalore di A
% piu' vicino ad un dato numero (reale o complesso)
% MU.
% LAMBDA = INVSHIFT(A,MU,TOL,KMAX,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 KMAX (il valore di default e' 100) e'
% stato raggiunto.
% [LAMBDA,X,ITER] = INVSHIFT(A,MU,TOL,KMAX,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); kmax = 100; tol = 1.e-06; mu = 0;
elseif nargin == 2
    x0 = rand(n,1); kmax = 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<=kmax
    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;
```

Esempio 6.5 Applichiamo il metodo delle potenze inverse per il calcolo dell'autovalore di modulo minimo della matrice $A(30)$ definita nell'Esempio 6.1. Richiamando il Programma 6.2 con il comando

```
[lambda,x,iter]=invshift(A(30))
```

otteniamo che il metodo converge in 5 iterazioni al valore 0.2854 che è l'approssimazione dell'autovalore di modulo minimo di $A(30)$. ■

Esempio 6.6 Fra tutti gli autovalori della matrice $A(30)$ dell'Esempio 6.1 cerchiamo quello più vicino al numero reale 17. Utilizziamo il Programma 6.2 con $\mu=17$, una tolleranza $\text{tol}=10^{-10}$ e $x0=[1;1;1;1]$. Dopo 8 iterazioni, il programma ritorna il valore $\lambda=17.82079703055703$. Una conoscenza meno accurata dello shift avrebbe richiesto più iterazioni. Ad esempio, ponendo $\mu = 13$, il programma avrebbe fornito il valore 17.82079703064106 dopo 19 iterazioni. ■

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.

6.4 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

$$C_i^{(r)} = \left\{ z \in \mathbb{C} : |z - a_{ii}| \leq \sum_{j=1, j \neq i}^n |a_{ij}| \right\},$$

$$C_i^{(c)} = \left\{ z \in \mathbb{C} : |z - a_{ii}| \leq \sum_{j=1, j \neq i}^n |a_{ji}| \right\}.$$

$C_i^{(r)}$ è chiamato l' i -esimo *cerchio riga*, mentre $C_i^{(c)}$ l' i -esimo *cerchio colonna*.

Con il Programma 6.3 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 **hold off** viene neutralizzato dal comando **hold off**.

title I comandi **title**, **xlabel** e **ylabel** hanno lo scopo di visualizzare nella figura un titolo e le etichette degli assi.

ylabel Il comando **patch** serve per colorare i cerchi, mentre il comando **axis image** serve a scalare opportunamente l'immagine.

axis

Programma 6.3. gershcircles: i cerchi di Gershgorin

```
function gershcircles(A)
%GERSHCIRCLES disegna i cerchi di Gershgorin
%   GERSHCIRCLES(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','FontSize',20);
set(gca,'FontSize',20)
xlabel('Re'); ylabel('Im');
figure(2); title('Cerchi colonna','FontSize',20);
set(gca,'FontSize',20)
xlabel('Re'); ylabel('Im');
color=[0.8,1,1];
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,:)),...
    color(1,:));
    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,:)),...
    color(1,:));
    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');
end

```

Esempio 6.7 In Figura 6.5 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. ■

I cerchi di Gershgorin possono essere usati per localizzare gli autovекторi di una matrice. Vale infatti il seguente risultato:

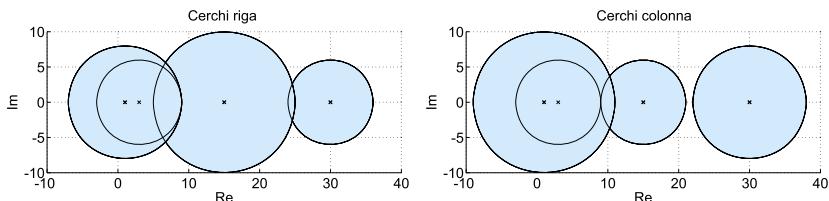


Figura 6.5. Cerchi riga (a sinistra) e colonna (a destra) per la matrice dell’Esempio 6.7

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*. Da questa proposizione segue che gli autovalori di una matrice a dominanza diagonale stretta per righe sono tutti diversi da zero.

Esempio 6.8 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 6.2 per calcolare l’autovalore di modulo massimo scegliendo come *shift* $\mu = 48$, troviamo che il metodo converge in 15 iterazioni, contro le 22 richieste dal metodo delle potenze a parità di dato iniziale $x0=[1;1;1;1]$ e di tolleranza $tol=1.e-10$. ■



Si vedano gli Esercizi 6.7–6.8.

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.

6.5 Calcolo di tutti gli autovalori

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 eig* di **MATLAB**. Più precisamente, se richiamata nella forma $D = \text{eig}(A)$, 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, $[X, D] = \text{eig}(A)$, 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 $A * X = X * D$.

Il metodo delle iterazioni QR è così chiamato perché impiega la fattorizzazione QR, introdotta nella Sezione 5.7. In questa Sezione presentiamo il metodo QR solo nel caso di matrici reali e nella sua forma più elementare che, in genere, non garantisce la convergenza agli autovalori di A . Per una descrizione completa dell'algoritmo rimandiamo a [QSSG14, Cap. 5], mentre per la sua estensione al caso complesso rimandiamo a [GL96, Sez. 5.2.10] e a [Dem97, Sez. 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

$$\lim_{k \rightarrow +\infty} A^{(k)} = T = \begin{bmatrix} \lambda_1 & t_{12} & \dots & t_{1n} \\ 0 & \ddots & \ddots & \vdots \\ \vdots & & \lambda_{n-1} & t_{n-1,n} \\ 0 & \dots & 0 & \lambda_n \end{bmatrix}. \quad (6.14)$$

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 6.4 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, kmax .

Programma 6.4. qrbasic: il metodo delle iterazioni QR

```
function D=qrbasic(A,tol,kmax)
%QRBASIC calcola gli autovalori di una matrice
%D=QRBASIC(A,TOL,KMAX) 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 KMAX. 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 = max(max(abs(tril(A,-1))));
while niter <= kmax & test > tol
    [Q,R]=qr(T); T = R*Q; niter = niter + 1;
    test = max(max(abs(tril(T,-1))));
end
if niter > kmax
    warning(['Il metodo non converge nel massimo',...
        'numero di iterazioni permesso\n']);
else
    fprintf('Il metodo converge in %d iterazioni\n',niter)
end
D = diag(T);
```

Esempio 6.9 Consideriamo la matrice $A(30)$ dell'Esempio 6.1 ed usiamo il Programma 6.4 per calcolarne gli autovalori. Otteniamo

```
D=qrbasic(A(30),1.e-14,100)
```

```
Il metodo converge in 56 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



Figura 6.6. L'immagine originale (*a sinistra*), quella ricostruita sulla base dei primi 20 valori singolari (*al centro*) e quella ottenuta usando i primi 60 (*a destra*)

simile troviamo che il metodo per soddisfare lo stesso criterio d'arresto richiede ora ben 1149 iterazioni

```
D=qrbasic(A(-30),1.e-14,2000)
```

```
Il metodo converge in 1149 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 **MATGOC**T, con il comando **eigs(A,k)** si possono calcolare i primi *k* **eigs** autovalori di **A** di modulo maggiore.

Esempio 6.10 (Compressione di immagini) Con il comando

```
A=imread('lena.jpg');
```

di **MATGOC**T carichiamo un'immagine in bianco e nero, in formato JPEG. L'immagine che abbiamo scelto è largamente utilizzata dalla comunità scientifica per testare programmi per la compressione di immagini ed è facilmente reperibile sul web. La variabile **A** è una matrice di 512×512 interi a 8 bit (**uint8**) che rappresentano delle intensità di grigio. Eseguendo i comandi

```
image(A); colormap(gray(256));
```

image

ci apparirà l'immagine a sinistra della Figura 6.6. 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) e quindi calcolare la SVD con i comandi:

```
A=double(A); [U,S,V]=svd(A);
```

Abbiamo riportato al centro l'immagine ricostruita usando solo i primi 20 valori singolari di **S**, ottenuta con i comandi

```
k=20; X=U(:,1:k)*S(1:k,1:k)*(V(:,1:k))';
image(uint8(X)); colormap(gray(256));
```

a destra della figura appare l'immagine ricostruita con i primi 60 valori singolari. Si osserva che l'immagine di destra è un'ottima approssimazione dell'immagine originale, e richiede la memorizzazione di 61500 coefficienti (due matrici di 512×60 più i primi 60 valori singolari) invece dei 262144 coefficienti richiesti dall'immagine originale. ■



Si vedano gli Esercizi 6.9–6.10.

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;
3. la sua velocità di convergenza dipende asintoticamente dal maggiore modulo del quoziente di due autovalori consecutivi.

6.6 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 [Wil88], [GL96] e [QSSG14, Cap. 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 5.10) 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 generale si veda [Saa92].

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.

6.7 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 $\mathbf{x}^{(0)} = (1, 2, 3)^T$

$$A_1 = \begin{bmatrix} 1 & 2 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \quad A_2 = \begin{bmatrix} 0.1 & 3.8 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \quad A_3 = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}.$$

Si commenti la convergenza del metodo nei tre casi.

Esercizio 6.2 (Dinamica di una popolazione) Le caratteristiche di una popolazione di pesci sono descritte nella seguente matrice di Leslie, introdotta nel Problema 6.2:

i	Intervallo d'età (mesi)	$x_i^{(0)}$	m_i	s_i
0	0–3	6	0	0.2
1	3–6	12	0.5	0.4
2	6–9	8	0.8	0.8
3	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 6.2.

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 pari a $\lambda_2 = \gamma e^{-i\vartheta}$, dove i è l'unità immaginaria, $\gamma \in \mathbb{R} \setminus \{0\}$ e $\vartheta \in \mathbb{R} \setminus \{k\pi, k \in \mathbb{Z}\}$.

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 si approssimino il massimo autovalore positivo e l'autovalore negativo di modulo massimo della *matrice di Wilkinson* di dimensione 7

$$A = \begin{bmatrix} 3 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 3 \end{bmatrix}.$$

`wilkinson` A può essere generata con il comando `wilkinson(7)`.

Esercizio 6.7 Utilizzando i cerchi di Gershgorin, si stimi il massimo numero di autovalori complessi delle matrici seguenti

$$A = \begin{bmatrix} 2 & -1/2 & 0 & -1/2 \\ 0 & 4 & 0 & 2 \\ -1/2 & 0 & 6 & 1/2 \\ 0 & 0 & 1 & 9 \end{bmatrix}, \quad B = \begin{bmatrix} -5 & 0 & 1/2 & 1/2 \\ 1/2 & 2 & 1/2 & 0 \\ 0 & 1 & 0 & 1/2 \\ 0 & 1/4 & 1/2 & 3 \end{bmatrix}.$$

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.

Ottimizzazione numerica

Sia $f : \mathbb{R}^n \rightarrow \mathbb{R}$, $n \geq 1$, una funzione che chiameremo *funzione obiettivo* o *funzione costo*. Il problema di determinare

$$\min_{\mathbf{x} \in \mathbb{R}^n} f(\mathbf{x}) \quad (7.1)$$

è detto di ottimizzazione *non vincolata*, mentre quello di determinare

$$\min_{\mathbf{x} \in \Omega} f(\mathbf{x}) \quad (7.2)$$

su un insieme $\Omega \subset \mathbb{R}^n$ è detto di ottimizzazione *vincolata*. L'insieme Ω è definito da vincoli descritti da uguaglianze o disuguaglianze e dettati dalla natura del problema da risolvere. Se ad esempio si devono allocare in maniera ottimale n risorse limitate x_i (per $i = 1, \dots, n$), i vincoli saranno espressi da disuguaglianze del tipo $0 \leq x_i \leq C_i$ (con C_i costanti assegnate) e l'insieme Ω sarà il sottoinsieme di \mathbb{R}^n determinato dal soddisfacimento di tutti questi vincoli, ovvero

$$\Omega = \{\mathbf{x} = (x_1, \dots, x_n) : 0 \leq x_i \leq C_i, i = 1, \dots, n\}.$$

Poiché calcolare il massimo di una funzione f equivale a calcolare il minimo della funzione $g = -f$, per semplicità considereremo solo algoritmi per problemi di minimizzazione.

Osserviamo infine che spesso, più che il valore minimo di una certa funzione, può interessare conoscere il valore della variabile indipendente che realizza questo estremo, ovvero il cosiddetto *punto di minimo*, che potrebbe tuttavia non essere unico.

In questo capitolo presenteremo alcuni fra i più noti metodi numerici per risolvere problemi di ottimizzazione non vincolata. Faremo solo qualche cenno alle strategie da utilizzare nel caso dell'ottimizzazione vincolata.

7.1 Alcuni problemi

Problema 7.1 (Dinamica delle popolazioni) Una colonia di 250 esemplari di batteri viene posta in un ambiente isolato e si riproduce seguendo il modello di Verhulst

$$b(t) = \frac{2500}{1 + 9e^{-t/3}}, \quad t > 0,$$

dove t rappresenta il tempo (espresso in giorni) trascorso a partire dal tempo $t = 0$ di inizio della coltura. Si vuole determinare dopo quanti giorni il tasso di crescita della popolazione di batteri raggiunge il valore massimo. Si tratta di un problema di ottimizzazione non vincolata, per la cui risoluzione rinviamo agli Esempi 7.1 e 7.2. ■

Problema 7.2 (Elaborazione di segnali) Alcune applicazioni per il riconoscimento vocale automatico, come quelle che possiamo trovare sul nostro *smartphone*, trasformano il segnale acustico in un insieme di parametri che lo caratterizzano in maniera univoca. Questa trasformazione è resa necessaria per l'impossibilità a memorizzare una grande quantità di dati nel dispositivo elettronico.

L'intensità di segnale è modellata dalla somma di m funzioni gaussiane (anche chiamate *picchi*)

$$g_k(t; a_k, \sigma_k) = \frac{1}{\sqrt{2\pi\sigma_k^2}} e^{-(t-a_k)^2/(2\sigma_k^2)}, \quad \text{per } k = 1, \dots, m, \quad t \in [t_0, t_f], \quad (7.3)$$

caratterizzate ognuna da 2 parametri: il *valore atteso* a_k del k -simo picco, che è il centro del picco stesso, e la sua *varianza* σ_k^2 (si veda Figura 7.1). Una volta nota la varianza di ogni picco, si possono valutare l'altezza $h_k = 1/\sqrt{2\pi\sigma_k^2}$ e l'ampiezza $w_k = 2\sqrt{\log 4\sigma_k^2}$ del picco stesso.

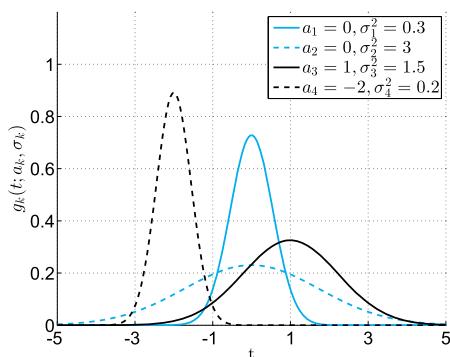


Figura 7.1. Alcuni esempi di funzioni gaussiane; in legenda sono specificati i valori attesi e le varianze ad esse associate

Data una campionatura del segnale, ovvero n coppie (t_i, y_i) per $i = 1, \dots, n$, vogliamo costruire la funzione

$$g(t; \mathbf{a}, \boldsymbol{\sigma}) = \sum_{k=1}^m g_k(t; a_k, \sigma_k), \quad (7.4)$$

essendo $\mathbf{a} = [a_1, \dots, a_m]$ e $\boldsymbol{\sigma} = [\sigma_1, \dots, \sigma_m]$ da determinare in modo da soddisfare il seguente problema di minimizzazione

$$\min_{\mathbf{a}, \boldsymbol{\sigma}} \sum_{i=1}^n (g(t_i; \mathbf{a}, \boldsymbol{\sigma}) - y_i)^2. \quad (7.5)$$

(7.5) è un problema ai minimi quadrati non lineari e verrà risolto nell'Esempio 7.12.

Il modello (7.4) qui descritto è noto anche come *Gaussian Mixture Model (GMM)* ed è ampiamente utilizzato in statistica per le operazioni di *data mining* e *pattern recognition*. ■

Problema 7.3 (Ottimizzazione di una mesh) Consideriamo una triangolazione di un dominio $D \subset \mathbb{R}^2$, come in Figura 7.2, a sinistra. Si vuole modificare la posizione dei vertici dei triangoli interni a D al fine di ottimizzare la qualità dei triangoli stessi, nel senso che ora andiamo a specificare.

Siano $(x_i^{(k)}, y_i^{(k)})$ (per $i = 0, 1, 2$) le coordinate dei vertici del triangolo k -simo T_k in D , si definiscono le matrici

$$\mathbf{A}_k = \begin{pmatrix} x_1^{(k)} - x_0^{(k)} & x_2^{(k)} - x_0^{(k)} \\ y_1^{(k)} - y_0^{(k)} & y_2^{(k)} - y_0^{(k)} \end{pmatrix}$$

e le quantità scalari

$$\mu_k = \frac{4 \det(\mathbf{A}_k)}{\sqrt{3} \|\mathbf{A}_k \mathbf{W}^{-1}\|_F^2}, \quad (7.6)$$

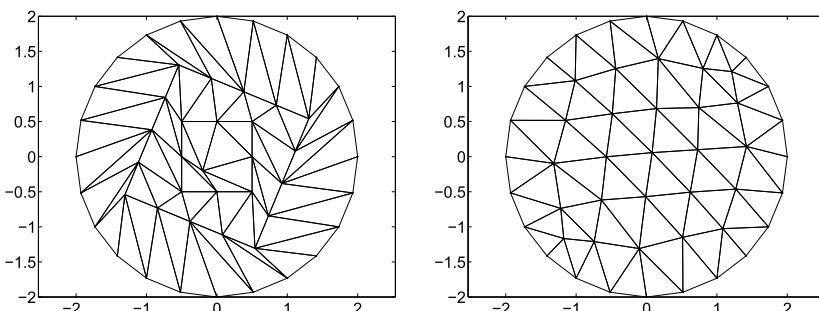


Figura 7.2. Le mesh del Problema 7.3. A sinistra quella originale, a destra quella ottimizzata

essendo $\mathbf{W} = [1, 1/2; 0, \sqrt{3}/2]$, mentre $\|\mathbf{B}\|_F = \sqrt{\sum_{i,j=1}^2 b_{ij}^2}$ è la cosiddetta norma di Frobenius di una matrice \mathbf{B} . Se il triangolo T_k è equilatero, risulta $\mu_k = 1$, mentre μ_k è tanto più prossimo a zero quanto più il triangolo T_k è deformato rispetto ad un triangolo equilatero. Per ottimizzare la mesh minimizziamo la funzione $\sum_{k=1}^{N_e} \mu_k^{-1}$ rispetto alla posizione dei vertici interni a D , con i vincoli che $\det(\mathbf{A}_k) \geq \tau$, per un $\tau > 0$ assegnato, e che non ci sia inversione nell'ordinamento dei nodi della mesh [Mun07].

Questo è un problema di minimizzazione vincolata con vincoli di disuguaglianza. La sua soluzione verrà presentata nell'Esempio 7.16.

Mostriamo in Figura 7.2 a sinistra la triangolazione originaria e a destra quella ottimizzata. Problemi di questo tipo si incontrano nella risoluzione numerica di problemi differenziali alle derivate parziali con il metodo degli Elementi Finiti (si veda ad esempio il Capitolo 9). ■

Problema 7.4 (Finanza) Un dato capitale è investito in tre fondi di investimento che hanno un rendimento atteso del 1.2%, 1.6%, 2.8%, rispettivamente. Il rischio associato a questo investimento è modellato da una funzione che dipende dalle frazioni x_i di capitale investito nei tre fondi (per $i = 1, 2, 3$), dai tassi di rischio dei singoli fondi e dalla correlazione tra di essi. Nel nostro esempio esso assume l'espressione

$$r(\mathbf{x}) = 0.04x_1^2 + 0.25x_2^2 + 0.64x_3^2 + 0.1x_1x_2 + 0.208x_2x_3. \quad (7.7)$$

L'obiettivo consiste nel minimizzare il rischio, garantendo un rendimento pari al 2%.

Questo è un problema di ottimizzazione vincolata con due vincoli di uguaglianza, per la sua risoluzione rimandiamo all'Esempio 7.14. ■

Problema 7.5 (Rete stradale) Consideriamo una rete di n strade e p incroci come ad esempio quella rappresentata in Figura 7.3. Ogni minuto entrano ed escono dalla rete M veicoli, su ogni tratto stradale s_j c'è un limite massimo di velocità pari a $v_{j,m}$ km/min e possono essere presenti al più $\rho_{j,m}$ veicoli per km.

Si vuole individuare quali siano le densità ρ_j (in veicoli/km) sui vari tratti stradali s_j (con $0 \leq \rho_j \leq \rho_{j,m}$) al fine di minimizzare il tempo di transito medio tra il punto di ingresso (nodo 1 in Figura 7.3) ed il punto di uscita (nodo 7 in Figura 7.3). Assumiamo che la velocità v_j di percorrenza del tratto j -simo dipenda dalla densità massima dei veicoli $\rho_{j,m}$ e dalla velocità massima $v_{j,m}$ secondo il modello $v_j = v_{j,m}(1 - \rho_j/\rho_{j,m})$ km/min. Di conseguenza il flusso di veicoli per unità di tempo lungo il tratto j -simo è $q_j = \rho_j v_j = \rho_j v_{j,m}(1 - \rho_j/\rho_{j,m})$ veicoli/min. Denotando con t_j (in min) il tempo necessario per percorrere il tratto j -simo di lunghezza L_j km, si ha $t_j = L_j/v_j = L_j/(v_{j,m}(1 - \rho_j/\rho_{j,m}))$ min.

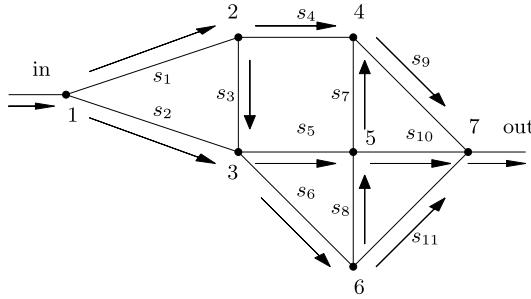


Figura 7.3. La rete di strade del Problema 7.5

La funzione obiettivo da minimizzare è

$$f(\rho) = \frac{\sum_{j=1}^n t_j \rho_j}{\sum_{j=1}^n \rho_j}. \quad (7.8)$$

In ogni nodo della rete imponiamo un'equazione di bilancio dei flussi. Ciò significa che, indicando con segno positivo i flussi in ingresso nel nodo i -simo ($q_{i,jin}$) e con segno negativo quelli in uscita dallo stesso nodo ($q_{i,jout}$), devono essere soddisfatti i vincoli

$$\sum_{jin} q_{i,jin} - \sum_{jout} q_{i,jout} = 0 \quad \text{per } i = 1, \dots, p. \quad (7.9)$$

Questo è un problema di minimizzazione vincolata con vincoli sia di uguaglianza che di disegualanza. Per la sua risoluzione si veda l'Esempio 7.17. ■

7.2 Ottimizzazione non vincolata

Nel minimizzare una funzione obiettivo potremmo essere interessati a trovare un punto di minimo globale o un punto di minimo locale. Diciamo che $\mathbf{x}^* \in \mathbb{R}^n$, è un punto di *minimo globale* per f se

$$f(\mathbf{x}^*) \leq f(\mathbf{x}) \quad \forall \mathbf{x} \in \mathbb{R}^n,$$

mentre \mathbf{x}^* è un punto di *minimo locale* per f se esiste una palla $B_r(\mathbf{x}^*) \subset \mathbb{R}^n$ di centro \mathbf{x}^* e raggio $r > 0$ tale che

$$f(\mathbf{x}^*) \leq f(\mathbf{x}) \quad \forall \mathbf{x} \in B_r(\mathbf{x}^*).$$

Se f è una funzione differenziabile in \mathbb{R}^n , denotiamo con

$$\nabla f(\mathbf{x}) = \left(\frac{\partial f}{\partial x_1}(\mathbf{x}), \dots, \frac{\partial f}{\partial x_n}(\mathbf{x}) \right)^T \quad (7.10)$$

il *gradiente* di f nel punto $\mathbf{x} \in \mathbb{R}^n$ e, se esistono tutte le derivate parziali seconde di f in \mathbf{x} , con $H(\mathbf{x})$ la *matrice Hessiana* di f valutata nel punto \mathbf{x} , i cui elementi sono

$$h_{ij}(\mathbf{x}) = \frac{\partial^2 f(\mathbf{x})}{\partial x_j \partial x_i}, \quad i, j = 1, \dots, n.$$

Inoltre, se $f \in C^2(\mathbb{R}^n)$, cioè esistono e sono continue tutte le sue derivate parziali prime e seconde, allora la matrice Hessiana $H(\mathbf{x})$ è simmetrica per ogni $\mathbf{x} \in \mathbb{R}^n$.

Un punto \mathbf{x}^* è detto *punto stazionario* (o *critico*) per f se $\nabla f(\mathbf{x}^*) = \mathbf{0}$, *punto regolare* se $\nabla f(\mathbf{x}^*) \neq \mathbf{0}$.

Una funzione $f : \mathbb{R}^n \rightarrow \mathbb{R}$ è *convessa* se per ogni $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ e per ogni $\alpha \in [0, 1]$,

$$f(\alpha \mathbf{x} + (1 - \alpha) \mathbf{y}) \leq \alpha f(\mathbf{x}) + (1 - \alpha) f(\mathbf{y}), \quad (7.11)$$

ed è *lipschitziana* se esiste una costante $L > 0$ tale che

$$|f(\mathbf{x}) - f(\mathbf{y})| \leq L \|\mathbf{x} - \mathbf{y}\| \quad \forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^n. \quad (7.12)$$

Proposizione 7.1 (Condizioni di ottimalità) *Sia $\mathbf{x}^* \in \mathbb{R}^n$. Se \mathbf{x}^* è un punto di minimo (locale o globale) per f e se esiste $r > 0$ tale che $f \in C^1(B_r(\mathbf{x}^*))$, allora $\nabla f(\mathbf{x}^*) = \mathbf{0}$. Se inoltre $f \in C^2(B_r(\mathbf{x}^*))$, allora $H(\mathbf{x}^*)$ è una matrice semidefinita positiva.*

Viceversa, sia $r > 0$ tale che $f \in C^2(B_r(\mathbf{x}^))$. Se $\nabla f(\mathbf{x}^*) = \mathbf{0}$ e $H(\mathbf{x}^*)$ è definita positiva in $B_r(\mathbf{x}^*)$, allora $\mathbf{x}^* \in B_r(\mathbf{x}^*)$ è un punto minimo locale per f .*

Infine, se f è una funzione differenziabile e convessa in \mathbb{R}^n e $\nabla f(\mathbf{x}^) = \mathbf{0}$, allora \mathbf{x}^* è un punto di minimo globale per f .*

Non è detto che una qualsiasi funzione reale definita in \mathbb{R}^n ammetta punti di minimo e, in caso affermativo, che ne esista uno solo (un minimo globale quindi). Ad esempio, la funzione $f(\mathbf{x}) = x_1 + 3x_2$ è illimitata, mentre $f(\mathbf{x}) = \sin(x_1) \sin(x_2) \cdots \sin(x_n)$ ammette infiniti punti di minimo e di massimo (sia locali che globali).

Quando si risolve numericamente un problema di minimizzazione, la situazione ideale è che la funzione obiettivo abbia un unico punto di minimo globale. Di fatto però succede spesso che la funzione in esame presenti diversi punti di minimo locale e la determinazione del punto di minimo assoluto avviene dopo un confronto dei minimi locali della funzione stessa. È per tale motivo che in questo capitolo descriveremo solamente metodi per l'approssimazione di punti di minimo locale.

I metodi per l'ottimizzazione numerica sono principalmente di tipo iterativo e si dividono in due categorie, a seconda che richiedano o meno la conoscenza delle derivate esatte della funzione obiettivo.

Gli ultimi sono i cosiddetti metodi *derivative free*, essi operano un confronto diretto di alcuni valori della funzione costo al fine di investigare il suo comportamento locale. Fra questi metodi, alcuni utilizzano l'approssimazione numerica delle derivate (tipicamente attraverso schemi alle differenze finite, come quelli introdotti nelle Sezioni 4.2 e 9.3), si veda Sezione 7.3.

I metodi che sfruttano le derivate invece possono beneficiare di informazioni locali molto accurate sulla funzione, ottenendo così una convergenza al punto di minimo in genere più veloce rispetto ai metodi *derivative free*. È dimostrabile infatti che, se f è differenziabile in un punto $\bar{x} \in \text{dom } f$ e $\nabla f(\bar{x})$ è non nullo, allora la massima crescita di f , a partire da \bar{x} , avviene nella direzione e verso del vettore gradiente $\nabla f(\bar{x})$ (si veda la Sezione 5.10).

Tra i metodi di minimizzazione che richiedono la conoscenza esatta delle derivate si distinguono fondamentalmente due importanti classi (basate su due strategie tra loro complementari): i *metodi di discesa* (o di tipo *line search*) ed i metodi di tipo *trust region*, che descriveremo rispettivamente nelle Sezioni 7.5 e 7.6.

7.3 Metodi *derivative free*

In questa sezione vedremo due semplici metodi per la ricerca di punti di minimo di funzioni in una variabile reale, ma utili anche nel caso multidimensionale quando si restrinja la ricerca del punto di minimo ad una particolare direzione di \mathbb{R}^n . Quindi descriveremo il metodo di Nelder e Mead per la minimizzazione di funzioni in più variabili.

7.3.1 I metodi della sezione aurea e dell'interpolazione quadratica

Sia $f : (a, b) \rightarrow \mathbb{R}$ una funzione continua che abbia un unico punto di minimo $x^* \in (a, b)$. Poniamo $I_0 = (a, b)$ e per $k \geq 0$ generiamo una successione di intervalli $I_k = (a^{(k)}, b^{(k)})$ di ampiezza decrescente e ognuno contenente il punto di minimo x^* . Quando l'ampiezza dell'intervallo I_k sarà sufficientemente piccola, accetteremo il punto medio di I_k (indicato con $x^{(k)}$) come approssimazione di x^* .

Supponiamo di aver già costruito l'intervallo I_k , al fine di individuare quello successivo I_{k+1} calcoliamo due punti $c^{(k)}, d^{(k)} \in I_k$ così definiti:

$$c^{(k)} = a^{(k)} + \frac{1}{\varphi^2}(b^{(k)} - a^{(k)}) \quad \text{e} \quad d^{(k)} = a^{(k)} + \frac{1}{\varphi}(b^{(k)} - a^{(k)})$$
(7.13)

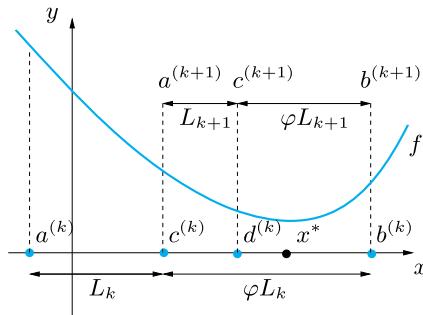


Figura 7.4. Una generica iterazione del metodo della sezione aurea per la ricerca del punto di minimo di una funzione f ; φ è la sezione aurea e $L_k = c^{(k)} - a^{(k)}$

essendo $\varphi = \frac{1+\sqrt{5}}{2} \simeq 1.628$ la *sezione aurea*. I punti $c^{(k)}$ e $d^{(k)}$ sono tali che $c^{(k)} < d^{(k)}$; inoltre essi soddisfano le uguaglianze

$$\frac{b^{(k)} - a^{(k)}}{d^{(k)} - a^{(k)}} = \frac{d^{(k)} - a^{(k)}}{b^{(k)} - d^{(k)}} = \varphi \quad (7.14)$$

e

$$\frac{b^{(k)} - a^{(k)}}{b^{(k)} - c^{(k)}} = \frac{b^{(k)} - c^{(k)}}{c^{(k)} - a^{(k)}} = \varphi. \quad (7.15)$$

Infine, essi sono simmetrici rispetto al punto medio dell'intervallo I_k , ovvero

$$\frac{a^{(k)} + b^{(k)}}{2} - c^{(k)} = d^{(k)} - \frac{a^{(k)} + b^{(k)}}{2}. \quad (7.16)$$

Posto $a^{(0)} = a$ e $b^{(0)} = b$, l'algoritmo della sezione aurea può essere così formulato (si veda la Figura 7.4): per $k = 0, 1, \dots$ fino a convergenza

```

calcolare  $c^{(k)}$  e  $d^{(k)}$  mediante (7.13)
if  $f(c^{(k)}) \geq f(d^{(k)})$ 
    porre  $I_{k+1} = (a^{(k+1)}, b^{(k+1)}) = (c^{(k)}, b^{(k)})$ 
else
    porre  $I_{k+1} = (a^{(k+1)}, b^{(k+1)}) = (a^{(k)}, d^{(k)})$ 
endif

```

(7.17)

Dalle (7.14), (7.15) discende anche che $c^{(k+1)} = d^{(k)}$ se $I_{k+1} = (c^{(k)}, b^{(k)})$, mentre $d^{(k+1)} = c^{(k)}$ se $I_{k+1} = (a^{(k)}, d^{(k)})$ (sfrutteremo questa proprietà nel Programma 7.1).

Un possibile test d'arresto per le iterazioni descritte è

$$\frac{b^{(k+1)} - a^{(k+1)}}{|c^{(k+1)}| + |d^{(k+1)}|} < \varepsilon \quad (7.18)$$

dove ε è una tolleranza assegnata. Nel caso il test sia soddisfatto, il punto medio dell'ultimo intervallo I_{k+1} (indicato con $x^{(k+1)}$) verrà accettato come approssimazione del punto di minimo x^* cercato e si avrà

$$|x^{(k+1)} - x^*| \leq \frac{b^{(k+1)} - a^{(k+1)}}{2}.$$

Poiché dalla (7.14) discende che

$$|b^{(k+1)} - a^{(k+1)}| = \frac{1}{\varphi} |b^{(k)} - a^{(k)}| = \dots = \frac{1}{\varphi^{k+1}} |b^{(0)} - a^{(0)}|, \quad (7.19)$$

otteniamo

$$|x^{(k+1)} - x^*| \leq \frac{b^{(0)} - a^{(0)}}{2\varphi^{k+1}},$$

ovvero il metodo della sezione aurea converge linearmente e con un fattore di decadimento dell'errore pari a $\varphi^{-1} \simeq 0.618$.

Il metodo della sezione aurea è implementato nel Programma 7.1: `fun` è una *anonymous function* che specifica la funzione f , `a` e `b` sono gli estremi dell'intervallo di ricerca, `tol` la tolleranza ε e `kmax` il massimo numero consentito di iterazioni.

In uscita, `xmin` contiene il punto di minimo calcolato, `fmin` il valore minimo di f in (a, b) , ovvero il valore assunto da f in `xmin`, `iter` il numero di iterazioni effettuate.

Programma 7.1. golden: il metodo della sezione aurea

```
function [xmin,fmin,iter]=golden(fun,a,b,tol,...  
kmax,varargin)  
%GOLDEN Trova un punto di minimo di una funzione.  
% XMIN=GOLDEN(FUN,A,B,TOL,KMAX) approssima un  
% punto di minimo della funzione FUN nell'intervallo  
% [A,B] con il metodo della sezione aurea.  
% Se la ricerca del punto di minimo di FUN fallisce,  
% il programma restituisce un messaggio d'errore.  
% FUN puo' essere una anonymous function  
% od una function definita in un M-file.  
% XMIN=GOLDEN(FUN,A,B,TOL,KMAX,P1,P2,...) passa  
% i parametri P1, P2,... alla funzione  
% FUN(X,P1,P2,...).  
% [XMIN,FMIN,ITER]= GOLDEN(FUN,...) restituisce  
% il valore della funzione FUN in XMIN e il numero  
% di iterazioni effettuate per calcolare XMIN.  
phi=(1+sqrt(5))/2; phi1=1/phi; phi2=1/(phi+1);  
c=a+phi2*(b-a); d=a+phi1*(b-a);
```

```

err=tol+1; k=0;
while err>tol && k< kmax
    if(fun(c) >= fun(d))
        a=c; c=d; d=a+phi1*(b-a);
    else
        b=d; d=c; c=a+phi2*(b-a);
    end
    k=k+1; err=abs(b-a)/(abs(c)+abs(d));
end
xmin=(a+b)/2; fmin=fun(xmin); iter=k;
if (iter==kmax && err > tol)
    fprintf(['Il metodo della sez. aurea si e' '\n',...
    'arrestato senza soddisfare la tolleranza '\n',...
    'richiesta avendo raggiunto il numero massimo '\n',...
    'di iterazioni\n']);
end

```

Esempio 7.1 Risolviamo con il metodo della sezione aurea il Problema 7.1. La funzione $f(t) = -b'(t) = -7500e^{t/3}/(e^{-t/3}+9)^2$ ammette un punto di minimo globale nell'intervallo $[6, 7]$, come possiamo osservare dal suo plot generato con le istruzioni `MAT&OCT`:

```

f=@(t) [-(7500*exp(t/3))/((exp(t/3) + 9)^2]
a=0; b=10; fplot(f,[a,b]); grid on

```

Richiamiamo il Programma 7.1 con una tolleranza per il test d'arresto pari a 10^{-8} e un numero massimo di iterazioni `kmax=100` utilizzando le istruzioni:

```

tol=1.e-8; kmax=100;
[tmin,fmin,iter]=golden(f,a,b,tol,kmax)

```

Dopo 38 iterazioni otteniamo

```

xmin=6.591673759332620          fmin=-2.08333333333333e+02

```

cioè il fattore di crescita massimo cercato è di $208.\overline{3}$ batteri al giorno e si realizza circa dopo 6.59 giorni dall'inizio della coltura. ■

Un metodo alternativo, ma spesso utilizzato in modo complementare a quello della sezione aurea, è il metodo dell'interpolazione quadratica. Sia f una funzione continua. Dati tre punti distinti $x^{(0)}$, $x^{(1)}$ e $x^{(2)}$, costruiamo una successione di punti $x^{(k)}$, con $k \geq 3$, tale che $x^{(k+1)}$ sia il vertice (e quindi punto di minimo) della parabola $p_k^{(2)}$ che interpola f nei punti $x^{(k)}$, $x^{(k-1)}$ e $x^{(k-2)}$ (si veda Figura 7.5):

$$\begin{aligned}
p_2^{(k)}(x) &= f(x^{(k-2)}) + f[x^{(k-2)}, x^{(k-1)}](x - x^{(k-2)}) \\
&\quad + f[x^{(k-2)}, x^{(k-1)}, x^{(k)}](x - x^{(k-2)})(x - x^{(k-1)}),
\end{aligned}$$

dove i valori reali

$$f[x_i, x_j] = \frac{f(x_j) - f(x_i)}{x_j - x_i}, \quad f[x_i, x_j, x_\ell] = \frac{f[x_j, x_\ell] - f[x_i, x_j]}{x_\ell - x_i} \tag{7.20}$$

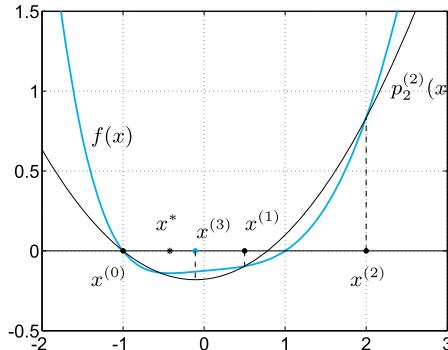


Figura 7.5. Il primo passo del metodo dell’interpolazione quadratica

sono le cosiddette *differenze divise di Newton* (si veda [QSSG14, Cap. 8]). Risolvendo l’equazione $p_2^{(k)}(x) = 0$, otteniamo che il punto di minimo di $p_2^{(k)}$ è

$$x^{(k+1)} = \frac{1}{2} \left(x^{(k-2)} + x^{(k-1)} - \frac{f[x^{(k-2)}, x^{(k-1)}]}{f[x^{(k-2)}, x^{(k-1)}, x^{(k)}]} \right) \quad (7.21)$$

Assegnata una tolleranza ε , si itera fino a quando $|x^{(k+1)} - x^{(k)}| < \varepsilon$ e quindi si accetta $x^{(k+1)}$ come approssimazione del punto x^* .

Se per ogni passo k la differenza divisa $f[x^{(k-2)}, x^{(k-1)}, x^{(k)}]$ non si annulla, questo metodo ha una convergenza super-lineare, precisamente converge al punto di minimo con un ordine $p \simeq 1.3247$ (si veda [Bre02]), altrimenti il metodo può non giungere a terminazione. Per questo motivo il metodo dell’interpolazione quadratica non viene implementato da solo, bensì in combinazione con altri metodi, come ad esempio il metodo della sezione aurea, la cui convergenza è garantita.

Il comando `fminbnd` di MATLAB calcola il punto di minimo di una funzione reale abbinando le due tecniche appena viste. La sintassi di chiamata è: `x = fminbnd(fun,a,b)` essendo `fun` il *function handle* associato alla funzione obiettivo e `a`, `b` gli estremi dell’intervallo in cui cercare il punto di minimo. L’output `x` fornisce la stima del punto di minimo cercato.

Esempio 7.2 Risolviamo con la function `fminbnd` lo stesso caso dell’Esempio 7.1 e fissiamo anche per questo metodo una tolleranza per il test d’arresto pari a 10^{-8} . Con le istruzioni:

```
options=optimset('TolX',1.e-8);
[tmin1,fmin1,exitflag,output]=fminbnd(f,a,b,options)
```

otteniamo convergenza a `fmin1= 6.591673708945312` in 8 iterazioni in MATLAB ed in 15 iterazioni in Octave (l’interpolazione quadratica viene utilizzata

diversamente dai due ambienti e questo influenza la velocità di convergenza del metodo), molto meno rispetto alle 38 iterazioni richieste dal solo metodo della sezione aurea. Il comando `optimset` permette di assegnare un valore desiderato alla tolleranza (`tol=1.e-8` in questo caso) differente da quello che sarebbe definito di default (`tol=1.e-4`). I parametri opzionali di output sono: `fmin1`, che contiene la valutazione della f nel punto di minimo, `exitflag`, che precisa lo stato di terminazione, e `output`, che è una struttura (si veda la Sezione 1.5) contenente il numero di iterazioni effettuate ed il numero di valutazioni funzionali richieste nei vari passi. ■

I due algoritmi finora visti non si prestano ad una estensione al caso di funzioni in più variabili, tuttavia essi possono essere utilizzati all'interno dei metodi di discesa (si veda la Sezione 7.5) per la ricerca di minimi di funzioni in più variabili, nel momento in cui il problema multidimensionale si riconduce alla risoluzione successiva di problemi monodimensionali.

7.3.2 Il metodo di Nelder e Mead

Sia $n > 1$, l' n -simplesso con $n + 1$ vertici $\mathbf{x}_i \in \mathbb{R}^n$ (con $i = 0, \dots, n$) è l'insieme

$$S = \left\{ \mathbf{y} \in \mathbb{R}^n : \mathbf{y} = \sum_{i=0}^n \lambda_i \mathbf{x}_i, \lambda_i \in \mathbb{R} \text{ e } \lambda_i \geq 0 : \sum_{i=0}^n \lambda_i = 1 \right\}, \quad (7.22)$$

a condizione che gli n vettori $\mathbf{x}_i - \mathbf{x}_0$ ($i = 1, \dots, n$) siano linearmente indipendenti. S risulta essere un segmento in \mathbb{R} , un triangolo in \mathbb{R}^2 e un tetraedro in \mathbb{R}^3 .

Data una funzione costo $f : \mathbb{R}^n \rightarrow \mathbb{R}$ continua, il metodo di Nelder e Mead [NM65] è un algoritmo di tipo *derivative free* per il calcolo del punto di minimo $\mathbf{x}^* \in \mathbb{R}^n$ della funzione costo f . Esso genera una successione di simplessi $\{S^{(k)}\}_{k \geq 0}$ in \mathbb{R}^n che inseguono o contengono il punto di minimo \mathbf{x}^* , utilizzando le valutazioni di f nei vertici dei simplessi e semplici trasformazioni geometriche come riflessioni, dilatazioni e contrazioni.

Per generare il simplesso iniziale $S^{(0)}$ prendiamo un punto $\tilde{\mathbf{x}} \in \mathbb{R}^n$ ed un numero reale positivo η , e poniamo $\mathbf{x}_i^{(0)} = \tilde{\mathbf{x}} + \eta \mathbf{e}_i$ per $i = 0, \dots, n$, dove \mathbf{e}_i sono i vettori della base canonica in \mathbb{R}^n .

Per ogni $k \geq 0$ (fino a convergenza) selezioniamo il vertice “peggiore” di $S^{(k)}$

$$\mathbf{x}_M^{(k)} = \underset{0 \leq i \leq n}{\operatorname{argmax}} f(\mathbf{x}_i^{(k)}) \quad (7.23)$$

e lo sostituiamo con un nuovo punto, al fine di costruire un nuovo simplesso $S^{(k+1)}$, migliore del precedente.

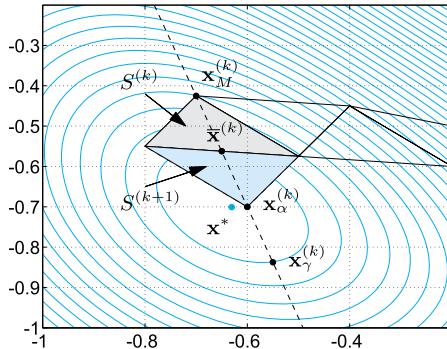


Figura 7.6. Un passo del metodo di Nelder e Mead, il punto $\mathbf{x}_M^{(k)}$ è sostituito da $\mathbf{x}_{\alpha}^{(k)}$

Il nuovo punto è scelto come segue. Dapprima selezioniamo

$$\mathbf{x}_m^{(k)} = \underset{0 \leq i \leq n}{\operatorname{argmin}} f(\mathbf{x}_i^{(k)}) \quad \text{e} \quad \mathbf{x}_{\mu}^{(k)} = \underset{\substack{0 \leq i \leq n \\ i \neq M}}{\operatorname{argmax}} f(\mathbf{x}_i^{(k)}) \quad (7.24)$$

($\mathbf{x}_m^{(k)}$) è il punto di minimo tra i vertici del simplex, mentre $\mathbf{x}_{\mu}^{(k)}$ è il punto di massimo tra i vertici del simplex, ad esclusione di $\mathbf{x}_M^{(k)}$) e definiamo il *centroide* dell'iperpiano $H^{(k)}$ che passa per i vertici $\{\mathbf{x}_i^{(k)}, i = 0, \dots, n, i \neq M\}$

$$\bar{\mathbf{x}}^{(k)} = \frac{1}{n} \sum_{\substack{i=0 \\ i \neq M}}^n \mathbf{x}_i^{(k)}. \quad (7.25)$$

(Quando $n = 2$, il centroide è il punto medio del lato di $S^{(k)}$ opposto a $\mathbf{x}_M^{(k)}$, si veda Figura 7.6.)

Quindi calcoliamo la riflessione $\mathbf{x}_{\alpha}^{(k)}$ di $\mathbf{x}_M^{(k)}$ rispetto all'iperpiano $H^{(k)}$, ovvero

$$\mathbf{x}_{\alpha}^{(k)} = (1 - \alpha) \bar{\mathbf{x}}^{(k)} + \alpha \mathbf{x}_M^{(k)}, \quad (7.26)$$

dove $\alpha < 0$ è il coefficiente di riflessione (tipicamente $\alpha = -1$). Il punto $\mathbf{x}_{\alpha}^{(k)}$ giace sulla linea retta che passa per $\bar{\mathbf{x}}^{(k)}$ e $\mathbf{x}_M^{(k)}$, dalla parte di $\bar{\mathbf{x}}^{(k)}$ opposta a $\mathbf{x}_M^{(k)}$ (si veda Figura 7.6). Prima di accettare $\mathbf{x}_{\alpha}^{(k)}$ come nuovo vertice, confrontiamo $f(\mathbf{x}_{\alpha}^{(k)})$ con i valori di f negli altri vertici del simplex. Contemporaneamente, muoviamo $\mathbf{x}_{\alpha}^{(k)}$ sulla retta che congiunge $\bar{\mathbf{x}}^{(k)}$ e $\mathbf{x}_M^{(k)}$, al fine di definire il nuovo simplex $S^{(k+1)}$. Più precisamente procediamo come segue.

- Se $f(\mathbf{x}_{\alpha}^{(k)}) < f(\mathbf{x}_m^{(k)})$, cioè la riflessione ha prodotto un nuovo minimo, calcoliamo

$$\mathbf{x}_\gamma^{(k)} = (1 - \gamma)\bar{\mathbf{x}}^{(k)} + \gamma\mathbf{x}_M^{(k)}, \quad (7.27)$$

dove $\gamma < -1$ (tipicamente, $\gamma = -2$). Quindi, se $f(\mathbf{x}_\gamma^{(k)}) < f(\mathbf{x}_m^{(k)})$, $\mathbf{x}_M^{(k)}$ è sostituito da $\mathbf{x}_\gamma^{(k)}$; altrimenti $\mathbf{x}_M^{(k)}$ è sostituito da $\mathbf{x}_\alpha^{(k)}$; quindi procediamo incrementando k di uno.

- Se $f(\mathbf{x}_m^{(k)}) \leq f(\mathbf{x}_\alpha^{(k)}) < f(\mathbf{x}_\mu^{(k)})$, $\mathbf{x}_M^{(k)}$ è sostituito da $\mathbf{x}_\alpha^{(k)}$; quindi incrementiamo k di uno.
- Se $f(\mathbf{x}_\mu^{(k)}) \leq f(\mathbf{x}_\alpha^{(k)}) < f(\mathbf{x}_M^{(k)})$ calcoliamo

$$\mathbf{x}_\beta^{(k)} = (1 - \beta)\bar{\mathbf{x}}^{(k)} + \beta\mathbf{x}_\alpha^{(k)}, \quad (7.28)$$

dove $\beta > 0$ (tipicamente, $\beta = 1/2$). Ora, se $f(\mathbf{x}_\beta^{(k)}) > f(\mathbf{x}_M^{(k)})$ definiamo i vertici del nuovo simplesso $S^{(k+1)}$ come

$$\mathbf{x}_i^{(k+1)} = \frac{1}{2}(\mathbf{x}_i^{(k)} + \mathbf{x}_m^{(k)}) \quad (7.29)$$

altrimenti $\mathbf{x}_M^{(k)}$ è sostituito da $\mathbf{x}_\beta^{(k)}$; quindi incrementiamo k di uno.

- Se $f(\mathbf{x}_\alpha^{(k)}) > f(\mathbf{x}_M^{(k)})$ calcoliamo

$$\mathbf{x}_\beta^{(k)} = (1 - \beta)\bar{\mathbf{x}}^{(k)} + \beta\mathbf{x}_M^{(k)}, \quad (7.30)$$

(ancora $\beta > 0$), se $f(\mathbf{x}_\beta^{(k)}) > f(\mathbf{x}_M^{(k)})$ definiamo i vertici del nuovo simplesso $S^{(k+1)}$ tramite (7.29), altrimenti $\mathbf{x}_M^{(k)}$ è sostituito da $\mathbf{x}_\beta^{(k)}$; quindi incrementiamo k di uno.

Non appena il criterio d'arresto $\max_{i=0,\dots,n} \|\mathbf{x}_i^{(k)} - \mathbf{x}_m^{(k)}\|_\infty < \varepsilon$ è soddisfatto, $\mathbf{x}_m^{(k)}$ sarà accettato come approssimazione del punto di minimo.

La convergenza del metodo di Nelder e Mead è garantita solo in casi molto speciali (si veda ad esempio [LRWW99]); infatti potrebbe verificarsi una stagnazione dell'algoritmo, che in tal caso deve essere fatto ripartire. Tuttavia in genere esso risulta abbastanza robusto ed efficiente per problemi di piccole dimensioni. La velocità di convergenza del metodo dipende fortemente dalla scelta dei vertici del simplesso iniziale. Il **fminsearch** metodo di Nelder e Mead è implementato nel comando **fminsearch** di **MATLAB**, la sua sintassi di chiamata è descritta nel prossimo esempio.

Esempio 7.3 (La funzione di Rosenbrock) La funzione di Rosenbrock

$$f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2,$$

le cui linee di livello sono rappresentate in Figura 7.7 [Ros61], è spesso utilizzata per testare l'efficienza e la robustezza dei metodi di minimizzazione. Pur avendo un unico punto di minimo globale $\mathbf{x}^* = (1, 1)$, f ha minime variazioni in una vasta zona attorno ad \mathbf{x}^* , cosicché risulta costoso (in termini computazionali) raggiungere \mathbf{x}^* . Per approssimare \mathbf{x}^* con **fminsearch** utilizziamo i comandi:

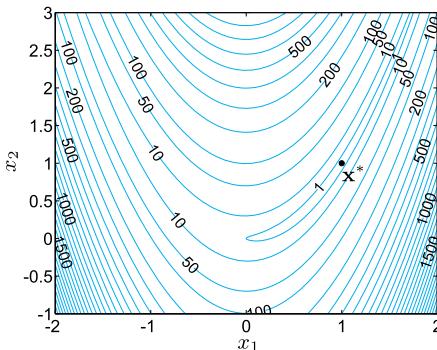


Figura 7.7. Linee di livello della funzione di Rosenbrock

```
fun=@(x) 100*(x(2)-x(1)^2)^2+(1-x(1))^2; x0=[-1.2,1]
xstar=fminsearch(fun,x0)
```

e otteniamo

```
xstar =
1.000022021783570      1.000042219751772
```

In MATLAB, sostituendo il comando di chiamata `xstar=fminsearch(fun,x0)` con

```
[xstar,fval,exitflag,output]=fminsearch(fun,x0)
```

otteniamo informazioni anche sul valore del minimo calcolato `fval=8.1777e-10` e sul numero di iterazioni e valutazioni funzionali effettuate, rispettivamente `output.iterations=85` e `output.funcCount=159`. Infine la tolleranza sul test d'arresto può essere ridotta richiamando il comando `optimset` come visto nell'Esempio 7.2. ■

Si vedano gli Esercizi 7.1–7.3.



7.4 Il metodo di Newton

Se la funzione $f : \mathbb{R}^n \rightarrow \mathbb{R}$ (con $n \geq 1$) è di classe $C^2(\mathbb{R}^n)$ e siamo in grado di calcolarne le derivate parziali prime e seconde, l'idea più naturale per il calcolo di un punto di minimo \mathbf{x}^* è di applicare il metodo di Newton visto nel Capitolo 2 all'equazione vettoriale $\mathbf{F}(\mathbf{x}) = \nabla f(\mathbf{x}) = \mathbf{0}$, in cui la matrice Jacobiana $J_{\mathbf{F}}(\mathbf{x}^{(k)})$ altro non è che la matrice Hessiana di f valutata nel punto $\mathbf{x}^{(k)}$. Il metodo si formula come segue: dato $\mathbf{x}^{(0)} \in \mathbb{R}^n$, per $k = 0, 1, \dots$ e fino a convergenza

$$\begin{aligned} &\text{risolvere } H(\mathbf{x}^{(k)})\delta\mathbf{x}^{(k)} = -\nabla f(\mathbf{x}^{(k)}) \\ &\text{porre } \mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \delta\mathbf{x}^{(k)} \end{aligned} \tag{7.31}$$

Fissata una tolleranza $\varepsilon > 0$, un possibile test d'arresto è

$$\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\| \leq \varepsilon.$$

Esempio 7.4 Consideriamo la funzione

$$f(\mathbf{x}) = \frac{2}{5} - \frac{1}{10}(5x_1^2 + 5x_2^2 + 3x_1x_2 - x_1 - 2x_2)e^{-(x_1^2+x_2^2)} \quad (7.32)$$

che è rappresentata a destra di Figura 7.8. Applichiamo il metodo di Newton per approssimare il suo punto di minimo globale $\mathbf{x}^* \simeq (-0.630658, -0.700742)$ (riportiamo le sole prime 6 cifre significative). Prendiamo $\mathbf{x}^{(0)} = (-0.9, -0.9)$ come punto iniziale e fissiamo una tolleranza $\varepsilon = 10^{-5}$ per il test d'arresto. Il metodo (7.31) converge in 5 iterazioni a $\mathbf{x} = [-0.63058; -0.70074]$. Scegliendo invece $\mathbf{x}^{(0)} = (-1, -1)$, dopo 400 iterazioni lo stesso metodo non giunge a convergenza. Come già abbiamo visto nella risoluzione di equazioni non lineari (si veda la Sezione 2.3), questo comportamento riflette la proprietà di *convergenza locale* e quadratica del metodo di Newton, cioè la convergenza ad \mathbf{x}^* è garantita solo se $\mathbf{x}^{(0)}$ è sufficientemente vicino al punto \mathbf{x}^* stesso e, in tal caso, la convergenza è molto veloce. Conviene osservare che il metodo di Newton applicato all'equazione $\nabla f(\mathbf{x}) = \mathbf{0}$ non necessariamente converge ad un punto di minimo di f . Esso in realtà può convergere ad un qualsiasi punto stazionario di f ; se partiamo ad esempio da $\mathbf{x}^{(0)} = (0.5, -0.5)$ il metodo converge al punto stazionario $\mathbf{x} = [0.80659; -0.54010]$ (né di minimo né di massimo) in 5 iterazioni. ■

Un criterio di convergenza per il metodo (7.31) è il seguente: se f è di classe $C^2(\mathbb{R}^n)$, se \mathbf{x}^* è un punto stazionario e la matrice Hessiana $H(\mathbf{x}^*)$ è definita positiva, se le componenti della matrice Hessiana $H(\mathbf{x})$ sono lipschitziane in un intorno di \mathbf{x}^* e $\mathbf{x}^{(0)}$ è sufficientemente vicino a \mathbf{x}^* allora il metodo di Newton (7.31) converge quadraticamente al punto di minimo \mathbf{x}^* (si vedano, ad esempio [SY06], pag. 132), [NW06]).

La semplicità di questo metodo contrasta da un lato con la sua pesantezza computazionale quando n è molto grande (dobbiamo fornire le derivate in forma analitica e valutare il gradiente e la matrice Hessiana ad ogni iterazione), dall'altro con la proprietà di *convergenza locale*.

L'idea di base per costruire algoritmi efficienti e robusti per la minimizzazione è allora quella di combinare metodi veloci ma a convergenza locale con una strategia *globalmente convergente*, cioè tale da garantire la convergenza ad un punto di minimo (non necessariamente il punto di minimo globale) per ogni $\mathbf{x}^{(0)} \in \mathbb{R}^n$. Ciò dà luogo ai metodi di discesa che presentiamo nella prossima Sezione.

7.5 Metodi di discesa o *line-search*

Per semplicità di esposizione, supponiamo lungo tutta questa sezione che la funzione f sia di classe $C^2(\mathbb{R})$ e che sia limitata inferiormente.

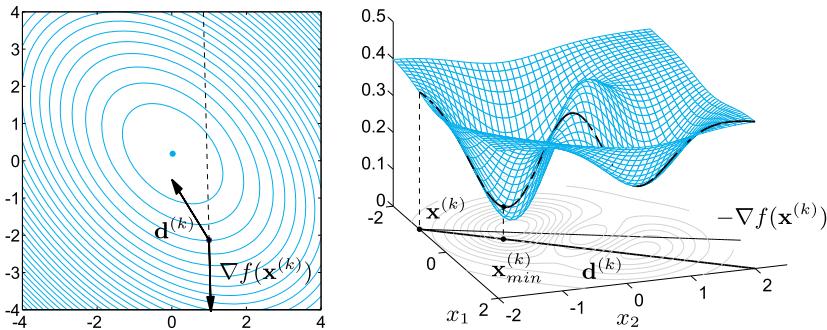


Figura 7.8. A sinistra, linee di livello di una funzione $f(\mathbf{x})$, il suo vettore gradiente in $\mathbf{x}^{(k)}$ e una possibile direzione di discesa $\mathbf{d}^{(k)}$. A destra, restrizione di una funzione $f(\mathbf{x})$ ad una direzione di discesa $\mathbf{d}^{(k)}$ e punto di minimo $\mathbf{x}_{\min}^{(k)}$ lungo la direzione $\mathbf{d}^{(k)}$

I metodi di discesa (noti anche come metodi *line-search*) sono metodi iterativi in cui, per ogni $k \geq 0$, $\mathbf{x}^{(k+1)}$ dipende da $\mathbf{x}^{(k)}$, da una direzione di discesa $\mathbf{d}^{(k)}$ che dipende a sua volta dal vettore gradiente $\nabla f(\mathbf{x}^{(k)})$ (si vedano la definizione (5.67) e la Figura 7.8) e da un passo $\alpha_k \in \mathbb{R}$.

Assegnato un punto iniziale $\mathbf{x}^{(0)} \in \mathbb{R}^n$, un metodo di discesa procede come segue: per $k = 0, 1, \dots$ fino a convergenza

$$\boxed{\begin{aligned} &\text{determinare una direzione } \mathbf{d}^{(k)} \in \mathbb{R}^n \\ &\text{determinare un passo } \alpha_k \in \mathbb{R} \\ &\text{porre } \mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k \mathbf{d}^{(k)} \end{aligned}} \quad (7.33)$$

Nella Sezione seguente riportiamo alcune possibili scelte per le direzioni di discesa, in particolare quelle che danno forma ai metodi più diffusi.

Una volta individuato il vettore $\mathbf{d}^{(k)}$, il valore ottimale del passo $\alpha_k \in \mathbb{R}$ è quello che garantisce la massima variazione (negativa) di f lungo la direzione $\mathbf{d}^{(k)}$ e può essere calcolato minimizzando la restrizione di f lungo $\mathbf{d}^{(k)}$ (si veda la Figura 7.8, a destra). In realtà, come vedremo nella Sezione 7.5.2, tale calcolo è molto oneroso se f non è una funzione quadratica e quindi si adottano delle tecniche di approssimazione del passo ottimale, tali da garantire la convergenza al punto di minimo.

7.5.1 Direzioni di discesa

Le direzioni di discesa $\mathbf{d}^{(k)}$ più note sono:

1. *direzioni di Newton*

$$\boxed{\mathbf{d}^{(k)} = -(\mathbf{H}(\mathbf{x}^{(k)}))^{-1} \nabla f(\mathbf{x}^{(k)})} \quad (7.34)$$

2. direzioni quasi-Newton

$$\mathbf{d}^{(k)} = -\mathbf{H}_k^{-1} \nabla f(\mathbf{x}^{(k)}) \quad (7.35)$$

dove \mathbf{H}_k è un'approssimazione della matrice Hessiana $\mathbf{H}(\mathbf{x}^{(k)})$. Questa scelta rappresenta una valida alternativa al metodo di Newton quando non sia possibile o sia molto costoso calcolare le derivate seconde della funzione f (si veda la Sezione 7.5.4);

3. direzioni del gradiente

$$\mathbf{d}^{(k)} = -\nabla f(\mathbf{x}^{(k)}) \quad (7.36)$$

(possono essere interpretate come direzioni quasi-Newton con la scelta triviale $\mathbf{H}_k = \mathbf{I}$), nel caso particolare in cui f è quadratica e convessa ritroviamo (5.73);

4. direzioni del gradiente coniugato

$$\begin{aligned} \mathbf{d}^{(0)} &= -\nabla f(\mathbf{x}^{(0)}) \\ \mathbf{d}^{(k+1)} &= -\nabla f(\mathbf{x}^{(k+1)}) + \beta_k \mathbf{d}^{(k)}, \quad \text{per } k \geq 0 \end{aligned} \quad (7.37)$$

I coefficienti β_k possono essere scelti secondo diversi criteri (si veda la Sezione 7.5.5), ma sempre in modo che queste direzioni di discesa coincidano con quelle del gradiente coniugato per sistemi lineari quando la funzione f è quadratica e convessa, si vedano (5.79) e (5.80).

Le direzioni (7.36) verificano le condizioni (5.67) (la verifica è immediata), i vettori (7.34) e (7.35) sono direzioni di discesa solo quando $\mathbf{H}(\mathbf{x}^{(k)})$ e \mathbf{H}_k (rispettivamente) sono matrici definite positive. Infine, i vettori (7.37) sono di discesa a patto che i coefficienti β_k vengano calcolati opportunamente, come sarà precisato nella Sezione 7.5.5.

Esempio 7.5 Consideriamo ancora la funzione $f(\mathbf{x})$ (7.32); essa ha due punti di minimo locale, un punto di massimo locale e due punti di sella. Vogliamo confrontare le successioni $\{\mathbf{x}^{(k)}\}$ generate dal metodo di Newton (7.31) e dai metodi di discesa utilizzando le direzioni (7.34)–(7.37).

Dapprima consideriamo il punto iniziale $\mathbf{x}_1^{(0)} = (0.5, -0.5)$. In Figura 7.9 vediamo che il metodo di Newton (7.31) converge al punto di sella $(0.8065, -0.5401)$; il metodo di discesa con direzioni di Newton (7.34) esegue un primo passo come il metodo di Newton, ma poi si arresta in quanto si genera una matrice $\mathbf{H}(\mathbf{x}^{(1)})$ non definita positiva (si veda l'Osservazione 7.2 per ovviare a questo inconveniente); gli altri metodi di discesa con direzioni (7.36), (7.35) e (7.37) (nell'ultimo caso utilizziamo due criteri diversi (FR e PR) per definire i parametri β_k , si veda Sezione 7.5.5) convergono al punto di minimo

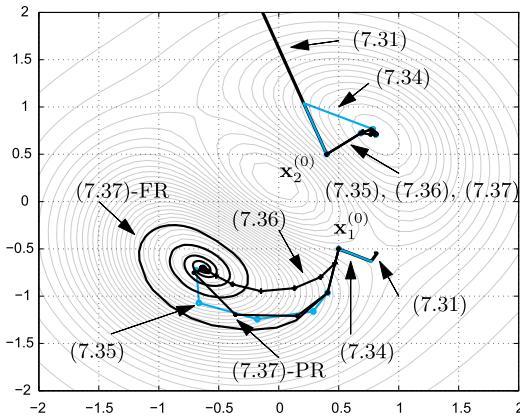


Figura 7.9. Storie di convergenza dei metodi di Newton e di discesa per la funzione dell’Esempio 7.5

locale $(-0.6306, -0.7007)$. La convergenza più veloce è ottenuta in 9 iterazioni utilizzando le direzioni quasi-Newton (7.35) (la curva in azzurro in Figura 7.9). Partendo invece dal punto iniziale $\mathbf{x}_2^{(0)} = (0.4, 0.5)$, il metodo di Newton (7.31) diverge, mentre il metodo (7.34), sebbene condivida la medesima direzione $\mathbf{d}^{(1)}$ con il metodo di Newton, costruisce un passo α_k minore rispetto a Newton e ciò porta alla convergenza al punto di minimo locale $(0.8095, 0.7097)$ in sole 4 iterazioni. Tutte le altre direzioni (7.35), (7.36), e (7.37) convergono in 10–15 iterazioni allo stesso punto di minimo locale. ■

La scelta del passo α_k sarà discussa nella Sezione 7.5.2, mentre l’analisi delle diverse direzioni di discesa è affrontata nelle Sezioni 7.5.3–7.5.5.

7.5.2 Strategie per il calcolo del passo α_k

Una volta che la direzione di discesa $\mathbf{d}^{(k)}$ è stata determinata, si deve calcolare il passo α_k in modo che il nuovo punto $\mathbf{x}^{(k+1)}$ sia il punto di minimo (o una sua approssimazione) di f lungo tale direzione.

La strategia più naturale consiste nello scegliere α_k che garantisca la minimizzazione esatta, ovvero

$$\alpha_k = \underset{\alpha \in \mathbb{R}}{\operatorname{argmin}} f(\mathbf{x}^{(k)} + \alpha \mathbf{d}^{(k)}).$$

Nel caso particolare in cui f sia una funzione quadratica, cioè della forma

$$f(\mathbf{x}) = \frac{1}{2} \mathbf{x}^T \mathbf{A} \mathbf{x} - \mathbf{x}^T \mathbf{b} + c$$

con $\mathbf{A} \in \mathbb{R}^{n \times n}$ simmetrica e definita positiva, $\mathbf{b} \in \mathbb{R}^n$ e $c \in \mathbb{R}$, differenziando f rispetto ad α e imponendo che la derivata prima sia nulla,

ritroviamo il parametro (5.70) costruito per i metodi del gradiente e del gradiente coniugato. Nel caso in cui le $\mathbf{d}^{(k)}$ siano le direzioni del gradiente (7.36), abbiamo $\mathbf{d}^{(k)} = \mathbf{r}^{(k)}$ e ritroviamo il metodo del gradiente per il quale vale la stima di convergenza (5.75). Se invece le $\mathbf{d}^{(k)}$ sono le direzioni del gradiente coniugato (7.37), definendo β_k come in (5.80), ritroviamo il metodo del gradiente coniugato per sistemi lineari (5.83) per cui vale la stima di convergenza (5.84).

Per una generica f (non quadratica), calcolare α_k in maniera ottimale richiederebbe di implementare un metodo iterativo per risolvere un problema di minimizzazione lungo la direzione $\mathbf{d}^{(k)}$. In questi casi può risultare più vantaggioso dal punto di vista computazionale calcolare un valore approssimato (invece che esatto) di α_k , in modo che la nuova iterata $\mathbf{x}^{(k+1)}$ definita in (7.33) soddisfi

$$f(\mathbf{x}^{(k+1)}) < f(\mathbf{x}^{(k)}). \quad (7.38)$$

Una strategia semplice da implementare potrebbe consistere nell'assegnare ad α_k un valore grande per poi ridurlo iterativamente finché sia soddisfatta la stima (7.38). Sfortunatamente questa strada non garantisce che la successione $\mathbf{x}^{(k)}$ costruita dal metodo di discesa converga al punto di minimo \mathbf{x}^* desiderato. Si vedano l'Esercizio 7.4 e la relativa Figura 10.14, a sinistra, per un esempio in cui i passi α_k calcolati con la strategia sopra descritta risultano troppo lunghi (e non si raggiunge convergenza), e l'Esercizio 7.5 e la relativa Figura 10.14, a destra, per un esempio in cui i passi α_k risultano troppo corti (anche in questo caso non si ha convergenza).

Un criterio migliore per la scelta dei passi α_k (positivi) è quello basato sulle *condizioni di Wolfe*:

$$\begin{aligned} f(\mathbf{x}^{(k)} + \alpha_k \mathbf{d}^{(k)}) &\leq f(\mathbf{x}^{(k)}) + \sigma \alpha_k \mathbf{d}^{(k)T} \nabla f(\mathbf{x}^{(k)}) \\ \mathbf{d}^{(k)T} \nabla f(\mathbf{x}^{(k)} + \alpha_k \mathbf{d}^{(k)}) &\geq \delta \mathbf{d}^{(k)T} \nabla f(\mathbf{x}^{(k)}) \end{aligned} \quad (7.39)$$

dove σ e δ , tali che $0 < \sigma < \delta < 1$, sono due costanti assegnate e $\mathbf{d}^{(k)T} \nabla f(\mathbf{x}^{(k)})$ è la derivata direzionale di f lungo la direzione $\mathbf{d}^{(k)}$.

La prima diseguaglianza in (7.39) è nota anche come *regola di Armijo*, essa impedisce che si verifichino variazioni di f troppo piccole rispetto alla lunghezza del passo α_k (si veda Figura 7.10, a sinistra). Più precisamente, maggiore è α_k e maggiore è la variazione $f(\mathbf{x}^{(k)}) - f(\mathbf{x}^{(k+1)})$ che si deve ottenere.

La seconda condizione di Wolfe asserisce che il valore della derivata di f lungo al direzione $\mathbf{d}^{(k)}$ nel nuovo punto $\mathbf{x}^{(k)} + \alpha_k \mathbf{d}^{(k)}$ debba essere maggiore del prodotto tra δ e la derivata direzionale di f nel punto $\mathbf{x}^{(k)}$ (si veda Figura 7.10, destra).

Dall'esempio riportato in Figura 7.10 si evince che le condizioni di Wolfe possono essere verificate anche lontano dal punto di minimo lungo

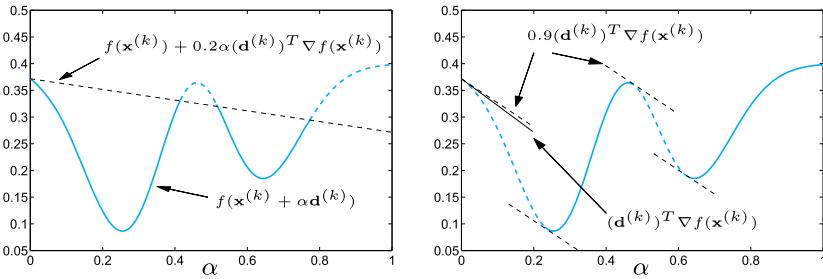


Figura 7.10. Le curve azzurre rappresentano i valori di $f(\mathbf{x}^{(k)}) + \alpha \mathbf{d}^{(k)}$ al variare di α . I tratti disegnati con linea continua corrispondono ai valori di α che soddisfano le condizioni di Wolfe (7.39): la prima condizione con $\sigma = 0.2$ a sinistra, la seconda condizione con $\delta = 0.9$ a destra. Le linee nere tratteggiate nella figura di destra hanno pendenza pari a $0.9\mathbf{d}^{(k)^T} \nabla f(\mathbf{x}^{(k)})$. Le due condizioni di Wolfe sono verificate simultaneamente per $0.23 \leq \alpha \leq 0.41$ o $0.62 \leq \alpha \leq 0.77$

la direzione $\mathbf{d}^{(k)}$ o dove la derivata direzionale di f assume valori grandi. Condizioni più restrittive di (7.39) sono le *condizioni forti di Wolfe*,

$$\begin{aligned} f(\mathbf{x}^{(k)} + \alpha_k \mathbf{d}^{(k)}) &\leq f(\mathbf{x}^{(k)}) + \sigma \alpha_k \mathbf{d}^{(k)^T} \nabla f(\mathbf{x}^{(k)}) \\ |\mathbf{d}^{(k)^T} \nabla f(\mathbf{x}^{(k)} + \alpha_k \mathbf{d}^{(k)})| &\leq -\delta \mathbf{d}^{(k)^T} \nabla f(\mathbf{x}^{(k)}) \end{aligned} \quad (7.40)$$

essendo $0 < \sigma < \delta < 1$ costanti fissate.

La prima condizione di (7.40) coincide con la prima di (7.39), mentre (7.40)₂ garantisce sia (7.39)₂, sia $\mathbf{d}^{(k)^T} \nabla f(\mathbf{x}^{(k)} + \alpha_k \mathbf{d}^{(k)}) \leq -\delta \mathbf{d}^{(k)^T} \nabla f(\mathbf{x}^{(k)})$ (osserviamo che il termine destro di (7.40)₂ è positivo grazie a (5.67)₁). La condizione (7.40)₂ vincola la scelta di α_k in modo che la derivata direzionale di f in $\mathbf{x}^{(k)} + \alpha_k \mathbf{d}^{(k)}$ non sia troppo grande in valore assoluto (si veda Figura 7.11 per un esempio).

Si può dimostrare (si veda, ad esempio [NW06, Lemma 3.1]) che, se $\mathbf{d}^{(k)}$ è una direzione di discesa in $\mathbf{x}^{(k)}$ e $f \in C^1(\mathbb{R}^n)$ è limitata inferiormente sull'insieme $\{\mathbf{x}^{(k)} + \alpha \mathbf{d}^{(k)}, \alpha > 0\}$, allora, presi $0 < \sigma < \delta < 1$, esistono degli intervalli non vuoti all'interno dei quali scegliere α_k tali che le condizioni (7.39) e (7.40) siano soddisfatte.

Nella pratica il coefficiente σ viene scelto abbastanza piccolo, ad esempio $\sigma = 10^{-4}$ [NW06], mentre valori tipici per δ sono $\delta = 0.9$ per le direzioni di Newton, quasi-Newton e del gradiente, o $\delta = 0.1$ per le direzioni del gradiente coniugato.

Una semplice strategia per determinare il passo α_k che soddisfi le condizioni di Wolfe è quella del *backtracking*, che consiste nel prendere $\alpha = 1$ e nel ridurlo di un fattore ρ assegnato (tipicamente $\rho \in [1/10, 1/2]$) fino a quando viene soddisfatta la prima condizione di (7.39). Tale strategia si formula come segue: dati il punto $\mathbf{x}^{(k)}$ e la direzione di discesa $\mathbf{d}^{(k)}$,

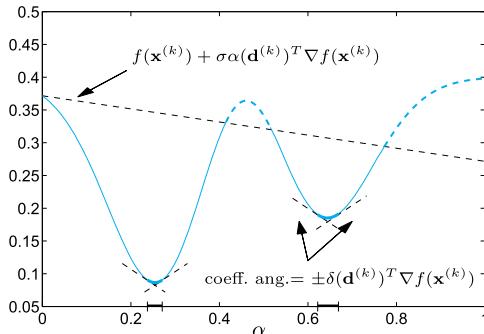


Figura 7.11. Le condizioni forti di Wolf (7.40) sono verificate per $0.238 \leq \alpha \leq 0.271$ o $0.623 \leq \alpha \leq 0.668$, ovvero in piccoli intorni dei punti di minimo. Sono stati presi $\sigma = 0.2$ e $\delta = 0.9$

per $\sigma \in (0, 1)$, $\rho \in [1/10, 1/2]$

```

porre  $\alpha = 1$ 
while  $f(\mathbf{x}^{(k)} + \alpha \mathbf{d}^{(k)}) > f(\mathbf{x}^{(k)}) + \sigma \alpha \mathbf{d}^{(k)^T} \nabla f(\mathbf{x}^{(k)})$ 
     $\alpha = \alpha \rho$ 
end
porre  $\alpha_k = \alpha$ 

```

(7.41)

La seconda condizione in (7.39) non viene controllata perché la tecnica di *backtracking* garantisce già di per sé che i passi non siano troppo piccoli.

Osservazione 7.1 La tecnica del backtracking è spesso combinata con l'approssimazione di f lungo la direzione $\mathbf{d}^{(k)}$ mediante un'interpolazione di tipo quadratico o cubico. Il passo α_k scelto fa sì che il nuovo punto $\mathbf{x}^{(k+1)}$ sia il punto di minimo dell'interpolatore di f lungo $\mathbf{d}^{(k)}$. L'algoritmo così ottenuto è noto come *quadratic o cubic line search*. Rimandiamo a [NW06, Cap. 3] per maggiori dettagli su questo approccio. ■

Il Programma **backtrack** 7.2 implementa la strategia (7.41). I parametri **fun** e **grad** sono *function handle* associati rispettivamente alle funzioni $f(\mathbf{x})$ e $\nabla f(\mathbf{x})$, **xk** e **dk** contengono rispettivamente il punto $\mathbf{x}^{(k)}$ e la direzione di discesa $\mathbf{d}^{(k)}$, mentre **sigma** e **rho** (opzionali) contengono i valori dei parametri σ e ρ . Se **sigma** e **rho** non vengono specificati, allora vengono inizializzati ai valori di *default* $\sigma = 10^{-4}$ e $\rho = 1/4$. La variabile **x** in output contiene il nuovo punto $\mathbf{x}^{(k+1)}$.

Programma 7.2. backtrack: la strategia di backtracking

```

function [x,alphak]= backtrack(fun,xk,gk,dk,varargin)
%BACKTRACK Metodo backtracking per line search.
%   [X,ALPHAK] = BACKTRACK(FUN,XK,GK,DK) calcola

```

```
% x_{k+1}=x_k+alpha_k d_k del metodo di discesa,
% in cui alpha_k e' costruito con la tecnica di
% backtracking, con sigma=1.e-4 e rho=1/4.
% [X,ALPHAK] = BACKTRACK(FUN,XK,GK,DK,SIGMA,RHO)
% permette di precisare i valori dei parametri
% sigma e rho. Tipicamente 1.e-4<sigma<0.1 e
% 1/10< rho <1/2. FUN e' un function handle
% associato alla funzione obiettivo.
% XK contiene l'elemento x_k della successione,
% GK il gradiente di FUN in XK e DK la direzione d_k.
if nargin==4
    sigma=1.e-4; rho=1/4;
else
    sigma=varargin{1}; rho=varargin{2};
end
alphamin=1.e-5; % valore minimo per il passo alpha
alphak = 1; fk = fun(xk);
k=0; x=xk+alphak*dk;
while fun(x)>fk+sigma*alphak*gk'*dk & alphak>alphamin
    alphak = alphak*rho;
    x = xk+alphak*dk; k = k+1;
end
```

Il Programma **descent** 7.3 implementa il metodo di discesa (7.33) con le direzioni (7.34)–(7.37) e passi α_k calcolati con la strategia di *backtracking*. Il test d’arresto è [DS96]:

$$\max_{1 \leq i \leq n} \left| \frac{[\nabla f(\mathbf{x}^{(k+1)})]_i \max\{|\mathbf{x}_i^{(k+1)}|, \text{typ}(\mathbf{x}_i)\}}{\max\{|f(\mathbf{x}^{(k+1)})|, \text{typ}(f(\mathbf{x}))\}} \right| \leq \varepsilon \quad (7.42)$$

per un certo $\varepsilon > 0$ assegnato. $\text{typ}(x)$ è un valore caratteristico dell’ordine di grandezza della variabile x e la sua presenza serve per evitare che il test fallisca quando \mathbf{x}^* o $f(\mathbf{x}^*)$ sono nulli.

I parametri **fun** e **grad** sono *function handle* associati rispettivamente alle funzioni $f(\mathbf{x})$ e $\nabla f(\mathbf{x})$, **x0** contiene il punto iniziale, **tol** la tolleranza per il test d’arresto e **kmax** il numero massimo di iterazioni consentite. La variabile **meth** seleziona la direzione di discesa: **meth=1** per le direzioni di Newton, **meth=2** per le direzioni quasi-Newton, **meth=3** per le direzioni del gradiente, **meth=41, 42, 43** per tre direzioni diverse del gradiente coniugato, rispettivamente CG-FR, CG-PR e CG-HS, come vedremo nella Sezione 7.5.5.

Programma 7.3. descent: il metodo di discesa

```
function [x,err,iter]= descent(fun,grad,x0,tol,kmax,%
                                meth,varargin)
% DESCENT Metodo di discesa per il calcolo di minimi
% [X,ERR,ITER]=DESCENT(FUN,GRAD,X0,TOL,KMAX,METH,HESS)
% approssima un punto di minimo della funzione FUN
% mediante il metodo di discesa con direzioni di
% Newton (METH=1), BFGS (METH=2), del gradiente
```

```
% (METH=3) o del gradiente coniugato con
% beta_k di Fletcher and Reeves (METH=41),
% beta_k di Polak and Ribiere (METH=42),
% beta_k di Hestenes and Stiefel (METH=43).
% Il passo e' costruito con la tecnica di back-
% tracking. FUN, GRAD ed HESS (quest'ultima usata
% solo se METH=1) sono function handle
% associati alla funzione obiettivo, al suo gradiente
% ed alla matrice Hessiana. Se METH=2, HESS e' una
% matrice approssimante l'Hessiana nel punto iniziale
% X0 della successione. TOL e' la tolleranza per il
% test d'arresto e KMAX e' il numero massimo di
% iterazioni. Si richiama le function backtrack.m
if nargin>6
if meth==1, hess=varargin{1};
elseif meth==2, H=varargin{1}; end
end
err=tol+1; k=0; xk=x0(:); gk=grad(xk); dk=-gk;
eps2=sqrt(eps);
while err>tol && k< kmax
if meth==1, H=hess(xk); dk=-H\gk; % Newton
elseif meth==2, dk=-H\gk; % BFGS
elseif meth==3, dk=-gk; % gradient
end
[xk1,alphak]= backtrack(fun,xk,gk,dk);
gk1=grad(xk1);
if meth==2 % BFGS update
yk=gk1-gk; sk=xk1-xk; yks=yk'*sk;
if yks> eps2*norm(sk)*norm(yk)
Hs=H*sk;
H=H+(yk*yk')/yks-(Hs*Hs')/(sk'*Hs);
end
elseif meth>=40 % CG upgrade
if meth == 41
betak=(gk1'*gk1)/(gk'*gk); % FR
elseif meth == 42
betak=(gk1'*(gk1-gk))/(gk'*gk); % PR
elseif meth == 43
betak=(gk1'*(gk1-gk))/(dk'*(gk1-gk)); % HS
end
dk=-gk1+betak*dk;
end
xk=xk1; gk=gk1; k=k+1; xkt=xk1;
for i=1:length(xk1); xkt(i)=max([abs(xk1(i)),1]); end
err=norm((gk1.*xkt)/max([abs(fun(xk1)),1]),inf);
end
xk=xk; iter=k;
if (k==kmax && err > tol)
fprintf(['descent si e'' arrestato senza aver ',...
'soddisfatto l''accuratezza richiesta, avendo\n',...
'raggiunto il massimo numero di iterazioni\n']);
end
```

Esempio 7.6 Riprendiamo la funzione $f(\mathbf{x})$ (7.32) per approssimarne il punto di minimo globale $(-0.6306, -0.7007)$. Utilizziamo il comando `diff` introdotto nella Sezione 1.6.3 per il calcolo simbolico del gradiente di f e della matrice hessiana H di f , quindi definiamo i *function handle* `f`, `grad.f`, `hess` associati

rispettivamente ad f , ∇f e H e richiamiamo il Programma 7.3 con le seguenti istruzioni:

```
x0=[0.5; -0.5]; tol=1.e-5; kmax=200;
meth=1; % discesa con direzioni di Newton
[x1,err1,k1]= descent(f,grad_f,x0,tol,kmax,meth,hess);
meth=2; hess=eye(2); % direzioni quasi-Newton
[x2,err2,k2]= descent(f,grad_f,x0,tol,kmax,meth,hess);
meth=3; % direzioni gradiente
[x3,err3,k3]= descent(f,grad_f,x0,tol,kmax,meth);
meth=42; % direzioni gradiente-coniugato con beta_PR
[x4,err4,k4]= descent(f,grad_f,x0,tol,kmax,meth);
```

Scegliendo $\mathbf{x}^{(0)} = (0.5, -0.5)$, tolleranza 10^{-5} per il test d'arresto ed un numero massimo di iterazioni pari a 200, otteniamo i seguenti risultati:

```
discesa Newton k=200,      x=[ 7.7015e-01, -6.3212e-01]
discesa quasi-Newton k=9,   x=[-6.3058e-01, -7.0075e-01]
discesa gradiente k=17,    x=[-6.3058e-01, -7.0075e-01]
discesa CG-PR k=17,        x=[-6.3060e-01, -7.0073e-01]
```

Il metodo di discesa con direzioni di Newton non è giunto a convergenza in quanto si sono generate direzioni $\mathbf{d}^{(k)}$ che non soddisfano le condizioni (5.67). ■

Nelle Sezioni successive descriviamo come calcolare le matrici Hessiane H_k ed i parametri β_k che intervengono nelle definizioni delle direzioni di discesa (7.35) e (7.37), rispettivamente. Inoltre accenniamo alle proprietà di convergenza dei vari metodi.

7.5.3 Il metodo di discesa con direzioni di Newton

Consideriamo una funzione $f \in C^2(\mathbb{R}^n)$ limitata inferiormente ed il metodo di discesa (7.33) con direzioni di discesa di Newton (7.34) e passi α_k che soddisfano le condizioni di Wolfe.

Supponiamo che per ogni $k \geq 0$ le matrici Hessiane $B_k = H(\mathbf{x}^{(k)})$ in (7.34), siano simmetriche e definite positive. Inoltre chiediamo che

$$\exists M > 0 : K(B_k) = \|B_k\| \|B_k^{-1}\| \leq M \quad \forall k \geq 0, \quad (7.43)$$

essendo $K(B_k)$ il numero di condizionamento spettrale di B_k (si veda (5.32).) Allora la successione $\mathbf{x}^{(k)}$ generata in (7.33) converge ad un punto stazionario \mathbf{x}^* di f . Inoltre, prendendo passi $\alpha_k = 1$ da un certo \bar{k} in poi, cioè quando si è sufficientemente vicini ad \mathbf{x}^* , l'ordine di convergenza è quadratico. Rimandiamo a [NW06, Teor. 3.2] per la dimostrazione di questo risultato.

Osservazione 7.2 Se tutte le matrici Hessiane generate durante le iterazioni risultano definite positive, il punto stazionario \mathbf{x}^* è necessariamente un punto di minimo. Se però per un certo k succede che $H(\mathbf{x}^{(k)})$ non sia definita positiva, la direzione (7.34) potrebbe non essere di discesa e le condizioni di Wolfe perderebbero di significato. Per ovviare a questo problema, si può introdurre

una opportuna matrice E_k e sostituire $H(\mathbf{x}^{(k)})$ con $B_k = H(\mathbf{x}^{(k)}) + E_k$ che risulti definita positiva, cosicché $\mathbf{d}^{(k)} = -B_k^{-1}\nabla f(\mathbf{x}^{(k)})$ sia una direzione di discesa. ■

Il metodo di discesa con direzioni di Newton è implementato nel Programma 7.3.

Esempio 7.7 Vogliamo calcolare il punto di minimo globale della funzione $f(\mathbf{x})$ (7.32) utilizzando il metodo di discesa (7.33), con direzioni di Newton (7.34) e passi α_k che soddisfano le condizioni di Wolfe. Fissiamo la tolleranza $\varepsilon = 10^{-5}$ per il test d'arresto e $\mathbf{x}^{(0)} = (-1, -1)$ come punto iniziale. Richiamando il Programma 7.3 con `meth=1` si ottiene convergenza in 4 iterazioni a $\mathbf{x}=[-0.63058; -0.70074]$. Se invece scegliamo $\mathbf{x}^{(0)} = (0.5, -0.5)$, il metodo va in stallo in quanto $H(\mathbf{x}^{(0)})$ non è definita positiva, si genera una direzione $\mathbf{d}^{(0)}$ non di discesa e la tecnica del backtracking non trova un $\alpha_0 > 0$ che soddisfi le condizioni di Wolfe. ■

7.5.4 Metodi di discesa con direzioni quasi-Newton

Consideriamo ora le direzioni (7.35) all'interno del metodo di discesa (7.33). Assegnata una matrice simmetrica e definita positiva H_0 , una tecnica ricorsiva molto diffusa per costruire le matrici H_k è quella basata sul cosiddetto *update di rango 1* secondo l'idea del metodo di Broyden (2.19) per la risoluzione dei sistemi non lineari. Alle matrici H_k si richiede di:

- soddisfare la condizione delle secanti

$$H_{k+1}(\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}) = \nabla f(\mathbf{x}^{(k+1)}) - \nabla f(\mathbf{x}^{(k)});$$

- essere simmetriche, in quanto la matrice $H(\mathbf{x})$ lo è;
- essere definite positive, per poter garantire che i vettori $\mathbf{d}^{(k)}$ siano direzioni di discesa;
- soddisfare

$$\lim_{k \rightarrow \infty} \frac{\|(H_k - H(\mathbf{x}^*))\mathbf{d}^{(k)}\|}{\|\mathbf{d}^{(k)}\|} = 0.$$

Questa condizione da un lato assicura che H_k sia una buona approssimazione dell'Hessiana $H(\mathbf{x}^*)$ lungo la direzione di discesa $\mathbf{d}^{(k)}$, dall'altro garantisce che la convergenza sia super-lineare (ovvero di ordine $p > 1$).

Fra le diverse strategie note per la costruzione di matrici che soddisfino questi requisiti, quella dovuta a Broyden, Fletcher, Goldfarb e Shanno (BFGS) prevede che le matrici H_k siano definite ricorsivamente come segue

$$H_{k+1} = H_k + \frac{\mathbf{y}^{(k)} \mathbf{y}^{(k)T}}{\mathbf{y}^{(k)T} \mathbf{s}^{(k)}} - \frac{H_k \mathbf{s}^{(k)} \mathbf{s}^{(k)T} H_k}{\mathbf{s}^{(k)T} H_k \mathbf{s}^{(k)}} \quad (7.44)$$

dove $\mathbf{s}^{(k)} = \mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}$ e $\mathbf{y}^{(k)} = \nabla f(\mathbf{x}^{(k+1)}) - \nabla f(\mathbf{x}^{(k)})$. Osserviamo che le matrici definite in (7.44) risultano simmetriche e definite positive a patto che si abbia $\mathbf{y}^{(k)T} \mathbf{s}^{(k)} > 0$, condizione che è garantita se i passi α_k soddisfano le condizioni di Wolfe (sia (7.39) che (7.40)) [DS96].

Il corrispondente metodo BFGS (implementato nel Programma 7.3) può riassumersi come segue: assegnato $\mathbf{x}^{(0)} \in \mathbb{R}^n$ e data un'opportuna matrice simmetrica e definita positiva $H_0 \in \mathbb{R}^{n \times n}$ che approssimi $H(\mathbf{x}^{(0)})$, per $k = 0, 1, \dots$, fino a convergenza

risolvere	$H_k \mathbf{d}^{(k)} = -\nabla f(\mathbf{x}^{(k)})$	(7.45)
calcolare	α_k che soddisfi le condizioni di Wolfe	
porre	$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k \mathbf{d}^{(k)}$	
	$\mathbf{s}^{(k)} = \mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}$	
	$\mathbf{y}^{(k)} = \nabla f(\mathbf{x}^{(k+1)}) - \nabla f(\mathbf{x}^{(k)})$	
calcolare	H_{k+1} con la formula (7.44)	

Se $f \in C^2(\mathbb{R}^n)$ ed è limitata inferiormente, nelle ipotesi che le matrici H_k siano definite positive per $k \geq 0$ ed abbiano un numero di condizionamento uniformemente limitato (si veda (7.43)), il metodo BFGS converge ad un punto di minimo con ordine di convergenza (super-lineare) $p \in (1, 2)$ (si vedano ad esempio [DS96, NW06]).

Esempio 7.8 Utilizziamo il metodo BFGS (7.45) per calcolare il punto di minimo della funzione $f(\mathbf{x})$ (7.32). Scegliamo $\varepsilon = 10^{-5}$ per il test d'arresto e H_0 uguale alla matrice identità (che ovviamente è simmetrica e definita positiva). Il Programma 7.3 con `meth=2` e `hess=eye(2)` converge a $\mathbf{x} = [-0.63058; -0.70074]$ in 6 iterazioni se $\mathbf{x}^{(0)} = (-1, -1)$ e in 9 iterazioni se $\mathbf{x}^{(0)} = (0.5, -0.5)$. ■

Osservazione 7.3 Come nel caso del metodo di Broyden (2.19), il costo computazionale di ordine $\mathcal{O}(n^3)$ per il calcolo di $\mathbf{d}^{(k)} = -H_k^{-1} \nabla f(\mathbf{x}^{(k)})$ può essere ridotto ad un ordine $\mathcal{O}(n^2)$, utilizzando ricorsivamente fattorizzazioni QR delle matrici H_k (si veda [GM72]).

Una strategia alternativa è basata sull'uso della matrice inversa \tilde{H}_k di H_k sia in (7.44) sia in (7.45) e può essere implementata con un ordine di $\mathcal{O}(n^2)$ operazioni per ogni passo, tuttavia in pratica è poco utilizzata perché meno stabile di quella standard (7.44). ■

fminunc Il metodo BFGS è implementato nella *function fminunc* del *toolbox optimization* di MATLAB e nella *function bfgsmin* del *package optim* di Octave.

Con i seguenti comandi:

```
fun=@(x) 100*(x(2)-x(1)^2)^2+(1-x(1))^2; x0=[1.2;-1];
options = optimset('LargeScale','off');
[x,fval,exitflag,output]=fminunc(fun,x0,options)
```

la *function fminunc* di MATLAB calcola il punto di minimo della funzione di Rosenbrock utilizzando il metodo BFGS. (Osserviamo che bisogna inizializzare l'opzione 'LargeScale' al valore 'off'.) I parametri di output hanno lo stesso significato di quelli della funzione fminsearch descritta nell'Esempio 7.3. La convergenza è raggiunta in 24 iterazioni con 93 valutazioni della funzione f , avendo fissato la tolleranza per il test d'arresto pari a $\varepsilon = 10^{-6}$.

Osserviamo che il gradiente di f non è stato specificato nella precedente chiamata, esso è infatti approssimato in fminunc con metodi alle differenze finite (si veda la Sezione 9.3). Tuttavia, qualora sia nota l'espressione esatta del gradiente di f , essa può essere passata in input alla *function fminunc* come segue:

```
fun=@(x) 100*(x(2)-x(1)^2)^2+(1-x(1))^2; x0=[1.2;-1];
grad_fun=@(x)[-400*(x(2)-x(1)^2)*x(1)-2*(1-x(1));
200*(x(2)-x(1)^2)];
options = optimset('LargeScale','off','GradObj','on');
[x,fval,exitflag,output]=fminunc({fun,grad_fun},...
x0,options)
```

Si osservi la variazione nel settaggio della variabile options rispetto all'esempio precedente. La convergenza ora è raggiunta in 25 iterazioni con 32 valutazioni della funzione f . ■

MAT||OCT **Osservazione 7.4** Il comando fminunc di Octave implementa un metodo di tipo *trust region* come descriveremo nella Sezione 7.6. ■

7.5.5 Metodi di discesa del gradiente e del gradiente coniugato

Consideriamo dapprima il metodo di discesa (7.33) con direzioni del gradiente (7.36). Come abbiamo già osservato queste sono sempre direzioni di discesa. Inoltre, se $f \in C^2(\mathbb{R}^n)$ è limitata inferiormente e i passi α_k soddisfano le condizioni di Wolfe, esso converge con velocità lineare ad un punto stazionario [NW06]. Si veda il Programma 7.3 per la sua implementazione.

Esempio 7.9 Consideriamo ancora la funzione (7.32). Fissiamo la tolleranza $\varepsilon = 10^{-5}$ per il test d'arresto e richiamiamo il Programma 7.3 scegliendo meth=3 (che corrisponde alle direzioni del gradiente). Ponendo $\mathbf{x}^{(0)} =$

$(-0.9, -0.9)$, $\mathbf{x}^{(0)} = (-1, -1)$, $\mathbf{x}^{(0)} = (0.5, -0.5)$, il metodo converge al punto di minimo globale $\mathbf{x} = [-0.63058; -0.70074]$, rispettivamente in 11, 12 e 17 iterazioni. Scegliendo invece il punto iniziale $\mathbf{x}^{(0)} = (0.9, 0.9)$, che è più vicino al punto di minimo locale $\mathbf{x}^* = (0.8094399, 0.7097390)$, il metodo converge a quest'ultimo in 21 iterazioni. ■

Consideriamo ora le direzioni (7.37) del gradiente coniugato. In letteratura sono note diverse scelte dei parametri β_k (si vedano ad esempio [SY06, NW06]), tra le quali citiamo le seguenti:

1. *Fletcher–Reeves (1964)*

$$\beta_k^{FR} = \frac{\|\nabla f(\mathbf{x}^{(k)})\|^2}{\|\nabla f(\mathbf{x}^{(k-1)})\|^2} \quad (7.46)$$

2. *Polak–Ribièvre (1969) (anche note come Polak–Ribièvre–Polyak)*

$$\beta_k^{PR} = \frac{\nabla f(\mathbf{x}^{(k)})^T (\nabla f(\mathbf{x}^{(k)}) - \nabla f(\mathbf{x}^{(k-1)}))}{\|\nabla f(\mathbf{x}^{(k-1)})\|^2} \quad (7.47)$$

3. *Hestenes–Stiefel (1952)*

$$\beta_k^{HS} = \frac{\nabla f(\mathbf{x}^{(k)})^T (\nabla f(\mathbf{x}^{(k)}) - \nabla f(\mathbf{x}^{(k-1)}))}{\mathbf{d}^{(k-1)^T} (\nabla f(\mathbf{x}^{(k)}) - \nabla f(\mathbf{x}^{(k-1)}))} \quad (7.48)$$

Nel caso in cui la funzione f sia quadratica e strettamente convessa, tutte queste scelte coincidono con (5.80). Per semplicità denotiamo con la sigla FR, PR e HS le direzioni (7.37) associate ai parametri β_k^{PR} , β_k^{HS} e β_k^{FR} , rispettivamente.

Le seguenti condizioni sono sufficienti a garantire che le direzioni FR convergano ad un punto stazionario [NW06, SY06]: $f \in C^1(\mathbb{R}^n)$, il suo gradiente è lipschitziano, il punto iniziale $\mathbf{x}^{(0)}$ è tale che l'insieme $A = \{\mathbf{x} : f(\mathbf{x}) \leq f(\mathbf{x}^{(0)})\}$ sia limitato, i passi α_k soddisfano le condizioni forti di Wolfe (7.40) con $0 < \sigma < \delta < 1/2$.

Sotto le stesse ipotesi su f e $\mathbf{x}^{(0)}$, ma a condizione che i passi α_k soddisfino una variante delle condizioni forti di Wolfe e che i parametri β_k^{PR} siano sostituiti da $\beta_k^{PR+} = \max\{-\beta_k^{PR}, 0\}$, anche il metodo del gradiente coniugato con β_k^{PR+} converge ad un punto stazionario. Le stesse conclusioni valgono per le direzioni HS. Per una analisi più dettagliata di questi metodi rimandiamo, ad esempio, a [Noc92, NW06, SY06].

Il metodo del gradiente coniugato con direzioni FR, PR e HS, e passi α_k calcolati mediante la tecnica del *backtracking* è implementato nel Programma 7.3.

Esempio 7.10 Concludiamo il confronto fra le varie direzioni del gradiente coniugato sulla funzione $f(\mathbf{x})$ (7.32). Richiamiamo il Programma 7.3 ponendo `meth=41` per le direzioni FR, `meth=42` per le direzioni PR e `meth=43` per le direzioni HS. Il numero di iterazioni necessarie a soddisfare il test d'arresto con tolleranza $\varepsilon = 10^{-5}$ e con diversi valori del punto iniziale $\mathbf{x}^{(0)}$ è riportato nella seguente tabella.

Direzioni	$\mathbf{x}^{(0)}$		
	$(-1, -1)$	$(1, 1)$	$(0.5, -0.5)$
FR	20	12	> 400
PR	21	28	17
HS	23	40	28

Per $\mathbf{x}^{(0)} = (-1, -1)$ e $\mathbf{x}^{(0)} = (0.5, -0.5)$ i metodi convergono al punto di minimo globale $\mathbf{x}=[-0.63058; -0.70074]$ (FR non converge quando $\mathbf{x}^{(0)} = (0.5, -0.5)$), mentre prendendo $\mathbf{x}^{(0)} = (1, 1)$ tutti i metodi convergono al punto di minimo locale $\mathbf{x}=[0.8094; 0.7097]$. ■

Concludiamo la sezione con alcune importanti osservazioni.

Osservando i risultati della precedente tabella e la Figura 7.9, deduciamo che nei casi analizzati le direzioni PR e HS sono da preferire alle direzioni FR. Queste ultime possono generare passi molto piccoli che portano ad una convergenza molto lenta o addirittura ad una stagnazione; in tal caso l'algoritmo può essere fatto ripartire utilizzando come direzione di discesa quella del gradiente $\mathbf{d}^{(k)} = -\nabla f(\mathbf{x}^{(k)})$.

Quando i passi α_k sono calcolati in maniera esatta (come descritto all'inizio della Sezione 7.5.1) si può dimostrare che la velocità di convergenza del gradiente coniugato è solo lineare, quella di Newton quadratiche e quella del metodo quasi-Newton super lineare. Tuttavia, il metodo del gradiente coniugato è molto semplice da implementare: esso non richiede la matrice Hessiana (né una sua approssimazione) e ad ogni iterazione sono richieste solo una valutazione di f e del suo gradiente.

Il metodo del gradiente coniugato è sicuramente preferibile nel caso di problemi di ottimizzazione di grande dimensione, mentre i metodi di Newton e quasi-Newton sono in genere più efficienti per problemi di piccole dimensioni.



Si vedano gli Esercizi 7.4–7.6.

7.6 Metodi di tipo *trust region*

Mentre i metodi di tipo *line search* determinano (al generico passo k) prima una direzione di discesa $\mathbf{d}^{(k)}$ e poi, in funzione di questa, stabiliscono il passo α_k , i metodi di tipo *trust region* scelgono direzione e passo simultaneamente costruendo una palla centrata nel punto $\mathbf{x}^{(k)}$ (la cosiddetta *trust region*) di raggio δ_k un modello quadratico \tilde{f}_k della funzione

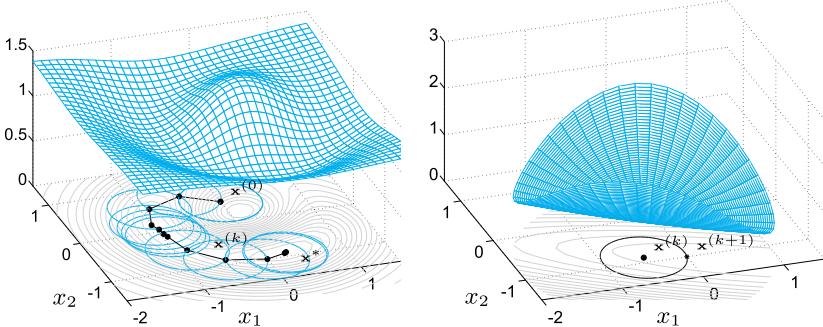


Figura 7.12. La funzione $f(\mathbf{x})$, la successione $\mathbf{x}^{(k)}$ costruita con il metodo *trust region* (a sinistra) e il modello quadrattico \tilde{f}_k (7.49) al passo $k = 8$ (a destra)

obiettivo f e quindi definiscono il nuovo passo $\mathbf{x}^{(k+1)}$ come il punto di minimo di \tilde{f}_k ristretto alla *trust region*, come vediamo in Figura 7.12.

Più precisamente, alla generica iterazione k :

1. si parte da un valore $\delta_k > 0$ “di fiducia” del passo;
2. si costruisce un modello quadrattico \tilde{f}_k di f con uno sviluppo di Taylor di ordine 2 centrato in $\mathbf{x}^{(k)}$:

$$\tilde{f}_k(\mathbf{s}) = f(\mathbf{x}^{(k)}) + \mathbf{s}^T \nabla f(\mathbf{x}^{(k)}) + \frac{1}{2} \mathbf{s}^T \mathbf{H}_k \mathbf{s} \quad \forall \mathbf{s} \in \mathbb{R}^n, \quad (7.49)$$

dove \mathbf{H}_k è l’Hessiana di f in $\mathbf{x}^{(k)}$ o una sua approssimazione;

3. si calcola

$$\mathbf{s}^{(k)} = \underset{\mathbf{s} \in \mathbb{R}^n : \| \mathbf{s} \| \leq \delta_k}{\operatorname{argmin}} \tilde{f}_k(\mathbf{s}); \quad (7.50)$$

4. si calcola

$$\rho_k = \frac{f(\mathbf{x}^{(k)} + \mathbf{s}^{(k)}) - f(\mathbf{x}^{(k)})}{\tilde{f}_k(\mathbf{s}^{(k)}) - \tilde{f}_k(\mathbf{0})} \quad (7.51)$$

e:

- a) se ρ_k è prossimo a uno, accettiamo $\mathbf{s}^{(k)}$, definiamo $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathbf{s}^{(k)}$ e passiamo all’iterazione successiva. Nel caso in cui il punto di minimo di \tilde{f}_k giaccia sul bordo della *trust region*, estendiamo quest’ultima prima di procedere con l’iterazione successiva;
- b) se ρ_k è un numero negativo oppure un numero positivo piccolo (molto più piccolo di uno), allora riduciamo la *trust region* e cerchiamo un nuovo $\mathbf{s}^{(k)}$ risolvendo nuovamente il problema (7.50);
- c) se ρ_k è molto più grande di uno, accettiamo $\mathbf{s}^{(k)}$, definiamo $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathbf{s}^{(k)}$, lasciamo inalterata la *trust region* e passiamo all’iterazione successiva.

La costruzione di H_k segue le seguenti regole:

se sono disponibili le derivate seconde di f poniamo H_k uguale all'Hessiana e, nel caso quest'ultima non fosse definita positiva, poniamo H_k uguale ad una variante dell'Hessiana come descritto nell'Osservazione 7.2;

altrimenti H_k può essere costruita ricorsivamente come abbiamo descritto per i metodi di discesa con le direzioni quasi-Newton (si veda la Sezione 7.5.4).

Vediamo ora come risolvere il problema di minimizzazione (7.50), passo centrale del metodo.

Se la matrice H_k è definita positiva, allora \tilde{f}_k è una funzione quadratica convessa. Il suo unico punto di minimo assoluto e unico punto stazionario è la soluzione dell'equazione $\nabla \tilde{f}_k(\mathbf{x}^{(k)}) = \nabla f(\mathbf{x}^{(k)}) + H_k \mathbf{s} = \mathbf{0}$, cioè

$$\mathbf{s} = -H_k^{-1} \nabla f(\mathbf{x}^{(k)}). \quad (7.52)$$

Se $\|\mathbf{s}\| \leq \delta_k$, allora la soluzione $\mathbf{s}^{(k)}$ del problema (7.50) coincide con \mathbf{s} calcolato in (7.52) ed appartiene alla *trust region*; altrimenti vuol dire che il punto di minimo assoluto di \tilde{f}_k giace all'esterno della *trust region* e la soluzione di (7.50) giace sul bordo della *trust region*, cioè

$$\mathbf{s}^{(k)} = \underset{\mathbf{s} \in \mathbb{R}^n: \|\mathbf{s}\| = \delta_k}{\operatorname{argmin}} \tilde{f}_k(\mathbf{s}). \quad (7.53)$$

Il problema (7.53) è un problema di minimo vincolato con vincoli di uguaglianza. Esso si può risolvere con il metodo dei moltiplicatori di Lagrange (si veda la Sezione 7.8 e in particolare (7.75)) cercando, tra i punti stazionari della funzione Lagrangiana

$$\mathcal{L}_k(\mathbf{s}, \lambda) = \tilde{f}_k(\mathbf{s}) + \frac{1}{2} \lambda (\mathbf{s}^T \mathbf{s} - \delta_k^2),$$

quello che sia di minimo per \tilde{f}_k sotto il vincolo $\|\mathbf{s}\| = \delta_k$. Dobbiamo quindi trovare un vettore $\mathbf{s}^{(k)}$ ed uno scalare $\lambda^{(k)} > 0$ soluzioni del sistema

$$\begin{aligned} (H_k + \lambda^{(k)} I) \mathbf{s}^{(k)} &= -\nabla f(\mathbf{x}^{(k)}), \\ \|\mathbf{s}^{(k)}\| - \delta_k &= 0, \end{aligned} \quad (7.54)$$

con $(H_k + \lambda^{(k)} I)$ semidefinita positiva.

Il sistema (7.54) è non lineare e può essere ricondotto ad un'unica equazione scalare

$$\varphi(\lambda^{(k)}) = \frac{1}{\|\mathbf{s}^{(k)}(\lambda^{(k)})\|} - \frac{1}{\delta_k} = 0, \quad (7.55)$$

dopo aver esplicitato

$$\mathbf{s}^{(k)} = \mathbf{s}^{(k)}(\lambda^{(k)}) = -(H_k + \lambda^{(k)}I)^{-1} \nabla f(\mathbf{x}^{(k)}) \quad (7.56)$$

dall'equazione (7.54)₁.

L'equazione (7.55) è equivalente a (7.54)₂, ma è più semplice da risolvere numericamente, infatti bastano poche iterazioni (tipicamente al massimo 3) del metodo di Newton per ottenerne la soluzione.

Osserviamo che il moltiplicatore $\lambda^{(k)}$ esaurisce la sua funzione nel momento in cui viene calcolato $\mathbf{s}^{(k)}$ e poi viene scartato, per questo motivo da ora in poi lo indicheremo semplicemente con λ .

Ponendo $\mathbf{g}^{(k)} = \nabla f(\mathbf{x}^{(k)})$ e $\mathbf{s}^{(k)}(\lambda) = -(H_k + \lambda I)^{-1} \mathbf{g}^{(k)}$, abbiamo

$$\frac{d\mathbf{s}^{(k)}(\lambda)}{d\lambda} = (H_k + \lambda I)^{-1} \mathbf{s}^{(k)}(\lambda),$$

$$\varphi'(\lambda) = -\frac{(\mathbf{s}^{(k)}(\lambda))^T (H_k + \lambda I)^{-1} \mathbf{s}^{(k)}(\lambda)}{\|\mathbf{s}^{(k)}(\lambda)\|^3}$$

e

$$\frac{\varphi(\lambda)}{\varphi'(\lambda)} = \frac{\|\mathbf{s}^{(k)}(\lambda)\| - \delta_k}{\delta_k} \frac{\|\mathbf{s}^{(k)}(\lambda)\|^2}{(\mathbf{s}^{(k)}(\lambda))^T (H_k + \lambda I)^{-1} \mathbf{s}^{(k)}(\lambda)}.$$

Per calcolare $\mathbf{s}^{(k)}$ applichiamo quindi il metodo di Newton all'equazione (7.55) come segue:

```

dato  $\lambda_0$ 
for  $\ell = 0, \dots, 2$ 
    calcolare  $R : R^T R = H_k + \lambda_\ell I$ 
    calcolare  $\mathbf{s}_\ell : R^T R \mathbf{s}_\ell = -\mathbf{g}^{(k)}$ 
    calcolare  $\mathbf{q}_\ell : R^T \mathbf{q}_\ell = \mathbf{s}_\ell$ 
    if  $\ell < 2$ 
        porre  $\lambda_{\ell+1} = \lambda_\ell + \left( \frac{\|\mathbf{s}_\ell\|}{\|\mathbf{q}_\ell\|} \right)^2 \frac{\|\mathbf{s}_\ell\| - \delta_k}{\delta_k}$ 
    endif
endfor
porre  $\mathbf{s}^{(k)} = \mathbf{s}_2$ 

```

(7.57)

I vettori \mathbf{s}_ℓ in (7.57) sono calcolati utilizzando la fattorizzazione di Cholesky (5.18) della matrice $(H_k + \lambda_\ell I)$ a patto che questa matrice sia definita positiva. (Si noti che $(H_k + \lambda_\ell I)$ è simmetrica grazie alla definizione di H_k e quindi i suoi autovalori sono tutti reali.) Se H_k non è definita positiva, denotando con β l'autovalore negativo di H_k di modulo massimo, una scelta possibile per λ_0 è $\lambda_0 = 2|\beta|$.

L'algoritmo per risolvere il problema (7.50) è: posto $\mathbf{g}^{(k)} = \nabla f(\mathbf{x}^{(k)})$ e dato $\delta_k > 0$:

```

    risolvere  $H_k \mathbf{s} = -\mathbf{g}^{(k)}$ 
    if  $\|\mathbf{s}\| \leq \delta_k$  e  $H_k$  è definita positiva
        porre  $\mathbf{s}^{(k)} = \mathbf{s}$ 
    else
        calcolare  $\beta =$  l'autovalore negativo di  $H_k$           (7.58)
        di modulo massimo
        porre  $\lambda_0^{(k)} = 2|\beta|$ 
        calcolare  $\mathbf{s}^{(k)}$  con (7.57)
    endif

```

Per concludere, scriviamo una semplice forma dell'algoritmo *trust region* per calcolare la soluzione del problema di minimo (7.1) [CL96a, CL96b].

Siano $\mathbf{x}^{(0)}$ il punto iniziale, $\hat{\delta} > 0$ il valore massimo accettabile del raggio della *trust region*, $\delta_0 \in (0, \hat{\delta})$ il raggio della *trust region* al passo $k = 0$, η_1 , η_2 , γ_1 e γ_2 quattro parametri reali che servono per aggiornare la *trust region* tali che $0 < \eta_1 < \eta_2 < 1$ e $0 < \gamma_1 < 1 < \gamma_2$ (scelte standard sono $\eta_1 = \gamma_1 = 1/4$, $\eta_2 = 3/4$, $\gamma_2 = 2$) e $\mu \in [0, \eta_1]$ un parametro reale che serve per valutare l'accettabilità della soluzione (tipicamente $\mu = 1/10$).

Per $k = 0, 1, \dots$, fino a convergenza:

```

    calcolare  $f(\mathbf{x}^{(k)})$ ,  $\nabla f(\mathbf{x}^{(k)})$  e  $H_k$ ,
    risolvere  $\min_{\|\mathbf{s}\|_2 \leq \delta_k} \tilde{f}_k(\mathbf{s})$  con (7.58)
    calcolare  $\rho_k$  utilizzando (7.51),
    if  $\rho_k > \mu$ 
        porre  $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathbf{s}^{(k)}$ 
    else
        porre  $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)}$           (7.59)
    endif
    if  $\rho_k < \eta_1$ 
        porre  $\delta_{k+1} = \gamma_1 \delta_k$ 
    elseif  $\eta_1 \leq \rho_k \leq \eta_2$ 
        porre  $\delta_{k+1} = \delta_k$ 
    elseif  $\rho_k > \eta_2$  e  $\|\mathbf{s}^{(k)}\| = \delta_k$ 
        porre  $\delta_{k+1} = \min\{\gamma_2 \delta_k, \hat{\delta}\}$ 
    endif

```

Osservazione 7.5 (Risoluzione approssimata del problema (7.50)) Il problema (7.50) può essere risolto in maniera approssimata, senza tuttavia inficiare le proprietà di convergenza del metodo *trust region*. Una strategia possibile consiste nel cercare la soluzione $\mathbf{s}^{(k)}$ non nello spazio \mathbb{R}^n bensì in un sottospazio $\mathcal{S}_k \subset \mathbb{R}^n$ di dimensione due. Più precisamente si cerca la soluzione di

$$\mathbf{s}^{(k)} = \underset{\mathbf{s} \in \mathcal{S}_k: \|\mathbf{s}\| \leq \delta_k}{\operatorname{argmin}} \tilde{f}_k(\mathbf{s}). \quad (7.60)$$

Se H_k è definita positiva, definiamo $\mathcal{S}_k = \operatorname{span}\{\nabla f(\mathbf{x}^{(k)}), H_k^{-1}\nabla f(\mathbf{x}^{(k)})\}$; altrimenti calcoliamo l'autovalore negativo β di H_k con modulo massimo, scegliamo $\alpha \in (-\beta, -2\beta]$ e definiamo $\mathcal{S}_k = \operatorname{span}\{\nabla f(\mathbf{x}^{(k)}), (H_k + \alpha I)^{-1}\nabla f(\mathbf{x}^{(k)})\}$. La scelta di questi sottospazi è motivata dalla ricerca del cosiddetto *punto di Cauchy*, ovvero il punto di minimo della funzione \tilde{f}_k lungo la direzione del gradiente $\nabla f(\mathbf{x}^{(k)})$ e interno alla *trust region*. Dal punto di vista computazionale la fase più pesante nel risolvere (7.60) è la fattorizzazione di H_k o di $H_k + \alpha I$ ed il calcolo dell'autovalore β . Tuttavia il costo computazionale globale per risolvere (7.60) è fortemente ridotto rispetto a quello richiesto per risolvere (7.50) in maniera esatta. ■

L'algoritmo (7.59) è implementato nel Programma 7.4. I parametri `fun`, `grad`, `x0`, `tol`, `kmax` assumono il medesimo significato assegnato nel Programma *descent* 7.3. Inoltre `delta0` contiene il raggio della *trust region* iniziale, `meth` seleziona il metodo di costruzione delle matrici H_k : se `meth=1`, `hess` contiene il *function handle* dell'Hessiana di f e H_k è l'Hessiana esatta. Se `meth` è diverso da uno, non c'è bisogno di passare la variabile di input `hess`; in tal caso la matrice H_k è una approssimazione di rango uno dell'Hessiana ed è calcolata come in (7.44).

Programma 7.4. trustregion: il metodo trust region

```
function [x,err,iter]= trustregion(fun,grad,x0, ...
    delta0,tol,kmax,meth,hess)
%TRUSTREGION Metodo trust region per la minimizzazione
% [X,ERR,ITER]=TRUSTREGION(FUN,GRAD,X0,TOL,KMAX, ...
% METH,HESS) approssima il punto di minimo della
% funzione FUN con gradiente GRAD mediante il metodo
% trust region. Se METH=1 si utilizza l'Hessiana di f
% passata in HESS, altrimenti si costruiscono appross-
% imazioni dell'Hessiana con update di rango 1, come
% in BFGS, HESS in tal caso non e' richiesta.
% FUN e GRAD (ed HESS) sono function handle
% associati alla funzione obiettivo, al suo gradiente
% (ed alla matrice Hessiana). X0 e' il punto iniziale
% della successione. TOL e' la tolleranza per il test
% d'arresto e KMAX e' il numero massimo di iterazioni.
delta=delta0; err=tol+1; k=0; mu=0.1;
eta1=0.25; eta2=0.75; gamma1=0.25; gamma2=2; deltam=5;
xk=x0(:); gk=grad(xk); eps2=sqrt(eps);
% definizione delle matrici Hk
if meth==1 Hk=hess(xk); else Hk=eye(length(xk)); end
while err>tol && k< kmax
% risoluzione del problema (7.54)
[s]=trustone(Hk,gk,delta);
```

```

rho=(fun(xk+s)-fun(xk))/(s'*gk+0.5*s'*Hk*s);
if rho> mu, xk1=xk+s; else, xk1=xk; end
if rho<eta1
    delta=gamma1*delta;
elseif rho> eta2 & abs(norm(s)-delta)<sqrt(eps)
    delta=min([gamma2*delta,deltam]);
end
gk1=grad(xk1);
err=norm((gk1.*xk1)/max([abs(fun(xk1)),1]),inf);
% aggiornamento matrici Hk
if meth==1 % Newton
    xk=xk1; gk=gk1; Hk=hess(xk);
else % quasiNewton
    gk1=grad(xk1); yk=gk1-gk; sk=xk1-xk;
    yks=yk'*sk;
    if yks> eps2*norm(sk)*norm(yk)
        Hs=Hk*sk;
        Hk=Hk+(yk*yk')/yks-(Hs*Hs')/(sk'*Hs);
    end
    xk=xk1; gk=gk1;
end
k=k+1;
end
x=xk; iter=k;
if (k==kmax && err > tol)
fprintf(['trustregion si e' arrestato senza aver ',...
'soddisfatto l''accuratezza richiesta, avendo\n',...
'raggiunto il massimo numero di iterazioni\n']);
end
end

function [s]=trustone(Hk,gk,delta)
s=-Hk\gk; d = eigs(Hk,1,'sa');
if norm(s)>delta | d<0
lambda=abs(2*d); I=eye(size(Hk));
for l=1:3
R=chol(Hk+lambda*I);
s=-R \ (R'\gk); q=R'\s;
lambda=lambda+(s'*s)/(q'*q)*(norm(s)-delta)/delta;
if lambda< -d
    lambda=abs(lambda*2);
end
end
end
end

```

Esempio 7.11 Calcoliamo il punto di minimo della funzione $f(x_1, x_2) = (x_1 + 2x_2 + 2x_1x_2 - 5x_1^2 - 5x_2^2)/(5e^{x_1^2+x_2^2}) + 7/5$ con il Programma 7.4. Come possiamo osservare dalla Figura 7.12 f presenta un punto di massimo locale, un punto di sella e due punti di minimo locale: \mathbf{x}_{m1} in prossimità di $(-1.0, 0.2)$ e \mathbf{x}_{m2} in prossimità di $(0.3, -0.9)$. \mathbf{x}_{m2} è anche punto di minimo globale. Scegliamo $\mathbf{x}^{(0)} = (0, 0.5)$ come punto iniziale e calcoliamo le matrici H_k ricorsivamente in accordo con (7.44). Richiamando il Programma 7.4 con le seguenti istruzioni:

```

fun=@(x)7/5+(x(1)+2*x(2)+2*x(1)*x(2)-5*x(1)^2-...
5*x(2)^2)/(5*exp(x(1)^2+x(2)^2));

```

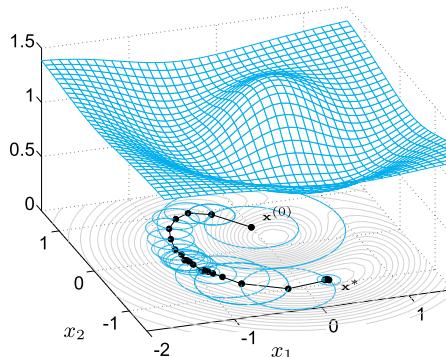


Figura 7.13. Le iterate del metodo *trust region* con H_k costruite secondo (7.44)

```
grad_fun=@(x) [(1+2*x(2)-10*x(1)-2*x(1)*(x(1)+2*x(2)+...
2*x(1)*x(2)-5*x(1)^2-5*x(2)^2))/(5*exp(x(1)^2+x(2)^2));
(2+2*x(1)-10*x(2)-2*x(2)*(x(1)+2*x(2)+...
2*x(1)*x(2)-5*x(1)^2-5*x(2)^2))/(5*exp(x(1)^2+x(2)^2))];
```

```
delta0=0.5; tol=1.e-5; kmax=100; meth=2; x0=[0;0.5];
[x,err,iter]= trustregion(fun,grad_fun,x0,delta0, ...
tol,kmax,meth)
```

la convergenza al punto $(0.27849, -0.89695)$ è raggiunta in 24 iterazioni.

Utilizzando invece `meth=1`, per cui le matrici H_k sono le Hessiane esatte, la convergenza è raggiunta in 12 iterazioni. In entrambi i casi è osservato un rallentamento della convergenza quando la soluzione $\mathbf{x}^{(k)}$ è prossima al punto di minimo locale \mathbf{x}_{m1} , come possiamo osservare in Figura 7.12, a sinistra, e in Figura 7.13. In particolare, in Figura 7.12, a sinistra, è riportata la successione delle iterate $\mathbf{x}^{(k)}$ costruite con Hessiane esatte H_k , mentre in Figura 7.13 è mostrata la successione delle iterate ottenute utilizzando le Hessiane approssimate, costruite come in (7.44). ■

Per l'analisi di convergenza del metodo *trust region* rimandiamo a [NW06, Sez. 4.2] e [SSB85].

Osservazione 7.6 Il comando `fminunc` in Octave implementa un algoritmo di tipo *trust region* in cui le matrici H_k sono costruite con la formula ricorsiva (7.44). MAT||OCT

La *function* `fminunc` di MATLAB implementa un algoritmo di tipo *trust region* a patto di inizializzare i campi `'LargeScale'` e `'GradObj'` entrambi pari ad `'on'` con il comando `optimset`. In tal caso, il primo argomento in input è una cella (si veda la Sezione 1.5) contenente i *function handle* della funzione obiettivo (`fun`) e del suo gradiente (`grad_fun`). Ad esempio, con le seguenti istruzioni:

```
fun=@(x) 100*(x(2)-x(1)^2)^2+(1-x(1))^2; x0=[1.2;-1];
grad_fun=@(x) [-400*(x(2)-x(1)^2)*x(1)-2*(1-x(1));
```

```

200*(x(2)-x(1)^2)];
options = optimset('LargeScale','on','GradObj','on');
[x,fval,exitflag,output]=fminunc({fun,grad_fun},...
x0,options)

```

otteniamo convergenza al punto di minimo della funzione di Rosenbrock in 8 iterazioni e sono richieste in tutto 9 valutazioni funzionali. ■



Si veda l'Esercizio 7.7.

7.7 Il metodo dei minimi quadrati non lineari

Nella Sezione 3.6 abbiamo introdotto il metodo dei minimi quadrati lineari per approssimare una funzione o un insieme discreto di dati mediante un polinomio o un'altra funzione \tilde{f} che dipende linearmente da un insieme di coefficienti incogniti a_j , $j = 1, \dots, m$.

Quando tale dipendenza non è lineare, diciamo che il problema ai minimi quadrati è non lineare. Un esempio è dato dal Problema 7.2 in cui si vogliono determinare m funzioni gaussiane g_k (per $k = 1, \dots, m$) che dipendono non linearmente dalle variabili incognite a_k e σ_k .

In termini astratti, siano $r_i : \mathbb{R}^m \rightarrow \mathbb{R}$ (con $i = 1, \dots, n$) n funzioni non lineari nella variabile \mathbf{x} e sia $\mathbf{R}(\mathbf{x}) = (r_1(\mathbf{x}), \dots, r_n(\mathbf{x}))^T$. Il nostro obiettivo è risolvere numericamente il seguente problema di minimizzazione

$$\min_{\mathbf{x} \in \mathbb{R}^m} f(\mathbf{x}) \quad \text{con } f(\mathbf{x}) = \frac{1}{2} \|\mathbf{R}(\mathbf{x})\|^2 = \frac{1}{2} \sum_{i=1}^n r_i^2(\mathbf{x}). \quad (7.61)$$

Poiché le funzioni r_i sono non lineari, la funzione f potrebbe non essere convessa su tutto \mathbb{R}^m ed avere molti punti stazionari.

Tutti i metodi considerati finora, cioè il metodo di Newton (7.31), i metodi di discesa (7.33) e quelli *trust region* (7.59), possono essere utilizzati per risolvere (7.61).

Grazie alla forma particolare di f , il suo gradiente e l'Hessiana possono essere scritti in termini della matrice Jacobiana $J_{\mathbf{R}}(\mathbf{x}) \in \mathbb{R}^{n \times m}$ e delle derivate prima e seconda di \mathbf{R} , come segue:

$$\begin{aligned}
\nabla f(\mathbf{x}) &= J_{\mathbf{R}}(\mathbf{x})^T \mathbf{R}(\mathbf{x}), \\
H(\mathbf{x}) &= J_{\mathbf{R}}(\mathbf{x})^T J_{\mathbf{R}}(\mathbf{x}) + S(\mathbf{x}),
\end{aligned} \tag{7.62}$$

con $S_{\ell j}(\mathbf{x}) = \sum_{i=1}^n \frac{\partial^2 r_i}{\partial x_\ell \partial x_j}(\mathbf{x}) r_i(\mathbf{x}), \quad \ell, j = 1, \dots, m.$

Il calcolo esatto dell'Hessiana può risultare molto oneroso quando m e n sono grandi, specialmente per la valutazione dei coefficienti della matrice $S(\mathbf{x})$. D'altro canto, in molte situazioni la matrice $J_{\mathbf{R}}(\mathbf{x})^T J_{\mathbf{R}}(\mathbf{x})$ è

predominante rispetto a $S(\mathbf{x})$, cosicché quest'ultima può essere approssimata, o addirittura trascurata durante la costruzione di $H(\mathbf{x})$. Questo è quanto viene fatto nei due metodi che andiamo a presentare nelle prossime due sezioni.

7.7.1 Il metodo di Gauss-Newton

Questo metodo è una variante del metodo di Newton (7.31) in cui la matrice Hessiana $H(\mathbf{x})$ (7.62)₂ è approssimata con $J_R(\mathbf{x}^{(k)})^T J_R(\mathbf{x}^{(k)})$, trascurando la matrice $S(\mathbf{x})$.

Esso si formula come segue: dato $\mathbf{x}^{(0)} \in \mathbb{R}^m$, per $k = 0, 1, \dots$, fino a convergenza

$$\begin{aligned} & \text{risolvere } [J_R(\mathbf{x}^{(k)})^T J_R(\mathbf{x}^{(k)})] \delta \mathbf{x}^{(k)} = -J_R(\mathbf{x}^{(k)})^T R(\mathbf{x}^{(k)}) \\ & \text{porre } \mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \delta \mathbf{x}^{(k)} \end{aligned} \quad (7.63)$$

Il sistema (7.63)₁ è un esempio di sistema delle equazioni normali (5.43). Se $J_R(\mathbf{x}^{(k)})$ ha rango pieno, (7.63)₁ può essere risolto utilizzando la fattorizzazione QR o la decomposizione in valori singolari della matrice $J_R(\mathbf{x}^{(k)})$, come abbiamo visto nella Sezione 5.7, mentre se $J_R(\mathbf{x}^{(k)})$ non è a rango pieno, il sistema lineare (7.63)₁ può avere infinite soluzioni, nel qual caso il metodo di Gauss-Newton può stagnare, divergere, o convergere ad un punto non stazionario.

Si dimostra (si veda l'Esercizio 7.8) che trascurare $S(\mathbf{x}^{(k)})$ nell'Hessiana equivale ad approssimare la funzione $R(\mathbf{x})$ con il suo sviluppo di Taylor centrato in $\mathbf{x}^{(k)}$ e troncato al primo ordine

$$\tilde{R}_k(\mathbf{x}) = R(\mathbf{x}^{(k)}) + J_R(\mathbf{x}^{(k)})(\mathbf{x} - \mathbf{x}^{(k)}). \quad (7.64)$$

La convergenza del metodo di Gauss-Newton non è sempre garantita, essa dipende sia dalle proprietà di f sia dalla scelta del punto iniziale.

Il seguente risultato è dimostrato in [DS96]: se \mathbf{x}^* è un punto stazionario per f e $J_R(\mathbf{x})$ ha rango pieno in un opportuno intorno di \mathbf{x}^* , allora:

1. se $S(\mathbf{x}^*) = 0$, che è il caso in cui $R(\mathbf{x})$ è lineare o $R(\mathbf{x}^*) = \mathbf{0}$, il metodo di Gauss-Newton è localmente (quadraticamente) convergente (di fatto coincide con il metodo di Newton);
2. se $\|S(\mathbf{x}^*)\|_2$ è piccolo rispetto al minimo autovalore (positivo) di $J_R(\mathbf{x}^*)^T J_R(\mathbf{x}^*)$, allora il metodo di Gauss-Newton converge linearmente. Questo è, ad esempio, il caso in cui $R(\mathbf{x})$ è non lineare con una non linearità debole o $\|R(\mathbf{x}^*)\|^2$ (detto *residuo*) è piccolo;
3. se $\|S(\mathbf{x}^*)\|_2$ è grande rispetto al minimo autovalore (positivo) di $J_R(\mathbf{x}^*)^T J_R(\mathbf{x}^*)$, il metodo di Gauss-Newton potrebbe non convergere anche se $\mathbf{x}^{(0)}$ è molto prossimo a \mathbf{x}^* . Questo succede se $R(\mathbf{x})$ è fortemente non lineare o se il residuo $\|R(\mathbf{x}^*)\|^2$ è grande.

Osservazione 7.7 Le tecniche di tipo *line-search* possono essere utilizzate in combinazione con il metodo di Gauss-Newton, sostituendo (7.63)₂ con $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k \delta \mathbf{x}^{(k)}$, dove i passi α_k sono calcolati come descritto nella Sezione 7.5.1. Se $J_{\mathbf{R}}(\mathbf{x}^{(k)})$ ha rango massimo, la matrice $J_{\mathbf{R}}(\mathbf{x}^{(k)})^T J_{\mathbf{R}}(\mathbf{x}^{(k)})$ risulta simmetrica e definita positiva ed il vettore $\delta \mathbf{x}^{(k)}$ è una direzione di discesa per f (si veda l'Esercizio 7.9). In tal caso, sotto opportune ipotesi su f , si ottiene un metodo globalmente convergente, detto *damped Gauss-Newton*. ■

Il metodo di Gauss-Newton è implementato nel Programma 7.5; **r** e **jr** sono *function handle* associati rispettivamente alle funzioni $\mathbf{R}(\mathbf{x})$ ed al suo Jacobiano $J_{\mathbf{R}}(\mathbf{x})$, **x0** è il punto iniziale, mentre **tol** e **kmax** contengono la tolleranza per il test d'arresto ed il numero massimo di iterazioni consentite. In output, **x** contiene la soluzione calcolata, **err** è l'errore stimato all'ultima iterazione ed **iter** il numero di iterazioni richieste per raggiungere la convergenza.

Programma 7.5. gaussnewton: il metodo di Gauss-Newton

```
function [x,err,iter]= gaussnewton(r,jr,x0,tol,%
    kmax,varargin)
%GAUSSNEWTON Minimi quadrati non lineari
% [X,ERR,ITER]=GAUSSNEWTON(R,JR,X0,TOL,KMAX)
% Risolve il problema dei minimi quadrati non lineari
% mediante il metodo di Gauss-Newton.
% R e JR sono function handle associati alla funzione
% R ed allo Jacobiano di R, in cui la prima variabile
% in input e' X, di seguito possono essere dati
% parametri opzionali. X0 e' il punto iniziale
% della successione. TOL e' la tolleranza per il
% test d'arresto e KMAX e' il numero massimo di
% iterazioni.
err=tol+1; k=0; xk=x0(:);
rk=r(xk,varargin{:}); jrk=jr(xk,varargin{:});
while err>tol && k< kmax
[Q,R]=qr(jrk,0); dk=-R \ (Q'*rk);
xk1=xk+dk;
rk1=r(xk1,varargin{:});
jrk1=jr(xk1,varargin{:});
k=k+1; err=norm(xk1-xk);
xk=xk1; rk=rk1; jrk=jrk1;
end
x=xk; iter=k;
if (k==kmax && err > tol)
fprintf(['GaussNewton si e'' arrestato senza aver ',...
    'soddisfatto l''accuratezza richiesta, avendo\n',...
    'raggiunto il massimo numero di iterazioni\n']);
end
```

Esempio 7.12 Consideriamo il Problema 7.2, (7.5) è un caso particolare di (7.61) con

$$r_i(\mathbf{x}) = g(t_i; \mathbf{a}, \boldsymbol{\sigma}) - y_i = \sum_{k=1}^m g_k(t_i; a_k, \sigma_k) - y_i,$$

avendo definito $\mathbf{x} = [\mathbf{a}, \boldsymbol{\sigma}]$. Ricordando la definizione (7.3), calcoliamo

$$\frac{\partial r_i}{\partial a_k} = g_k(t_i; a_k, \sigma_k) \frac{t_i - a_k}{\sigma_k^2}, \quad \frac{\partial r_i}{\partial \sigma_k} = g_k(t_i; a_k, \sigma_k) \left[\frac{(t_i - a_k)^2}{\sigma_k^3} - \frac{1}{2\sigma_k} \right]$$

e li salviamo nella matrice $J_{\mathbf{R}}(\mathbf{x})$.

Le *user-defined function* per costruire $\mathbf{R}(\mathbf{x})$ e $J_{\mathbf{R}}(\mathbf{x})$ sono rispettivamente `gmmr` e `gmmjr`:

```
function [R]=gmmr(x,t,y)
gaussk=@(t,ak,sigmak)[exp(-((t-ak)/(sqrt(2)*sigmak))...
.^2)/(sqrt(pi*2)*sigmak)];
x=x(:); m=length(x)/2;
a=x(1:m); sigma=x(m+1:end);
n=length(t); R=zeros(n,1);
for k=1:m
R=R+gaussk(t,a(k),sigma(k));
end
R=R-y;

function [Jr]=gmmjr(x,t,y)
gaussk=@(t,ak,sigmak)[exp(-((t-ak)/(sqrt(2)*sigmak))...
.^2)/(sqrt(pi*2)*sigmak)];
x=x(:); m=length(x)/2;
a=x(1:m); sigma=x(m+1:end);
n=length(t); Jr=zeros(n,m*2);
gk=zeros(n,m);
for k=1:m
gk(:,k)=gaussk(t,a(k),sigma(k));
Jr(:,k)=(gk(:,k).*(t-a(k))/sigma(k)^2)';
Jr(:,k+m)=(gk(:,k).*((t-a(k)).^2/...
sigma(k)^3-1/(2*sigma(k))))';
end
```

Entrambe le *function* `gmmr` e `gmmjr` richiedono in input i vettori t e y contenenti gli n punti campionati (t_i, y_i) con $i = 1, \dots, n$, $0 \leq t_i \leq 10$. Per generarli, sommiamo 5 funzioni Gaussiane (7.3) con $\mathbf{a} = [2.3, 3.25, 4.82, 5.3, 6.6]$ e $\boldsymbol{\sigma} = [0.2, 0.34, 0.50, 0.23, 0.39]$ e aggiungiamo un disturbo *random*. I comandi `MATEOCT` sono:

```
a=[2.3,3.25,4.82,5.3,6.6]; m=length(a);
sigma=[0.2,0.34,0.50,0.23,0.39];
gaussk=@(t,ak,s...)
exp(-((t-ak)/(sqrt(2)*s)).^2)/(sqrt(pi*2)*s);
n=2000; t=linspace(0,10,n)'; y=zeros(n,1);
for k=1:m, y=y+gaussk(t,a(k),sigma(k)); end
y=y+0.05*randn(n,1);
```

Quindi richiamiamo il Programma 7.5 con le seguenti istruzioni:

```
x0=[2,3,4,5,6,0.3,0.3,0.6,0.3,0.3];
tol=1.e-5; kmax=200;
[x,err,iter]=gaussnewton(@gmmr,@gmmjr,x0,tol,kmax,t,y)
xa=x(1:m); xsigma=x(m+1:end);
```

I vettori \mathbf{xa} e \mathbf{xsigma} contengono le approssimazioni delle incognite a_k e σ_k cercate (per $k = 1, \dots, m$). Con i comandi:

```
plot(t,y,'c.');
```

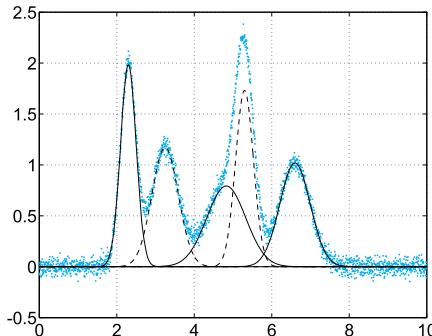


Figura 7.14. I dati (in azzurro) e la soluzione (in nero) dell’Esempio 7.12

```
for k=1:m
    y1=gaussk(t,xa(k),xsigma(k));
    plot(t,y1,'k');
end
```

possiamo visualizzare i punti campionati (t_i, y_i) (in blue) rappresentanti il segnale e le 5 funzioni Gaussiane (7.3) (linee nere) costruite a partire dalla soluzione numerica calcolata (si veda la Figura 7.14).

Il metodo di Gauss-Newton converge in 22 iterazioni, avendo arrestato il metodo al più piccolo k per cui $\|\mathbf{R}(\mathbf{x}^{(k+1)}) - \mathbf{R}(\mathbf{x}^{(k)})\| < \varepsilon$ con $\varepsilon = 10^{-5}$.

Questo è un esempio in cui il residuo $\|\mathbf{R}(\mathbf{x}^*)\|^2 = 2.2992$ è grande, essendo \mathbf{x}^* la soluzione esatta. Cambiando di poco il dato iniziale, ad esempio modificando l’ultima componente del vettore $\mathbf{x}^{(0)}$ da 0.3 a 0.5, il metodo non converge più, evidenziando che la scelta del dato iniziale $\mathbf{x}^{(0)}$ è cruciale per il buon funzionamento di questo algoritmo. ■

7.7.2 Il metodo di Levenberg-Marquardt

Questo è un metodo di tipo *trust region* utilizzato per risolvere il problema di minimo (7.61). Vista la natura particolare della funzione f definita in (7.61), il modello quadrattico \tilde{f}_k introdotto in (7.49) risulta

$$\tilde{f}_k(\mathbf{s}) = \frac{1}{2} \|\mathbf{R}(\mathbf{x}^{(k)}) + \mathbf{J}_{\mathbf{R}}(\mathbf{x}^{(k)})\mathbf{s}\|^2. \quad (7.65)$$

Si noti che \tilde{f}_k (7.65) è un’approssimazione quadrattica di f in un intorno di $\mathbf{x}^{(k)}$, ottenuta approssimando $\mathbf{R}(\mathbf{x})$ con il suo modello lineare $\tilde{\mathbf{R}}_k(\mathbf{x})$ (7.64) (si veda l’Esercizio 7.11).

Sebbene $J_{\mathbf{R}}(\mathbf{x})$ non abbia rango massimo, questo è un buon metodo per risolvere problemi di minimo che presentano una forte non linearità o un residuo $\|\mathbf{R}(\mathbf{x}^*)\|^2$ grande in corrispondenza di un punto di minimo locale \mathbf{x}^* .

Poiché l’approssimazione utilizzata per la matrice Hessiana coincide con quella utilizzata nel metodo di Gauss-Newton, i due metodi esibiscono le stesse proprietà di convergenza. In particolare, se le iterazioni di

Levenberg-Marquardt convergono, la velocità di convergenza è quadratica se il residuo si annulla nel punto di minimo locale, mentre è lineare negli altri casi.



Si vedano gli Esercizi [7.8–7.11](#).

Riassumendo

1. Per calcolare il minimo di una funzione f , i metodi *derivative free* utilizzano solo valori della funzione f . Sono molto robusti in pratica, anche se difficilmente è possibile studiarne le proprietà teoriche;
2. i metodi di discesa sfruttano la conoscenza delle derivate della funzione e calcolano ad ogni iterazione una direzione di discesa ed un passo di avanzamento, basandosi su strategie di tipo *line search*;
3. i metodi di discesa con direzioni di Newton abbinate a strategie *line search* risultano globalmente convergenti se le matrici $H(\mathbf{x}^{(k)})$ sono definite positive. Essi esibiscono ordine di convergenza quadratico in prossimità del punto di minimo e sono adatti per problemi di piccole e medie dimensioni;
4. i metodi di discesa con direzioni quasi-Newton utilizzano opportune approssimazioni H_k delle matrici Hessiane ad ogni iterazione. Quando sono associati a strategie di tipo *line search*, sono metodi globalmente convergenti con ordine di convergenza super-lineare, a patto che le matrici H_k siano definite positive. Essi sono adatti per problemi di piccola e media taglia;
5. i metodi di discesa con direzioni di tipo gradiente coniugato, abbinati a strategie *line search* sono globalmente convergenti con ordine di convergenza lineare. Sono particolarmente indicati per problemi di grandi dimensioni;
6. le strategie di tipo *trust region* sono relativamente recenti e meno diffuse delle precedenti. Costruiscono un modello quadratico locale della funzione obiettivo e ne cercano il minimo in una palla n -dimensionale.

7.8 Ottimizzazione vincolata

Senza alcuna pretesa di essere esaustivi, in questa sezione presentiamo due semplici strategie per la risoluzione di problemi di minimo vincolato: un metodo di penalizzazione per problemi con vincoli di uguaglianza e disuguaglianza e il metodo della Lagrangiana aumentata per problemi con soli vincoli di uguaglianza.

Questi due metodi si prestano per la risoluzione di semplici problemi e costituiscono le basi di algoritmi più complessi e robusti per i quali rimandiamo a testi più specifici quali [[NW06](#), [SY06](#), [BDF⁺10](#)].

Un problema di ottimizzazione vincolata si formula come segue: consideriamo il problema (7.2) dove il dominio Ω può essere definito come

$$\Omega = \{\mathbf{x} \in \mathbb{R}^n : h_i(\mathbf{x}) = 0, \text{ per } i = 1, \dots, p\}, \quad (7.66)$$

dove $h_i : \mathbb{R}^n \rightarrow \mathbb{R}$ per $i = 1, \dots, p$, sono funzioni assegnate, oppure come

$$\Omega = \{\mathbf{x} \in \mathbb{R}^n : g_j(\mathbf{x}) \geq 0, \text{ per } j = 1, \dots, q\} \quad (7.67)$$

date $g_j : \mathbb{R}^n \rightarrow \mathbb{R}$ per $j = 1, \dots, q$. In casi più generale, il dominio Ω può essere definito da vincoli sia di uguaglianza sia di disuguaglianza, ovvero

$$\Omega = \{\mathbf{x} \in \mathbb{R}^n : h_i(\mathbf{x}) = 0, \text{ per } i = 1, \dots, p, g_j(\mathbf{x}) \geq 0, \text{ per } j = 1, \dots, q\}. \quad (7.68)$$

Le tre diverse definizioni (7.66), (7.67) e (7.68) del dominio Ω sono casi particolari della definizione più generale

$$\Omega = \{\mathbf{x} \in \mathbb{R}^n : h_i(\mathbf{x}) = 0, \text{ per } i \in \mathcal{I}_h, g_j(\mathbf{x}) \geq 0, \text{ per } j \in \mathcal{I}_g\},$$

dove \mathcal{I}_h e \mathcal{I}_g sono opportuni insiemi di numeri naturali, con la convenzione che $\mathcal{I}_h = \emptyset$ in (7.67) e $\mathcal{I}_g = \emptyset$ in (7.66).

Il problema (7.2) può essere quindi scritto come

$$\begin{aligned} & \min_{\mathbf{x} \in \mathbb{R}^n} f(\mathbf{x}) \\ & \text{soggetto ai vincoli} \\ & h_i(\mathbf{x}) = 0 \quad \forall i \in \mathcal{I}_h, \\ & g_j(\mathbf{x}) \geq 0 \quad \forall j \in \mathcal{I}_g \end{aligned} \quad (7.69)$$

Supporremo in tutta la sezione che le funzioni f , h_i e g_j siano di classe C^1 su \mathbb{R}^n .

Un punto $\mathbf{x} \in \Omega$ è detto *ammissibile* (cioè esso soddisfa tutti i vincoli dati) ed Ω è l'insieme dei punti ammissibili.

Un punto $\mathbf{x}^* \in \Omega \subset \mathbb{R}^n$ è detto di *minimo globale* per il problema (7.2) se

$$f(\mathbf{x}^*) \leq f(\mathbf{x}) \quad \forall \mathbf{x} \in \Omega,$$

mentre \mathbf{x}^* è detto di *minimo locale* per il problema (7.2) se esiste una palla $B_r(\mathbf{x}^*) \subset \mathbb{R}^n$ di centro \mathbf{x}^* e raggio $r > 0$ tale che

$$f(\mathbf{x}^*) \leq f(\mathbf{x}) \quad \forall \mathbf{x} \in B_r(\mathbf{x}^*) \cap \Omega.$$

Diciamo che un *vincolo* è *attivo* in $\mathbf{x} \in \Omega$ se esso è soddisfatto con l'uguaglianza nel punto $\mathbf{x} \in \Omega$. In accordo con questa definizione, tutti i vincoli attivi nel punto \mathbf{x} sono tutti i vincoli di uguaglianza h_i e quelli di tipo disuguaglianza g_j tali che $g_j(\mathbf{x}) = 0$.

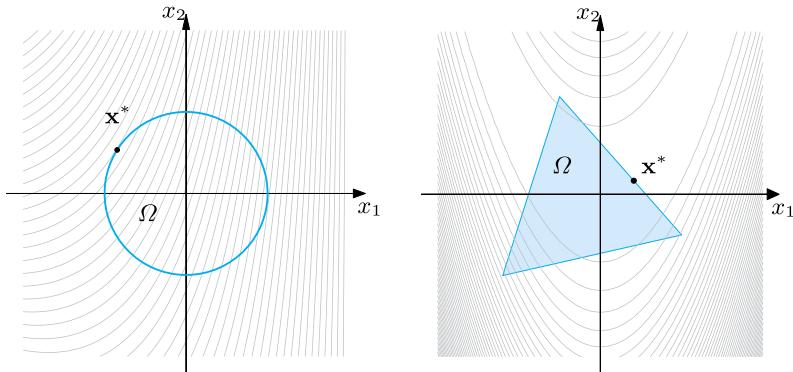


Figura 7.15. Linee di livello della funzione obiettivo, insieme Ω definito dai vincoli e punto di minimo globale \mathbf{x}^* per f vincolato ad Ω . La figura di sinistra si riferisce al Problema 1 (7.70), quella di destra al Problema 2 (7.71)

Esempio 7.13 Si considerino i seguenti problemi di ottimizzazione vincolata:
Problema 1:

$$\begin{aligned} \min_{\mathbf{x} \in \mathbb{R}^2} f(\mathbf{x}), \quad & \text{con } f(\mathbf{x}) = \frac{3}{5}x_1^2 + \frac{1}{2}x_1x_2 - x_2 + 3x_1 \\ & \text{con il vincolo} \\ & h_1(\mathbf{x}) = x_1^2 + x_2^2 - 1 = 0, \end{aligned} \tag{7.70}$$

Problema 2:

$$\begin{aligned} \min_{\mathbf{x} \in \mathbb{R}^2} f(\mathbf{x}), \quad & \text{con } f(\mathbf{x}) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 \\ & \text{con i vincoli} \\ & g_1(\mathbf{x}) = -34x_1 - 30x_2 + 19 \geq 0 \\ & g_2(\mathbf{x}) = 10x_1 - 5x_2 + 11 \geq 0 \\ & g_3(\mathbf{x}) = 3x_1 + 22x_2 + 8 \geq 0. \end{aligned} \tag{7.71}$$

In Figura 7.15, a sinistra, sono riportate le linee di livello delle due funzioni obiettivo ed i relativi insiemi Ω . Si osservi che Ω è una curva per il Problema 1, mentre è una regione chiusa e limitata di \mathbb{R}^2 per il Problema 2. In entrambi i problemi, il numero dei vincoli attivi nel punto \mathbf{x}^* di minimo assoluto per f è pari a uno. ■

Nel caso in cui Ω sia un insieme non vuoto, chiuso e limitato e f sia continua su Ω , il teorema di Weierstrass garantisce che f ammette massimo e minimo in Ω , e quindi esiste soluzione del problema (7.69).

Ricordiamo che una funzione $f : \Omega \subseteq \mathbb{R}^n \rightarrow \mathbb{R}$ è *fortemente convessa* in Ω se esiste $\rho > 0$ tale che per ogni $\mathbf{x}, \mathbf{y} \in \Omega$ e per ogni $\alpha \in [0, 1]$ vale

$$f(\alpha\mathbf{x} + (1 - \alpha)\mathbf{y}) \leq \alpha f(\mathbf{x}) + (1 - \alpha)f(\mathbf{y}) - \alpha(1 - \alpha)\rho\|\mathbf{x} - \mathbf{y}\|^2. \tag{7.72}$$

Nel caso in cui $\rho = 0$ ricadiamo nella definizione di funzione convessa (7.11).

Proposizione 7.2 (Condizioni di ottimalità) *Sia $\Omega \subset \mathbb{R}^n$ un insieme convesso, $\mathbf{x}^* \in \Omega$ e sia $r > 0$ t.c. $f \in C^1(B_r(\mathbf{x}^*))$. Se \mathbf{x}^* è un punto di minimo locale per (7.2) allora*

$$\nabla f(\mathbf{x}^*)^T(\mathbf{x} - \mathbf{x}^*) \geq 0 \quad \forall \mathbf{x} \in \Omega. \quad (7.73)$$

Inoltre, se f è convessa in Ω e (7.73) è soddisfatta, allora \mathbf{x}^* è un punto di minimo globale per il problema (7.2).

Infine, se Ω è anche chiuso e f è anche fortemente convessa, allora esiste un unico punto di minimo per il problema (7.2).

Siano $\lambda_i, \mu_j \in \mathbb{R}$, $\boldsymbol{\lambda} = (\lambda_i)$ (per $i \in \mathcal{I}_h$) e $\boldsymbol{\mu} = (\mu_j)$ (per $j \in \mathcal{I}_g$). Definiamo la funzione Lagrangiana

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\mathbf{x}) - \sum_{i \in \mathcal{I}_h} \lambda_i h_i(\mathbf{x}) - \sum_{j \in \mathcal{I}_g} \mu_j g_j(\mathbf{x}). \quad (7.74)$$

$\boldsymbol{\lambda}$ e $\boldsymbol{\mu}$ sono i cosiddetti *moltiplicatori di Lagrange* associati rispettivamente ai vincoli di uguaglianza e disuguaglianza. Si dice che \mathbf{x}^* è un punto di *Karush–Kuhn–Tucker* (KKT) per \mathcal{L} se esistono $\boldsymbol{\lambda}^*$ e $\boldsymbol{\mu}^*$ tali che la terna $(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ soddisfa le seguenti condizioni (dette di Karush–Kuhn–Tucker):

$$\begin{aligned} \nabla_{\mathbf{x}} \mathcal{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) &= \nabla f(\mathbf{x}^*) - \sum_{i \in \mathcal{I}_h} \lambda_i^* \nabla h_i(\mathbf{x}^*) - \sum_{j \in \mathcal{I}_g} \mu_j^* \nabla g_j(\mathbf{x}^*) = \mathbf{0} \\ h_i(\mathbf{x}^*) &= 0 \quad \forall i \in \mathcal{I}_h \\ g_j(\mathbf{x}^*) &\geq 0 \quad \forall j \in \mathcal{I}_g \\ \mu_j^* &\geq 0 \quad \forall j \in \mathcal{I}_g \\ \mu_j^* g_j(\mathbf{x}^*) &= 0 \quad \forall j \in \mathcal{I}_g \end{aligned}$$

Dato un punto \mathbf{x} , diciamo che i vincoli soddisfano la condizione LICQ (*linear independence constraint qualification*) in \mathbf{x} se i vettori gradienti $\nabla h_i(\mathbf{x})$ e $\nabla g_j(\mathbf{x})$ associati ai soli vincoli attivi in \mathbf{x} formano un sistema di vettori linearmente indipendenti.

Si ha il seguente risultato [NW06, Teor. 12.1].

Teorema 7.1 (Condizioni necessarie KKT del primo ordine) *Se \mathbf{x}^* è punto di minimo locale per il problema (7.69), se f , h_i e g_j sono di classe $C^1(\Omega)$, se i vincoli soddisfano la condizione LICQ in \mathbf{x}^* , allora esistono $\boldsymbol{\lambda}^*$ e $\boldsymbol{\mu}^*$ tali che $(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ è un punto KKT.*

Grazie a questo teorema, i punti di minimo locale per il problema (7.69) vanno cercati tra i punti KKT ed i punti in cui non sono soddisfatte le condizioni LICQ.

Osserviamo che quando l'insieme \mathcal{I}_g è vuoto, (cioè sono presenti solo vincoli di uguaglianza), la funzione Lagrangiana assume la forma $\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) - \sum_{i \in \mathcal{I}_h} \lambda_i h_i(\mathbf{x})$ e le condizioni KKT si riducono alle condizioni necessarie classiche (dette *condizioni di Lagrange*)

$$\begin{aligned}\nabla_{\mathbf{x}} \mathcal{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*) &= \nabla f(\mathbf{x}^*) - \sum_{i \in \mathcal{I}_h} \lambda_i^* \nabla h_i(\mathbf{x}^*) = \mathbf{0} \\ h_i(\mathbf{x}^*) &= 0 \quad \forall i \in \mathcal{I}_h\end{aligned}\tag{7.75}$$

Condizioni sufficienti affinché un punto KKT sia di minimo per f vincolato ad Ω richiedono la conoscenza della matrice Hessiana della Lagrangiana, oppure ipotesi di stretta convessità sia per f che per i vincoli [NW06, SY06].

In linea generale un problema di ottimizzazione vincolata può essere riscritto come un problema senza vincoli attraverso la formulazione penalizzata o quella della Lagrangiana aumentata, come vedremo nelle prossime due sezioni.

Osservazione 7.8 Se f ammette un punto di minimo \mathbf{x}^* su cui non agiscono vincoli attivi, allora la Lagrangiana si riduce alla funzione obiettivo f in quanto $\mathcal{I}_h = \emptyset$ e $\mu_j^* = 0$ per ogni $j \in \mathcal{I}_g$, grazie alle condizioni KKT. Di conseguenza si ricade in un problema di minimo non vincolato per la cui risoluzione si possono utilizzare i metodi visti nelle Sezioni precedenti. ■

Un caso notevole di ottimizzazione vincolata è quello della cosiddetta *Programmazione Quadratica*: f è una funzione quadratica, i vincoli sono lineari ed il problema (7.69) si può scrivere come

$$\begin{aligned}\min_{\mathbf{x} \in \mathbb{R}^n} f(\mathbf{x}), \quad f(\mathbf{x}) &= \frac{1}{2} \mathbf{x}^T \mathbf{A} \mathbf{x} + \mathbf{x}^T \mathbf{b} \\ \text{soggetto ai vincoli} \quad \mathbf{C} \mathbf{x} - \mathbf{d} &= \mathbf{0}, \quad \mathbf{D} \mathbf{x} - \mathbf{e} \geq \mathbf{0}\end{aligned}\tag{7.76}$$

con $\mathbf{A} \in \mathbb{R}^{n \times n}$, $\mathbf{b} \in \mathbb{R}^n$, $\mathbf{C} \in \mathbb{R}^{p \times n}$, $\mathbf{d} \in \mathbb{R}^p$, $\mathbf{D} \in \mathbb{R}^{q \times n}$, $\mathbf{e} \in \mathbb{R}^q$, p, q opportuni interi positivi e dove la scrittura $\mathbf{v} \geq \mathbf{0}$ significa $v_i \geq 0$ per ogni i .

Consideriamo il caso particolare in cui siano presenti solo vincoli di uguaglianza (rimandando a [Bom10, NW06] per una presentazione accurata della Programmazione Quadratica). La forma matriciale delle condizioni di Lagrange (7.75) è

$$\begin{bmatrix} \mathbf{A} & -\mathbf{C}^T \\ \mathbf{C} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} -\mathbf{b} \\ \mathbf{d} \end{bmatrix}, \tag{7.77}$$

essendo $\boldsymbol{\lambda} \in \mathbb{R}^p$ il vettore dei moltiplicatori di Lagrange.

Se la matrice A è simmetrica e definita positiva sul nucleo della matrice C, cioè

$$\mathbf{y}^T \mathbf{A} \mathbf{y} > 0 \quad \forall \mathbf{y} \in \ker(C) = \{\mathbf{z} : C\mathbf{z} = \mathbf{0}\}, \quad \mathbf{y} \neq \mathbf{0},$$

ed inoltre la matrice C ha rango massimo, il sistema (7.77) ammette un'unica soluzione e quindi esiste un unico punto di minimo globale per la funzione obiettivo definita in (7.76).

Un problema di programmazione quadratica può essere quindi affrontato risolvendo il sistema lineare (7.77) con i metodi visti nel Capitolo 5. Generalmente la matrice M = [A, -C^T; C, 0] risulta non definita, ovvero presenta autovalori con segno discorde e, qualora si vogliano utilizzare metodi iterativi, è preferibile richiamare metodi di Krylov come ad esempio il GMRES o il Bi-CGSTab (si veda la Sezione 5.10).

Esempio 7.14 Vogliamo risolvere il Problema 7.4. La funzione obiettivo (il rischio) è definita in (7.7) ed è di tipo quadratico, mentre i vincoli sono

$$h_1(\mathbf{x}) = 1.2x_1 + 1.6x_2 + 2.8x_3 = 2.0, \quad h_2(\mathbf{x}) = x_1 + x_2 + x_3 = 1. \quad (7.78)$$

Il primo esprime la richiesta che il rendimento finale sia del 2%, mentre il secondo assicura che la somma delle 3 quantità ripartite sui 3 fondi corrisponda all'intero capitale. Dobbiamo quindi risolvere un problema di Programmazione Quadratica che riscriviamo nella forma (7.77). Verifichiamo che siano soddisfatte le ipotesi sulle matrici A e C affinché esso ammetta un unico punto di minimo. Abbiamo

$$A = \begin{bmatrix} 0.08 & 0.1 & 0 \\ 0.1 & 0.5 & 0.208 \\ 0 & 0.208 & 1.28 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$

$$C = \begin{bmatrix} 1.2 & 1.6 & 2.8 \\ 1 & 1 & 1 \end{bmatrix}, \quad \mathbf{d} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}.$$

La matrice C ha rango 2 (massimo) e la dimensione del suo nucleo $\ker(C) = \{\mathbf{z} = \alpha[1, -3, 2]^T, \alpha \in \mathbb{R}\}$ è pari a 1.

Essendo la matrice A simmetrica, resta da verificare che sia definita positiva su $\ker(C)$, cioè che $\mathbf{z}^T \mathbf{A} \mathbf{z} > 0$ per ogni $\mathbf{z} = \alpha[1, -3, 2]^T$, con $\alpha \neq 0$. Si ha $\mathbf{z}^T \mathbf{A} \mathbf{z} = \alpha^2[1, -3, 2]^T \mathbf{A} [1, -3, 2] = 6.6040\alpha^2 > 0$. Quindi costruiamo la matrice M = [A, -C^T; C, 0] ed il termine noto $\mathbf{f} = [-\mathbf{b}; \mathbf{d}]$ e risolviamo il sistema lineare (7.77) con le seguenti istruzioni:

```
A=[0.08 0.1 0; 0.1 0.5 0.208; 0 0.208 1.28];
b=[0;0;0];
C=[1.2 1.6 2.8;1,1,1];
d=[2.;1];
M=[A -C';
C, zeros(2)];
f=[-b;d];
x1=M\f
```

ottenendo la soluzione

```
x1 =
0.4272
0.0970
```

0.4757
0.3658
-0.3951

Le prime 3 componenti del vettore \mathbf{x}_1 corrispondono alle 3 frazioni di capitale da investire nei 3 fondi, mentre le ultime 2 componenti rappresentano i moltiplicatori di Lagrange associati ai vincoli. Il rischio corrispondente a questa suddivisione del capitale, pari al valore della funzione obiettivo valutata in $\mathbf{x}_1(1:3)$, è circa del 17%. ■

7.8.1 Il metodo di penalizzazione

Un modo naturale di risolvere il problema (7.69) consiste nel ricondursi ad un problema di ottimizzazione non vincolata per una funzione modificata, detta *funzione di penalizzazione*

$$\mathcal{P}_\alpha(\mathbf{x}) = f(\mathbf{x}) + \frac{\alpha}{2} \sum_{i \in \mathcal{I}_h} h_i^2(\mathbf{x}) + \frac{\alpha}{2} \sum_{j \in \mathcal{I}_g} (\max\{-g_j(\mathbf{x}), 0\})^2 \quad (7.79)$$

dove $\alpha > 0$ è un parametro fissato.

Se i vincoli dati non sono soddisfatti nel punto \mathbf{x} , le sommatorie che compaiono in (7.79) forniscono una misura di quanto il punto disti dall'insieme Ω dei punti ammissibili. Poiché in quest'ultimo caso \mathbf{x} viola i vincoli dati, grandi valori di α penalizzano severamente tale distanza.

Ogni soluzione \mathbf{x}^* del problema (7.69) è evidentemente un punto di minimo di \mathcal{P} . Viceversa, sotto ipotesi di regolarità per f , h_i e g_j , denotando con $\mathbf{x}^*(\alpha)$ il punto di minimo di $\mathcal{P}_\alpha(\mathbf{x})$, abbiamo [Ber82]

$$\lim_{\alpha \rightarrow \infty} \mathbf{x}^*(\alpha) = \mathbf{x}^*.$$

Per $\alpha \gg 1$, $\mathbf{x}^*(\alpha)$ può quindi considerarsi una buona approssimazione della soluzione \mathbf{x}^* . Tuttavia, poiché le instabilità numeriche che si generano nella minimizzazione di $\mathcal{P}_\alpha(\mathbf{x})$ aumentano all'aumentare di α , non è una buona strategia quella di minimizzare $\mathcal{P}_\alpha(\mathbf{x})$ semplicemente scegliendo un parametro α molto grande. Una strategia migliore consiste nel risolvere una successione di problemi di minimo e quindi nel calcolare la successione

$$\mathbf{x}^{(k)} = \underset{\mathbf{x} \in \mathbb{R}^n}{\operatorname{argmin}} \mathcal{P}_{\alpha_k}(\mathbf{x}), \quad k \geq 0, \quad (7.80)$$

dove $\{\alpha_k\}$ è una successione crescente e non limitata di parametri (con, ad esempio, $\alpha_0 = 1$). Per ogni k , α_k è scelto in funzione di α_{k-1} , mentre $\mathbf{x}^{(k-1)}$ fornisce il valore iniziale per risolvere il problema (7.80) al nuovo passo k .

Un approccio euristico consiste nello scegliere $\alpha_k = \delta \alpha_{k-1}$ dove δ è dell'ordine dell'unità (ad esempio $\delta \in [1.5, 2]$) se servono molte iterazioni

per risolvere (7.80) al passo $k - 1$, altrimenti si sceglie un valore di δ di un ordine maggiore, ad esempio $\delta \simeq 10$.

Le iterazioni in k vengono fermate quando

$$\|\nabla_{\mathbf{x}} \mathcal{P}_{\alpha_k}(\mathbf{x}^{(k)})\| \leq \bar{\varepsilon}$$

per un certo $\bar{\varepsilon} > 0$ assegnato.

Per risolvere i problemi (7.80) possiamo utilizzare uno qualsiasi dei metodi presentati nelle sezioni precedenti per l'ottimizzazione non vincolata, avendo cura di scegliere la tolleranza ε_k (per il test d'arresto del metodo di minimizzazione non vincolata) né troppo blanda né troppo restrittiva per il passo k in esame. Una possibile strategia consiste nello scegliere $\varepsilon_0 = 1/10$ e, per $k > 0$, nel definire $\varepsilon_{k+1} = \max\{\bar{\varepsilon}, \varepsilon_k/10\}$.

Formuliamo l'algoritmo come segue. Dati α_0 (tipicamente $\alpha_0 = 1$), ε_0 (solitamente $\varepsilon_0 = 1/10$), $\bar{\varepsilon} > 0$ e $\mathbf{x}_0^{(0)} \in \mathbb{R}^n$, per $k = 0, 1, \dots$ fino a convergenza

```

calcolare un'approssimazione  $\mathbf{x}^{(k)}$  di (7.80)
con uno dei metodi di ottimizzazione non vincolata
con dato iniziale  $\mathbf{x}_0^{(k)}$  e tolleranza  $\varepsilon_k$  sul test d'arresto
if  $\|\nabla_{\mathbf{x}} \mathcal{P}_{\alpha_k}(\mathbf{x}^{(k)})\| \leq \bar{\varepsilon}$ 
    convergenza raggiunta:
    accettare  $\mathbf{x}^{(k)}$  come approssimazione di  $\mathbf{x}^*$ 
else
    scegliere  $\alpha_{k+1}$  t.c.  $\alpha_{k+1} > \alpha_k$ 
    porre  $\varepsilon_{k+1} = \max\{\bar{\varepsilon}, \varepsilon_k/10\}$ 
    porre  $\mathbf{x}_0^{(k+1)} = \mathbf{x}^{(k)}$ 
endif

```

(7.81)

Questo algoritmo è implementato nel Programma 7.6. `fun` e `grad_fun` sono *function handle* associati alla funzione obiettivo ed al suo gradiente, `h` e `grad_h` sono quelli associati ai vincoli di uguaglianza, mentre `g` e `grad_g` sono quelli associati ai vincoli di disegualanza. Se \mathcal{I}_h (risp., \mathcal{I}_g) è un insieme vuoto, allora `h` e `grad_h` (risp. `g` e `grad_g`) sono variabili inizializzate vuote. Gli output delle *function* `grad_fun`, `grad_h` e `grad_g` contengono rispettivamente: un vettore colonna `y` di n componenti tale che $y_i = \partial f / \partial x_i$, una matrice `C` di dimensione $n \times p$ tale che $C_{ji} = \partial h_i / \partial x_j$, una matrice `G` di dimensione $n \times q$ tale che $G_{j\ell} = \partial g_\ell / \partial x_j$. Il vettore `x0` contiene il punto iniziale del metodo iterativo, `tol` e `kmax` contengono tolleranza e numero massimo di iterazioni per il ciclo di penalizzazione, mentre `kmaxd` è il massimo numero di iterazioni

per il metodo di discesa qualora esso venga richiamato per risolvere ad ogni passo il problema non vincolato. Infine la variabile `meth` è utilizzata per selezionare il metodo di minimizzazione non vincolata: se `meth=0` viene richiamata la *function* `fminsearch` di `MATGOCT` che implementa il metodo di Nelder e Mead, mentre se `meth>1`, allora `meth` svolge lo stesso ruolo assunto nel programma `descent 7.3` per selezionare il metodo di discesa. In quest'ultimo caso, un ulteriore input opzionale conterrà la matrice Hessiana necessaria per implementare il metodo di discesa con le direzioni di Newton se `meth=1`, mentre conterrà la matrice H_0 per il metodo BFGS (7.45) se `meth=2`.

Programma 7.6. `penalty`: il metodo di penalizzazione

```

function [x,err,k]=penalty(fun,grad_fun,h,grad_h, ...
g,grad_g,x0,tol,kmax,kmaxd,meth,varargin)
% PENALTY Ottimizzazione vincolata con penalizzazione
% [X,ERR,K]=PENALTY(FUN,GRAD_FUN,H,GRAD_H, ...
% G,GRAD_G,X0,TOL,KMAX,KMAXD,METH)
% Approssima un punto di minimo della funzione FUN
% soggetto ai vincoli di uguaglianza  $H=0$  e di disu-
% guaglianza  $G \geq 0$ , con un metodo di penalizzazione,
% partendo da un punto  $X_0$ , fino al
% raggiungimento della tolleranza TOL o di KMAX
% iterazioni. GRAD_FUN, GRAD_H, e GRAD_G contengono
% i gradienti di FUN, H e G, rispettivamente. I
% vincoli ed i rispettivi gradienti sono da
% inizializzare con [], qualora non siano presenti.
% Per risolvere il problema di minimo non vincolato
% si richiama il programma FMINSEARCH (se METH=0)
% o DESCENT (se METH>0). Quando METH>0, KMAXD e
% METH contengono rispettivamente il numero
% massimo di iterazioni e la scelta del metodo
% di discesa per il programma DESCENT. Se METH=1
% (o METH=2) si richiede l'espressione dell'Hes-
% siana (o una sua approssimazione al passo 0) come
% ultimo parametro in input.
xk=x0(:); alpha0=1;
if meth==1, hess=varargin{1};
elseif meth==2, hess=varargin{1};
else hess=[]; end
if ~isempty(h), [nh,mh]=size(h(xk)); end
if ~isempty(g), [ng,mg]=size(g(xk)); else, ng=[]; end
err=tol+1; k=0;
alphak=alpha0; alphak2=alphak/2; told=.1;
while err>tol && k< kmax
    P=@(x)Pf(x,fun,g,h,alphak2,ng);
    grad_P=@(x)grad_Pf(x,grad_fun,h,g,grad_h, ...
                           grad_g,alphak,ng);
    if meth==0
        options=optimset('TolX',told);
        [x,err,kd]=fminsearch(P,xk,options);
        err=norm(x-xk);
    else
        [x,err,kd]=descent(P,grad_P,xk,told,kmaxd, ...
                           meth,hess);
        err=norm(grad_P(x));
    end
    alphak=alphak2;
    alphak2=alphak/2;
    told=err;
    k=k+1;
end

```

```

    end
    if kd<kmaxd
        alphak=alphak*10; alphak2=alphak/2;
    else
        alphak=alphak*1.5; alphak2=alphak/2;
    end
    k=k+1; xk=x; told=max([tol,told/10]);
    end
end % end of the function penalty
function y=Pf(x,fun,g,h,alphak2,ng)
y=fun(x);
if ~isempty(h), y=y+alphak2*sum((h(x)).^2); end
if ~isempty(g), G=g(x);
for j=1:ng, y=y+alphak2*max([-G(j),0])^2; end
end
end % end of function Pf
function y=grad_Pf(x,grad_fun,h,g,%
                     grad_h,grad_g,alphak,ng)
y=grad_fun(x);
if ~isempty(h), y=y+alphak*grad_h(x)*h(x); end
if ~isempty(g), G=g(x); Gg=grad_g(x);
for j=1:ng
if G(j)<0, y=y+alphak*Gg(:,j)*G(j); end
end, end
end % end of function grad_Pf

```

Esempio 7.15 Risolviamo il Problema 2 dell’Esempio 7.13 richiamando il Programma 7.6. Fissiamo $\mathbf{x}^{(0)} = (1.2, 0.2)$ e tolleranza $\bar{\epsilon} = 10^{-5}$. Con le seguenti istruzioni:

```

fun=@(x) 100*(x(2)-x(1).^2).^2+(1-x(1)).^2;
grad_fun=@(x) [-400*(x(2)-x(1)^2)*x(1)-2*(1-x(1));
                200*(x(2)-x(1)^2)];
g=@(x) [-34*x(1)-30*x(2)+19; 10*x(1)-5*x(2)+11;
            3*x(1)+22*x(2)+8];
grad_g=@(x) [-34,10,3;-30,-5,22];
x0=[1.2,.2]; tol=1.e-5; kmax=100; kmaxd=100;
meth=2; hess=eye(2);
[x,err,k]=penalty(fun,grad_fun,[],[],g,grad_g,%
                    x0,tol,kmax,kmaxd,meth,hess)

```

otteniamo convergenza al punto $(0.41183, 0.16660)$ in 3 iterazioni con un residuo sul gradiente $\|\nabla_{\mathbf{x}} \mathcal{P}(\mathbf{x}^{(3)}, \mu_3)\| \simeq 2.6379 \cdot 10^{-7}$. Per risolvere il problema di minimo non vincolato abbiamo utilizzato il Programma 7.3 descent, in particolare le direzioni quasi-Newton generate dal metodo BFGS descritto nella Sezione 7.5.4. I vincoli nel punto di minimo trovato valgono $g_1(\mathbf{x}) = 2.0036e-04$, $g_2(\mathbf{x}) = 1.4285e+01$ e $g_3(\mathbf{x}) = 1.2901e+01$. ■

Esempio 7.16 Vogliamo risolvere il Problema 7.3 con il metodo di penalizzazione. Sia Ω il cerchio di centro l’origine e raggio 2, costruiamo la mesh rappresentata a sinistra di Figura 7.2 con 49 nodi (i vertici) e $Ne = 72$ triangoli. I 24 nodi del bordo rimangono fissi, mentre le coordinate dei 25 nodi interni ad Ω vengono memorizzate nel vettore \mathbf{x} e rappresentano le variabili indipendenti del problema. La funzione obiettivo è

$$f(\mathbf{x}) = \sum_{k=1}^{Ne} \frac{1}{\mu_k(\mathbf{x})} = \sum_{k=1}^{Ne} \frac{\sqrt{3} \|\mathbf{A}_k(\mathbf{x}) \mathbf{W}^{-1}\|_F^2}{4 \det(\mathbf{A}_k(\mathbf{x}))},$$

mentre i vincoli di disuguaglianza sono

$$g_k(\mathbf{x}) = \det(\mathbf{A}_k(\mathbf{x})) - \tau \geq 0, \quad k = 1, \dots, Ne,$$

dove $\tau = 0.10876$ è il doppio dell'area del più piccolo triangolo della mesh iniziale. Il risultato mostrato a destra di Figura 7.2 è stato ottenuto dopo 21 iterazioni dell'algoritmo di penalizzazione, avendo posto $\bar{\epsilon} = 10^{-8}$ per il test d'arresto. Il numero massimo di iterazioni per il metodo di Nelder and Mead è stato fissato pari a 100. ■

Esempio 7.17 Risolviamo il Problema 7.5 considerando la rete di strade di Figura 7.3. Abbiamo $11 (= n)$ strade s_j e $7 (= p)$ incroci o nodi. Supponiamo che ogni minuto entrino ed escano dalla rete $M = 20$ veicoli, che le lunghezze delle strade s_j siano memorizzate nel vettore $\mathbf{L} = (1, 1, 1.5, 1.5, 1.5, 2.2, 1.5, 1.5, 2.2, 1.5, 2.2)$ km (la j -ima componente è la lunghezza della strada s_j , si veda Figura 7.3), che la velocità massima su ogni tratto stradale sia $v_{j,m} = 1$ km/min e che le densità massime $\rho_{j,m}$ di veicoli sui tratti di strada s_j siano memorizzate nel vettore $\rho_m = (60, 40, 20, 60, 60, 40, 60, 20, 40, 20, 60)$. In questo problema abbiamo sia vincoli di uguaglianza che vincoli di disuguaglianza, quindi utilizziamo il metodo di penalizzazione. Per risolvere il problema di minimo non vincolato associato, richiamiamo il metodo di discesa con direzioni quasi-Newton, per il quale dobbiamo fornire l'espressione dei gradienti della funzione obiettivo e dei vincoli. Esprimendo la funzione obiettivo f ed i vincoli di uguaglianza solo in termini delle variabili indipendenti ρ_j , abbiamo

$$\begin{aligned} f(\boldsymbol{\rho}) &= \left(\sum_{j=1}^n \frac{L_j}{v_{j,m}} \frac{\rho_j}{1 - \rho_j/\rho_{j,m}} \right) / \sum_{j=1}^n \rho_j, \\ h_1(\boldsymbol{\rho}) &= M - \sum_{j=1}^2 v_{j,m} (1 - \rho_j/\rho_{j,m}) \\ h_2(\boldsymbol{\rho}) &= v_{1,m} (1 - \rho_1/\rho_{1,m}) - \sum_{j=3}^4 v_{j,m} (1 - \rho_j/\rho_{j,m}) \\ &\dots \\ h_7(\boldsymbol{\rho}) &= \sum_{j=9}^{11} v_{j,m} (1 - \rho_j/\rho_{j,m}) - M \\ g_j(\boldsymbol{\rho}) &= \rho_j \quad j = 1, \dots, 11 \\ g_{11+j}(\boldsymbol{\rho}) &= \rho_{j,m} - \rho_j \quad j = 1, \dots, 11. \end{aligned} \tag{7.82}$$

Fissato un vettore $\boldsymbol{\rho}$, il gradiente $\nabla f(\boldsymbol{\rho})$ e le matrici $[\nabla h_1(\boldsymbol{\rho}), \dots, \nabla h_p(\boldsymbol{\rho})]$ e $[\nabla g_1(\boldsymbol{\rho}), \dots, \nabla g_n(\boldsymbol{\rho}), \nabla g_{n+1}(\boldsymbol{\rho}), \dots, \nabla g_{2n}(\boldsymbol{\rho})]$ possono essere costruiti mediante 3 distinte *user-defined function* `grad_fun.m`, `grad_h.m` e `grad_g.m`. Richiamando il programma 7.6 con vettore iniziale di valori unitari e tolleranza 10^{-5} per il test d'arresto otteniamo convergenza in 5 iterazioni al vettore

```
rho_opt =
15.0246    12.8942    4.6535    9.0594    5.5847    9.4996
 0.5278   -0.0000   11.5494    6.9723    8.4272.
```

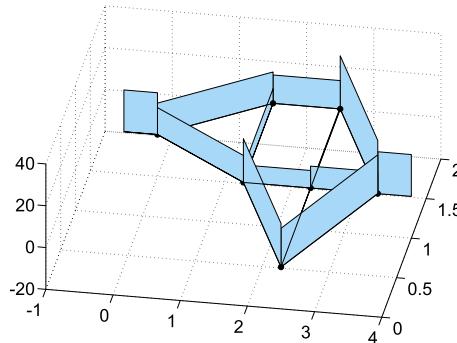


Figura 7.16. Le densità ρ_j per la rete di strade del Problema 7.5

Le componenti, ordinate per righe e rappresentate graficamente in Figura 7.16, sono le densità ρ_j dei vari tratti stradali s_j che minimizzano la funzione obiettivo. Il minimo tempo medio di transito trovato è $f(\boldsymbol{\rho}_{opt}) = 2.0782$ min. ■

7.8.2 Il metodo della Lagrangiana aumentata

In questa sezione consideriamo problemi di minimo con soli vincoli di uguaglianza, quindi poniamo $\mathcal{I}_g = \emptyset$ in (7.69). Sia $\alpha > 0$ un parametro assegnato, la funzione

$$\mathcal{L}_\alpha(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) - \sum_{i \in \mathcal{I}_h} \lambda_i h_i(\mathbf{x}) + \frac{\alpha}{2} \sum_{i \in \mathcal{I}_h} h_i^2(\mathbf{x}) \quad (7.83)$$

ottenuta da (7.74) è detta *Lagrangiana aumentata*. Come nel metodo della penalizzazione, se \mathbf{x} non soddisfa i vincoli $h_i(\mathbf{x}) = 0$, un valore molto grande di α amplifica l'effetto su \mathcal{L} della violazione dei vincoli stessi.

Il metodo della Lagrangiana aumentata è un metodo iterativo che, alla k -sima iterazione, dati α_k e $\boldsymbol{\lambda}^{(k)}$, calcola

$$\mathbf{x}^{(k)} = \underset{\mathbf{x} \in \mathbb{R}^n}{\operatorname{argmin}} \mathcal{L}_{\alpha_k}(\mathbf{x}, \boldsymbol{\lambda}^{(k)}) \quad (7.84)$$

in modo che la successione $\mathbf{x}^{(k)}$ converga ad un punto KKT (si veda (7.75)) della Lagrangiana $\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) - \sum_{i \in \mathcal{I}_h} \lambda_i h_i(\mathbf{x})$. (7.84) è un problema di minimizzazione non vincolata che può essere risolto con uno dei metodi visti nelle precedenti sezioni.

I valori α_0 e $\boldsymbol{\lambda}^{(0)}$ sono assegnati arbitrariamente, al generico passo $k > 0$, dopo aver determinato $\mathbf{x}^{(k)}$, si calcolano α_{k+1} e $\lambda_i^{(k+1)}$ per $i \in \mathcal{I}_h$ per il passo successivo. Il coefficiente α_{k+1} è ottenuto da α_k come nel metodo di penalizzazione presentato nella Sezione 7.8.1. Per calcolare $\boldsymbol{\lambda}^{(k+1)}$ scriviamo l'espressione

$$\nabla_{\mathbf{x}} \mathcal{L}_{\alpha_k}(\mathbf{x}^{(k)}, \boldsymbol{\lambda}^{(k)}) = \nabla f(\mathbf{x}^{(k)}) - \sum_{i \in \mathcal{I}_h} (\lambda_i^{(k)} - \alpha_k h_i(\mathbf{x}^{(k)})) \nabla h_i(\mathbf{x}^{(k)})$$

e confrontiamola con la condizione di ottimalità (7.75). Se $\mathbf{x}^{(k)}$ approssima \mathbf{x}^* , allora $(\lambda_i^{(k)} - \alpha_k h_i(\mathbf{x}^{(k)}))$ deve approssimare λ_i^* e quindi definiamo

$$\lambda_i^{(k+1)} = \lambda_i^{(k)} - \alpha_k h_i(\mathbf{x}^{(k)}) \quad (7.85)$$

Come per l'algoritmo di penalizzazione, assegniamo una tolleranza $\bar{\varepsilon} > 0$ per il test d'arresto sulle iterazioni k in modo da arrestare le iterazioni al primo k per cui $\|\nabla_{\mathbf{x}} \mathcal{L}_{\alpha_k}(\mathbf{x}^{(k)}, \boldsymbol{\lambda}^{(k)})\| \leq \bar{\varepsilon}$, e una tolleranza $\varepsilon_0 > 0$ per risolvere (7.84) per $k = 0$. I valori ε_k (con $k > 0$) sono aggiornati con la formula $\varepsilon_{k+1} = \max\{\bar{\varepsilon}, \varepsilon_k/10\}$ come nel metodo di penalizzazione.

Riassumiamo l'algoritmo come segue: dato α_0 (tipicamente, $\alpha_0 = 1$), ε_0 (tipicamente, $\varepsilon_0 = 1/10$), $\bar{\varepsilon} > 0$, $\mathbf{x}_0^{(0)} \in \mathbb{R}^n$ e $\boldsymbol{\lambda}_0^{(0)} \in \mathbb{R}^p$, per $k = 0, 1, \dots$ fino a convergenza:

```

calcolare un'approssimazione  $\mathbf{x}^{(k)}$  di (7.84)
con uno dei metodi di ottimizzazione non vincolata
con dato iniziale  $\mathbf{x}_0^{(k)}$  e tolleranza  $\varepsilon_k$  sul test d'arresto
if  $\|\nabla_{\mathbf{x}} \mathcal{L}_{\alpha_k}(\mathbf{x}^{(k)}, \boldsymbol{\lambda}^{(k)})\| \leq \bar{\varepsilon}$ 
    convergenza raggiunta:
    accettare  $\mathbf{x}^{(k)}$  come approssimazione di  $\mathbf{x}^*$ 
else
    scegliere  $\alpha_{k+1}$  t.c.  $\alpha_{k+1} > \alpha_k$ 
    porre  $\varepsilon_{k+1} = \max\{\bar{\varepsilon}, \varepsilon_k/10\}$ 
    porre  $\mathbf{x}_0^{(k+1)} = \mathbf{x}^{(k)}$ 
endif

```

(7.86)

Questo algoritmo è implementato nel Programma 7.7. Ad eccezione del parametro `lambda0`, che qui contiene il vettore iniziale $\boldsymbol{\lambda}^{(0)}$ dei moltiplicatori di Lagrange, gli altri parametri di input e di output coincidono con quelli del Programma 7.6.

Programma 7.7. auglagrange: il metodo della Lagrangiana aumentata

```

function [x,err,k]=auglagrange(fun,grad_fun,h,grad_h,%
    % x0,lambda0,tol,kmax,kmaxd,meth,varargin)
% AUGLAGRANGE Ottimizz. vincolata con Lagr. aumentata
% [X,ERR,K]=AUGLAGRANGE(FUN,GRAD_FUN,H,GRAD_H,%
% X0,LAMBDAO,TOL,KMAX,KMAXD,METH)

```

```
% Approssima un punto di minimo della funzione FUN
% soggetto ai vincoli di uguaglianza H con il metodo
% della Lagrangiana aumentata, partendo da un punto
% XO, fino al raggiungimento della tolleranza TOL
% o di KMAX iterazioni. GRAD_FUN e GRAD_H contengono
% i gradienti di FUN e H, rispettivamente.
% Per risolvere il problema di minimo non vincolato
% si richiama il programma FMINSEARCH (se METH=0)
% o DESCENT (se METH>0). Quando METH>0, KMAXD e
% METH contengono rispettivamente il numero
% massimo di iterazioni e la scelta del metodo
% di discesa per il programma DESCENT. Se METH=1
% (o METH=2) si richiede l'espressione dell'Hessiana
% (o una sua approssimazione al passo 0) come
% ultimo parametro in input.
alpha0=1;
if meth==1, hess=varargin{1};
elseif meth==2, hess=varargin{1};
else, hess=[]; end
err=tol+1; k=0; xk=x0(:); lambdak=lambda0(:);
if ~isempty(h), [nh,mh]=size(h(xk)); end
alphak=alpha0; alphak2=alphak/2; told=0.1;
while err>tol && k< kmax
    L=@(x)Lf(x,fun,lambdak,alphak2,h);
    grad_L=@(x)grad_Lf(x,grad_fun,lambdak,alphak,...
                    h,grad_h);
    if meth==0
        options=optimset('TolX',told);
        [x,err,kd]=fminsearch(L,xk,options);
        err=norm(x-xk);
    else
        [x,err,kd]=descent(L,grad_L,xk,told,kmaxd,meth,...
                            hess);
        err=norm(grad_L(x));
    end
    lambdak=lambdak+alphak*h(x);
    if kd<kmaxd
        alphak=alphak*10; alphak2=alphak/2;
    else
        alphak=alphak*1.5; alphak2=alphak/2;
    end
    k=k+1; xk=x; told=max([tol,told/10]);
    end
end % end auglagrange

function y=Lf(x,fun,lambdak,alphak2,h)
y=fun(x);
if ~isempty(h)
    y=y-sum(lambdak.*h(x))+alphak2*sum((h(x)).^2);
end
end % end function Lf

function y=grad_Lf(x,grad_fun,lambdak,alphak,h,grad_h)
y=grad_fun(x);
if ~isempty(h)
    y=y+grad_h(x)*(alphak*h(x)-lambdak);
end
end % end function grad_Lf
```

Esempio 7.18 Risolviamo il Problema 1 dell’Esempio 7.13 con il metodo della Lagrangiana aumentata e richiamando il Programma 7.7 con le seguenti istruzioni MATLAB:

```
fun=@(x)0.6*x(1).^2+0.5*x(2).*x(1)-x(2)+3*x(1);
grad_fun=@(x) [1.2*x(1)+0.5*x(2)+3; 0.5*x(1)-1];
h=@(x)x(1).^2+x(2).^2-1;
grad_h=@(x)[2*x(1); 2*x(2)];
x0=[1.2,.2]; tol=1.e-5; kmax=500; kmaxd=100;
p=1; % number of equality constraints
lambda0=rand(p,1); meth=2; hess=eye(2);
[xmin,err,k]=auglagrange(fun,grad_fun,h,grad_h, ...
x0,lambda0,tol,kmax,kmaxd,meth,hess)
```

Abbiamo posto la tolleranza per il test d’arresto pari a 10^{-5} e risolto il problema di minimo non vincolato con le direzioni di discesa quasi-Newton (fissando i parametri `meth=2` e `hess=eye(2)`).

Il valore iniziale del moltiplicatore di Lagrange è stato scelto casualmente e si è ottenuta convergenza in 5 iterazioni al punto di minimo

```
x_lag =
-8.454667252699469e-01
5.340281045624525e-01
```

Il valore del vincolo h nel punto calcolato è $h(\mathbf{x}_{lag}) \simeq 5.6046 \cdot 10^{-10}$. La soluzione di questo problema è riportata in Figura 7.15, a sinistra.

Utilizzando il metodo di penalizzazione anziché quello della Lagrangiana aumentata, usando gli stessi parametri in input ed optando sempre per le direzioni quasi-Newton per la risoluzione del problema non vincolato, si ottiene convergenza in 6 iterazioni al punto

```
x_p =
-8.454715822058602e-01
5.340328869427682e-01
```

Il valore del vincolo h nel punto calcolato è stavolta $h(\mathbf{x}_p) \simeq 1.3320 \cdot 10^{-4}$, maggiore di ben 6 ordini di grandezza di quello ottenuto con il metodo della Lagrangiana aumentata. È molto frequente che, a parità di tolleranza imposta per il test d’arresto, la soluzione calcolata con il metodo di penalizzazione violi i vincoli in misura molto maggiore rispetto a quella ottenuta con il metodo della Lagrangiana aumentata. Per questo motivo il metodo della Lagrangiana aumentata è in genere da preferirsi per risolvere problemi di minimo con soli vincoli di uguaglianza. ■

Si vedano gli Esercizi 7.12–7.14.



Riassumendo

- Per un problema di minimo vincolato, i punti di minimo devono essere cercati tra i punti KKT associati alla funzione Lagrangiana, oltre che tra i punti in cui la funzione obiettivo non è regolare;
- in un problema di programmazione quadratica la funzione obiettivo è quadratica ed i vincoli sono lineari e, sotto opportune ipotesi sulla

- matrice associata al termine quadratico e sui vincoli, esiste un unico punto di minimo che viene calcolato risolvendo un sistema lineare;
3. un problema di minimo vincolato può essere trasformato in un problema non vincolato mediante la formulazione penalizzata. Il problema penalizzato può risultare molto mal condizionato a causa del valore molto grande che viene assegnato al parametro di penalizzazione;
 4. il metodo della Lagrangiana aumentata è un metodo di penalizzazione applicato alla funzione Lagrangiana, per la ricerca dei punti KKT.

7.9 Cosa non vi abbiamo detto

I problemi di ottimizzazione di grandi dimensioni sono molto onerosi, sia in termini di tempo di calcolo sia per l'allocazione di memoria richiesta. Sia i metodi di tipo *line search* che quelli di tipo *trust region* richiedono la fattorizzazione della matrice Hessiana o la costruzione di sue approssimazioni che potrebbero essere matrici dense anche quando l'Hessiana è sparsa. In questi ultimi anni sono state sviluppate particolari varianti dei metodi descritti nelle precedenti sezioni con l'obiettivo di limitare l'uso della memoria. Tali varianti si basano sulle iterazioni del Gradiente Coniugato o del metodo di Lanczos. Si vedano ad esempio [Ste83, NW06, GOT05].

Un metodo classico ed efficiente per la risoluzione di problemi di minimo vincolati è il cosiddetto *Sequential Quadratic Programming* (SQP), che trasforma un problema di minimo con funzione obiettivo f e vincoli arbitrari nella soluzione successiva di problemi di Programmazione Quadratica. In particolare, ad ogni iterazione f è approssimata da una funzione quadratica come (7.76) e si cercano i punti KKT associati alla funzione Lagrangiana (si vedano ad esempio [Fle10], [NW06]).

Nel caso di vincoli di sola uguaglianza i *metodi di barriera* rappresentano un'alternativa al metodo di penalizzazione: la funzione obiettivo è modificata aggiungendo una funzione che dipende dai vincoli di diseguaglianza e che impedisce che un punto ammissibile $\mathbf{x} \in \Omega$ porti ad un successivo punto non ammissibile. Questa funzione di barriera è definita solo all'interno dell'insieme Ω dei valori ammissibili ed è illimitata sul bordo di Ω . Questi metodi richiedono che il punto iniziale, scelto dell'utente, sia ammissibile, e questa è una condizione difficile da soddisfare. Per una presentazione di questi metodi rimandiamo a [Ter10].

7.10 Esercizi

Esercizio 7.1 Si calcoli il punto di minimo della funzione $f(x) = (x - 1)e^{-x^2}$ con il metodo della sezione aurea con e senza interpolazione quadratica.

Esercizio 7.2 Due navi partono contemporaneamente da due porti diversi e si muovono lungo due traiettorie descritte dalle seguenti curve parametriche

$$\gamma_1(t) = \begin{cases} 7 \cos\left(\frac{t}{3} + \frac{\pi}{2}\right) + 5 \\ -4 \sin\left(\frac{t}{3} + \frac{\pi}{2}\right) - 3 \end{cases}, \quad \gamma_2(t) = \begin{cases} 6 \cos\left(\frac{t}{6} - \frac{\pi}{3}\right) - 4 \\ -6 \sin\left(\frac{t}{6} - \frac{\pi}{3}\right) + 5 \end{cases}.$$

Il parametro $t > 0$ rappresenta il tempo in ore, mentre le posizioni sono espresse in miglia marine rispetto all'origine del sistema di riferimento. Determinare la minima distanza a cui si trovano le due navi lungo il loro moto.

Esercizio 7.3 Calcolare i punti di minimo globale della funzione $f(\mathbf{x}) = x_1^4 + x_2^4 + x_1^3 + 3x_1x_2^2 - 3x_1^2 - 3x_2^2 + 10$ con il metodo di Nelder e Mead.

Esercizio 7.4 Dopo aver posto $x^{(0)} = 3/2$, $d^{(k)} = (-1)^{k+1}$ e $\alpha_k = 2 + 2/3^{k+1}$, mostrare che il metodo di discesa genera una successione $\{x^{(k)}\}$ che non converge al punto di minimo della funzione $f(x) = x^4$, anche se la successione $\{f(x^{(k)})\}$ è decrescente. Verificare inoltre che i passi α_k non soddisfano le condizioni di Wolfe (7.39).

Esercizio 7.5 Mostrare che valgono le medesime conclusioni dell'esercizio precedente qualora si prendano $x^{(0)} = -2$, $d^{(k)} = 1$ e $\alpha_k = 3^{-(k+1)}$.

Esercizio 7.6 Si vuole approssimare il punto di minimo della funzione di Rosenbrock definita nell'Esempio 7.3 con il metodo di discesa e differenti scelte delle direzioni di discesa (7.34)–(7.37). Fissare il punto iniziale $\mathbf{x}^{(0)} = (-1.2, 1)$ e tolleranza $\varepsilon = 10^{-8}$ per il test d'arresto. Rappresentare graficamente le storie di convergenza dei vari metodi e commentare l'efficienza dei metodi in termini di numero di iterazioni necessarie a convergere.

Esercizio 7.7 Calcolare il minimo della funzione $f(\mathbf{x}) = (x_1^2 - x_1^3 x_2 - 2x_2 + 2x_1 x_2^2)^2 + (3 - x_1 x_2)^2$ con il metodo BFGS ed il metodo *trust region* in cui si usino le direzioni quasi-Newton per risolvere il problema (7.50). Si prendano $\mathbf{x}^{(0)} = (2, -1)$, $\mathbf{x}^{(0)} = (2, 1)$ e $\mathbf{x}^{(0)} = (-1, -1)$.

Esercizio 7.8 Verificare che il metodo di Gauss-Newton (7.63) può essere riscritto come segue: per $k = 0, 1, \dots$ fino a convergenza

$$\min_{\mathbf{x} \in \mathbb{R}^n} \frac{1}{2} \|\tilde{\mathbf{R}}_k(\mathbf{x})\|^2 \text{ con } \tilde{\mathbf{R}}_k(\mathbf{x}) \text{ definito in (7.64)}. \quad (7.87)$$

Esercizio 7.9 Si consideri il metodo di Gauss-Newton descritto nella Sezione 7.7.1. Dimostrare che se $J_{\mathbf{R}}(\mathbf{x}^{(k)})$ ha rango massimo, allora la soluzione $\delta \mathbf{x}^{(k)}$ di (7.63)₁ è una direzione di discesa per la funzione f definita in (7.61).

Esercizio 7.10 Si considerino i valori

t_i	0.055	0.181	0.245	0.342	0.419	0.465	0.593	0.752
y_i	2.80	1.76	1.61	1.21	1.25	1.13	0.52	0.28

cerchiamo una funzione $\phi(t) = x_1 + x_2 t + x_3 t^2 + x_4 e^{-x_5 t}$ con x_1, x_2, \dots, x_5 incogniti che approssimi i dati (t_i, y_i) nel senso dei minimi quadrati.

Esercizio 7.11 Dimostrare che la funzione $\tilde{f}_k(\mathbf{x})$ definita in (7.65) è un'approssimazione quadratica di f , ottenuta approssimando $\mathbf{R}(\mathbf{x})$ con il suo modello lineare (7.64).

Esercizio 7.12 Si vuole trovare la posizione ottimale di un magazzino merci che deve servire tre punti vendita dislocati in tre diverse posizioni, le cui coordinate sono riportate nella seguente tabella:

Punto vendita	coordinate (x_i, y_i)	consegne annuali
1	(6, 3)	140
2	(−9, 9)	134
3	(−8, −5)	88

L'origine del sistema di riferimento rappresenta il centro città e le coordinate sono espresse in km. Nella terza colonna della tabella sono indicati i viaggi che devono essere effettuati in un anno dal magazzino verso ogni punto vendita. Il magazzino può essere posizionato nella regione $\Omega = \{(x, y) \in \mathbb{R}^2 : y \leq x - 10\}$.

Esercizio 7.13 Calcolare il punto di minimo del problema di Programmazione Quadratica (7.76) con soli vincoli di uguaglianza e

$$\mathbf{A} = \begin{bmatrix} 2 & -1 & 1 \\ -1 & 3 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 1 \\ -2 \\ -1 \end{bmatrix},$$

$$\mathbf{C} = \begin{bmatrix} 2 & -2 & 0 \\ 2 & 1 & -3 \end{bmatrix}, \quad \mathbf{d} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

Esercizio 7.14 Un punto materiale si muove lungo una traiettoria ellittica di equazione $x^2/4 + y^2 = 1$ con velocità $v(x, y) = (\sin(\pi xy) + 1)(2x + 3y + 4)$. Determinare il valore massimo della velocità assunta dal corpo e la posizione in cui essa viene raggiunta.

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. In questo capitolo ci occuperemo di equazioni differenziali ordinarie, mentre dedicheremo il capitolo successivo allo studio di equazioni alle derivate parziali.

8.1 Alcuni problemi

Le equazioni differenziali ordinarie consentono di descrivere l'evoluzione di numerosi fenomeni nei campi più svariati. Vediamone alcuni esempi.

Problema 8.1 (Termodinamica) Consideriamo un corpo avente temperatura interna T e 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 Stefan-Boltzmann (pari a $5.6 \cdot 10^{-8} \text{ J}/(\text{m}^2 \text{K}^4 \text{s})$, 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

$$\frac{dT}{dt} = -\frac{v}{mC}. \quad (8.1)$$

Per la sua soluzione si veda l'Esercizio 8.15. ■

Problema 8.2 (Dinamica delle popolazioni) 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

$$\frac{dy}{dt} = Cy \left(1 - \frac{y}{B}\right), \quad (8.2)$$

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 (8.2) si dovranno sostituire le equazioni seguenti:

$$\begin{aligned} \frac{dy_1}{dt} &= C_1 y_1 (1 - b_1 y_1 - d_2 y_2), \\ \frac{dy_2}{dt} &= -C_2 y_2 (1 - b_2 y_2 - d_1 y_1), \end{aligned} \quad (8.3)$$

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 (8.3) sono note come equazioni di Lotka-Volterra e sono alla base di numerose applicazioni. Per una loro soluzione si veda l'Esempio 8.9. ■

Problema 8.3 (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 8.1 le equazioni del moto sono date dal seguente sistema di equazioni differenziali ordinarie (si vedano [Ada90] e [GN06])

$$\frac{d\mathbf{x}}{dt} = \mathbf{v}, \quad \frac{d\mathbf{v}}{dt} = \mathbf{F},$$

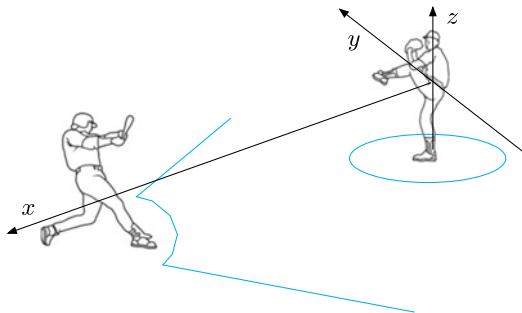


Figura 8.1. Il sistema di riferimento adottato per il Problema 8.3

dove $\mathbf{x}(t) = (x(t), y(t), z(t))^T$ è la posizione della palla al tempo t , $\mathbf{v}(t) = (v_x(t), v_y(t), v_z(t))^T$ la sua velocità e \mathbf{F} il vettore di componenti

$$\begin{aligned} F_x &= -F(v)vv_x + B\omega(v_z \sin \phi - v_y \cos \phi), \\ F_y &= -F(v)vv_y + B\omega v_x \cos \phi, \\ F_z &= -g - F(v)vv_z - B\omega v_x \sin \phi, \end{aligned} \quad (8.4)$$

essendo v il modulo di \mathbf{v} , $B = 4.1 \cdot 10^{-4}$ una costante di normalizzazione, ϕ 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 (8.4) e che tiene conto dell'effetto d'attrito dell'aria, è pari a [GN06]

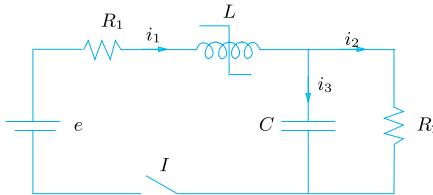
$$F(v) = 0.0039 + \frac{0.0058}{1 + e^{(v-35)/5}}.$$

Per la risoluzione numerica di questo problema si veda l'Esercizio 8.20. ■

Problema 8.4 (Elettrotecnica) Consideriamo il circuito elettrico indicato in Figura 8.2. 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 [HRK04] 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 8.2, derivando rispetto a t entrambi i membri della legge di Kirchoff

**Figura 8.2.** Il circuito elettrico del Problema 8.4

$i_1 = i_2 + i_3$ 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 8.10. ■

8.2 Il problema di Cauchy

Ci limitiamo al caso di un'equazione differenziale ordinaria del prim'ordine. Ricordiamo che 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 nella Sezione 8.10.

Un'equazione differenziale ordinaria ammette in generale infinite soluzioni. Per fissarne una è necessario impostare una condizione che prescriva il valore assunto dalla soluzione in un punto dell'intervallo di integrazione. Ad esempio, l'equazione (8.2) 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 $y : I \subset \mathbb{R} \rightarrow \mathbb{R}$ tale che

$$\begin{cases} y'(t) = f(t, y(t)) & \forall t \in I, \\ y(t_0) = y_0, \end{cases} \quad (8.5)$$

dove I è un intervallo, $f : I \times \mathbb{R} \rightarrow \mathbb{R}$ è 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 8.1 *Supponiamo che la funzione $f(t, y)$ sia*

1. *limitata e continua rispetto ad entrambi gli argomenti;*
2. *lipschitziana rispetto al secondo argomento, ossia esista una costante L positiva (detta costante di Lipschitz) tale che*

$$|f(t, y_1) - f(t, y_2)| \leq L|y_1 - y_2| \quad \forall t \in I, \forall y_1, y_2 \in \mathbb{R}.$$

Allora la soluzione del problema di Cauchy (8.5) esiste, è unica ed è di classe C^1 su I .

Il problema di Cauchy (8.5) è detto *lineare* se la funzione $f(t, y)$ è lineare rispetto alla variabile y .

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 $y'(t) = e^{-t^2}$ 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, per ogni nodo $t_n = t_0 + nh$ (per $n = 1, \dots, N_h$), si cerca il valore incognito u_n che approssimi $y_n = y(t_n)$. L'insieme dei valori $\{u_0 = y_0, u_1, \dots, u_{N_h}\}$ forma la *soluzione numerica*.

Il passo di discretizzazione è indicato qui con il simbolo h in modo da conformarci alla notazione utilizzata nella letteratura classica concernente l'approssimazione dei problemi di Cauchy. Segnaliamo tuttavia al lettore che, nel Capitolo 9 in cui tratteremo i problemi ai valori iniziali ed ai limiti, la notazione h indicherà il passo di discretizzazione spaziale, mentre il passo di discretizzazione temporale verrà indicato con il simbolo Δt .

8.3 I metodi di Eulero e il metodo di Crank-Nicolson

Un metodo classico, il cosiddetto *metodo di Eulero in avanti*, genera la successione seguente

$$u_{n+1} = u_n + hf(t_n, u_n), \quad n = 0, \dots, N_h - 1 \quad (8.6)$$

Questo metodo è derivato dall'equazione differenziale (8.5) 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 (4.6).

Procedendo in maniera analoga, ma utilizzando questa volta il rapporto incrementale (4.10) per approssimare $y'(t_{n+1})$, si ottiene il *metodo di Eulero all'indietro*

$$u_{n+1} = u_n + hf(t_{n+1}, u_{n+1}), \quad n = 0, \dots, N_h - 1 \quad (8.7)$$

Sommendo membro a membro il generico passo dei metodi di Eulero in avanti e di Eulero all'indietro si ottiene un altro metodo implicito ad un passo, il cosiddetto *metodo di Crank-Nicolson*

$$u_{n+1} = u_n + \frac{h}{2}[f(t_n, u_n) + f(t_{n+1}, u_{n+1})], \quad n = 0, \dots, N_h - 1 \quad (8.8)$$

Quest'ultimo metodo può essere anche derivato applicando il teorema fondamentale del calcolo integrale (richiamato nella Sezione 1.6.3) al problema di Cauchy (8.5), ottenendo

$$y_{n+1} = y_n + \int_{t_n}^{t_{n+1}} f(t, y(t)) dt, \quad (8.9)$$

per poi approssimare l'integrale su $[t_n, t_{n+1}]$ con la formula del trapezio (4.26).

Si tratta di tre esempi di metodi *ad un passo* in quanto per calcolare la soluzione numerica nel nodo t_{n+1} necessitiamo 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 , nei metodi di Eulero all'indietro e di Crank-Nicolson essa dipende, tramite $f(t_{n+1}, u_{n+1})$, anche da se stessa. Per tale motivo, il primo metodo è detto *esplicito*, mentre i restanti due sono detti *impliciti* (i metodi (8.6) e (8.7) sono noti rispettivamente anche con i nomi di Eulero esplicito e di Eulero隐式的).

Ad esempio, la discretizzazione di (8.2) 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

$$u_{n+1} = u_n + hCu_{n+1} (1 - u_{n+1}/B).$$

I metodi impliciti sono pertanto più costosi di quelli esplicativi, in quanto se la funzione f del problema (8.5) 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 proprietà di stabilità degli schemi esplicativi.

Il metodo di Eulero in avanti è implementato nel Programma 8.1; l'intervallo di integrazione è `tspan = [t0,tfinal]`, `odefun` è un *function handle* che specifica la funzione $f(t,y)$ e che dipende dalle variabili `t` e `y` (oltre che da eventuali altri parametri opzionali).

Programma 8.1. feuler: il metodo di Eulero in avanti

```
function [t,u]=feuler(odefun,tspan,y0,Nh)
%FEULER Risolve equazioni differenziali
% usando il metodo di Eulero in avanti.
% [T,Y] = FEULER(ODEFUN,TSPAN,Y0,NH) con
% TSPAN = [T0,TF] integra il sistema di equazioni
% differenziali y' = f(t,y) dal tempo T0 a TF con
% condizione iniziale Y0 usando il metodo di Eulero
% in avanti su una griglia equispaziata di NH
% intervalli.
% La funzione ODEFUN(T,Y) deve ritornare un vettore
% contenente f(t,y), della stessa dimensione di y.
% Ogni riga del vettore soluzione Y corrisponde ad
% un istante temporale del vettore colonna T.
h=(tspan(2)-tspan(1))/Nh;
t=linspace(tspan(1),tspan(2),Nh+1)';
y0=y0(:); % genera sempre un vettore colonna
d=length(y0);
u=zeros(Nh+1,d);
u(1,:)=y0.';
for n=1:Nh
    wn=u(n,:).';
    w=wn+h*odefun(t(n),wn);
    u(n+1,:)=w.';
end
```

Il metodo di Eulero all'indietro è implementato nel Programma 8.2. Si noti che abbiamo utilizzato il Programma 2.4 `broyden` per la soluzione del problema non lineare che appare ad ogni passo.

Programma 8.2. beuler: il metodo di Eulero all'indietro

```
function [t,u]=beuler(odefun,tspan,y0,Nh,varargin)
%BEULER Risolve equazioni differenziali
% usando il metodo di Eulero all'indietro.
% [T,Y] = BEULER(ODEFUN,TSPAN,Y0,NH) con
```

```
% TSPAN = [T0,TF] integra il sistema di equazioni
% differenziali y' = f(t,y) dal tempo T0 a TF con
% condizione iniziale Y0 usando il metodo di Eulero
% all'indietro su una griglia equispaziata di NH
% intervalli.
%
% La funzione ODEFUN(T,Y) deve ritornare un vettore
% contenente f(t,y), della stessa dimensione di y.
% Ogni riga del vettore soluzione Y corrisponde ad
% un istante temporale del vettore colonna T.
% Richiama broyden.m per la risoluzione del sistema
% non lineare. Se sono assegnati solo 4 parametri
% di input, tolleranza e numero max di iterazioni per
% il metodo di Broyden sono posti pari a tol=1.e-8
% e kmax=20 rispettivamente, altrimenti
% [T,Y] = BEULER(ODEFUN,TSPAN,Y0,NH,TOL,KMAX)
% permette di specificare anche i parametri per la
% function broyden.m.
if nargin == 4
    tol=1.e-8; kmax=20;
else
    tol=varargin{1}; kmax=varargin{2};
end
h=(tspan(2)-tspan(1))/Nh;
t=linspace(tspan(1),tspan(2),Nh+1)';
y0=y0(:); % genera sempre un vettore colonna
d=length(y0);
u=zeros(Nh+1,d);
B0=eye(d); % matrice iniziale per innestare Broyden
u(1,:)=y0.'; % trasposta anche di variabili complesse
for n=1:Nh
    wn=u(n,:).';
    F=@(x)x-wn-h*odefun(t(n+1),x);
    w=broyden(F,B0,wn,tol,kmax);
    u(n+1,:)=w.';
end
```

Il metodo di Crank-Nicolson è implementato nel Programma 8.3. I parametri di ingresso e di uscita in questo programma sono gli stessi di quelli impiegati per il metodo di Eulero all'indietro.

Programma 8.3. cranknic: il metodo di Crank-Nicolson

```
function [t,u]=cranknic(odefun,tspan,y0,Nh,varargin)
%CRANKNIC Risolve equazioni differenziali
% usando il metodo di Crank-Nicolson.
% [T,Y]=CRANKNIC(ODEFUN,TSPAN,Y0,NH) con
% TSPAN = [T0,TF] integra il sistema di equazioni
% differenziali y' = f(t,y) dal tempo T0 a TF con
% condizione iniziale Y0 usando il metodo di
% Crank-Nicolson su una griglia equispaziata di NH
% intervalli.
%
% La funzione ODEFUN(T,Y) deve ritornare un vettore
% contenente f(t,y), della stessa dimensione di y.
% Ogni riga del vettore soluzione Y corrisponde ad
% un istante temporale del vettore colonna T.
% Richiama broyden.m per la risoluzione del sistema
% non lineare. Se sono assegnati solo 4 parametri
% di input, tolleranza e numero max di iterazioni per
```

```
% il metodo di Broyden sono posti pari a tol=1.e-8
% e kmax=20 rispettivamente, altrimenti
% [T,Y] = CRANKNICH(ODEFUN,TSPAN,Y0,NH,TOL,KMAX)
% permette di specificare anche i parametri per
% function broyden.m.
if nargin == 4
    tol=1.e-8; kmax=20;
else
    tol=varargin{1}; kmax=varargin{2};
end
h=(tspan(2)-tspan(1))/Nh; h2=h/2;
t=linspace(tspan(1),tspan(2),Nh+1)';
y0=y0(:); % genera sempre un vettore colonna
d=length(y0);
u=zeros(Nh+1,d);
B0=eye(d); % matrice iniziale per innestare Broyden
u(1,:)=y0.'; % trasposta anche di variabili complesse
for n=1:Nh
    wn=u(n,:).';
    F=@(x)x-wn-h2*(odefun(t(n),wn)+odefun(t(n+1),x));
    w=broyden(F,B0,wn,tol,kmax);
    u(n+1,:)=w.';
end
```

8.4 Convergenza, Consistenza, Stabilità

Un metodo numerico per l'approssimazione di (8.5) si dice *convergente* se

$$\forall n = 0, \dots, N_h, \quad |y_n - u_n| \leq C(h) \quad (8.10)$$

dove $C(h)$ è una funzione infinitesima per h che tende a 0. Se $C(h) = \mathcal{O}(h^q)$ per un certo intero $q \geq 1$, diremo che il metodo converge con *ordine* q .

8.4.1 Consistenza

Una condizione necessaria per la convergenza è che il metodo sia consistente. Per caratterizzare la proprietà di consistenza introduciamo l'*errore di troncamento locale* $\tau_n(h)$, definito implicitamente attraverso il metodo: $h\tau_n(h)$ è l'errore che si genera “forzando” la soluzione esatta del problema di Cauchy (8.5) a soddisfare lo schema numerico.

L'errore di troncamento locale $\tau_n(h)$ per il metodo di Eulero in avanti è ottenuto a partire da (8.6) e per definizione soddisfa l'identità

$$y_n = y_{n-1} + hf(t_{n-1}, y_{n-1}) + h\tau_n(h). \quad (8.11)$$

Procedendo in maniera analoga per il metodo di Eulero all'indietro, abbiamo

$$y_n = y_{n-1} + hf(t_n, y_n) + h\tau_n(h),$$

mentre l'errore di troncamento locale del metodo di Crank-Nicolson soddisfa

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

Inoltre, definiamo l'*errore di troncamento globale* (o, più semplicemente, *errore di troncamento*) come

$$\tau(h) = \max_{n=1,\dots,N_h} |\tau_n(h)|.$$

Un metodo per il quale l'errore di troncamento globale tende a 0 per h che tende a 0 è detto *consistente*. Diremo inoltre che è consistente con ordine q se $\tau(h) = \mathcal{O}(h^q)$ per un opportuno intero $q \geq 1$.

Per quantificare $\tau_n(h)$ nel caso del metodo di Eulero in avanti, supponiamo che la derivata seconda di y esista e sia continua, pertanto possiamo scrivere lo sviluppo di Taylor di y centrato in t_{n-1}

$$y_n = y_{n-1} + hy'(t_{n-1}) + \frac{h^2}{2}y''(\eta_n), \quad (8.12)$$

essendo η_n un opportuno punto dell'intervallo (t_{n-1}, t_n) . Dal confronto di (8.12) con (8.11) e grazie a (8.5)₁ abbiamo

$$\tau_n(h) = \frac{h}{2}y''(\eta_n) \quad (8.13)$$

e quindi l'errore di troncamento globale per il metodo di Eulero in avanti soddisfa

$$\tau(h) \leq \frac{M}{2}h, \quad (8.14)$$

dove $M = \max_{t \in [t_0, T]} |y''(t)|$.

Deduciamo pertanto che il *metodo di Eulero in avanti è consistente con ordine 1*.

Per il metodo di Eulero all'indietro, usando lo sviluppo di Taylor di y centrato questa volta in t_n , otteniamo $y_{n-1} = y_n - hy'(t_n) + \frac{h^2}{2}y''(\eta_n)$ per un opportuno $\eta_n \in (t_{n-1}, t_n)$, a patto che $y \in C^2$. Pertanto si trova

$$\tau_n(h) = -\frac{h}{2}y''(\eta_n)$$

e anche *il metodo di Eulero all'indietro è consistente con ordine 1 rispetto a h* .

Infine, passando al metodo di Crank-Nicolson, grazie alla (8.9) si ha

$$\tau_n(h) = \frac{1}{h} \left[\int_{t_{n-1}}^{t_n} f(t, y(t)) dt - \frac{h}{2}(f(t_n, y(t_n)) + f(t_{n-1}, y(t_{n-1}))) \right].$$

L'espressione racchiusa nelle parentesi quadre rappresenta l'errore associato all'uso della formula del trapezio (4.26) per l'integrazione numerica. Se supponiamo che $y \in C^3$, dalla (4.27) si ricava

$$\tau_n(h) = -\frac{h^2}{12}y'''(\eta_n) \quad \text{per un opportuno } \eta_n \in (t_{n-1}, t_n). \quad (8.15)$$

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 .

8.4.2 Stabilità

Un'altra proprietà che ha un ruolo fondamentale per la convergenza è la *stabilità*.

In generale, per stabilità di un metodo numerico intendiamo la possibilità di controllare l'effetto sulla soluzione di eventuali perturbazioni sui dati. L'esigenza di formulare una richiesta di stabilità per il metodo numerico è suggerita, prima di ogni altra, dalla necessità di tenere sotto controllo gli inevitabili errori che l'aritmetica finita di ogni calcolatore introduce. Se il metodo numerico non fosse zero-stabile, gli errori di arrotondamento introdotti su y_0 e nel calcolo di $f(t_n, u_n)$ renderebbero infatti la soluzione calcolata del tutto priva di significato.

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, se soddisfatta, garantisce 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 (8.5) sull'intervallo limitato $I = [t_0, T]$ è *zero-stabile* se:

$$\begin{aligned} & \exists h_0 > 0, \exists C > 0, \exists \varepsilon_0 > 0 \text{ t.c. } \forall h \in (0, h_0], \forall \varepsilon \in (0, \varepsilon_0], \\ & \text{se } |\rho_n| \leq \varepsilon, 0 \leq n \leq N_h, \text{ allora } |z_n - u_n| \leq C\varepsilon, 0 \leq n \leq N_h \end{aligned} \quad (8.16)$$

dove:

- C è una costante che può dipendere dalla lunghezza $T - t_0$ dell'intervallo di integrazione I , ma è indipendente da h ;
- u_n è la soluzione che si ottiene con il metodo numerico;
- z_n è la soluzione che si otterebbe applicando il metodo numerico al problema *perturbato*;
- ρ_n rappresenta la perturbazione introdotta al passo n -simo;
- ε_0 indica la massima grandezza della perturbazione.

Naturalmente, ε_0 (e quindi ε) deve essere sufficientemente piccolo da garantire che il problema perturbato ammetta comunque un'unica soluzione sull'intervallo di integrazione I .

Ad esempio, nel caso del metodo di Eulero in avanti, u_n soddisfa il problema

$$\begin{cases} u_{n+1} = u_n + hf(t_n, u_n), & n = 0, \dots, N_h - 1 \\ u_0 = y_0, \end{cases} \quad (8.17)$$

mentre z_n soddisfa il problema perturbato

$$\begin{cases} z_{n+1} = z_n + h [f(t_n, z_n) + \rho_{n+1}], & n = 0, \dots, N_h - 1 \\ z_0 = y_0 + \rho_0. \end{cases} \quad (8.18)$$

Una possibile stima della costante C che compare in (8.16) (sempre per il metodo di Eulero in avanti) è

$$C \leq S_1 = e^{L(T-t_0)}(L+1)/L, \quad (8.19)$$

essendo L la costante di Lipschitz associata alla funzione f . Infatti, grazie alla lipschitzianità di f ed alla limitatezza delle perturbazioni $|\rho_n|$, abbiamo per ogni $n \geq 1$:

$$\begin{aligned} |u_n - z_n| &\leq |u_{n-1} - z_{n-1}| + h|f(t_{n-1}, u_{n-1}) - f(t_{n-1}, z_{n-1})| + h|\rho_n| \\ &\leq (1+hL)|u_{n-1} - z_{n-1}| + h\varepsilon \\ &\leq (1+hL)^2|u_{n-2} - z_{n-2}| + [1 + (1+hL)]h\varepsilon \\ &\quad \dots \\ &\leq (1+hL)^n|\rho_0| + [1 + (1+hL) + \dots + (1+hL)^{n-1}]h\varepsilon \\ &\leq (1+hL)^n|\rho_0| + \frac{(1+hL)^n - 1}{L}\varepsilon \\ &\leq e^{nhL}|\rho_0| + \frac{e^{nhL} - 1}{L}\varepsilon. \end{aligned} \quad (8.20)$$

Abbiamo usato l'identità

$$\sum_{k=0}^{n-1} (1+hL)^k = \frac{(1+hL)^n - 1}{hL} \quad (8.21)$$

e la diseguaglianza $1+hL \leq e^{hL}$. La stima (8.19) segue osservando che $nh = t_n - t_0$ e sfruttando il fatto che $|\rho_0| \leq \varepsilon$.

Qualora $\rho_0 = 0$ e $h < 1/L$ (per cui $|1+hL| < 2$),¹ ripartendo dalla terz'ultima riga di (8.20) otteniamo

¹ Questa limitazione su h è più restrittiva del necessario nei casi in cui f sia derivabile rispetto al suo secondo argomento con derivata negativa. Si veda la Sezione 8.5.

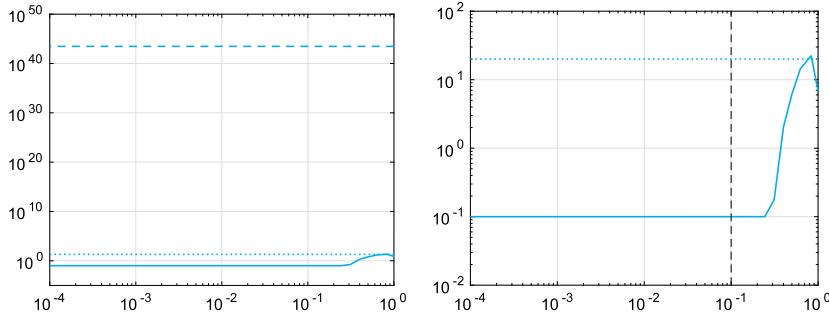


Figura 8.3. La perturbazione $S(h) = \max_n |u_n - z_n|$ (in linea continua) al variare di h , ottenuta risolvendo il problema (8.23) con il metodo di Eulero in avanti e perturbazioni sui dati pari a $\varepsilon = 10^{-3}$. La linea tratteggiata blu rappresenta la costante S_1 , mentre la linea punteggiata rappresenta la costante S_2 . Il grafico di destra è un ingrandimento di quello di sinistra

$$\begin{aligned} |u_n - z_n| &\leq (1 + hL)^n |\rho_0| + [1 + (1 + hL) + \dots + (1 + hL)^{n-1}] h\varepsilon \\ &\leq 2nh\varepsilon \end{aligned}$$

ed una stima più accurata di C (indipendente da L) è

$$C \leq S_2 = 2(T - t_0). \quad (8.22)$$

Consideriamo ad esempio il problema di Cauchy

$$\begin{cases} y'(t) = 5 \cos(2y(t)), & t \in (0, 10], \\ y(0) = 0, \end{cases} \quad (8.23)$$

la cui soluzione esatta è $y(t) = \frac{1}{2} \arcsin((e^{20t} - 1)/(e^{20t} + 1))$. Denotiamo con u_n e z_n le soluzioni numeriche ottenute con il metodo di Eulero in avanti come in (8.17) e (8.18), rispettivamente, prendendo $|\rho_n| = \varepsilon = 10^{-3}$ per ogni $n \geq 0$. La costante di Lipschitz associata a $f(t, y) = 5 \cos(2y)$ è $L = 10$. In Figura 8.3 sono rappresentate la perturbazione massima $S(h) = \max_n |u_n - z_n|$ al variare di h e le costanti S_1 e S_2 definite in (8.19) e (8.22), rispettivamente. Osserviamo che, nonostante ρ_0 sia non nulla, per $h < 1/L = 1/10$, la differenza tra soluzione non perturbata e soluzione perturbata rimane limitata da S_2 .

8.4.3 Analisi di convergenza per il metodo di Eulero in avanti

Per verificare che il metodo di Eulero in avanti è convergente, scriviamo l'errore nel seguente modo

$$e_n = y_n - u_n = (y_n - u_n^*) + (u_n^* - u_n), \quad (8.24)$$

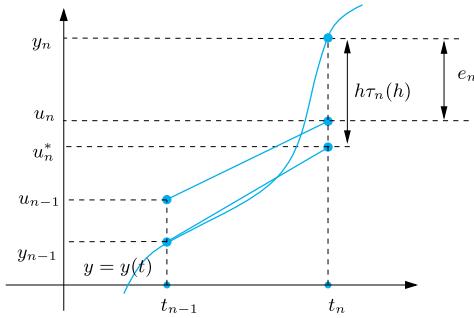


Figura 8.4. Rappresentazione geometrica di un passo del metodo di Eulero in avanti

dove

$$u_n^* = y_{n-1} + h f(t_{n-1}, y_{n-1}) \quad (8.25)$$

denota la soluzione numerica calcolata in t_n a partire dalla soluzione esatta al tempo t_{n-1} (si veda la Figura 8.4). Il termine $y_n - u_n^*$ nella (8.24) 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 $h \rightarrow 0$.

Grazie a (8.25) e (8.11) abbiamo

$$y_n - u_n^* = h \tau_n(h) \quad (8.26)$$

e, ricordando (8.13), concludiamo che il primo contributo dell'errore (8.24) tende a zero per $h \rightarrow 0$.

Consideriamo ora il secondo addendo della (8.24). Abbiamo

$$u_n^* - u_n = e_{n-1} + h [f(t_{n-1}, y_{n-1}) - f(t_{n-1}, u_{n-1})]. \quad (8.27)$$

Essendo f lipschitziana rispetto al suo secondo argomento, si trova

$$|u_n^* - u_n| \leq (1 + hL)|e_{n-1}|.$$

Poiché $u_0 = y_0$, abbiamo $e_0 = 0$ e l'errore al generico passo può essere così maggiorato:

$$\begin{aligned} |e_n| &\leq |y_n - u_n^*| + |u_n^* - u_n| \\ &\leq h|\tau_n(h)| + (1 + hL)|e_{n-1}| \\ &\leq [1 + (1 + hL) + \dots + (1 + hL)^{n-1}]h\tau(h) \\ &= \frac{(1 + hL)^n - 1}{L}\tau(h) \leq \frac{e^{L(t_n - t_0)} - 1}{L}\tau(h), \end{aligned} \quad (8.28)$$

dove abbiamo usato l'identità (8.21), la diseguaglianza $1 + hL \leq e^{hL}$ ed abbiamo osservato che $nh = t_n - t_0$. Ricordando (8.14), troviamo

$$|e_n| \leq \frac{e^{L(t_n-t_0)} - 1}{L} \frac{M}{2} h \quad \forall n = 0, \dots, N_h, \quad (8.29)$$

pertanto *il metodo di Eulero in avanti converge con ordine 1*. Come si nota l'ordine di convergenza 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.

Osservazione 8.1 L'analisi dell'errore in (8.28) ricalca i passaggi svolti per l'analisi della zero-stabilità in (8.20), più precisamente, l'errore di troncamento locale in (8.28) può essere interpretato come la perturbazione ρ_n di (8.20), tale che $\rho_0 = 0$.

Risulta quindi evidente che per garantire la convergenza del metodo di Eulero in avanti si debba richiedere che siano soddisfatte sia la proprietà di consistenza che la proprietà di zero-stabilità. In particolare, se la consistenza 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 $h \rightarrow 0$.

D'altro canto la zero-stabilità ci garantisce che l'errore tra la soluzione numerica e la soluzione esatta sia controllato dagli errori di troncamento locale introdotti ad ogni passo. ■

Esempio 8.1 Consideriamo il problema di Cauchy

$$\begin{cases} y'(t) = \cos(2y(t)) & t \in (0, 1], \\ y(0) = 0, \end{cases} \quad (8.30)$$

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 8.1) usando diversi valori di h , $h = 1/2, 1/4, 1/8, \dots, 1/512$:

```
tspan=[0,1]; y0=0; f=@(t,y) cos(2*y);
u=@(t) 0.5*asin((exp(4*t)-1)./(exp(4*t)+1));
Nh=2;
for k=1:10
    [tn,un]=feuler(f,tspan,y0,Nh);
    fe(k)=max(abs(un-u(tn)));
    Nh = 2*Nh;
end
```

Il massimo errore $\max_n |y_n - u_n|$ è memorizzato, per ogni h , nel vettore **fe**. Per stimare l'ordine di convergenza usiamo la formula (1.13). Tramite il comando seguente

```
p=log(abs(fe(1:end-1)/fe(2:end)))/log(2); p(1:2:end)
```

otteniamo

1.2194	1.0346	1.0086	1.0023	1.0006
--------	--------	--------	--------	--------

e possiamo verificare che il metodo converge con ordine 1. ■

Osservazione 8.2 (Effetto degli errori di arrotondamento) La stima dell'errore (8.29) è stata derivata supponendo che la soluzione numerica $\{u_n\}$ sia calcolata in aritmetica esatta. Se si dovesse tener conto degli (inevitabili) errori di arrotondamento, l'errore esploderebbe comportandosi come $1/h$ quando h tende a 0 (si veda, ad esempio, [QSSG14, Sez. 10.3.2]). Questa osservazione suggerisce che non è ragionevole prendere h al di sotto di un valore di soglia h^* (che è generalmente piccolissimo) nei calcoli. ■

8.4.4 Stimatori dell'errore per il metodo di Eulero in avanti

La stima di convergenza (8.29) è stata ottenuta richiedendo semplicemente che f sia continua e lipschitziana.

Tuttavia l'errore $|e_n|$ è eccessivamente sovrastimato in (8.29) e di fatto questa disuguaglianza non ci consente di misurare l'errore realmente commesso.

Consideriamo ad esempio il problema

$$\begin{cases} y'(t) = -15y(t) & t \in (0, T], \\ y(0) = 1, \end{cases} \quad (8.31)$$

(qui la costante di Lipschitz è $L = 15$) e sia $C_L(t_n) = (e^{L(t_n - t_0)} - 1)/L$ il fattore indipendente da h , ma dipendente da L e dall'ampiezza dell'intervallo (t_0, t_n) , che compare nella stima dell'errore (8.29). È immediato verificare che $C_L(t_n)$ assume valori molto elevati anche per tempi t_n moderati, ad esempio $C_L(1) \simeq 2 \cdot 10^5$, $C_L(10) \simeq 9 \cdot 10^{63}$ e $C_L(20) \simeq 10^{129}$. Per quantificare meglio quanto la stima (8.29) sia poco significativa definiamo la funzione

$$E(h) = \max_{0 \leq n \leq N_h} |e_n|, \quad (8.32)$$

(essendo la soluzione numerica calcolata con il metodo di Eulero in avanti con passo h) e gli stimatori

$$\begin{aligned} \mathcal{E}_1(h) &= \frac{e^{L(T-t_0)} - 1}{L} \frac{M}{2} h, \\ \mathcal{E}_2(h) &= \frac{(1 + hL)^{N_h+1} - 1}{L} \frac{M}{2} h \end{aligned} \quad (8.33)$$

che abbiamo incontrato nella sequenza di disuguaglianze (8.28) e per i quali vale $E(h) \leq \mathcal{E}_2(h) \leq \mathcal{E}_1(h)$ per ogni $h > 0$. Osservando i grafici di Figura 8.5, ottenuti prendendo $T = 10$, possiamo verificare che $\mathcal{E}_2(h)$ è una buona stima dell'errore $E(h)$ solo per h grandi, mentre $\mathcal{E}_1(h)$ maggiora l'errore vero $E(h)$ di ben 60 ordini di grandezza!

Una stima più accurata di (8.29) può essere provata nell'ipotesi che $\partial f(t, y)/\partial y$ sia definita e continua e che esista $\lambda_{\max} > 0$ tale che

$$-\lambda_{\max} < \frac{\partial f}{\partial y}(t, y) \leq 0 \quad \forall t \in [t_0, T], \quad \forall y \in \mathbb{R}.$$

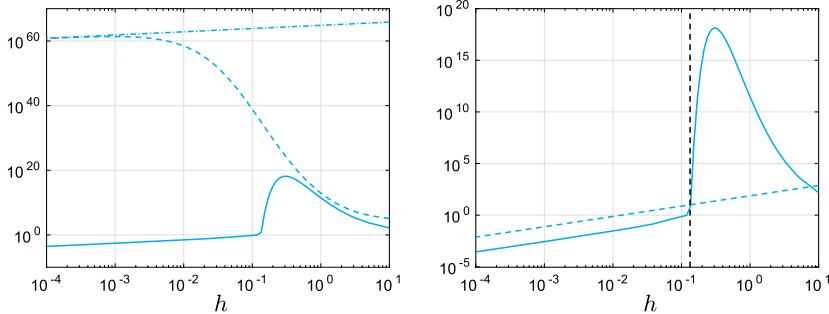


Figura 8.5. L'errore (8.32) (linea continua) del metodo di Eulero in avanti per risolvere il problema (8.31) sull'intervallo $(0, 10)$ e gli stimatori $E_1(h)$ (linea tratto punto) ed $E_2(h)$ (linea tratteggiata) definiti in (8.33) (a sinistra). L'errore (8.32) (linea continua) del metodo di Eulero in avanti per risolvere il problema (8.31) sull'intervallo $(0, 10)$ e lo stimatore $E_3(h)$ (linea tratteggiata) definito in (8.35) (a destra). La linea verticale tratteggiata nel grafico di destra rappresenta il valore $2/\lambda_{\max} = 2/15$

Infatti sotto tali ipotesi, per il teorema del valor medio esiste un punto η_{n-1} tra y_{n-1} e u_{n-1} tale che

$$f(t_{n-1}, y_{n-1}) - f(t_{n-1}, u_{n-1}) = \frac{\partial f}{\partial y}(t_{n-1}, \eta_{n-1})(y_{n-1} - u_{n-1}),$$

cosicché

$$u_n^* - u_n = \left(1 + h \frac{\partial f}{\partial y}(t_{n-1}, \eta_{n-1})\right) e_{n-1}.$$

Inoltre, se²

$$0 < h < \frac{2}{\lambda_{\max}} \quad (8.34)$$

si ha $|1 + h \partial f(t, y)/\partial y| < 1$ e, grazie alla (8.14) ed al fatto che $e_0 = 0$, otteniamo

$$|e_n| \leq |y_n - u_n^*| + |e_{n-1}| \leq \dots \leq nh\tau(h) + |e_0| \leq (t_n - t_0) \frac{M}{2}h.$$

Definiamo quindi un nuovo stimatore dell'errore

$$\mathcal{E}_3(h) = (T - t_0)\tau(h) = (T - t_0)\frac{M}{2}h. \quad (8.35)$$

Osservando i grafici di Figura 8.5, a destra, possiamo verificare che $E(h) \leq \mathcal{E}_3(h)$ per ogni h che soddisfa la limitazione (8.34) (a sinistra)

² La limitazione (8.34) sul passo h è di fatto una *condizione di stabilità assoluta* (si veda la Sezione 8.5). Sottolineiamo il fatto che, sebbene l'assoluta stabilità sarà studiata su un intervallo $I = (t_0, \infty)$ illimitato, la restrizione (8.34) risulta utile anche per una scelta di h nel regime di convergenza su intervalli limitati.

della linea nera tratteggiata) e che $\mathcal{E}_3(h)$ è molto più significativo di entrambi $\mathcal{E}_1(h)$ ed $\mathcal{E}_2(h)$.

Il valore $\max_h E(h)$ dipende dall'ampiezza dell'intervallo $(t_0, T]$ in cui si approssima la soluzione del problema di Cauchy oltre che dall'espressione della funzione f .

8.4.5 Analisi di convergenza per metodi ad un passo

I metodi di Eulero ed il metodo di Crank-Nicolson possono essere riscritti nella forma più generale

$$u_{n+1} = u_n + h \Phi(t_n, h, u_n, u_{n+1}; f), \quad n = 0, \dots, N_h - 1 \quad (8.36)$$

dove

$$\Phi(t_n, h, u_n, u_{n+1}; f) = \begin{cases} f(t_n, u_n) & \text{in (8.6)} \\ f(t_{n+1}, u_{n+1}) & \text{in (8.7)} \\ \frac{1}{2}[f(t_n, u_n) + f(t_{n+1}, u_{n+1})] & \text{in (8.8)} \end{cases}$$

è detta *funzione di incremento*.

Molti altri metodi possono essere scritti nella forma (8.36), un esempio notevole è dato dai metodi Runge-Kutta che vedremo nella Sezione 8.9. Nel caso generale si chiede che la funzione Φ soddisfi queste due proprietà:

$$\begin{aligned} \Phi(t_n, h, u_n, u_{n+1}; 0) &= 0, \\ \exists L_\Phi > 0 : |\Phi(t_n, h, u_n, u_{n+1}; f) - \Phi(t_n, h, w_n, w_{n+1}; f)| &\leq L_\Phi \sum_{k=0,1} |u_{n+k} - w_{n+k}|, \end{aligned} \quad (8.37)$$

dove la costante L_Φ può dipendere da f , ma non da h , $\{u_n\}$, $\{w_n\}$, e t_n . Di fatto (8.37) è una proprietà di lipschitzianità di Φ rispetto agli argomenti u_n e u_{n+1} e dipende a sua volta dalla lipschitzianità di f rispetto al suo secondo argomento.

Vale la seguente proprietà, per la cui dimostrazione rimandiamo, ad esempio, a [Lam91, Sez. 2.5]:

Teorema 8.1 *Un metodo ad un passo (8.36), la cui funzione Φ soddisfi (8.37), è zero-stabile.*

Osserviamo che questa proprietà non è garantita per metodi più generali, come ad esempio alcuni metodi *multistep* che vedremo nella Sezione 8.9.2.

Infine, il seguente teorema, fondamentale nella teoria dei metodi numerici per l'approssimazione di equazioni differenziali ordinarie (si veda, ad esempio, [IK66, Sez. 8.5]), fornisce un criterio per stabilire quando un metodo è convergente:

Teorema 8.2 *Un metodo ad un passo (8.36) che sia consistente e la cui funzione di incremento Φ soddisfi (8.37) è convergente.*

Grazie a questi due teoremi possiamo concludere che anche i metodi di Eulero all'indietro e di Crank-Nicolson sono convergenti. Infatti è immediato verificare che la funzione Φ ad essi associate soddisfa le proprietà (8.37) e quindi i metodi, oltre ad essere consistenti (si veda la Sezione 8.4) sono anche zero-stabili e convergenti.

Inoltre si verifica che per entrambi i metodi l'ordine di convergenza coincide con l'ordine di consistenza, per cui il metodo di Eulero all'indietro è convergente di ordine 1 rispetto a h ed il metodo di Crank-Nicolson è convergente con ordine 2 rispetto a h .

Esempio 8.2 Risolviamo il problema (8.30) con i metodi di Eulero all'indietro e di Crank-Nicolson, con gli stessi valori di h usati nell'Esempio 8.1. Come si vede, i risultati confermano che l'errore per il metodo di Eulero all'indietro tende a zero con ordine 1 rispetto a h , mentre l'errore per il metodo di Crank-Nicolson tende a zero con ordine 2 rispetto a h :

```
y0=0; tspan=[0 1]; f=@(t,y) cos(2*y); Nh=2;
y=@(t) 0.5*asin((exp(4*t)-1)./(exp(4*t)+1));
for k=1:10
    [tn,une]=beuler(f,tspan,y0,Nh);
    be(k)=max(abs(une-u(tn)));
    [tn,uncn]=cranknic(f,tspan,y0,Nh);
    cne(k)=max(abs(uncn-y(tn)));
    Nh=2*Nh;
end
```

Gli errori $\max_n |y_n - u_n|$ sono memorizzati nei vettori **be** (per il metodo di Eulero all'indietro) e **cne** (per quello di Crank-Nicolson). Per stimare l'ordine di convergenza usiamo sempre la formula (1.13). Tramite i comandi seguenti

```
p=log(abs(be(1:end-1)./be(2:end)))/log(2); p(1:2:end)
```

otteniamo i valori

0.8770	0.9649	0.9908	0.9978	0.9994
--------	--------	--------	--------	--------

per il metodo di Eulero all'indietro e

```
p=log(abs(cne(1:end-1)./cne(2:end)))/log(2); p(1:2:end)
```

1.9627	1.9986	2.0001	1.9999	2.0000
--------	--------	--------	--------	--------

per il metodo di Crank-Nicolson.

Si vedano gli Esercizi 8.1–8.5.



8.5 Stabilità su intervalli illimitati

Nella precedente Sezione ci siamo occupati dell'approssimazione 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 la stima dell'errore (8.29) o la stima di stabilità (8.16) con C data in (8.19) perdono di significato in quanto a secondo membro compaiono quantità illimitate (si vedano gli stimatori (8.33)). 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 su intervalli temporali arbitrariamente grandi. Sfortunatamente il metodo di Eulero in avanti, di così semplice implementazione, non gode di questa proprietà.

Introduciamo il seguente problema di Cauchy lineare detto *problema modello*

$$\begin{cases} y'(t) = \lambda y(t), & t \in (0, \infty), \\ y(0) = 1, \end{cases} \quad (8.38)$$

dove λ è un numero reale negativo. La soluzione esatta è $y(t) = e^{\lambda t}$ e tende a 0 per t che tende all'infinito. Per h fissato, chiediamo che la soluzione numerica sia in grado di riprodurre l'andamento della soluzione esatta quando t_n tende all'infinito. Più precisamente diciamo che un metodo numerico per l'approssimazione del problema di Cauchy (8.5) soddisfa l'*assoluta stabilità* per un valore fissato di h se la soluzione u_n calcolata con tale h verifica

$$\lim_{n \rightarrow \infty} u_n = 0. \quad (8.39)$$

Se applichiamo a (8.38) il metodo di Eulero in avanti, troviamo

$$u_0 = 1, \quad u_{n+1} = u_n(1 + \lambda h) = (1 + \lambda h)^{n+1}, \quad n \geq 0. \quad (8.40)$$

Avremo $\lim_{n \rightarrow \infty} u_n = 0$ se e solo se

$$-1 < 1 + h\lambda < 1, \quad \text{ovvero } h < \bar{h}(\lambda) = 2/|\lambda| \quad (8.41)$$

Se $h > 2/|\lambda|$, allora $\lim_{n \rightarrow \infty} |u_n| = +\infty$, violando pertanto la proprietà (8.39).

Esempio 8.3 Risolviamo con il metodo di Eulero in avanti il problema (8.38) con $\lambda = -1$. In tal caso dobbiamo avere $h < 2$ per garantire l'assoluta stabilità.

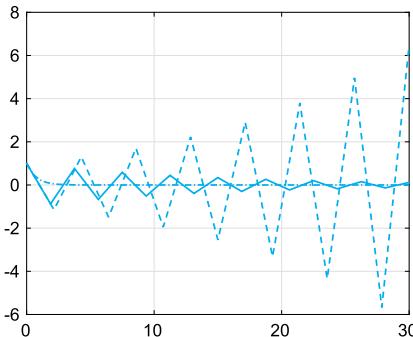


Figura 8.6. Soluzioni del problema (8.38), con $\lambda = -1$, ottenute con il metodo di Eulero in avanti, corrispondenti a $h = 30/14(> 2)$ (linea tratteggiata), $h = 30/16(< 2)$ (linea continua) e $h = 1/2$ (in linea tratto-punto)

In Figura 8.6 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 nei primi due casi la soluzione oscilla. Tuttavia solo 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 la Sezione 8.5.1) o quando $\lambda = \lambda(t)$ in (8.38) è una funzione negativa di t . In quest’ultimo caso, nella condizione di stabilità (8.41), $|\lambda|$ dovrà essere sostituito da $\max_{t \in [0, \infty)} |\lambda(t)|$. Questa condizione, che nella pratica può risultare molto restrittiva, può essere indebolita se si utilizza un passo variabile \bar{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*:

sia $\alpha < 1$ una costante assegnata, posto $u_0 = y_0$ e $\bar{h}_0 = 2\alpha/|\lambda(t_0)|$, allora

per $n = 0, 1, \dots$, calcolare

$$\begin{aligned} t_{n+1} &= t_n + \bar{h}_n, \\ u_{n+1} &= u_n + \bar{h}_n \lambda(t_n) u_n, \\ \bar{h}_{n+1} &= 2\alpha/|\lambda(t_n)|. \end{aligned} \tag{8.42}$$

La richiesta che α sia minore di 1 garantisce che il metodo sia assolutamente stabile.

Esempio 8.4 Consideriamo il problema

$$\begin{cases} y'(t) = -(e^{-t} + 1)y(t), & t \in (0, 4), \\ y(0) = e \end{cases} \tag{8.43}$$

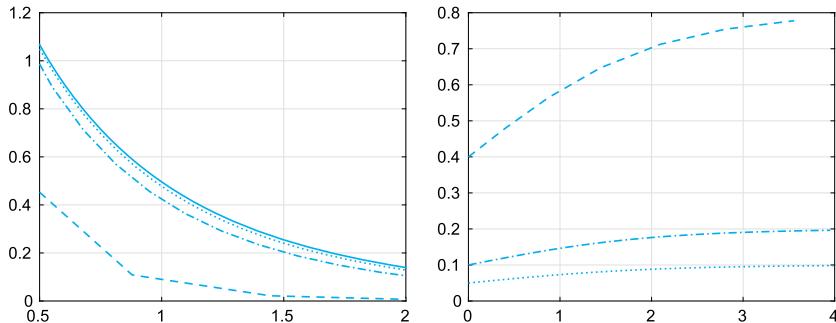


Figura 8.7. A sinistra, le soluzioni numeriche dell’Esempio 8.4 ristrette all’intervallo temporale $(0.5, 2)$ ottenute con il metodo di Eulero in avanti a passo costante $h = 0.04$ (in linea continua) e con il metodo adattivo (8.42) per tre diversi valori di α : $\alpha = 0.4$ (linea tratteggiata), $\alpha = 0.1$ (linea tratto punto), $\alpha = 0.05$ (linea punteggiata). A destra, l’andamento del passo variabile \bar{h}_n per il metodo adattivo (8.42). I tre tipi di linea corrispondono ai valori di α come nella figura a sinistra

Poiché $|\lambda(t)|$ è decrescente, la condizione più restrittiva per garantire assoluta stabilità del metodo di Eulero in avanti è $h < 2/|\lambda(0)| = 1$. In Figura 8.7, a sinistra, sono rappresentate le soluzioni ottenute con il metodo di Eulero in avanti con passo fisso $h = 0.04$ e quelle ottenute con il metodo adattivo (8.42) 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. Infatti, denotando con $\{u_n^{(\alpha)}\}$ la soluzione numerica ottenuta con un valore di α fissato e con $e^{(\alpha)} = \max_n |y_n - u_n^{(\alpha)}|$ l’errore rispetto alla soluzione esatta, otteniamo $e^{(0.4)} \simeq 0.767$ con 8 passi, $e^{(0.1)} \simeq 0.133$ con 27 passi e $e^{(0.05)} \simeq 0.064$ con 52 passi. In Figura 8.7, a destra riportiamo i valori di \bar{h}_n per i tre valori di α . Da questo grafico si deduce che la successione $\{\bar{h}_n\}$ è monotona crescente in quanto $|\lambda(t)|$ è decrescente. ■

Nella Sezione 8.8 vedremo un’altra strategia adattiva per la scelta del passo, basata sul controllo dell’errore di troncamento locale.

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 (8.38) infatti si trova $u_{n+1} = u_n + \lambda h u_{n+1}$ e

$$u_{n+1} = \left(\frac{1}{1 - \lambda h} \right)^{n+1}, \quad n \geq 0, \quad (8.44)$$

che tende a zero per $n \rightarrow \infty$ per tutti i valori $h > 0$. Analogamente applicando il metodo di Crank-Nicolson si trova

$$u_{n+1} = \left[\left(1 + \frac{h\lambda}{2} \right) / \left(1 - \frac{h\lambda}{2} \right) \right]^{n+1}, \quad n \geq 0, \quad (8.45)$$

che tende nuovamente a zero per $n \rightarrow \infty$ per tutti i possibili valori di $h > 0$.

8.5.1 La regione di assoluta stabilità

Supponiamo ora che nel problema (8.38) λ sia un numero complesso con parte reale negativa. In tal caso la soluzione $y(t) = e^{\lambda t}$ è una funzione a valori complessi ed entrambe la sua parte reale e la sua parte immaginaria tendono 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 $\lim_{n \rightarrow \infty} |u_n| = 0$ (si tenga presente che u_n dipende sia da h che da λ):

$$\mathcal{A} = \{z = h\lambda \in \mathbb{C} : |u_n| \rightarrow 0 \text{ per } t_n \rightarrow 0\}. \quad (8.46)$$

Ad esempio, in virtù di (8.40), troviamo che la regione di assoluta stabilità del metodo di Eulero in avanti è costituita dai valori $z = h\lambda \in \mathbb{C}$ tali che $|1+h\lambda| < 1$, ovvero dal cerchio di raggio unitario e centro $(-1, 0)$. Esplicitando questa condizione rispetto a h si ottiene la limitazione

$$0 < h < \bar{h}(\lambda) = \frac{-2\operatorname{Re}(\lambda)}{|\lambda|^2} \quad (8.47)$$

che è una generalizzazione di (8.41) al caso in cui $\lambda \in \mathbb{C}$.

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 8.8). 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.

Un metodo che risulti assolutamente stabile per tutti i numeri complessi $z = h\lambda \in \mathbb{C}$ con $\operatorname{Re}(z) < 0$ è detto *incondizionatamente assolutamente stabile* o *A-stabile*, altrimenti è detto *condizionatamente assolutamente stabile*.

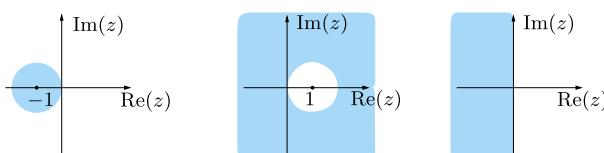


Figura 8.8. Le regioni di assoluta stabilità (*in colore*) per i metodi di Eulero in avanti (*a sinistra*), di Eulero all'indietro (*al centro*) e di Crank-Nicolson (*a destra*)

Quindi il metodo di Eulero in avanti è *condizionatamente assolutamente stabile*, mentre i metodi di Eulero all'indietro e di Crank-Nicolson sono 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 esplicativi.

Esempio 8.5 Calcoliamo la restrizione a cui deve essere soggetta h qualora si utilizzi il metodo di Eulero in avanti per la risoluzione del problema di Cauchy $y'(t) = \lambda y$ con $\lambda = -1 + i$. Questo valore di λ 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 la restrizione più severa $h \in (0, 1/2)$. ■

8.6 L-stabilità

Risolviamo il problema lineare modello (8.38) sull'intervallo $(0, 4)$, fissando $\lambda = -300$, con i metodi di Eulero all'indietro e di Crank-Nicolson. In Figura 8.9 sono riportate le soluzioni numeriche ottenute con passo $h = 0.1$ (a sinistra) e con passo $h = 0.005$ (a destra).

Abbiamo visto nella Sezione precedente che entrambi i metodi sono A-stabili, quindi le oscillazioni che presenta la soluzione del metodo di Crank-Nicolson per $h = 0.1$ non sono dovute ad assenza di assoluta stabilità (la soluzione numerica comunque tende a zero per $t_n \rightarrow \infty$!).

Per capire meglio perché, a differenza del metodo di Eulero all'indietro, la soluzione del metodo di Crank-Nicolson presenta delle oscillazioni, riscriviamo la soluzione esatta del problema (8.38) ricordando che $t_n = t_0 + hn$, $t_0 = 0$ e $y(0) = 1$:

$$y_{n+1} = y(t_{n+1}) = e^{\lambda t_{n+1}} = e^{t_0} e^{\lambda h(n+1)} = (e^{h\lambda})^{n+1}, \quad n \geq 0. \quad (8.48)$$

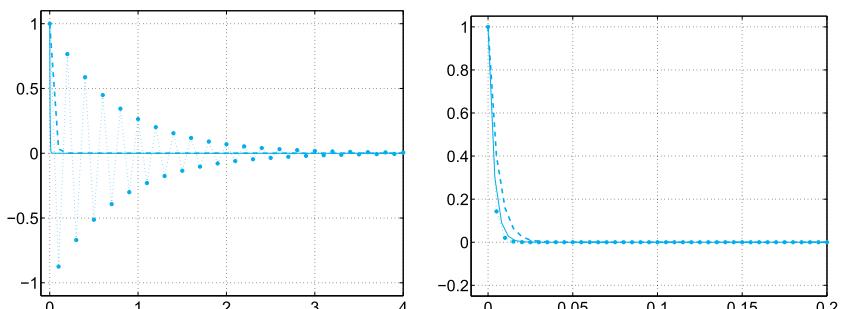


Figura 8.9. Le soluzioni del problema (8.38) con $\lambda = -300$ e passo $h = 0.1$ (a sinistra), $h = 0.005$ (a destra): la soluzione esatta (linea continua), la soluzione del metodo di Eulero in avanti (linea tratteggiata), la soluzione del metodo di Crank-Nicolson (pallini)

Ora riprendiamo l'espressione delle soluzioni (8.44) e (8.45) ottenute rispettivamente con i metodi di Eulero all'indietro e di Crank-Nicolson. È evidente che le soluzioni numeriche u_{n+1} calcolate con (8.44) e (8.45) approssimeranno bene la soluzione esatta (8.48) a patto che

$$\mathcal{R}_{BE}(h\lambda) = \frac{1}{1 - h\lambda} \quad \text{e} \quad \mathcal{R}_{CN}(h\lambda) = \frac{2 + h\lambda}{2 - h\lambda} \quad (8.49)$$

siano buone approssimazioni di $e^{h\lambda}$.

Se $|\operatorname{Re}(\lambda)|$ è abbastanza grande e h non è troppo piccolo, abbiamo

$$e^{h\lambda} \sim 0, \quad \mathcal{R}_{BE}(h\lambda) \sim 0, \quad \text{ma} \quad \mathcal{R}_{CN}(h\lambda) \sim -1(!) \quad (8.50)$$

Ne segue che, mentre la soluzione del metodo di Eulero all'indietro descrive bene il comportamento della soluzione esatta anche per i primi valori di n , la soluzione prodotta dal metodo di Crank-Nicolson presenta nella prima parte dell'intervallo temporale delle oscillazioni che si smorzano al crescere di n . Queste oscillazioni sono tanto più ampie quanto più $|\operatorname{Re}(h\lambda)|$ è grande e tendono a scomparire per $|\operatorname{Re}(h\lambda)|$ piccoli (come nel caso di Figura 8.9, a destra). Infatti, quando $h \rightarrow 0$, si ha $\mathcal{R}_{BE}(h\lambda) \rightarrow 1$ e $\mathcal{R}_{CN}(h\lambda) \rightarrow 1$ ed entrambe sono buone approssimazioni di $e^{h\lambda}$, anch'esso tendente a 1.

Diamo allora la seguente definizione ([Lam91]): un metodo ad un passo A-stabile è detto L-stabile se, quando è applicato al problema di Cauchy lineare modello (8.38) con $\lambda \in \mathbb{C}$ e $\operatorname{Re}(\lambda) < 0$, esso genera una successione ricorsiva della forma $u_{n+1} = \mathcal{R}(h\lambda)u_n$ con

$$|\mathcal{R}(h\lambda)| \rightarrow 0 \quad \text{quando } \operatorname{Re}(h\lambda) \rightarrow -\infty \quad (8.51)$$

Il metodo di Eulero all'indietro è quindi L-stabile, in quanto

$$|\mathcal{R}_{BE}(h\lambda)| = 1/|1 - h\lambda| \rightarrow 0 \quad \text{per } \operatorname{Re}(h\lambda) \rightarrow -\infty,$$

mentre il metodo di Crank-Nicolson non lo è in quanto

$$|\mathcal{R}_{CN}(h\lambda)| = |2 + h\lambda|/|2 - h\lambda| \rightarrow 1 \quad \text{per } \operatorname{Re}(h\lambda) \rightarrow -\infty.$$

I metodi L-stabili sono quindi in grado di catturare bene forti variazioni della soluzione del problema differenziale che si verificano quando $\operatorname{Re}(\lambda) \ll 0$ (ovvero comportamenti transitori) anche con passi di discretizzazione moderati. Di fatto questi metodi sono intrinsecamente dissipativi. Al contrario, i metodi che non sono L-stabili possono approssimare forti gradienti della soluzione solo prendendo h sufficientemente piccoli, come si può evincere dalla Figura 8.9, a destra. In genere, i metodi che non siano L-stabili vengono implementati seguendo una strategia adattiva per la scelta del passo h : là dove la soluzione presenta forti variazioni sarà usato un passo piccolo, mentre dove la soluzione è meno variabile potranno essere utilizzati passi di ampiezza maggiore. Strategie per la scelta adattiva del passo verranno presentate nelle prossime sezioni.

8.7 L'assoluta stabilità controlla le perturbazioni

Consideriamo il seguente *problema modello generalizzato*

$$\begin{cases} y'(t) = \lambda(t)y(t) + r(t), & t \in (0, +\infty), \\ y(0) = 1, \end{cases} \quad (8.52)$$

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 (8.52). D'altra parte, mostreremo che un metodo numerico che sia assolutamente stabile per il problema modello (8.38), quando applicato al problema generalizzato (8.52) 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 (8.52)

$$\begin{cases} u_{n+1} = u_n + h(\lambda_n u_n + r_n), & n \geq 0, \\ u_0 = 1. \end{cases}$$

La soluzione è (si veda l'Esercizio 8.7)

$$u_n = u_0 \prod_{k=0}^{n-1} (1 + h\lambda_k) + h \sum_{k=0}^{n-1} r_k \prod_{j=k+1}^{n-1} (1 + h\lambda_j), \quad (8.53)$$

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

$$\begin{cases} z_{n+1} = z_n + h(\lambda_n z_n + r_n + \rho_{n+1}), & n \geq 0, \\ z_0 = u_0 + \rho_0, \end{cases} \quad (8.54)$$

dove ρ_0, ρ_1, \dots sono perturbazioni note che vengono introdotte ad ogni passo temporale. La soluzione di (8.54) assume una forma simile a (8.53) 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

$$z_n - u_n = \rho_0 \prod_{k=0}^{n-1} (1 + h\lambda_k) + h \sum_{k=0}^{n-1} \rho_{k+1} \prod_{j=k+1}^{n-1} (1 + h\lambda_j). \quad (8.55)$$

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 $h < \bar{h}(\lambda) = 2/|\lambda|$; ricordiamo che questa condizione su h assicura l'assoluta stabilità del metodo di Eulero in avanti quando applicato al problema modello (8.38). Utilizzando la seguente identità

$$\sum_{k=0}^{n-1} a^k = \frac{1 - a^n}{1 - a}, \quad \text{se } a \neq 1, \quad (8.56)$$

troviamo

$$z_n - u_n = \rho \left\{ (1 + h\lambda)^n \left(1 + \frac{1}{\lambda} \right) - \frac{1}{\lambda} \right\}. \quad (8.57)$$

Segue che l'errore di perturbazione soddisfa (si veda l'Esercizio 8.8)

$$|z_n - u_n| \leq \varphi(\lambda)|\rho|, \quad (8.58)$$

con $\varphi(\lambda) = 1$ se $\lambda \leq -1$, mentre $\varphi(\lambda) = |1 + 2/\lambda|$ se $-1 < \lambda < 0$. Possiamo quindi concludere che l'errore di perturbazione è limitato da $|\rho|$ per una costante che dipende dal dato λ del problema, ma che è indipendente da n e h . Inoltre, dalla (8.57), deduciamo

$$\lim_{n \rightarrow \infty} |z_n - u_n| = \frac{|\rho|}{|\lambda|}.$$

La Figura 8.10 corrisponde al caso in cui $r(t) \equiv 0$, $\rho = 0.1$, $\lambda = -2$ (sinistra) e $\lambda = -0.5$ (destra) avendo preso $h < \bar{h}(\lambda)$. Più precisamente abbiamo considerato $h = 100/101 < \bar{h}(\lambda) = 1$ per $\lambda = -2$ e $h = 1000/251 < \bar{h}(\lambda) = 4$ per $\lambda = -4$. Si noti che la stima (8.58) è esattamente verificata. Naturalmente, l'errore di perturbazione esplode al crescere di n se si viola la limitazione $h < \bar{h}(\lambda)$.

Osservazione 8.3 Se l'unica perturbazione fosse quella sul dato iniziale, ovvero se $\rho_k = 0$, $k = 1, 2, \dots$, dalla (8.55) dedurremmo che $\lim_{n \rightarrow \infty} |z_n - u_n| = 0$ sotto la condizione di stabilità $h < \bar{h}(\lambda)$. ■

ii. Nel caso generale quando λ e r non sono costanti, richiediamo che h soddisfi la restrizione $h < \bar{h}(\lambda)$, dove stavolta $\bar{h}(\lambda) = 2/\lambda_{\max}$. Allora,

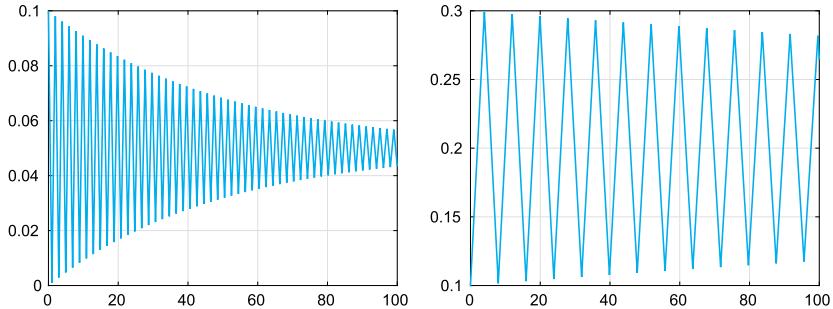


Figura 8.10. L'errore di perturbazione quando $r(t) \equiv 0$, $\rho = 0.1$, $\lambda = -2$ e $h = 100/101 < \bar{h}(\lambda) = 1$ (a sinistra) e $\lambda = -0.5$ e $h = 1000/251 < \bar{h}(\lambda) = 4$ (a destra)

$$|1 + h\lambda_k| \leq a(h) = \max\{|1 - h\lambda_{\min}|, |1 - h\lambda_{\max}|\}.$$

Essendo $0 < \frac{\lambda_{\max} - \lambda_{\min}}{\lambda_{\max} + \lambda_{\min}} \leq a(h) < 1$, possiamo usare ancora l'identità (8.56) in (8.55) ricavando

$$|z_n - u_n| \leq \bar{\rho} \left([a(h)]^n + h \frac{1 - [a(h)]^n}{1 - a(h)} \right), \quad (8.59)$$

dove $\bar{\rho} = \sup_k |\rho_k|$.

Supponiamo dapprima che $h \leq h^* = 2/(\lambda_{\min} + \lambda_{\max})$, nel qual caso $a(h) = (1 - h\lambda_{\min})$. Si ha

$$|z_n - u_n| \leq \frac{\bar{\rho}}{\lambda_{\min}} [1 - [a(h)]^n (1 - \lambda_{\min})], \quad (8.60)$$

ovvero

$$\sup_n |z_n - u_n| \leq \frac{\bar{\rho}}{\lambda_{\min}} \sup_n [1 - [a(h)]^n (1 - \lambda_{\min})].$$

Se $\lambda_{\min} = 1$, si ottiene

$$\sup_n |z_n - u_n| \leq \bar{\rho}. \quad (8.61)$$

Se $\lambda_{\min} < 1$, si ha $1 - [a(h)]^n (1 - \lambda_{\min}) < 1$ e

$$\sup_n |z_n - u_n| \leq \frac{\bar{\rho}}{\lambda_{\min}}. \quad (8.62)$$

Infine, se $\lambda_{\min} > 1$, la successione $b_n = 1 - [a(h)]^n (1 - \lambda_{\min})$ è monotona decrescente e $\sup_n b_n = b_0 = \lambda_{\min}$, dunque si ottiene di nuovo la (8.61).

Quando invece $h^* < h < \bar{h}(\lambda)$, si ha

$$1 + h\lambda_k = 1 - h|\lambda_k| \leq 1 - h^*|\lambda_k| \leq 1 - h^*\lambda_{\min}. \quad (8.63)$$

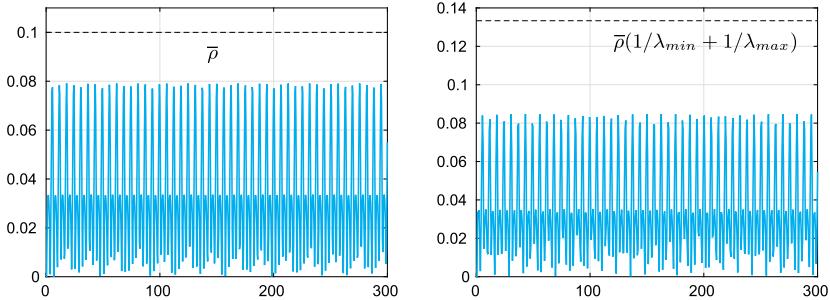


Figura 8.11. L'errore di perturbazione quando $r(t) \equiv 0$, $\rho(t) = 0.1 \sin(t)$ e $\lambda(t) = -2 - \sin(t)$ per $t \in (0, nh)$ con $n = 500$: il passo di discretizzazione è $h = h^* - 0.1 = 0.4$ (a sinistra) e $h = h^* + 0.1 = 0.6$ (a destra). In questo caso $\lambda_{\min} = 1$, per cui vale la stima (8.61) quando $h \leq h^*$, mentre vale (8.66) quando $h > h^*$

Utilizzando la (8.63) e l'identità (8.56) in (8.55) si trova, ponendo $a = 1 - h^* \lambda_{\min}$,

$$\begin{aligned} z_n - u_n &\leq \bar{\rho} \left(a^n + h \frac{1 - a^n}{1 - a} \right) \\ &= \frac{\bar{\rho}}{\lambda_{\min}} \left(a^n \left(\lambda_{\min} - \frac{h}{h^*} \right) + \frac{h}{h^*} \right). \end{aligned} \quad (8.64)$$

Ora osserviamo che si hanno due possibili situazioni.

Se $\lambda_{\min} \geq \frac{h}{h^*}$, allora $\frac{h}{h^*} \leq a^n \left(\lambda_{\min} - \frac{h}{h^*} \right) + \frac{h}{h^*} < \lambda_{\min}$ e troviamo

$$z_n - u_n \leq \bar{\rho} \quad \forall n \geq 0. \quad (8.65)$$

Se invece $\lambda_{\min} < \frac{h}{h^*}$, allora $\lambda_{\min} \leq a^n \left(\lambda_{\min} - \frac{h}{h^*} \right) + \frac{h}{h^*} < \frac{h}{h^*}$ e dunque

$$z_n - u_n \leq \frac{\bar{\rho}}{\lambda_{\min}} \frac{h}{h^*} \leq \frac{\bar{\rho}}{\lambda_{\min}} \frac{\bar{h}(\lambda)}{h^*} = \bar{\rho} \left(\frac{1}{\lambda_{\min}} + \frac{1}{\lambda_{\max}} \right). \quad (8.66)$$

In Figura 8.11 riportiamo gli errori di perturbazione calcolati per il problema (8.52), dove $r(t) \equiv 0$, $\lambda_k = \lambda(t_k) = -2 - \sin(t_k)$, $\rho_k = \rho(t_k) = 0.1 \sin(t_k)$ con $h \leq h^*$ (a sinistra) e con $h^* < h < \bar{h}(\lambda)$ (a destra). Si noti che il termine di destra di (8.66) maggiora in realtà anche il valore assoluto di $z_n - u_n$.

iii. Consideriamo ora il problema di Cauchy (8.5) con $f(t, y(t))$ generica. La sua soluzione numerica può essere messa in relazione con quella del problema modello generalizzato (8.52) nei casi in cui si abbia

$$-\lambda_{\max} < \frac{\partial f}{\partial y}(t, y) < -\lambda_{\min} \quad \forall t \geq 0 \quad \forall y \in (-\infty, \infty), \quad (8.67)$$

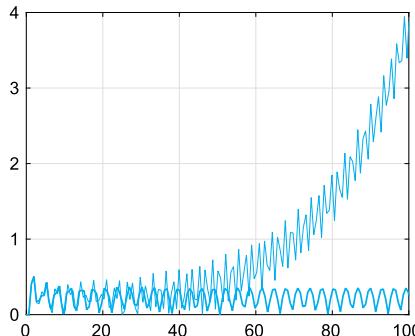


Figura 8.12. L'errore di perturbazione quando $\rho(t) = \sin(t)$ con $h = 1000/1522 \simeq 0.657 < 2/3$ (linea spessa) e $h = 1000/1477 \simeq 0.677 > 2/3$ (linea sottile) per il problema di Cauchy (8.68); $\bar{h}(\lambda_{\max}) = 2/3$

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 (8.6) da (8.18) 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 ancora l'identità (8.55), a partire dalla quale giungiamo alle stesse conclusioni ottenute al punto *ii.*, purché valga la condizione di stabilità $0 < h < 2/\lambda_{\max}$. Si noti che questa è esattamente la condizione (8.34).

Esempio 8.6 Consideriamo il problema di Cauchy

$$y'(t) = \arctan(3y) - 3y + t, \quad t > 0, \quad y(0) = 1. \quad (8.68)$$

Essendo $f_y = 3/(1+9y^2) - 3$ negativa, possiamo scegliere $\lambda_{\max} = \max |f_y| = 3$ e porre $h < \bar{h}(\lambda_{\max}) = 2/3$. 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 8.12. Si noti che in questo esempio, prendendo $h = 1000/1477 \simeq 0.677 > 2/3$ (che viola la precedente condizione di stabilità) l'errore di perturbazione esplode al crescere di t . ■

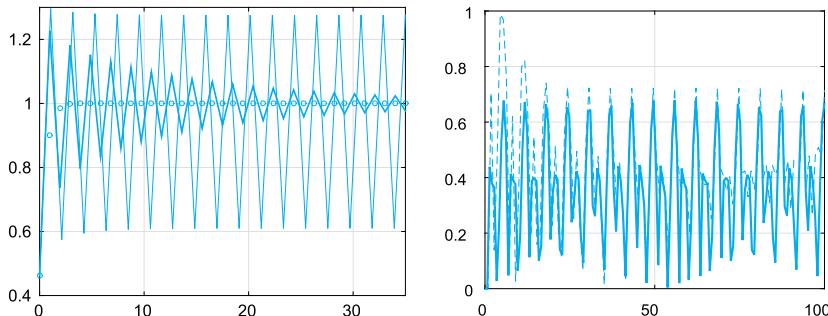


Figura 8.13. A sinistra, le soluzioni numeriche del problema (8.69) calcolate con il metodo di Eulero in avanti per $h = 1.05$ (in linea sottile) e per $h = 0.95$ (in linea spessa). I valori della soluzione esatta sono stati indicati con dei cerchietti. A destra, gli errori di perturbazione corrispondenti a $\rho(t) = \sin(t)$ per $h = h^* = 2/2.9$ (in linea spessa continua) e $h = 0.9$ (in linea sottile tratteggiata)

Esempio 8.7 Cerchiamo una limitazione superiore per h che garantisca la stabilità per il metodo di Eulero in avanti applicato al problema di Cauchy

$$y' = 1 - y^2, \quad t > 0, \quad (8.69)$$

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 $h < 1$. In Figura 8.13, a sinistra, riportiamo le soluzioni ottenute sull'intervallo $(0, 35)$ con $h = 0.95$ (linea spessa) e $h = 1.05$ (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 8.13, a destra, riportiamo gli errori di perturbazione corrispondenti a $\rho(t) = \sin(t)$ con $h = h^* = 2/2.9$ (linea spessa continua) e $h = 0.9$ (linea sottile tratteggiata). In entrambi i casi gli errori di perturbazione si mantengono limitati, in particolare per $h = h^* = 2/2.9$ è soddisfatta la stima (8.61), mentre per $h^* < h = 0.9$ è soddisfatta la stima (8.66). ■

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

$$0 < h_n < 2 \frac{\alpha}{|f_y(t_n, u_n)|}, \quad (8.70)$$

per opportuni valori di α strettamente minori di 1. Si noti che il denominatore dipende dal valore u_n che è noto. In Figura 8.14 riportiamo gli errori di perturbazione corrispondenti all'Esempio 8.7 per due diversi valori di α .

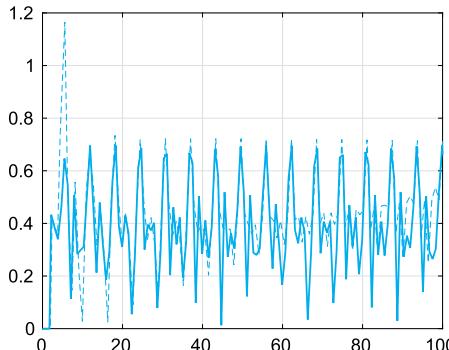


Figura 8.14. Gli errori di perturbazione corrispondenti a $\rho(t) = \sin(t)$ con $\alpha = 0.8$ (in linea spessa) e $\alpha = 0.9$ (in linea sottile) per l’Esempio 8.7, usando la strategia adattiva

L’analisi precedente può essere condotta anche per altri metodi, in particolare per quelli di Eulero all’indietro e di Crank-Nicolson. Per questi metodi, che sono A-stabili, si ricavano le stesse conclusioni sul 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.

8.8 Adattività del passo per il metodo di Eulero in avanti

Nelle sezioni precedenti abbiamo visto come il passo di discretizzazione h debba essere scelto ad ogni $n = 0, 1, \dots$ al fine di soddisfare la condizione di assoluta stabilità, si vedano (8.42) e (8.70).

D’altro canto, una volta soddisfatta tale condizione, possiamo pensare di modificare il passo h fra un nodo e il successivo in modo da garantire che sia raggiunta una determinata accuratezza. Questa procedura, che è chiamata di *adattività del passo*, presuppone la conoscenza di una opportuna stima dell’errore locale. A tale scopo si costruisce uno *stimatore a posteriori dell’errore*; stime a priori dell’errore locale (come ad esempio (8.29)) richiederebbero infatti informazioni sulla derivata seconda della funzione incognita. Illustriamo questa tecnica nel caso del metodo di Eulero in avanti.

Supponiamo di aver calcolato la soluzione numerica fino al tempo \tilde{t} . Scegliamo un valore iniziale per h e denotiamo con u_h (rispettivamente $u_{h/2}$) la soluzione ottenuta con un singolo passo di ampiezza h del metodo di Eulero in avanti (rispettivamente con due passi di lunghezza $h/2$),

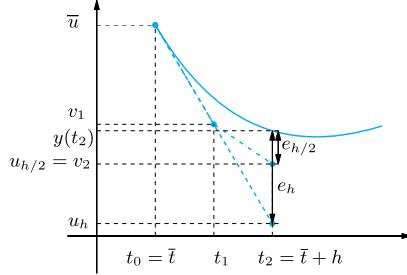


Figura 8.15. Soluzione numerica con il metodo di Eulero in avanti con un passo di ampiezza h e due passi di ampiezza $h/2$. La curva in linea continua rappresenta la soluzione di (8.71)

prendendo come valore iniziale il valore noto \bar{u} al tempo \bar{t} :

$$\begin{aligned}\tilde{u}_h &= \bar{u} + hf(\bar{t}, \bar{u}), \\ v_1 &= \bar{u} + \frac{h}{2}f(\bar{t}, \bar{u}), \quad \tilde{u}_{h/2} = v_2 = v_1 + \frac{h}{2}f\left(\bar{t} + \frac{h}{2}, v_1\right).\end{aligned}$$

Ora esaminiamo gli errori $e_h = y(\bar{t} + h) - u_h$ ed $e_{h/2} = y(\bar{t} + h) - u_{h/2}$, essendo ora $y(t)$ la soluzione esatta del problema di Cauchy

$$\begin{cases} y'(t) = f(t, y(t)) & t \geq \bar{t}, \\ y(\bar{t}) = \bar{u}. \end{cases} \quad (8.71)$$

Grazie alla (8.26) abbiamo, per un opportuno $\xi \in (\bar{t}, \bar{t} + h)$,

$$e_h = \frac{h^2}{2}y''(\xi). \quad (8.72)$$

Per semplicità di esposizione, ora poniamo (si veda la Figura 8.15)

$$t_0 = \bar{t}, \quad t_1 = \bar{t} + h/2, \quad t_2 = \bar{t} + h$$

e riscriviamo l'errore $e_{h/2}$ nella forma (8.24), abbiamo

$$e_{h/2} = (y(t_2) - v_2^*) + (v_2^* - v_2), \quad (8.73)$$

dove $v_2^* = y(t_1) + \frac{h}{2}f(t_1, y(t_1))$. Il primo addendo di (8.73) è l'errore di troncamento locale, pertanto esiste un opportuno $\eta_2 \in (t_1, t_2)$ tale che

$$y(t_2) - v_2^* = \frac{(h/2)^2}{2}y''(\eta_2).$$

Il secondo addendo è dovuto alla propagazione dell'errore al passo precedente di ampiezza $h/2$; grazie alla (8.27)abbiamo

$$v_2^* - v_2 = (y(t_1) - v_1) + \frac{h}{2}[f(t_1, y(t_1)) - f(t_1, v_1)].$$

Il termine $(y(t_1) - v_1)$ è ancora un errore di troncamento locale e lo possiamo scrivere come $y(t_1) - v_1 = \frac{(h/2)^2}{2} y''(\eta_1)$ per un opportuno $\eta_1 \in (t_0, t_1)$.

Per valutare $[f(t_1, y(t_1)) - f(t_1, v_1)]$ richiediamo che f sia di classe C^1 e scriviamo lo sviluppo di Taylor di f di ordine zero centrato in (t_1, v_1) , ottenendo

$$f(t_1, y(t_1)) = f(t_1, v_1) + (y(t_1) - v_1) \frac{\partial f}{\partial y}(t_1, \zeta)$$

per un opportuno $\zeta \in \mathbb{R}$. Di conseguenza

$$v_2^* - v_2 = (y(t_1) - v_1) \left[1 + h \frac{\partial f}{\partial y}(t_1, \zeta) \right] = \frac{(h/2)^2}{2} y''(\eta_1) + o(h^2).$$

Supponendo che y'' abbia piccole variazioni nell'intervallo $(\bar{t}, \bar{t}+h)$, si ha

$$e_{h/2} = \frac{(h/2)^2}{2} [y''(\eta_2) + y''(\eta_1)] + o(h^2) = \frac{h^2}{4} y''(\eta) + o(h^2), \quad (8.74)$$

per un opportuno $\eta \in (\bar{t}, \bar{t}+h)$.

Per ottenere una stima di y'' da utilizzare in (8.74) sottraiamo (8.74) da (8.72); sempre nell'ipotesi che y'' vari poco in $(\bar{t}, \bar{t}+h)$, abbiamo

$$\tilde{u}_{h/2} - \tilde{u}_h = e_h - e_{h/2} = \frac{h^2}{4} (2y''(\xi) - y''(\eta)) + o(h^2) = \frac{h^2}{4} y''(\hat{\xi}) + o(h^2),$$

per un opportuno $\hat{\xi} \in (\bar{t}, \bar{t}+h)$. D'altro canto,

$$|e_{h/2}| \simeq \frac{h^2}{4} |y''(\hat{\xi})| \simeq |\tilde{u}_{h/2} - \tilde{u}_h|, \quad (8.75)$$

ovvero la quantità $|\tilde{u}_{h/2} - \tilde{u}_h|$ fornisce uno stimatore a-posteriori dell'errore $|y(\bar{t}+h) - \tilde{u}_{h/2}|$ a meno di un termine $o(h^2)$. Quindi, fissata la tolleranza ϵ , se otteniamo

$$|\tilde{u}_{h/2} - \tilde{u}_h| < \frac{\epsilon}{2}$$

(la divisione per 2 è fatta per via cautelativa per aver assunto y'' approssimativamente costante sull'intervallo $(\bar{t}, \bar{t}+h)$), allora accettiamo il passo di lunghezza h per avanzare e prendiamo $\tilde{u}_{h/2}$ come la soluzione numerica al passo $\bar{t}+h$. Altrimenti h viene dimezzato e la procedura è ripetuta fino a convergenza. In ogni caso, al fine di evitare di raggiungere passi di integrazione troppo piccoli, viene considerato un valore minimo h_{\min} e pretendiamo che $h \geq h_{\min}$.

Osserviamo infine che qualche volta lo stimatore dell'errore $|\tilde{u}_{h/2} - \tilde{u}_h|$ è sostituito dall'errore relativo $|\tilde{u}_{h/2} - \tilde{u}_h|/u_{\max}$, dove u_{\max} rappresenta il massimo valore assunto dalla soluzione numerica nell'intervallo $[t_0, \bar{t}]$.

Formuliamo di seguito l'algoritmo che implementa il metodo di Eulero in avanti adattivo. Data una tolleranza $\varepsilon > 0$, un passo iniziale h (ad esempio $h = (T - t_0)/10$) e un passo minimo $h_{\min} > 0$:

```

porre  $u_0 = y_0$ ,  $n = 0$ ,
while  $t_n < T$  calcolare
     $\tilde{u}_h = u_n + hf(t_n, u_n)$ 
     $v_1 = u_n + \frac{h}{2}f(t_n, u_n)$ ,  $\tilde{u}_{h/2} = v_1 + \frac{h}{2}f\left(t_n + \frac{h}{2}, v_1\right)$ 
    if  $|\tilde{u}_{h/2} - \tilde{u}_h| / \max_{k \leq n} |u_k| < \varepsilon/2$  or  $h < h_{\min}$ 
         $t_{n+1} = t_n + h$ ,  $u_{n+1} = \tilde{u}_{h/2}$ 
         $h = 2h$ ,  $n = n + 1$ 
    else
         $h = h/2$ 
    endif
end

```

(8.76)

Esempio 8.8 Risolviamo il problema

$$\begin{cases} y'(t) = \arctan(5(1-t))y(t), & t \in (0, 3), \\ y(0) = 2^{6/10}, \end{cases} \quad (8.77)$$

la cui soluzione esatta è

$$y(t) = \exp((t-1)\arctan(5-5t) + \arctan 5 - \frac{1}{10}\log(1+25(t-1)^2)),$$

con l'algoritmo adattivo (8.76) e tre diversi valori della tolleranza ϵ sullo stimatore dell'errore. A sinistra di Figura 8.16 sono rappresentati i passi variabili

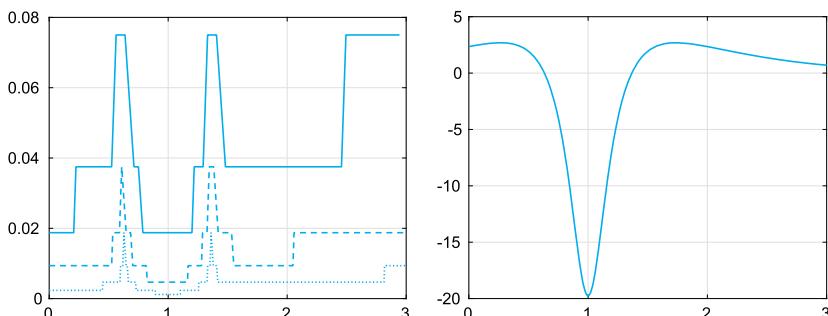


Figura 8.16. A sinistra, l'andamento del passo variabile per il metodo adattivo (8.76) per tre diversi valori della tolleranza: $\epsilon = 10^{-3}$ (linea continua), $\epsilon = 10^{-4}$ (linea tratteggiata) e $\epsilon = 10^{-5}$ (linea punteggiata). A destra, la funzione $y''(t)$

ottenuti con l'algoritmo (8.76), mentre a destra della stessa figura è disegnata la derivata seconda della soluzione $y(t)$. Dal confronto dei due grafici emerge il legame tra il valore assoluto di y'' e l'ampiezza dei passi, come si può osservare in (8.75): i passi si riducono dove $|y''|$ è alto e, viceversa, si ampliano dove $|y''|$ è più prossimo a zero. Prendendo $\epsilon = 10^{-3}$ l'intervallo $(0, 3)$ è coperto da 88 passi, con $\epsilon = 10^{-4}$ da 282 passi e con $\epsilon = 10^{-5}$ da 902 passi, mentre l'errore $\max_n |y_n - u_n|$ corrispondente è pari a 0.0162, 0.0056 e 0.0015, rispettivamente per $\epsilon = 10^{-3}$, $\epsilon = 10^{-4}$ e $\epsilon = 10^{-5}$. ■



Si vedano gli Esercizi 8.6–8.10.

Riassumendo

1. Un metodo è consistente se l'errore di troncamento globale è infinitesimo per $h \rightarrow 0$;
2. un metodo è zero-stabile (su un intervallo limitato) se le perturbazioni sulla soluzione sono controllate dalle perturbazioni sui dati, uniformemente rispetto a h ;
3. un metodo è convergente se le soluzioni numeriche calcolate con passi h diversi convergono uniformemente alla soluzione esatta per $h \rightarrow 0$;
4. sotto opportune ipotesi sulla funzione di incremento Φ , un metodo ad un passo è zero-stabile. Se inoltre il metodo è consistente allora esso è anche convergente;
5. un metodo è assolutamente stabile per una scelta del passo di discretizzazione h quando, applicato al problema modello (8.38), genera una soluzione u_n che tende a zero quando t_n tende all'infinito;
6. un metodo è detto A-stabile quando è assolutamente stabile per ogni possibile scelta del passo di integrazione h ed ogni $\lambda \in \mathbb{C}$ con $\operatorname{Re}(\lambda) < 0$ (in caso contrario si dice che il metodo è condizionatamente assolutamente stabile e h dovrà essere minore di una costante che dipende da λ);
7. un metodo ad un passo A-stabile è detto L-stabile se la sua soluzione numerica relativa al problema modello (8.38) può essere scritta nella forma $u_{n+1} = \mathcal{R}(h\lambda)u_n$ con $\mathcal{R}(h\lambda) \rightarrow 0$ per $\operatorname{Re}(h\lambda) \rightarrow -\infty$;
8. quando un metodo assolutamente stabile per un certo valore di h viene applicato ad un problema modello generalizzato, come (8.52), 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*;
9. 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 (8.5) nel caso in cui f soddisfi (8.67). 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 (8.70) per un opportuno $\alpha \in (0, 1)$, oppure (8.34) nel caso di passo costante h ;

10. le strategie di adattività del passo di discretizzazione basate su stime a posteriori scelgono il passo h in funzione del comportamento della soluzione incognita, utilizzando stimatori dell'errore ottenuti a partire da due soluzioni numeriche differenti.

8.9 Metodi di ordine elevato

Tutti i metodi presentati finora sono esempi elementari di metodi ad un passo ed i ordine al massimo pari a 2. Accenniamo ora ad altri metodi che consentono il raggiungimento di un maggior ordine di accuratezza: i *metodi Runge-Kutta* ed i *metodi multistep*.

8.9.1 I metodi Runge-Kutta

I metodi *Runge-Kutta* (in breve, RK) sono metodi ad un passo che richiedono 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

$$u_{n+1} = u_n + h \sum_{i=1}^s b_i K_i, \quad n \geq 0, \quad (8.78)$$

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 $\mathbf{A} = (a_{ij}) \in \mathbb{R}^{s \times s}$, $\mathbf{b} = (b_1, \dots, b_s)^T \in \mathbb{R}^s$ e $\mathbf{c} = (c_1, \dots, c_s)^T \in \mathbb{R}^s$. Si assume che

$$c_i = \sum_{j=1}^s a_{ij}.$$

Se i coefficienti a_{ij} di \mathbf{A} sono uguali a zero per $j \geq i$, con $i = 1, 2, \dots, s$, allora ciascun K_i può essere ottenuto esplicitamente 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 forma

$$u_{n+1} = u_n + \frac{h}{6}(K_1 + 2K_2 + 2K_3 + K_4) \quad (8.79)$$

dove

$$\begin{aligned} K_1 &= f_n, & 0 \\ K_2 &= f\left(t_n + \frac{h}{2}, u_n + \frac{h}{2}K_1\right), & \frac{1}{2} \Big| \frac{1}{2} \\ K_3 &= f\left(t_n + \frac{h}{2}, u_n + \frac{h}{2}K_2\right), & \frac{1}{2} \Big| 0 \quad \frac{1}{2} \\ K_4 &= f(t_{n+1}, u_n + hK_3), & 1 \Big| 0 \quad 0 \quad 1 \\ & & \hline & \frac{1}{6} \quad \frac{1}{3} \quad \frac{1}{3} \quad \frac{1}{6} \end{aligned}$$

Esso si può derivare dalla (8.9) usando la formula di quadratura di Simpson (4.30) per approssimare l'integrale fra t_n e t_{n+1} . Si tratta di un metodo esplicito di ordine 4 rispetto a h che richiede, ad ogni passo, 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

$$\begin{array}{c|cc} \frac{3-\sqrt{3}}{6} & \frac{1}{4} & \frac{3-2\sqrt{3}}{12} \\ \frac{3+\sqrt{3}}{6} & \frac{3+2\sqrt{3}}{12} & \frac{1}{4} \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}.$$

I metodi RK (8.78) possono essere riscritti nella forma (8.36), purché si definisca

$$\Phi(t_n, h, u_n, u_{n+1}; f) = \sum_{i=1}^s b_i K_i. \quad (8.80)$$

Si può verificare che, nella sola ipotesi che f sia lipschitziana rispetto al suo secondo argomento, la funzione di incremento Φ definita in (8.80) soddisfa le proprietà (8.37). Di conseguenza, grazie al Teorema 8.1, i metodi RK sono zero-stabili.

Per garantire che il metodo RK (8.78) sia consistente i coefficienti b_j devono soddisfare la condizione

$$\sum_{j=1}^s b_j = 1. \quad (8.81)$$

Sotto questa ipotesi possiamo quindi concludere che i metodi RK sono convergenti, grazie al Teorema 8.2.

La regione di assoluta stabilità \mathcal{A} dei metodi RK, sia impliciti che esplicativi, può crescere in estensione al crescere dell'ordine. Un esempio è fornito dal grafico in Figura 8.19 a sinistra, dove è riportata la regione \mathcal{A} dei seguenti metodi esplicativi di ordine crescente: RK1, ovvero il metodo di Eulero in avanti, RK2, ovvero il cosiddetto *metodo di Eulero migliorato* (o *metodo di Heun*) che verrà introdotto oltre (si veda (8.93)), RK3, ovvero il metodo associato al seguente *array* di Butcher

$$\begin{array}{c|cc} 0 & & \\ \frac{1}{2} & \frac{1}{2} & \\ 1 & -1 & 2 \\ \hline & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \end{array} \quad (8.82)$$

e infine RK4, il metodo introdotto in (8.79).

Analogamente a quanto fatto con il metodo di Eulero in avanti, anche i metodi RK, in quanto metodi ad un passo, possono essere implementati con adattività di passo; lo stimatore dell'errore per questi metodi può essere costruito in due modi:

- utilizzando lo stesso metodo RK, ma con due differenti passi, come abbiamo fatto con il metodo di Eulero. Lo stimatore dell'errore così ottenuto è molto affidabile, ma il costo computazionale per valutarlo è piuttosto alto, richiedendo un numero di valutazioni della funzione f pari al triplo del numero di stadi s del metodo;
- utilizzando due metodi RK di ordine diverso (rispettivamente p e $p+1$), ma con la stessa matrice A nell'*array* di Butcher (e quindi lo stesso numero s di stadi), cosicché il numero di valutazioni richieste di f rimane pari a s . Coppie di metodi RK che soddisfano questa proprietà sono dette *embedded*. Dei due metodi, uno svolge un ruolo primario, in quanto lo stimatore dell'errore approssima il suo errore di troncamento locale e la soluzione da esso prodotta è quella che viene accettata ad ogni passo; l'altro metodo invece è soltanto funzionale a stimare l'errore.

La seconda procedura è quella utilizzata sia da MATLAB che da Octave all'interno delle proprie *function* in cui sono implementati i metodi RK.

Osservazione 8.4 In MATLAB, il programma `ode45` implementa il metodo di Dormand-Prince 5(4), in cui è utilizzata una coppia di metodi *embedded* RK esplicativi a 6 stadi, il metodo primario ha ordine 5, il secondo ha ordine 4 [Lam91, DP80]. Il programma `ode23` invece implementa il metodo Bogacki e Shampine 3(2), caratterizzato da un'altra coppia di metodi RK *embedded* esplicativi: il metodo primario ha ordine 3, il secondo ha ordine 2 [BS89].

MAT||OCT
`ode45`
`ode23`

ode23 In Octave invece, i programmi **ode23** e **ode45** implementano i metodi *embedded* RK Fehlberg 2–3 e 4–5, rispettivamente, in cui il metodo primario è quello di ordine inferiore [Lam91]. Il metodo Dormand-Prince 5(4) è implementato nel programma **ode54**.

In tutti questi programmi il passo di integrazione viene calcolato in modo da garantire che l'errore tra soluzione numerica e soluzione esatta si mantenga al di sotto di una tolleranza fissata (pari a 10^{-3} in MATLAB e pari a 10^{-6} in Octave). ■

8.9.2 I metodi multistep

Un *metodo multistep lineare* a $(p + 1)$ passi (o *step*) può essere scritto nella forma generale

$$u_{n+1} = \sum_{j=0}^p a_j u_{n-j} + h \sum_{j=0}^p b_j f_{n-j} + hb_{-1} f_{n+1}, \quad n = p, p+1, \dots \quad (8.83)$$

per opportuni coefficienti $\{a_k\}$ e $\{b_k\}$ e per un opportuno intero $p \geq 0$.

Esso è esplicito se $b_{-1} = 0$, implicito altrimenti.

I valori u_0, u_1, \dots, u_p devono essere noti prima di innestare il metodo. Mentre u_0 è assegnato uguale a y_0 , i valori u_1, \dots, u_p debbono essere generati con opportuni metodi sufficientemente accurati, come ad esempio i metodi Runge-Kutta che abbiamo descritto nella Sezione 8.9.1.

Il polinomio di grado $p + 1$ nella variabile r

$$\pi(r) = r^{p+1} - \sum_{j=0}^p a_j r^{p-j} \quad (8.84)$$

è detto il *primo polinomio caratteristico* associato al metodo numerico (8.83). Denotando le sue radici con r_j , $j = 0, \dots, p$, si può provare che il metodo (8.83) è zero-stabile se e solo se vale la seguente *condizione delle radici*

$$|r_j| \leq 1 \text{ per ogni } j = 0, \dots, p; \text{ inoltre}$$

$$\pi'(r_j) \neq 0 \text{ per quei } j \text{ tali che } |r_j| = 1$$

(8.85)

Ad esempio, per il metodo di Eulero in avanti abbiamo

$$p = 0, \quad a_0 = 1, \quad b_{-1} = 0, \quad b_0 = 1,$$

per il metodo di Eulero all'indietro abbiamo

$$p = 0, \quad a_0 = 1, \quad b_{-1} = 1, \quad b_0 = 0,$$

e per il metodo di Crank-Nicolson abbiamo

$$p = 0, \quad a_0 = 1, \quad b_{-1} = 1/2, \quad b_0 = 1/2.$$

In tutti questi casi $\pi(r) = r - 1$, l'unica radice di $\pi(r)$ vale 1 e, di conseguenza, tutti questi metodi sono zero-stabili.

Coerentemente con quanto fatto in precedenza, l'errore di troncamento locale per un metodo *multistep* (8.83) è definito come

$$\begin{aligned}\tau_n(h) = \frac{1}{h} & \left(y_{n+1} - \sum_{j=0}^p a_j y_{n-j} \right. \\ & \left. - h \sum_{j=0}^p b_j f(t_{n-j}, y_{n-j}) - hb_{-1} f(t_{n+1}, y_{n+1}) \right). \quad (8.86)\end{aligned}$$

Ricordiamo che un metodo è detto consistente se $\tau(h) = \max_n |\tau_n(h)|$ tende a zero per h che tende a zero ed è consistente di ordine q (≥ 1) se $\tau(h) = \mathcal{O}(h^q)$ per h che tende a zero.

Mediante l'uso degli sviluppi di Taylor si può dimostrare che un metodo *multistep* è consistente se e solo se sono soddisfatte le seguenti relazioni sui coefficienti:

$$\sum_{j=0}^p a_j = 1, \quad -\sum_{j=0}^p ja_j + \sum_{j=-1}^p b_j = 1 \quad (8.87)$$

Osserviamo che la prima delle due condizioni è soddisfatta se e solo se $r = 1$ è una radice del polinomio $\pi(r)$ introdotto in (8.84) (per la dimostrazione si veda ad esempio [QSSG14, Cap. 10]). Pertanto, condizione necessaria affinché un metodo *multistep* sia consistente è che $\pi(1) = 0$.

Sebbene si possano costruire metodi *multistep* a $(p+1)$ passi che siano consistenti di ordine $2(p+1)$, solo gli schemi con un ordine di consistenza al più pari a $q = p+3$ possono essere zero-stabili e, quindi, convergenti in virtù del seguente del Teorema [IK66]:

Teorema 8.3 *Un metodo multistep (8.83) consistente e zero-stabile è convergente.*

Più precisamente, in accordo con la cosiddetta *prima barriera di Dahlquist* (si veda [IK66]), un metodo *multistep* esplicito a $(p+1)$ passi che sia zero-stabile (quindi che soddisfa la condizione delle radici) ha al più ordine di consistenza $q = p+1$; un metodo *multistep* implicito a $(p+1)$ passi (con $(p+1)$ dispari) che sia zero-stabile ha al più ordine di consistenza $q = p+2$; un metodo *multistep* implicito a $(p+1)$ passi (con $(p+1)$ pari) che sia zero-stabile ha al più ordine di consistenza $q = p+3$.

Vediamo ora alcune famiglie notevoli di metodi *multistep*.

Applicando prima la formula (8.9) e quindi approssimando l'integrale che vi compare con una formula di quadratura che utilizza il polinomio

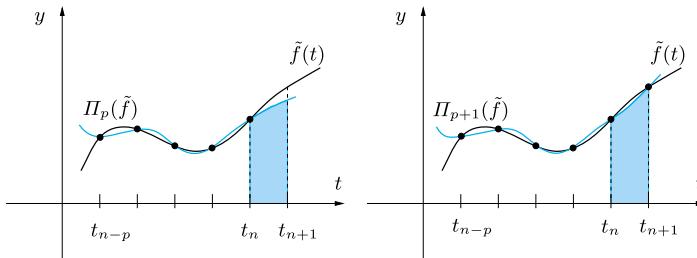


Figura 8.17. Costruzione dei metodi di Adams-Bashforth (*a sinistra*) e di Adams-Moulton (*a destra*). In nero la funzione \tilde{f} , in blu chiaro il suo polinomio interpolatore

interpolatore $\Pi_p(\tilde{f})$ di $\tilde{f}(t) = f(t, y(t))$ nei nodi t_{n-p}, \dots, t_n si ottengono i metodi esplicativi di Adams-Bashforth, mentre se si interpola $\tilde{f}(t)$ sui nodi t_{n-p}, \dots, t_{n+1} (in tal caso il polinomio interpolatore $\Pi_{p+1}(\tilde{f})$ ha grado $p+1$) si ottengono i metodi impliciti di Adams-Moulton (si veda la Figura 8.17).

I metodi di Adams-Bashforth (esplicativi) per $p = 0, 1, 2, 3$ sono:

$$\begin{aligned}
 \text{AB1: } & u_{n+1} = u_n + hf_n \\
 \text{AB2: } & u_{n+1} = u_n + \frac{h}{2}(3f_n - f_{n-1}) \\
 \text{AB3: } & u_{n+1} = u_n + \frac{h}{12}(23f_n - 16f_{n-1} + 5f_{n-2}) \\
 \text{AB4: } & u_{n+1} = u_n + \frac{h}{24}(55f_n - 59f_{n-1} + 37f_{n-2} - 9f_{n-3})
 \end{aligned} \tag{8.88}$$

mentre quelli di Adams-Moulton (impliciti), sempre per $p = 0, 1, 2, 3$, sono:

$$\begin{aligned}
 \text{AM2: } & u_{n+1} = u_n + \frac{h}{2}(f_{n+1} + f_n) \\
 \text{AM3: } & u_{n+1} = u_n + \frac{h}{12}(5f_{n+1} + 8f_n - f_{n-1}) \\
 \text{AM4: } & u_{n+1} = u_n + \frac{h}{24}(9f_{n+1} + 19f_n - 5f_{n-1} + f_{n-2}) \\
 \text{AM5: } & u_{n+1} = u_n + \frac{h}{720}(251f_{n+1} + 646f_n - 264f_{n-1} \\
 & \quad + 106f_{n-2} - 19f_{n-3})
 \end{aligned} \tag{8.89}$$

Osserviamo anzitutto che AB1 coincide con il metodo di Eulero in avanti, mentre AM2 con il metodo di Crank-Nicolson.

Il numero k che segue l'acronimo AB o AM indica l'ordine di convergenza del metodo. In effetti i metodi di Adams-Bashfort a $(p+1)$ passi hanno ordine di convergenza $q = p+1$, mentre i metodi di Adams-Moulton a $(p+1)$ passi hanno ordine di convergenza $q = p+2$.

Ad esempio, la formula esplicita AB3 a tre passi ($p=2$) è un metodo del terz'ordine, essa è ottenuta sostituendo $\tilde{f}(t) = f(t, y(t))$ nella (8.9) con il suo polinomio interpolatore di grado 2 nei nodi t_{n-2}, t_{n-1}, t_n , mentre AM4 è una formula implicita a tre passi del quart'ordine ed è ottenuta approssimando $\tilde{f}(t) = f(t, y(t))$ in (8.9) 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*, noti come *backward difference formula* (BDF), 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, ottenuto approssimando $y'(t_{n+1})$ con la formula (4.13)₂, è dato dalla formula BDF2 a due passi implicita e convergente del secondo ordine in h

$$u_{n+1} = \frac{4}{3}u_n - \frac{1}{3}u_{n-1} + \frac{2h}{3}f_{n+1} \quad (8.90)$$

e un altro dalla formula BDF3 a tre passi implicita e del terz'ordine in h

$$u_{n+1} = \frac{18}{11}u_n - \frac{9}{11}u_{n-1} + \frac{2}{11}u_{n-2} + \frac{6h}{11}f_{n+1} \quad (8.91)$$

Tutti questi metodi possono essere scritti nella forma generale (8.83).

È facile verificare che le condizioni (8.87) sono soddisfatte per tutti i metodi (8.88)–(8.91) e, di conseguenza, essi sono consistenti; inoltre essi sono tutti zero-stabili.

Ad esempio, per entrambe le formule AB3 e AM4 il primo polinomio caratteristico è $\pi(r) = r^3 - r^2$ le cui radici sono $r_0 = 1, r_1 = r_2 = 0$, quello di (8.90) è $\pi(r) = r^2 - (4/3)r + 1/3$ ed ha radici $r_0 = 1$ e $r_1 = 1/3$, mentre quello di (8.91) 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 (8.85) è soddisfatta.

Inoltre, quando applicati al problema modello (8.38) con $\lambda \in \mathbb{R}^-$, AB3 è assolutamente stabile se $h < 0.545/|\lambda|$, mentre AM4 è assolutamente stabile se $h < 3/|\lambda|$.

Il metodo BDF2 è incondizionatamente assolutamente stabile per ogni $\lambda \in \mathbb{C}$ con parte reale negativa (cioè è A-stabile).

Se $\lambda \in \mathbb{R}^-$, BDF3 è incondizionatamente assolutamente stabile, ma questa proprietà non è valida per tutti i $\lambda \in \mathbb{C}$ con parte reale negativa. In altre parole, il metodo BDF3 non è A-stabile.

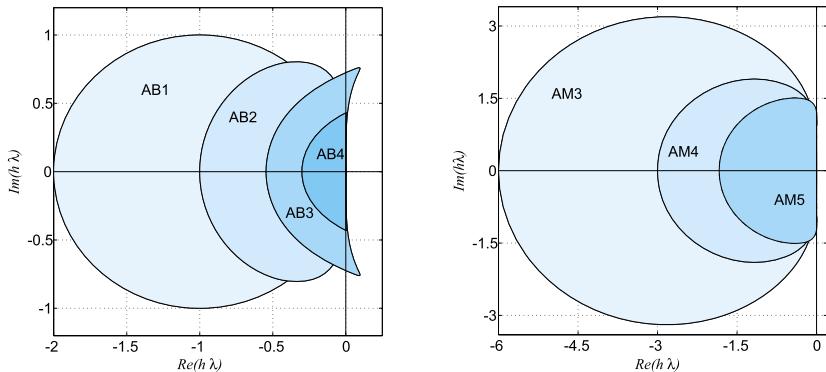


Figura 8.18. Le regioni di assoluta stabilità di alcuni metodi di Adams-Bashforth (*a sinistra*) e di Adams-Moulton (*a destra*)

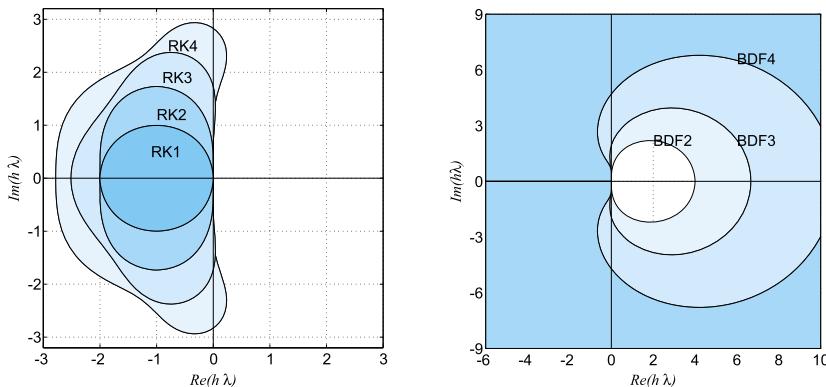


Figura 8.19. Le regioni di assoluta stabilità di alcuni metodi RK esplicativi (*a sinistra*) e BDF (*a destra*). In quest'ultimo caso le regioni sono illimitate e si estendono esternamente alle linee chiuse

Più in generale, non esistono metodi *multistep* A-stabili esplicativi e nemmeno metodi *multistep* A-stabili di ordine strettamente maggiore di 2. Questo risultato è noto come *seconda barriera di Dahlquist* [Lam91].

In Figura 8.18 riportiamo le regioni di assoluta stabilità di alcuni metodi di Adams-Bashforth e di Adams-Moulton. Si noti come l'estensione della regione decresca al crescere dell'ordine di convergenza. Nei grafici di Figura 8.19 a destra riportiamo le regioni (illimitate) di assoluta stabilità di alcuni metodi BDF: anch'esse ricoprono un'area del piano complesso che si riduce al crescere dell'ordine, cioè $\mathcal{A}_{BDF(k+1)} \subset \mathcal{A}_{BDF(k)}$, contrariamente a quelle dei metodi Runge-Kutta

(riportate a sinistra) che al crescere dell'ordine aumentano di estensione, ovvero $\mathcal{A}_{RK(k)} \subset \mathcal{A}_{RK(k+1)}$, $k \geq 1$.

Osservazione 8.5 (Come disegnare le regioni di assoluta stabilità) Il bordo $\partial\mathcal{A}$ della regione di assoluta stabilità \mathcal{A} di un metodo *multistep* può essere ottenuto con un semplice artificio. Il bordo è infatti il luogo dei punti $h\lambda$ del piano complesso tali che

$$h\lambda = \left(r^{p+1} - \sum_{j=0}^p a_j r^{p-j} \right) / \left(\sum_{j=-1}^p b_j r^{p-j} \right), \quad (8.92)$$

con r numero complesso di modulo unitario. Di conseguenza, per ottenere una rappresentazione approssimata di $\partial\mathcal{A}$ è sufficiente valutare il secondo membro di (8.92) al variare di r sulla circonferenza unitaria (ad esempio, ponendo $r = \exp(i\pi*(0:2000)/1000)$, dove i è l'unità immaginaria). Ad esempio, le istruzioni `MATLAB` per disegnare la regione di assoluta stabilità del metodo AB3 sono:

```
r = exp(i*pi*(0:2000)/1000);
num = r-1; den = (23-16./r+5./r.^2)/12;
dA=num./den;
plot(dA,'c','LineWidth',1)
fill(real(dA),imag(dA),[0.6,1,1])
axis equal; grid on
```

I grafici delle Figure 8.18 e 8.19 sono stati ottenuti in questo modo. ■

Osservazione 8.6 (Metodi compositi ciclici) È possibile superare le barriere di Dahlquist combinando opportunamente più metodi *multistep*. 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 entrambi instabili. Utilizzandoli però in modo combinato (il primo se n è pari, il secondo se n è dispari) si ottiene un metodo A-stabile di ordine 5 a 3 passi. ■

Osservazione 8.7 I metodi *multistep* sono implementati ad esempio nei programmi `lsode` di Octave, o `ode113` e `ode15s` di MATLAB. ■

`MATLAB`

- `lsode`
- `ode113`
- `ode15s`

8.9.3 I metodi predictor-corrector

Come abbiamo osservato nella Sezione 8.3, 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 utilizzare le *function* di **MAT&OCT**.

Un’ulteriore alternativa consiste nell’eseguire delle iterazioni di punto fisso ad ogni passo temporale. Ad esempio, per il metodo di Crank-Nicolson (8.8), per $k = 0, 1, \dots$, fino a convergenza si calcola

$$u_{n+1}^{(k+1)} = u_n + \frac{h}{2} [f_n + f(t_{n+1}, u_{n+1}^{(k)})].$$

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 implicito originale. Precisamente, se il metodo implicito 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*), già riferito precedentemente con l’acronimo RK2

$$\begin{aligned} u_{n+1}^* &= u_n + h f_n, \\ u_{n+1} &= u_n + \frac{h}{2} [f_n + f(t_{n+1}, u_{n+1}^*)] \end{aligned} \tag{8.93}$$

In generale, il metodo esplicito viene denominato *predictor*, mentre quello隐式 è detto *corrector*. Un altro esempio di questa famiglia di metodi è dato dalla combinazione del metodo (AB3) (8.88), usato come predictor, con il metodo (AM4) (8.89), usato come corrector. Schemi di questo genere vengono quindi chiamati metodi *predictor-corrector* (PC). Essi garantiscono l’ordine di accuratezza del metodo corrector. D’altra parte, essendo esplicativi, presentano una regione di assoluta stabilità più 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 8.20). Questi schemi non sono pertanto adeguati per la risoluzione di problemi di Cauchy su intervalli illimitati.

Nel Programma 8.4 implementiamo un generico metodo predictor-corrector. I *function handle* **predictor** e **corrector** identificano il tipo di metodo scelto. Ad esempio, se si usano le funzioni **feonestep** e **cnonestep**, definite rispettivamente nei Programmi 8.5 e 8.7, richiamiamo **predcor** come segue

```
[t, u] = predcor(f, [t0, T], y0, N, @feonestep, @cnonestep);
```

ottenendo così il metodo di Heun.

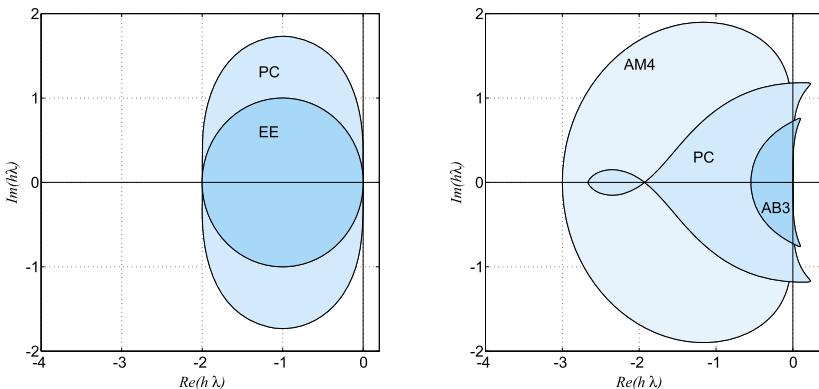


Figura 8.20. Le regioni di assoluta stabilità dei metodi predictor-corrector ottenuti combinando i metodi di Eulero esplicito (EE) e di Crank-Nicolson (*a sinistra*) e AB3 e AM4 (*a destra*). Si noti la riduzione dell'estensione della regione rispetto a quella dei corrispondenti metodi impliciti (nel primo caso la regione del metodo di Crank-Nicolson non è stata riportata in quanto coincide con tutto il semipiano complesso $\operatorname{Re}(h\lambda) < 0$)

Programma 8.4. predcor: un generico metodo predictor-corrector

```
function [t,u]=predcor(odefun,tspan,y0,Nh,...  
    predictor,corrector)  
%PREDCOR Risolve equazioni differenziali  
% con il metodo predictor-corrector  
% [T,Y]=PREDCOR(ODEFUN,TSPAN,Y0,NH,@PRED,@CORR) con  
% TSPAN=[T0 TF] integra il sistema di equazioni  
% differenziali  $y' = f(t,y)$  dal tempo T0 a TF con  
% condizioni iniziali Y0 utilizzando un generico  
% metodo predictor-corrector su una griglia  
% equispaziata di NH intervalli.  
% La funzione ODEFUN(T,Y) deve ritornare un vettore  
% contenente  $f(t,y)$ , della stessa dimensione di  $y$ .  
% Ogni riga del vettore soluzione Y corrisponde ad  
% un istante temporale del vettore colonna T.  
% Le function PRED e CORR caratterizzano il tipo  
% di metodo predictor-corrector scelto.  
% Possono essere scelte, ad esempio, tra le  
% function proposte:  
% feonestep, beonestep, cnonestep.  
h=(tspan(2)-tspan(1))/Nh;  
t=linspace(tspan(1),tspan(2),Nh+1)';  
y=y0(:); % genera sempre un vettore colonna  
d=length(y0);  
u=zeros(Nh+1,d);  
u(1,:)=y0.'; % trasposta anche di variabili complesse  
for n=1:Nh  
    wn=u(n,:).';  
    fn = odefun(t(n),wn);  
    upre = predictor(t(n),wn,h,fn);
```

```
w = corrector(t(n+1),wn,upre,h,odefun,fn);
u(n+1,:)=w.';
```

Programma 8.5. feonestep: un passo del metodo di Eulero in avanti

```
function [u]=feonestep(t,y,h,f)
% FEONESTEP un passo del metodo di Eulero in avanti
u = y + h*f;
```

Programma 8.6. beonestep: un passo del metodo di Eulero all'indietro

```
function [u]=beonestep(t,u,y,h,f,fn)
% BEONESTEP un passo del metodo di Eulero all'indietro
u = u + h*f(t,y);
```

Programma 8.7. cnonestep: un passo del metodo di Crank-Nicolson

```
function [u]=cnonestep(t,u,y,h,f,fn)
% CNONESTEP un passo del metodo di Crank-Nicolson
u = u + 0.5*h*(f(t,y)+fn);
```

ode113 Il programma **ode113** di MATLAB implementa un metodo predictor-corrector di Adams-Bashforth-Moulton con passo di discretizzazione variabile.



Si vedano gli Esercizi 8.11–8.17.

8.10 Sistemi di equazioni differenziali

Consideriamo il seguente sistema di equazioni differenziali di ordine uno nelle incognite $y_1 = y_1(t), \dots, y_m = y_m(t)$:

$$\begin{cases} y'_1(t) = f_1(t, y_1, \dots, y_m), \\ \vdots \\ y'_m(t) = f_m(t, y_1, \dots, y_m), \end{cases}$$

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

$$\begin{cases} u_{n+1,1} = u_{n,1} + h f_1(t_n, u_{n,1}, \dots, u_{n,m}), \\ \vdots \\ u_{n+1,m} = u_{n,m} + h f_m(t_n, u_{n,1}, \dots, u_{n,m}). \end{cases}$$

Scrivendo il sistema in forma vettoriale

$$\begin{cases} \mathbf{y}'(t) = \mathbf{F}(t, \mathbf{y}(t)), & t > t_0 \\ \mathbf{y}(t_0) = \mathbf{y}_0, \end{cases}$$

l'estensione dei metodi precedentemente sviluppati nel caso di una singola equazione appare immediata. Ad esempio, il metodo

$$\mathbf{u}_{n+1} = \mathbf{u}_n + h(\theta \mathbf{F}(t_{n+1}, \mathbf{u}_{n+1}) + (1 - \theta) \mathbf{F}(t_n, \mathbf{u}_n)), \quad n \geq 0,$$

con $\mathbf{u}_0 = \mathbf{y}_0$, $0 \leq \theta \leq 1$, rappresenta la forma vettoriale del metodo di Eulero in avanti se $\theta = 0$, di Eulero all'indietro se $\theta = 1$ e di Crank-Nicolson se $\theta = 1/2$.

Esempio 8.9 (Biologia) Risolviamo il sistema delle equazioni di Lotka-Volterra (8.3) 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 8.1 nel caso di un *sistema* di equazioni differenziali ordinarie, costruiamo un *function handle* f che precisi le componenti del vettore \mathbf{F} come segue:

```
C1=1; C2=1; d1=1; d2=1; b1=0; b2=0;
f=@(t,y)[C1*y(1)*(1-b1*y(1)-d2*y(2));
-C2*y(2)*(1-b2*y(2)-d1*y(1))];
```

A questo punto, eseguiamo il Programma 8.1 con le istruzioni:

```
[t,u]=feuler(f,[0,10],[2 2],20000);
```

Abbiamo quindi risolto il sistema di Lotka-Volterra sull'intervallo temporale $[0, 10]$ con un passo di integrazione $h = 5 \cdot 10^{-4}$.

Il grafico di sinistra di Figura 8.21 rappresenta l'evoluzione nel tempo delle due componenti della soluzione. Come si vede, esse mostrano un andamento periodico. 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 $(1, 1)$ (mentre l'altro è $(0, 0)$).

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 coppie di valori la soluzione si mantiene costante nel tempo. Il punto $(0, 0)$ è un

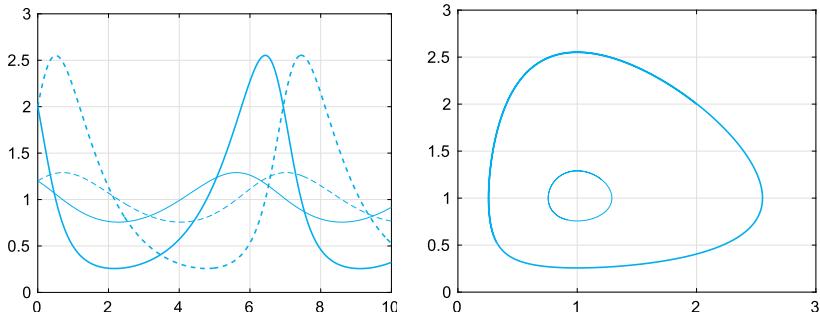


Figura 8.21. Soluzioni numeriche del sistema (8.3). A sinistra, rappresentazione in funzione del tempo dell’evoluzione delle due componenti della soluzione (y_1 in linea piena, y_2 in linea tratteggiata) per due diversi valori del dato iniziale ((2, 2) in linea più marcata, (1.2, 1.2) in linea sottile). A destra, le corrispondenti traiettorie nel piano delle fasi

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 nella Sezione 8.5. 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 (8.41)) derivate per il problema di Cauchy scalare.

Osservazione 8.8 Tutti i programmi descritti in questo Capitolo (sia quelli di MATLAB che quelli presentati in questo libro) possono essere facilmente richiamati anche per risolvere *sistemi di equazioni differenziali ordinarie*. La sintassi da usare è

```
[t, y] = function_name(f, [t0 tf], y0);
```

dove `function_name` è il nome della *function* che si vuole richiamare, `f` è un *function handle*, `[t0, tf]` è l’intervallo temporale su cui si vuole calcolare la soluzione numerica, `y0` è il vettore delle condizioni iniziali. Alternativamente, possiamo utilizzare la sintassi

```
[t, y] = function_name(@f, [t0 tf], y0);
```

a patto che ora `f` sia una *user defined function*. ■

8.10.1 Equazioni differenziali di ordine superiore a uno

Consideriamo ora il caso di un’equazione differenziale di ordine m

$$y^{(m)}(t) = f(t, y, y', \dots, y^{(m-1)}) \quad (8.94)$$

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

$$y(t_0) = y_0, \quad y'(t_0) = y_1, \quad \dots, \quad y^{(m-1)}(t_0) = y_{m-1}.$$

Ponendo

$$w_1(t) = y(t), \quad w_2(t) = y'(t), \quad \dots, \quad w_m(t) = y^{(m-1)}(t),$$

la (8.94) può essere trasformata nel seguente sistema di m equazioni differenziali di ordine 1

$$\begin{aligned} w'_1(t) &= w_2(t), \\ w'_2(t) &= w_3(t), \\ &\vdots \\ w'_{m-1}(t) &= w_m(t), \\ w'_m(t) &= f(t, w_1, \dots, w_m), \end{aligned}$$

con condizioni iniziali

$$w_1(t_0) = y_0, \quad w_2(t_0) = y_1, \quad \dots, \quad w_m(t_0) = y_{m-1}.$$

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 8.10 (Elettrotecnica) Consideriamo il circuito del Problema 8.4 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

$$\begin{cases} v'(t) = w(t), \\ w'(t) = -\frac{1}{LC} \left(\frac{L}{R} + RC \right) w(t) - \frac{2}{LC} v(t) + \frac{e}{LC}, \end{cases} \quad (8.95)$$

con condizioni iniziali $v(0) = 0$, $w(0) = 0$. Esso è stato ricavato a partire dalla seguente equazione differenziale di ordine 2

$$LC \frac{d^2v}{dt^2} + \left(\frac{L}{R_2} + R_1 C \right) \frac{dv}{dt} + \left(\frac{R_1}{R_2} + 1 \right) v = e. \quad (8.96)$$

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 8.1:

```
L=0.1; C=1.e-03; R=10; e=5; LC=L*C;
fsys=@(t,y)[y(2);
-(L/R+R*C)/(LC)*y(2)-2/(LC)*y(1)+e/(LC)];
[t,u]=feuler(fsys,[0,0.1],[0 0],100);
```

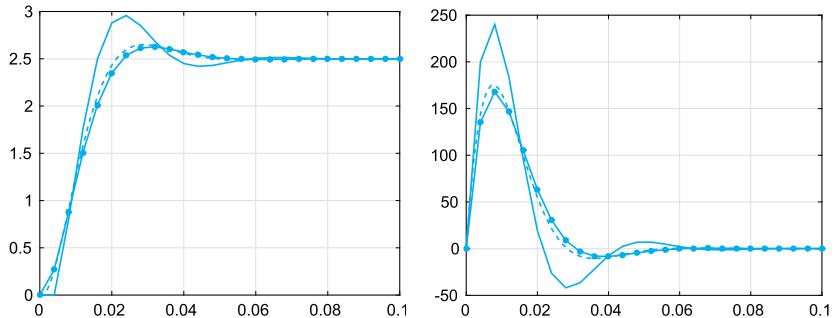


Figura 8.22. Soluzioni numeriche del sistema (8.95). A sinistra, riportiamo la differenza di potenziale $v(t)$ in funzione del tempo, a destra, la sua derivata $w(t)$: in tratteggio la soluzione ottenuta per $h = 0.001$ con il metodo di Eulero in avanti, in linea continua quella generata con lo stesso metodo con $h = 0.004$, con i pallini quella prodotta con il metodo di Newmark (8.100) (con $\zeta = 1/4$ e $\theta = 1/2$) per $h = 0.004$

In Figura 8.22 vengono riportati i valori approssimati di $v(t)$ e $w(t)$. Come ci si aspetta, $v(t)$ tende a $e/2 = 2.5$ Volt per $t \rightarrow \infty$. Per questo problema, la matrice $A = [\partial\mathbf{F}/\partial\mathbf{y}](t, \mathbf{y})$ è indipendente dal tempo e risulta $A = [0, 1; -20000, -200]$. I suoi autovalori sono $\lambda_{1,2} = -100 \pm 100i$, da cui deduciamo che la restrizione su h che garantisce l'assoluta stabilità è $h < -2\operatorname{Re}(\lambda_i)/|\lambda_i|^2 = 0.01$.

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:

$$\begin{cases} y''(t) = f(t, y(t), y'(t)) & t \in (t_0, T], \\ y(t_0) = \alpha_0, & y'(t_0) = \beta_0 \end{cases} \quad (8.97)$$

e costruiamo due successioni u_n e v_n che forniranno, rispettivamente, una approssimazione di $y(t_n)$ e $y'(t_n)$. Si può costruire un metodo di approssimazione elementare nel modo seguente: trovare u_{n+1} tale che

$$\frac{u_{n+1} - 2u_n + u_{n-1}}{h^2} = f(t_n, u_n, v_n), \quad n = 1, \dots, N_h, \quad (8.98)$$

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 $y''(t_n)$ (si vedano (9.17) e l'Esercizio 9.5) consideriamo una approssimazione di ordine 2 anche per $y'(t_n)$, ovvero

$$v_n = \frac{u_{n+1} - u_{n-1}}{2h}, \quad \text{con } v_0 = \beta_0. \quad (8.99)$$

Si ottiene il cosiddetto *metodo leap-frog* (8.98)–(8.99) che è accurato di ordine 2 rispetto a h .

Un metodo più generale è quello *di Newmark* in cui si costruiscono le due successioni (il cui significato è quello spiegato precedentemente)

$$\begin{aligned} u_{n+1} &= u_n + hv_n + h^2 [\zeta f(t_{n+1}, u_{n+1}, v_{n+1}) \\ &\quad + (1/2 - \zeta) f(t_n, u_n, v_n)], \\ v_{n+1} &= v_n + h [(1 - \theta) f(t_n, u_n, v_n) + \theta f(t_{n+1}, u_{n+1}, v_{n+1})], \end{aligned} \quad (8.100)$$

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 garantire 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 (si veda [RT83, Cap. 8.6]).

Nel Programma 8.8 implementiamo il metodo di Newmark. Il vettore `param` serve a precisare nelle sue due componenti i valori dei coefficienti del metodo: `param(1)=zeta` e `param(2)=theta`.

Programma 8.8. newmark: il metodo di Newmark

```
function [t,u]=newmark(odefun,tspan,y0,Nh,param,...  
varargin)  
%NEWMARK Risolve equazioni differenziali del II ord.  
% con il metodo di Newmark.  
% [T,Y]=NEWMARK(ODEFUN,TSPAN,Y0,NH,PARAM) con  
% TSPAN=[T0 TF] integra il sistema di equazioni dif-  
% ferenziali y''= f(t,y,y') dal tempo T0 a TF con  
% condizioni iniziali Y0=(y(t0),y'(t0)) utilizzando  
% il metodo di Newmark su una griglia equispaziata di  
% NH intervalli. Il vettore PARAM contiene, in ordine,  
% i parametri zeta e theta del metodo di Newmark.  
% La funzione ODEFUN(T,Y) deve ritornare uno scalare,  
% la variabile Y e' un array che contiene la funzione  
% soluzione in prima componente e la sua derivata in  
% seconda componente.  
% Ogni riga del vettore soluzione Y corrisponde ad  
% un istante temporale del vettore colonna T.  
% Richiama broyden.m per la risoluzione del sistema  
% non lineare nel caso il metodo sia implicito.  
% Se sono assegnati solo 4 parametri  
% di input, tolleranza e numero max di iterazioni per  
% il metodo di Broyden sono posti pari a tol=1.e-8  
% e kmax=20 rispettivamente, altrimenti  
% [T,Y] = NEWMARK(ODEFUN,TSPAN,Y0,NH,PARAM,TOL,KMAX)  
% permette di specificare anche i parametri per  
% la function broyden.m  
if nargin == 5  
    tol=1.e-8; kmax=20;  
else
```

```

    tol=varargin{1}; kmax=varargin{2};
end
h=(tspan(2)-tspan(1))/Nh; h2=h^2;
t=linspace(tspan(1),tspan(2),Nh+1)';
y0=y0(:); % genera sempre un vettore colonna
u=zeros(Nh+1,2);
u(1,:)=y0'; % trasposta anche di variabili complesse
B0=eye(2); % matrice iniziale per innestare Broyden
zeta=param(1); zeta12=0.5-zeta;
theta=param(2); theta1=1-theta;
for n=1:Nh
    wn=u(n,:).';
    fn=odefun(t(n),wn);
    if theta==0 && zeta==0
        w=[wn(1)+h*wn(2)+h2*zeta12*fn; wn(2)+h*fn];
    else
        F=@(x)[x(1)-wn(1)-h*wn(2)-h2*...
                  (zeta*odefun(t(n+1),x)+zeta12*fn);
                  x(2)-wn(2)-h*(theta*odefun(t(n+1),x)+...
                  theta1*fn)];
        w=broyden(F,B0,wn,tol,kmax);
    end
    u(n+1,:)=w.';
end

```

Esempio 8.11 (Elettrotecnica) Consideriamo nuovamente il circuito del Problema 8.4 e risolviamo l'equazione del second'ordine (8.96) con lo schema di Newmark. In Figura 8.22 confrontiamo l'approssimazione della funzione v calcolata usando lo schema di Eulero in avanti (per $h = 0.001$ in linea tratteggiata e per $h = 0.004$ in linea continua) ed il metodo di Newmark con $\theta = 1/2$ e $\zeta = 1/4$ (linea con i cerchietti), e passo di discretizzazione $h = 0.004$. La miglior accuratezza di quest'ultima approssimazione è dovuta al fatto che il metodo (8.100) è accurato di ordine 2 rispetto a h . ■

8.11 Alcuni problemi *stiff*

Consideriamo il seguente problema di Cauchy lineare [Gea71] come variante del problema modello (8.38):

$$\begin{cases} y'(t) = \lambda(y(t) - g(t)) + g'(t), & t > 0, \\ y(0) = y_0, \end{cases} \quad (8.101)$$

dove g è una funzione regolare e $\lambda < 0$ è molto grande in valore assoluto. La soluzione di (8.101) è

$$y(t) = (y_0 - g(0))e^{\lambda t} + g(t), \quad t \geq 0 \quad (8.102)$$

ed è somma di due componenti che possono presentare caratteristiche molto diverse fra di loro. Nell'intervallo di tempo iniziale di ampiezza $\mathcal{O}(1/\lambda)$ prevale la prima componente, detta anche *soluzione transiente*, mentre per tempi sufficientemente grandi la soluzione $y(t)$ tende ad

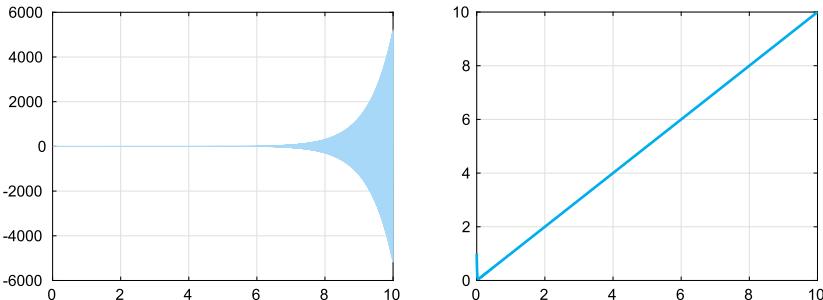


Figura 8.23. Le soluzioni ottenute con il metodo AB3 (8.88) per il problema (8.101) violando la condizione di stabilità (con $h = 0.0055$ a sinistra) e rispettandola ($h = 0.0054$, a destra)

assumere il comportamento della seconda componente (detta *soluzione persistente*), in quanto la prima è diventata trascurabile.

Poniamo in particolare $g(t) = t$, $\lambda = -100$ e $y_0 = 1$ e risolviamo il problema (8.101) 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 nella Sezione 8.4 suggerisce di scegliere $h < 2/100$. Questa restrizione è dettata dalla presenza della componente di $y(t)$ che si comporta come e^{-100t} ed appare del tutto ingiustificata se si pensa al peso che essa ha rispetto all'intera soluzione per tempi sufficientemente grandi (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 AB3 (si veda (8.88)), la cui regione di assoluta stabilità è meno estesa di quella del metodo di Eulero in avanti (si veda la Figura 8.18). Conseguentemente, la restrizione su h diventa ancora più severa, precisamente $h < 0.00545$. Violare anche di poco questa restrizione produce soluzioni del tutto inaccettabili (come mostrato in Figura 8.23 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). In effetti, anche se per t grande la soluzione $y(t)$ ha lo stesso comportamento della sua componente persistente $g(t)$ (che è nota, nel nostro caso specifico è addirittura una retta), per approssimarla correttamente siamo costretti ad imporre una forte limitazione di stabilità sul passo h .

Un problema di questo genere si dice *stiff* o, più precisamente, *stiff sull'intervallo in cui prevale la soluzione persistente*. Di fatto la scelta di h è dettata da vincoli di stabilità e non di accuratezza e l'uso di metodi esplicativi, anche se implementati in forma adattiva, diventa proibitivo.

Anche se i programmi che implementano metodi adattivi non controllano esplicitamente le condizioni di assoluta stabilità, lo stimatore

dell'errore conduce alla scelta di un passo h sufficientemente piccolo per il quale $h\lambda$ cada nella regione di assoluta stabilità.

Consideriamo ora un sistema di equazioni differenziali lineari della forma

$$\mathbf{y}'(t) = \mathbf{A}\mathbf{y}(t) + \boldsymbol{\varphi}(t), \quad \mathbf{A} \in \mathbb{R}^{n \times n}, \quad \boldsymbol{\varphi}(t) \in \mathbb{R}^n, \quad (8.103)$$

in cui \mathbf{A} abbia n autovalori distinti λ_j , $j = 1, \dots, n$, con $\operatorname{Re}(\lambda_j) < 0$.

La soluzione esatta di (8.103) è

$$\mathbf{y}(t) = \sum_{j=1}^n C_j e^{\lambda_j t} \mathbf{v}_j + \boldsymbol{\psi}(t), \quad (8.104)$$

dove C_1, \dots, C_n sono n costanti e $\{\mathbf{v}_j\}$ è una base formata dagli autovettori di \mathbf{A} , mentre $\boldsymbol{\psi}(t)$ è una soluzione particolare del sistema differenziale.

Le componenti $C_j e^{\lambda_j t} \mathbf{v}_j$ sono soluzioni transienti e per t grande, $\mathbf{y}(t) \simeq \boldsymbol{\psi}(t)$ (la componente persistente della soluzione). Se $|\operatorname{Re}(\lambda_j)|$ è grande, allora la corrispondente componente transiente tenderà a zero per $t \rightarrow \infty$ in breve tempo, mentre se $|\operatorname{Re}(\lambda_j)|$ è piccolo essa decadrà a zero in tempi più lunghi. Approssimando la soluzione con un metodo che non sia A-stabile, nell'ipotesi che le parti immaginarie degli autovalori λ_j siano fra loro confrontabili o tutte nulle, la componente transiente con il massimo valore di $|\operatorname{Re}(\lambda_j)|$ sarà quella che comporterà la restrizione più pesante sul passo di integrazione pur essendo quella che tenderà a zero più velocemente di tutte.

Un parametro utilizzato per misurare il carattere *stiff* del sistema è

$$r_s = \frac{\max_j |\operatorname{Re}(\lambda_j)|}{\min_j |\operatorname{Re}(\lambda_j)|},$$

tuttavia da solo questo numero non è sufficientemente significativo. Infatti il carattere *stiff* di un sistema dipende non solo dal rapporto r_s , ma anche dalle quantità $|\lambda_j|$ (se gli autovalori non sono reali), dalle condizioni iniziali, dalla componente persistente e dall'intervallo temporale in cui si vuole approssimare la soluzione. D'altro canto non si può affermare che il carattere *stiff* dipenda solo dalla soluzione esatta del sistema. Esistono infatti esempi di sistemi differenti, gli uni *stiff* e gli altri non *stiff*, che tuttavia ammettono la stessa soluzione esatta (si veda ad esempio [Lam91, Cap. 6]).

Come si può stabilire allora se un sistema è *stiff* o meno? Riportiamo la seguente definizione proposta in [Lam91, pag. 220]:

Un sistema di equazioni differenziali ordinarie è detto stiff se, una volta approssimato con un metodo numerico caratterizzato da una regione di assoluta stabilità di estensione finita, obbliga quest'ultimo, per ogni condizione iniziale per la quale il problema ammetta soluzione, ad utilizzare un passo di discretizzazione

ecessivamente piccolo rispetto a quello necessario per descrivere ragionevolmente l'andamento della soluzione esatta.

Nel caso del problema (8.101) (o (8.103)) il sistema risulta *stiff* non nell'intervallo iniziale dove la soluzione varia molto rapidamente e quindi la scelta di un h piccolo è giustificata dall'esigenza di catturare bene lo *strato limite* (ovvero la regione a forte gradiente), bensì nell'intervallo successivo dove la soluzione è a derivata limitata. In quest'ultimo intervallo il transiente più veloce, pur essendosi di fatto esaurito perché trascurabile rispetto alle altre componenti, impone ancora una scelta restrittiva di h dettata dalla stabilità.

I metodi impliciti A-stabili (ovvero quelli la cui regione di assoluta stabilità include il semipiano complesso $\text{Re}(z) < 0$) e con scelta adattiva del passo sono quelli più efficienti per risolvere problemi *stiff*.

Il carattere隐式 di questi metodi rende ogni passo molto più costoso rispetto ad un passo di un corrispondente metodo esplicito, tuttavia trae vantaggio dal fatto di poter utilizzare pochi passi di lunghezza maggiore. I metodi esplicativi, pur garantendo una soluzione accurata e stabile per h piccoli, risulteranno invece proibitivi per l'alto costo computazionale che deriva dalla forte limitazione sul passo di integrazione.

Osservazione 8.9 L'algoritmo implementato nella *function* `ode15s` di MATLAB si basa su metodi *multistep* e su formule di differenziazione numerica, quali ad esempio le formule BDF introdotte nella Sezione 8.9. L'ordine di accuratezza del risolutore è variabile ed al più pari a 5. Questo metodo risulta particolarmente efficiente anche per problemi non *stiff* in cui la matrice Jacobiana di $\mathbf{f}(t, \mathbf{y})$ è costante o ha piccole variazioni.

MATLAB
`ode15s`

Il programma `ode23tb` di MATLAB implementa invece un metodo Runge-Kutta implicito il cui primo stadio è basato sulla formula del trapezio, mentre il secondo stadio è basato su una formula di differenziazione all'indietro di ordine 2 (si veda (8.90)).

`ode23tb`

La *function* `ode23s` di MATLAB implementa un metodo *multistep* lineare implicito basato sui metodi di Rosenbrock [SR97].

`ode23s`

Il package `odepkg` di Octave-Forge raccoglie varie *function* per la risoluzione di sistemi di equazioni stiff e sistemi di *Differential Algebraic Equations* (DAE) che implementano diversi metodi Runge-Kutta impliciti e BDF (esempi sono `ode2r`, `ode5r`, `odebda`, `oders` e `odesx`). ■

`odepkg`

Esempio 8.12 Consideriamo il sistema $\mathbf{y}' = \mathbf{A}\mathbf{y}$ 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

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ -\lambda_1 \lambda_2 & \lambda_1 + \lambda_2 \end{bmatrix},$$

dove λ_1 e λ_2 sono due numeri negativi distinti con $|\lambda_1| \gg |\lambda_2|$. La matrice \mathbf{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 (8.104) la soluzione del sistema è pari a

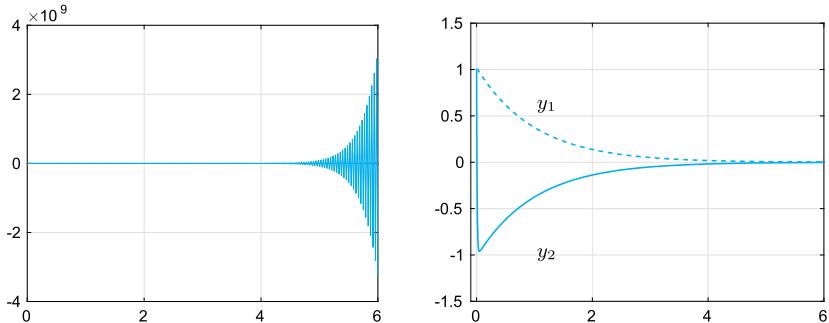


Figura 8.24. Le soluzioni calcolate per il problema dell’Esempio 8.12 per $h = 0.0207$ (a sinistra) e $h = 0.01$ (a destra). Nel primo caso la condizione $h < 2/|\lambda_1| = 0.02$ è violata ed il metodo è instabile. Nel secondo caso si osserva la forte variazione iniziale del transiente veloce y_2 . Si tenga conto della scala completamente diversa dei due grafici

$$\mathbf{y}(t) = \begin{pmatrix} C_1 e^{\lambda_1 t} + C_2 e^{\lambda_2 t} \\ C_1 \lambda_1 e^{\lambda_1 t} + C_2 \lambda_2 e^{\lambda_2 t} \end{pmatrix}^T. \quad (8.105)$$

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 $\lambda_1 = -100$, $\lambda_2 = -1$ (per cui $r_s = 100$), $y_{1,0} = y_{2,0} = 1$. In Figura 8.24 riportiamo le soluzioni calcolate violando (a sinistra) o rispettando (a destra) la condizione di stabilità $h < 1/50$. ■

Esempio 8.13 (Equazione di Van der Pol) Uno dei problemi *stiff* non lineari più studiati è dato dall’equazione di Van der Pol

$$\frac{d^2x}{dt^2} = \mu(1-x^2)\frac{dx}{dt} - x, \quad (8.106)$$

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 $\mathbf{y} = (x, z)^T$, con $z = dx/dt$, la (8.106) è equivalente al seguente sistema non lineare del prim’ordine

$$\mathbf{y}' = \mathbf{F}(t, \mathbf{y}) = \begin{bmatrix} z \\ -x + \mu(1-x^2)z \end{bmatrix}. \quad (8.107)$$

Tale sistema diventa sempre più *stiff* quanto più cresce il parametro μ . Nella soluzione compaiono infatti due componenti che al crescere di μ presentano

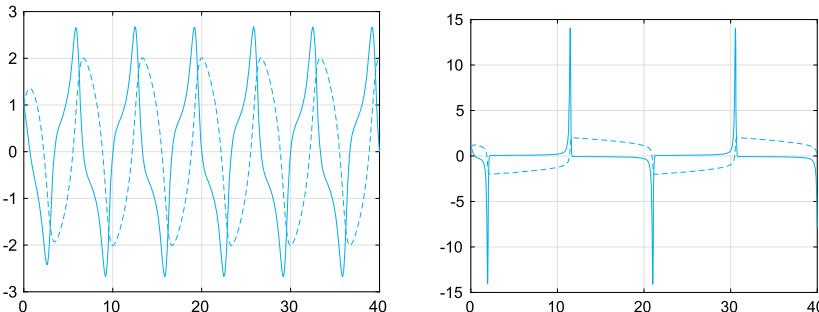


Figura 8.25. Andamento delle componenti x (in linea tratteggiata) e z (in linea continua) della soluzione \mathbf{y} del sistema (8.107) per $\mu = 1$ (a sinistra) e $\mu = 10$ (a destra)

Tabella 8.1. Andamento del numero di passi di integrazione per vari metodi di approssimazione al crescere del parametro μ . I risultati si riferiscono a *run* svolti in MATLAB

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

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 . Si veda per un esempio la Figura 8.25.

Se risolviamo (8.106) in ambiente MATLAB usando `ode23` e `ode45`, ci rendiamo conto che questi metodi sono troppo onerosi quando μ è grande. Se $\mu = 100$ e la condizione iniziale è $\mathbf{y} = (1, 1)^T$, `ode23` richiede 7835 passi e `ode45` ben 23473 passi per integrare fra $t = 0$ e $t = 100$. Questo perché sono metodi esplicativi e quindi non adatti per problemi *stiff*. L'alternativa è utilizzare metodi impliciti come ad esempio `ode23s` o `ode15s`. La differenza in termini di numero di passi è notevole, come indicato in Tabella 8.1. 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*). ■

Si vedano gli Esercizi 8.18–8.21.



Riassumendo

- I metodi Runge-Kutta (RK) sono metodi ad un passo che richiedono diverse valutazioni della funzione f per passare dal tempo t_n a quello successivo t_{n+1} . Possono essere sia impliciti che esplicativi e di vari ordini di convergenza. Essi sono spesso implementati unita-

mente a strategie di calcolo adattivo del passo di discretizzazione. In particolare le versioni implicite sono adatte a risolvere problemi *stiff*;

2. i metodi *multistep* lineari sono metodi a più passi e necessitano di una sola valutazione di f per passare da t_n a t_{n+1} . A differenza dei metodi RK, essi richiedono di memorizzare più passi calcolati precedentemente. Per essere innestati, i metodi *multistep* hanno bisogno di più dati iniziali, calcolabili ad esempio con i metodi RK;
3. Per risolvere i problemi *stiff* si utilizzano solitamente metodi implicitti con scelta adattiva del passo.

8.12 Alcuni esempi

Concludiamo questo capitolo proponendo e risolvendo tre esempi non elementari di sistemi di equazioni differenziali ordinarie.

8.12.1 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

$$\ddot{\mathbf{x}} = \frac{1}{m} \left(\mathbf{F} - \frac{m \dot{\mathbf{x}}^T \mathbf{H} \dot{\mathbf{x}} + \nabla \Phi^T \mathbf{F}}{|\nabla \Phi|^2} \nabla \Phi \right) \quad \text{per } t > 0. \quad (8.108)$$

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 $2\mathbf{x}$, con \mathbf{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 \mathbf{H} è una matrice diagonale di coefficienti pari a 2. Al sistema (8.108) dobbiamo aggiungere le condizioni iniziali $\mathbf{x}(0) = \mathbf{x}_0$ e $\dot{\mathbf{x}}(0) = \mathbf{v}_0$.

Al fine di risolvere numericamente (8.108) 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

$$\lambda = \frac{m(y_4, y_5, y_6)^T \mathbf{H}(y_4, y_5, y_6) + \nabla \Phi^T \mathbf{F}}{|\nabla \Phi|^2},$$

otteniamo per $i = 1, 2, 3$

$$\begin{aligned} \dot{y}_i &= y_{3+i}, \\ \dot{y}_{3+i} &= \frac{1}{m} \left(F_i - \lambda \frac{\partial \Phi}{\partial y_i} \right). \end{aligned} \quad (8.109)$$

Applichiamo i metodi di Eulero e di Crank-Nicolson. Dapprima è necessario definire una *user-defined function* (`fvinc.m` del Programma 8.9) che restituisca le espressioni dei termini di destra delle equazioni del sistema (8.109).

Programma 8.9. `fvinc`: termine forzante per il problema del pendolo sferico

```
function [f]=fvinc(t,y)
% FVINC Function per l'esempio del pendolo sferico
[n,m]=size(y); f=zeros(n,m);
phix=2*y(1); phiy=2*y(2); phiz=2*y(3);
H=2*eye(3); mass=1;
xp=zeros(3,1); xp(1:3)=y(4:6);
F=[0;0;-mass*9.8];
G=[phix;phiy;phiz];
lambda=(mass*xp'*H*xp+F'*G)/(G'*G);
f(1:3)=y(4:6);
for k=1:3;
    f(k+3)=(F(k)-lambda*G(k))/mass;
end
```

Supponiamo inoltre che le condizioni iniziali siano date dal vettore $\mathbf{y}_0 = [0, 1, 0, .8, 0, 1.2]$ 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);
```

(analogamente si possono richiamare i metodi di Eulero all'indietro e di Crank-Nicolson), dove nt è il numero di intervalli (di ampiezza costante) impiegati per la discretizzazione dell'intervallo `[tspan(1),tspan(2)]`. Nei grafici di Figura 8.26 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 essa 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 8.27, mentre il metodo di Crank-Nicolson (di ordine 2) con soli 1000 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 $r = 0.0928$ 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

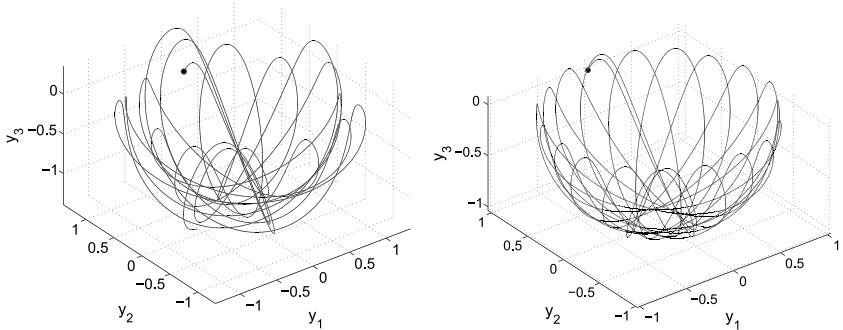


Figura 8.26. Le traiettorie ottenute con il metodo di Eulero esplicito con $h = 0.00025$ (a sinistra) e $h = 0.00025$ (a destra). Il punto annerito indica il dato iniziale

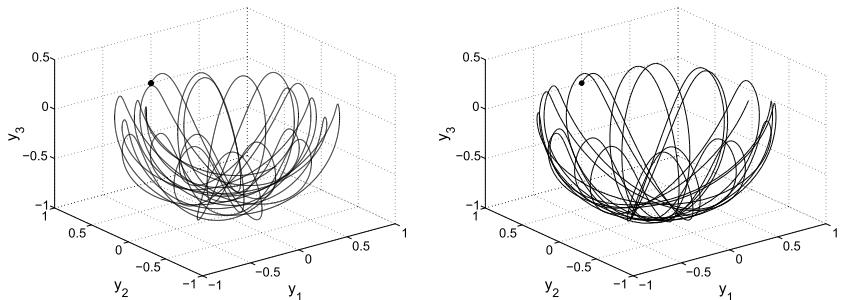


Figura 8.27. Le traiettorie ottenute con il metodo di Eulero implicito con $h = 0.00125$ (a sinistra) e con il metodo di Crank-Nicolson con $h = 0.025$ (a destra)

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

```
y0=[0,1,0,.8,0,1.2]; tspan=[0,25];
[t1,y1]=ode23(@fvinc,tspan,y0);
```

ed otteniamo le soluzioni presentate in Figura 8.28.

I due metodi hanno usato 786 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 con quelli di `ode23`. In particolare

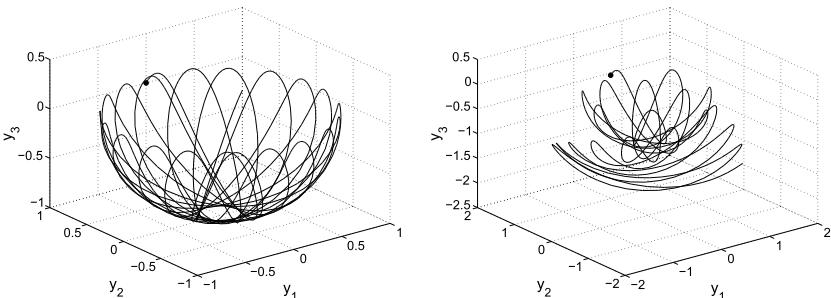


Figura 8.28. Le traiettorie ottenute con i metodi `ode23` (a sinistra) e `ode45` (a destra) di MATLAB con tolleranza di *default* sull'accuratezza (pari a 10^{-3}). Nel secondo caso il controllo sull'errore fallisce e la soluzione che si trova è meno accurata

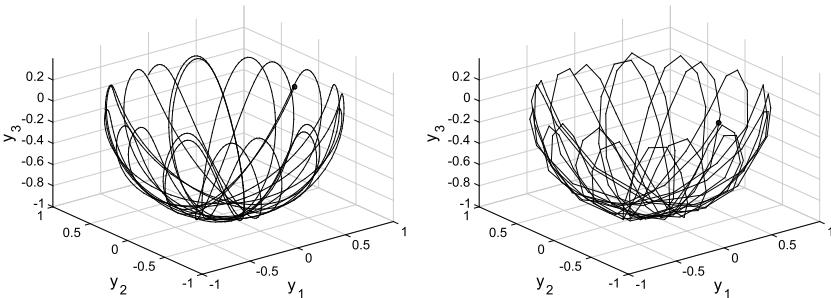


Figura 8.29. Le traiettorie ottenute richiamando i metodi `ode23` (a sinistra) e `ode45` (a destra) di Octave con una tolleranza `tol=1.e-03` sull'accuratezza

`ode23` utilizza 1751 nodi di discretizzazione e fornisce un residuo pari a $r = 0.003$, mentre `ode45` utilizza 1089 nodi di discretizzazione e produce un residuo pari a $r = 0.060$.

In Octave, fissando la tolleranza sull'accuratezza pari a 10^{-3} per il calcolo adattivo del passo h , `ode23` richiede 1004 passi producendo un residuo pari a $r = -0.0965$, mentre `ode45` richiede 185 passi con un residuo pari a $r = -0.0891$ (si veda la Figura 8.29). Si noti che le soluzioni prodotte dalle *function* `ode23` e `ode45` di Octave sono ben diverse da quelle prodotte dalle omonime *function* di MATLAB in quanto, pur riportando lo stesso nome, le *function* in MATLAB e Octave implementano algoritmi diversi (si veda l'Osservazione 8.4).

8.12.2 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 uti-

lizzando 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, con M_t quella della Terra e con M_m quella di Marte. Essendo la massa del Sole circa 330000 volte più grande di 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

$$\mathbf{F}_t = \mathbf{F}_{ts} + \mathbf{F}_{tm} = M_t \frac{d^2 \mathbf{x}_t}{dt^2}, \quad (8.110)$$

dove $\mathbf{x}_t = (x_t, y_t)^T$ denota la posizione della Terra rispetto al Sole, mentre \mathbf{F}_{ts} e \mathbf{F}_{tm} denotano rispettivamente la forza esercitata dal Sole e da Marte sulla Terra. Applicando la legge di gravitazione universale, indicando con G la costante di gravitazione universale e con \mathbf{x}_m la posizione di Marte rispetto al Sole, la (8.110) diventa

$$M_t \frac{d^2 \mathbf{x}_t}{dt^2} = -GM_t M_s \frac{\mathbf{x}_t}{|\mathbf{x}_t|^3} + GM_t M_m \frac{\mathbf{x}_m - \mathbf{x}_t}{|\mathbf{x}_m - \mathbf{x}_t|^3}.$$

Considerando come unità di lunghezza l'unità astronomico (1 AU), come unità di tempo l'anno (1 yr) e definendo la massa del sole $M_s = \frac{4\pi^2(1 \text{ AU})^3}{G(1 \text{ yr})^2}$, possiamo adimensionalizzare l'equazione. Denotando ancora con \mathbf{x}_t , \mathbf{x}_m , \mathbf{x}_s e t le variabili adimensionalizzate, si ottiene la seguente equazione per il moto della Terra:

$$\frac{d^2 \mathbf{x}_t}{dt^2} = 4\pi^2 \left(\frac{M_m}{M_s} \frac{\mathbf{x}_m - \mathbf{x}_t}{|\mathbf{x}_m - \mathbf{x}_t|^3} - \frac{\mathbf{x}_t}{|\mathbf{x}_t|^3} \right). \quad (8.111)$$

Con calcoli simili si perviene all'equazione analoga per il pianeta Marte

$$\frac{d^2 \mathbf{x}_m}{dt^2} = 4\pi^2 \left(\frac{M_t}{M_s} \frac{\mathbf{x}_t - \mathbf{x}_m}{|\mathbf{x}_t - \mathbf{x}_m|^3} - \frac{\mathbf{x}_m}{|\mathbf{x}_m|^3} \right). \quad (8.112)$$

Il sistema del second'ordine (8.111)–(8.112) si riduce immediatamente ad un sistema di 8 equazioni di ordine 1. Il Programma 8.10 consente la valutazione di una *function* contenente i termini di destra del sistema (8.111)–(8.112).

Programma 8.10. threebody: termine forzante per il problema semplificato dei tre corpi

```
function f=threebody(t,y)
% THREEBODY Function per l'esempio dei tre corpi
[n,m]=size(y); f=zeros(n,m); 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); 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);
```

Confrontiamo fra loro il metodo di Crank-Nicolson (implicito) ed il metodo adattivo di 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 nodi 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 8.30 mostrano che i due metodi sono entrambi in grado di riprodurre le orbite ellittiche dei due pianeti attorno al Sole. La function `ode23` di MATLAB ha richiesto solo 543 nodi di discretizzazione per generare una soluzione più accurata di quella generata da

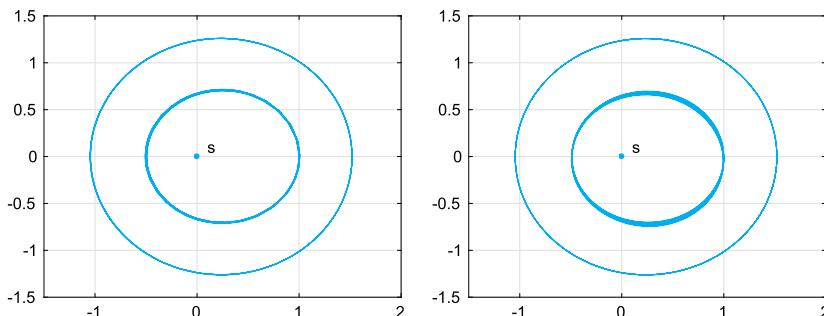


Figura 8.30. L'orbita della Terra (più interna) e quella di Marte rispetto al Sole calcolate con il metodo adattivo `ode23` con 543 nodi di discretizzazione (*a sinistra*) e con quello di Crank-Nicolson con 2000 nodi di discretizzazione (*a destra*)

un metodo implicito dello stesso ordine di accuratezza, ma che non usa adattività del passo. La *function* `ode23` di Octave richiede 847 nodi di discettizzazione per generare una soluzione con una tolleranza pari a 10^{-3} .

8.12.3 Cinetica chimica

Vogliamo studiare l'evoluzione di specie coinvolte in reazioni chimiche in mezzi omogenei e, più precisamente, la loro concentrazione che supponiamo dipenda solo dal tempo. Spesso nelle reazioni chimiche coesistono specie *veloci*, la cui concentrazione evolve in un tempo molto breve, e *lente*, che evolvono invece in tempi maggiori. Presentiamo nel seguito un modello matematico che descrive una semplificazione, comunque significativa, di questo fenomeno. Si tratta di un modello per lo studio dell'evoluzione delle due specie $y_1(t)$ e $y_2(t)$, detto di Davis-Skodje (si veda, ad esempio, [VGCN05])

$$\begin{cases} \frac{dy_1}{dt} = \frac{1}{\varepsilon} \left(-y_1 + \frac{y_2}{1+y_2} \right) - \frac{y_2}{(1+y_2)^2}, & t > 0 \\ \frac{dy_2}{dt} = -y_2, & t > 0 \\ y_1(0) = y_{1,0}, \quad y_2(0) = y_{2,0}, \end{cases} \quad (8.113)$$

dove $\varepsilon > 0$, $y_{1,0}$ e $y_{2,0}$ sono assegnati.

La soluzione esatta del sistema è:

$$\begin{aligned} y_1(t) &= \left(y_{1,0} - \frac{y_{2,0}}{1+y_{2,0}} \right) e^{-t/\varepsilon} + \frac{y_{2,0} e^{-t}}{1+y_{2,0} e^{-t}} \\ y_2(t) &= y_{2,0} e^{-t}. \end{aligned}$$

La grandezza $1/\varepsilon$ è una misura di quanto il sistema sia *stiff*: maggiore è $1/\varepsilon$ e maggiore è il divario tra le scale temporali di evoluzione delle due specie e, di conseguenza, maggiore è la complessità computazionale della simulazione numerica.

Per risolvere numericamente il sistema (8.113) con $\varepsilon = 10^{-6}$ e condizione iniziale $\mathbf{y}_0 = (1.5, 1)^T$, abbiamo definito il *function handle*

```
epsilon=1.e-6;
funds=@(t,y)[-1/epsilon*y(1)+((1/epsilon-1)*y(2)+...
    1/epsilon*y(2)*y(2))/(1+y(2))^2;
    -y(2)];
```

ed abbiamo richiamato la *function* `ode23s` di MATLAB con i seguenti comandi:

```
y0=[1.5,1];
tspan=[0,10];
[t,y]=ode23s(funds,tspan,y0);
```

In Tabella 8.2 è mostrato il numero di passi temporali richiesti dai metodi esplicativi `ode23`, `ode45` e da quelli impliciti `ode23s`, `ode15s` di MATLAB. Dal confronto è evidente la superiorità dei metodi `ode23s`, `ode15s`, scritti appositamente per equazioni *stiff*.

Tabella 8.2. Numero di passi di integrazione richiesti da alcuni metodi di approssimazione al diminuire del parametro ε per risolvere il problema (8.113). I risultati si riferiscono a *run* svolti in MATLAB

ε	ode23	ode45	ode23s	ode15s
10^{-2}	409	1241	73	73
10^{-3}	3991	12081	84	81
10^{-4}	39808	120553	87	85

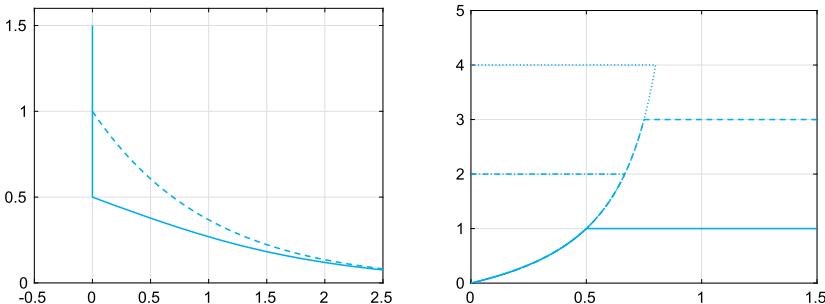


Figura 8.31. A sinistra, le soluzioni numeriche ($y_1(t)$ (in linea continua) e $y_2(t)$ (in linea tratteggiata)) del sistema (8.113) corrispondente alle condizioni iniziali $\mathbf{y}_0 = (1.5, 1)$. A destra, le traiettorie del modello (8.113) nel piano delle fasi per alcuni dati iniziali: $\mathbf{y}_0 = (1.5, 1)$ (in linea continua), $\mathbf{y}_0 = (1.5, 3)$ (in linea tratteggiata), $\mathbf{y}_0 = (0, 2)$ (in linea tratto-punto), $\mathbf{y}_0 = (0, 4)$ (in linea punteggiata). In entrambe le figure $\varepsilon = 10^{-6}$

In Figura 8.31, a sinistra, possiamo osservare le soluzioni numeriche: la concentrazione y_1 varia molto velocemente in un intervallo iniziale di ampiezza proporzionale a ε e molto più lentamente nel periodo temporale successivo, mentre y_2 varia lentamente ed in modo uniforme nell'arco di tutta la simulazione.

Nel grafico di destra di Figura 8.31 sono rappresentate le traiettorie del modello (8.113) nel piano delle fasi, quando $\varepsilon = 10^{-6}$ e per differenti condizioni iniziali $\mathbf{y}_0 = (y_{1,0}, y_{2,0})$. I tratti orizzontali delle traiettorie vengono percorsi nel brevissimo intervallo di tempo iniziale di ampiezza $\mathcal{O}(\varepsilon)$, mentre la parte curva di traiettoria è percorsa nel restante intervallo temporale di ampiezza $10 - \mathcal{O}(\varepsilon)$. Dall'analisi del piano delle fasi associato al sistema di equazioni differenziali ordinarie è possibile estrarre informazioni caratteristiche del fenomeno chimico.

8.13 Cosa non vi abbiamo detto

Per una completa derivazione dell'intera famiglia dei metodi Runge-Kutta rimandiamo a [But87], [Lam91] e [QSS07, Cap. 11].

Per la derivazione e l'analisi dei metodi *multistep* rimandiamo a [Arn73] e [Lam91].

8.14 Esercizi

Esercizio 8.1 Si applichino il metodo di Eulero in avanti e di Eulero all'indietro per la risoluzione del seguente problema di Cauchy

$$y' = \sin(t) + y, \quad t \in (0, 1], \text{ con } y(0) = 0, \quad (8.114)$$

e se ne verifichi la convergenza lineare.

Esercizio 8.2 Si consideri il problema di Cauchy

$$y' = -te^{-y}, \quad t \in (0, 1], \text{ con } y(0) = 0. \quad (8.115)$$

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 8.3 Il metodo di Eulero implicito applicato al problema (8.115) 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:

$$\text{per } k = 0, 1, \dots, \text{ calcolare } u_{n+1}^{(k+1)} = \phi(u_{n+1}^{(k)}), \quad \text{con } u_{n+1}^{(0)} = u_n.$$

Si determinino eventuali restrizioni su h affinché tale metodo converga.

Esercizio 8.4 Si applichi il metodo di Crank-Nicolson per la risoluzione di (8.114) e si verifichi la convergenza di ordine 2.

Esercizio 8.5 Si verifichi che il metodo di Crank-Nicolson può essere ottenuto a partire dalla seguente forma integrale del problema di Cauchy (8.5)

$$y(t) - y_0 = \int_{t_0}^t f(\tau, y(\tau)) d\tau,$$

approssimando l'integrale con la formula del trapezio (4.26).

Esercizio 8.6 Si risolva il problema modello (8.38) con $\lambda = -1 + i$ con il metodo di Eulero in avanti. Per quali valori di h il metodo è assolutamente stabile?

Esercizio 8.7 Si dimostri la formula (8.53).

Esercizio 8.8 Si dimostri la diseguaglianza (8.58).

Esercizio 8.9 Si dimostri la disuguaglianza (8.59).

Esercizio 8.10 Sotto quali condizioni su h il seguente metodo (detto di *Eulero modificato*)

$$u_{n+1}^* = u_n + hf(t_n, u_n), \quad u_{n+1} = u_n + hf(t_{n+1}, u_{n+1}^*) \quad (8.116)$$

è assolutamente stabile?

Esercizio 8.11 Si mostri che il metodo RK2 (o di Heun) definito in (8.93) è consistente. Si scriva un programma `MAT&OCT` che lo implementa e si verifichi sperimentalmente l'ordine 2 di convergenza rispetto a h , risolvendo il problema di Cauchy (8.114).

Esercizio 8.12 Si mostri che, per λ reale e negativo, il metodo RK2 (o di Heun) definito in (8.93) è assolutamente stabile se $-2 < h\lambda < 0$.

Esercizio 8.13 Si verifichi che il metodo RK3 (8.82) è consistente. Si scriva un programma `MAT&OCT` che lo implementa e si verifichi sperimentalmente l'ordine 3 di accuratezza rispetto a h , risolvendo il problema di Cauchy (8.114).

Esercizio 8.14 Si mostri che, per λ reale e negativo, il metodo RK3 (8.82) è assolutamente stabile se $-2.5 < h\lambda < 0$.

Esercizio 8.15 (Termodinamica) Si risolva l'equazione (8.1) con i metodi di Crank-Nicolson e con il metodo RK2 quando il corpo in esame è un cubo di lato 1 m e massa pari a 1 kg. Si supponga $T_0 = 180$ K, $T_e = 200$ K, $\gamma = 0.5$ e $C = 100$ J/(kg/K). Si confrontino i risultati ottenuti usando $h = 20$ e $h = 10$ per $t \in [0, 200]$ secondi.

Esercizio 8.16 Si individui, tramite `MAT&OCT`, la regione di assoluta stabilità del metodo di Heun.

Esercizio 8.17 Si risolva il problema di Cauchy (8.30) con il metodo di Heun e se ne verifichi l'ordine.

Esercizio 8.18 Lo spostamento $x(t)$ di un sistema vibrante costituito da un corpo di un certo peso e 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 8.19 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 8.20 (Sport) Si risolva con `ode23` il Problema 8.3 supponendo che la velocità iniziale della palla sia $\mathbf{v}(0) = v_0(\cos(\phi), 0, \sin(\phi))^T$ con $v_0 = 38$ m/s e ϕ 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 (ovvero la quota $z = 0$)?

Esercizio 8.21 (Chimica) Il seguente sistema

$$\begin{cases} \frac{dy_1}{dt} = \frac{1}{\varepsilon}(-5y_1 - y_1 y_2 + 5y_2^2 + y_3) + y_2 y_3 - y_1, & t > 0 \\ \frac{dy_2}{dt} = \frac{1}{\varepsilon}(10y_1 - y_1 y_2 - 10y_2^2 + y_3) - y_2 y_3 + y_1, & t > 0 \\ \frac{dy_3}{dt} = \frac{1}{\varepsilon}(y_1 y_2 - y_3) - y_2 y_3 + y_1, & t > 0 \\ y_1(0) = y_{1,0} \\ y_2(0) = y_{2,0}, \quad y_3(0) = y_{3,0}, \end{cases} \quad (8.117)$$

con $y_{1,0}$, $y_{2,0}$ e $y_{3,0}$ assegnati, simula l'evoluzione delle concentrazioni di tre specie di una reazione chimica. Fissando il dato iniziale $\mathbf{y}_0 = (1, 0.5, 0)^T$ e $\varepsilon = 10^{-2}$, si risolva numericamente (8.117) per $t \in [0, 10]$, con le *function* `ode23` e `ode23s` di MATLAB e si dica se il sistema è *stiff*. Infine si rappresenti graficamente la soluzione nello spazio delle fasi al variare del dato iniziale $\mathbf{y}_0 = (y_{1,0}, y_{2,0}, y_{3,0})^T$ con $0 \leq y_{i,0} \leq 1$ per $i = 1, 3$.

Metodi numerici per problemi ai limiti stazionari ed evolutivi

I *problemì ai limiti* (o ai valori al bordo) 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.

Nel caso in cui oltre alle derivate rispetto alle variabili spaziali compaiano anche derivate rispetto alla variabile temporale (indicata con t), i corrispondenti problemi differenziali verranno detti *problemì ai limiti e ai valori iniziali*, o anche *problemì ai limiti evolutivi*. 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

$$-u''(x) = f(x), \quad x \in (a, b), \quad (9.1)$$

o, in più dimensioni,

$$-\Delta u(\mathbf{x}) = f(\mathbf{x}), \quad \mathbf{x} = (x_1, \dots, x_d)^T \in \Omega, \quad (9.2)$$

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

$$\frac{\partial u}{\partial x_i}(\mathbf{x}^0) = \lim_{h \rightarrow 0} \frac{u(\mathbf{x}^0 + h\mathbf{e}_i) - u(\mathbf{x}^0)}{h}, \quad (9.3)$$

dove \mathbf{e}_i è l' i -esimo vettore unitario di \mathbb{R}^d .

2. L'equazione del calore

$$\frac{\partial u(x, t)}{\partial t} - \mu \frac{\partial^2 u(x, t)}{\partial x^2} = f(x, t), \quad x \in (a, b), \quad t > 0, \quad (9.4)$$

o, in più dimensioni,

$$\frac{\partial u(\mathbf{x}, t)}{\partial t} - \mu \Delta u(\mathbf{x}, t) = f(\mathbf{x}, t), \quad \mathbf{x} \in \Omega, \quad t > 0, \quad (9.5)$$

dove $\mu > 0$ è un coefficiente assegnato che rappresenta la diffusività termica e f è nuovamente una funzione assegnata.

3. L'equazione delle onde

$$\frac{\partial^2 u(x, t)}{\partial t^2} - c \frac{\partial^2 u(x, t)}{\partial x^2} = 0, \quad x \in (a, b), \quad t > 0, \quad (9.6)$$

o, in più dimensioni,

$$\frac{\partial^2 u(\mathbf{x}, t)}{\partial t^2} - c \Delta u(\mathbf{x}, t) = 0, \quad \mathbf{x} \in \Omega, \quad t > 0, \quad (9.7)$$

dove c è una costante positiva assegnata.

L'equazione di Poisson (9.2) è un classico esempio di *problema ellittico*, l'equazione del calore (9.5) è un esempio di *problema parabolico*, mentre l'equazione delle onde (9.7) è un esempio di *problema iperbolico*. Le equazioni appena presentate ammettono infinite soluzioni. Al fine di ottenere un'unica soluzione devono essere imposte opportune condizioni sul bordo del dominio computazionale (l'intervallo (a, b) nel caso monodimensionale o il dominio Ω nel caso multidimensionale) e, per problemi evolutivi, anche delle opportune condizioni iniziali al tempo $t = 0$.

Per una trattazione più completa di problemi ai limiti ellittici e di problemi ai limiti ed ai valori iniziali parabolici ed iperbolici rimandiamo ad esempio a [Eva98], [Sal10], mentre per la loro approssimazione numerica citiamo [Qua16], [QV94], [EG04], [LeV02], [Tho06] o [Lan03].

9.1 Alcuni problemi

Problema 9.1 (Idrogeologia) In alcuni casi, lo studio della filtrazione delle acque nel sottosuolo può essere ricondotto alla risoluzione dell'equazione (9.2). Consideriamo una regione tridimensionale Ω occupata da un mezzo poroso (come la terra o l'argilla). 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

$$\mathbf{q} = -K \nabla \phi, \quad (9.8)$$

dove K è la costante di conducibilità idraulica del mezzo poroso e $\nabla \phi$ indica il gradiente di ϕ rispetto alle coordinate spaziali. Se la densità del fluido è costante, allora il principio di conservazione della massa fornisce l'equazione $\operatorname{div} \mathbf{q} = 0$, dove $\operatorname{div} \mathbf{q}$ è la *divergenza* del vettore \mathbf{q} ed è definita come

$$\operatorname{div} \mathbf{q} = \sum_{i=1}^3 \frac{\partial q_i}{\partial x_i}$$

Allora, per la (9.8) si ha che ϕ soddisfa all'equazione di Poisson $\Delta \phi = 0$ (si veda l'Esercizio 9.1). ■

Problema 9.2 (Termodinamica) Sia $\Omega \subset \mathbb{R}^3$ un volume occupato da un materiale. Indicando rispettivamente con $\mathbf{J}(\mathbf{x}, t)$ e $T(\mathbf{x}, t)$ il flusso di calore e la temperatura del materiale, la legge di Fourier stabilisce che il flusso sia proporzionale alla variazione di temperatura T , ovvero

$$\mathbf{J}(\mathbf{x}, t) = -k \nabla T(\mathbf{x}, t),$$

dove k è una costante positiva che rappresenta la conducibilità termica del materiale. Imponendo la conservazione dell'energia, ovvero che la velocità di variazione dell'energia in un volume arbitrario del corpo eguali la velocità con cui il calore fluisce nel volume stesso, otteniamo l'equazione del calore

$$\rho c \frac{\partial T}{\partial t} = k \Delta T, \quad (9.9)$$

dove ρ è la densità di massa del materiale e c è la capacità di calore specifico (per unità di massa). Se inoltre, a causa di altri fenomeni fisici, viene prodotto del calore a velocità $f(\mathbf{x}, t)$ all'interno del volume (ad esempio per effetto di riscaldamento elettrico), l'equazione (9.9) diventa

$$\rho c \frac{\partial T}{\partial t} = k \Delta T + f. \quad (9.10)$$

Il coefficiente $\mu = k/(\rho c)$ è detto *diffusività termica*. Per la soluzione di questo problema si veda l'Esempio 9.6. ■

Problema 9.3 (Elettrotecnica) Consideriamo un cavo del telegrafo di resistenza R e induttanza L per unità di lunghezza (si veda la Figura 9.1). Supponendo che la corrente possa dissiparsi al suolo attraverso

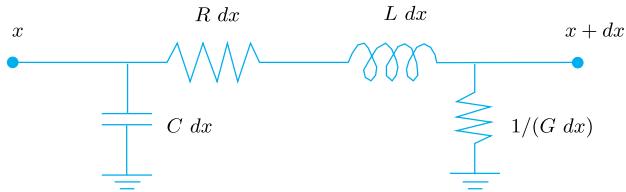


Figura 9.1. Un elemento del cavo di lunghezza dx

una capacità C ed una conduttanza G per unità di lunghezza (si veda la Figura 9.1), l'equazione per il potenziale v è

$$\frac{\partial^2 v}{\partial t^2} - c \frac{\partial^2 v}{\partial x^2} = -\alpha \frac{\partial v}{\partial t} - \beta v, \quad (9.11)$$

dove $c = 1/(LC)$, $\alpha = R/L + G/C$ e $\beta = RG/(LC)$. L'equazione (9.11) è un esempio di equazione iperbolica del secondo ordine ed è nota come *equazione del telegrafista* (oppure *equazione del telegrafo* (si veda [Str07]). La soluzione di questo problema è affrontata nell'Esempio 9.11. ■

9.2 Il problema di Poisson con condizioni di Dirichlet e di Neumann

In questa sezione consideriamo l'equazione di Poisson monodimensionale (9.1) e quella in due dimensioni (9.2). Come abbiamo già accennato all'inizio di questo Capitolo, per ottenere un'unica soluzione del problema ai limiti dobbiamo imporre opportune condizioni sul bordo del dominio computazionale.

Nel caso monodimensionale (9.1), una possibilità consiste nell'assegnare il valore di u in $x = a$ ed in $x = b$, ottenendo il sistema

$$\begin{aligned} -u''(x) &= f(x) \quad \text{per } x \in (a, b), \\ u(a) &= \alpha, \quad u(b) = \beta \end{aligned} \tag{9.12}$$

dove α e β sono due valori dati. Questo è detto *problema ai limiti di Dirichlet* e verrà studiato nella prossima sezione. Ricorrendo ad una doppia integrazione per parti è facile vedere che se $f \in C([a, b])$, 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 (9.12) non può essere posto nella forma di un problema di Cauchy in quanto il valore di u è assegnato in due punti differenti.

Un'alternativa alle condizioni di Dirichlet è rappresentata dalle condizioni di Neumann: date due costanti γ e δ che soddisfano la condizione

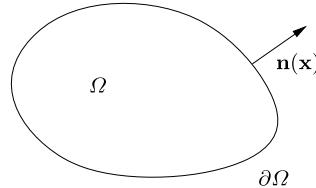


Figura 9.2. Un dominio bidimensionale Ω ed il versore normale al bordo con verso uscente

di compatibilità

$$\gamma - \delta = \int_a^b f(x)dx, \quad (9.13)$$

il problema ai limiti di Neumann è

$$\boxed{\begin{aligned} -u''(x) &= f(x) \quad \text{per } x \in (a, b), \\ u'(a) &= \gamma, \quad u'(b) = \delta \end{aligned}} \quad (9.14)$$

Si noti che la sua soluzione è definita a meno di una costante additiva.

Nel caso bidimensionale, il problema ai limiti di Dirichlet prende la seguente forma: assegnate due funzioni $f = f(\mathbf{x})$ e $g_D = g_D(\mathbf{x})$, si cerca la funzione $u = u(\mathbf{x})$ tale che

$$\boxed{\begin{aligned} -\Delta u(\mathbf{x}) &= f(\mathbf{x}) \quad \text{per } \mathbf{x} \in \Omega, \\ u(\mathbf{x}) &= g_D(\mathbf{x}) \quad \text{per } \mathbf{x} \in \partial\Omega \end{aligned}} \quad (9.15)$$

In alternativa alla condizione al bordo imposta in (9.15) si può assegnare una condizione sulla derivata parziale di u nella direzione normale a $\partial\Omega$ (si veda la Figura 9.2), ovvero

$$\frac{\partial u}{\partial \mathbf{n}}(\mathbf{x}) = \nabla u(\mathbf{x}) \cdot \mathbf{n}(\mathbf{x}) = g_N(\mathbf{x}) \quad \text{per } \mathbf{x} \in \partial\Omega$$

per un'opportuna funzione g_N che soddisfa la condizione di compatibilità

$$\int_{\partial\Omega} g_N d(\partial\Omega) = - \int_{\Omega} f d\Omega, \quad (9.16)$$

ottenendo un problema ai limiti di Neumann. (Si veda l'Esercizio 9.2.)

Se f e g_D sono due funzioni continue e la frontiera di Ω è sufficientemente regolare, allora esiste un'unica soluzione u del problema ai limiti di Dirichlet (9.15) (mentre la soluzione del problema ai limiti di Neumann è unica a meno di una costante additiva). Si veda, ad esempio, [Sal10].

I metodi numerici adatti a risolvere i problemi alle derivate parziali in 2 (o più) dimensioni si basano sugli stessi presupposti usati per risolvere problemi ai limiti monodimensionali. È per questa ragione che nelle Sezioni 9.3 e 9.5 ci occuperemo della risoluzione numerica del problema monodimensionale (9.12), rispettivamente con differenze finite ed elementi finiti, prima di affrontare il caso bidimensionale.

A tale scopo introduciamo sull'intervallo $[a, b]$ una decomposizione in sottointervalli $I_j = [x_j, x_{j+1}]$ per $j = 0, \dots, N$ con $x_0 = a$ e $x_{N+1} = b$. Supponiamo per semplicità che gli intervalli I_j abbiano tutti la stessa ampiezza $h = (b - a)/(N + 1)$.¹

9.3 Approssimazione alle differenze finite del problema di Poisson monodimensionale

L'equazione differenziale (9.12) 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

$$\delta^2 u(\bar{x}) = \frac{u(\bar{x} + h) - 2u(\bar{x}) + u(\bar{x} - h)}{h^2} \quad (9.17)$$

fornisce una approssimazione di $u''(\bar{x})$ di ordine 2 rispetto a h (si veda l'Esercizio 9.5). Questo suggerisce di usare la seguente approssimazione del problema (9.12): trovare $\{u_j\}_{j=1}^N$ tale che

$$-\frac{u_{j+1} - 2u_j + u_{j-1}}{h^2} = f(x_j), \quad j = 1, \dots, N \quad (9.18)$$

con $u_0 = \alpha$ e $u_{N+1} = \beta$. Naturalmente u_j sarà un'approssimazione di $u(x_j)$. Le equazioni (9.18) formano il sistema lineare

$$\mathbf{A}_{\text{fd}} \mathbf{u} = \mathbf{f}, \quad (9.19)$$

dove $\mathbf{u} = (u_1, \dots, u_N)^T$ è il vettore delle incognite, $\mathbf{f} = (f(x_1) + \alpha/h^2, f(x_2), \dots, f(x_{N-1}), f(x_N) + \beta/h^2)^T$ e

¹ Si osservi che ora h indica il passo di discretizzazione spaziale (e non più il passo di discretizzazione temporale come fatto nel capitolo precedente).

$$\mathbf{A}_{\text{fd}} = h^{-2} \widehat{\mathbf{A}}, \quad (9.20)$$

essendo $\widehat{\mathbf{A}}$ la matrice tridiagonale

$$\widehat{\mathbf{A}} = \text{tridiag}(-1, 2, -1) = \begin{bmatrix} 2 & -1 & 0 & \dots & 0 \\ -1 & 2 & \ddots & & \vdots \\ 0 & \ddots & \ddots & -1 & 0 \\ \vdots & & -1 & 2 & -1 \\ 0 & \dots & 0 & -1 & 2 \end{bmatrix}. \quad (9.21)$$

Tale sistema ammette un'unica soluzione in quanto \mathbf{A}_{fd} è simmetrica e definita positiva (si veda l'Esercizio 9.3) e potrà essere risolto utilizzando il metodo di Thomas presentato nella Sezione 5.6. Si tenga comunque conto che per h piccolo (e quindi per grandi valori di N) la matrice \mathbf{A}_{fd} è malcondizionata. Infatti $K(\mathbf{A}_{\text{fd}}) = \lambda_{\max}(\mathbf{A}_{\text{fd}})/\lambda_{\min}(\mathbf{A}_{\text{fd}}) = Ch^{-2}$, per un'opportuna costante C indipendente da h (si veda l'Esercizio 9.4). Di conseguenza, la risoluzione numerica del sistema (9.19) 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, come ad esempio quelli introdotti nella Sezione 5.10.1).

Nel Programma 9.1 viene risolto il problema ai limiti seguente, detto *di diffusione, trasporto e reazione*:

$$\begin{cases} -\mu u''(x) + \eta u'(x) + \sigma u(x) = f(x) & \text{per } x \in (a, b), \\ u(a) = \alpha, \quad u(b) = \beta, \end{cases} \quad (9.22)$$

essendo $\mu > 0$, $\sigma > 0$ e η costanti, che è una generalizzazione del problema di Poisson (9.12). Lo schema alle differenze finite utilizzato, che generalizza (9.18), è il seguente:

$$\begin{cases} -\mu \frac{u_{j+1} - 2u_j + u_{j-1}}{h^2} + \eta \frac{u_{j+1} - u_{j-1}}{2h} + \sigma u_j = f(x_j), & j = 1, \dots, N, \\ u_0 = \alpha, \quad u_{N+1} = \beta. \end{cases} \quad (9.23)$$

Si ottiene ancora un sistema del tipo di (9.19), ma ora la matrice ed il termine noto includono anche i contributi dovuti ai termini di trasporto ($\eta u'$) e di reazione (σu).

I parametri d'ingresso del Programma 9.1 sono gli estremi a e b dell'intervallo di definizione, il numero N di nodi interni all'intervallo, i coefficienti costanti μ , η e σ e il *function handle* `bvpfun` che dovrà precisare l'espressione della funzione $f(x)$. Infine, `ua` e `ub` sono i valori α e β che la soluzione deve assumere in $x = a$ ed in $x = b$, rispettivamente. In uscita,

vengono restituiti il vettore \mathbf{xh} dei nodi di discretizzazione e la soluzione calcolata \mathbf{uh} . Si noti che le soluzioni generate con questo programma possono essere affette da oscillazioni spurie se $h \geq 2\mu/\eta$.

Programma 9.1. bvp_fd_dir_1d: approssimazione di un problema ai limiti di diffusione, trasporto e reazione con il metodo delle differenze finite

```
function [xh,uh]=bvp_fd_dir_1d(a,b,N,mu,eta,sigma,...  
    bvpfun,ua,ub,varargin)  
%BVP_FD_DIR_1D Risolve un problema ai limiti  
% [XH,UH]=BVP_FD_DIR_1D(A,B,N,MU,ETA,SIGMA,BVPFUN,...  
% UA,UB) risolve con il metodo delle  
% differenze finite centrate in N nodi equispaziati  
% interni ad (A,B) il problema  
% -MU*D(DU/DX)/DX + ETA*DU/DX + SIGMA*U = BVPFUN  
% sull'intervallo (A,B) con condizioni al bordo  
% U(A)=UA e U(B)=UB. BVPFUN puo' essere un function  
% handle o una user defined function.  
% XH e UH contengono, rispettivamente, i nodi  
% e la soluzione numerica, inclusi i valori al bordo.  
h = (b-a)/(N+1);  
xh = (linspace(a,b,N+2))';  
hm = mu/h^2;  
hd = eta/(2*h);  
e = ones(N,1);  
Afd = spdiags([- (hm+hd)*e (2*hm+sigma)*e (-hm+hd)*e] ,...  
    -1:1, N, N);  
xi = xh(2:end-1);  
f = bvpfun(xi,varargin{:});  
f(1) = f(1)+ua*(hm+hd);  
f(end) = f(end)+ub*(hm-hd);  
uh = Afd\f;  
uh=[ua; uh; ub];
```

(Il problema (9.14) con condizioni di Neumann invece può essere risolto richiamando il Programma `bvp_fd_neu_1d.m` 10.6 illustrato nel Capitolo 10, si veda l'Esercizio 9.8.)

9.3.1 Analisi dell'approssimazione con differenze finite del problema di Poisson monodimensionale

Nella precedente Sezione abbiamo stabilito che la soluzione del problema approssimato esiste ed è unica. Siamo ora interessati a studiare l'errore di approssimazione. Sia $u(x)$ la soluzione del problema (9.12). Se

$$\max_{j=0,\dots,N+1} |u(x_j) - u_j| \leq C(h) \quad \text{quando } h \rightarrow 0 \quad (9.24)$$

essendo $C(h)$ un infinitesimo rispetto a h , ovvero $\lim_{h \rightarrow 0} C(h) = 0$, diremo che il metodo usato per calcolare la soluzione numerica rappresentata dai valori nodali u_j è convergente.

Vale il seguente risultato [QSSG14, IK66]: se $f \in C^2([a, b])$, allora

$$\max_{j=0, \dots, N+1} |u(x_j) - u_j| \leq \frac{(b-a)^2}{96} h^2 \max_{x \in [a, b]} |f''(x)| \quad (9.25)$$

cioè il metodo alle differenze finite (9.18) converge con ordine 2 rispetto a h .

Come nell'approssimazione delle equazioni differenziali ordinarie, anche per l'approssimazione di problemi ai limiti lineari (in cui cioè l'operatore differenziale è lineare) la convergenza di un metodo numerico è conseguenza della consistenza e della stabilità del metodo stesso. Vediamo come procedere nel caso del metodo (9.18).

Anzitutto definiamo l'*errore di troncamento locale* τ_h di un metodo numerico: esso è l'errore che si commette "forzando" la soluzione esatta del problema continuo a soddisfare lo schema numerico stesso (si osservi l'analogia con la definizione data nella Sezione 8.4 per le equazioni differenziali ordinarie). Ad esempio per il metodo (9.18), τ_h è così definito

$$\tau_h(x_j) = -\frac{u(x_{j+1}) - 2u(x_j) + u(x_{j-1})}{h^2} - f(x_j), \quad j = 1, \dots, N. \quad (9.26)$$

Quindi diciamo che un metodo è *consistente* se

$$\|\tau_h\| \rightarrow 0 \quad \text{quando } h \rightarrow 0 \quad (9.27)$$

e *consistente di ordine q* (con q un intero positivo) se $\|\tau_h\| = \mathcal{O}(h^q)$ per $h \rightarrow 0$, essendo $\|\cdot\|$ una opportuna norma discreta.

Nell'ipotesi che la derivata quarta di u esista e sia continua, (9.17) fornisce un'approssimazione di $u''(x_j)$ accurata al secondo ordine rispetto a h (si veda l'Esercizio 9.5). Quindi grazie a (9.12)₁, l'errore di troncamento locale di (9.18) è

$$\tau_h(x_j) = \frac{h^2}{12} u^{(4)}(\eta_j) + \mathcal{O}(h^4), \quad (9.28)$$

per un opportuno η_j in un intorno di x_j . Definendo la norma

$$\|\tau_h\| = \max_{j=1, \dots, N} |\tau_h(x_j)|, \quad (9.29)$$

possiamo concludere che lo schema (9.18) è consistente di ordine 2 rispetto a h . La norma introdotta in (9.29) è nota come *norma del massimo discreto*.

Il concetto di *stabilità* che invochiamo ora è simile al concetto di zero-stabilità che abbiamo dato nella Sezione 8.4, ovvero chiediamo che a piccole perturbazioni sui dati il metodo possa generare piccole perturbazioni sulla soluzione. Poiché nel problema in esame la soluzione

dipende linearmente dai dati (il nostro operatore differenziale è infatti lineare), chiedere che lo schema numerico soddisfi questo tipo di stabilità equivale a chiedere che la soluzione numerica dipenda con continuità dai dati, ovvero che esista $h_0 > 0$ ed una costante $C > 0$ tale che, per ogni $h < h_0$, si abbia

$$\max_{j=1,\dots,N} |u_j| \leq C \max_{j=1,\dots,N} |f(x_j)| + |\alpha| + |\beta|. \quad (9.30)$$

Si può dimostrare che lo schema (9.18) soddisfa questo tipo di stabilità (si veda, ad esempio, [QSSG14, Cap. 11]) con $C = (b - a)^2/8$.

Definiamo ora l'errore

$$e_j = u(x_j) - u_j, \quad j = 1, \dots, N \quad (9.31)$$

nei nodi e analizziamo le equazioni dell'errore

$$-\frac{e_{j+1} - 2e_j + e_{j-1}}{h^2} = \tau_h(x_j), \quad j = 1, \dots, N \quad (9.32)$$

ottenute sottraendo le equazioni (9.18) dalle corrispondenti (9.26). L'insieme delle equazioni (9.32) è esattamente dello stesso tipo di quelle in (9.18), quindi grazie a (9.30), a (9.28), a (9.12)₁ ed al fatto che $e_0 = e_{N+1} = 0$ (poiché abbiamo imposto le condizioni al bordo di Dirichlet (9.12)₂), possiamo concludere che

$$\max_{j=0,\dots,N+1} |e_j| \leq C \max_{j=1,\dots,N} |\tau_h(x_j)| \leq \frac{C}{12} h^2 \max_{x \in [a,b]} |f''(x)|, \quad (9.33)$$

ovvero la (9.25).

Esempio 9.1 Risolviamo il problema (9.12) sull'intervallo $(0, \pi)$ con $f(x) = 4\sin(2x) - e^{1-x}$, $\alpha = 0$ e $\beta = e^{1-2\pi}$, richiamando il Programma 9.1 per diversi valori di $N = 10, 20, \dots, 100$. Quindi misuriamo l'errore nei nodi della discretizzazione rispetto alla soluzione esatta $u(x) = \sin(2x) + e^{1-x}$. Con le seguenti istruzioni **MATLAB**:

```
a=0;b=2*pi;
mu=1; eta=0; sigma=0;
bvpfun=@(x)4*sin(2*x)-exp(1-x);
uex=@(x)sin(2*x)+exp(1-x);
ua=uex(a); ub=uex(b);
```

definiamo i dati di input per il Programma 9.1 quindi con le istruzioni:

```
H=[]; Err=[];
for N=10:10:100
h=(b-a)/(N+1); H=[H;h];
[xh,uh]=bvp_fd_dir_1d(a,b,N,mu,eta,sigma,bvpfun,ua,ub);
err=norm(uex(xh)-uh,inf)/norm(uex(xh),inf);
Err=[Err;err];
end
```

risolviamo il problema di Poisson e memorizziamo nel vettore `Err` gli errori commessi nella norma del massimo discreta e calcolati con il comando `norm(v,inf)` di `MAT&OCT`. Infine, con le istruzioni:

```
p=log(abs(Err(1:end-1)./Err(2:end)))./...
    (log(H(1:end-1)./H(2:end)));
p(1:2:end)',
```

calcoliamo numericamente l'ordine di convergenza del metodo sfruttando la formula (1.13). Otteniamo

```
p =
1.9911      2.0324      1.9959      1.9913      2.0099
```

ovvero ordine di convergenza 2, a conferma della stima (9.25).

In maniera analoga possiamo risolvere il problema (9.23) prendendo ad esempio $\mu = 1/10$, $\eta = 2$, $\sigma = 3$ e $f(x) = 4 \cos(2x) + 17/5 \sin(2x) + 9/10e^{1-x}$. Poiché per approssimare la derivata prima in (9.22) abbiamo utilizzato la differenza finita centrata (4.11) del secondo ordine rispetto a h , possiamo verificare anche in questo caso l'ordine di convergenza quadratico del metodo alle differenze finite centrali. ■

9.4 Approssimazione alle differenze finite di un problema di diffusione-trasporto a trasporto dominante

Consideriamo la seguente generalizzazione del problema ai limiti (9.12)

$$\begin{aligned} -\mu u''(x) + \eta u'(x) &= f(x) \quad \text{per } x \in (a, b), \\ u(a) &= \alpha, \quad u(b) = \beta \end{aligned} \tag{9.34}$$

essendo μ e η costanti positive. Esso è denominato *problema di diffusione-trasporto* in quanto i termini $-\mu u''(x)$ e $\eta u'(x)$ sono responsabili, rispettivamente, della diffusione e del trasporto della grandezza incognita $u(x)$. Il *numero di Péclet globale*, associato all'equazione (9.34) e definito come

$$\text{Pe}_{gl} = \frac{\eta(b-a)}{2\mu}, \tag{9.35}$$

rappresenta una misura di quanto il termine di trasporto domini quello diffusivo e diremo a trasporto dominante un problema in cui $\text{Pe}_{gl} \gg 1$.

Una possibile discretizzazione di (9.34) è:

$$\begin{cases} -\mu \frac{u_{j+1} - 2u_j + u_{j-1}}{h^2} + \eta \frac{u_{j+1} - u_{j-1}}{2h} = f(x_j), & j = 1, \dots, N, \\ u_0 = \alpha, \quad u_{N+1} = \beta, \end{cases} \tag{9.36}$$

in cui è stata utilizzata la differenza finita centrata (4.11) per approssimare il termine di trasporto.

In maniera analoga a quanto fatto per l'approssimazione dell'equazione di Poisson nella Sezione precedente, si può dimostrare che l'errore tra la soluzione del problema discreto (9.36) e quella del problema continuo (9.34) soddisfa la seguente stima

$$\max_{j=0,\dots,N+1} |u(x_j) - u_j| \leq Ch^2 \max_{x \in [a,b]} |f''(x)|. \quad (9.37)$$

La costante C , che è di fatto la costante di stabilità del metodo (9.36) (si veda (9.30)), è ora proporzionale a Pe_{gl} , pertanto essa è molto grande quando il trasporto è fortemente dominante. Questo comporta che, se non si utilizza un passo di discretizzazione h sufficientemente piccolo, la soluzione numerica che si ottiene con lo schema (9.36) può essere molto inaccurata ed in particolare può presentare forti oscillazioni che nulla hanno a che fare con la soluzione esatta del problema. Per una analisi più dettagliata del fenomeno si introduce il cosiddetto *numero di Pécelt locale* (o "di griglia")

$$\text{Pe} = \frac{\eta h}{2\mu}. \quad (9.38)$$

Si può dimostrare che la soluzione del problema discreto (9.36) è priva di oscillazioni quando $\text{Pe} < 1$ (si veda a tale proposito [Qua16, Cap. 12]). Questo comporta che in presenza di problemi con trasporto fortemente dominante, al fine di garantire la bontà della soluzione numerica, è sufficiente scegliere un passo di discretizzazione $h < 2\mu/\eta$. Ciò tuttavia è assai dispendioso quando il rapporto $2\mu/\eta$ è molto piccolo. Una possibile alternativa consiste nell'utilizzare una diversa approssimazione del termine u' ; in particolare, invece della differenza finita centrata (4.11), sarà sufficiente considerare la differenza finita all'indietro (4.10), cosicché il sistema (9.36) viene sostituito da:

$$\begin{cases} -\mu \frac{u_{j+1} - 2u_j + u_{j-1}}{h^2} + \eta \frac{u_j - u_{j-1}}{h} = f(x_j), & j = 1, \dots, N, \\ u_0 = \alpha, \quad u_{N+1} = \beta, \end{cases} \quad (9.39)$$

noto come schema *upwind*.

Approssimando (9.34) con (9.39), la soluzione numerica ottenuta non è più affetta da oscillazioni, come si può osservare in Figura 9.3. Osserviamo tuttavia che ora l'ordine di convergenza del metodo è solo pari a uno in quanto la derivata prima è stata approssimata con un rapporto incrementale accurato al primo ordine rispetto a h .

La scelta del rapporto incrementale all'indietro è legata al fatto che $\eta > 0$. Nel caso in cui fosse $\eta < 0$, il rapporto incrementale corretto da utilizzare sarebbe quello in avanti. Per una spiegazione di carattere fisico si veda ad esempio [Qua16, Cap 12].

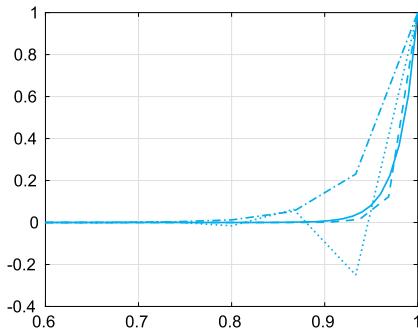


Figura 9.3. Soluzione esatta (*linea continua*), approssimazione con differenze finite centrate con $h = 1/15$ ($\text{Pe} > 1$) (*linea punteggiata*), approssimazione con differenze finite centrate con $h = 1/32$ ($\text{Pe} < 1$) (*linea tratteggiata*), approssimazione con differenze finite *upwind* $h = 1/15$ (*linea tratto-punto*) della soluzione del problema (9.34) con $a = 0$, $b = 1$, $\alpha = 0$, $\beta = 1$, $f(x) = 0$, $\mu = 1/50$ e $\eta = 1$. Per maggior chiarezza le soluzioni numeriche sono state disegnate nell’intervallo $[0.6, 1]$ anziché in $[0, 1]$

Il problema (9.39) è risolto nel Programma `bvp_fd_upwind_1d.m` 10.7, si veda l’Esercizio 9.9.

9.5 Approssimazione agli elementi finiti del problema di Poisson monodimensionale

Il *metodo degli elementi finiti* rappresenta un’alternativa al metodo delle differenze finite appena introdotto per l’approssimazione di problemi ai limiti. Esso viene derivato da un’opportuna riformulazione del problema (9.12).

Consideriamo l’equazione (9.12) e moltiplichiamo ambo i membri per una generica funzione $v \in C^1([a, b])$. Integrando la corrispondente uguaglianza sull’intervallo (a, b) ed utilizzando 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 (9.12) diventa: trovare $u \in C^1([a, b])$ tale che $u(a) = \alpha$, $u(b) = \beta$ e

$$\int_a^b u'(x)v'(x)dx = \int_a^b f(x)v(x)dx \quad (9.40)$$

per ogni $v \in C^1([a, b])$ tale che $v(a) = v(b) = 0$. La (9.40) viene chiamata *formulazione debole* del problema (9.12) (di fatto, sia la funzione u , sia

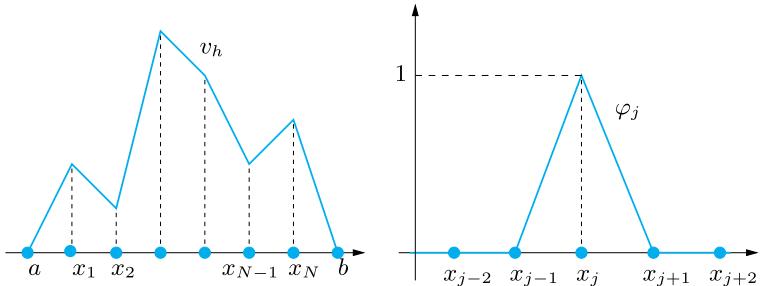


Figura 9.4. A sinistra, una generica funzione $v_h \in V_h^0$. A destra, la funzione di base per V_h^0 associata al nodo j -esimo

le funzioni test v possono essere meno regolari di $C^1([a, b])$, si veda, ad esempio [QSSG14] o [Qua16]).

Utilizzando la suddivisione dell'intervallo $[a, b]$ introdotta alla fine della Sezione 9.2, l'approssimazione ad elementi finiti di (9.40) è definita come segue:

trovare $u_h \in V_h$ tale che $u_h(a) = \alpha, u_h(b) = \beta$ e

$$\sum_{j=0}^N \int_{x_j}^{x_{j+1}} u'_h(x)v'_h(x)dx = \int_a^b f(x)v_h(x)dx \quad \forall v_h \in V_h^0 \quad (9.41)$$

dove

$$V_h = \{v_h \in C^0([a, b]) : v_{h|I_j} \in \mathbb{P}_1, j = 0, \dots, N\}, \quad (9.42)$$

cioè V_h è lo spazio delle funzioni continue in $[a, b]$ le cui restrizioni ad ogni sotto intervallo I_j sono polinomi di grado uno; $V_h^0 \subset V_h$ è il sottospazio delle funzioni di V_h che si annullano agli estremi a e b .

V_h è detto lo spazio degli elementi finiti di grado 1. Se u soddisfa condizioni di Dirichlet omogenee in $x = a$ e $x = b$ allora $u_h \in V_h^0$. In quest'ultimo caso il problema (9.41) verrebbe sostituito da

trovare $u_h \in V_h^0$ tale che

$$\sum_{j=0}^N \int_{x_j}^{x_{j+1}} u'_h(x)v'_h(x)dx = \int_a^b f(x)v_h(x)dx \quad \forall v_h \in V_h^0 \quad (9.43)$$

ed è noto come *approssimazione di Galerkin* di (9.40) con $\alpha = \beta = 0$.

Le funzioni di V_h^0 sono polinomi composti di grado 1 che si annullano agli estremi dell'intervallo $[a, b]$ (si veda la Figura 9.4 a sinistra). Di conseguenza, ogni funzione v_h di V_h^0 ammette la seguente rappresentazione

$$v_h(x) = \sum_{j=1}^N v_j \varphi_j(x),$$

dove, per $j = 1, \dots, N$, $v_j = v_h(x_j)$ e

$$\varphi_j(x) = \begin{cases} \frac{x-x_{j-1}}{x_j-x_{j-1}} & \text{se } x \in I_{j-1}, \\ \frac{x-x_{j+1}}{x_j-x_{j+1}} & \text{se } x \in I_j, \\ 0 & \text{altrimenti.} \end{cases} \quad (9.44)$$

La generica φ_j è dunque nulla in tutti i nodi x_i fuorché in x_j dove $\varphi_j(x_j) = 1$ (si veda la Figura 9.4 a destra). Le funzioni φ_j , $j = 1, \dots, N$ sono dette *funzioni di forma* (o *funzioni di base*) e costituiscono una base per lo spazio V_h^0 .

Di conseguenza, verificare la (9.41) per ogni funzione di V_h^0 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 (9.41) otteniamo

$$\int_{I_{j-1} \cup I_j} u'_h(x) \varphi'_j(x) dx = \int_{I_{j-1} \cup I_j} f(x) \varphi_j(x) dx, \quad j = 1, \dots, N. \quad (9.45)$$

D'altra parte, ponendo $u_j = u_h(x_j)$, possiamo scrivere

$$u_h(x) = \sum_{j=0}^{N+1} u_j \varphi_j(x) = \sum_{j=1}^N u_j \varphi_j(x) + \alpha \varphi_0(x) + \beta \varphi_{N+1}(x), \quad (9.46)$$

dove $\varphi_0(x) = (x_1 - x)/(x_1 - a)$ per $a \leq x \leq x_1$ e $\varphi_0(x) = 0$ altrove, $\varphi_{N+1}(x) = (x - x_N)/(b - x_N)$ per $x_N \leq x \leq b$ e $\varphi_{N+1}(x) = 0$ altrove.

Sostituendo (9.46) in (9.45), troviamo:

$$\begin{aligned} & u_1 \int_{I_0 \cup I_1} \varphi'_1(x) \varphi'_1(x) dx + u_2 \int_{I_1} \varphi'_2(x) \varphi'_1(x) dx \\ &= \int_{I_0 \cup I_1} f(x) \varphi_1(x) dx - \alpha \int_{I_0} \varphi'_0(x) \varphi'_1(x) dx \\ & u_{j-1} \int_{I_{j-1}} \varphi'_{j-1}(x) \varphi'_j(x) dx + u_j \int_{I_{j-1} \cup I_j} \varphi'_j(x) \varphi'_j(x) dx \\ & \quad + u_{j+1} \int_{I_j} \varphi'_{j+1}(x) \varphi'_j(x) dx \\ &= \int_{I_{j-1} \cup I_j} f(x) \varphi_j(x) dx, \quad j = 2, \dots, N-1, \\ & u_{N-1} \int_{I_{N-1}} \varphi'_{N-1}(x) \varphi'_N(x) dx + u_N \int_{I_{N-1} \cup I_N} \varphi'_N(x) \varphi'_N(x) dx \\ &= \int_{I_{N-1} \cup I_N} f(x) \varphi_j(x) dx - \beta \int_{I_N} \varphi'_{N+1}(x) \varphi'_N(x) dx. \end{aligned}$$

Nel caso particolare in cui tutti gli intervalli abbiano la stessa ampiezza h , abbiamo $\varphi'_{j-1}(x) = -1/h$ in I_{j-1} , $\varphi'_j(x) = 1/h$ in I_{j-1} e $\varphi'_j(x) = -1/h$ in I_j , $\varphi'_{j+1}(x) = 1/h$ in I_j . Di conseguenza, otteniamo

$$\begin{aligned} \frac{2u_1 - u_2}{h} &= \int_{I_0 \cup I_1} f(x)\varphi_1(x)dx + \frac{\alpha}{h}, \\ \frac{-u_{j-1} + 2u_j - u_{j+1}}{h} &= \int_{I_{j-1} \cup I_j} f(x)\varphi_j(x)dx, \quad j = 2, \dots, N-1, \\ \frac{-u_{N-1} + 2u_N}{h} &= \int_{I_{N-1} \cup I_N} f(x)\varphi_N(x)dx + \frac{\beta}{h}. \end{aligned} \quad (9.47)$$

Si trova dunque il sistema lineare

$$A_{\text{fe}}\mathbf{u} = \mathbf{f}_{\text{fe}} \quad (9.48)$$

nell'incognita $\mathbf{u} = (u_1, \dots, u_N)^T$. Osserviamo che abbiamo eliminato le incognite u_0 e u_{N+1} dal sistema, sostituendole con i valori noti α e β . Il vantaggio di questo artificio consiste nel fatto che la matrice A_{fe} associata a (9.47) risulta simmetrica e definita positiva e per la risoluzione del sistema (9.48) possiamo utilizzare metodi *ad hoc*.

La matrice A_{fe} differisce da quella ottenuta nel caso delle differenze finite per un fattore h (ovvero $A_{\text{fe}} = hA_{\text{fd}}$) e il termine noto \mathbf{f}_{fe} ha per componenti i termini noti del sistema (9.47). La soluzione \mathbf{u} , a dispetto delle notazioni coincidenti, differisce naturalmente da quella ottenuta con le differenze finite.

9.5.1 Cenni all'analisi del metodo agli elementi finiti

Elementi finiti di grado 1 e differenze finite condividono invece la stessa accuratezza rispetto a h quando si calcoli l'errore massimo nei nodi.

Osserviamo a tale proposito che l'accuratezza quadratica rispetto a h è garantita qualora $f \in C^2([a, b])$ nel caso di approssimazione con differenze finite (si veda la (9.37)), mentre nel caso di approssimazione con elementi finiti per garantire convergenza quadratica nella norma del massimo discreta è sufficiente che f sia continua.

Infine, in numerose applicazioni succede che f sia limitata ma discontinua, ad esempio quando f è una intensità di forza costante a tratti. In tal caso, la soluzione ottenuta con elementi finiti converge quadraticamente nella norma integrale, ovvero l'errore $\|u - u_h\| = (\int_a^b (u - u_h)^2(x)dx)^{1/2}$ tende a zero come $\mathcal{O}(h^2)$ (si veda, ad esempio, [Qua16]).²

² L'analisi del metodo degli elementi finiti richiede strumenti analitici più raffinati che non intendiamo proporre in questo testo. Il lettore interessato può consultare, ad esempio, [Qua16, BS08, QV94].

Osserviamo tuttavia che al fine di garantire che la discretizzazione ad elementi finiti converga con ordine 2, gli integrali che compaiono nei termini di destra di (9.47) devono essere approssimati con formule di quadratura sufficientemente accurate.

Nel caso in cui $f \in C^2([a, b])$ è sufficiente utilizzare le formule composite del punto medio (4.20) o dei trapezi (4.24), sulla suddivisione in intervalli I_j stabilita dalla discretizzazione (si veda, ad esempio, l'Esercizio 9.10).

Nel caso in cui la funzione f sia poco regolare, ad esempio se è limitata, ma discontinua e di classe C^2 solo a tratti, per garantire l'accuratezza ottimale della formula di quadratura è necessario far coincidere i punti di discontinuità con opportuni nodi x_j della griglia di discretizzazione. Infatti, solo con questa scelta dei nodi la restrizione di f ad ogni intervallino I_j ha la regolarità necessaria per poter garantire che l'errore sia $\mathcal{O}(h^3)$ (per h che tende a zero) su ogni intervallino (si veda (4.23)). In caso contrario, pur continuando ad essere infinitesimo per $h \rightarrow 0$, l'errore di quadratura non sarebbe più un $\mathcal{O}(h^3)$ e questo comprometterebbe l'accuratezza di ordine 2 del metodo degli elementi finiti (si veda l'Esempio 9.2).

Osserviamo anche che, quando la funzione f è discontinua è meglio utilizzare formule di quadratura di tipo aperto (come ad esempio la formula di punto medio) anziché quelle di tipo chiuso (come la formula dei trapezi), in quanto queste ultime richiedono di valutare la funzione agli estremi degli intervallini I_j , dove la funzione potrebbe essere discontinua (si veda [QSSG14, Cap. 8.6]).

Per calcolare gli integrali che compaiono in (9.47) applichiamo la formula del punto medio composita rispetto alla suddivisione dell'intervallo $[a, b]$ indotta dai nodi x_j della discretizzazione, per $j = 1, \dots, N$. Abbiamo

$$\begin{aligned} (\mathbf{f}_{\text{fe}})_j &= \int_{I_{j-1} \cup I_j} f(x) \varphi_j(x) dx \\ &= h f\left(\frac{x_{j-1} + x_j}{2}\right) \varphi_j\left(\frac{x_{j-1} + x_j}{2}\right) \\ &\quad + h f\left(\frac{x_j + x_{j+1}}{2}\right) \varphi_j\left(\frac{x_j + x_{j+1}}{2}\right) \\ &= \frac{h}{2} \left[f\left(\frac{x_{j-1} + x_j}{2}\right) + f\left(\frac{x_j + x_{j+1}}{2}\right) \right]. \end{aligned} \quad (9.49)$$

Nel caso in cui $f \in C^2([a, b])$, si può sostituire con l'interpolatore composito lineare di Lagrange $\Pi_1^h f(x)$ (si veda la Sezione 3.4) che intercala f nei nodi x_j della discretizzazione, indi integrare in maniera esatta le funzioni $\Pi_1^h f(x) \varphi_j(x)$. Si veda a tale proposito l'Esercizio 9.11.

Un'ulteriore generalizzazione del metodo degli elementi finiti di grado 1 consiste nell'usare polinomi composti di grado maggiore di uno

tipicamente 2, più raramente 3. In tal caso si incrementa l'ordine di accuratezza dello schema.

Esempio 9.2 Risolviamo il problema di Poisson (9.12) richiamando il Programma 9.2 con i seguenti dati: $(a, b) = (-1, 1)$, $f(x) = -x^2$ per $x \in [-0.5, 0.5]$ e $f(x) = 0$ altrove (f è pertanto limitata e discontinua) e condizioni al bordo di Dirichlet $u(-1) = u(1) = 5/192$. Dopo aver calcolato la soluzione numerica u_h ed aver osservato che la soluzione esatta è

$$u(x) = \begin{cases} -x/24 - 1/64 & \text{se } x < -1/2, \\ x^4/12 & \text{se } -1/2 \leq x \leq 1/2 \\ x/24 - 1/64 & \text{se } x > 1/2 \end{cases}$$

valutiamo l'errore nella norma integrale

$$\|u - u_h\| = \left(\int_a^b (u(x) - u_h(x))^2 dx \right)^{1/2}. \quad (9.50)$$

A tale scopo utilizziamo la versione composita della formula di quadratura di Gauss-Legendre con $n = 4$ descritta nella Sezione 4.4. Tale formula ha grado di esattezza pari a 9 e quindi in questo caso integra esattamente la funzione errore $(u - u_h)^2$. Con le istruzioni **MAT&OCT**:

```
a=-1; b=1; mu=1; eta=0; sigma=0;
bvpfun=@(x)0+0*x-x.^2.*((x>=-1/2 & x<=1/2));
uex=@(x)(-x/24-1/64).*((x<-1/2)+...
           x.^4/12.*((x>=-1/2 & x<=1/2)+...
           (x/24-1/64).*((x>1/2));
ua=uex(a); ub=uex(b);
Err=[]; H=[];
for N=[199,399,799,999];
h=(b-a)/(N+1);H=[H;h];
[xh,uh]=bvp_fe_dir_1d(a,b,N,mu,eta,sigma,bvpfun,ua,ub);
err=norma_g14c(uex,xh,uh);
Err=[Err;err];
end
p=log(abs(Err(1:end-1)./Err(2:end)))./...
      (log(H(1:end-1)./H(2:end)))
```

otteniamo

```
p =
    2.0000    2.0000    2.0000
```

ovvero convergenza di ordine 2 rispetto a h nella norma integrale.

La function **norma_g14c.m** implementa il calcolo della norma integrale dell'errore; rimandiamo all'Esercizio 9.12 per la sua realizzazione.

Per via della discontinuità di f , il metodo degli elementi finiti converge quadraticamente a patto che fra i nodi di discretizzazione x_j vi siano i punti di discontinuità $-1/2$ e $1/2$. Se invece di prendere $N \in \{199, 399, 799, 999\}$, scegliersimo $N \in \{200, 400, 800, 1000\}$, i punti $-1/2$ e $+1/2$ cadrebbero all'interno di due intervallini I_j in cui f non sarebbe più di classe C^2 . Di conseguenza, otterremmo ordine di convergenza in h solo pari a 1. In Figura 9.5 sono mostrati gli errori (9.50) al variare di $h = 2/(N + 1)$ con $N \in \{199, 399, 799, 999\}$ (in linea continua) e con $N \in \{200, 400, 800, 1000\}$ (in linea tratteggiata): a conferma di quanto affermato prima, nel primo caso abbiamo ordine di convergenza pari a 2 rispetto a h , mentre nel secondo caso ordine di convergenza solo pari a 1. ■

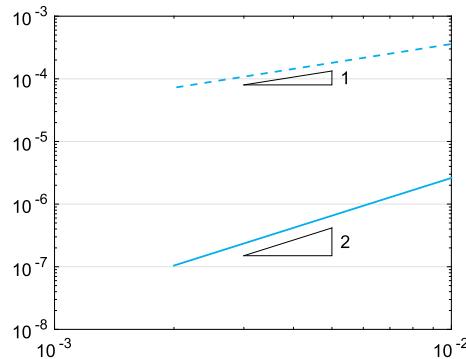


Figura 9.5. Gli errori (9.50) al variare di $h = 2/(N + 1)$ con $N \in \{199, 399, 799, 999\}$ (linea continua) e con $N \in \{200, 400, 800, 1000\}$ (linea tratteggiata)

9.6 Approssimazione agli elementi finiti di un problema di diffusione-trasporto a trasporto dominante

Il metodo degli elementi finiti può ovviamente essere generalizzato a problemi del tipo (9.22) (anche nel caso in cui μ , η e σ dipendano da x) e (9.34).

Qualora si voglia approssimare il problema di diffusione-trasporto a trasporto dominante (9.34), la procedura di tipo *upwind* utilizzata per le differenze finite può essere riprodotta anche in elementi finiti. Infatti, svolgendo conti analoghi a quelli mostrati per discretizzare il termine di diffusione, otteniamo

$$\int_a^b \eta u'_h(x) \varphi_j(x) dx = \eta \frac{u_{j+1} - u_{j-1}}{2}, \quad j = 1, \dots, N.$$

A questo punto, osservando che

$$u_j - u_{j-1} = \frac{u_{j+1} - u_{j-1}}{2} - \frac{h}{2} \frac{u_{j+1} - 2u_j + u_{j-1}}{h},$$

concludiamo che utilizzare il rapporto incrementale *upwind* invece di quello centrato $(u_{i+1} - u_{i-1})/2h$ equivale a perturbare quest'ultimo con un termine corrispondente alla discretizzazione di una derivata seconda con coefficiente $\eta h/2$. È naturale intrepretare questo termine addizionale come un termine di diffusione con viscosità artificiale $\eta h/2$ che si somma al termine diffusivo con viscosità μ nel problema fisico originario. In altre parole, eseguire l'*upwind* in elementi finiti equivale a risolvere con

il metodo di Galerkin (centrato) il seguente problema perturbato

$$-\mu_h u''(x) + \eta u'(x) = f(x), \quad (9.51)$$

dove $\mu_h = (1 + \text{Pe})\mu$. Il nuovo termine $\text{Pe}\mu = \eta h/2$ gioca il ruolo di viscosità artificiale.

Il Programma 9.2 risolve il problema di diffusione trasporto reazione (9.22) con il metodo degli elementi finiti e trattamento *upwind* del termine di trasporto. I parametri in input ed output assumono lo stesso significato di quelli del Programma 9.1. Gli integrali che coinvolgono la funzione f sono calcolati con la formula del punto medio, come descritto in (9.49).

Programma 9.2. bvp_fe_dir_1d: approssimazione di un problema ai limiti di diffusione, trasporto e reazione con il metodo degli elementi finiti di grado 1

```
function [xh,uh]=bvp_fe_dir_1d(a,b,N,mu,eta,sigma,...  
    bvpfun,ua,ub,varargin)  
%BVP_FE_DIR_1D Risolve un problema ai limiti  
% [XH,UH]=BVP_FE_DIR_1D(A,B,N,MU,ETA,SIGMA,BVPFUN,...  
%     UA,UB) con il metodo degli elementi  
% finiti di grado 1 con passo h=1/N il problema  
% -MU*D(DU/DX)/DX + ETA*DU/DX + SIGMA*U = BVPFUN  
% sull'intervallo (A,B) con condizioni al bordo  
% U(A)=UA e U(B)=UB. Il termine di trasporto e'  
% trattato con schema upwind.  
% BVPFUN puo' essere un function handle o una  
% user defined function.  
% In output, XH e UH contengono, risp., i nodi e la  
% soluzione numerica, inclusi i valori al bordo.  
h = (b-a)/(N+1); xh = (linspace(a,b,N+2))';  
Pe=eta*h/(2*mu); hm = mu*(1+Pe)/h; hd = eta/2;  
hs=sigma*h/6; e =ones(N,1);  
% stiffness matrix  
Afe = spdiags([[-hm-hd+hs]*e (2*hm+4*hs)*e...  
                (-hm+hd+hs)*e], -1:1, N, N);  
f=quad_mp(bvpfun,xh);  
f(1) = f(1)+mu*ua/h;  
f(end) = f(end)+mu*ub/h;  
uh = Afe\f;  
uh=[ua; uh; ub];  
return  
function [f]=quad_mp(bvpfun,xh);  
N=length(xh)-2; h=xh(2)-xh(1);  
f=zeros(N,1);  
for j=1:N  
    f(j)=h/2*(bvpfun((xh(j)+xh(j+1))/2)+...  
              bvpfun((xh(j+1)+xh(j+2))/2));  
end  
return
```



Si vedano gli Esercizi 9.1–9.12.

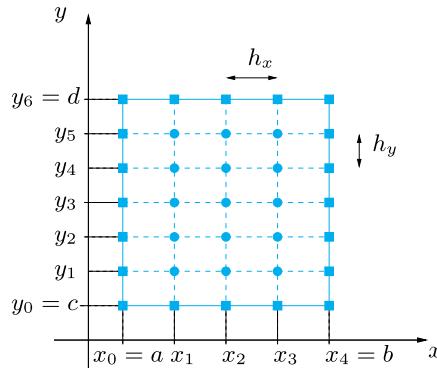


Figura 9.6. Griglia di calcolo Δ_h di 15 nodi interni su un dominio rettangolare

9.7 Approssimazione alle differenze finite del problema di Poisson in 2 dimensioni

Consideriamo il problema di Poisson (9.15) in una regione Ω del piano.

L'idea alla base delle differenze finite consiste nell'approssimare le derivate parziali che compaiono nell'equazione (9.15) con rapporti incrementali su una opportuna griglia (detta griglia computazionale) costituita da un insieme finito di nodi. In questo modo la soluzione u dell'equazione differenziale verrà approssimata solo in questi nodi.

Il primo passo consiste quindi nel 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_i, x_{i+1}) per $i = 0, \dots, N_x$, 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 $h_x = \max_{i=0, \dots, N_x} (x_{i+1} - x_i)$ la loro massima lunghezza.

In modo del tutto analogo introduciamo una discretizzazione lungo l'asse y : $\Delta_y = \{y_0, \dots, y_{N_y+1}\}$, con $y_0 = c$ e $y_{N_y+1} = d$ e $h_y = \max_{j=0, \dots, N_y} (y_{j+1} - y_j)$. Il prodotto cartesiano $\Delta_h = \Delta_x \times \Delta_y$ definisce la griglia computazionale su Ω (come mostrato in Figura 9.6), dove $h = \max\{h_x, h_y\}$ è una misura caratteristica della finezza della griglia. 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 incrementali

$$\begin{aligned}\delta_x^2 u_{i,j} &= \frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{h_x^2}, \\ \delta_y^2 u_{i,j} &= \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{h_y^2}.\end{aligned}\quad (9.52)$$

Qualora u sia sufficientemente regolare, 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) (la dimostrazione ricalca quella del caso monodimensionale svolta nell'Esercizio 9.5). Se sostituiamo le derivate parziali seconde di u con le formule (9.52), richiedendo che l'equazione alle derivate parziali venga soddisfatta in tutti i nodi interni di Δ_h , perveniamo alle seguenti equazioni

$$-(\delta_x^2 u_{i,j} + \delta_y^2 u_{i,j}) = f(x_i, y_j), \quad i = 1, \dots, N_x, \quad j = 1, \dots, N_y. \quad (9.53)$$

Ad esse vanno aggiunte le equazioni che impongono il dato di Dirichlet sul bordo $\partial\Omega$

$$u_{i,j} = g_D(x_i, y_j) \quad \forall i, j \text{ tale che } (x_i, y_j) \in \partial\Delta_h, \quad (9.54)$$

dove $\partial\Delta_h$ denota l'insieme dei punti di Δ_h che appartengono al bordo del rettangolo. Tali punti sono indicati in Figura 9.6 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 (9.53) otteniamo

$$\begin{aligned}-\frac{1}{h^2}(u_{i-1,j} + u_{i,j-1} - 4u_{i,j} + u_{i,j+1} + u_{i+1,j}) &= f(x_i, y_j), \\ i &= 1, \dots, N_x, \quad j = 1, \dots, N_y\end{aligned}\quad (9.55)$$

Il sistema formato dalle equazioni (9.53) (o (9.55)) e (9.54) 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 (9.55) coinvolge 5 valori nodali, come mostrato in Figura 9.7. Per questo motivo questo schema è noto come *schema a 5 punti* per l'operatore di Laplace.

Le incognite associate ai nodi di bordo, possono essere eliminate usando (9.54) e quindi (9.53) (o (9.55)) 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 (9.19). Tuttavia stavolta la matrice $A_{fd} \in \mathbb{R}^{N \times N}$ ha la seguente forma (tridiagonale a blocchi)

$$A_{fd} = \text{tridiag}(D, T, D). \quad (9.56)$$

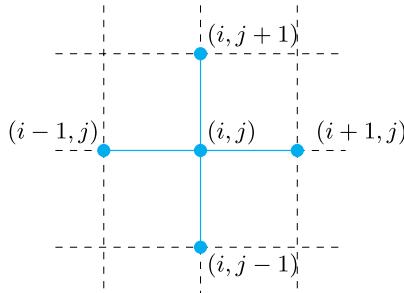


Figura 9.7. Supporto dello schema a 5 punti per l'operatore di Laplace

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 tridiagonale simmetrica

$$T = \text{tridiag}\left(-\frac{1}{h_x^2}, \frac{2}{h_x^2} + \frac{2}{h_y^2}, -\frac{1}{h_x^2}\right).$$

La matrice A_{fd} è simmetrica in quanto tutti i blocchi diagonali sono simmetrici. Verifichiamo che è anche definita positiva dimostrando che $\mathbf{v}^T A_{fd} \mathbf{v} > 0$ per ogni $\mathbf{v} \in \mathbb{R}^N$, $\mathbf{v} \neq \mathbf{0}$. Partizionando \mathbf{v} in N_y vettori \mathbf{v}_k di lunghezza N_x , otteniamo

$$\mathbf{v}^T A_{fd} \mathbf{v} = \sum_{k=1}^{N_y} \mathbf{v}_k^T T \mathbf{v}_k - \frac{2}{h_y^2} \sum_{k=1}^{N_y-1} \mathbf{v}_k^T \mathbf{v}_{k+1}. \quad (9.57)$$

Possiamo scrivere $T = 2/h_y^2 I + 1/h_x^2 \hat{A}$, dove \hat{A} è la matrice (simmetrica e definita positiva) definita in (9.21) e I è la matrice identità. Di conseguenza, sfruttando l'identità $2a(a-b) = a^2 - b^2 + (a-b)^2$ e operando alcuni passaggi algebrici, (9.57) diventa

$$\begin{aligned} \mathbf{v}^T A_{fd} \mathbf{v} &= \frac{1}{h_x^2} \sum_{k=1}^{N_y-1} \mathbf{v}_k^T \hat{A} \mathbf{v}_k \\ &\quad + \frac{1}{h_y^2} \left(\mathbf{v}_1^T \mathbf{v}_1 + \mathbf{v}_{N_y}^T \mathbf{v}_{N_y} + \sum_{k=1}^{N_y-1} (\mathbf{v}_k - \mathbf{v}_{k+1})^T (\mathbf{v}_k - \mathbf{v}_{k+1}) \right) \\ &\geq \frac{1}{h_x^2} \sum_{k=1}^{N_y-1} \mathbf{v}_k^T \hat{A} \mathbf{v}_k \end{aligned}$$

che è un numero reale strettamente positivo in quanto \hat{A} è definita positiva ed almeno uno dei vettori \mathbf{v}_k è non nullo.

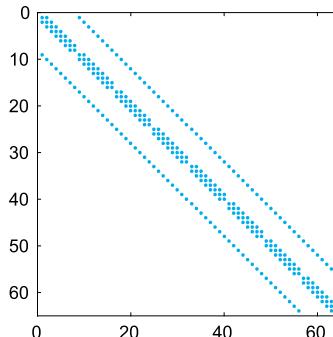


Figura 9.8. Pattern della matrice associata allo schema a 5 punti usando un ordinamento lessicografico delle incognite

Se A_{fd} è simmetrica e definita positiva, allora è non singolare e il sistema ammette un'unica soluzione \mathbf{u}_h .

Osserviamo che A_{fd} è una matrice *sparsa* e come tale verrà memorizzata in **MAT&OCT** nel formato **sparse** (si veda la Sezione 5.3). In Figura 9.8 riportiamo il *pattern* della matrice A_{fd} (ottenuto con il comando `spy(A)`) per una griglia uniforme di 10×10 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_{fd} 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_{fd} è mal condizionata per $h \rightarrow 0$ in quanto il suo numero di condizionamento cresce come h^{-2} al decrescere di h (rimandiamo ad esempio a [LeV02] per una dimostrazione).

Nel Programma 9.3 costruiamo e risolviamo (con il metodo richiamato dal comando `\`, si veda la Sezione 5.8) il sistema (9.53)–(9.54). I parametri d'ingresso `a`, `b`, `c` e `d` servono per precisare gli estremi degli intervalli che generano il dominio rettangolare $\Omega = (a, b) \times (c, d)$, mentre `nx` e `ny` devono contenere i valori N_x e N_y (si può avere $N_x \neq N_y$). Infine, `fun` e `gd` sono *function handle* che precisano la funzione $f = f(x, y)$ (detto anche termine sorgente) ed il dato di Dirichlet $g_D = g_D(x, y)$. La variabile `output uh` è una matrice tale che $uh(j, i) = u_{i,j}$, mentre `xh` e `yh` sono vettori le cui componenti sono i nodi x_i e y_j , rispettivamente, inclusi i punti di bordo. La soluzione numerica può essere visualizzata con il comando `mesh(xh, yh, uh)`. La variabile (opzionale) di input `uex` è un *function handle* che precisa la soluzione esatta $u(x, y)$ 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`)

$$\text{error} = \max_{i,j} |u(x_i, y_j) - u_{i,j}| / \max_{i,j} |u(x_i, y_j)|. \quad (9.58)$$

Programma 9.3. `poisson_fd_2d`: approssimazione del problema di Poisson con condizioni di Dirichlet usando il metodo delle differenze finite a 5 punti

```

function [xh,yh,uh,error]=poisson_fd_2d(a,b,c,d,%
                                         nx,ny,fun,gd,uem,uey,varargin)
%POISSON_FD_2D approssimazione del problema di Poisson
% in due dimensioni
% [XH,YH,UH]=POISSON_FD_2D(A,B,C,D,NX,NY,FUN,GD)
% risolve con lo schema alle differenze finite
% a 5 punti il problema -LAPL(U) = FUN in un
% rettangolo (A,B)X(C,D) con condizioni al bordo
% di Dirichlet U(X,Y)=GD(X,Y) per ogni (X,Y)
% sul bordo del rettangolo.
% [XH,YH,UH,ERROR]=POISSONFD(A,B,C,D,NX,NY,FUN,%
% GD,UEX) calcola anche l'errore sulla soluzione
% esatta UEX.
% FUN, GD e UEX possono function handle o user
% defined functions.
% [XH,YH,UH,ERROR]=POISSON_FD_2D(A,B,C,D,NX,NY,FUN,%
% GD,UEX,P1,P2, ...) passa i parametri opzionali
% P1,P2,... alle funzioni FUN, GD, UEX.
if nargin == 8
    uex = @(x,y) 0*x+0*y;
end
nx1 = nx+2; ny1=ny+2; dim = nx1*ny1;
hx = (b-a)/(nx+1); hy = (d-c)/(ny+1);
    hx2 = hx^2; hy2 = hy^2;
aix = 2/hx2+2/hy2; aix = -1/hx2; aiy = -1/hy2;
A = speye(dim,dim); f = zeros(dim,1);
y = c;
for m = 2:ny+1
    x = a; y = y + hy;
    for n = 2:nx+1
        i = n+(m-1)*nx1; x = x + hx;
        f(i) = fun(x,y,varargin{:});
        A(i,i) = aii; A(i,i-1) = aix; A(i,i+1) = aix;
        A(i,i+nx1) = aiy; A(i,i-nx1) = aiy;
    end
end
ub = zeros(dim,1); xh = [a:hx:b]'; yh = [c:hy:d];
ub(1:nx1) = gd(xh,c,varargin{:});
ub(dim-nx-1:dim) = gd(xh,d,varargin{:});
ub(1:nx1:dim-nx-1) = gd(a,yh,varargin{:});
ub(nx1:nx1:dim) = gd(b,yh,varargin{:});
f = f - A*ub;
nbound = [[1:nx1],[dim-nx-1:dim],[1:nx1:dim-nx-1],...
           [nx1:nx1:dim]];
ninternal = setdiff([1:dim],nbound);
A = A(ninternal,ninternal);
f = f(ninternal);
utemp = A\ f;
u = ub; u(ninternal) = utemp;
k = 1; y = c;
for j = 1:ny1
    x = a;
    for i = 1:nx1
        uh(j,i) = u(k); k = k + 1;
        ue(j,i) = uex(x,y,varargin{:});
    end
end

```

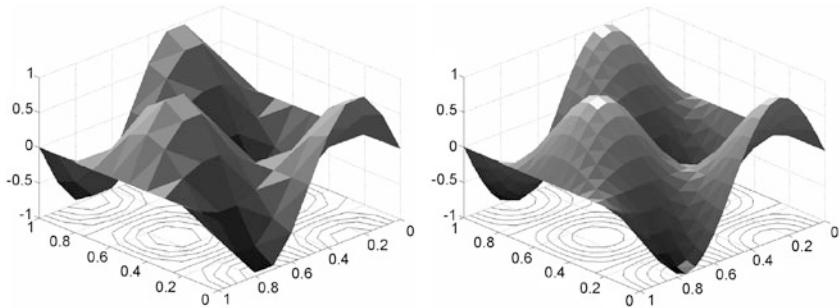


Figura 9.9. Spostamento trasversale di una membrana calcolato con una griglia più rada (*a sinistra*) e con una più fitta (*a destra*). Sul piano orizzontale vengono riportate le isoline della soluzione. La decomposizione in triangoli che compare nelle figure tridimensionali serve per la sola visualizzazione

```

x = x + hx;
end
y = y + hy;
end
if nargout == 4 && nargin >= 9
    error = max(max(abs(uh-ue)))/max(max(abs(ue)));
elseif nargout == 4 && nargin == 8
    warning('Soluzione esatta non disponibile');
    error = [ ];
end
end

```

Esempio 9.3 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 (9.2) in Ω . Le condizioni al bordo di Dirichlet sullo spostamento sono: $g_D = 0$ sui lati $x = 0$ e $x = 1$, e $g_D(x, 0) = g_D(x, 1) = \sin(2\pi x)$, $0 < x < 1$. Questo problema ammette come soluzione esatta la funzione $u(x, y) = \sin(2\pi x) \cos(2\pi y)$. In Figura 9.9 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 (9.58) passa da 0.0292 per $h = 1/10$ a 0.0081 per $h = 1/20$. ■

Anche il metodo degli elementi finiti può essere facilmente esteso al caso bidimensionale. Si dovrà scrivere una opportuna formulazione integrale del problema (9.2) e sostituire alla decomposizione dell'intervallo (a, b) in sottointervalli una decomposizione in poligoni (tipicamente, triangoli) detti *elementi*. La generica funzione di forma 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 (rimandiamo a [Qua16] per un'ampia descrizione del metodo agli elementi finiti in 2 e 3 dimensioni). Per una implemen-

tazione di tale metodo in 2 dimensioni si può usare il toolbox `pdetool` `pdetool`
di MATLAB o il package `fem-fenics` di Octave. `fem-fenics`

9.7.1 Analisi dell'approssimazione con differenze finite del problema di Poisson in 2 dimensioni

Nella precedente Sezione abbiamo stabilito che la soluzione del problema approssimato esiste ed è unica. Come abbiamo fatto per il caso monodimensionale nella Sezione 9.3.1, siamo ora interessati a stimare l'errore di approssimazione. Supponiamo sempre che $h_x = h_y = h$. Sia $u(x, y)$ è la soluzione del problema continuo. Se

$$\max_{i,j} |u(x_i, y_j) - u_{i,j}| \leq C(h) \quad \text{quando } h \rightarrow 0 \quad (9.59)$$

essendo $C(h)$ un infinitesimo rispetto a h , ovvero $\lim_{h \rightarrow 0} C(h) = 0$, diremo che il metodo usato per calcolare la soluzione numerica rappresentata dai valori nodali u_j è convergente.

Si dimostra che lo schema a 5 punti (9.55) è convergente, in particolare vale il seguente risultato [IK66].

Proposizione 9.1 *Supponiamo che $u \in C^4(\overline{\Omega})$, cioè che la soluzione esatta abbia tutte le sue derivate parziali fino al quart'ordine continue sulla chiusura $\overline{\Omega}$ del dominio. Allora, esiste una costante $C > 0$ tale che*

$$\max_{i,j} |u(x_i, y_j) - u_{i,j}| \leq CMh^2 \quad (9.60)$$

where M è il massimo valore assoluto assunto dalle derivate quarte di u in $\overline{\Omega}$.

Accenniamo sinteticamente alla dimostrazione della stima (9.60) seguendo la traccia data nella Sezione 9.3.1 per il caso monodimensionale, cioè riconducendoci all'analisi di consistenza e di stabilità dello schema.

L'errore di troncamento locale τ_h per lo schema (9.55) è definito da

$$\begin{aligned} \tau_h(x_i, y_j) = & -f(x_i, y_j) \\ & - \frac{u(x_{i-1}, y_j) + u(x_i, y_{j-1}) - 4u(x_i, y_j) + u(x_i, y_{j+1}) + u(x_{i+1}, y_j)}{h^2} \end{aligned} \quad (9.61)$$

per ogni nodo (x_i, y_j) interno a Δ_h . Procedendo come nel caso monodimensionale, si dimostra che $\tau_h(x_i, y_j) = \mathcal{O}(h^2)$ per $h \rightarrow 0$, ovvero lo

schema (9.55) è consistente di ordine 2 (la definizione (9.27) viene estesa al caso bidimensionale).

La stabilità che si invoca qui è dello stesso tipo di quella descritta nella Sezione 9.3.1, ovvero chiediamo che la soluzione numerica dipenda con continuità dai dati del problema, come in (9.30). Si può dimostrare che lo schema (9.55) soddisfa la seguente stima di stabilità [IK66, Cap 9.1]:

$$\max_{(i,j) \in D_h} |u_{i,j}| \leq \max_{(i,j) \in C_h} |g_D(x_i, y_j)| + C \max_{(i,j) \in D_h} |f(x_i, y_j)|, \quad (9.62)$$

essendo $D_h = \{(i,j) : (x_i, y_j) \in \Delta_h \setminus \partial\Delta_h\}$, $C_h = \{(i,j) : (x_i, y_j) \in \partial\Delta_h\}$ e C una costante positiva che dipende dal dominio Ω , ma che è indipendente da h .

A questo punto, come abbiamo fatto nel caso monodimensionale, basta definire l'errore $e_{i,j} = u(x_i, y_j) - u_{i,j}$ nei nodi di Δ_h , sottrarre l'equazione (9.61) da (9.55), osservare che $e_{i,j} = 0$ in ogni nodo $(x_i, y_j) \in \partial\Delta_h$ per aver imposto le condizioni al bordo di Dirichlet (9.54), e concludere che

$$\max_{(i,j) \in D_h \cup C_h} |e_{i,j}| \leq C \max_{(i,j) \in D_h} |\tau_h(x_i, y_j)| \leq \tilde{C} h^2 \max_{(i,j) \in D_h} |f(x_i, y_j)|. \quad (9.63)$$

Esempio 9.4 Verifichiamo sperimentalmente che il metodo a 5 punti applicato al problema di Poisson dell'Esempio 9.3 converge con ordine 2 rispetto a h . Risolviamo tale problema a partire da $h = 1/4$ e, per dimezzamenti successivi, fino a $h = 1/64$ con le seguenti istruzioni `MAT&OCT`:

```
a=0; b=1; c=0; d=1;
f=@(x,y) 8*pi^2*sin(2*pi*x).*cos(2*pi*y);
gd=@(x,y) sin(2*pi*x).*cos(2*pi*y);
uex=gd; nx=4; ny=4;
for n=1:5
    [xh ,yh ,uh ,err(n)]=poisson_fd_2d(a,b,c,d, ...
                                         nx,ny,f,gd,uex);
    nx = 2*nx; ny = 2*ny;
end
```

Il vettore contenente l'errore al variare di h è

```
err =
1.3565e-1 4.3393e-2 1.2308e-2 3.2775e-3 8.4557e-4
```

Con il comando (si veda la formula (1.13)):

```
p=log(abs(err(1:end-1)./err(2:end)))/log(2)
```

otteniamo infatti

```
p =
1.6443      1.8179      1.9089      1.9546
```

pertanto il metodo è convergente con ordine 2 quando $h \rightarrow 0$. ■



Si veda l'Esercizio 9.13.

9.8 Approssimazione dell'equazione del calore monodimensionale

Consideriamo l'equazione del calore monodimensionale

$$\begin{aligned} \frac{\partial u}{\partial t}(x, t) - \mu \frac{\partial^2 u}{\partial x^2}(x, t) &= f(x, t) \quad x \in (a, b), \quad t > 0, \\ u(a, t) = u(b, t) &= 0 \quad t > 0, \\ u(x, 0) &= u^0(x) \quad x \in (a, b) \end{aligned} \tag{9.64}$$

dove $f(x, t)$ e $u^0(x)$ sono funzioni assegnate, $(9.64)_2$ sono le condizioni al bordo di Dirichlet omogenee, mentre $(9.64)_3$ è la condizione iniziale al tempo $t = 0$.

Per risolvere numericamente (9.64) dovremo approssimare sia le derivate rispetto a x che quelle rispetto a t e quindi dovremo discretizzare sia l'intervallo spaziale (a, b) sia quello temporale $(0, T)$. Il passo di discretizzazione in spazio verrà indicato con h , mentre quello in tempo con Δt . Infine, se x_j e t^n sono due punti delle discretizzazioni in spazio e tempo, rispettivamente, u_j^n sarà un'approssimazione del valore esatto $u(x_j, t^n)$.

Nelle prossime Sezioni consideriamo l'approssimazione di (9.64) con i metodi delle differenze finite e degli elementi finiti.

9.8.1 Approssimazione alle differenze finite dell'equazione del calore monodimensionale

Discretizziamo anzitutto rispetto alla variabile x seguendo l'approccio delle differenze finite utilizzato nella Sezione 9.3.

Per ogni $t > 0$ denotiamo con $u_j(t)$ una approssimazione di $u(x_j, t)$, per $j = 0, \dots, N+1$, e approssimiamo il problema (9.64) con il seguente schema: per ogni $t > 0$:

$$\begin{cases} \frac{du_j}{dt}(t) - \frac{\mu}{h^2}(u_{j-1}(t) - 2u_j(t) + u_{j+1}(t)) = f_j(t), & j = 1, \dots, N, \\ u_0(t) = u_{N+1}(t) = 0, \end{cases}$$

dove $f_j(t) = f(x_j, t)$, mentre per $t = 0$ imponiamo la condizione iniziale

$$u_j(0) = u^0(x_j), \quad j = 0, \dots, N+1.$$

Questa è in realtà una *semi-discretizzazione* dell'equazione del calore, che produce un sistema di equazioni differenziali ordinarie del tipo:

$$\begin{cases} \frac{d\mathbf{u}}{dt}(t) = -\mu A_{fd}\mathbf{u}(t) + \mathbf{f}(t) & \forall t > 0, \\ \mathbf{u}(0) = \mathbf{u}^0, \end{cases} \quad (9.65)$$

dove $\mathbf{u}(t) = (u_1(t), \dots, u_N(t))^T$ è il vettore colonna delle incognite, $\mathbf{f}(t) = (f_1(t), \dots, f_N(t))^T$, $\mathbf{u}^0 = (u^0(x_1), \dots, u^0(x_N))^T$ e A_{fd} è la matrice tridiagonale definita in (9.20). Osserviamo che per derivare (9.65) abbiamo supposto che $u^0(x_0) = u^0(x_{N+1}) = 0$, il che è coerente con l'aver imposto condizioni al bordo di Dirichlet omogenee.

Uno schema classico per l'integrazione in tempo di (9.65) è il cosiddetto *θ -metodo*, θ essendo un parametro: $0 \leq \theta \leq 1$. Consideriamo un passo di discretizzazione in tempo $\Delta t > 0$ costante, quindi denotiamo con v^k il valore che la variabile v assume al livello temporale $t^k = k\Delta t$. Il θ -metodo prende la seguente forma:

$$\boxed{\begin{aligned} \frac{\mathbf{u}^{k+1} - \mathbf{u}^k}{\Delta t} &= -\mu A_{fd}(\theta \mathbf{u}^{k+1} + (1-\theta) \mathbf{u}^k) + \theta \mathbf{f}^{k+1} + (1-\theta) \mathbf{f}^k, \\ &\quad k = 0, 1, \dots \\ \mathbf{u}^0 &\text{ assegnato} \end{aligned}} \quad (9.66)$$

o, equivalentemente,

$$(I + \mu\theta\Delta t A_{fd}) \mathbf{u}^{k+1} = (I - \mu\Delta t(1-\theta)A_{fd}) \mathbf{u}^k + \mathbf{g}^{k+1}, \quad (9.67)$$

avendo posto $\mathbf{g}^{k+1} = \Delta t(\theta \mathbf{f}^{k+1} + (1-\theta) \mathbf{f}^k)$ ed avendo indicato con I la matrice identità di dimensione N .

Per ogni k , (9.67) è un sistema lineare di dimensione N la cui matrice è

$$K = I + \mu\theta\Delta t A_{fd}.$$

Poiché la matrice A_{fd} è simmetrica e definita positiva, anche la matrice K risulta essere tale. Essa, inoltre, è invariante rispetto a k e pertanto può essere fattorizzata una volta per tutte al tempo $t = 0$. Nel caso monodimensionale in esame tale fattorizzazione è basata sull'algoritmo di Thomas (si veda la sezione 5.6) e richiede quindi un numero di operazioni proporzionale a N .

Per valori particolari del parametro θ possiamo riconoscere in (9.67) alcuni metodi già introdotti nel Capitolo 8. Per esempio, se $\theta = 0$ il metodo (9.67) coincide con lo schema di Eulero in avanti e possiamo ottenere \mathbf{u}^{k+1} esplicitamente; altrimenti, ad ogni passo temporale dobbiamo risolvere un sistema lineare con matrice K .

Nel caso in cui $f = 0$ la soluzione esatta $u(x, t)$ di (9.4) tende a zero per ogni x , per $t \rightarrow \infty$. Se anche la soluzione numerica ha lo stesso tipo di comportamento, lo schema (9.67) è detto *asintoticamente stabile*, un concetto analogo a quello di assoluta stabilità introdotto nella Sezione 8.5 nel caso delle equazioni differenziali ordinarie.

Al fine di analizzare la stabilità asintotica, consideriamo dunque l'equazione vettoriale (9.67) nel caso in cui $\mathbf{g}^{k+1} = \mathbf{0} \forall k \geq 0$. Se $\theta = 0$ troviamo

$$\mathbf{u}^k = (\mathbf{I} - \mu \Delta t \mathbf{A}_{\text{fd}})^k \mathbf{u}^0, \quad k = 1, 2, \dots$$

pertanto $\mathbf{u}^k \rightarrow \mathbf{0}$ per $k \rightarrow \infty$ se e solo se

$$\rho(\mathbf{I} - \mu \Delta t \mathbf{A}_{\text{fd}}) < 1. \quad (9.68)$$

D'altro canto, gli autovalori λ_j di \mathbf{A}_{fd} sono, per $j = 1, \dots, N$ (si veda l'Esercizio 9.4):

$$\lambda_j = \frac{2}{h^2} (1 - \cos(j\pi/(N+1))) = \frac{4}{h^2} \sin^2(j\pi/(2(N+1))).$$

Quindi (9.68) è soddisfatta se

$$\Delta t < \frac{1}{2\mu} h^2.$$

Come ci saremmo aspettati, il metodo di Eulero in avanti è asintomaticamente stabile sotto la condizione che Δt decresca come il quadrato del parametro h di discretizzazione spaziale quando $h \rightarrow 0$.

Nel caso del metodo di Eulero all'indietro ($\theta = 1$), da (9.67) otteniamo

$$\mathbf{u}^k = [(\mathbf{I} + \mu \Delta t \mathbf{A}_{\text{fd}})^{-1}]^k \mathbf{u}^0, \quad k = 1, 2, \dots$$

Poiché tutti gli autovalori della matrice $(\mathbf{I} + \mu \Delta t \mathbf{A}_{\text{fd}})^{-1}$ sono reali, positivi e strettamente minori di 1 per ogni valore di Δt , questo schema è incondizionatamente asintoticamente stabile. Più in generale, il θ -metodo è incondizionatamente asintoticamente stabile per tutti i valori $1/2 \leq \theta \leq 1$ e condizionatamente asintoticamente stabile se $0 \leq \theta < 1/2$ (si veda ad esempio [QSSG14, Cap. 12]).

Per quanto riguarda l'accuratezza del θ -metodo, possiamo affermare che l'errore di troncamento locale è dell'ordine di $\Delta t + h^2$ se $\theta \neq \frac{1}{2}$ mentre è dell'ordine di $\Delta t^2 + h^2$ se $\theta = \frac{1}{2}$. Nell'ultimo caso si ottiene il cosiddetto metodo di Crank-Nicolson (si veda la Sezione 8.3) che è quindi incondizionatamente asintoticamente stabile; il corrispondente metodo totalmente discretizzato è pertanto accurato al second'ordine sia rispetto a Δt sia rispetto a h .

Si perviene alle stesse conclusioni qualora si consideri l'equazione del calore in domini bidimensionali. In tal caso nello schema (9.66) si deve sostituire la matrice \mathbf{A}_{fd} definita in (9.20) con la matrice \mathbf{A}_{fd} definita in (9.56).

Il Programma 9.4 risolve numericamente l'equazione del calore sull'intervallo $(0, T)$ e sul dominio $\Omega = (a, b)$ utilizzando il θ -metodo per la

discretizzazione temporale. Al bordo sono imposte condizioni di Dirichlet (anche non omogenee) $u(x, t) = g_D(x, t)$ per $x = a, x = b$ e $t \in (0, T)$, essendo $g_D(x, t)$ una funzione assegnata.

Il dato di Dirichlet è trattato in maniera analoga a come è stato fatto per il problema di Poisson nella Sezione 9.3. Più precisamente, i termini di destra della prima e dell'ultima riga del sistema (9.67) vengono modificati ad ogni passo k aggiungendo i contributi che coinvolgono il dato di Dirichlet al bordo e che derivano dall'approssimazione della derivata seconda. Ad esempio, denotando con \mathbf{b}^{k+1} il vettore di destra dell'equazione (9.67), la modifica da effettuare sulla componente b_1^{k+1} è

$$b_1^{k+1} = b_1^{k+1} + \mu\theta\Delta t \frac{1}{h^2} g_D(a, t^{k+1}) + \mu(1-\theta)\Delta t \frac{1}{h^2} g_D(a, t^k);$$

analogamente bisognerà modificare la componente b_N^{k+1} .

I parametri di input sono i vettori `xspan=[a, b]`, e `tspan=[0, T]`, il numero di intervalli della discretizzazione in spazio (`nstep(1)`) ed in tempo (`nstep(2)`), lo scalare `mu` che contiene il coefficiente reale positivo μ , i *function handle* `u0`, `fun` e `gd` che contengono il dato iniziale $u^0(x)$, il termine forzante $f(x, t)$ e il dato di Dirichlet $g_D(x, t)$, rispettivamente. Infine, la variabile `theta` contiene il coefficiente θ . La variabile di *output* `uh` contiene la soluzione numerica all'istante finale $t = T$.

Programma 9.4. heat_fd_1d: θ -metodo e differenze finite per l'equazione del calore monodimensionale

```
function [xh,uh]=heat_fd_1d(xspan,tspan,nstep,mu,%
                               u0,gd,f,theta,varargin)
%HEAT_FD_1D risolve l'equazione del calore con il
% theta-metodo in tempo e differenze finite in spazio.
% [XH,UH]=HEAT_FD_1D(XSPAN,TSPAN,NSTEP,MU,U0,GD,F,%
% THETA) risolve l'equazione del calore
% D U/DT - MU D^2U/DX^2 = F nel dominio
% (XSPAN(1),XSPAN(2))x(TSPAN(1),TSPAN(2)) utilizzando
% il theta-metodo con condizione iniziale U(X,0)=U0(X)
% e condizioni al bordo di Dirichlet U(X,T)=GD(X,T)
% per X=XSPAN(1) e X=XSPAN(2). MU>0 e' una costante.
% F=F(X,T), GD=GD(X,T) e U0=U0(X) sono function handle
% o user defined function.
% NSTEP(1) e' il n.ro di intervalli in spazio
% NSTEP(2) e' il n.ro di intervalli in tempo
% XH contiene i nodi della discretizzazione
% UH contiene la soluzione numerica al tempo TSPAN(2).
% [XH,UH]=HEAT_FD_1D(XSPAN,TSPAN,NSTEP,MU,U0,GD,F,%
% THETA,P1,P2,...) passa i parametri opzionali
% P1,P2,...to alle funzioni U0,GD,F.
h = (xspan(2)-xspan(1))/nstep(1);
dt = (tspan(2)-tspan(1))/nstep(2);
N = nstep(1)-1; e = ones(N,1);
Afd = spdiags([-e 2*e -e],[-1,0,1],N,N)/h^2;
I = speye(N);
A = I+mu*dt*theta*Afd; An = I-mu*dt*(1-theta)*Afd;
```

```

ad=theta*mu/h^2*dt; adn=(1-theta)*mu/h^2*dt;
xh = (linspace(xspan(1),xspan(2),N+2))';
xhi=xh(2:end-1);
fn = f(xhi,tspan(1),varargin{:});
un = u0(xhi,varargin{:});
R=chol(A);
gdn=gd([xspan(1);xspan(2)],tspan(1),varargin{:});
for t = tspan(1)+dt:dt:tspan(2)
    fn1 = f(xhi,t,varargin{:});
    b = An*un+dt*(theta*fn1+(1-theta)*fn);
    gdn1 = gd([xspan(1);xspan(2)],t,varargin{:});
    b([1,end]) = b([1,end])+ad*gdn1+adn*gdn;
    u = R'\b; un = R\u; fn = fn1; un = u;
    gdn=gdn1;
end
uh=[gdn1(1);u;gdn1(end)];

```

Esempio 9.5 Consideriamo l'equazione del calore (9.4) in $(a, b) = (0, 1)$ con $\mu = 1$, $f(x, t) = -\sin(x)\sin(t) + \sin(x)\cos(t)$, condizione iniziale $u(x, 0) = \sin(x)$ e condizioni al bordo $u(0, t) = 0$ e $u(1, t) = \sin(1)\cos(t)$. In questo caso la soluzione è $u(x, t) = \sin(x)\cos(t)$. In Figura 9.10 confrontiamo il comportamento degli errori $\max_{i=0,\dots,N} |u(x_i, 1) - u_i^M|$ rispetto al passo temporale su una griglia uniforme lungo la direzione spaziale con tre valori di h . I valori $\{u_i^M\}$ rappresentano la soluzione alle differenze finite calcolata al tempo $t^M = 1$. Come ci si poteva aspettare, per $\theta = 0.5$ il θ -metodo è accurato al secondo ordine rispetto a Δt , fino a quando il passo temporale risulta talmente piccolo che l'errore in spazio domina sull'errore generato dall'approssimazione in tempo. ■

Esempio 9.6 (Termodinamica) Consideriamo una barra omogenea di alluminio lunga tre metri e con sezione uniforme. Siamo interessati a simulare l'evoluzione della temperatura nella barra partendo da una opportuna condizione iniziale, risolvendo l'equazione del calore (9.5). Se noi imponiamo condizioni

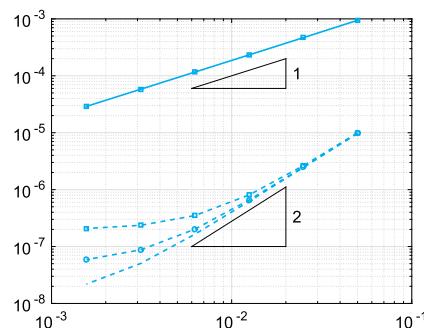


Figura 9.10. L'errore al variare di Δt per il θ -metodo (per $\theta = 1$ in linea continua, e $\theta = 0.5$ in linea tratteggiata), per tre differenti valori di h : 0.008 (\square), 0.004 (\circ) e 0.002 (nessun simbolo)

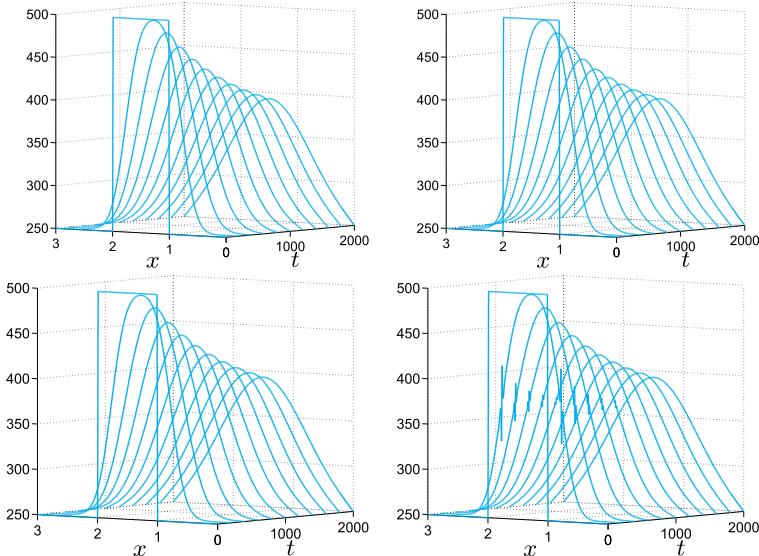


Figura 9.11. Profili di temperatura in una barra di alluminio a differenti istanti temporali (da $t = 0$ a $t = 2000$ secondi con passo di integrazione Δt di 0.25 secondi (*in alto*) e 20 secondi (*in basso*), ottenuti utilizzando gli schemi di Eulero all'indietro (*a sinistra*) e Crank-Nicolson (*a destra*).

adiabatiche sulla superficie laterale della barra (ovvero condizioni di Neumann omogenee) e condizioni di Dirichlet alle estremità della barra, la temperatura dipende solo dalla coordinata assiale (denotata con x). Quindi il problema può essere modellato dall'equazione del calore monodimensionale (9.9), completata con la condizione iniziale al tempo $t = t_0$ e da condizioni al bordo di Dirichlet agli estremi del dominio computazionale ridotto $\Omega = (0, L)$ ($L = 3$ m). L'alluminio puro ha conducibilità termica pari a $k = 237$ W/(m K), una densità $\rho = 2700$ kg/m³ e una capacità di calore specifico $c = 897$ J/(kg K), di conseguenza la diffusività termica è $\mu = 9.786 \cdot 10^{-5}$ m²/s. Infine consideriamo la condizione iniziale $T(x, 0) = 500$ K se $x \in (1, 2)$, 250 K altrimenti (si noti che tale funzione è discontinua) e condizioni al bordo di Dirichlet $T(0, t) = T(3, t) = 250$ K. In Figura 9.11 riportiamo l'evoluzione della temperatura della barra calcolata con il metodo di Eulero all'indietro ($\theta = 1$, a sinistra) e con il metodo di Crank-Nicolson ($\theta = 0.5$, a destra) utilizzando il Programma 9.4. In entrambi i casi sono state utilizzate differenze finite centrate per la discretizzazione in spazio con passo $h = 0.01$. I risultati mostrano che quando il passo di integrazione temporale è grande ($\Delta t = 20$ sec), il metodo di Crank-Nicolson mostra delle leggere oscillazioni nei primi passi temporali a causa della discontinuità del dato iniziale. (Riguardo a tale questione, si veda anche [QV94, Chapter 11].) Al contrario, il metodo di Eulero implicito produce una soluzione stabile in quanto è uno schema più dissipativo di Crank-Nicolson. Entrambi i metodi calcolano una soluzione che decade al valore corretto di 250 K quando $t \rightarrow \infty$. ■

9.8.2 Approssimazione ad elementi finiti dell'equazione del calore monodimensionale

Consideriamo l'equazione (9.64) con condizioni al bordo di Dirichlet omogenee $u(a, t) = u(b, t) = 0$, $\forall t > 0$. Per discretizzarla in spazio ora utilizziamo il metodo di Galerkin agli elementi finiti procedendo come abbiamo fatto nella Sezione 9.5 nel caso dell'equazione di Poisson. Anzitutto, per ogni $t > 0$ moltiplichiamo la (9.4) per una funzione test $v = v(x) \in C^1([a, b])$ che si annulla in $x = a$ ed in $x = b$ e poi integriamo su (a, b) applicando la formula di integrazione per parti al termine contenente la derivata seconda in x . Quindi, $\forall t > 0$ cerchiamo una funzione $x \mapsto u(x, t) \in C^1([a, b])$ tale che

$$\begin{aligned} & \int_a^b \frac{\partial u}{\partial t}(x, t)v(x)dx + \int_a^b \mu \frac{\partial u}{\partial x}(x, t) \frac{dv}{dx}(x)dx \\ &= \int_a^b f(x)v(x)dx \quad \forall v \in C^1([a, b]), \end{aligned} \quad (9.69)$$

con $u(x, 0) = u^0(x)$. Da qui in avanti, per alleggerire le notazioni, la dipendenza di u , v e f dalla variabile spaziale x verrà sottintesa. Come l'equazione (9.40), anche la (9.69) di fatto continua a valere per funzioni u e v meno regolari di $C^1([a, b])$, come ad esempio quelle globalmente continue ma derivabili solo a tratti in $[a, b]$.

Sia V_h lo spazio di dimensione finita definito in (9.42), $V_h^0 \subset V_h$ il sottospazio delle funzioni di V_h che si annullano in $x = a$ ed in $x = b$, e consideriamo il seguente problema di Galerkin: $\forall t > 0$, trovare $u_h(t) \in V_h^0$ tale che

$$\begin{aligned} & \int_a^b \frac{\partial u_h}{\partial t}(t)v_h dx + \int_a^b \mu \frac{\partial u_h}{\partial x}(t) \frac{dv_h}{dx} dx \\ &= \int_a^b f(t)v_h dx \quad \forall v_h \in V_h^0, \end{aligned} \quad (9.70)$$

dove $u_h(0) = u_h^0$ e $u_h^0 \in V_h$ è una conveniente approssimazione di u^0 . La formulazione (9.70) rappresenta una *semi-discretizzazione* (solo spaziale) del problema (9.69).

Per quanto concerne la discretizzazione ad elementi finiti di (9.70), consideriamo le funzioni di forma φ_j (9.44) che costituiscono una base per lo spazio V_h^0 . Quindi, la soluzione $u_h \in V_h^0$ di (9.70) si può scrivere come

$$u_h(t) = \sum_{j=1}^N u_j(t) \varphi_j,$$

dove le quantità $\{u_j(t)\}$ sono i coefficienti incogniti e N rappresenta la dimensione di V_h^0 . Dalla (9.70), prendendo $v_h = \varphi_i$ per $i = 1, \dots, N$, si ottiene

$$\begin{aligned} & \int_a^b \sum_{j=1}^N \frac{du_j}{dt}(t) \varphi_j \varphi_i dx + \mu \int_a^b \sum_{j=1}^N u_j(t) \frac{d\varphi_j}{dx} \frac{d\varphi_i}{dx} dx \\ & = \int_a^b f(t) \varphi_i dx, \quad i = 1, \dots, N \end{aligned}$$

ossia

$$\begin{aligned} & \sum_{j=1}^N \frac{du_j}{dt}(t) \int_a^b \varphi_j \varphi_i dx + \mu \sum_{j=1}^N u_j(t) \int_a^b \frac{d\varphi_j}{dx} \frac{d\varphi_i}{dx} dx \\ & = \int_a^b f(t) \varphi_i dx, \quad i = 1, \dots, N. \end{aligned}$$

Utilizzando la medesima notazione impiegata nella (9.65), otteniamo

$$M \frac{d\mathbf{u}}{dt}(t) + A_{fe} \mathbf{u}(t) = \mathbf{f}_{fe}(t), \quad (9.71)$$

dove $(A_{fe})_{ij} = \mu \int_a^b \frac{d\varphi_j}{dx} \frac{d\varphi_i}{dx} dx$, $(\mathbf{f}_{fe}(t))_i = \int_a^b f(t) \varphi_i dx$ e M è la matrice di massa i cui elementi sono $M_{ij} = \int_a^b \varphi_j(x) \varphi_i(x) dx$. Osserviamo che, a differenza della formulazione alle differenze finite, qui il coefficiente μ è stato inglobato nella definizione della matrice A_{fe} .

Per risolvere (9.71) in modo approssimato possiamo ancora ricorrere al θ -metodo, ottenendo:

$$M \frac{\mathbf{u}^{k+1} - \mathbf{u}^k}{\Delta t} + A_{fe} [\theta \mathbf{u}^{k+1} + (1 - \theta) \mathbf{u}^k] = \theta \mathbf{f}_{fe}^{k+1} + (1 - \theta) \mathbf{f}_{fe}^k,$$

$$k = 0, 1, \dots$$

\mathbf{u}^0 assegnato

(9.72)

o, equivalentemente,

$$\left(\frac{1}{\Delta t} M + \theta A_{fe} \right) \mathbf{u}^{k+1} = \left(\frac{1}{\Delta t} M - (1 - \theta) A_{fe} \right) \mathbf{u}^k + \mathbf{g}^{k+1}, \quad (9.73)$$

avendo posto $\mathbf{g}^{k+1} = \theta \mathbf{f}_{fe}^{k+1} + 1 - \theta \mathbf{f}_{fe}^k$.

Come nel caso delle differenze finite, per $\theta = 0, 1$ e $1/2$, otteniamo rispettivamente i metodi di Eulero in avanti, di Eulero all'indietro e di Crank-Nicolson, essendo quest'ultimo l'unico di ordine due rispetto a Δt .

Per ogni k , (9.73) è un sistema lineare di dimensione N la cui matrice è

$$K = \frac{1}{\Delta t} M + \theta A_{fe}.$$

Poiché entrambe le matrici M e A_{fe} sono simmetriche e definite positive, anche la matrice K risulta essere tale. Essa, inoltre, è invariante rispetto a k e pertanto può essere fattorizzata una volta per tutte al tempo $t = 0$. Nel caso monodimensionale in esame tale fattorizzazione è basata sull'algoritmo di Thomas (si veda la Sezione 5.6) e richiede quindi un numero di operazioni proporzionale a N . Nel caso multidimensionale converrà invece fare ricorso alla decomposizione di Cholesky $K = R^T R$, essendo R una matrice triangolare superiore (si veda (5.17)). Pertanto, ad ogni istante temporale è necessario risolvere i due seguenti sistemi lineari triangolari, ciascuno di dimensione pari a N :

$$\begin{cases} R^T \mathbf{y} = \left[\frac{1}{\Delta t} M - (1-\theta) A_{fe} \right] \mathbf{u}^k + \theta \mathbf{f}_{fe}^{k+1} + (1-\theta) \mathbf{f}_{fe}^k, \\ R \mathbf{u}^{k+1} = \mathbf{y}. \end{cases}$$

Quando $\theta = 0$, una opportuna diagonalizzazione di M permetterebbe di disaccoppiare fra loro le equazioni del sistema (9.72). Questa procedura è nota come *mass-lumping* e consiste nell'approssimare M con una matrice diagonale non singolare \tilde{M} .

Nel caso di elementi finiti di grado 1 per problemi monodimensionali, \tilde{M} può essere ricavata usando la formula composita del trapezio sui nodi $\{x_i\}$ per calcolare gli integrali $\int_a^b \varphi_j \varphi_i dx$, ottenendo $\tilde{m}_{ij} = h \delta_{ij}$, $i, j = 1, \dots, N$.

Se $\theta \geq 1/2$, il θ -metodo è incondizionatamente stabile per ogni valore positivo di Δt , mentre se $0 \leq \theta < 1/2$ il θ -metodo è stabile solo se

$$0 < \Delta t \leq \frac{2}{(1-2\theta)\lambda_{\max}(M^{-1}A_{fe})},$$

si veda a tale proposito [Qua16, Cap. 5]. Inoltre, si può dimostrare che esistono due costanti positive c_1 e c_2 , indipendenti da h , tali che

$$c_1 h^{-2} \leq \lambda_{\max}(M^{-1}A_{fe}) \leq c_2 h^{-2}$$

(per la dimostrazione, si veda [QV94], Sezione 6.3.2). In base a questa proprietà, otteniamo che se $0 \leq \theta < 1/2$ il metodo è stabile solo se

$$0 < \Delta t \leq C_1(\theta) h^2, \quad (9.74)$$

per una opportuna costante $C_1(\theta)$ indipendente da entrambi i parametri h e Δt .

Il Programma 9.5 risolve numericamente l'equazione del calore nell'intervallo temporale $(0, T)$ e sul dominio spaziale $\Omega = (a, b)$, utilizzando il θ -metodo per la discretizzazione temporale ed elementi finiti di grado 1 per la discretizzazione in spazio. Al bordo sono imposte condizioni di Dirichlet (anche non omogenee) $u(x, t) = g_D(x, t)$ per $x = a$, $x = b$ e $t \in (0, T)$, essendo $g_D(x, t)$ una funzione assegnata.

Il dato di Dirichlet è trattato in maniera analoga a come è stato fatto per il problema di Poisson nella Sezione 9.5. Più precisamente, i termini noti della prima e dell'ultima equazione del sistema (9.67) vengono modificati ad ogni passo k aggiungendo i contributi che coinvolgono il dato di Dirichlet al bordo e che derivano dall'approssimazione della derivata seconda e dalla matrice di massa M .

Denotiamo con \mathbf{b}^{k+1} il vettore di destra dell'equazione (9.73) e calcoliamo gli integrali che coinvolgono la funzione f con il metodo del punto medio come in (9.49). Allora la modifica da effettuare sulla componente b_1^{k+1} è

$$b_1^{k+1} = b_1^{k+1} + \left(-\frac{h}{6\Delta t} + \theta \frac{\mu}{h} \right) g_D(a, t^{k+1}) + \left(\frac{h}{6\Delta t} + (1-\theta) \frac{\mu}{h} \right) g_D(a, t^k).$$

In modo analogo bisognerà modificare la componente b_N^{k+1} .

I parametri di input e di output del Programmma 9.5 hanno il medesimo significato di quelli del Programma 9.4.

Programma 9.5. heat_fe_1d: θ -metodo ed elementi finiti di grado 1 per l'equazione del calore monodimensionale

```

function [xh,uh]=heat_fe_1d(xspan,tspan,nstep,mu,%
    u0,gd,f,theta,varargin)
%HEAT_FE_1D risolve l'equazione del calore con il
% theta-metodo in tempo ed elementi finiti in spazio.
% [XH,UH]=HEAT_FE_1D(XSPAN,TSPAN,NSTEP,MU,U0,GD,F,%
% THETA) risolve l'equazione del calore
% D U/DT - MU D^2U/DX^2 = F nel dominio
% (XSPAN(1),XSPAN(2))x(TSPAN(1),TSPAN(2)) utilizzando
% il theta-metodo con condizione iniziale U(X,0)=U0(X)
% e condizioni al bordo di Dirichlet U(X,T)=GD(X,T)
% per X=XSPAN(1) e X=XSPAN(2). MU e' una costante
% positiva. F=F(X,T), GD=GD(X,T) e U0=U0(X) sono
% function handle o user defined function.
% NSTEP(1) e' il n.ro di intervalli in spazio
% NSTEP(2) e' il n.ro di intervalli in tempo
% XH contiene i nodi della discretizzazione
% UH contiene la soluzione numerica al tempo TSPAN(2).
% [XH,UH]=HEAT_FE_1D(XSPAN,TSPAN,NSTEP,MU,U0,GD,F,%
% THETA,P1,P2,...) passa i parametri opzionali
% P1,P2,...to alle funzioni U0, GD, F.
h = (xspan(2)-xspan(1))/nstep(1);
dt = (tspan(2)-tspan(1))/nstep(2);
N = nstep(1)-1; theta1=1-theta;
e = ones(N,1); hm=mu/h;
Afe = spdiags([-hm*e 2*hm*e -hm*e], -1:1, N, N);
M= spdiags([e 4*e e], -1:1, N, N)/6*h;
A = M/dt+theta*Afe; An = M/dt-theta1*Afe;
ad=-h/(6*dt)+theta*hm; adn=h/(6*dt)+theta1*hm;
xh = (linspace(xspan(1),xspan(2),N+2))';
xhi=xh(2:end-1);
un = u0(xhi,varargin{:});
fn=quad_mpt(f,xh,tspan(1));
R=chol(A);

```

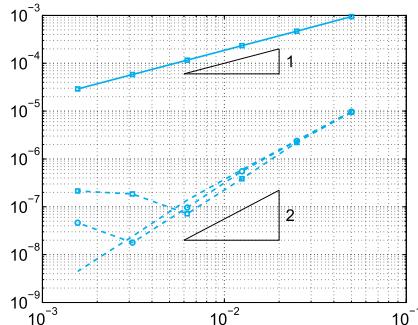


Figura 9.12. L'errore al variare di Δt per il θ -metodo e approssimazione agli elementi finiti in spazio (per $\theta = 1$ in linea continua, e $\theta = 0.5$ in linea tratteggiata), per tre differenti valori di h : 0.008 (\square), 0.004 (\circ) e 0.002 (nessun simbolo)

```

gdn=gd([xspan(1);xspan(2)],tspan(1),varargin{:});
for t = tspan(1)+dt:dt:tspan(2)
    fn1=quad_mpt(f,xh,t);
    b = An*un+theta*fn1+theta1*fn;
    gdn1 = gd([xspan(1);xspan(2)],t,varargin{:});
    b([1,end]) = b([1,end])+ad*gdn1+adn*gdn;
    u = R'\b; u = R\u; fn = fn1; un = u;
    gdn=gdn1;
end
uh=[gdn1(1);u;gdn1(end)];
return
function [fn]=quad_mpt(f,xh,t)
N=length(xh)-2; h=xh(2)-xh(1);
fn=zeros(N,1);
for j=1:N
    fn(j)=h/2*(f((xh(j)+xh(j+1))/2,t)+...
        f((xh(j+1)+xh(j+2))/2,t));
end
return

```

Esempio 9.7 Riprendiamo il problema dell'esempio 9.5 e risolviamolo stavolta con il metodo degli elementi finiti. In Figura 9.12 confrontiamo il comportamento degli errori $\max_{i=0,\dots,N} |u(x_i, 1) - u_i^M|$ rispetto al passo temporale su una griglia uniforme lungo la direzione spaziale con tre diversi valori di h . I valori $\{u_i^M\}$ rappresentano la soluzione agli elementi finiti calcolata al tempo $t^M = 1$. ■

9.9 Equazioni iperboliche: un problema di trasporto scalare

Consideriamo il seguente problema iperbolico scalare

$$\begin{cases} \frac{\partial u}{\partial t} + a \frac{\partial u}{\partial x} = 0, & x \in \mathbb{R}, t > 0, \\ u(x, 0) = u^0(x), & x \in \mathbb{R}, \end{cases} \quad (9.75)$$

dove a è un numero reale positivo. La sua soluzione è data da

$$u(x, t) = u^0(x - at), \quad t \geq 0,$$

che rappresenta un'onda viaggiante che si propaga con velocità pari ad a . Le curve $(x(t), t)$ nel piano (x, t) , che soddisfano la seguente equazione differenziale ordinaria scalare

$$\begin{cases} \frac{dx}{dt}(t) = a, & t > 0, \\ x(0) = x_0, \end{cases} \quad (9.76)$$

sono chiamate *curve caratteristiche* (o, semplicemente, *caratteristiche*) e sono le rette $x(t) = x_0 + at$, $t > 0$. La soluzione di (9.75) si mantiene costante lungo di esse in quanto

$$\frac{du}{dt} = \frac{\partial u}{\partial t} + \frac{\partial u}{\partial x} \frac{dx}{dt} = 0 \quad \text{su } (x(t), t).$$

Se si considera il problema più generale

$$\begin{cases} \frac{\partial u}{\partial t} + a \frac{\partial u}{\partial x} + a_0 u = f, & x \in \mathbb{R}, t > 0, \\ u(x, 0) = u^0(x), & x \in \mathbb{R}, \end{cases} \quad (9.77)$$

dove a , a_0 e f sono funzioni assegnate delle variabili (x, t) , le curve caratteristiche sono ancora definite come nella (9.76). In questo caso, le soluzioni di (9.77) soddisfano lungo le caratteristiche la seguente equazione differenziale ordinaria

$$\frac{du}{dt} = f - a_0 u \quad \text{su } (x(t), t).$$

Consideriamo ora il problema (9.75) su di un intervallo limitato $[\alpha, \beta]$

$$\begin{cases} \frac{\partial u}{\partial t} + a \frac{\partial u}{\partial x} = 0, & x \in (\alpha, \beta), t > 0, \\ u(x, 0) = u^0(x), & x \in (\alpha, \beta). \end{cases} \quad (9.78)$$

Consideriamo dapprima $a > 0$. Poiché u è costante lungo le caratteristiche, dalla Figura 9.13 (a sinistra) si deduce che il valore della soluzione in P è uguale al valore di u^0 in P_0 , il piede della caratteristica spiccata da P . D'altro canto, la caratteristica spiccata da Q interseca la retta $x(t) = \alpha$ ad un certo istante $t = \bar{t} > 0$. Conseguentemente, il punto $x = \alpha$ è detto di *inflow* (mentre $x = \beta$ è detto di *outflow*) ed è necessario assegnare un valore al contorno per u in $x = \alpha$ per ogni $t > 0$. Si noti che se fosse $a < 0$ allora il punto di inflow sarebbe $x = \beta$ e sarebbe necessario assegnare lì una condizione al bordo per u , per ogni $t > 0$.

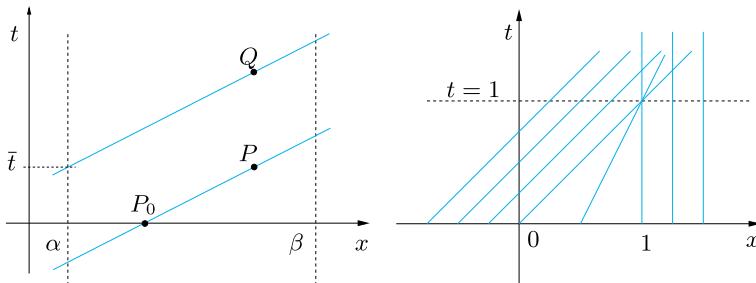


Figura 9.13. Esempi di caratteristiche che sono rette spiccate dai punti P e Q (a sinistra). Rette caratteristiche per l'equazione di Burgers (9.79) (a destra)

Con riferimento al problema (9.75) è bene osservare che se il dato u^0 è discontinuo in un punto x_0 , allora tale discontinuità si propaga lungo la caratteristica spiccata da x_0 . Questo processo può essere rigorosamente formalizzato introducendo il concetto di *soluzione debole* di un problema iperbolico (si veda, ad esempio, [GR96]). Un'altra ragione per introdurre le soluzioni deboli è che nel caso di problemi iperbolici non lineari le caratteristiche possono intersecarsi. In tale circostanza la soluzione non può essere continua e pertanto il problema non ammette alcuna soluzione in senso classico.

Esempio 9.8 (Equazione di Burgers) Consideriamo l'equazione di Burgers

$$\frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} = 0, \quad x \in \mathbb{R}, \quad t > 0, \quad (9.79)$$

che rappresenta il più semplice esempio di equazione iperbolica non lineare. Prendendo come condizione iniziale

$$u(x, 0) = u^0(x) = \begin{cases} 1, & x \leq 0, \\ 1 - x, & 0 < x \leq 1, \\ 0, & x > 1, \end{cases}$$

la linea caratteristica spiccata dal punto $(x_0, 0)$ è data da

$$x(t) = x_0 + tu^0(x_0) = \begin{cases} x_0 + t, & x_0 \leq 0, \\ x_0 + t(1 - x_0), & 0 < x_0 \leq 1, \\ x_0, & x_0 > 1. \end{cases}$$

Si noti che le linee caratteristiche non si intersecano solo se $t < 1$ (si veda la Figura 9.13, a destra). ■

9.9.1 Metodi alle differenze finite per la discretizzazione dell'equazione scalare iperbolica

Discretizziamo il semipiano $\{(x, t) : -\infty < x < \infty, t > 0\}$ scegliendo un passo di griglia spaziale $h > 0$ un passo di griglia temporale $\Delta t > 0$

e i punti di griglia (x_j, t^n) come segue

$$x_j = jh, \quad j \in \mathbb{Z}, \quad t^n = n\Delta t, \quad n \in \mathbb{N}.$$

Poniamo

$$\lambda = \Delta t/h,$$

e definiamo $x_{j+1/2} = x_j + h/2$. Cerchiamo soluzioni discrete u_j^n che forniscano una approssimazione ai valori $u(x_j, t^n)$ della soluzione esatta per ogni j e n . Per avanzare in tempo problemi ai valori iniziali di tipo iperbolico si ricorre spesso a metodi di tipo esplicito.

Ogni metodo alle differenze finite di tipo esplicito può essere scritto nella forma

$$u_j^{n+1} = u_j^n - \lambda(H_{j+1/2}^n - H_{j-1/2}^n), \quad (9.80)$$

dove $H_{j+1/2}^n = H(u_j^n, u_{j+1}^n)$ per ogni j e $H(\cdot, \cdot)$ è una funzione, da scegliersi in modo opportuno, detta *flusso numerico*.

Illustriamo nel seguito diversi esempi di metodi esplicativi per l'approssimazione del problema (9.75):

1. Eulero in avanti/centrato

$$u_j^{n+1} = u_j^n - \frac{\lambda}{2}a(u_{j+1}^n - u_{j-1}^n) \quad (9.81)$$

che può essere scritto nella forma (9.80) ponendo

$$H_{j+1/2}^n = \frac{1}{2}a(u_{j+1}^n + u_j^n); \quad (9.82)$$

2. Lax-Friedrichs

$$u_j^{n+1} = \frac{1}{2}(u_{j+1}^n + u_{j-1}^n) - \frac{\lambda}{2}a(u_{j+1}^n - u_{j-1}^n) \quad (9.83)$$

che risulta della forma (9.80) ponendo

$$H_{j+1/2}^n = \frac{1}{2}[a(u_{j+1}^n + u_j^n) - \lambda^{-1}(u_{j+1}^n - u_j^n)]; \quad (9.84)$$

3. Lax-Wendroff

$$u_j^{n+1} = u_j^n - \frac{\lambda}{2}a(u_{j+1}^n - u_{j-1}^n) + \frac{\lambda^2}{2}a^2(u_{j+1}^n - 2u_j^n + u_{j-1}^n) \quad (9.85)$$

che può essere scritto nella forma (9.80) con la scelta

$$H_{j+1/2}^n = \frac{1}{2}[a(u_{j+1}^n + u_j^n) - \lambda a^2(u_{j+1}^n - u_j^n)]; \quad (9.86)$$

Tabella 9.1. Coefficienti di viscosità artificiale, flusso di diffusione artificiale ed errore di troncamento per i metodi di Lax-Friedrichs, Lax-Wendroff e *upwind*

metodo	k	$H_{j+1/2}^{diff}$	$\tau(\Delta t, h)$
Lax-Friedrichs	h^2	$-\frac{1}{2\lambda}(u_{j+1} - u_j)$	$\mathcal{O}(h^2/\Delta t + \Delta t + h^2)$
Lax-Wendroff	$a^2 \Delta t^2$	$-\frac{\lambda a^2}{2}(u_{j+1} - u_j)$	$\mathcal{O}(\Delta t^2 + h^2 + \Delta t h^2)$
<i>upwind</i>	$ a h \Delta t$	$-\frac{ a }{2}(u_{j+1} - u_j)$	$\mathcal{O}(\Delta t + h)$

4. *Upwind* (o Eulero in avanti/centrato)

$$u_j^{n+1} = u_j^n - \frac{\lambda}{2} a(u_{j+1}^n - u_{j-1}^n) + \frac{\lambda}{2} |a|(u_{j+1}^n - 2u_j^n + u_{j-1}^n) \quad (9.87)$$

che risulta della forma (9.80) se il flusso numerico è definito come

$$H_{j+1/2}^n = \frac{1}{2}[a(u_{j+1}^n + u_j^n) - |a|(u_{j+1}^n - u_j^n)]. \quad (9.88)$$

Ognuno di questi ultimi tre metodi si può ricavare a partire dal metodo di Eulero in avanti/centrato aggiungendo un termine proporzionale alla differenza finita centrata (4.11), in modo tale da poterli scrivere nella forma equivalente

$$u_j^{n+1} = u_j^n - \frac{\lambda}{2} a(u_{j+1}^n - u_{j-1}^n) + \frac{1}{2} k \frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{(h)^2}. \quad (9.89)$$

L'ultimo termine esprime la discretizzazione della derivata seconda

$$\frac{k}{2} \frac{\partial^2 u}{\partial x^2}(x_j, t^n).$$

Il coefficiente $k > 0$ gioca il ruolo di coefficiente di viscosità artificiale. La sua espressione per i tre casi precedenti è riportata nella Tabella 9.1.

Come conseguenza, il flusso numerico di ciascuno schema si può scrivere in modo equivalente come

$$H_{j+1/2} = H_{j+1/2}^{EA} + H_{j+1/2}^{diff},$$

dove $H_{j+1/2}^{EA}$ è il flusso numerico relativo al metodo di Eulero in avanti/centrato (che è dato dalla (9.82)) mentre il *flusso di diffusione artificiale* $H_{j+1/2}^{diff}$ è riportato per i tre casi in Tabella 9.1.

L'esempio più classico di metodo implicito è il metodo di *Eulero all'indietro/centrato*

$$u_j^{n+1} + \frac{\lambda}{2} a(u_{j+1}^{n+1} - u_{j-1}^{n+1}) = u_j^n. \quad (9.90)$$

Anch'esso può essere scritto nella forma (9.80) a patto di sostituire H^n con H^{n+1} . Nel caso in esame, il flusso numerico è il medesimo del metodo di Eulero in avanti/centrato.

9.9.2 Analisi dei metodi alle differenze finite per l'equazione scalare iperbolica

L'analisi della convergenza dei metodi alle differenze finite introdotti nella Sezione precedente richiede la verifica preliminare delle proprietà di consistenza e stabilità.

A titolo di esempio consideriamo il metodo di Eulero in avanti/centrato (9.81) e studiamone la consistenza. Come illustrato nella Sezione 8.4, indicando con u la soluzione del problema (9.75), l'*errore di troncamento locale* nel punto (x_j, t^n) rappresenta l'errore che si genererebbe forzando la soluzione esatta a soddisfare quello specifico schema numerico. In particolare, per lo schema di Eulero in avanti/centrato, esso è

$$\tau_j^n = \frac{u(x_j, t^{n+1}) - u(x_j, t^n)}{\Delta t} + a \frac{u(x_{j+1}, t^n) - u(x_{j-1}, t^n)}{2h},$$

mentre l'*errore di troncamento (globale)* è definito come

$$\tau(\Delta t, h) = \max_{j,n} |\tau_j^n|.$$

Quando accade che $\tau(\Delta t, h)$ tende a zero al tendere a zero di Δt e di h in modo indipendente, si dice che lo schema numerico è *consistente*. Più in generale, diciamo che uno schema è di *ordine p* in tempo e di *ordine q* in spazio (per opportuni valori positivi p e q) se, per una soluzione esatta sufficientemente regolare, si ha

$$\tau(\Delta t, h) = \mathcal{O}(\Delta t^p + h^q).$$

Infine, diciamo che uno schema numerico è *convergente* (rispetto alla norma del massimo) se

$$\lim_{\Delta t, h \rightarrow 0} \max_{j,n} |u(x_j, t^n) - u_j^n| = 0.$$

Se la soluzione esatta è abbastanza regolare, mediante un impiego opportuno dello sviluppo in serie di Taylor si può caratterizzare l'errore di troncamento dei metodi precedentemente introdotti come illustrato nella Tabella 9.1. Il metodo di Eulero in avanti (o all'indietro) centrato è di ordine $\mathcal{O}(\Delta t + h^2)$, per gli altri metodi si veda la Tabella 9.1.

Per quanto concerne lo studio della stabilità, diciamo che un metodo numerico per un problema iperbolico (lineare o non lineare) si dice *stabile* se, per ogni tempo T , si possono determinare due costanti $C_T > 0$ (eventualmente dipendente da T) e $\delta_0 > 0$ tali che

$$\|\mathbf{u}^n\|_{\Delta} \leq C_T \|\mathbf{u}^0\|_{\Delta}, \quad (9.91)$$

per ogni n tale che $n\Delta t \leq T$ e per ogni $\Delta t, h$ tali che $0 < \Delta t \leq \delta_0$, $0 < h \leq \delta_0$. Con il simbolo $\|\cdot\|_{\Delta}$ abbiamo indicato un'opportuna norma discreta, ad esempio una di quelle riportate qui di seguito:

$$\|\mathbf{v}\|_{\Delta,p} = \left(h \sum_{j=-\infty}^{\infty} |v_j|^p \right)^{\frac{1}{p}} \quad \text{per } p = 1, 2, \quad \|\mathbf{v}\|_{\Delta,\infty} = \sup_j |v_j|. \quad (9.92)$$

Courant, Friedrichs e Lewy [CFL28] hanno dimostrato che una condizione necessaria e sufficiente affinché uno schema esplicito della forma (9.80) sia stabile è che i passi di discretizzazione temporale e spaziale obbediscano alla seguente condizione

$$|a\lambda| \leq 1, \quad \text{ovvero } \Delta t \leq \frac{h}{|a|} \quad (9.93)$$

nota come *condizione CFL*. Il numero adimensionale $a\lambda$ (a rappresenta una velocità) è comunemente chiamato *numero CFL*. Se a non è costante la condizione CFL diventa

$$\Delta t \leq \frac{h}{\sup_{x \in \mathbb{R}, t > 0} |a(x, t)|}.$$

È possibile dimostrare che:

1. il metodo di *Eulero in avanti/centrato* (9.81) è incondizionatamente instabile ovvero è instabile per ogni scelta possibile dei parametri $h > 0$ e $\Delta t > 0$;
2. il metodo *upwind* (detto anche di *Eulero in avanti/decentrato*) (9.87) è condizionatamente stabile rispetto alla norma $\|\cdot\|_{\Delta,1}$ ovvero

$$\|\mathbf{u}^n\|_{\Delta,1} \leq \|\mathbf{u}^0\|_{\Delta,1} \quad \forall n \geq 0,$$

a patto che sia soddisfatta la condizione CFL (9.93); un risultato analogo può essere dimostrato anche per gli schemi di *Lax-Friedrichs* (9.83) e di *Lax-Wendroff* (9.85);

3. il metodo di *Eulero all'indietro/centrato* (9.90) è incondizionatamente stabile rispetto alla norma $\|\cdot\|_{\Delta,2}$, ovvero per ogni $\Delta t > 0$

$$\|\mathbf{u}^n\|_{\Delta,2} \leq \|\mathbf{u}^0\|_{\Delta,2} \quad \forall n \geq 0.$$

Si veda l'Esercizio 9.15.

Per una dimostrazione dei risultati appena descritti si vedano ad esempio [QSSG14, Cap. 12] e [Qua16, Cap. 13].

Come già osservato nel caso parabolico, la proprietà di stabilità di cui si discute qui è quella *assoluta*. In effetti la diseguaglianza sopra riportate sono indipendenti dall'ampiezza dell'intervallo temporale in esame.

Vogliamo ora accennare a due caratteristiche salienti di un metodo numerico, quelle di dissipazione e di dispersione. A questo fine, supponiamo che il dato iniziale $u^0(x)$ del problema (9.75) sia una funzione periodica con periodo 2π cosicché possiamo riscriverla come una serie di Fourier, ovvero

$$u^0(x) = \sum_{k=-\infty}^{\infty} \alpha_k e^{ikx},$$

dove

$$\alpha_k = \frac{1}{2\pi} \int_0^{2\pi} u^0(x) e^{-ikx} dx$$

è il k -simo coefficiente di Fourier. La soluzione esatta u del problema (9.75) soddisfa (formalmente) le condizioni nodali

$$u(x_j, t^n) = \sum_{k=-\infty}^{\infty} \alpha_k e^{ikjh} (g_k)^n, \quad j \in \mathbb{Z}, n \in \mathbb{N} \quad (9.94)$$

con $g_k = e^{-iak\Delta t}$, mentre la soluzione numerica u_j^n , ottenuta con uno qualsiasi degli schemi introdotti nella Sezione 9.9.1, assume la forma

$$u_j^n = \sum_{k=-\infty}^{\infty} \alpha_k e^{ikjh} (\gamma_k)^n, \quad j \in \mathbb{Z}, n \in \mathbb{N}. \quad (9.95)$$

L'espressione dei coefficienti $\gamma_k \in \mathbb{C}$ dipende dallo specifico schema numerico utilizzato; ad esempio per lo schema (9.81) si può dimostrare che $\gamma_k = 1 - a\lambda i \sin(kh)$.

Osserviamo che, mentre $|g_k| = 1$ per ogni $k \in \mathbb{Z}$, le quantità $|\gamma_k|$ dipendono dal fattore $a\lambda$ (il numero CFL introdotto in 9.93) e quindi dalla discretizzazione scelta. In particolare, scegliendo $\|\cdot\|_\Delta = \|\cdot\|_{\Delta,2}$, si può dimostrare che la condizione $|\gamma_k| \leq 1$, $\forall k \in \mathbb{Z}$ è necessaria e sufficiente a garantire la diseguaglianza di stabilità (9.91).

Il rapporto

$$\epsilon_a(k) = |\gamma_k| / |g_k| = |\gamma_k| \quad (9.96)$$

è detto *coefficiente di dissipazione* (o *di amplificazione*) della k -sima frequenza associata allo schema numerico. Ricordiamo che la soluzione esatta di (9.75) è l'onda viaggiante $u(x, t) = u^0(x - at)$ la cui ampiezza è indipendente dal tempo; per la sua approssimazione numerica (9.95) succederà che tanto più $\epsilon_a(k)$ è piccolo, tanto maggiore sarà la riduzione dell'ampiezza dell'onda, dunque, in definitiva, più grande sarà la dissipazione numerica. Risulta inoltre evidente che violare la condizione di stabilità comporta un aumento dell'ampiezza dell'onda e quindi un

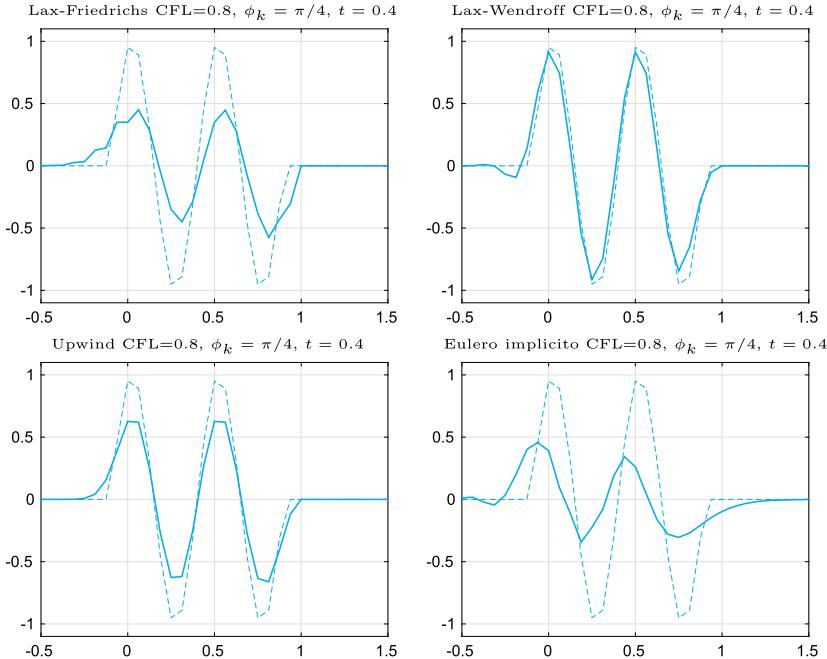


Figura 9.14. Soluzione esatta (linea tratteggiata) e soluzione numerica (linea continua), all’istante $t = 0.4$ e al variare di x , del problema (9.78) per $a = 1$ e dato iniziale definito in (9.98) con lunghezza d’onda $\ell = 1/2$

effetto di *blow-up* della soluzione numerica per tempi sufficientemente grandi.

Oltre ad un effetto dissipativo, gli schemi numerici introducono anche un effetto dispersivo ovvero un ritardo o un anticipo nella propagazione dell’onda. Per rendercene conto riscriviamo g_k e γ_k come segue:

$$g_k = e^{-ia\lambda\phi_k}, \quad \gamma_k = |\gamma_k|e^{-i\omega\Delta t} = |\gamma_k|e^{-i\frac{\omega}{k}\lambda\phi_k},$$

essendo $\phi_k = kh$ il cosiddetto *angolo di fase* associato alla frequenza k -sima.

Confrontando le due espressioni e ricordando che a rappresenta la velocità di propagazione dell’onda “esatta”, definiamo *coefficiente di dispersione* legato alla frequenza k -sima la quantità

$$\epsilon_d(k) = \frac{\omega}{ak} = \frac{\omega\Delta t}{\phi_k a \lambda}. \quad (9.97)$$

Nelle Figure 9.14 e 9.15 sono riportate la soluzione esatta del problema (9.78) con $a = 1$ e le soluzioni numeriche ottenute con alcuni degli

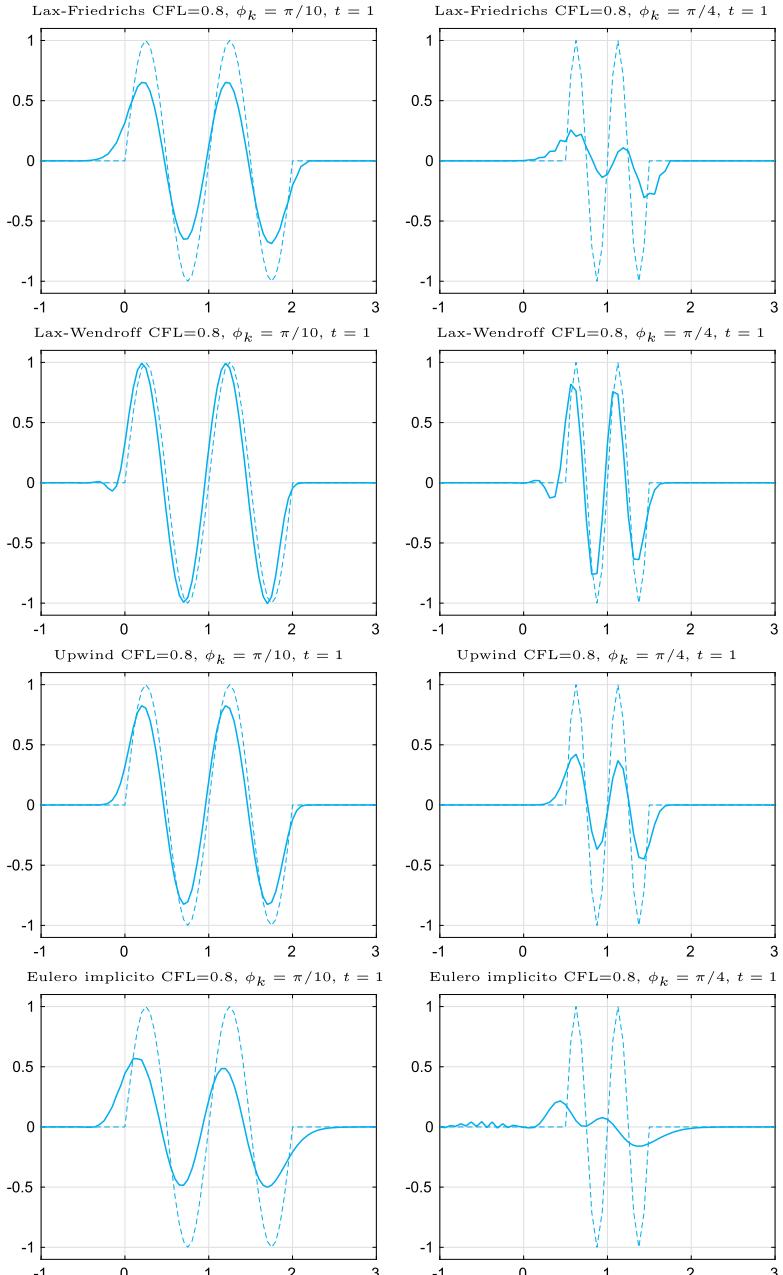


Figura 9.15. Soluzione esatta (in linea tratteggiata) e soluzione numerica (in linea continua), all’istante $t = 1$ e al variare di x , del problema (9.78) per $a = 1$ e dato iniziale definito in (9.98) con lunghezza d’onda $\ell = 1$ (grafici a sinistra) e $\ell = 1/2$ (grafici a destra)

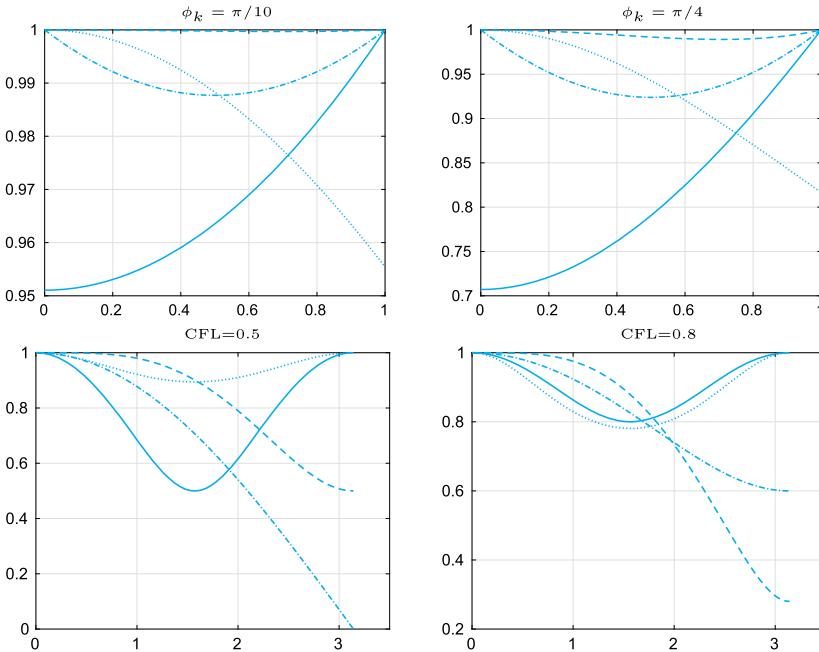


Figura 9.16. Coefficienti di dissipazione $\epsilon_a(k)$ per i metodi Lax-Friedrichs (*linea continua*), Lax-Wendroff (*linea tratteggiata*), Upwind (*linea tratto punto*) ed Eulero all'indietro/centrato (*linea punteggiata*), al variare del numero CFL (*in alto*) ed al variare di $\phi_k = kh$ (*in basso*)

schemi della sezione 9.9.1. Il dato iniziale è

$$u^0(x) = \begin{cases} \sin(2\pi x/\ell) & -1 \leq x \leq \ell \\ 0 & \ell < x < 3, \end{cases} \quad (9.98)$$

di lunghezza d'onda $\ell = 1$ (a sinistra) e $\ell = 1/2$ (a destra). In entrambi i casi è stato fissato numero CFL pari a 0.8. Per $\ell = 1$ è stato scelto $h = \ell/20 = 1/20$, in modo che $\phi_k = 2\pi h/\ell = \pi/10$ e $\Delta t = 1/25$. Per $\ell = 1/2$ è stato scelto $h = \ell/8 = 1/16$, in modo che $\phi_k = \pi/4$ e $\Delta t = 1/20$.

In Figura 9.16 ed in Figura 9.17 sono riportati rispettivamente i coefficienti di dissipazione e di dispersione al variare del numero CFL (grafici in alto) e dell'angolo di fase $\phi_k = kh$ (grafici in basso).

Osserviamo dalla Figura 9.16 che, in corrispondenza del numero CFL=0.8, lo schema meno dissipativo è quello di Lax-Wendroff, informazione che trova conferma nella rappresentazione delle soluzioni numeriche di Figura 9.15, sia per $\phi_k = \pi/10$ che per $\phi_k = \pi/4$. Per quanto riguarda il coefficiente di dispersione, sempre in corrispondenza del numero CFL=0.8, dalla Figura 9.17 emerge che *upwind* è lo schema

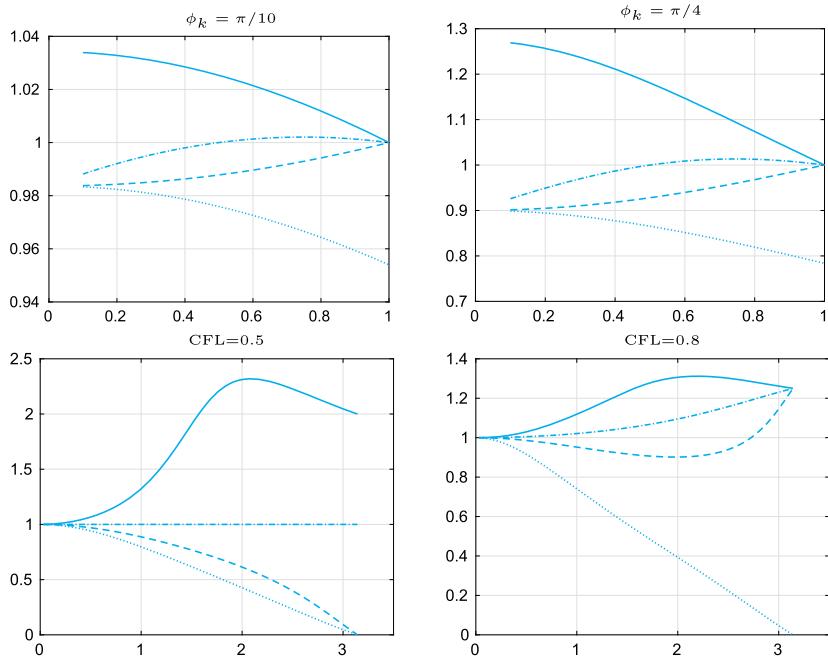


Figura 9.17. Coefficienti di dispersione $\epsilon_d(k)$ al variare del numero CFL (in alto) ed al variare di $\phi_k = kh$ (in basso). Si veda la legenda della Figura 9.16 per la descrizione dei tipi di linea

meno dispersivo con un leggero anticipo di fase, che lo schema di Lax-Friederichs ha un significativo anticipo di fase, mentre entrambi i metodi Lax-Wendroff e Eulero implicito/centrato presentano un ritardo di fase. Ciò è confermato dalle soluzioni numeriche riportate in Figura 9.14.

Si noti come il coefficiente di dissipazione è responsabile dell'abbattimento dell'ampiezza d'onda, quello di dispersione della sua inesatta velocità di propagazione.

9.9.3 Discretizzazione in spazio dell'equazione scalare iperbolica con elementi finiti

Per costruire un'approssimazione semi-discreta del problema (9.75) si può ricorrere al metodo di Galerkin (si veda la Sezione 9.5). Assumiamo che $a = a(x) > 0 \forall x \in [\alpha, \beta]$, cosicché il punto $x = \alpha$ sia il *bordo di inflow* e che il valore al contorno vada imposto in tale punto. Sia φ una opportuna funzione nota al variare di $t > 0$ e completiamo il sistema (9.75) con la condizione al bordo

$$u(\alpha, t) = \varphi(t), \quad t > 0. \quad (9.99)$$

Dopo aver definito gli spazi

$$\begin{aligned} V_h &= \{v_h \in C^0([\alpha, \beta]) : v_h|_{I_j} \in \mathbb{P}_1, j = 1, \dots, N\}, \\ V_h^{in} &= \{v_h \in V_h : v_h(\alpha) = 0\}, \end{aligned}$$

consideriamo la seguente approssimazione agli elementi finiti del problema (9.75), (9.99): per ogni $t \in (0, T)$ trovare $u_h(t) \in V_h$ tale che

$$\begin{cases} \int_\alpha^\beta \frac{\partial u_h(t)}{\partial t} v_h dx + \int_\alpha^\beta a \frac{\partial u_h(t)}{\partial x} v_h dx = 0 & \forall v_h \in V_h^{in}, \\ u_h(t) = \varphi(t) & \text{in } x = \alpha, \end{cases} \quad (9.100)$$

con $u_h(0) = u_h^0 \in V_h$ (u_h^0 essendo una opportuna approssimazione ad elementi finiti del dato iniziale u^0 , per esempio il suo interpolatore lineare composito).

Per la discretizzazione temporale di (9.100) si possono utilizzare ancora schemi alle differenze finite. Se, ad esempio, impieghiamo il metodo di Eulero all'indietro, per ogni $n \geq 0$, si ottiene: trovare $u_h^{n+1} \in V_h$ tale che

$$\frac{1}{\Delta t} \int_\alpha^\beta (u_h^{n+1} - u_h^n) v_h dx + \int_\alpha^\beta a \frac{\partial u_h^{n+1}}{\partial x} v_h dx = 0 \quad \forall v_h \in V_h^{in},$$

con $u_h^{n+1}(\alpha) = \varphi^{n+1}$.

Se $\varphi = 0$, possiamo concludere che

$$\|u_h^n\|_{L^2(\alpha, \beta)} \leq \|u_h^0\|_{L^2(\alpha, \beta)} \quad \forall n \geq 0,$$

ovvero lo schema di Eulero all'indietro è incondizionatamente stabile rispetto alla norma $\|v\|_{L^2(\alpha, \beta)} := (\int_\alpha^\beta v^2(x) dx)^{1/2}$.

Si vedano gli Esercizi 9.14–9.18.



9.10 L'equazione delle onde

Consideriamo ora la seguente equazione iperbolica del secondo ordine in una dimensione

$$\boxed{\frac{\partial^2 u}{\partial t^2} - c \frac{\partial^2 u}{\partial x^2} = f} \quad (9.101)$$

dove c è una costante positiva assegnata. Quando $f = 0$, la soluzione generale di (9.101) è la cosiddetta onda viaggiante di d'Alembert

$$u(x, t) = \psi_1(\sqrt{ct} - x) + \psi_2(\sqrt{ct} + x), \quad (9.102)$$

dove ψ_1 e ψ_2 sono funzioni arbitrarie.

Nel seguito considereremo il problema (9.101) per $x \in (a, b)$ e $t > 0$ e completeremo l'equazione differenziale con i dati iniziali

$$u(x, 0) = u_0(x) \text{ e } \frac{\partial u}{\partial t}(x, 0) = v_0(x), \quad x \in (a, b), \quad (9.103)$$

ed i dati al bordo

$$u(a, t) = 0 \text{ e } u(b, t) = 0, \quad t > 0. \quad (9.104)$$

Consideriamo il caso in cui u rappresenti lo spostamento trasversale di una corda elastica vibrante di lunghezza $b - a$, fissata alle estremità, e c sia un coefficiente positivo che dipende dalla massa specifica della corda e dalla sua tensione. La corda è soggetta ad una forza verticale di densità f . Le funzioni $u_0(x)$ e $v_0(x)$ rappresentano rispettivamente lo spostamento iniziale e la velocità iniziale dei punti della corda.

Il cambio di variabile

$$\omega_1 = \frac{\partial u}{\partial x}, \quad \omega_2 = \frac{\partial u}{\partial t},$$

trasforma (9.101) nel sistema del primo ordine

$$\frac{\partial \boldsymbol{\omega}}{\partial t} + A \frac{\partial \boldsymbol{\omega}}{\partial x} = \mathbf{f}, \quad x \in (a, b), \quad t > 0 \quad (9.105)$$

dove

$$\boldsymbol{\omega} = \begin{bmatrix} \omega_1 \\ \omega_2 \end{bmatrix}, \quad A = \begin{bmatrix} 0 & -1 \\ -c & 0 \end{bmatrix}, \quad \mathbf{f} = \begin{bmatrix} 0 \\ f \end{bmatrix},$$

e le condizioni iniziali sono $\omega_1(x, 0) = u'_0(x)$ e $\omega_2(x, 0) = v_0(x)$ per $x \in (a, b)$.

In generale, possiamo considerare sistemi del tipo (9.105) dove $\boldsymbol{\omega}, \mathbf{f} : \mathbb{R} \times [0, \infty) \rightarrow \mathbb{R}^p$ sono funzioni vettoriali assegnate e $A \in \mathbb{R}^{p \times p}$ è una matrice a coefficienti costanti. Il sistema è detto *iperbolico* se A è diagonalizzabile ed ha autovalori reali, cioè se esiste una matrice non singolare $T \in \mathbb{R}^{p \times p}$ tale che

$$A = T \Lambda T^{-1},$$

dove $\Lambda = \text{diag}(\lambda_1, \dots, \lambda_p)$ è la matrice diagonale degli autovalori reali di A , mentre $T = (\mathbf{v}^1, \mathbf{v}^2, \dots, \mathbf{v}^p)$ è la matrice i cui vettori colonna sono gli autovettori destri di A . Così

$$A \mathbf{v}^k = \lambda_k \mathbf{v}^k, \quad k = 1, \dots, p.$$

Introducendo le *variabili caratteristiche* $\mathbf{w} = T^{-1}\boldsymbol{\omega}$, il sistema (9.105) diventa

$$\frac{\partial \mathbf{w}}{\partial t} + \Lambda \frac{\partial \mathbf{w}}{\partial x} = \mathbf{g},$$

dove $\mathbf{g} = \mathbf{T}^{-1}\mathbf{f}$. Questo è un sistema di p equazioni scalari indipendenti della forma

$$\frac{\partial w_k}{\partial t} + \lambda_k \frac{\partial w_k}{\partial x} = g_k, \quad k = 1, \dots, p. \quad (9.106)$$

Quando $g_k = 0$, la sua soluzione è data da $w_k(x, t) = w_k(x - \lambda_k t, 0)$, $k = 1, \dots, p$ e la soluzione $\boldsymbol{\omega} = \mathbf{T}\mathbf{w}$ del problema (9.105) con $\mathbf{f} = \mathbf{0}$ può essere scritta come

$$\boldsymbol{\omega}(x, t) = \sum_{k=1}^p w_k(x - \lambda_k t, 0) \mathbf{v}^k.$$

La curva $(x_k(t), t)$ nel piano (x, t) che soddisfa $x'_k(t) = \lambda_k$ è la k -sima curva caratteristica (si veda la Sezione 9.9) e w_k è costante lungo di essa. Quindi $\boldsymbol{\omega}(\bar{x}, \bar{t})$ dipende solo dal dato iniziale nei punti $\bar{x} - \lambda_k \bar{t}$.

Per questa ragione l'insieme dei p punti che formano i piedi delle caratteristiche uscenti dal punto (\bar{x}, \bar{t}) ,

$$D(\bar{t}, \bar{x}) = \{x \in \mathbb{R} : x = \bar{x} - \lambda_k \bar{t}, k = 1, \dots, p\}, \quad (9.107)$$

è detto *dominio di dipendenza* della soluzione $\boldsymbol{\omega}(\bar{x}, \bar{t})$.

Se (9.105) è assegnato sull'intervallo limitato (a, b) invece che su tutto l'asse reale, il punto di *inflow* per ogni variabile caratteristica w_k è determinato dal segno di λ_k . Di conseguenza, il numero di autovalori positivi determina il numero di condizioni al bordo che possono essere assegnate in $x = a$, mentre in $x = b$ bisogna assegnare un numero di condizioni pari al numero di autovalori negativi.

Esempio 9.9 Il sistema (9.105) è iperbolico perché \mathbf{A} è diagonalizzabile mediante la matrice

$$\mathbf{T} = \begin{bmatrix} -\frac{1}{\sqrt{c}} & \frac{1}{\sqrt{c}} \\ 1 & 1 \end{bmatrix}$$

e presenta due autovalori reali distinti $\pm\sqrt{c}$ (rappresentanti le velocità di propagazione dell'onda). Inoltre, deve essere prescritta una condizione al bordo in ognuno dei due punti del bordo, come in (9.104). ■

9.10.1 Discretizzazione dell'equazione delle onde

A seconda che si consideri l'equazione delle onde del secondo ordine (9.101) o il sistema equivalente del primo ordine (9.105), possiamo utilizzare schemi di discretizzazione diversi.

Per discretizzare in tempo l'equazione delle onde nella forma (9.101) possiamo utilizzare il metodo di Newmark già proposto nel Capitolo 8 per le equazioni differenziali ordinarie del secondo ordine, oppure il metodo *leap-frog*. Denotando ancora con Δt il passo temporale (uniforme) ed utilizzando per la discretizzazione in spazio il metodo classico delle differenze finite su una griglia di nodi $x_j = x_0 + jh$, $j = 0, \dots, N+1$, $x_0 = a$ e $x_{N+1} = b$, lo schema di Newmark per (9.101) è: per ogni $n \geq 1$ si cercano $\{u_j^n, v_j^n, j = 1, \dots, N\}$ tali che

$$\begin{aligned} u_j^{n+1} &= u_j^n + \Delta t v_j^n \\ &\quad + \Delta t^2 [\zeta(cw_j^{n+1} + f(x_j, t^{n+1})) + (1/2 - \zeta)(cw_j^n + f(x_j, t^n))], \\ v_j^{n+1} &= v_j^n + \Delta t [(1 - \theta)(cw_j^n + f(x_j, t^n)) + \theta(cw_j^{n+1} + f(x_j, t^{n+1}))], \end{aligned} \quad (9.108)$$

con $u_j^0 = u_0(x_j)$, $v_j^0 = v_0(x_j)$ e $w_j^k = (u_{j+1}^k - 2u_j^k + u_{j-1}^k)/(h)^2$ per $k = n$ o $k = n + 1$. Il sistema (9.108) deve essere completato imponendo le condizioni al bordo (9.104).

Il metodo di Newmark è implementato nel Programma 9.6. I parametri di input sono i vettori `xspan=[a, b]` e `tspan=[0, T]`, il numero di intervalli della discretizzazione in spazio (`nstep(1)`) ed in tempo (`nstep(2)`), lo scalare `c` (corrispondente alla costante positiva c), i *function handle* `u0` e `v0` per definire i dati iniziali $u_0(x)$ e $v_0(x)$, rispettivamente, ed i *function handle* `gd` e `fun` che contengono le funzioni $g_D(x, t)$ e $f(x, t)$, rispettivamente. Infine, il vettore `param` permette di specificare i coefficienti (`param(1)=zeta`, `param(2)=theta`). Ricordiamo che il metodo di Newmark è accurato di ordine 2 rispetto a Δt se $\theta = 1/2$, altrimenti esso è solo accurato di ordine 1 se $\theta \neq 1/2$, e che la condizione $\theta \geq 1/2$ garantisce stabilità allo schema (si veda la Sezione 8.10).

Programma 9.6. newmarkwave: metodo di Newmark per l'equazione delle onde

```
function [xh,uh]=newmarkwave(xspan,tspan,nstep,param,...  
    c,u0,v0,gd,f,varargin)  
% NEWMARKWAVE risolve l'equazione delle onde  
%     con il metodo di Newmark  
%     [XH,UH]=NEWMARKWAVE(XSPAN,TSPAN,NSTEP,PARAM,C,...  
%     U0,VO,GD,F)  
%     risolve l'equazione delle onde  
%     D^2 U/DT^2 - C D^2U/DX^2 = F in  
%     (XSPAN(1),XSPAN(2)) X (TSPAN(1),TSPAN(2)) con il  
%     metodo di Newmark, con condizioni iniziali  
%     U(X,0)=U0(X), DU/DX(X,0)=VO(X) e condizioni al  
%     bordo di Dirichlet U(X,T)=GD(X,T) in X=XSPAN(1)  
%     ed in X=XSPAN(2). C e' una costante positiva.  
%     NSTEP(1) e' il numero di intervalli di spazio.  
%     NSTEP(2) e' il numero di intervalli in tempo.  
%     PARAM(1)=ZETA e PARAM(2)=THETA.
```

```

% U0(X), VO(X), GD(X,T) e F(X,T) possono essere defi-
% nite come function handle o user defined function.
% XH contiene i nodi della discretizzazione in spazio
% UH contiene la soluzione numerica al tempo TSPAN(2).
% [XH,UH]=NEWMARKWAVE(XSPAN,TSPAN,NSTEP,PARAM,C, ...
% U0,VO,GD,F,P1,P2,...) passa i parametri addizio-
% nali P1,P2,... alle funzioni U0,VO,GD,F.
h = (xspan(2)-xspan(1))/nstep(1);
dt = (tspan(2)-tspan(1))/nstep(2);
zeta = param(1); theta = param(2);
N = nstep(1)+1; e = ones(N,1);
Afd = spdiags([e -2*e e],[-1,0,1],N,N)/h^2;
I = speye(N);
A = I-c*dt^2*zeta*Afd;
An = I+c*dt^2*(0.5-zeta)*Afd;
A(1,:) = 0; A(1,1) = 1;
A(N,:) = 0; A(N,N) = 1;
xh = (linspace(xspan(1),xspan(2),N))';
fn = f(xh,tspan(1),varargin{:});
un = u0(xh,varargin{:});
vn = v0(xh,varargin{:});
[L,U]=lu(A);
alpha = dt^2*zeta;
beta = dt^2*(0.5-zeta);
theta1 = 1-theta;
for t = tspan(1)+dt:dt:tspan(2)
    fn1 = f(xh,t,varargin{:});
    rhs = An*un+dt*I*vn+alpha*fn1+beta*fn;
    temp = gd([xspan(1),xspan(2)],t,varargin{:});
    rhs([1,N]) = temp;
    uh = L\rhs; uh = U\uh;
    v = vn + dt*((1-theta)*(c*Afd*un+fn)+...
        theta*(c*Afd*uh+fn1));
    fn = fn1; un = uh; vn = v;
end

```

Il metodo *leap-frog* applicato all'equazione (9.101),

$$u_j^{n+1} - 2u_j^n + u_j^{n-1} = c \left(\frac{\Delta t}{h} \right)^2 (u_{j+1}^n - 2u_j^n + u_{j-1}^n) \quad (9.109)$$

è ottenuto discretizzando sia la derivata temporale sia la derivata spaziale con la differenza finita centrata (9.17).

Si può dimostrare che entrambi i metodi di Newmark (9.108) e *leap-frog* (9.109) sono accurati al second'ordine sia rispetto a Δt che a h . Riguardo alla stabilità, si ha che il metodo *leap-frog* è stabile sotto la condizione CFL $\Delta t \leq h/\sqrt{c}$, mentre il metodo di Newmark è incondizionatamente stabile se $2\zeta \geq \theta \geq \frac{1}{2}$ (si veda [Joh90]).

Esempio 9.10 Utilizzando il Programma 9.6 studiamo l'evoluzione della condizione iniziale $u_0(x) = e^{-10x^2}$ per $x \in (-2, 2)$, prendendo $f = 0$ e $c = 1$ nell'equazione (9.101). Assumiamo che $v_0 = 0$ e consideriamo condizioni al bordo di Dirichlet omogenee. In Figura 9.18 confrontiamo le soluzioni ottenute al tempo $t = 3$ con $h = 0.04$ e passi temporali $\Delta t = 0.15$ (linea tratteggiata),

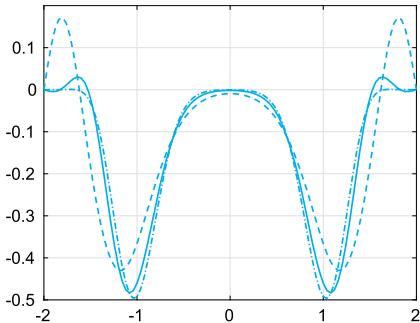


Figura 9.18. Confronto tra le soluzioni ottenute con il metodo di Newmark per una discretizzazione con $h = 0.04$ e $\Delta t = 0.15$ (linea tratteggiata), $\Delta t = 0.075$ (linea continua) e $\Delta t = 0.0375$ (linea tratto-punto)

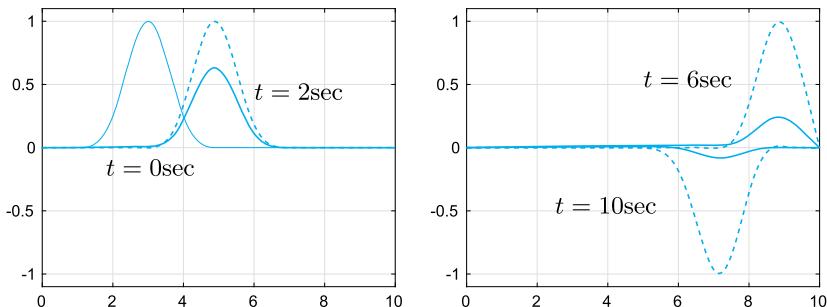


Figura 9.19. Propagazione di un impulso di potenziale ottenuto risolvendo l'equazione delle onde (linea tratteggiata) e l'equazione del telegrafo (linea continua). A sinistra, la linea continua sottile descrive la condizione iniziale $u_0(x)$

$\Delta t = 0.075$ (linea continua) e $\Delta t = 0.0375$ (linea tratto-punto). I parametri del metodo di Newmark sono $\theta = 1/2$ e $\zeta = 0.25$ e garantiscono accuratezza del secondo ordine e stabilità incondizionata al metodo. ■

Esempio 9.11 (Elettrotecnica) In questo esempio consideriamo l'equazione (9.11) per modellare come il cavo di un telegrafo trasmette un impulso di potenziale. L'equazione è una combinazione di un'equazione di diffusione e di un'equazione delle onde e tiene conto degli effetti della velocità finita in una equazione standard del trasporto della massa. Prendiamo come condizione iniziale un impulso, precisamente una B-spline cubica (si veda [QSSG14, Cap. 7]) centrata in $x = 3$ e non nulla nell'intervallo $(1,5)$ e studiamone l'evoluzione sull'intervallo temporale $(0,10)$ fissando $c = 1$, $\alpha = 0.5$ e $\beta = 0.04$. In Figura 9.19 confrontiamo la soluzione ottenuta risolvendo l'equazione delle onde (9.101) (in linea tratteggiata) e quella ottenuta risolvendo l'equazione del telegrafo (9.11) (in linea continua). La velocità iniziale scelta è $v_0(x) = -cu'_0(x)$ per l'equazione delle onde e $v_0(x) = -cu'_0(x) - \alpha/2u_0(x)$ per l'equazione del telegrafo, cosicché l'onda viaggia con velocità c in entrambi i casi. Sia l'equazione delle onde sia quella del telegrafo sono state risolte utilizzando lo schema

di Newmark con $h = 0.025$, $\Delta t = 0.1$, $\zeta = 1/4$ e $\theta = 1/2$. Per approssimare l'equazione delle onde abbiamo utilizzato il Programma 9.6, mentre per risolvere l'equazione del telegrafo abbiamo utilizzato una sua variante. La presenza di un effetto di dissipazione è evidente nella soluzione dell'equazione del telegrafo. ■

Come anticipato, un approccio alternativo ai metodi di Newmark e *leap-frog* consiste nel discretizzare il sistema equivalente del primo ordine (9.105). Consideriamo per semplicità il caso $\mathbf{f} = \mathbf{0}$; possiamo generalizzare al caso del sistema iperbolico (9.105) il metodo di Lax-Wendroff e quello *upwind* rispettivamente, come segue:

1. *metodo di Lax-Wendroff*

$$\begin{aligned}\omega_j^{n+1} &= \omega_j^n - \frac{\lambda}{2} A(\omega_{j+1}^n - \omega_{j-1}^n) \\ &\quad + \frac{\lambda^2}{2} A^2(\omega_{j+1}^n - 2\omega_j^n + \omega_{j-1}^n);\end{aligned}\tag{9.110}$$

2. *metodo upwind (o Eulero in avanti/decentrato)*

$$\begin{aligned}\omega_j^{n+1} &= \omega_j^n - \frac{\lambda}{2} A(\omega_{j+1}^n - \omega_{j-1}^n) \\ &\quad + \frac{\lambda}{2} |A|(\omega_{j+1}^n - 2\omega_j^n + \omega_{j-1}^n),\end{aligned}\tag{9.111}$$

avendo posto $|A| = T|A|T^{-1}$ ed essendo $|A|$ la matrice diagonale dei moduli degli autovalori di A .

Lo schema di Lax-Wendroff è accurato al secondo ordine (sia in tempo che in spazio), mentre il metodo *upwind* è accurato al primo ordine.

Riguardo alla stabilità, valgono le considerazioni svolte nella Sezione 9.9.1, a patto di generalizzare la condizione CFL (9.93) come segue:

$$\Delta t < \frac{h}{\rho(A)},\tag{9.112}$$

dove, come al solito, $\rho(A)$ denota il raggio spettrale della matrice A . Per la dimostrazione di questi risultati si vedano ad esempio [QV94], [LeV02], [GR96], [QSS07, Cap. 13].

Riassumendo

1. I problemi ai limiti in una dimensione sono problemi differenziali assegnati in un intervallo con condizioni imposte sulla soluzione (o sulla sua derivata) agli estremi dell'intervallo;
2. l'approssimazione numerica può essere basata sul metodo delle differenze finite (ottenute da sviluppi di Taylor troncati) o su quello degli elementi finiti (ottenuti da una formulazione integrale del problema in cui la soluzione e la funzione test sono polinomi a tratti);

3. gli stessi principi si possono applicare nel caso di problemi multidimensionali. Nel caso di problemi ai limiti bidimensionali, le approssimazioni agli elementi finiti utilizzano funzioni polinomiali “a tratti”, con cui si intendono i triangoli o i quadrilateri di una griglia che realizzzi una partizione della regione spaziale;
4. le matrici associate a discretizzazioni agli elementi finiti ed alle differenze finite sono sparse e mal condizionate;
5. i problemi ai valori iniziali e ai limiti sono problemi ai limiti in cui vi siano anche derivate temporali della soluzione. Queste ultime sono discretizzate con formule alle differenze finite, esplicite o implicite;
6. nel caso esplicito si trovano condizioni di stabilità che si esprimono attraverso una limitazione del passo di discretizzazione temporale in funzione di quello spaziale. Nel caso隐式 si ottiene ad ogni passo un sistema algebrico simile a quello che si ottiene nel caso del corrispondente problema ai limiti stazionario;
7. in questo capitolo abbiamo considerato semplici problemi lineari di tipo ellittico, parabolico e iperbolico. Per una trattazione più completa rinviamo il lettore alla bibliografia citata nella prossima Sezione 9.11.

9.11 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 [Qua16], [QV94], [EEHJ96] e [TW98]).

È bene comunque segnalare che il metodo agli elementi finiti è quello probabilmente più diffuso per la risoluzione di problemi differenziali (si vedano, ad esempio, [QV94], [Bra97], [BS01]). Come già notato il *toolbox* **bim** pde di MATLAB ed i *package* **fem-fenics** e **bim** di Octave consentono di risolvere una famiglia abbastanza grande di equazioni alle derivate parziali con gli elementi finiti, in particolare per la discretizzazione delle variabili spaziali.

Altre tecniche molto diffuse sono i metodi spettrali (si vedano a tal proposito [CHQZ06], [CHQZ07] [Fun92], [BM92], [KS99]) ed il metodo dei volumi finiti (si vedano [Krö98], [Hir88] e [LeV02]).

9.12 Esercizi

Esercizio 9.1 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 9.2 Si verifichi che la condizione di compatibilità (9.13) è una condizione necessaria affinché il problema di Neumann in una dimensione ammetta soluzioni. Analogamente, si mostri che la condizione di compatibilità (9.16) è necessaria per l'esistenza di soluzioni del problema di Neumann in due dimensioni.

Esercizio 9.3 Si verifichi che la matrice A_{fd} definita in (9.20) è simmetrica e definita positiva.

Esercizio 9.4 Si verifichi che la matrice A_{fd} definita in (9.20) ha autovalori pari a

$$\lambda_j = \frac{2}{h^2}(1 - \cos(j\theta)), \quad j = 1, \dots, N,$$

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 $K(A_{fd})$ è proporzionale a h^{-2} .

Esercizio 9.5 Si dimostri che, se $u \in C^4(I(\bar{x}))$, la quantità (9.17) fornisce una approssimazione di $u''(\bar{x})$ di ordine 2 rispetto a h , ovvero

$$\delta^2 u(\bar{x}) - u''(\bar{x}) = \mathcal{O}(h^2) \quad \text{quando } h \rightarrow 0. \quad (9.113)$$

Esercizio 9.6 Si ricavino matrice e termine noto relativi allo schema (9.23) per la risoluzione del problema (9.22).

Esercizio 9.7 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 9.8 Si scriva un programma `MATGOCT` per risolvere con le differenze finite il problema (9.22) nel caso in cui si considerino le condizioni di Neumann

$$u'(a) = \gamma, \quad u'(b) = \delta.$$

Si usino le formule (4.13) accurate di ordine 2 per approssimare $u'(a)$ e $u'(b)$. Con tale programma si risolva il problema (9.22) in $(a, b) = (0, 3/4)$, con $f(x) = ((2\pi)^2 + 0.1) \cos(2\pi x)$, $\alpha = 0$, $\beta = 2\pi$, $\mu = 1$, $\eta = 0$, $\sigma = 0.1$. Sapendo che la soluzione esatta di questo problema è $u(x) = \cos(2\pi x)$, si verifichi che la soluzione numerica converge con ordine 2 rispetto a h .

Esercizio 9.9 Si risolva il problema (9.34) sull’intervallo $(0, 1)$, con $\mu = 1/1000$, $\eta = 1$, $f(x) = 1$, $\alpha = 0$ e $\beta = 0$, prima utilizzando il Programma 9.1 `bvp_df_dir_1d.m` che implementa lo schema centrato (9.36) e poi scrivendo e utilizzando un nuovo programma `MAT&OCT` che implementi lo schema *upwind* (9.39).

Esercizio 9.10 Si verifichi che per una griglia uniforme, il termine noto del sistema (9.19) associato allo schema alle differenze finite coincide, a meno di un fattore h , con quello dello schema agli elementi finiti (9.45) quando in quest’ultimo si usi la formula composita del trapezio per approssimare gli integrali sugli elementi I_{j-1} and I_j .

Esercizio 9.11 Sia $f \in C^2([a, b])$; si calcolino gli integrali che compaiono nei termini di destra del sistema (9.47), approssimando f con l’interpolatore composito lineare di Lagrange $\Pi_1^h f$ (si veda la Sezione 3.4) che interpola f nei nodi x_j della discretizzazione e poi integrando in maniera esatta le funzioni $(\Pi_1^h f)\varphi_j$ per $j = 1, \dots, N$.

Esercizio 9.12 Si scriva una *function* `MAT&OCT` per calcolare l’errore nella norma integrale (9.50) tra la soluzione elementi finiti di grado 1 ed una funzione nota (la soluzione esatta). A tale propostio si utilizzi la formula di quadratura composita di Gauss-Legendre con $n = 4$ descritta nella Sezione 4.4.

Esercizio 9.13 Si consideri una piastra bidimensionale quadrata di lato 100 cm e di condutività 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.

Esercizio 9.14 Si verifichi che la soluzione del problema (9.101), (9.103)–(9.104) (con $f = 0$) soddisfa l’identità

$$\begin{aligned} & \int_a^b (u_t(x, t))^2 dx + c \int_a^b (u_x(x, t))^2 dx \\ &= \int_a^b (v_0(x))^2 dx + c \int_a^b (u_{0,x}(x))^2 dx, \end{aligned} \quad (9.114)$$

prendendo $u_0(a) = u_0(b) = 0$.

Esercizio 9.15 Si dimostri che la soluzione fornita dallo schema (9.90) di Eulero all’indietro/centrato è incondizionatamente stabile, ovvero $\forall \Delta t > 0$,

$$\|\mathbf{u}^n\|_{\Delta, 2} \leq \|\mathbf{u}^0\|_{\Delta, 2} \quad \forall n \geq 0. \quad (9.115)$$

Esercizio 9.16 Si dimostri che la soluzione fornita dallo schema *upwind* (9.87) soddisfa la stima

$$\|\mathbf{u}^n\|_{\Delta, \infty} \leq \|\mathbf{u}^0\|_{\Delta, \infty} \quad \forall n \geq 0 \quad (9.116)$$

a condizione che venga verificata la condizione CFL. La relazione (9.116) è detta *principio del massimo discreto*.

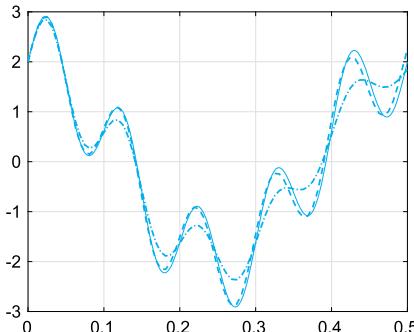


Figura 9.20. La soluzione esatta (*linea continua*) e le soluzioni numeriche del problema (9.75) con i dati dell'esercizio 9.17 al tempo $t = 5$ ottenute con lo schema di Lax-Wendroff (*linea tratteggiata*) e lo schema Upwind (*linea tratto punto*). Il numero CFL è 0.8

Esercizio 9.17 Si approssimi il problema (9.75) con $a = 1$, $x \in (0, 0.5)$, $t \in (0, 1)$, dato iniziale $u^0(x) = 2 \cos(4\pi x) + \sin(20\pi x)$ e condizione al contorno $u(0, t) = 2 \cos(4\pi t) - \sin(20\pi t)$ per $t \in (0, 1)$ con gli schemi Lax-Wendroff (9.85) e *upwind* (9.87). Si fissi il numero CFL pari a 0.5. Verificare numericamente che lo schema Lax-Wendroff è accurato al secondo ordine sia rispetto a h che a Δt , mentre lo schema *upwind* è accurato al primo ordine sia rispetto a h che a Δt . Si consideri la norma $\|\cdot\|_{\Delta,2}$ per la valutazione dell'errore.

Esercizio 9.18 In Figura 9.20 sono riportate la soluzione esatta e le soluzioni numeriche del problema (9.75) ottenute con gli schemi di Lax-Wendroff (9.85) e *upwind* (9.87) al tempo $t = 5$ ed avendo considerato i dati dell'esercizio 9.17. Sapendo che il numero CFL è 0.8 ed è stato scelto $\Delta t = 5.e-3$, commentare i risultati ottenuti (in termini di coefficiente di dissipazione e di dispersione).

Soluzione degli esercizi proposti

In questo capitolo forniremo le soluzioni degli esercizi proposti alla fine dei precedenti nove capitoli. L'espressione "Soluzione $n.m$ " è di fatto una versione abbreviata di "Soluzione dell'Esercizio m -simo del Capitolo n -simo".

10.1 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 **MAT&OCT**

```
exp(-pi/2)
ans =
    0.2079
i^i
ans =
    0.2079
```

Soluzione 1.4 Si devono utilizzare le istruzioni:

```
L=2*eye(10)-3*diag(ones(8,1),-2)
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.

Soluzione 1.6 Si costruisce la matrice `A = [v1;v2;v3;v4]` dove `v1, v2, v3` e `v4` sono i vettori riga `MAT&OCT` 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 I=sequence(n)
I = zeros(n+2,1); I(1) = (exp(1)-1)/exp(1);
for i = 0:n, I(i+2) = 1 - (i+1)*I(i+1); end
```

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'elemento successivo z_{n+1} della successione sarà nullo. Ciò accade per $n \geq 30$.

Soluzione 1.11 Il metodo in esame è un esempio di metodo di tipo Monte Carlo. Un programma che lo implementa è il seguente:

```
function mypi=pimontecarlo(n)
x = rand(n,1); y = rand(n,1);
z = x.^2+y.^2;
v = (z <= 1);
m=sum(v); mypi=4*m/n;
```

Abbiamo fatto uso del comando `rand` per generare i numeri casuali. L'istruzione `v = (z <= 1)` è una versione abbreviata della seguente procedura: controlliamo se $z(k) \leq 1$ per ogni componente del vettore `z`. Se la disegualanza è soddisfatta per la componente k -sima di `z` (ovvero il punto $(x(k), y(k))$ giace all'interno del cerchio di raggio unitario) `v(k)` è posto uguale a 1, altrimenti a 0. Il comando `sum(v)` esegue quindi la somma di tutte le componenti del vettore `v` ovvero fornisce il numero di punti che cadono all'interno del cerchio.

Se si lancia il programma come `mypi=pimontecarlo(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 `mypi=3.1120`, mentre per `n=300000` otteniamo `mypi=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=bbpalgorithm(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
```

Per `n=10` si trova che il valore `pig` così calcolato coincide (nella precisione di `MAT&OCT`) con `pi`. In effetti, questo algoritmo 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`):

```
function bc=bincoeff(n,k)
k = fix(k); n = fix(n);
if k > n, disp('k deve essere compreso tra 0 e n');
    return; 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
```

fix Il comando `fix(k)` elimina l'eventuale parte decimale della variabile `k` (ovvero ne calcola la parte intera). 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 `return` provoca l'interruzione della `function`. Infine, il comando `prod(el)` calcola il prodotto di tutte le componenti del vettore `el`.

Soluzione 1.14 Le *function* che seguono implementano il calcolo di f_n con la formula $f_i = f_{i-1} + f_{i-2}$ (`fibrec`) o usando la formula (1.15) (`fibmat`):

```
function f=fibrec(n)
if n == 0
    f = 0;
elseif n == 1
    f = 1;
else
    f = fibrec(n-1)+fibrec(n-2);
end

function f=fibmat(n)
f = [0;1];
A = [1 1; 1 0];
f = A^n*f;
f = f(1);
```

Per $n=20$ troviamo i seguenti risultati e tempi di calcolo:

```
t=cputime; fn=fibrec(20), cpu=cputime-t
fn =
       6765
cpu =
      0.48

t=cputime; fn=fibmat(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 `MAT&OCT`).

Soluzione 1.15 Costruiamo alcuni elementi della successione $a_n = (1+1/n)^n$ con $n \in [1, 10^{20}]$ e rappresentiamoli graficamente con le seguenti istruzioni `MAT&OCT`:

```
n=logspace(1,20,40);
an=(1+1./n).^n;
semilogx(n,an,'co');
hold on; grid on
semilogx([1,1e20],[exp(1),exp(1)],'c--')
```

logspace Il comando `logspace(a,b,n)` costruisce un vettore riga di `n` elementi equidistribuiti in scala logaritmica tra 10^a e 10^b . La scala logaritmica è molto utile quando si devono rappresentare intervalli in cui l'ordine di grandezza dei numeri varia notevolmente. Il comando `semilogx(x,y)` crea un grafico con scala logaritmica in x e lineare in y .

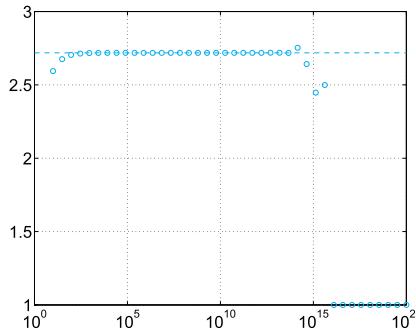


Figura 10.1. Alcuni valori della successione $a_n = (1 + 1/n)^n$ calcolati in **MATLAB** per il problema dell'Esercizio 1.15

Il grafico ottenuto è mostrato in Figura 10.1; da esso leggiamo che finché $n \lesssim 10^{13}$ i valori di `an` sono prossimi ad e (linea tratteggiata), poi essi cominciano ad oscillare e, per $n \gtrsim 10^{16}$, sono tutti valori pari a 1.

Qui abbiamo un esempio della non unicità dello zero nel sistema *floating point* \mathbb{F} . Quando $\frac{1}{n} < \frac{\epsilon_M}{2}$, dove ϵ_M è l'*epsilon* di macchina introdotto in Sezione 1.2.1, il numero $f\ell_t(1 + 1/n)$ risulta pari a 1 e, di conseguenza, anche $(f\ell_t(1 + 1/n))^n$ rimarrà uguale a 1.

Poiché si ha

```
1/(eps/2)
ans =
9.007199254740992e+15
```

avremo $f\ell_t(1 + 1/n) = 1$ per ogni $n > 9.007199254740992e+15$.

10.2 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 2.1) 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=@(x) 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, scegliendo $a=1$ e $b=3$, 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 $f(V) = pV + aN^2/V - abN^3/V^2 - pNb - kNT$, dove N è il numero di molecole. 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=@(x) 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 Il valore incognito di ω è lo zero della funzione $f(\omega) = s(1,\omega) - 1 = 9.8[\sinh(\omega) - \sin(\omega)]/(2\omega^2) - 1$. Dal grafico di f si deduce che f ha un unico zero nell'intervallo $(0.5, 1)$. A partire da questo intervallo, il metodo di bisezione converge con l'accuratezza desiderata in 15 iterazioni alla radice $\omega = 0.61214447021484$:

```
f=@(omega) 9.8/2*(sinh(omega)-sin(omega))./omega.^2-1;
[zero,res,niter]=bisection(f,0.5,1,1.e-05,100)
```

```
zero =
6.1214e-01
res =
3.1051e-06
niter =
15
```

Soluzione 2.4 La diseguaglianza (2.6) 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 se k_{\min} è tale che $2^{-k_{\min}-1}(b-a) < \varepsilon$ cioè se $2^{-k_{\min}-1} < \varepsilon/(b-a)$, da cui si ricava la (2.6).

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 al variare di γ . Consideriamo il caso $\gamma = 2$: partendo dal dato iniziale $x^{(0)} = 1$, ed avendo posto $\text{tol}=1.e-10$, il metodo di Newton (richiamato con il Programma 2.2) converge al valore $\hat{\alpha} = 1.4961 \cdot 10^{-4}$ in 31 iterazioni in MATLAB e ad $\hat{\alpha} = 2.4295 \cdot 10^{-4}$ in 30 iterazioni in Octave, mentre il valore esatto dello zero di f è $\alpha = 0$. Questa discrepanza può essere spiegata solo tenendo conto dell'andamento estremamente appiattito della funzione in un intorno della radice stessa, in quanto il problema della ricerca degli zeri di una funzione è mal condizionato (si veda l'osservazione alla fine della Sezione 2.8.2). Anche prendendo una tolleranza pari alla precisione di macchina

ϵ_M , il metodo converge alla stessa soluzione nello stesso numero di iterazioni. In effetti, si nota che il residuo corrispondente risulta 0 sia in MATLAB che in Octave. Per $\gamma = 3$, ponendo la tolleranza pari a ϵ_M , il metodo converge sia in MATLAB che in Octave in 9 iterazioni al valore 1.8579208291502 a partire da $x^{(0)} = 1$, mentre a partire da $x^{(0)} = -1$ si ha convergenza in 9 iterazioni al valore -1.8579208291502 (in entrambi i casi il residuo è nullo).

Soluzione 2.7 Bisogna risolvere le equazioni $x^2 = a$ e $x^3 = a$, rispettivamente. Gli algoritmi cercati sono pertanto: dato $x^{(0)}$ calcolare,

$$\begin{aligned}x^{(k+1)} &= \frac{1}{2} \left(x^{(k)} + \frac{a}{x^{(k)}} \right), \quad k \geq 0 \quad \text{per la radice quadrata,} \\x^{(k+1)} &= \frac{1}{3} \left(2x^{(k)} + \frac{a}{(x^{(k)})^2} \right), \quad k \geq 0 \quad \text{per la radice cubica.}\end{aligned}$$

Soluzione 2.8 Poniamo $e^{(k)} = x^{(k)} - \alpha$. Allora, sviluppando f in serie di Taylor in un intorno del punto $x^{(0)}$ si trova:

$$0 = f(\alpha) = f(x^{(k)}) - e^{(k)} f'(x^{(k)}) + \frac{1}{2}(e^{(k)})^2 f''(x^{(k)}) + \mathcal{O}((e^{(k)})^3). \quad (10.1)$$

Per il metodo di Newton si ha

$$e^{(k+1)} = e^{(k)} - f(x^{(k)})/f'(x^{(k)}) \quad (10.2)$$

e combinando (10.1) con (10.2), si ottiene

$$e^{(k+1)} = \frac{1}{2}(e^{(k)})^2 \frac{f''(x^{(k)})}{f'(x^{(k)})} + \mathcal{O}((e^{(k)})^3).$$

Dopo aver diviso per $(e^{(k)})^2$, passando al limite per $k \rightarrow \infty$, si trova la relazione cercata.

Soluzione 2.9 Si devono calcolare le radici dell'equazione (2.2) 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. Si suppone di risolvere il problema per i soli valori di β pari a $k\pi/150$ per $k = 0, \dots, 100$ (se β è maggiore di 2.6389 il metodo di Newton non converge in quanto il sistema non ammette configurazioni possibili). Con i comandi riportati di seguito possiamo ottenere il risultato cercato, mostrato in Figura 10.2 (a sinistra).

```
a1=10; a2=13; a3=8; a4=10;
ss = (a1^2 + a2^2 - a3^2 + a4^2)/(2*a2*a4);
n=150; x01=-0.1; x02=2*pi/3; kmax=100;
beta=zeros(100,1);
for k=0:100
    w = k*pi/n; i=k+1; beta(i) = w;
    f = @(x) 10/13*cos(w)-cos(x)-cos(w-x)+ss;
    df = @(x) sin(x)-sin(w-x);
    [zero,res,niter]=newton(f,df,x01,1e-5,kmax);
```

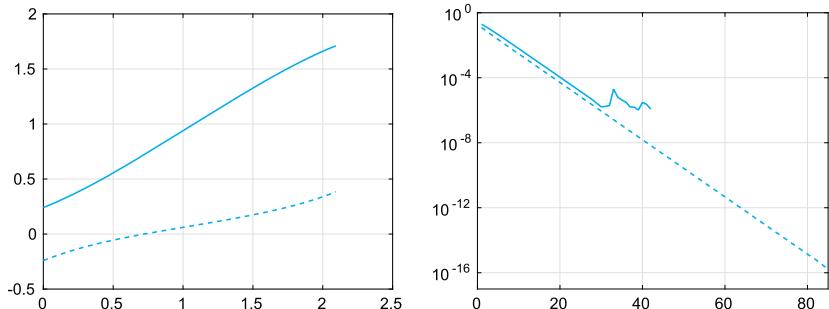


Figura 10.2. A sinistra: le due curve rappresentano le due possibili configurazioni del sistema di aste (precisamente l'angolo α) al variare di $\beta \in [0, 2\pi/3]$ (*Soluzione 2.9*). A destra: andamento dell'errore nel metodo di Newton per il calcolo dello zero di $f(x) = x^3 - 3x^2 2^{-x} + 3x 4^{-x} - 8^{-x}$ (linea continua) e di $f(x) = (x - 2^{-x})^3$ (linea tratteggiata) (*Soluzione 2.11*)

```

alpha1(i) = zero; niter1(i) = niter;
[zero,res,niter]=newton(f,df,x02,1e-5,kmax);
alpha2(i) = zero; niter2(i) = niter;
end
plot(beta, alpha1, 'c--', beta, alpha2, 'c', 'Linewidth', 2)
grid on

```

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 2 e le 6).

Soluzione 2.10 Analizzando il grafico di f vediamo che essa ammette due zeri positivi ($\alpha_2 \simeq 1.5$ e $\alpha_3 \simeq 2.5$) ed uno negativo ($\alpha_1 \simeq -0.5$). Il metodo di Newton converge in 4 iterazioni (avendo posto $x^{(0)} = -0.5$ e `tol = 1.e-10`) al valore α_1 :

```

f=@(x) exp(x)-2*x^2; df=@(x) exp(x)-4*x;
x0=-0.5; tol=1.e-10; kmax=100;
format long; [zero,res,niter]=newton(f,df,x0,tol,kmax)

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 ad una delle radici positive, α_2 o α_3 .

Soluzione 2.11 Poniamo $x^{(0)} = 0$ e `tol = ϵ_M` . Digitando i seguenti comandi

```
f=@(x) x.^3-3.*x.^2.*2.^(-x)+3.*x.*4.^(-x)-8.^(-x);
df=@(x) 3.*x.^2-6.*x.*2.^(-x)+3.*x.^2.*2.^(-x).*...
    log(2)+3.*4.^(-x)-3.*x.*4.^(-x).*log(4)+...
    8.^(-x).*log(8);
[zero,res,niter,difv]=newton(f,df,0,eps,100);
semilogy(difv,'c','Linewidth',2); grid on
```

osserviamo che il metodo di Newton converge in 43 iterazioni al valore 0.641182985886554 in MATLAB, ed in 33 iterazioni al valore 0.64118239763648 in Octave. In entrambi i casi deduciamo che la convergenza non è quadratica. Grazie alla relazione (2.27) sappiamo che la differenza tra due iterate successive $|x^{(k+1)} - x^{(k)}|$ è una buona stima dell'errore $|\alpha - x^{(k)}|$ per k grande. Rappresentiamo pertanto in scala semilogaritmica il vettore `difv` fornito in output dal Programma `newton.m` al variare dell'iterazione k (si veda la Figura 10.2, a destra). Prendiamo ad esempio l'esecuzione in MATLAB: l'errore decresce linearmente per $k \leq 30$, poi comincia ad oscillare attorno a valori dell'ordine di 10^{-6} e infine le iterazioni si fermano in quanto si genera $-f(x^{(k)})/f'(x^{(k)}) = 0$ ed il test d'arresto $|x^{(k+1)} - x^{(k)}| = |f(x^{(k)})/f'(x^{(k)})| < \epsilon_M$ è soddisfatto.

Il comportamento dell'errore per $k \leq 30$ è chiaramente lineare, sintomo che la radice non è semplice, ma multipla. Analizzando meglio l'espressione di f osserviamo infatti che $f(x) = (x - 2^{-x})^3$, ovvero la radice ha molteplicità 3. Per recuperare il secondo ordine di convergenza potremmo ricorrere al metodo di Newton modificato.

La ragione del comportamento oscillante dell'errore per $30 \leq k \leq 43$ è da imputare agli errori di cancellazione che si generano dalla somma degli addendi di f e della sua derivata in prossimità della radice. Infatti, risolvendo nuovamente l'equazione non lineare con il metodo di Newton, ora con $f(x) = (x - 2^{-x})^3$ anziché nella forma originaria:

```
hold on
f=@(x)(x-2.^(-x)).^3;
df=@(x)(1+2.^(-x)*log(2))*3.*(x-2.^(-x)).^2;
[zero1,res1,niter1,difv1]=newton(f,df,1,eps,100);
```

otteniamo convergenza in 85 iterazioni al valore 0.6411857445049863 (diverso dalla sesta cifra decimale in poi rispetto al valore calcolato precedentemente), vediamo che le oscillazioni nel vettore `difv` spariscono e l'errore decade (sempre linearmente perché la radice è multipla) fino alla tolleranza imposta per il test d'arresto, ovvero l'`epsilon` macchina. (Si veda la Soluzione 2.20 per l'utilizzo di un altro test d'arresto.)

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 studio all'intervallo $(0, \pi/2)$. Il metodo di Newton con $x^{(0)} = \pi/4$, `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:

```
M=6000; v=1000; f=@(r) M-v*(1+r)./r.*((1+r).^5-1);
df=@(r) v*((1+r).^5.*(1-5*r)-1)./(r.^2);
[zero,res,niter]=bisection(f,0.01,0.1,1.e-12,5);
```

```
[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 (2.38) ha uno zero in $(\pi/6, \pi/4)$. Il metodo di Newton, richiamato con le seguenti istruzioni:

```
l1=8; l2=10; g=3*pi/5;
f=@(a) -12*cos(g+a)/sin(g+a)^2-11*cos(a)/sin(a)^2;
df=@(a) [l2/sin(g+a)+2*l1*cos(g+a)^2/sin(g+a)^3+...
           11/sin(a)+2*l1*cos(a)^2/sin(a)^3];
[zero,res,niter]=newton(f,df,pi/4,1.e-15,100)
L=12/sin(2*pi/5-zero)+11/sin(zero)
```

converge al valore cercato, 0.596279927465474, in 6 iterazioni a partire da $x^{(0)} = \pi/4$. La lunghezza massima sarà allora pari a $L = 30.5484$.

Soluzione 2.15 Anzitutto riportiamo il problema nella forma $f(x) = (1 - x)/(100x^2 - 100x + 26) - x/3 = 0$ in modo da poter calcolare le radici della funzione f con i metodi richiesti. Eseguendo i comandi

```
f=@(x)(1-x)./(100*x.^2-100*x+26)-x/3;
fplot(f,[-2,2]); axis equal; grid on
```

osserviamo che la funzione data ha un'unica radice nell'intervallo $(0.5, 1)$ e che questa è semplice in quanto non c'è tangenza tra il grafico di f e l'asse delle ascisse (si veda la Figura 10.3, a sinistra). Quindi, se i dati iniziali sono sufficientemente vicini alla radice, Newton convergerà quadraticamente e secanti con ordine $p = (1 + \sqrt{5})/2$. Osserviamo che la funzione ammette due punti stazionari nell'intervallo $(0, 0.5)$ che potrebbero dare problemi alla convergenza dei metodi qualora si scegliessero "male" i dati iniziali.

Dopo aver calcolato la funzione $f'(x)$, richiamiamo il metodo di Newton con $x^{(0)} = 0.45$ con le seguenti istruzioni:

```
df=@(x)(25*x.^2-50*x+37/2)./(50*x.^2-50*x+13).^2-1/3
tol=1.e-8; kmax=200;
x0=0.4; [zero,res,niter,difv]=newton(f,df,x0,tol,kmax)
```

ottenendo convergenza in 26 iterazioni al valore $\text{zero}=6.000000000000001e-01$. Ricordando che la radice è semplice, Newton dovrebbe convergere quadraticamente in un numero di iterazioni ben inferiore a 26. Infatti l'errore ad ogni passo dovrebbe essere circa il quadrato dell'errore al passo precedente e, supponendo di partire da un errore dell'ordine di 10^{-1} , dovremmo arrivare alla tolleranza richiesta in circa 5 iterazioni. Disegnando il vettore `difv` con i comandi

```
semilogy(difv,'c','LineWidth',2); grid on
```

osserviamo che durante le prime 21 iterazioni i valori $|x^{(k+1)} - x^{(k)}|$ rimangono molto alti, per poi convergere nel giro di 4 iterazioni alla tolleranza richiesta (si veda la Figura 10.3, a destra). La ragione di questo comportamento è da imputare alla scelta non ottimale del punto iniziale. Infatti la radice della retta tangente ad f nel punto $x^{(0)} = 0.4$ (cioè $x^{(1)}$) è più vicina al punto di minimo relativo prossimo a 0, che non alla radice di f ed il comportamento crescente e decrescente di f fa sì che le prime iterate di Newton continuino ad oscillare attorno al punto di massimo relativo della funzione f (si veda la Figura 10.3, a sinistra).

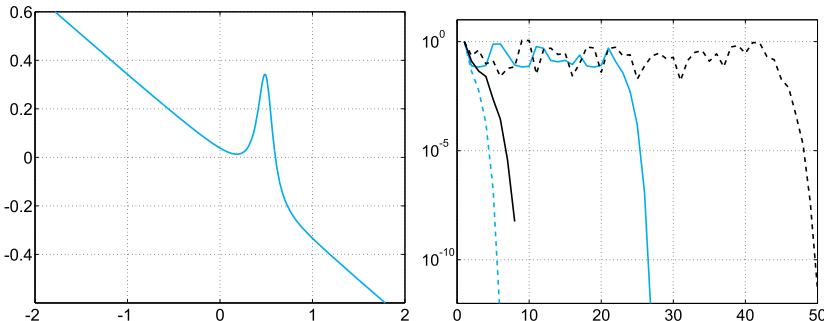


Figura 10.3. La funzione della Soluzione 2.15 (a sinistra). Le storie di convergenza dei metodi utilizzati per lo svolgimento dell'esercizio 2.15 (a destra): Newton con $x^{(0)} = 0.4$ (azzurro in linea tratteggiata), Newton con $x^{(0)} = 0.55$ (azzurro in linea continua), secanti con $x^{(0)} = 0.45$ e $x^{(1)} = 0.75$ (nero in linea continua), secanti con $x^{(0)} = 0.5$ e $x^{(1)} = 0.96$ (nero in linea tratteggiata)

Prendendo invece $x^{(0)} = 0.55$, otteniamo convergenza alla radice in 5 iterazioni, in linea con l'ordine quadratico del metodo di Newton. Questa volta la retta tangente ad f nel punto $x^{(0)}$ ha una radice molto prossima ad α e quindi Newton converge quadraticamente dalle prime iterate. In entrambi i casi otteniamo un residuo pari a circa 10^{-17} .

Per quanto riguarda il metodo delle secanti, la prima scelta dei dati iniziali ($x^{(0)} = 0.45$, $x^{(1)} = 0.75$) porta a convergenza in 7 iterazioni con un residuo dell'ordine di 10^{-13} , mentre la seconda scelta ($x^{(0)} = 0.5$, $x^{(1)} = 0.96$) porta a convergenza in ben 49 iterazioni con un residuo pari a circa 10^{-17} . Le istruzioni MATLAB per il primo run sono

```
x0=0.45;x1=0.75;
[zero,res,niter,difv]=secant(f,x0,x1,tol,nmax)
semilogy(difv,'k','Linewidth',2); grid on
```

Come nel caso di Newton con $x^{(0)} = 0.4$, anche qui le iterate prodotte da secanti continuano ad oscillare attorno al punto di massimo relativo per ben 40 iterazioni, prima che si innesti la convergenza superlineare (si veda la Figura 10.3, a destra).

Soluzione 2.16 Le funzioni $f_1(x, y) = x^3 + y - 2x^2 - 2$ e $f_2(x, y) = (x - 0.5)^2 - y^2 + x + \gamma$ del sistema sono definite da \mathbb{R}^2 a valori in \mathbb{R} , quindi possiamo localizzare le radici del sistema nel piano cartesiano disegnando le curve di livello di f_1 e f_2 corrispondenti a $z = 0$. I comandi MATLAB sono:

```
gamma=2;
f1=@(x,y) x.^3+y-2*x.^2-2;
f2=@(x,y)(x-0.5).^2-y.^2+x+gamma;
[xx,yy]=meshgrid(-5:.1:5);
z1=f1(xx,yy); z2=f2(xx,yy);
contour(xx,yy,z1,[0,0], 'c');
hold on; grid on
contour(xx,yy,z2,[0,0], 'k');
```

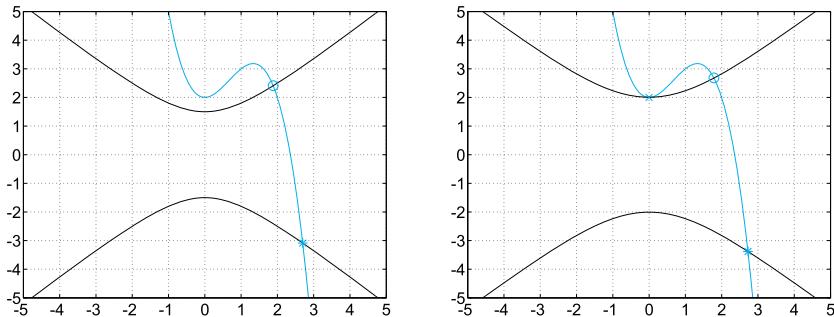


Figura 10.4. La rappresentazione grafica del sistema della Soluzione 2.16, $\gamma = 2$ (a sinistra) e $\gamma = 3.75$ (a destra)

Come si vede in Figura 10.4, a sinistra, esistono due intersezioni semplici tra le due curve, quindi scegliendo opportunamente il punto iniziale, il metodo di Newton dovrà convergere quadraticamente. Definiamo la funzione vettoriale $\mathbf{f}(\mathbf{x}) = [f_1(x_1, x_2), f_2(x_1, x_2)]^T$ e la matrice Jacobiana $\mathbf{J}_\mathbf{f}(\mathbf{x})$ come segue:

```
F=@(x)[x(1)^3+x(2)-2*x(1)^2-2;
        (x(1)-0.5)^2-x(2)^2+x(1)+gamma];
J=@(x)[3*x(1)^2-4*x(1),1;
        2*(x(1)-0.5)+1,-2*x(2)];
```

Fissiamo tolleranza `tol=1.e-10` e numero massimo di iterazioni `kmax=100` per il test d'arresto, scegliamo $\mathbf{x}^{(0)} = [2, 2]^T$ e poi $\mathbf{x}^{(0)} = [3, -3]^T$, e richiamiamo il Programma `newtonsys.m` 2.3. Con le istruzioni:

```
tol = 1e-10; kmax = 100;
x0 = [2;2];
[alpha1,res,niter,difv] = newtonsys(F, J, x0, tol, kmax)
x0 = [3;-3];
[alpha2,res,niter,difv] = newtonsys(F, J, x0, tol, kmax)
```

otteniamo:

```
alpha1 =
    1.884905812441627e+00
    2.408914677147414e+00
res =
    1.601186416994689e-15
niter =
    5

alpha2 =
    2.698595219862635e+00
    -3.087461118891291e+00
res =
    3.972054645195637e-15
niter =
    5
```

La convergenza ad entrambe le radici è quadratica, come previsto dalle proprietà di convergenza del metodo.

Procedendo in modo analogo, ma prendendo stavolta $\gamma = 3.75$, osserviamo che ora ci sono due radici semplici ed una multipla, per la quale ci aspettiamo

una convergenza di tipo lineare (si veda Figura 10.4, a destra). Scegliendo $\mathbf{x}^{(0)} = [2, 2]^T$ e $\mathbf{x}^{(0)} = [3, -3]^T$ otteniamo ancora convergenza in 5 iterazioni alle radici semplici, mentre ponendo $\mathbf{x}^{(0)} = [-3, 3]^T$ otteniamo:

```
alpha3 =
-6.552497605544840e-09
2.000000000000000e+00
res =
0
niter =
30
```

ovvero convergenza lineare.

Soluzione 2.17 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 ϕ_N 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 (2.9). Se invece $m > 1$ il metodo converge con ordine 1 per la Proposizione 2.1.

Soluzione 2.18 Da uno studio grafico, effettuato con i comandi:

```
f=@(x) x^3+4*x^2-10; fplot(f,[0,2], 'c');
grid on; axis([0,2,-10,15])
```

si ricava che f ammette un solo zero reale pari a circa 1.36 (in Figura 10.5 a sinistra viene riportato l'ultimo grafico ottenuto). La funzione di iterazione e la sua derivata sono:

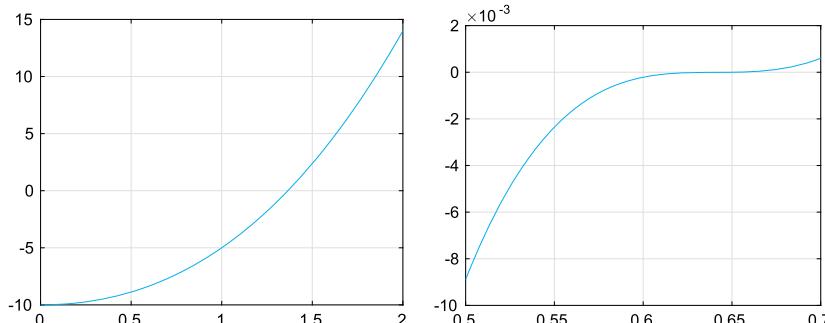


Figura 10.5. A sinistra il grafico di $f(x) = x^3 + 4x^2 - 10$ per $x \in [0, 2]$ (Soluzione 2.18). A destra il grafico di $f(x) = x^3 - 3x^2 2^{-x} + 3x 4^{-x} - 8^{-x}$ per $x \in [0.5, 0.7]$ (Soluzione 2.20)

$$\begin{aligned}\phi(x) &= \frac{2x^3 + 4x^2 + 10}{3x^2 + 8x} = -\frac{f(x)}{3x^2 + 8x} + x, \\ \phi'(x) &= \frac{(6x^2 + 8x)(3x^2 + 8x) - (6x + 8)(2x^3 + 4x^2 + 10)}{(3x^2 + 8x)^2} \\ &= \frac{(6x + 8)f(x)}{(3x^2 + 8x)^2},\end{aligned}$$

e $\phi(\alpha) = \alpha$. Sostituendo il valore di α , si ricava $\phi'(\alpha) = 0$, in quanto $f(\alpha) = 0$. Di conseguenza, il metodo proposto è convergente con ordine 2.

Soluzione 2.19 Il metodo in esame è almeno del second'ordine in quanto $\phi'(\alpha) = 0$.

Soluzione 2.20 Richiamiamo il metodo di Newton con test d'arresto sul residuo anziché sull'incremento, con le seguenti istruzioni

```
f=@(x) x^3-3*x^2*2^(-x)+3*x*4^(-x)-8^(-x);
df=@(x) [3*x^2-6*x*2^(-x)+3*x^2*2^(-x)*log(2)+...
           3*4^(-x)-6*x*4^(-x)*log(2)+3*8^(-x)*log(2)];
x0=0; tol=eps; kmax=100;
[zero,res,niter,difv]=newtonr(f,df,x0,tol,kmax)
semilogy(difv,'c');
```

Il Programma newtonr.m è una copia di newton.m in cui le istruzioni del ciclo while

```
while diff >= tol && k < kmax
    k = k + 1;
    diff = - fx/dfx;
    x = x + diff;
    diff = abs(diff); difv=[difv; diff];
    fx = fun(x,varargin{:});
    dfx = dfun(x,varargin{:});
end
```

sono sostituite da

```
while diff >= tol && k < kmax
    k = k + 1;
    x = x - fx/dfx;
    fx = fun(x,varargin{:});
    dfx = dfun(x,varargin{:});
    diff = abs(fx); difv=[difv; diff];
end
```

Otteniamo convergenza in 30 iterazioni al valore 0.6411822108638944, con una discrepanza dell'ordine di 10^{-7} sul secondo risultato calcolato nella Soluzione 2.11. Osserviamo anzitutto che il test d'arresto sul residuo non risente degli errori di cancellazione, tuttavia, poiché l'andamento della funzione è estremamente schiacciato in prossimità dello zero (quindi $f' \simeq 0$ in un intorno della radice), esso è poco affidabile, si veda la Sezione 2.3.1, e riteniamo più affidabile il valore della radice calcolato nella Soluzione 2.11 con l'espressione di $f(x) = (x - 2^{-x})^3$. In Figura 10.5 (a destra), riportiamo il grafico di f in $(0.5, 0.7)$ ottenuto con i seguenti comandi:

```
f=@(x) x^3-3*x^2*2^(-x)+3*x*4^(-x)-8^(-x);
fplot(f,[0.5 0.7], 'c'); grid on
```

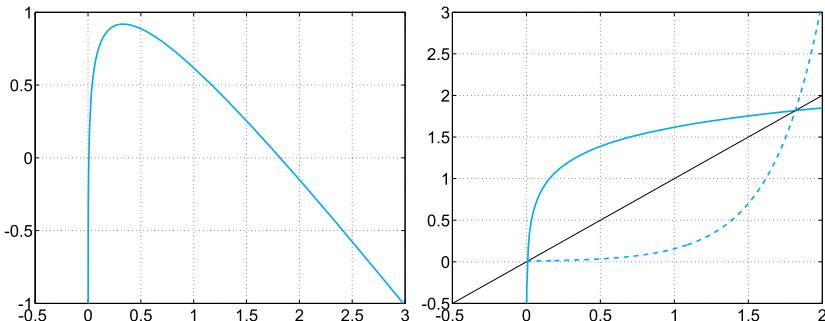


Figura 10.6. La funzione della Soluzione 2.21 (a sinistra). Le funzioni di punto fisso $\phi_1(x)$ (linea continua) e $\phi_2(x)$ (linea tratteggiata) (a destra)

Soluzione 2.21 Calcolare le intersezioni tra le due funzioni equivale a trovare le radici della funzione $f(x) = f_1(x) - f_2(x) = \log(x/\pi)/3 - x + 2$. Eseguendo i comandi

```
fun=@(x)1/3*log(x/pi)-x+2;
xx=linspace(0.0001,3,1000);
plot(xx,fun(xx),'c','Linewidth',2);
axis([-0.5,3,-1,1]); grid on
```

disegniamo la funzione f ed osserviamo che essa ha due radici semplici, la prima, α_1 , molto prossima a $x = 0$, la seconda, α_2 , nell'intervallo $(1.5, 2)$ (si veda la Figura 10.6, a sinistra). Per approssimare α_1 , fissiamo tolleranza $tol=1.e-8$ e numero massimo di iterazioni pari a $kmax=200$. Scegliendo $x^{(0)} = 0.1$, i seguenti comandi

```
dfun=@(x)1/(3*x)-1;
kmax=100;tol=1.e-6; x0=0.1;
[zero,res,niter,difv]=newton(fun,dfun,x0,tol,kmax)
```

producono

```
zero =
 1.817594719494452 - 0.0000000000000000i
res =
 0 + 2.545482423889976e-19i
niter =
 7
```

ovvero Newton è converge alla radice $\alpha_2 \in (1.5, 2)$. La presenza di parti immaginarie nei risultati mette in luce da un lato la capacità del metodo di Newton di trattare anche quantità in campo complesso. D'altro canto, sono stati generati valori $x^{(k)}$ negativi (quando l'argomento della funzione \log è negativo, MATLAB invoca la definizione della funzione logaritmo in campo complesso). Infatti la tangente ad f in $x^{(0)} = 0.1$ ha una radice negativa e otteniamo $x^{(1)} \simeq -0.2218$. Per garantire che i valori $x^{(k)}$ rimangano positivi, basterà scegliere un punto iniziale positivo, ma a sinistra di α_1 . Prendendo ad esempio $x^{(0)} = 0.001$ otteniamo convergenza ad $\alpha_1 \simeq 0.007975804965810$ in 6 iterazioni, producendo un residuo dell'ordine di 10^{-15} .

Scegliendo poi $x^{(0)} = 1$, otteniamo

```
zero =
1.817594719494452
res =
2.220446049250313e-16
niter =
4
```

ovvero, ancora convergenza quadratica, ma stavolta alla radice α_2 .

Per quanto riguarda la richiesta sulle funzioni di punto fisso, dall'espressione di f ricaviamo subito una prima funzione $\phi_1(x) = \log(x/\pi)/3 + 2$. Representandola graficamente, insieme alla retta $y = x$ (si veda la Figura 10.6, a destra), osserviamo che i suoi due punti fissi coincidono con le radici di f , ed inoltre $|\phi_1(\alpha_1)| \gg 1$, mentre $|\phi_1(\alpha_2)| < 1$. In base al teorema di Ostrowski 2.1, possiamo concludere che il metodo di punto fisso con funzione di iterazione ϕ_1 convergerà al punto α_2 , ma non ad α_1 . Con le istruzioni `MAT&OCT`:

```
phi=@(x)1/3*log(x/pi)+2;
x0=1; [alpha,niter]=fixedpoint(phi,x0,tol,kmax)
```

otteniamo convergenza al punto 1.817594669337015 in 10 iterazioni. Di seguito riportiamo il Programma `fixedpoint.m` utilizzato.

Programma 10.1. `fixedpoint`: il metodo di punto fisso

```
function [alpha,niter,difv]=fixedpoint(phi,x0,tol,kmax)
%FIXEDPOINT Trova un punto fisso di una funzione.
% [ALPHA,K,DIFV]=FIXEDPOINT(PHI,X0,TOL,KMAX)
% Cerca il punto fisso ALPHA della funzione PHI
% partendo dal punto iniziale X0. TOL e KMAX sono
% rispettivamente tolleranza e numero massimo di
% iterazioni per il test d'arresto (sull'incremento).
% NITER e' il numero di iterazioni effettuate e
% DIFV e' un vettore che contiene gli errori
% |x^{k+1}-x^k| ad ogni passo k.
x = x0;
k = 0; err = tol+1; difv=[];
while err >= tol && k < kmax
    k=k+1; xnew=phi(x);
    err=abs(xnew-x); difv=[difv;err];
    x=xnew;
end
alpha = x; niter=k;
```

Pur prendendo un dato iniziale $x^{(0)} = 0.01$ molto prossimo ad α_1 , otteniamo comunque convergenza ad α_2 , stavolta in 12 iterazioni. Per costruire una funzione di punto fisso che converga ad α_1 , isoliamo la variabile x all'interno del logaritmo di f , otteniamo $\phi_2(x) = \pi e^{3(x-2)}$. In questo secondo caso si ha una situazione speculare rispetto a prima, ovvero $|\phi_2(\alpha_2)| \gg 1$ e $|\phi_2(\alpha_1)| < 1$ (si veda la Figura 10.6, a destra). Con le istruzioni `MAT&OCT`:

```
phi2= @(x)pi*exp(3*(x-2));
x0=0.1; [alpha,niter]=fixedpoint(phi2,x0,tol,kmax)
```

otteniamo convergenza al punto 0.007975805800262 in 5 iterazioni. In questo secondo caso la convergenza è molto veloce e risente positivamente del valore

molto piccolo di $|\phi_2'(\alpha_1)| \simeq 10^{-2}$. Dal Teorema di Ostrowski si evince infatti che tale valore fornisce una stima del fattore di riduzione dell'errore.

10.3 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: $\max_{x \in [-1, 1]} |f_1^{(5)}| \simeq 1.18$; $\max_{x \in [-1, 1]} |f_2^{(5)}| \simeq 1.54$; $\max_{x \in [-\pi/2, \pi/2]} |f_3^{(5)}| \simeq 1.41$. Grazie alla (3.7), gli errori corrispondenti sono allora limitati superiormente da 0.0018, 0.0024 e 0.0211, rispettivamente.

Soluzione 3.3 Tramite il comando `polyfit` di MATLAB si calcolano i polinomi interpolatori di grado 3 nei due casi:

```
anni=[1975 1980 1985 1990];
eoc=[72.8 74.2 75.2 76.4];
eor=[70.2 70.2 70.3 71.2];
coc=polyfit(anni,eoc,3);
cor=polyfit(anni,eor,3);
stimaoc=polyval(coc,[1977 1983 1988]);
stimaor=polyval(cor,[1977 1983 1988]);
```

I valori stimati per il 1977, 1983 e 1988 sono:

```
stimaoc =
    73.4464    74.8096    75.8576
stimaor =
    70.2328    70.2032    70.6992
```

rispettivamente per l'Europa occidentale ed orientale.

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 1987, fino a $t_7 = 157$, corrispondente al mese di novembre del 2000. 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 `[prezzo2002]=polyval(c,181)` si trova che il prezzo previsto a novembre del 2002 è di circa 11.24 euro.

Soluzione 3.5 In questo caso particolare, per cui abbiamo 4 nodi di interpolazione, la *spline* cubica interpolatoria calcolata con il comando `spline` coincide con il polinomio interpolatore. Infatti la *spline* calcolata interpola i dati, ha derivate prima e seconda continue ed ha derivata terza continua negli unici nodi interni x_1 e x_2 , per via della 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 2
    1.000740787500000e+03      1.000488237500000e+03
Columns 3 through 4
    1.000022450000000e+03      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 indichiamo con **s3**), quella naturale (**s3n**) e quella con derivata prima nulla agli estremi (**s3d**) (ottenuta con il Programma 3.2). 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 = cublicspline(anno,produzione,z);
s3d = cublicspline(anno,produzione,z,0,[0 0]);
```

Nella tabella seguente riportiamo i valori ottenuti (in migliaia di quintali ($=10^5$ kg) di agrumi):

Anno	1962	1977	1992
s3	5146.1	22641.8	41894.4
s3n	13285.3	22934.2	37798.0
s3d	24313.0	23126.0	22165.8

Il confronto con i dati effettivamente misurati negli anni 1962, 1977 e 1992 (12380, 27403 e 32059 in migliaia di quintali) 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 10.7 a sinistra). 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* **s3**, 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);
```

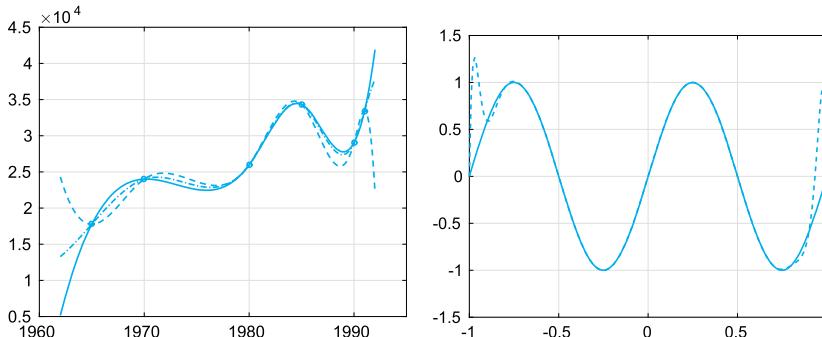


Figura 10.7. A sinistra: confronto fra i grafici delle *spline* cubiche generate nel corso della Soluzione 3.7: s_3 (linea continua), s_{3d} (linea tratteggiata), s_{3n} (linea tratto-punto). I cerchietti rappresentano i valori interpolati. A destra: il polinomio interpolatore (linea tratteggiata) e la *spline* cubica interpolatoria (linea continua) a confronto nel caso in cui si usino dati perturbati nella Soluzione 3.8. Si noti lo scollamento fra i due grafici agli estremi dell'intervallo

Quando usiamo i dati non perturbati (`pert=0`) i grafici di `p` e `s3` non sono distinguibili da quello della funzione f data. La situazione cambia drasticamente se si usano i dati perturbati (`pert=1.e-04`). In particolare il polinomio interpolatore presenta delle forti oscillazioni agli estremi dell'intervallo mentre la *spline* si mantiene sostanzialmente immutata (si veda la Figura 10.7 a destra). Questo esempio mostra come l'approssimazione con funzioni *spline* sia dunque più stabile rispetto alle piccole perturbazioni, di quanto non lo sia l'interpolazione polinomiale di Lagrange.

Soluzione 3.9 La funzione assegnata è di classe $C^\infty(\mathbb{R})$, quindi l'interpolatore globale di Lagrange di grado n su $(n+1)$ nodi di Chebyshev-Gauss-Lobatto (3.11) convergerà alla funzione f per $n \rightarrow \infty$.

Grazie alla Proposizione 3.3 possiamo concludere che anche l'interpolatore lineare composito $\Pi_1^H f$ convergerà ad f per $H \rightarrow 0$ con ordine 2 rispetto ad H (H è l'ampiezza dei sottointervalli). Infine ricordando la stima dell'errore (3.28) con $r = 0$, deduciamo che la spline cubica s_3 convergerà ad f per $H \rightarrow 0$ con ordine 4 rispetto ad H .

Verifichiamo numericamente quanto affermato con le seguenti istruzioni `MATLAB`. Denotiamo con N il numero globale di nodi, quindi $n = N - 1$ è il grado polinomiale per l'interpolazione globale di Lagrange e anche il numero dei sottointervalli sia nell'interpolazione composita che nelle spline. Con le seguenti istruzioni:

```
f=@(x)0.0039+0.0058./(1+exp(x));
xa=-5; xb=5; c1=(xa+xb)/2; c2=(xb-xa)/2;
x1=linspace(xa,xb,1000); y1=f(x1);
errlag=[ ]; errcomp=[ ]; errspline=[ ];
nn=4:4:70;
for n=nn
xi=-cos(pi*(0:n)'/n); xg=c1+c2*xi; yg=f(xg);
p1=barycentric(xg,yg,x1);
e=norm(p1-y1,inf); errlag=[errlag;e];
```

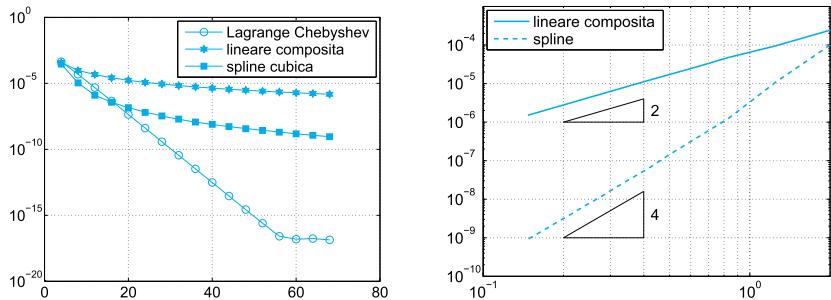


Figura 10.8. Gli errori di interpolazione per la Soluzione 3.9: a sinistra al variare di n , a destra al variare di H

```

xe=linspace(xa ,xb ,n+1); ye=f(xe);
pc1=interp1(xe,ye,x1);
ec=norm(pc1-y1,inf); errcomp=[errcomp;ec];
s=spline(xe,ye,x1);
es=norm(s-y1,inf); errspline=[errspline;es];
end

```

costruiamo al variare di n i tre interpolatori, valutiamo gli errori in 1000 nodi equispaziati nell'intervallo di interpolazione e li memorizziamo in tre vettori: **errlag** (per l'interpolatore di Lagrange globale), **errcomp** (per l'interpolatore lineare composito) e **errspline** (per la spline). Per costruire l'interpolatore globale di Lagrange abbiamo utilizzato la forma baricentrica (3.15) ed il Programma 3.1, mentre per costruire $\Pi_1^H f$ e la spline s_3 , abbiamo richiamato i programmi **MATGOCT**, rispettivamente **interp1** e **spline**.

In Figura 10.8 riportiamo l'andamento degli errori rispetto ad n , a sinistra, e rispetto ad H , a destra (in quest'ultimo caso solo per interpolazione composita e per la spline.) Osserviamo che l'errore dell'interpolazione globale su nodi di Chebyshev decresce fino al raggiungimento dell'epsilon di macchina e poi si assesta su tale valore per effetto della propagazione degli errori di arrotondamento. L'errore dell'interpolazione globale su nodi di Chebyshev decresce più velocemente di una qualsiasi potenza di n : in tal caso si dice che la convergenza è di tipo esponenziale. Gli errori dell'interpolazione composita e spline sono invece di tipo polinomiale rispetto ad H , il primo decresce come H^2 ed il secondo come H^4 , come si evince dalla grafica in Figura 3.9.

Soluzione 3.10 La funzione assegnata è di classe $C^\infty(\mathbb{R})$, quindi possiamo applicare la Proposizione 3.3. Infatti, se $\frac{H^2}{8} \max_{x \in [x_0, x_n]} |f''(x)| \leq \varepsilon$, allora vale anche che $\max_{x \in [x_0, x_n]} |f(x) - \Pi_1^H f(x)| \leq \varepsilon$. In particolare, isolando H nella formula precedente abbiamo

$$H \leq \left(\frac{8}{\varepsilon \max_{x \in [x_0, x_n]} |f''(x)|} \right)^{1/2}. \quad (10.3)$$

Rappresentando graficamente la funzione $f''(x) = 2e^{-x^2}(2x^2 - 1)$, deduciamo che $\max_{[-2, 2]} |f''(x)| = 2$, quindi dalla diseguaglianza (10.3) ricaviamo

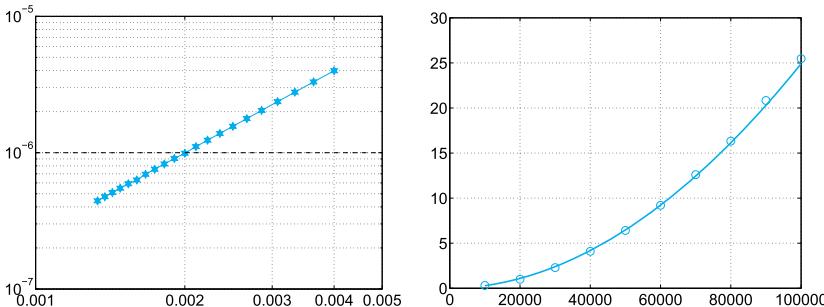


Figura 10.9. Gli errori di interpolazione per la Soluzione 3.10 (a sinistra). Approssimazione nel senso dei minimi quadrati per i dati della Soluzione 3.17 (a destra)

che, se $H \leq 2 \cdot 10^{-3}$, ovvero $n \geq 4/H = 2000$, allora l'errore di interpolazione è minore o uguale a 10^{-6} . Per verificare numericamente quanto trovato, eseguiamo le seguenti istruzioni MATLAB:

```
f=@(x)exp(-x.^2)
xa=-2; xb=2;
x1=linspace(xa,xb,20000); y1=f(x1);
err=[ ]; nn=1000:100:3000
for n=nn
xe=linspace(xa,xb,n+1); ye=f(xe);
pc1=interp1(xe, ye, x1);
ec=norm(pc1-y1,inf); err=[err;ec];
end
```

L'errore, valutato in norma infinito su 20000 punti in $[-2, 2]$ è rappresentato in Figura 10.9. L'errore con $H = 0.002$ risulta pari a $9.900\text{e-}07$, in perfetto accordo con quanto stimato teoricamente.

Soluzione 3.11 Se $n = m$, ponendo $\tilde{f} = \Pi_n f$ si ha addirittura 0 a primo membro della (3.29) 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 3.12 I polinomi hanno i seguenti coefficienti (riportati con le sole prime 4 cifre significative ed ottenuti con il comando polyfit):

$$\begin{aligned} K &= 0.67, a_4 = 7.211 \cdot 10^{-8}, a_3 = -6.088 \cdot 10^{-7}, a_2 = -2.988 \cdot 10^{-4}, a_1 = \\ &\quad 1.650 \cdot 10^{-3}, a_0 = -3.030; \\ K &= 1.5, a_4 = -6.492 \cdot 10^{-8}, a_3 = -7.559 \cdot 10^{-7}, a_2 = 3.788 \cdot 10^{-4}, a_1 = \\ &\quad 1.67310^{-3}, a_0 = 3.149; \\ K &= 2, a_4 = -1.050 \cdot 10^{-7}, a_3 = 7.130 \cdot 10^{-8}, a_2 = 7.044 \cdot 10^{-4}, a_1 = \\ &\quad -3.828 \cdot 10^{-4}, a_0 = 4.926; \\ K &= 3, a_4 = -2.319 \cdot 10^{-7}, a_3 = 7.740 \cdot 10^{-7}, a_2 = 1.419 \cdot 10^{-3}, a_1 = \\ &\quad -2.574 \cdot 10^{-3}, a_0 = 7.315. \end{aligned}$$

A sinistra di Figura 10.10 riportiamo il polinomio ottenuto per i dati della colonna relativa a $K = 0.67$ nella Tabella 3.1.

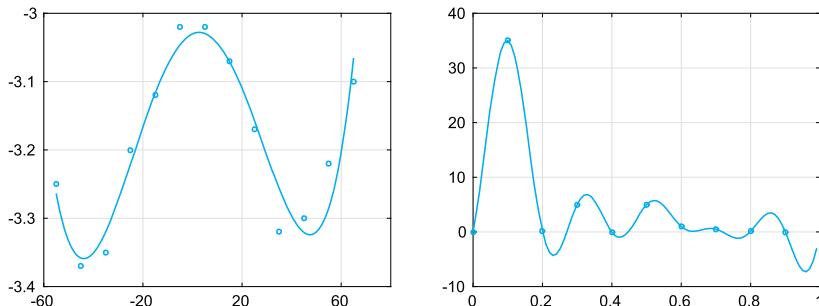


Figura 10.10. A sinistra: polinomio di grado 4 dei minimi quadrati (*linea continua*) a confronto con i dati della colonna di Tabella 3.1 per $K = 0.67$ (Soluzione 3.12). A destra: l’interpolatore trigonometrico ottenuto con le istruzioni della Soluzione 3.16. I pallini si riferiscono ai dati sperimentali disponibili

Soluzione 3.13 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): 15280.12 nel 1962; 27407.10 nel 1977; 32019.01 nel 1992. Essi sono delle ottime approssimazioni dei valori effettivamente misurati (12380, 27403 e 32059 rispettivamente).

Soluzione 3.14 Possiamo riscrivere i coefficienti del sistema (3.31) in funzione della media e della varianza osservando che quest’ultima può essere scritta come $v = \frac{1}{n+1} \sum_{i=0}^n x_i^2 - M^2$. Così i coefficienti della prima equazione sono $(n+1)$ e M , mentre quelli della seconda equazione sono M e $(n+1)(v+M^2)$.

Soluzione 3.15 L’equazione della retta dei minimi quadrati, soluzione del problema dato, è $y = a_0 + a_1 x$, dove a_0 e a_1 sono le soluzioni del sistema (3.31). La prima equazione di (3.31) dice che il punto di ascissa M e ordinata $\sum_{i=0}^n y_i / (n+1)$ appartiene alla retta dei minimi quadrati.

Soluzione 3.16 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 a destra di Figura 10.10.

Soluzione 3.17 Cerchiamo un polinomio che approssimi i valori dati nel senso dei minimi quadrati. L’idea è cercare una costante $c > 0$ ed un $q > 0$ tali che i tempi T dipendano da N secondo la legge $T(N) = c N^q$. Applicando il logaritmo ad entrambi i termini dell’equazione abbiamo $\log(T) = \log(c N^q) = q \log(N) + \log(c)$ e, posto $y = \log(T)$, $x = \log(N)$, $a_1 = q$ e $a_2 = \log(c)$, il problema del calcolo di c e q è ricondotto al calcolo dei coefficienti a_1 e a_2 della retta dei minimi quadrati $y = a_1 x + a_2$, che approssima i dati (x_i, y_i) per $i = 1, \dots, N$. Le istruzioni `MAT&OCT` per calcolare c e q sono allora:

```
N=(10000:10000:100000); L=length(N);
T=[0.31 0.99 2.29 4.10 6.41 9.20, ...]
```

```

12.60 16.32 20.83 25.47] ';
y=log(T); x=log(N);
a=polyfit(x,y,1);
q=a(1); c=exp(a(2));

```

Otteniamo $c=4.63e-9$, $q=1.94$, quindi possiamo affermare che la complessità computazionale dell'algoritmo è di tipo polinomiale di ordine 2. In Figura 10.9, a destra, sono mostrati i dati e la funzione $T(N) = 4.63 \cdot 10^{-9} \cdot N^{1.94}$, ottenuti con i comandi:

```

plot(N,T,'co','Markersize',10);
hold on
n1=linspace(N(1),N(end),100);
t1=c*n1.^q; grid on
plot(n1,t1,'c-','Linewidth',2);

```

10.4 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 formula di (4.13), 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 $\xi_0 \in (x_0, x_2)$.

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 all'ordine due. 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 $\xi_+ \in (\bar{x}, \bar{x} + h)$. Sottraendo queste due espressioni e dividendo per $2h$ otteniamo la formula (4.12), che è una approssimazione di ordine 2 di $f'(\bar{x})$.

Soluzione 4.3 Eseguendo operazioni analoghe a quelle indicate nella Soluzione 4.2, si trovano i seguenti errori (supponendo $f \in C^4$):

$$a. -\frac{1}{4}f^{(4)}(\xi)h^3, \quad b. -\frac{1}{12}f^{(4)}(\xi)h^3, \quad c. \frac{1}{6}f^{(4)}(\xi)h^3.$$

Soluzione 4.4 Usiamo l'approssimazione (4.11). Si trovano i seguenti valori

t (Mes)	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

$$\frac{(b-a)^3}{24M^2} \max_{x \in [a,b]} |f''(x)|,$$

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. Affinché dunque l'errore sia minore di 10^{-4} si dovrà avere $2 \cdot 5^3 / (24M^2) < 10^{-4}$ cioè $M > 322$.

Anche la funzione f_2 è derivabile per ogni ordine. Con un semplice studio si ricava che $\max_{x \in [0,\pi]} |f_2''(x)| = \sqrt{2}e^{3\pi/4}$; di conseguenza, perché 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 71 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 (4.21).

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 $M = 8$ per la formula composita di Gauss-Legendre con $n = 1$ (per questa formula si può usare il Programma 10.2). Come si vede il vantaggio nell'uso di quest'ultima formula è estremamente rilevante.

Programma 10.2. glcomp1: formula composita di quadratura di Gauss-Legendre con $n = 1$

```
function [intgl]=glcomp1(fun,a,b,M,varargin)
% GLCOMP1 calcola un integrale con formule Gaussiane
% INTGL=GLCOMP1(FUN,A,B,M,VARARGIN)
% INTGL e' l'approssimazione dell'integrale di FUN
% sull'intervallo (A,B) calcolata con la formula com-
% posita di Gauss-Legendre di grado 1 su M intervallini
y = [-1/sqrt(3),1/sqrt(3)];
H2 = (b-a)/(2*M);
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 = fun(x,varargin{:});
intgl = H2*sum(f);
return
```

Soluzione 4.10 Dalla (4.25) 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

$$I(f) = I_1 + CH_1^2, \quad I(f) = I_2 + CH_2^2, \quad (10.4)$$

possiamo calcolare la costante C come

$$C = \frac{I_1 - I_2}{H_2^2 - H_1^2}$$

e, sostituendo tale valore in una delle due uguaglianze di (10.4), troviamo la (4.43).

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 Abbiamo:

$$f_1^{(4)}(x) = 24 \frac{1 - 10(x - \pi)^2 + 5(x - \pi)^4}{(1 + (x - \pi)^2)^5},$$

$$f_2^{(4)}(x) = -4e^x \cos(x).$$

Pertanto il massimo di $|f_1^{(4)}(x)|$ è limitato da $M_1 \simeq 23$, quello di $|f_2^{(4)}(x)|$ da $M_2 \simeq 18$. Di conseguenza, per la (4.29) si trova nel primo caso $H < 0.21$ e nel secondo $H < 0.16$.

Soluzione 4.13 Con i comandi **MAT&OCT**:

```
syms x
I=int(exp(-x^2/2),0,2);
Iex=double(I)
```

otteniamo che l'integrale in questione vale circa 1.19628801332261. Il comando **double** (*I*) di **MAT&OCT** restituisce il valore reale della variabile simbolica *I*. Il calcolo con la formula di Gauss-Legendre implementata nel Programma 10.2 (con $M = 1$) fornisce il valore 1.20278027622354 (con un errore assoluto pari a 6.4923e-03), mentre per la formula di Simpson semplice 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 **MAT&OCT**:

```
I(1)=1/exp(1); for k=2:20, I(k)=1-k*I(k-1); end
```

In MATLAB si ottiene $I(20) = 104.86$, mentre in Octave $I(20) = -30.1924$. Utilizzando la formula di Simpson con $H < 0.0625$ si ottiene l'accuratezza richiesta, infatti, denotando con $f(x)$ la funzione integranda, il valore assoluto della sua derivata quarta è limitato da $M \simeq 1.46 \cdot 10^5$. Di conseguenza, da (4.29) ricaviamo $H < 0.066$.

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 (4.43) 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:

$$I_1 = 1.19616568040561, \quad I_2 = 1.19628173356793,$$

$$I_R = 1.19628947044542,$$

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 Dobbiamo approssimare con la formula di Simpson composita i valori $j(r, 0) = \sigma/(\varepsilon_0 r^2) \int_0^r f(\xi) d\xi$ con $r = k/10$, per $k = 1, \dots, 10$ e $f(\xi) = e^\xi \xi^2$. Per stimare l'errore dobbiamo calcolare la derivata quarta della funzione integranda. Si trova $f^{(4)}(\xi) = e^\xi (\xi^2 + 8\xi + 12)$. 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 $H^4 < 10^{-10} 2880/(rf^{(4)}(r))$. 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:

```
r=[0.1:0.1:1]; maxf4=exp(r).*(r.^2+8*r+12);  
H=(10^(-10)*2880./(r.*maxf4)).^(1/4); M=fix(r./H)
```

```
M =  
4 11 20 30 41 53 67 83 100 118
```

I valori di $j(r, 0)$ sono allora ottenuti con i seguenti comandi:

```
sigma=0.36; epsilon0 = 8.859e-12;  
f=@(x) exp(x).*x.^2;  
for k = 1:10  
    r = k/10;  
    j(k)=simpsonc(f,0,r,M(k));  
    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=@(x) 1./(x.^5.*(exp(1.432./(213*x))-1));  
a=3.e-04; b=14.e-04;  
i=1; err = 1; Iold = 0; while err >= 1.e-11  
I=2.39e-11*simpsonc(f,a,b,i);  
err = abs(I-Iold)/abs(I);  
Iold=I;  
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. Si osserva che se avessimo utilizzato la formula composita dei trapezi sarebbero stati necessari 1609 punti.

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 2 in ogni sotto intervallo).

Soluzione 4.19 Anzitutto riportiamo l'area della sezione in metri: $A = 10^{-4}$ m. La funzione integranda $\lambda(x) = 10^{-4} \sin(x)/x$ non è definita nell'estremo sinistro dell'intervallo di integrazione $x = 0$. Una prima possibilità per integrarla su $[0, L]$ consiste nell'utilizzare una qualsiasi formula di quadratura aperta (cioè tale che i nodi di quadratura siano tutti interni all'intervallo di integrazione), come ad esempio la formula del punto medio composita. In alternativa, sapendo che $\lim_{x \rightarrow 0^+} \frac{\sin x}{x} = 1$, possiamo estendere la funzione $\lambda(x)$ ad una funzione $\tilde{\lambda}(x)$ tale che $\tilde{\lambda}(0) = 10^{-4}$ e $\tilde{\lambda}(x) = \lambda(x)$ per $x > 0$. Optiamo per la prima scelta e cominciamo col calcolare la massa dell'asta. Per garantire che l'errore di quadratura sia minore di 10^{-8} sfruttiamo la formula (4.21) e determiniamo il massimo H per cui

$$\left| \int_0^2 \lambda(x) dx - I_{pm}^c(\lambda) \right| \leq \frac{L}{24} H^2 \max_{0 \leq x \leq L} |\lambda''(x)| < 10^{-8}.$$

Abbiamo $\lambda''(x) = -10^{-4}(x^2 \sin(x) - 2 \sin(x) + 2x \cos(x))/x^3$ e da una sua rappresentazione grafica deduciamo che $\max_{0 \leq x \leq 1} |\lambda''(x)|$ è assunto in $x = 0$ e vale $A/3$. Quindi, se $H \leq \sqrt{24 \cdot 10^{-8}/(1/3)} \simeq 8.4852 \cdot 10^{-2}$, l'errore di quadratura nel calcolo della massa sarà minore di 10^{-8} . Il numero di intervalli da utilizzare nella formula composita è quindi $M = [2/H] + 1$. Per approssimare il valore della massa con punto medio composito utilizziamo le seguenti istruzioni **MAT&OCT**:

```
a=0; b=1; A=1.e-4;
lambda=@(x) A*sin(x)./x;
max12=A/3; epsi=1.e-8;
hmax=sqrt(24*epsi/(b-a)/maxf2);
M=fix((b-a)/hmax)+1
massa=midpointc(lambda,a,b,M);
```

Otteniamo $\text{massa}=9.461702244627381\text{e-}05$ kg. Per calcolare il centro di massa, basta osservare che $\int_0^L x \lambda(x) dx = \int_0^L \sin(x) dx = -\cos(L) + 1$. Con l'istruzione **centro=(1-cos(1))/massa*A**

otteniamo $\bar{x} = 4.858509412435690\text{e-}01$ m.

Soluzione 4.20 Essendo la misura del semiasse maggiore nota in milioni km, richiedere precisione di 10^4 km equivale a chiedere che l'errore di quadratura sia al più $\varepsilon = 10^4$. Per garantire ciò sfruttiamo la formula (4.25) e determiniamo il massimo H per cui

$$\left| \int_0^{2\pi} f(t) dt - I_T^c(f) \right| \leq \frac{2\pi}{12} H^2 \max_{0 \leq t \leq 2\pi} |f''(t)| < 10^4.$$

La funzione integranda è $f(t) = \sqrt{a^2 \cos^2(t) + b^2 \sin^2(t)}$, calcoliamo la derivata seconda in **MAT&OCT** sfruttando il calcolo simbolico (si veda la Sezione 1.6.3). Le istruzioni sono:

```
a=149.60e6; e=.0167086; b=a*sqrt(1-e^2);
syms t;
f=sqrt(a^2*cos(t).^2+b^2*sin(t).^2);
f2=diff(f,2);
ff2=matlabFunction(f2); % in Matlab
% ff2=function_handle(f2); %in Octave
tt=linspace(0,2*pi,10000); ff2y=ff2(tt);
f2max=max(ff2y);
```

Si noti che prima abbiamo definito la funzione f come variabile simbolica, l'abbiamo derivata simbolicamente ($f2$) e poi abbiamo convertito quest'ultima in un *function handle* ($ff2$) per valutarne il massimo su un insieme di 10000 punti equispaziati tra 0 e 2π . Otteniamo $f2max=4.177075523722157e+04$. Quindi, isolando H dalla precedente formula, otteniamo che se

$$H \leq \sqrt{12 \cdot 10^4 / (2\pi \cdot f2max)} \simeq 0.67618,$$

l'errore soddisfa la precisione richiesta. Il numero corrispondente di intervalli è $M = 10$. La lunghezza calcolata dell'orbita terrestre è $L = 9.398989 \cdot 10^8$ km. Per calcolarla abbiamo eseguito le istruzioni:

```
f2max=4.177054688795780e+04
H=sqrt(6/(pi*f2max)*1.e4)
M=fix(2*pi/H)+1
t=linspace(0,2*pi,M);
ff=matlabFunction(f); % in Matlab
% ff=function_handle(f); %in Octave
y=ff(t); I=trapz(t,y)
```

Soluzione 4.21 La formula di quadratura (4.44) è di tipo interpolatorio su 4 nodi di quadratura $y_0 = a$, $y_1 = (2a + b)/3$, $y_2 = (a + 2b)/3$, $y_3 = b$ e pesi $\alpha = 0 = \alpha_3 = (b - a)/8$, $\alpha_1 = \alpha_2 = 3(b - a)/8$. Per determinare l'ordine di accuratezza, consideriamo $f(x) = x^k$ con $k = 0, \dots, 4$, sappiamo che gli integrali esatti sono $\int_a^b x^k dx = (b^{k+1} - a^{k+1})/(k+1)$. Dobbiamo verificare che $\tilde{I}(x^k) = \int_a^b x^k dx$ per $k = 0, 1, 2, 3$ e che $\tilde{I}(x^4) \neq \int_a^b x^4 dx$. La verifica con $k = 0$ è immediata perché basta osservare che la somma dei pesi di quadratura è pari a $(b - a)$. Per gli altri valori di k , procediamo col valutare $\tilde{I}(f)$:

$$\tilde{I}(x) = \frac{b-a}{8} \left(a + 3\frac{2a+b}{3} + 3\frac{a+2b}{3} + b \right) = \frac{b^2 - a^2}{2}$$

$$\tilde{I}(x^2) = \frac{b-a}{8} \left(a^2 + 3\frac{(2a+b)^2}{9} + 3\frac{(a+2b)^2}{9} + b^2 \right) = \frac{b^3 - a^3}{3}$$

$$\tilde{I}(x^3) = \frac{b-a}{8} \left(a^3 + 3\frac{(2a+b)^3}{27} + 3\frac{(a+2b)^3}{27} + b^3 \right) = \frac{b^4 - a^4}{4}$$

$$\tilde{I}(x^4) = \frac{b-a}{8} \left(a^4 + 3\frac{(2a+b)^4}{81} + 3\frac{(a+2b)^4}{81} + b^4 \right) = \frac{b^5 - a^5}{5} + \frac{(b-a)^5}{270}$$

Quindi la formula ha grado di esattezza 3 perché integra esattamente tutti i polinomi di grado ≤ 3 e non tutti i polinomi di grado maggiore.

Seguendo le notazioni introdotte nella Sezione 4.3.1, la formula di quadratura composita è

$$\tilde{I}^c(f) = \sum_{k=1}^M \frac{x_{k+1} - x_k}{8} [f(x_k) + 3f(\bar{x}_k) + 3f(\hat{x}_k) + f(x_{k+1})]$$

con $\bar{x}_k = (2x_k + x_{k+1})/3$ e $\hat{x}_k = (x_k + 2x_{k+1})/3$. La function `MAT&OCT` che approssima l'integrale con questa formula è:

```
function [s]=fdq(a,b,M,f)
x=linspace(a,b,M+1)';
H=(b-a)/M;
w=[1 3 3 1];
s=0;
for k=1:M
y=[x(k);(2*x(k)+x(k+1))/3;(2*x(k+1)+x(k))/3;x(k+1)];
s=s+w*f(y);
end
s=s*H/8;
```

Per testarne l'ordine di accuratezza rispetto ad H integriamo la funzione $f(x) = x^5$ sull'intervallo $[a, b] = [0, 1]$ (sappiamo che l'integrale esatto è $I = 1/6$) considerando diversi valori di $M = 10, 20, 40, 80, 160, 320$, valutiamo gli errori tra gli integrali approssimati e quello esatto e li memorizziamo nel vettore `e` usando le seguenti istruzioni `MAT&OCT`:

```
Iex=1/6; f=@(x)x.^5; a=0; b=1;
Mv=[10,20,40,80,160,320]; e=[];
for M=Mv
s=fdq(a,b,M,f);
err=abs(s-Iex); e=[e, err];
end
H=(b-a)./Mv;
```

Per determinare l'ordine p di accuratezza dell'errore rispetto ad H , sfruttiamo la formula (1.12), la cui codifica `MAT&OCT` è:

```
P=log(e(2:end)./e(1:end-1))./log(H(2:end)./H(1:end-1))
```

Otteniamo:

<code>P =</code>	4.0000	4.0000	4.0000	4.0000	3.9999
------------------	--------	--------	--------	--------	--------

e l'ordine di accuratezza della formula data è $p = 4$.

10.5 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. Vale la seguente formula ricorsiva

$$x_k - kx_{k-1} = 2k - 1, \quad k \geq 2,$$

avendo posto $x_1 = 0$. Dividendo entrambi i termini dell'equazione per $k!$ si ha

$$\frac{x_k}{k!} - \frac{x_{k-1}}{(k-1)!} = \frac{2k-1}{k!}$$

e sommando su k da 2 a n troviamo

$$x_n = n! \sum_{k=2}^n \frac{2k-1}{k!}.$$

Ricordando che $\sum_{k=0}^{\infty} \frac{1}{k!} = e$, si ha

$$\sum_{k=2}^n \frac{2k-1}{k!} = 2 \sum_{k=1}^{n-1} \frac{1}{k!} - \sum_{k=2}^n \frac{1}{k!} \simeq 2.718,$$

e $x_n \simeq 3n!$. Si osserva a questo punto che per risolvere un sistema lineare quadrato con matrice piena di dimensione n con il metodo di Cramer (si veda la Sezione 5.2) servono in tutto circa $3(n+1)!$ operazioni elementari.

Soluzione 5.2 Utilizziamo i seguenti comandi MATLAB per calcolare i determinanti ed i tempi di CPU necessari:

```
t = [] ; NN=3:500;
for n = NN
A=magic(n); 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 $NN=[3:500]$ e t . Troviamo:

```
c=polyfit(NN,t,3)
c =
1.4055e-10    7.1570e-08   -3.6686e-06   3.1897e-04
```

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(NN,t,4)
c =
7.6406e-15    1.3286e-10   7.4064e-08   -3.9505e-06   3.2637e-04
```

ovvero il coefficiente di n^4 è vicino alla precisione di macchina, mentre gli altri sono quasi invariati rispetto ai coefficienti della proiezione su \mathbb{P}_3 . In base a questo risultato possiamo concludere che il tempo di CPU richiesto da MATLAB per calcolare il determinante di una matrice di dimensione n si comporta come n^3 .

Soluzione 5.3 Denotando con A_i le sottomatrici principali di A di ordine i , 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 fattorizzazione di Gauss di A non esiste (si veda la Proposizione 5.1). La matrice A è singolare se $\varepsilon = -6$: in tal caso la fattorizzazione di Gauss può comunque essere portata a termine e si trova

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 1.25 & 1 \end{bmatrix}, \quad U = \begin{bmatrix} 1 & 7 & 3 \\ 0 & -12 & -4 \\ 0 & 0 & 0 \end{bmatrix}.$$

Tuttavia, poiché U è singolare (com'era del resto da attendersi essendo A singolare), il sistema triangolare superiore $U\mathbf{x} = \mathbf{y}$ ammette infinite soluzioni.

Si osservi anche che il metodo delle sostituzioni all'indietro (5.10) non può essere utilizzato.

Soluzione 5.4 Consideriamo l'algoritmo (5.13). Al passo $k = 1$, servono $n - 1$ divisioni per calcolare i valori l_{i1} , per $i = 2, \dots, n$. Quindi servono $(n - 1)^2$ moltiplicazioni e $(n - 1)^2$ addizioni per costruire i valori $a_{ij}^{(2)}$, per $i, j = 2, \dots, n$. Al passo $k = 2$, il numero di divisioni è $(n - 2)$, mentre il numero di prodotti ed addizioni sarà $(n - 2)^2$. All'ultimo passo $k = n - 1$ è richiesta solo una addizione, una moltiplicazione e una divisione. Quindi, grazie alle identità

$$\sum_{s=1}^q s = \frac{q(q+1)}{2}, \quad \sum_{s=1}^q s^2 = \frac{q(q+1)(2q+1)}{6}, \quad q \geq 1,$$

possiamo concludere che per realizzare la fattorizzazione di Gauss serve il seguente numero di operazioni

$$\begin{aligned} \sum_{k=1}^{n-1} \sum_{i=k+1}^n \left(1 + \sum_{j=k+1}^n 2 \right) &= \sum_{k=1}^{n-1} (n-k)(1+2(n-k)) \\ &= \sum_{j=1}^{n-1} j + 2 \sum_{j=1}^{n-1} j^2 = \frac{(n-1)n}{2} + 2 \frac{(n-1)n(2n-1)}{6} = \frac{2}{3}n^3 - \frac{n^2}{2} - \frac{n}{6}. \end{aligned}$$

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 \mathbf{x}_j di X risolve il sistema lineare $A\mathbf{x}_j = \mathbf{e}_j$ 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 5.1 si trovano i seguenti fattori:

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & -3.38 \cdot 10^{15} & 1 \end{bmatrix}, \quad U = \begin{bmatrix} 1 & 1 & 3 \\ 0 & -8.88 \cdot 10^{-16} & 14 \\ 0 & 0 & 4.73 \cdot 10^{16} \end{bmatrix},$$

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. Più in generale, una strategia di *pivoting* che coinvolge lo scambio di righe e colonne è chiamata *pivoting totale* (si veda la Sezione 5.4).

Soluzione 5.8 Il calcolo simbolico dei fattori L e U della matrice A fornisce

$$L = \begin{bmatrix} 1 & 0 & 0 \\ (\varepsilon - 2)/2 & 1 & 0 \\ 0 & -1/\varepsilon & 1 \end{bmatrix}, \quad U = \begin{bmatrix} 2 & -2 & 0 \\ 0 & \varepsilon & 0 \\ 0 & 0 & 3 \end{bmatrix},$$

cosicché $l_{32} \rightarrow \infty$, quando $\varepsilon \rightarrow 0$. Se noi scegliamo $\mathbf{b} = (0, \varepsilon, 2)^T$, è facile verificare che $\mathbf{x} = (1, 1, 1)^T$ è la soluzione esatta di $A\mathbf{x} = \mathbf{b}$. Per analizzare l'errore rispetto alla soluzione esatta per $\varepsilon \rightarrow 0$, prendiamo $\varepsilon = 10^{-k}$, per $k = 0, \dots, 9$. Le seguenti istruzioni:

```
e=1;xex=ones(3,1);err=[];
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)=norm(x-xex)/norm(xex); e=e*0.1;
end
```

producono

```
err =
0 0 0 0 0 0 0 0 0 0
```

cioè, la soluzione numerica non è affetta da errori di arrotondamento. Questo fatto può essere spiegato notando che tutti gli elementi di L, U e \mathbf{b} sono numeri *floating-point* non affetti da errori di arrotondamento e di conseguenza, in maniera molto insolita, gli errori di arrotondamento non vengono propagati durante le risoluzioni in avanti e all'indietro, anche se il numero di condizionamento di A è proporzionale a $1/\varepsilon$.

Al contrario, ponendo $\mathbf{b} = (2 \log(2.5) - 2, (\varepsilon - 2) \log(2.5) + 2, 2)^T$, a cui corrisponde la soluzione esatta $\mathbf{x} = (\log(2.5), 1, 1)^T$, ed analizzando gli errori relativi con $\varepsilon = 1/3 \cdot 10^{-k}$, per $k = 0, \dots, 9$, le istruzioni:

```
e=1/3; xex=[log(5/2),1,1]'; err=[];
for k=1:10
b=[2*log(5/2)-2,(e-2)*log(5/2)+2,2]';
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)=norm(x-xex)/norm(xex); e=e*0.1;
end
```

producono

```
err =
Columns 1 through 5
1.8635e-16 5.5327e-15 2.6995e-14 9.5058e-14 1.3408e-12
Columns 6 through 10
1.2828e-11 4.8726e-11 4.5719e-09 4.2624e-08 2.8673e-07
```

Nell'ultimo caso gli errori dipendono dal numero di condizionamento di A (che obbedisce alla legge $K(A) = C/\varepsilon$) e quindi soddisfano la stima a priori (5.34).

Soluzione 5.9 La soluzione calcolata diventa sempre meno accurata al crescere del pedice i . Gli errori in norma sono infatti pari a $1.10 \cdot 10^{-14}$ per $i = 1$, $9.32 \cdot 10^{-10}$ per $i = 2$ e $2.51 \cdot 10^{-7}$ per $i = 3$. (Mettiamo in guardia il lettore che questi risultati dipendono fortemente dalla versione di MATLAB o Octave utilizzate!!) 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 $B(\omega)$, abbiamo

$$\begin{aligned} \left| \prod_{i=1}^n \lambda_i \right| &= |\det B(\omega)| \\ &= |\det[(I - \omega D^{-1}E)^{-1}]| \cdot |\det[(1 - \omega)I + \omega D^{-1}F]|. \end{aligned}$$

Osservando ora che, date due matrici A e B con $A = I + \alpha B$, per ogni $\alpha \in \mathbb{R}$ vale $\lambda_i(A) = 1 + \alpha \lambda_i(B)$, e che gli autovalori di $D^{-1}E$ e di $D^{-1}F$ sono tutti nulli,abbiamo

$$\left| \prod_{i=1}^n \lambda_i \right| = \left| \prod_{i=1}^n \frac{(1 - \omega) + \omega \lambda_i(D^{-1}F)}{1 - \omega \lambda_i(D^{-1}E)} \right| = |1 - \omega|^n.$$

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 $A = \begin{bmatrix} 3 & 2 \\ 2 & 6 \end{bmatrix}$ è a dominanza diagonale stretta per righe, una condizione sufficiente affinché il metodo di Gauss Seidel converga. La matrice $A = \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix}$ non è a dominanza stretta per righe, tuttavia è simmetrica e quindi verifichiamo se essa è anche definita positiva, ovvero che $\mathbf{z}^T A \mathbf{z} > 0$ per ogni $\mathbf{z} \neq \mathbf{0}$ di \mathbb{R}^2 . Eseguiamo i calcoli con **MAT&OCT** nel modo seguente (naturalmente in questo caso li potremmo fare anche a mano!):

```
syms z1 z2
z=[z1;z2]; A=[1 1; 1 2];
pos=z'*A*z;
simplify(pos)
ans =
z1^2+2*z1*z2+2*z2^2
```

dove il comando **simplify** 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 $(z_1+z_2)^2+2z_2^2$. La matrice è dunque simmetrica definita positiva ed è quindi una condizione sufficiente affinché il metodo di Gauss-Seidel converga.

Soluzione 5.15 Si trova:

per il metodo di Jacobi:

$$\begin{cases} x_1^{(1)} = \frac{1}{2}(1 - x_2^{(0)}) \\ x_2^{(1)} = -\frac{1}{3}(x_1^{(0)}) \end{cases} \Rightarrow \begin{cases} x_1^{(1)} = \frac{1}{4} \\ x_2^{(1)} = -\frac{1}{3} \end{cases}$$

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

$$\mathbf{r}^{(0)} = \mathbf{b} - \mathbf{Ax}^{(0)} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} - \begin{bmatrix} 2 & 1 \\ 1 & 3 \end{bmatrix} \mathbf{x}^{(0)} = \begin{bmatrix} -3/2 \\ -5/2 \end{bmatrix}.$$

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,

$$\alpha_0 = \frac{(\mathbf{z}^{(0)})^T \mathbf{r}^{(0)}}{(\mathbf{z}^{(0)})^T \mathbf{A} \mathbf{z}^{(0)}} = \frac{77}{107},$$

e

$$\mathbf{x}^{(1)} = \mathbf{x}^{(0)} + \alpha_0 \mathbf{z}^{(0)} = (197/428, -32/321)^T.$$

Soluzione 5.16 Gli autovalori della matrice $\mathbf{B}_\alpha = \mathbf{I} - \alpha \mathbf{P}^{-1} \mathbf{A}$ sono $\lambda_i(\alpha) = 1 - \alpha \mu_i$, essendo μ_i l' i -esimo autovalore di $\mathbf{P}^{-1} \mathbf{A}$. Allora

$$\rho(\mathbf{B}_\alpha) = \max_{i=1,\dots,n} |1 - \alpha \mu_i| = \max(|1 - \alpha \mu_{\min}|, |1 - \alpha \mu_{\max}|).$$

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 (5.60). A questo punto la (5.104) si trova calcolando $\rho(\mathbf{B}_{\alpha_{opt}})$.

Soluzione 5.17 Per ogni matrice \mathbf{P} simmetrica definita positiva, la matrice simmetrica e definita positiva $\mathbf{P}^{1/2}$ è l'unica soluzione dell'equazione matriciale $\mathbf{X}^2 = \mathbf{P}$ ed è la cosiddetta radice quadrata di \mathbf{P} (si veda, ad esempio [QV94, Sect. 2.5]). Indicato con $\mathbf{P}^{-1/2}$ l'inversa di $\mathbf{P}^{1/2}$, si osserva che $\mathbf{P}^{-1} \mathbf{A} = \mathbf{P}^{-1/2} (\mathbf{P}^{-1/2} \mathbf{A} \mathbf{P}^{-1/2}) \mathbf{P}^{1/2}$, quindi $\mathbf{P}^{-1} \mathbf{A}$ è simile alla matrice $\tilde{\mathbf{A}} = \mathbf{P}^{-1/2} \mathbf{A} \mathbf{P}^{1/2}$. Dobbiamo allora dimostrare che $\tilde{\mathbf{A}}$ è simmetrica e definita positiva.

La simmetria di $\tilde{\mathbf{A}}$ segue immediatamente dalla simmetria di \mathbf{A} e di $\mathbf{P}^{1/2}$.

Per dimostrare che $\tilde{\mathbf{A}}$ è definita positiva denotiamo con μ un generico autovalore di $\tilde{\mathbf{A}}$ e con $\mathbf{y} (\neq \mathbf{0})$ un autovettore associato a μ , per cui abbiamo $\tilde{\mathbf{A}}\mathbf{y} = \mu\mathbf{y}$. Premoltiplicando per $\mathbf{P}^{1/2}$ entrambe i membri di $\tilde{\mathbf{A}}\mathbf{y} = \mu\mathbf{y}$ e introducendo il vettore $\tilde{\mathbf{y}} = \mathbf{P}^{-1/2}\mathbf{y}$,abbiamo $\tilde{\mathbf{A}}\tilde{\mathbf{y}} = \mu\mathbf{P}\tilde{\mathbf{y}}$, da cui segue $\tilde{\mathbf{y}}^T \tilde{\mathbf{A}} \tilde{\mathbf{y}} = \mu \tilde{\mathbf{y}}^T \mathbf{P} \tilde{\mathbf{y}}$. Poiché sia \mathbf{A} che \mathbf{P} sono simmetriche definite positive e $\tilde{\mathbf{y}} \neq \mathbf{0}$ otteniamo $\mu = (\tilde{\mathbf{y}}^T \tilde{\mathbf{A}} \tilde{\mathbf{y}}) / (\tilde{\mathbf{y}}^T \mathbf{P} \tilde{\mathbf{y}}) > 0$. Dall'arbitrarietà di μ segue che tutti gli autovalori di $\tilde{\mathbf{A}}$ sono strettamente positivi e quindi $\tilde{\mathbf{A}}$ è anche e definita positiva.

Soluzione 5.18 Cominciamo con il minimizzare la funzione $\tilde{\varphi}(\alpha) = \Phi(\mathbf{x}^{(k)} + \alpha \mathbf{d}^{(k)})$ al variare di $\alpha \in \mathbb{R}$, abbiamo

$$\begin{aligned} \tilde{\varphi}(\alpha) &= \frac{1}{2} (\mathbf{x}^{(k)} + \alpha \mathbf{d}^{(k)})^T \mathbf{A} (\mathbf{x}^{(k)} + \alpha \mathbf{d}^{(k)}) - (\mathbf{x}^{(k)} + \alpha \mathbf{d}^{(k)})^T \mathbf{b} \\ &= \frac{1}{2} (\mathbf{x}^{(k)})^T (\mathbf{A} \mathbf{x}^{(k)} - \mathbf{b}) + \alpha (\mathbf{d}^{(k)})^T (\mathbf{A} \mathbf{x}^{(k)} - \mathbf{b}) + \frac{\alpha^2}{2} (\mathbf{d}^{(k)})^T \mathbf{A} \mathbf{d}^{(k)}. \end{aligned}$$

Il minimo di $\tilde{\varphi}(\alpha)$ si trova in corrispondenza di α_k tale che $\tilde{\varphi}'(\alpha_k) = 0$, ovvero

$$-(\mathbf{d}^{(k)})^T \mathbf{r}^{(k)} + \alpha_k (\mathbf{d}^{(k)})^T \mathbf{A} \mathbf{d}^{(k)} = 0,$$

dunque (5.70) è soddisfatta.

Minimizziamo ora la funzione $\varphi(\alpha) = \|\mathbf{e}^{(k+1)}\|_A^2$ al variare di $\alpha \in \mathbb{R}$. Poiché $\mathbf{e}^{(k+1)} = \mathbf{x} - \mathbf{x}^{(k+1)} = \mathbf{e}^{(k)} - \alpha \mathbf{d}^{(k)}$, abbiamo

$$\varphi(\alpha) = \|\mathbf{e}^{(k+1)}\|_A^2 = \|\mathbf{e}^{(k)}\|_A^2 + \alpha^2 \|\mathbf{d}^{(k)}\|_A^2 - 2\alpha (\mathbf{A}\mathbf{e}^{(k)}, \mathbf{d}^{(k)}).$$

Il minimo di $\varphi(\alpha)$ si trova in corrispondenza di α_k tale che $\varphi'(\alpha_k) = 0$, ovvero

$$\alpha_k \|\mathbf{d}^{(k)}\|_A^2 - (\mathbf{A}\mathbf{e}^{(k)}, \mathbf{d}^{(k)}) = 0,$$

dunque $\alpha_k = (\mathbf{A}\mathbf{e}^{(k)}, \mathbf{d}^{(k)})/\|\mathbf{d}^{(k)}\|_A^2$. Infine la formula (5.70) segue osservando che $\mathbf{A}\mathbf{e}^{(k)} = \mathbf{r}^{(k)}$.

Soluzione 5.19 Sfruttando la relazione $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k \mathbf{d}^{(k)}$ e la formula (5.70) per α_k , il vettore residuo al passo $(k+1)$ è

$$\mathbf{r}^{(k+1)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(k+1)} = \mathbf{b} - \mathbf{A}(\mathbf{x}^{(k)} + \alpha_k \mathbf{d}^{(k)}) = \mathbf{r}^{(k)} - \frac{(\mathbf{d}^{(k)})^T \mathbf{r}^{(k)}}{(\mathbf{d}^{(k)})^T \mathbf{A} \mathbf{d}^{(k)}} \mathbf{A} \mathbf{d}^{(k)},$$

quindi

$$(\mathbf{d}^{(k)})^T \mathbf{r}^{(k+1)} = (\mathbf{d}^{(k)})^T \mathbf{r}^{(k)} - \frac{(\mathbf{d}^{(k)})^T \mathbf{r}^{(k)}}{(\mathbf{d}^{(k)})^T \mathbf{A} \mathbf{d}^{(k)}} (\mathbf{d}^{(k)})^T \mathbf{A} \mathbf{d}^{(k)} = 0.$$

Soluzione 5.20 La matrice A del modello di Leontieff è simmetrica, ma 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)), max(eig(A))]
```

```
ans =
-448.5830    30.5830
```

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 $A^T A x = A^T b$ 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 $x^{(0)} = \mathbf{0}^T$:

```
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 .

Soluzione 5.21 Il metodo iterativo (5.105) può essere riscritto nella forma (5.51) con $B = (I - \alpha A)$ e $\mathbf{g} = \alpha \mathbf{b}$. Il metodo converge se e solo se $\rho(B) < 1$. Poiché

$$\rho(B) = \max_i |\lambda_i(B)| = \max_i |\lambda_i(I - \alpha A)| = \max_i |1 - \alpha \lambda_i(A)|,$$

la convergenza è garantita se $|1 - \alpha \lambda_i(A)| < 1$ per ogni $\lambda_i(A)$. Calcoliamo gli autovalori della matrice A con il comando `eig` di MATLAB:

```
eig(A)
ans =
    2.3747
    3.7935
    8.0434
   15.7884
```

Poiché gli autovalori sono tutti reali e positivi, la diseguaglianza richiesta è soddisfatta se $0 < \alpha < 2/\lambda_i(A)$ per ogni $i = 1, \dots, 4$ e quindi in particolare per $0 < \alpha < 2/15.7884 \simeq 0.1267$. Di fatto, il metodo proposto è il metodo di Richardson stazionario con $P=I$. Il Programma 10.3 implementa il metodo (5.105): B è la matrice di iterazione, g contiene il vettore \mathbf{g} , x0 il vettore iniziale, kmax il numero massimo di iterazioni e tol la tolleranza per il test d'arresto sull'incremento. In output: x è il vettore soluzione, niter il numero di iterazioni effettuate, e il vettore delle norme degli incrementi $\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\|$.

Programma 10.3. `itermethb`: metodo iterativo per sistemi lineari

```
function [x,iter,e]= itermethb(B,g,x0,tol,kmax)
% ITERMETHB metodo iterativo classico per sist. lineari
% [X,ITER,E]= ITERMETHB(B,G,X0,TOL,KMAX)
% Calcola la soluzione del metodo iterativo
% x^{k+1}=B x^k+G, con x^0 assegnato, per k=0,1....
% B e' la matrice di iterazione, il vettore iniziale
% e' memorizzato in X0. TOL e KMAX sono tolleranza e
% numero massimo di iterazioni per il test d'arresto
% sull'incremento. La soluzione e' memorizzata in X,
% ITER e' il numero di iterazioni richieste per
% giungere a convergenza.
% E e' il vettore degli errori ||x^{k+1}-x^k||.
k=0;
err=tol+1;
e=[];
while err> tol && k< kmax
    x=B*x0+g;
    err=norm(x-x0);
    e=[e;err];
    x0=x;
    k=k+1;
end
iter=k;
```

Con le seguenti istruzioni MATLAB:

```
A=[7 1 2 4; 1 10 4 3; 2 4 6 2; 4 3 2 7];
b=[14; 18; 14; 17];
x0=rand(4,1); tol=1.e-12; kmax=100;
for alpha=0.1:0.01:0.13
B=eye(4)-alpha*A; g=alpha*b;
```

```
[x,iter,e]= itermethb(B,g,x0,tol,kmax);
fprintf('alpha=%f iter=%d\n',alpha,iter);
end
```

possiamo verificare quanto trovato, in particolare con $\alpha = 0.1$ si raggiunge convergenza in 83 iterazioni, con $\alpha = 0.11$ in 93 iterazioni, con $\alpha = 0.12$ in 253 iterazioni. Infine con $\alpha = 0.13$ il metodo diverge.

Soluzione 5.22 Per stimare l'errore tra la soluzione del sistema perturbato e quella del sistema originario applichiamo la stima (5.34), osservando che la perturbazione δA è nulla e $\delta b = [10^{-3}, 0, \dots, 0]^T$, otteniamo

$$\frac{\|x - \hat{x}\|}{\|x\|} \leq K_2(A) \frac{\|\delta b\|}{\|b\|}.$$

Eseguendo i seguenti comandi **MAT&OCT**:

```
A=[ 1 1 0; 1 1 4; 500 -1.5 -0.002];
b=[ 9; 5 ;500]; n=3;
bp=b+1.e-3*ones(n,1); deltab=bp-b;
ka=cond(full(A)); stima=ka*norm(deltab)/norm(b)
```

otteniamo $\|x - \hat{x}\|/\|x\| \leq 0.0018$, a fronte di un errore relativo sui dati $\|\delta b\|/\|b\| \simeq 3.46 \cdot 10^{-6}$, possiamo cioè perdere circa tre ordini di grandezza di precisione nel risolvere il sistema perturbato anziché quello originario. Per verificare questa stima, utilizziamo il comando \ di **MAT&OCT** che implementa, in questo caso, la fattorizzazione LU con pivotazione per righe:

```
x=A\b; xp=A\bp;
err=norm(xp-x)/norm(x)
```

Otteniamo un errore effettivo $\|x - \hat{x}\|/\|x\| \simeq 1.23 \cdot 10^{-4}$ (che soddisfa la stima a priori). La perdita di tre ordini di grandezza è da imputare all'alto numero di condizionamento della matrice A, abbiamo infatti **ka = 514.8966**.

La matrice non è simmetrica e pertanto non possiamo applicare la fattorizzazione di Cholesky. Usiamo pertanto la fattorizzazione LU. Poiché il minore principale di ordine 2 è singolare, siamo costretti ad applicare la pivotazione per righe affinché essa arrivi a terminazione (si veda la Proposizione 5.1). Tra i metodi iterativi visti, non possiamo applicare i metodi del gradiente e del gradiente coniugato (sempre perché la matrice non è simmetrica e definita positiva). La condizione sufficiente che garantisce la convergenza del metodo di Gauss-Seidel non è soddisfatta, quindi a priori non possiamo concludere nulla. Una risposta certa è data dalla valutazione del raggio spettrale $\rho(B)$ della matrice di iterazione di Gauss-Seidel, che si può ottenere grazie alle istruzioni:

```
Bgs=eye(n)-tril(A)\A;
rhogs=max(abs(eig(Bgs)))
```

Poiché **rhogs = 3.3037e+03 > 1**, il metodo di Gauss-Seidel non converge. Pe-raltro, avendo la matrice dimensione 3, essa può essere interpretata come una matrice tridiagonale e, grazie alla Proposizione 5.4, possiamo concludere che nemmeno il metodo di Jacobi converge. Fissando $x^{(0)} = \mathbf{0}$, numero massimo di iterazioni pari a 10, tolleranza per il test d'arresto pari alla precisione di macchina, possiamo richiamare il metodo Bi-CGStab, implementato in **MAT&OCT**, con la seguente istruzione:

```
[x,flag,relres,it,resvec]=bicgstab(A,b,eps,5);
```

La convergenza è raggiunta in 5 iterazioni. Benché il Bi-CGStab sia un metodo a terminazione finita in aritmetica esatta, qui otteniamo convergenza in un numero di iterazioni maggiore della dimensione del sistema, per effetto degli errori di arrotondamento, che sono amplificati dall'alto numero di condizionamento della matrice del sistema.

10.6 Capitolo 6

Soluzione 6.1 A₁: il metodo converge in 34 passi al valore 2.00000000004989. A₂: 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 A₃ 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

$$\mathbf{y}^{(0)} = \beta^{(0)} \left(\alpha_1 \mathbf{x}_1 + \alpha_2 \mathbf{x}_2 + \sum_{i=3}^n \alpha_i \mathbf{x}_i \right),$$

con $\beta^{(0)} = 1/\|\mathbf{x}^{(0)}\|$. Ripetendo i calcoli svolti nel paragrafo 6.2, al generico passo k avremo:

$$\mathbf{y}^{(k)} = \gamma^k \beta^{(k)} \left(\alpha_1 \mathbf{x}_1 e^{ik\vartheta} + \alpha_2 \mathbf{x}_2 e^{-ik\vartheta} + \sum_{i=3}^n \alpha_i \frac{\lambda_i^k}{\gamma^k} \mathbf{x}_i \right).$$

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 10.11). Questo perché esistono due autovalori distinti aventi modulo massimo uguale a 1.

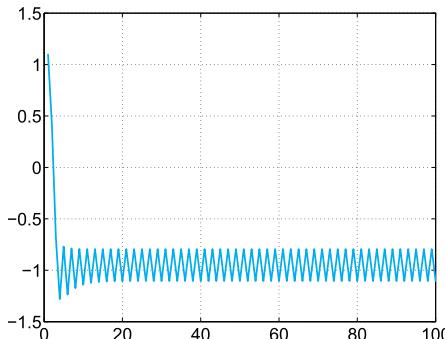


Figura 10.11. Approssimazioni dell'autovalore di modulo massimo calcolate dal metodo delle potenze al variare del numero di iterazioni per la matrice della Soluzione 6.5

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 6.1 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);
```

Dopo 35 iterazioni otteniamo $\text{lambdab}=3.76155718183189$. Poiché λ_a è l'autovalore più lontano da λ_b , per calcolare quest'ultimo applichiamo il metodo delle potenze alla matrice $A_b = A - \lambda_b I$, ovvero calcoliamo l'autovalore di modulo massimo della matrice A_b . Quindi porremo $\lambda_a = \lambda + \lambda_b$. Con le istruzioni:

```
[lambda,x,iter]=eigpower(A-lambdab*eye(7),tol,nmax,x0);
lambdaa=lambda+lambdab
```

troviamo $\text{lambdaa}=-1.12488541976457$ in 33 iterazioni. I risultati trovati sono delle ottime approssimazioni degli autovalori cercati.

Soluzione 6.7 Consideriamo la matrice A. Dall'esame dei cerchi riga vediamo che c'è un cerchio isolato di centro $(9, 0)$ 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 $(2, 0)$ e $(4, 0)$, rispettivamente (si veda la Figura 10.12, a destra). 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 10.13, a destra) deduciamo che c'è un solo cerchio isolato di centro $(-5, 0)$ 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

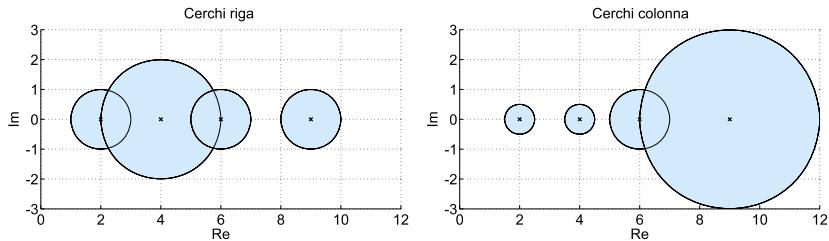


Figura 10.12. Cerchi riga (*a sinistra*) e colonna (*a destra*) per la matrice A della Soluzione 6.7

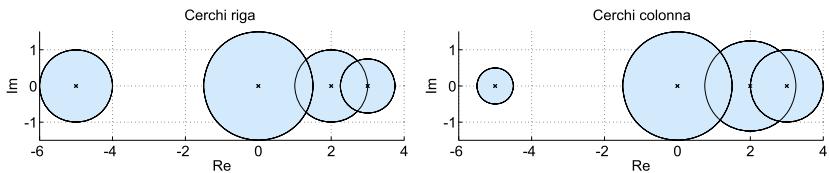


Figura 10.13. Cerchi riga (*a sinistra*) e colonna (*a destra*) per la matrice B della Soluzione 6.7

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 $(5, 0)$ 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=[1;2;3;4]; nmax=100;
tic; [lambda,x,iter]=eigpower(A,tol,nmax,x0);
toc, iter

Elapsed time is 0.001854 seconds.
iter =
35

tic; [lambda,x,iter]=invshift(A,5,tol,nmax,x0);
toc, iter

Elapsed time is 0.000865 seconds.
iter =
12
```

Come si vede il metodo delle potenze inverse con *shift* converge in un numero di iterazioni minore rispetto al metodo delle potenze e, pur richiedendo il calcolo della fattorizzazione LU di A in testa alla procedura, il tempo totale di esecuzione risulta inferiore a quello del metodo senza *shift*.

Soluzione 6.9 Abbiamo

$$A^{(k)} = Q^{(k+1)} R^{(k+1)} \quad \text{e} \quad A^{(k+1)} = R^{(k+1)} Q^{(k+1)}$$

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 Possiamo utilizzare il comando `[X,D]=eig(A)`, dove `X` è la matrice le cui colonne sono gli autovettori di norma unitaria di `A`, mentre `D` è la matrice diagonale i cui elementi sono gli autovalori di `A`. Per le matrici `A` e `B` dell'Esercizio 6.7 possiamo eseguire le seguenti istruzioni

```
A=[2 -1/2 0 -1/2; 0 4 0 2; -1/2 0 6 1/2; 0 0 1 9];
sort(eig(A))
ans =
    2.0000
    4.0268
    5.8003
    9.1728
B=[-5 0 1/2 1/2; 1/2 2 1/2 0; 0 1 0 1/2; 0 1/4 1/2 3];
sort(eig(B))
ans =
    -4.9921
    -0.3038
    2.1666
    3.1292
```

Le conclusioni dedotte in base alla Proposizione 6.1 sono abbastanza inaccurate.

10.7 Capitolo 7

Soluzione 7.1 Rappresentando il grafico della funzione f osserviamo che essa ammette un punto di minimo nell'intervallo $[-2, 1]$. Richiamiamo il Programma 7.1 con una tolleranza per il test d'arresto pari a 10^{-8} usando le istruzioni:

```
a=-2; b=1; tol=1.e-8; kmax=100;
[xmin,fmin,iter]=golden(f,a,b,tol,kmax)
```

Si ottiene $x_{\min}=-3.660253989004456e-01$ in 42 iterazioni ed il valore minimo della funzione è $f_{\min}=-1.194742596743503$. L'alto numero di iterazioni evidenzia la convergenza di tipo lineare del metodo (si veda (7.19)).

Richiamando il comando `fminbnd` di MATLAB con le istruzioni:

```
options=optimset('TolX',1.e-8);
[xminf,fminf,exitflag,output]=fminbnd(f,a,b,options)
```

risolviamo il medesimo problema con il metodo della sezione aurea con interpolazione quadratica ed otteniamo convergenza in 9 iterazioni al punto $x_{\min}=-3.660254076197302e-01$.

Soluzione 7.2 Date $\gamma_i(t) = (x_i(t), y_i(t))$, per $i = 1, 2$, dobbiamo minimizzare la distanza

$$d(t) = \sqrt{(x_1(t) - x_2(t))^2 + (y_1(t) - y_2(t))^2}$$

o equivalentemente il suo quadrato al variare del tempo t . Si tratta di risolvere un problema di minimo monodimensionale e possiamo utilizzare il metodo della sezione aurea con interpolazione quadratica implementato nella *function fminbnd*. Con le seguenti istruzioni:

```
x1=@(t)7*cos(t/3+pi/2)+5; y1=@(t)-4*sin(t/3+pi/2)-3;
x2=@(t)6*cos(t/6-pi/3)-4; y2=@(t)-6*sin(t/6-pi/3)+5;
d=@(t)(x1(t)-x2(t))^2+(y1(t)-y2(t))^2;
ta=0; tb=20; options=optimset('TolX',1.e-8);
[tmin,dmin,exitflag,output]=fminbnd(d,ta,tb,options)
```

otteniamo convergenza in 10 iterazioni alla soluzione $t_{\min}=8.438731484275010$. Le due navi si trovano alla distanza minima pari a $d_{\min}=5.691754805947144$ miglia marine dopo quasi 8 ore e mezza dalla partenza.

Soluzione 7.3 Definiamo la funzione obiettivo, rappresentiamola insieme alle sue linee di livello su un dominio circolare di centro $(-1, 0)$ e raggio 3 con le seguenti istruzioni:

```
fun=@(x) x(1)^4+x(2)^4+x(1)^3+3*x(1)*...
x(2)^2-3*x(1)^2-3*x(2)^2+10;
[r,theta]=meshgrid(0:.1:3,0:pi/25:2*pi);
x1=r.*cos(theta)-1; y1=r.*sin(theta);
[n,m]=size(x1); z1=zeros(n,m);
for i=1:n, for j=1:m
    z1(i,j)=fun([x1(i,j);y1(i,j)]);
end, end
figure(1); clf; p1=mesh(x1,y1,z1);
set(p1,'Edgecolor',[0,1,1]); hold on
contour(x1,y1,z1,100,'Linecolor',[0.8,0.8,0.8]);
```

Vediamo che la funzione ammette un punto di massimo locale, un punto di sella e due punti di minimo globale (la funzione è pari rispetto alla variabile x_2). Scegliendo $\mathbf{x}^{(0)}=(-3, 0)$ e imponendo tolleranza $\varepsilon = 10^{-8}$ per il test d'arresto, digitiamo i comandi:

```
x0=[-3;0]; options=optimset('TolX',1.e-8);
[xm,fval,exitf,out]=fminsearch(fun,x0,options)
```

Il punto di minimo $\mathbf{x}_m=[-2.1861e+00, 2.1861e+00]$ è raggiunto in 181 iterazioni e sono state effettuate 353 valutazioni funzionali. Per simmetria, l'altro punto di minimo globale è $(-2.1861, 2.1861)$. Mettiamo in guardia il lettore che scegliendo $\mathbf{x}_0=[1;0]$ la *function fminsearch* di MATLAB converge al punto di massimo locale $(0.75000, 0.61237)$ anziché ad un punto di minimo, mentre quella di Octave converge sempre al punto di minimo $(-2.1861, 2.1861)$.

Soluzione 7.4 Riscriviamo la successione $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k \mathbf{d}^{(k)}$ come

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(0)} + \sum_{\ell=0}^k \alpha_\ell \mathbf{d}^{(\ell)},$$

abbiamo $\mathbf{x}^{(0)} = 3/2$ e

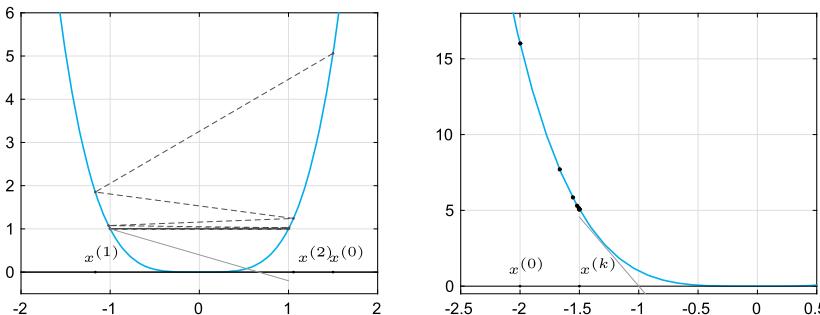


Figura 10.14. A sinistra: la successione del metodo di discesa della Soluzione 7.4. Prendendo $x^{(k)} \simeq -1$, il punto $(x^{(k+1)}, f(x^{(k+1)}))$ dovrebbe trovarsi sotto la retta azzurra per soddisfare la prima condizione di Wolfe con $\sigma = 0.2$, al contrario esso rimane abbondantemente sopra, infatti si ha $(x^{(k+1)}, f(x^{(k+1)})) \simeq (1, 1)$. A destra: la successione del metodo di discesa della Soluzione 7.5. Il punto $(x^{(k+1)}, f(x^{(k+1)}))$ dovrebbe trovarsi a destra del punto di tangenza tra la curva e la retta azzurra per soddisfare la seconda condizione di Wolfe con $\delta = 0.9$, invece rimane molto prossimo a $(-1.5, 5.06)$

$$\begin{aligned} x^{(k+1)} &= \frac{3}{2} + \left(2 + \frac{2}{3^{k+1}}\right)(-1)^{k+1} = \frac{3}{2} - 2 \sum_{\ell=0}^k (-1)^\ell - \frac{1}{2} - \frac{1}{6} \left(-\frac{1}{3}\right)^k \\ &= (-1)^{k+1} \left(1 + \frac{1}{6 \cdot 3^k}\right), \end{aligned}$$

cioè la successione $x^{(k)}$ non converge a zero per $k \rightarrow \infty$ anche se la successione dei valori $f(x^{(k)})$ è decrescente, come possiamo osservare in Figura 10.14, a sinistra. Quando i punti $x^{(k)}$ sono prossimi a $+1$ e -1 , la prima condizione di Wolfe (7.39)₁ non è soddisfatta in quanto la variazione di f tra un passo ed il successivo diventa infinitesima mentre la misura dei passi rimane circa pari a 2.

Soluzione 7.5 Procediamo come nell'esercizio precedente, abbiamo $x^{(0)} = -2$ e $x^{(k+1)} = -2 + (1 - 3^{-k})/2 \rightarrow -3/2$ per $k \rightarrow \infty$. Anche in questo caso la successione dei valori $f(x^{(k)})$ è decrescente, come possiamo osservare in Figura 10.14, a destra. Quando i punti $x^{(k)}$ sono prossimi a $-3/2$ la seconda condizione di Wolfe (7.39)₂ non è soddisfatta in quanto la derivata prima della curva (intesa con il proprio segno) nel punto successivo dovrebbe essere maggiore di δ volte quella nel punto $x^{(k)}$.

Soluzione 7.6 Definiamo la funzione f , il suo gradiente e la matrice Hessiana e richiamiamo il Programma 7.3 con le seguenti istruzioni:

```
fun=@(x) 100*(x(2)-x(1)^2)^2+(1-x(1))^2;
grad=@(x) [-400*(x(2)-x(1)^2)*x(1)-2*(1-x(1));
            200*(x(2)-x(1)^2)];
hess=@(x) [-400*x(2)+1200*x(1)^2+2, -400*x(1);
            -400*x(1), 200];
x0=[-1.2,1]; tol=1.e-8; kmax=500;
meth=1; % Newton
```

```
[x1,err1,k1]=descent(fun,grad,x0,tol,kmax,meth,hess);
meth=2; H0=eye(length(x0)); % BFGS
[x2,err2,k2]=descent(fun,grad,x0,tol,kmax,meth,H0);
meth=3; %gradiente
[x3,err3,k3]=descent(fun,grad,x0,tol,kmax,meth);
meth=41; %gradiente coniugato con beta_FR
[x41,err41,k41]=descent(fun,grad,x0,tol,kmax,meth);
meth=42; %gradiente coniugato con beta_PR
[x42,err42,k42]=descent(fun,grad,x0,tol,kmax,meth);
meth=43; %gradiente coniugato con beta_HS
[x43,err43,k43]=descent(fun,grad,x0,tol,kmax,meth);
```

Tutti i metodi convergono al punto di minimo globale (1, 1).

```
Newton: k1 = 22, err = 1.8652e-12
BFGS: k2 = 35, err = 1.7203e-09
Grad: k3 = 352, err = 8.1954e-09
CG-FR: k41 = 284, err = 5.6524e-10
CG-PR: k42 = 129, err = 5.8148e-09
CG-HS: k43 = 65, err = 9.8300e-09
```

Il numero di iterazioni richieste è in accordo con le proprietà teoriche di convergenza dei vari metodi. In particolare osserviamo che il metodo più veloce è quello di Newton, riflettendo una convergenza quadratica, BFGS, che è un metodo a convergenza super-lineare, impiega una decina di iterazioni più di Newton. Il metodo del gradiente richiede più di 300 iterazioni evidenziando una convergenza di tipo lineare, mentre tra i metodi di tipo gradiente coniugato, tutti a convergenza lineare, è da preferirsi quello con parametri HS (Hestenes-Stiefel). La variabile `err` riportata in tutti i casi contiene l'ultimo valore dello stimatore dell'errore utilizzato per il test d'arresto.

Soluzione 7.7 Valutando la funzione $f(\mathbf{x})$ sul quadrato $[-5, 5]^2$ e rappresentandone graficamente le linee di livello corrispondenti ai valori nell'intervallo $[0, 20]$, vediamo che essa ha un punto di sella in prossimità di $(0, 0)$ e due punti di minimo locale, uno vicino a $(-1, -1)$, l'altro a $(2, 2)$ (si veda Figura 10.15). Uno dei due punti sarà il punto di minimo globale cercato. Fissiamo la tolleranza `tol=1.e-5` per il test d'arresto e un numero massimo di iterazioni pari a

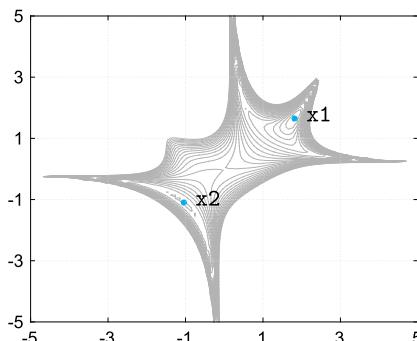


Figura 10.15. Linee di livello tra i valori 0 e 20 della funzione della Soluzione 7.7

100, quindi prendiamo `delta0=0.5` quale raggio iniziale della *trust-region* per il Programma 7.4. Dopo aver definito i *function handle* della funzione obiettivo e del suo gradiente, fissiamo `meth=2` per entrambi i Programmi 7.4 e 7.3 in modo che essi utilizzino direzioni di discesa quasi-Newton e `hess=eye(2)` per il Programma 7.3. Scegliendo $\mathbf{x}_0 = (2, -1)$, il metodo *trust-region* converge in 28 iterazioni al punto $\mathbf{x}_1=(1.8171, 1.6510)$ (riportiamo per comodità le sole prime 4 cifre decimali), mentre il metodo BFGS converge in 27 iterazioni all'altro punto di minimo locale $\mathbf{x}_2=(-5.3282e-01, -5.8850e-01)$. Si ha $f(\mathbf{x}_1) \simeq 3.6661$ e $f(\mathbf{x}_2) \simeq 8.2226$. Prendendo $\mathbf{x}^{(0)} = (2, 1)$, entrambi i metodi convergono al punto di minimo globale \mathbf{x}_1 in 11 iterazioni.

Soluzione 7.8 Calcolare i punti stazionari di $\tilde{f}_k(\mathbf{x}) = \frac{1}{2}\|\tilde{\mathbf{R}}_k(\mathbf{x})\|^2$ equivale a risolvere il sistema non lineare

$$\nabla \tilde{f}_k(\mathbf{x}) = J_{\tilde{\mathbf{R}}_k}(\mathbf{x})^T \tilde{\mathbf{R}}_k(\mathbf{x}) = \mathbf{0}. \quad (10.5)$$

Per la definizione (7.64) si ha $J_{\tilde{\mathbf{R}}_k}(\mathbf{x}) = J_{\mathbf{R}}(\mathbf{x}^{(k)})$ per ogni $\mathbf{x} \in \mathbb{R}^n$ e il sistema (10.5) diventa

$$J_{\mathbf{R}}(\mathbf{x}^{(k)})^T \mathbf{R}(\mathbf{x}^{(k)}) + J_{\mathbf{R}}(\mathbf{x}^{(k)})^T J_{\mathbf{R}}(\mathbf{x}^{(k)}) (\mathbf{x} - \mathbf{x}^{(k)}) = \mathbf{0},$$

ovvero (7.63).

Soluzione 7.9 Dobbiamo dimostrare che $\delta\mathbf{x}^{(k)}$ soddisfa le condizioni (5.67). Ricordiamo che per ogni matrice A con rango massimo, la matrice $A^T A$ è simmetrica e definita positiva.

Dimostriamo (5.67)₂. Dalla definizione $\nabla f(\mathbf{x}^{(k)}) = J_{\mathbf{R}}(\mathbf{x}^{(k)})^T \mathbf{R}(\mathbf{x}^{(k)})$, segue che $\nabla f(\mathbf{x}^{(k)}) = \mathbf{0} \Leftrightarrow \mathbf{R}(\mathbf{x}^{(k)}) = \mathbf{0}$ (poiché $J_{\mathbf{R}}(\mathbf{x}^{(k)})$ ha rango massimo) e quindi da (7.63)₁ segue $\delta\mathbf{x}^{(k)} = \mathbf{0}$.

Supponiamo ora che $\mathbf{R}(\mathbf{x}^{(k)}) \neq \mathbf{0}$, si ha

$$\begin{aligned} & (\delta\mathbf{x}^{(k)})^T \nabla f(\mathbf{x}^{(k)}) \\ &= -\left\{ [J_{\mathbf{R}}(\mathbf{x}^{(k)})^T J_{\mathbf{R}}(\mathbf{x}^{(k)})]^{-1} J_{\mathbf{R}}(\mathbf{x}^{(k)})^T \mathbf{R}(\mathbf{x}^{(k)}) \right\}^T J_{\mathbf{R}}(\mathbf{x}^{(k)})^T \mathbf{R}(\mathbf{x}^{(k)}) \\ &\quad - (J_{\mathbf{R}}(\mathbf{x}^{(k)})^T \mathbf{R}(\mathbf{x}^{(k)}))^T [J_{\mathbf{R}}(\mathbf{x}^{(k)})^T J_{\mathbf{R}}(\mathbf{x}^{(k)})]^{-1} (J_{\mathbf{R}}(\mathbf{x}^{(k)})^T \mathbf{R}(\mathbf{x}^{(k)})) < 0, \end{aligned}$$

ovvero (5.67)₁ è soddisfatta.

Soluzione 7.10 Posto $r_i(\mathbf{x}) = x_1 + x_2 t_i + x_3 t_i^2 + x_4 e^{-x_5 t_i} - y_i$, per $i = 1, \dots, 8$, i coefficienti cercati x_1, \dots, x_5 , sono quelli in corrispondenza dei quali la funzione (7.61) ottiene il suo minimo. Richiamiamo il Programma 7.5 con le seguenti istruzioni:

```
t= [0.055; 0.181; 0.245; 0.342; 0.419; 0.465; 0.593; 0.752];
y= [2.80; 1.76; 1.61; 1.21; 1.25; 1.13; 0.52; 0.28];
tol=1.e-12; kmax=500;
x0=[2, -2.5, -2, 5, 35];
[x,err,iter]= gaussnewton(@mqnlr, @mqnljr, ...
    x0,tol,kmax,t,y);
```

dove `mqnlr` e `mqnljr` sono le *function* di definizione di $\mathbf{R}(\mathbf{x})$ e $J_{\mathbf{R}}(\mathbf{x})$ rispettivamente:

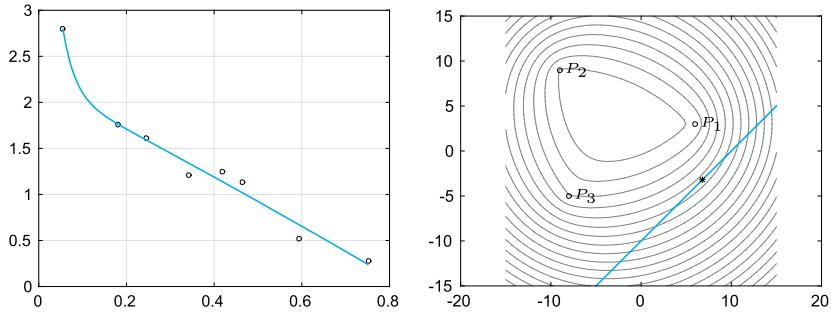


Figura 10.16. A sinistra: i dati e la soluzione dell'Esercizio 7.10. A destra: la soluzione dell'Esercizio 7.12. In grigio le curve di livello della funzione obiettivo. Il dominio Ω di ammissibilità è la parte illimitata del piano sottostante la retta azzurra.

```

function r=mqnlr(x,t,y)
m=length(t); n=length(x);
r=zeros(m,1);
for i=1:m
r(i)=sqrt(2)*(x(1)+t(i)*x(2)+t(i)^2*x(3)+...
    x(4)*exp(-t(i)*x(5))-y(i));
end

function jr=mqnljr(x,t,y)
m=length(t); n=length(x); jr=zeros(m,n);
for i=1:m
jr(i,1)=1; jr(i,2)=t(i);
jr(i,3)=t(i)^2; jr(i,4)=exp(-t(i)*x(5));
jr(i,5)=-t(i)*x(4)*exp(-t(i)*x(5));
end
jr=jr*sqrt(2);

```

Otteniamo convergenza in 19 iterazioni al punto $x = [2.2058e+00 \ -2.4583e+00 \ -2.1182e-01 \ 5.2106e+00 \ 3.5733e+01]$. Il residuo nel punto calcolato non è nullo, bensì vale $f(x) = 1.8428e-01$, tuttavia possiamo classificare il problema dato tra i problemi a residuo piccolo, quindi con convergenza di tipo lineare. Richiamando il metodo di Newton (7.31) per risolvere lo stesso problema otteniamo infatti convergenza in sole 8 iterazioni. Scegliendo un punto iniziale non sufficientemente vicino al punto di minimo, ad esempio $x_0 = [1, 1, 1, 1, 10]$, il metodo di Gauss-Newton non arriva a convergenza, mentre il metodo damped Gauss-Newton converge in 21 iterazioni. In Figura 10.16, a sinistra, è mostrata la funzione $\phi(t)$ i cui coefficienti x_1, \dots, x_5 sono quelli calcolati numericamente. I cerchietti neri rappresentano i dati (t_i, y_i) .

Soluzione 7.11 Sia $f(\mathbf{x}) = \frac{1}{2}\|\mathbf{R}(\mathbf{x})\|^2$, una sua approssimazione quadratica centrata in $\mathbf{x}^{(k)}$ è

$$\tilde{f}_k(\mathbf{s}) = f(\mathbf{x}^{(k)}) + \mathbf{s}^T \nabla f(\mathbf{x}^{(k)}) + \frac{1}{2} \mathbf{s}^T \mathbf{H}_k \mathbf{s} \quad \forall \mathbf{s} \in \mathbb{R}^n,$$

dove H_k è l'Hessiana in $\mathbf{x}^{(k)}$ o una sua approssimazione. Ricordando (7.62)₁ e prendendo

$$H_k = J_{\mathbf{R}}(\mathbf{x}^{(k)})^T J_{\mathbf{R}}(\mathbf{x}^{(k)}),$$

otteniamo

$$\begin{aligned}\tilde{f}_k(\mathbf{s}) &= \frac{1}{2} \| \mathbf{R}(\mathbf{x}^{(k)}) \|^2 + \mathbf{s}^T J_{\mathbf{R}}(\mathbf{x}^{(k)})^T \mathbf{R}(\mathbf{x}^{(k)}) + \frac{1}{2} \mathbf{s}^T J_{\mathbf{R}}(\mathbf{x}^{(k)})^T J_{\mathbf{R}}(\mathbf{x}^{(k)}) \mathbf{s} \\ &= \frac{1}{2} \| \mathbf{R}(\mathbf{x}^{(k)}) + J_{\mathbf{R}}(\mathbf{x}^{(k)}) \mathbf{s} \|^2 \\ &= \frac{1}{2} \| \tilde{\mathbf{R}}_k(\mathbf{x}) \|^2,\end{aligned}$$

grazie a (7.64) con $\mathbf{s} = \mathbf{x} - \mathbf{x}^{(k)}$. In conclusione possiamo interpretare \tilde{f}_k come un modello quadratico di f in un intorno di $\mathbf{x}^{(k)}$, ottenuto approssimando $\mathbf{R}(\mathbf{x})$ con $\tilde{\mathbf{R}}_k(\mathbf{x}) = \mathbf{R}(\mathbf{x}^{(k)}) + J_{\mathbf{R}}(\mathbf{x})(\mathbf{x} - \mathbf{x}^{(k)})$.

Soluzione 7.12 Si tratta di dover risolvere un problema di minimo con funzione obiettivo $f(x, y) = \sum_{i=1}^3 v_i \sqrt{(x - x_i)^2 + (y - y_i)^2}$ e dominio di ammissibilità $\Omega = \{(x, y) \in \mathbb{R}^2 : y \leq x - 10\}$ e dove i valori v_i rappresentano il numero di viaggi verso il punto vendita P_i .

Definiamo la funzione obiettivo ed i vincoli e richiamiamo il Programma **penalty.m** con le seguenti istruzioni:

```
x1=[6; 3]; x2=[-9; 9]; x3=[-8;-5]; v=[140;134;88];
d=@(x)v(1)*sqrt((x(1)-x1(1)).^2+(x(2)-x1(2)).^2)+...
    v(2)*sqrt((x(1)-x2(1)).^2+(x(2)-x2(2)).^2)+...
    v(3)*sqrt((x(1)-x3(1)).^2+(x(2)-x3(2)).^2);
g=@(x)[x(1)-x(2)-10];
meth=0; x0=[10;-10]; tol=1.e-8; kmax=200;kmaxd=200;
[xmin,err,k]=penalty(d,[],[],[],g,[],x0,tol,...
    kmax,kmaxd,meth);
```

All'interno dell'algoritmo di penalizzazione abbiamo utilizzato il metodo di Nelder e Mead per la minimizzazione non vincolata. Non abbiamo richiamato un metodo di discesa in quanto la funzione obiettivo ammette dei punti di non derivabilità e potrebbero essere mal condizionate le matrici H_k per la costruzione delle direzioni $\mathbf{d}^{(k)}$. La posizione ottimale calcolata per il magazzino è data dal punto $xmin=[6.7734, -3.2266]$. La convergenza è stata raggiunta con 13 iterazioni del metodo di penalizzazione.

Soluzione 7.13 Non essendo presenti vincoli di disuguaglianza, il problema può essere riscritto nella forma (7.77) e possiamo procedere come abbiamo fatto nell'Esempio 7.14. La matrice C ha rango 2 e la dimensione del nucleo di C è pari a 1, inoltre $\ker C = \{\mathbf{z} = \alpha[1, 1, 1]^T, \alpha \in \mathbb{R}\}$. La matrice A è simmetrica e, poiché $\sum_{i,j=1}^3 a_{ij} > 0$, essa è anche definita positiva sul nucleo della matrice C. Quindi costruiamo la matrice M = [A, C^T; C, 0] ed il termine noto $\mathbf{f} = [-\mathbf{b}, \mathbf{d}]^T$ e risolviamo il sistema lineare (7.77) con le istruzioni:

```
A=[2,-1,1;-1,3,4;1,4,1]; b=[1;-2;-1];
C=[2,-2,0;2,1,-3]; d=[1;1];
M=[A C'; C, zeros(2)]; f=[-b;d];
x1=M\f;
```

ottenendo la soluzione

```
x1 =
5.7143e-01
7.1429e-02
7.1429e-02
-1.0476e+00
-2.3810e-02
```

L'approssimazione del punto di minimo è data dalle prime 3 componenti del vettore $x1$, mentre le ultime due rappresentano i moltiplicatori di Lagrange associati ai vincoli. Il valore della funzione nel punto di minimo calcolato è $6.9388e-01$.

Soluzione 7.14 Rappresentiamo la funzione $v(x,y)$ sul quadrato $[-2.5, 2.5]^2$ e la sua restrizione al vincolo $h(x,y) = x^2/4 + y^2 - 1 = 0$ (si veda la Figura 10.17). La funzione presenta vari punti di massimo locale nel dominio di ammissibilità, quello di massimo globale è in un intorno del punto $(2, 0.5)$.

Definiamo i dati per richiamare il Programma 7.7 e risolviamo il problema di minimo non vincolato per la funzione $f(x,y) = -v(x,y)$ con il metodo BFGS:

```
fun=@(x)-(sin(pi*x(1)*x(2))+1)*(2*x(1)+3*x(2)+4);
grad_fun=@(x)[-pi*x(2)*cos(pi*x(1)*x(2))*...
(2*x(1)+3*x(2)+4)-(sin(pi*x(1)*x(2))+1)*2;
-pi*x(1)*cos(pi*x(1)*x(2))*(2*x(1)+3*x(2)+4)-...
(sin(pi*x(1)*x(2))+1)*3];
h=@(x)x(1)^2/4+x(2)^2-1; grad_h=@(x)[x(1)/2;2*x(2)];
x0=[2;1]; lambda0=1; tol=1.e-8; kmax=100; kmaxd=100;
meth=0; hess=eye(2);
[x,err,k]=auglagrange(fun,grad_fun,h,grad_h, ...
x0,lambda0,tol,kmax,kmaxd,meth)
```

Fissato $\mathbf{x}^{(0)} = (2, 1)$, si ha convergenza in 9 iterazioni al punto $\mathbf{x}_1 = (0.56833, 0.95877)$, che è un punto di massimo locale vincolato per v , ma, come si evince dal grafico di Figura 10.17, non è il punto di massimo globale cercato. Si ha $v(\mathbf{x}_1) = 15.94$. Prendendo $\mathbf{x}^{(0)} = (1, 0)$, otteniamo convergenza

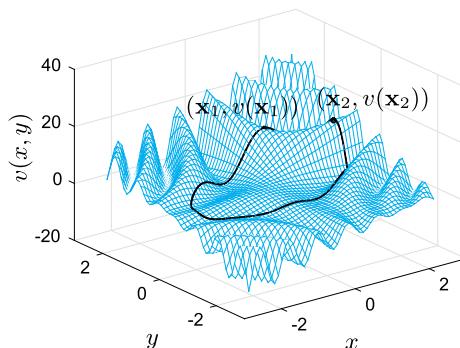


Figura 10.17. La funzione $v(x,y)$ dell'Esercizio 7.14 e i due massimi vincolati calcolati con il metodo della Lagrangiana aumentata

in 9 iterazioni al punto di coordinate $\mathbf{x}_2 = (1.9242, 0.27265)$ in cui la velocità vale $v(\mathbf{x}_2) = 17.307$. Di conseguenza il punto di massimo globale cercato è \mathbf{x}_2 .

10.8 Capitolo 8

Soluzione 8.1 Per verificare l'ordine osserviamo che la soluzione esatta della (8.114) è $y(t) = \frac{1}{2}[e^t - \sin(t) - \cos(t)]$. Risolviamo allora il problema (8.114) con il metodo di Eulero esplicito con h che va da $1/2$ fino a $1/512$ per dimezzamenti successivi:

```
t0=0; y0=0; T=1; f=@(t,y) sin(t)+y;
y=@(t) 0.5*(exp(t)-sin(t)-cos(t));
Nh=2; e=zeros(1,10);
for k=1:10;
[tt,u]=feuler(f,[t0,T],y0,Nh);
e(k)=max(abs(u-y(tt))); Nh=2*Nh;
end
```

Per la (1.13), 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 8.1 con la corrispondente chiamata al Programma 8.2 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 8.2 Risolviamo il problema di Cauchy con il metodo di Eulero esplicito con le seguenti istruzioni:

```
t0=0; T=1; N=100; f=@(t,y) -t*exp(-y);
y0=0; [t,u]=feuler(f,[t0,T],y0,N);
```

Per calcolare il numero di cifre corrette, vogliamo usare la (8.29) 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 ($-1 < y < 0$), potremo prendere $L = e$.

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 $M = e + e^2$. Possiamo sostanziare queste conclusioni operando uno studio grafico del campo vettoriale

$\mathbf{v}(t, y) = [v_1, v_2]^T = [1, f(t, y(t))]^T$ associato al problema di Cauchy dato. Si ha infatti che le soluzioni dell'equazione differenziale $y'(t) = f(t, y(t))$ sono le linee tangenti al campo vettoriale \mathbf{v} .

Con le seguenti istruzioni

```
[T, Y]=meshgrid(0:0.05:1, -1:0.05:0);
V1=ones(size(T)); V2=-T.*exp(Y); quiver(T, Y, V1, V2)
```

vediamo che la soluzione del problema di Cauchy dato ha una derivata seconda non positiva e che cresce in valore assoluto per t crescenti. Questo ci permette di concludere che $M = \max_{0 \leq t \leq 1} |y''(t)|$ è assunto in $t = 1$.

Una via alternativa consiste nell'osservare che $f(t, y(t)) = y'(t) < 0$ e quindi che la funzione $-y$ è positiva e crescente. Di conseguenza sono positivi e crescenti anche i termini e^{-y} e $t^2 e^{-2y}$ e quindi la funzione $y'' = -e^{-y} - t^2 e^{-2y}$ è negativa e decrescente. Questo ci permette di concludere che $M = \max_{0 \leq t \leq 1} |y''(t)|$ è assunto in $t = 1$.

Dalla (8.29), per $h = 0.01$ si ricava allora

$$|u_{100} - y(1)| \leq \frac{e^L - 1}{L} \frac{M}{200} \simeq 0.26$$

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 8.3 La funzione di iterazione è $\phi(u) = u_n - ht_{n+1}e^{-u}$. Grazie al Teorema di Ostrowsky 2.1, il metodo di punto fisso è convergente se $|\phi'(u)| = ht_{n+1}e^{-u} < 1$. 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 $u = -1$ (dal testo dell'Esercizio 8.2 sappiamo che la soluzione esatta è limitata fra -1 e 0). 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 8.4 Basta ripetere le istruzioni date nella Soluzione 8.1, utilizzando il Programma 8.3. 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 8.5 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 8.6 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 $A = \{z = h\lambda \in \mathbb{C} : |1 + h\lambda| < 1\}$. Sostituendo in questa espressione $\lambda = -1 + i$ otteniamo la limitazione su h : $h^2 - h < 0$, ovvero $h \in (0, 1)$.

Soluzione 8.7 Proviamo per induzione su n la proprietà (8.53), che per semplicità denotiamo \mathcal{P}_n . A tale scopo, è sufficiente provare che se vale \mathcal{P}_1 e se \mathcal{P}_{n-1} implica \mathcal{P}_n per ogni $n \geq 2$, allora \mathcal{P}_n vale per ogni $n \geq 2$.

Si verifica facilmente che $u_1 = u_0 + h(\lambda_0 u_0 + r_0)$, mentre per provare $\mathcal{P}_{n-1} \Rightarrow \mathcal{P}_n$, è sufficiente notare che $u_n = u_{n-1}(1 + h\lambda_{n-1}) + hr_{n-1}$.

Soluzione 8.8 Poiché $|1 + h\lambda| < 1$, dalla (8.57) segue che

$$|z_n - u_n| \leq |\rho| \left(\left| 1 + \frac{1}{\lambda} \right| + \left| \frac{1}{\lambda} \right| \right).$$

Se $\lambda \leq -1$, abbiamo $1/\lambda < 0$ e $1 + 1/\lambda \geq 0$, quindi

$$\left| 1 + \frac{1}{\lambda} \right| + \left| \frac{1}{\lambda} \right| = 1 + \frac{1}{\lambda} - \frac{1}{\lambda} = 1 = \varphi(\lambda).$$

Invece, se $-1 < \lambda < 0$, abbiamo $1/\lambda < 1 + 1/\lambda < 0$, quindi

$$\left| 1 + \frac{1}{\lambda} \right| + \left| \frac{1}{\lambda} \right| = -1 - \frac{2}{\lambda} = \left| 1 + \frac{2}{\lambda} \right| = \varphi(\lambda).$$

Soluzione 8.9 Dalla (8.55)abbiamo

$$|z_n - u_n| \leq \bar{\rho}[a(h)]^n + h\bar{\rho}\sum_{k=0}^{n-1} [a(h)]^{n-k-1}$$

ed il risultato segue per la (8.56).

Soluzione 8.10 Il metodo (8.116) quando applicato al problema modello (8.38) fornisce l'equazione $u_{n+1} = u_n(1 + h\lambda + (h\lambda)^2)$, quindi è assolutamente stabile a patto che $|1 + h\lambda + (h\lambda)^2| < 1$. Risolvendo la disequazione $|1 + h\lambda + (h\lambda)^2| < 1$ con $\lambda \in \mathbb{R}^-$ si ottiene $-1 < h\lambda < 0$.

Soluzione 8.11 Per comodità riscriviamo il metodo di Heun seguendo la notazione comune ai metodi Runge-Kutta:

$$\begin{aligned} u_{n+1} &= u_n + \frac{h}{2}(K_1 + K_2) \\ K_1 &= f(t_n, u_n), \quad K_2 = f(t_{n+1}, u_n + hK_1). \end{aligned} \tag{10.6}$$

Abbiamo $h\tau_{n+1}(h) = y(t_{n+1}) - y(t_n) - h(\hat{K}_1 + \hat{K}_2)/2$, con $\hat{K}_1 = f(t_n, y(t_n))$ e $\hat{K}_2 = f(t_{n+1}, y(t_n) + h\hat{K}_1)$. 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 RK2 o di Heun è dunque consistente. Proviamo ora che τ_{n+1} è accurato al secondo ordine rispetto ad h . Supponiamo che $y \in C^3([t_0, T])$. Per semplicità di notazione, poniamo $y_n = y(t_n)$ per ogni $n \geq 0$. Abbiamo

$$\begin{aligned}\tau_{n+1} &= \frac{y_{n+1} - y_n}{h} - \frac{1}{2}[f(t_n, y_n) + f(t_{n+1}, y_n + hf(t_n, y_n))] \\ &= \frac{y_{n+1} - y_n}{h} - \frac{1}{2}y'(t_n) - \frac{1}{2}f(t_{n+1}, y_n + hy'(t_n)).\end{aligned}$$

Grazie alla formula dell'errore (4.27) legata alla formula di quadratura dei trapezi, esiste $\xi_n \in]t_n, t_{n+1}[$ tale che

$$y_{n+1} - y_n = \int_{t_n}^{t_{n+1}} y'(t) dt = \frac{h}{2}[y'(t_n) + y'(t_{n+1})] - \frac{h^3}{12}y'''(\xi_n),$$

quindi

$$\begin{aligned}\tau_{n+1} &= \frac{1}{2} \left(y'(t_{n+1}) - f(t_{n+1}, y_n + hy'(t_n)) - \frac{h^2}{6}y'''(\xi_n) \right) \\ &= \frac{1}{2} \left(f(t_{n+1}, y_{n+1}) - f(t_{n+1}, y_n + hy'(t_n)) - \frac{h^2}{6}y'''(\xi_n) \right).\end{aligned}$$

Inoltre, tenendo in considerazione il fatto che la funzione f è di Lipschitz rispetto alla seconda variabile (si veda la Proposizione 8.1), si ha

$$|\tau_{n+1}| \leq \frac{L}{2}|y_{n+1} - y_n - hy'(t_n)| + \frac{h^2}{12}|y'''(\xi_n)|.$$

Infine, grazie agli sviluppi di Taylor

$$y_{n+1} = y_n + hy'(t_n) + \frac{h^2}{2}y''(\eta_n), \quad \eta_n \in]t_n, t_{n+1}[,$$

otteniamo

$$|\tau_{n+1}| \leq \frac{L}{4}h^2|y''(\eta_n)| + \frac{h^2}{12}|y'''(\xi_n)| \leq Ch^2.$$

Il metodo RK2 o di Heun è implementato nel Programma 10.4. Utilizzando le istruzioni `MAT&OCT`:

```
p=log(abs(e(1:end-1)./e(2:end)))/log(2);
p(1:2:end)
```

si trovano le seguenti stime per l'ordine:

<code>p =</code>	1.7642	1.9398	1.9851	1.9963	1.9991
------------------	--------	--------	--------	--------	--------

che sono in accordo con l'ordine 2 previsto teoricamente.

Programma 10.4. rk2: metodo Runge-Kutta esplicito di ordine 2 (o di Heun)

```

function [tt,u]=rk2(odefun,tspan,y0,Nh)
% RK2 Risolve un sistema di e.d.o. con Runge-Kutta2
% (noto anche come metodo di Heun) e passo costante
h=(tspan(2)-tspan(1))/Nh; hh=h*0.5;
tt=linspace(tspan(1),tspan(2),Nh+1)';
y0=y0(:); % genera sempre un vettore colonna
d=length(y0);
u=zeros(Nh+1,d);
u(1,:)=y0.';
u=zeros(Nh+1,d);
for n=1:Nh
    wn=u(n,:).';
    k1=odefun(tt(n),wn);
    y=wn+h*k1;
    k2=odefun(tt(n+1),y);
    w=wn+hh*(k1+k2);
    u(n+1,:)=w.';
end

```

Soluzione 8.12 Applicando il metodo (10.6) al problema (8.38) si trova $K_1 = \lambda u_n$ e $K_2 = \lambda u_n(1 + h\lambda)$. 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 $-2 < h\lambda < 0$. Essendo λ reale negativo, quest'ultima è la restrizione cercata.

Soluzione 8.13 Abbiamo:

$$\begin{aligned} h\tau_{n+1}(h) &= y(t_{n+1}) - y(t_n) - \frac{h}{6}(\hat{K}_1 + 4\hat{K}_2 + \hat{K}_3), \\ \hat{K}_1 &= f(t_n, y(t_n)), \quad \hat{K}_2 = f(t_n + \frac{h}{2}, y(t_n) + \frac{h}{2}\hat{K}_1), \\ \hat{K}_3 &= f(t_{n+1}, y(t_n) + h(2\hat{K}_2 - \hat{K}_1)). \end{aligned}$$

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 è consistente. Esso è implementato nel Programma 10.5.

Programma 10.5. rk3: metodo Runge-Kutta esplicito di ordine 3

```

function [tt,u]=rk3(odefun,tspan,y0,Nh)
% RK3 Risolve un sistema di e.d.o. con Runge-Kutta3
% con passo costante
h=(tspan(2)-tspan(1))/Nh; hh=h*0.5; h6=h/6; h2=h*2;
tt=linspace(tspan(1),tspan(2),Nh+1)';
y0=y0(:); % genera sempre un vettore colonna
d=length(y0);
u=zeros(Nh+1,d);
u(1,:)=y0.';
u=zeros(Nh+1,d);
for n=1:Nh

```

```

wn=u(n,:);
k1=odefun(tt(n),wn);
t1=tt(n)+hh; y=wn+hh*k1;
k2=odefun(t1,y);
y=wn+hh*(2*k2-k1);
k3=odefun(tt(n+1),y);
w=wn+hh*(k1+4*k2+k3);
u(n+1,:)=w.';
end

```

Utilizzando comandi del tutto analoghi a quelli usati nella Soluzione 8.11 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      2.9989

```

che verificano la stima teorica.

Soluzione 8.14 Utilizzando passaggi del tutto analoghi a quelli della Soluzione 8.12 si trova la relazione

$$u_{n+1} = u_n \left[1 + h\lambda + \frac{1}{2}(h\lambda)^2 + \frac{1}{6}(h\lambda)^3 \right] = u_n p_3(h\lambda).$$

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, se $-2.5 < h\lambda < 0$, allora $|p_3(h\lambda)| < 1$.

Soluzione 8.15 Per risolvere il Problema 8.1 con i valori indicati, basta ripetere le seguenti istruzioni prima con $N = 10$ e poi con $N = 20$:

```

f=@(t,y) -1.68e-9*y^4+2.6880;
[tc,uc]=cranknic(f,[0,200],180,N);
[tp,up]=rk2(f,[0,200],180,N);

```

Le corrispondenti soluzioni vengono riportate nei grafici di Figura 10.18.

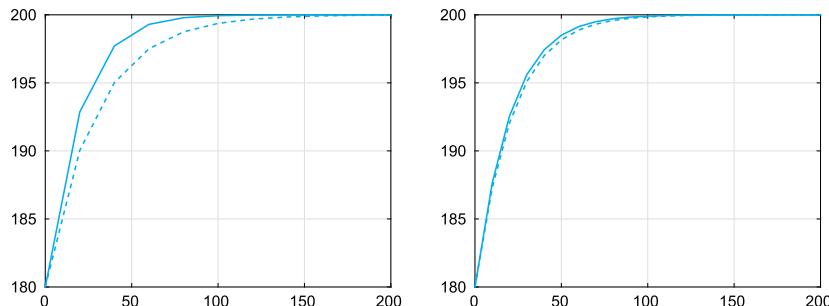


Figura 10.18. Soluzioni calcolate con $N = 10$ (a sinistra) e $N = 20$ (a destra) per il problema di Cauchy della Soluzione 8.15: in linea continua le soluzioni ottenute con il metodo di Crank-Nicolson, in linea tratteggiata quelle ricavate con il metodo RK2 o di Heun

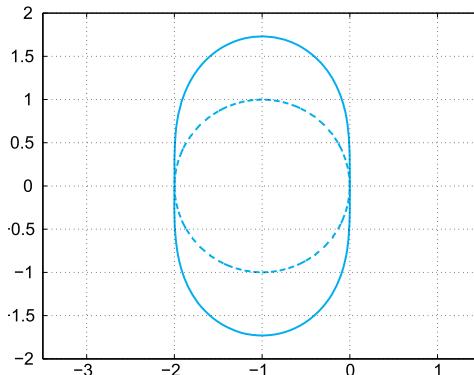


Figura 10.19. Bordo delle regioni di assoluta stabilità per i metodi di Eulero esplicito (linea tratteggiata) e di Heun (linea continua). Le regioni si estendono all'interno delle aree delimitate dalle rispettive curve

Soluzione 8.16 La soluzione numerica del metodo di Heun, applicato al problema modello (8.38), 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,y)[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,f(x,y),[0 0]); grid on
```

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 . 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 10.19 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 regione di assoluta stabilità del metodo di Eulero esplicito (delimitata dal cerchio in linea tratteggiata) ed è anch'essa tangente nell'origine all'asse immaginario.

Soluzione 8.17 Con le seguenti istruzioni:

```
t0=0; y0=0; f=@(t,y)cos(2*y);
y=@(t) 0.5*asin((exp(4*t)-1)./(exp(4*t)+1));
T=1; N=2; for k=1:10;
[tt,u]=rk2(f,[t0,T],y0,N);
```

```
e(k)=max(abs(u-y(tt))); N=2*N; end
p=log(abs(e(1:end-1)./e(2:end)))/log(2); p(1:2:end)

otteniamo

p =
    2.3925      2.1092      2.0269      2.0068      2.0017
```

Come previsto dalla teoria, l'ordine di convergenza del metodo è 2. Il costo computazionale di questo metodo è tuttavia confrontabile con quello di Eulero in avanti, che è accurato solo al primo ordine.

Soluzione 8.18 L'equazione differenziale del secondo ordine data è equivalente al seguente sistema del primo ordine

$$x'(t) = z(t), \quad z'(t) = -5z(t) - 6x(t),$$

con $x(0) = 1$, $z(0) = 0$. Richiamiamo Heun con le seguenti istruzioni:

```
t0=0; y0=[1 0]; T=5;
[t,u]=rk2(@fmolle,[t0,T],y0,N);
```

dove N è il numero di nodi che utilizzeremo, mentre `fmolle.m` è la seguente funzione:

```
function fn=fmolle(t,y)
b=5;
k=6;
[n,m]=size(y);
fn=zeros(n,m);
fn(1)=y(2);
fn(2)=-b*y(2)-k*y(1);
```

In Figura 10.20 riportiamo le 2 componenti della soluzione, calcolate con $N = 20$ e $N = 40$ e confrontate con la soluzione esatta $x(t) = 3e^{-2t} - 2e^{-3t}$ e con la sua derivata.

Soluzione 8.19 Riduciamo il sistema di equazioni di ordine 2 ad un sistema di equazioni del prim'ordine dato da:

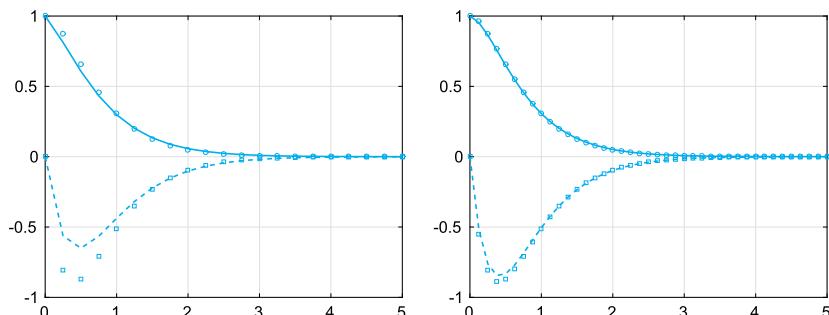


Figura 10.20. Approssimazioni di $x(t)$ (linea continua) e $x'(t)$ (linea tratteggiata) calcolate in corrispondenza di $N = 20$ (a sinistra) e $N = 40$ (a destra). I cerchietti ed i quadratini si riferiscono alle quantità esatte $x(t)$ e $x'(t)$, rispettivamente

$$\begin{cases} x'(t) = z(t), \\ y'(t) = v(t), \\ z'(t) = 2\omega \sin(\Psi)v(t) - k^2x(t), \\ v'(t) = -2\omega \sin(\Psi)z(t) - k^2y(t). \end{cases} \quad (10.7)$$

Se supponiamo che il pendolo all'istante iniziale $t_0 = 0$ sia fermo nella posizione $(1, 0)$, il sistema (10.7) viene completato dalle seguenti condizioni iniziali:

$$x(0) = 1, \quad y(0) = 0, \quad z(0) = 0, \quad v(0) = 0.$$

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(@ffoucault, [0, 300], [1 0 0 0], N);
```

dove N è il numero di passi e `ffoucault.m` la funzione seguente:

```
function fn=ffoucault(t,y)
l=20; k2=9.8/l; psi=pi/4; omega=7.29*1.e-05;
[n,m]=size(y); fn=zeros(n,m);
fn(1)=y(3); fn(2)=y(4);
fn(3)=2*omega*sin(psi)*y(4)-k2*y(1);
fn(4)=-2*omega*sin(psi)*y(3)-k2*y(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 10.21 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 (8.38) con valori di λ puramente immaginari. In tal caso infatti la soluzione esatta è una funzione sinusoidale in t .

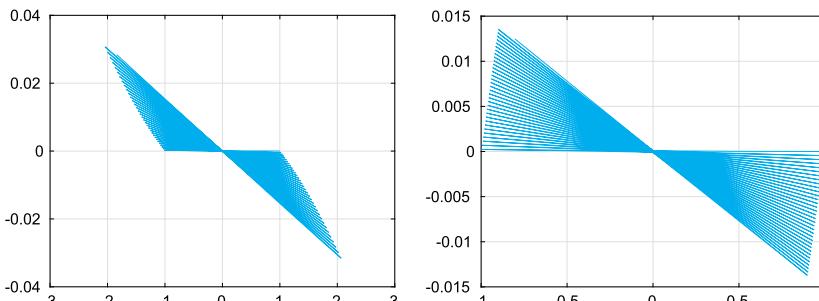


Figura 10.21. Traiettorie nel piano delle fasi per il pendolo di Focault della Soluzione 8.19, ottenute con il metodo di Eulero esplicito (*a sinistra*) e con un metodo Runge-Kutta adattivo (*a destra*)

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 10.21, a destra, la soluzione ottenuta con la funzione `ode23` di MATLAB. 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(@ffoucault, [0, 300], [1 0 0 0]);
```

fornisce una soluzione ragionevole, pur usando solo 1022 passi di integrazione.

Soluzione 8.20 Impostiamo il termine noto del problema nella seguente *function*

```
function fn=baseball(t,y)
phi = pi/180; omega = 1800*1.047198e-01;
B = 4.1*1.e-4; g = 9.8;
[n,m]=size(y); fn=zeros(n,m);
vmodulo = sqrt(y(4)^2+y(5)^2+y(6)^2);
Fv = 0.0039+0.0058/(1+exp((vmodulo-35)/5));
fn(1)=y(4);
fn(2)=y(5);
fn(3)=y(6);
fn(4)=-Fv*vmodulo*y(4)+...
B*omega*(y(6)*sin(phi)-y(5)*cos(phi));
fn(5)=-Fv*vmodulo*y(5)+B*omega*y(4)*cos(phi);
fn(6)=-g-Fv*vmodulo*y(6)-B*omega*y(4)*sin(phi);
```

A questo punto basta richiamare `ode23` nel modo seguente:

```
[t, u] = ode23(@baseball, [0 0.4], ...
[0 0 0 38*cos(pi/180) 0 38*sin(pi/180)]);
```

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
```

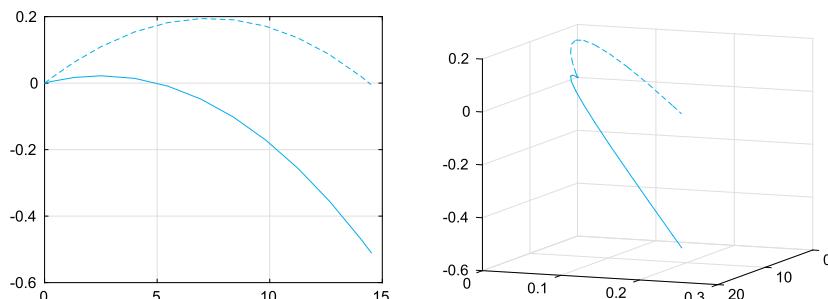


Figura 10.22. Le traiettorie seguite da una palla da baseball lanciata con angolo iniziale pari a 1 grado (linea continua) e 3 gradi (linea tratteggiata)

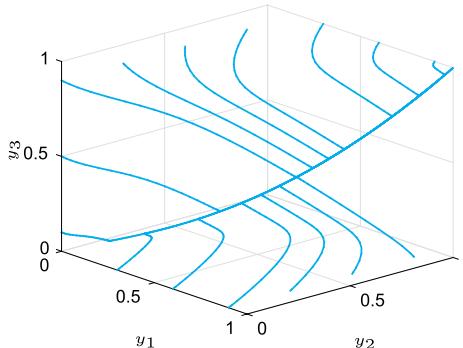


Figura 10.23. Le traiettorie del modello (8.117) per vari dati iniziali ed $\varepsilon = 10^{-2}$

In Figura 10.22 riportiamo le traiettorie della palla da baseball con un'inclinazione di 1 grado e di 3 gradi, rispettivamente in una rappresentazione sul piano x_1x_3 ed in una rappresentazione tridimensionale.

Soluzione 8.21 Definiamo la funzione

```
function f=fchem3(t,y)
e=1.e-2;
[n,m]=size(y);f=zeros(n,m);
f(1)=1/e*(-5*y(1)-y(1)*y(2)+5*y(2)^2+...
y(3))+y(2)*y(3)-y(1);
f(2)=1/e*(10*y(1)-y(1)*y(2)-10*y(2)^2+y(3))...
-y(2)*y(3)+y(1);
f(3)=1/e*(y(1)*y(2)-y(3))-y(2)*y(3)+y(1);
```

e diamo le seguenti istruzioni

```
y0=[1,0.5,0]; tspan=[0,10];
[t1,y1]=ode23(@fchem3,tspan,y0);
[t2,y2]=ode23s(@fchem3,tspan,y0);
fprintf('Passi ode23=%d, passi ode23s=%d\n',...
length(t1),length(t2))
```

`ode23` richiede 8999 passi di integrazione contro i 43 di `ode23s` e possiamo affermare che il problema dato è *stiff*. Le soluzioni ottenute sono mostrate in Figura 10.23.

10.9 Capitolo 9

Soluzione 9.1 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 9.2 Consideriamo il problema di Poisson in una dimensione. Integrando l'equazione (9.14)₁ sull'intervallo (a, b) e applicando la regola di integrazione per parti otteniamo

$$\int_a^b f(x)dx = - \int_a^b u''(x)dx = [-u'(x)]_a^b = \gamma - \delta.$$

Per il problema di Poisson in due dimensioni (9.15)₁ invochiamo il teorema della divergenza (o di Gauss) secondo il quale se \mathbf{v} è una funzione vettoriale le cui componenti sono di classe $C^1(\Omega)$, allora

$$\int_{\Omega} \nabla \cdot \mathbf{v} d\Omega = \int_{\partial\Omega} \mathbf{v} \cdot \mathbf{n} d(\partial\Omega). \quad (10.8)$$

Osservando che $\Delta u = \nabla \cdot (\nabla u)$ e assumendo che $(\nabla u) \in [C^1(\Omega)]^2$, applichiamo (10.8) a $\mathbf{v} = \nabla u$ e otteniamo:

$$\begin{aligned} \int_{\Omega} f d\Omega &= \int_{\Omega} -\Delta u d\Omega = - \int_{\Omega} \nabla \cdot (\nabla u) d\Omega \\ &= - \int_{\partial\Omega} \nabla u \cdot \mathbf{n} d(\partial\Omega) = - \int_{\partial\Omega} g_N d(\partial\Omega). \end{aligned}$$

Soluzione 9.3 Poiché $A_{fd} = h^{-2}\hat{A}$, verifichiamo la proprietà richiesta mostrando che $\mathbf{x}^T \hat{A} \mathbf{x} > 0$ per ogni $\mathbf{x} \neq \mathbf{0}$. (È evidente che A_{fd} è simmetrica.) Abbiamo

$$\begin{bmatrix} x_1 & x_2 & \dots & x_{N-1} & x_N \end{bmatrix} \begin{bmatrix} 2 & -1 & 0 & \dots & 0 \\ -1 & 2 & \ddots & & \vdots \\ 0 & \ddots & \ddots & -1 & 0 \\ \vdots & & -1 & 2 & -1 \\ 0 & \dots & 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{N-1} \\ x_N \end{bmatrix}$$

$$= 2x_1^2 - 2x_1x_2 + 2x_2^2 - 2x_2x_3 + \dots - 2x_{N-1}x_N + 2x_N^2.$$

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 9.4 Verifichiamo che $A_{fd} \mathbf{q}_j = \lambda_j \mathbf{q}_j$. Eseguiamo il prodotto matrice-vettore $\mathbf{w} = A_{fd} \mathbf{q}_j$ ed imponiamo che \mathbf{w} sia uguale al vettore $\lambda_j \mathbf{q}_j$. Troviamo le seguenti equazioni (dopo aver moltiplicato ambo i membri per h^2):

$$\begin{cases} 2 \sin(j\theta) - \sin(2j\theta) = 2(1 - \cos(j\theta)) \sin(j\theta), \\ -\sin(j(k-1)\theta) + 2 \sin(jk\theta) - \sin(j(k+1)\theta) = 2(1 - \cos(j\theta)) \sin(kj\theta), \\ k = 2, \dots, N-1 \\ 2 \sin(Nj\theta) - \sin((N-1)j\theta) = 2(1 - \cos(j\theta)) \sin(Nj\theta). \end{cases}$$

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 per la formula di prostaferesi vale

$$\sin((k-1)j\theta) + \sin((k+1)j\theta) = 2 \sin(kj\theta) \cos(j\theta)$$

e, in particolare per l'ultima equazione, che $\sin((N+1)j\theta) = 0$ in quanto $\theta = \pi/(N+1)$. Essendo A_{fd} simmetrica e definita positiva, $K(A_{fd}) = \lambda_{\max}/\lambda_{\min}$ ovvero $K(A_{fd}) = \lambda_N/\lambda_1 = (1 - \cos(N\pi/(N+1)))/(1 - \cos(\pi/(N+1)))$. Se

si osserva che $\cos(N\pi/(N+1)) = -\cos(\pi/(N+1))$ e si sviluppa in serie la funzione coseno e si arresta lo sviluppo al second'ordine, si trova allora $K(A_{fd}) \simeq (N+1)^2$ cioè $K(A_{fd}) \simeq h^{-2}$.

Soluzione 9.5 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 $\xi_- \in (x-h, x)$.

Sommendo membro a membro le due espressioni si trova

$$u(\bar{x} + h) + u(\bar{x} - h) = 2u(\bar{x}) + h^2 u''(\bar{x}) + \frac{h^4}{24}(u^{(4)}(\xi_+) + u^{(4)}(\xi_-)),$$

da cui la proprietà desiderata.

Soluzione 9.6 La matrice è ancora tridiagonale ed ha elementi $(A_{fd})_{i,i-1} = -\mu/h^2 - \eta/(2h)$, $(A_{fd})_{ii} = 2\mu/h^2 + \sigma$, $(A_{fd})_{i,i+1} = -\mu/h^2 + \eta/(2h)$. Il termine noto, una volta incorporate le condizioni al contorno, diventa conseguentemente $\mathbf{f} = (f(x_1) + \alpha(\mu/h^2 + \eta/(2h)), f(x_2), \dots, f(x_{N-1}), f(x_N) + \beta(\mu/h^2 - \eta/(2h)))^T$.

Soluzione 9.7 Con le seguenti istruzioni `MAT&OCT` calcoliamo le soluzioni relative ai 3 valori di h indicati nel testo:

```
a=0; b=1; mu=1; eta=0; sigma=0.1; ua=0; ub=0;
f=@(x) 1+sin(4*pi*x);
[x,uh11]=bvp_fd_dir_1d(a,b,9,mu,eta,sigma,f,ua,ub);
[x,uh21]=bvp_fd_dir_1d(a,b,19,mu,eta,sigma,f,ua,ub);
[x,uh41]=bvp_fd_dir_1d(a,b,39,mu,eta,sigma,f,ua,ub);
```

Si ricordi che $h = (b-a)/(N+1)$. Per stimare l'ordine di convergenza, non avendo a disposizione la soluzione esatta, calcoliamo una soluzione approssimata relativa ad una griglia estremamente fitta (ponendo ad esempio $h = 1/1000$). A questo punto utilizziamo la soluzione così calcolata invece della soluzione esatta e calcoliamo l'errore nella norma discreta del massimo rispetto ad essa con le istruzioni:

```
[x,uhref]=bvp_fd_dir_1d(a,b,999,mu,eta,sigma,f,ua,ub);
e1=norm(uh11-uhref(1:100:end),inf)
e2=norm(uh21-uhref(1:50:end),inf)
e3=norm(uh41-uhref(1:25:end),inf)
```

Troviamo:

```
e1 =
 8.6782e-04
e2 =
 2.0422e-04
e3 =
 5.2789e-05
```

Dimezzando h l'errore si divide per 4, a conferma dell'ordine 2 rispetto a h .

Soluzione 9.8 Modifichiamo il Programma 9.1 in modo da imporre le condizioni di Neumann invece di quelle di Dirichlet. Poiché stavolta anche i valori di u agli estremi dell'intervallo sono incogniti, il vettore soluzione diventa $\mathbf{u} = (u_0, \dots, u_{N+1})^T$ e la matrice ha dimensione $N + 2$. Nella prima e nell'ultima equazione del sistema dobbiamo imporre le condizioni

$$\frac{3u_0 - 4u_1 + u_2}{2h} = \gamma, \quad \frac{3u_{N+1} - 4u_N + u_{N-1}}{2h} = \delta.$$

Il Programma 10.6 implementa quanto richiesto, gli input e gli output coincidono con quelli del Programma 9.1, ad eccezione di `ua` e `ub`, che ora sono sostituiti da `gamma` e `delta`.

Programma 10.6. bvp_fd_neu_1d: approssimazione di un problema ai limiti di Neumann con differenze finite

```
function [xh,uh]=bvp_fd_neu_1d(a,b,N,mu,eta,sigma,%
                                bvpfun,gamma,delta,varargin)
% BVP_FD_NEU_1D Risolve il problema di diffusione-tra-
% sporto-reazione e condizioni di Neumann su (a,b)
% con differenze finite centrate e N+2 nodi equispaz-
% ziati.
% [xh,uh]=bvp_fd_neu_1d(a,b,N,mu,eta,sigma,%
%                         bvpfun,gamma,delta)
%
h =(b-a)/(N+1); xh = (linspace(a,b,N+2))';
hm =mu/h^2; hd = eta/(2*h);
e =ones(N+2,1);
Afd =spdiags([-(-hm+hd)*e (2*hm+sigma)*e (-hm+hd)*e],...
              -1:1, N, N);
Afd(1,1)=3/(2*h); Afd(1,2)=-2/h; Afd(1,3)=1/(2*h);
Afd(N+2,N+2)=3/(2*h); Afd(N+2,N+1)=-2/h;
Afd(N+2,N)=1/(2*h);
f =bvpfun(xh,varargin{:});
f(1)=gamma; f(N+2)=delta;
uh = Afd\f;
```

A questo punto, con le istruzioni `MAT&OCT`:

```
a=0;b=0.75;
bvpfun=@(x)((2*pi)^2+0.1)*cos(2*pi*x);
mu=1; eta=0; sigma=1/10;
gamma=0; delta=2*pi;
uex=@(x)cos(2*pi*x);
H=[]; Err=[];
for N=100:50:500
h=(b-a)/(N+1); H=[H;h];
[x,uh]=bvp_fd_neu_1d(a,b,N,mu,eta,sigma,%
                      bvpfun,gamma,delta);
Err=[Err;norm(uex(x)-uh,inf)];
end
```

richiamiamo il Programma 10.6 per diversi valori di N e calcoliamo l'errore nella norma del massimo discreta rispetto alla soluzione. Avendo memorizzato i valori di h e gli errori corrispondenti nei vettori `H` e `Err`, con l'istruzione

```
p=log(abs(Err(1:end-1)./Err(2:end)))./...
      (log(H(1:end-1)./H(2:end)));
p(1:2:end)'
```

otteniamo

2.0213	2.0119	2.0083	2.0064
--------	--------	--------	--------

ovvero ordine di convergenza 2 rispetto a h .

Soluzione 9.9 Richiamiamo il Programma 9.1 con $N = 200, 400, 800$, dando le seguenti istruzioni MATLAB:

```
a=0;b=1; bvpfun=@(x)1+0*x;
mu=1/1000; eta=1; sigma=0; alpha=0; beta=0;
figure(1); clf
color=['c: ';'c--';'c- '];
k=1;
for N=[200,400,800]
h=(b-a)/(N+1);
[xh,uh]=bvp_fd_dir_1d(a,b,N,mu,eta,sigma, ...
    bvpfun,alpha,beta);
plot(xh,uh,color(k,:),'Linewidth',2); hold on
grid on; axis([0.9,1,0.,1.5])
k=k+1;
end
```

Per $N = 200$ sono evidenti le oscillazioni della soluzione numerica (si veda la Figura 10.24, a sinistra), infatti $\text{Pe} = 500/201 > 1$ e lo schema alle differenze finite centrate (per approssimare sia la derivata seconda che la derivata prima) è instabile. Le oscillazioni sono meno evidenti per $N = 400$ e svaniscono completamente per $N = 800$, infatti in questi due casi otteniamo $\text{Pe} = 500/401 > 1$ e $\text{Pe} = 500/801 < 1$, rispettivamente.

Il Programma 10.7 implementa lo schema *upwind* (9.39). Gli input e gli output assumono lo stesso significato di quelli del Programma 9.1.

Programma 10.7. bvp_fd_upwind_1d: approssimazione del problema di diffusione-trasporto con differenze finite upwind

```
function [xh,uh]=bvp_fd_upwind_1d(a,b,N,mu,eta,sigma, ...
    bvpfun,ua,ub,varargin)
% BVP_FD_UPWIND_1D Risolve il problema di diffusione-
% trasporto e condizioni di Dirichlet su (a,b)
% implementando lo schema upwind (differenze finite
% all'indietro per il termine del trasporto, con eta>0).
% [xh,uh]=bvp_fd_upwind_1d(a,b,N,mu,eta,sigma, ...
%     bvpfun,ua,ub)
%
h = (b-a)/(N+1);
xh = (linspace(a,b,N+2))';
hm = mu/h^2;
hd = eta/h;
e = ones(N,1);
Afd = spdiags([-(-hm+hd)*e (2*hm+hd+sigma)*e -hm*e],...
    -1:1, N, N);
xi = xh(2:end-1);
f =bvpfun(xi,varargin{:});
f(1) = f(1)+ua*(hm+hd);
f(end) = f(end)+ub*(hm-hd);
uh = Afd\f;
uh=[ua; uh; ub];
```

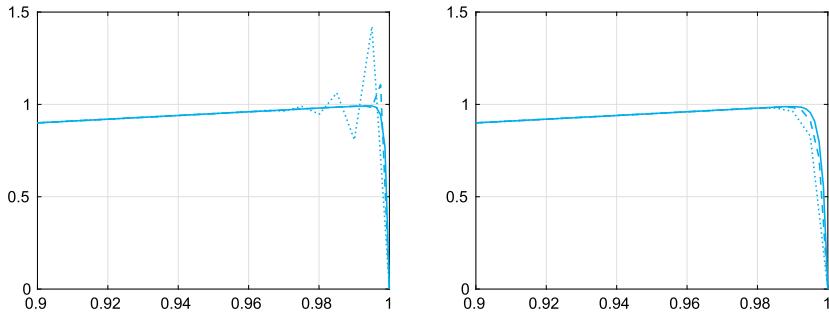


Figura 10.24. Le soluzioni dell’Esercizio 9.9 utilizzando lo schema centrale (9.36) (a sinistra) e lo schema upwind (9.39) (a destra). $N = 200$ (linea continua), $N = 500$ (linea tratteggiata) e $N = 800$ (linea punteggiata)

Richiamiamo quindi il Programma 10.7, sempre con $N = 200, 400, 800$, ripetendo le istruzioni precedenti (cambiamo solo la chiamata al Programma). Come possiamo osservare dalla Figura 10.24, a destra, ora le oscillazioni sono assenti anche per $N = 200$.

Soluzione 9.10 La formula di integrazione del trapezio, applicata su ciascun intervallo I_{j-1} e I_j fornisce il seguente valore:

$$\int_{I_{j-1} \cup I_j} f(x)\varphi_k(x)dx = \frac{h}{2}f(x_k) + \frac{h}{2}f(x_k) = hf(x_k),$$

essendo $\varphi_j(x_i) = \delta_{ij}$ per ogni i, j . Quando $j = 1$ o $j = N$ possiamo procedere in maniera analoga tenendo conto delle condizioni al bordo di Dirichlet. Osserviamo che abbiamo ottenuto lo stesso termine noto del sistema alle differenze finite (9.19) a meno di un fattore h .

Soluzione 9.11 Anzitutto osserviamo che l’interpolatore composito lineare di f nei nodi x_j può essere scritto come

$$\Pi_1^h f(x) = \sum_{i=0}^N f(x_i)\varphi_i(x),$$

essendo φ_j le funzioni di base definite in (9.44). Quindi, per ogni $j = 1, \dots, N$, abbiamo

$$\begin{aligned} \int_{I_{j-1} \cup I_j} f(x)\varphi_j(x)dx &\simeq \int_{I_{j-1} \cup I_j} \Pi_1^h f(x)\varphi_j(x)dx \\ &= \sum_{i=0}^N f(x_i) \int_{I_{j-1} \cup I_j} \varphi_i(x)\varphi_j(x)dx. \end{aligned}$$

Poiché l’espressione delle φ_i è nota, gli integrali che coinvolgono le sole funzioni di forma possono essere facilmente calcolati in maniera esatta e memorizzati nella cosiddetta *matrice di massa* $M \in \mathbb{R}^{N \times N}$,

$$\mathbf{M}_{ji} = \int_a^b \varphi_i(x) \varphi_j(x) dx = \int_{I_{j-1} \cup I_j} \varphi_i(x) \varphi_j(x) dx, \quad i, j = 1, \dots, N, \quad (9.9)$$

ovvero

$$\mathbf{M} = \frac{h}{6} \begin{bmatrix} 4 & 1 & 0 & \dots & 0 \\ 1 & 4 & \ddots & & \vdots \\ 0 & \ddots & \ddots & 1 & 0 \\ \vdots & & 1 & 4 & 1 \\ 0 & \dots & 0 & 1 & 4 \end{bmatrix}. \quad (9.10)$$

Infine il termine noto \mathbf{f}_{fe} di (9.48) può essere ottenuto con una operazione di prodotto matrice vettore tra la matrice di massa \mathbf{M} ed il vettore $(f(x_1), \dots, f(x_N))^T$.

L'errore di interpolazione che si commette è infinitesimo del secondo ordine rispetto a h (si veda la Proposizione 3.3), non avendo commesso errori nell'integrazione successiva, concludiamo che l'approssimazione del termine noto di (9.47) con questa procedura ha ordine di accuratezza due rispetto a h .

Osserviamo che la matrice \mathbf{M} che abbiamo costruito in (9.10) coinvolge le sole funzioni di forma $\varphi_1, \dots, \varphi_N$ associate ai nodi interni all'intervallo (a, b) ; se volessimo includere anche le due funzioni di base estreme φ_0 e φ_{N+1} , essa avrebbe dimensione $N+2$ anziché N e gli elementi diagonali in prima ed ultima riga sarebbero 2 anziché 4.

Soluzione 9.12 Sia u_h la soluzione ad elementi finiti calcolata come descritto nella Sezione 9.5 e u una funzione nota, dobbiamo calcolare (9.50). Ricordando (9.46) e (9.44) abbiamo:

$$\begin{aligned} err^2 &= \int_a^b (u(x) - u_h(x))^2 dx = \sum_{j=0}^N \int_{x_j}^{x_{j+1}} (u(x) - u_h(x))^2 dx \\ &= \sum_{j=0}^N \int_{x_j}^{x_{j+1}} \left(u(x) - \sum_{i=0}^N u_i \varphi_i(x) \right)^2 dx \\ &= \sum_{j=0}^N \int_{x_j}^{x_{j+1}} (u(x) - u_j \varphi_j(x) - u_{j+1} \varphi_{j+1}(x))^2 dx \\ &= \sum_{j=0}^N \int_{x_j}^{x_{j+1}} \left(u(x) - u_j \frac{x - x_{j+1}}{x_j - x_{j+1}} - u_{j+1} \frac{x - x_j}{x_{j+1} - x_j} \right)^2 dx. \end{aligned}$$

Denotiamo con $e(x)$ la funzione integranda e approssimiamo l'integrale sull'intervallo $I_j = [x_j, x_{j+1}]$ con una formula di quadratura interpolatoria di Gauss-Legendre del tipo (4.32) con $n = 4$ i cui nodi e pesi di quadratura sull'intervallo di riferimento $(-1, 1)$ sono forniti in Tabella 4.1 e sono rimappati su un intervallo generico (a, b) mediante le formule (4.33). Si ha:

$$\begin{aligned} Int_j &= \int_{x_j}^{x_{j+1}} e(x) dx \simeq \sum_{i=0}^n e(y_i) \alpha_i \\ &= \sum_{i=0}^n e\left(\frac{x_{j+1} - x_j}{2}\bar{y}_i + \frac{x_{j+1} + x_j}{2}\right) \frac{x_{j+1} - x_j}{2} \bar{\alpha}_i \end{aligned}$$

Il calcolo di Int_j è realizzato nella seguente *function*:

```
function int=gl4c(u,uj,uj1,xj,xj1)
% int=gl4c(u,uj,u2,x1,x2) integrale con Gauss-Legendre
% calcola l'integrale di  $(u(x)-u_h(x))^2$ 
% sull'intervallo  $[x_j, x_{j+1}]$ , dove
%  $u_h(x)=u_j\phi_j(x)+u_{j+1}\phi_{j+1}(x)$ 
% e' la soluzione elementi finiti di grado 1
ybar=[-sqrt(245+14*sqrt(70))/21;
      -sqrt(245-14*sqrt(70))/21; 0;
      sqrt(245-14*sqrt(70))/21;
      sqrt(245+14*sqrt(70))/21];
alphabar=[(322-13*sqrt(70))/900;
           (322+13*sqrt(70))/900; 128/225;
           (322+13*sqrt(70))/900;
           (322-13*sqrt(70))/900];
y=((xj1-xj)*ybar+(xj1+xj))/2;
alpha=alphabar*(xj1-xj)/2;
e=(u(y)-uj*(y-xj1)/(xj-xj1)-uj1*(y-xj)/(xj1-xj)).^2;
int=e'*alpha;
```

mentre il calcolo di err è svolto nella *function*

```
function [err]=norma_gl4c(u,xh,uh)
% [err]=norma_gl4c(u,xh,uh) calcolo errore nella norma
% integrale tra  $u(x)$  e  $u_h(x)$ .
% u e' un function handle, uh e' la soluzione elementi
% finiti di grado 1 calcolata sulla griglia xh
err=0; Ne=length(xh)-1;
for j=1:Ne
err=err+gl4c(u,uh(j),uh(j+1),xh(j),xh(j+1));
end
err=sqrt(err);
```

Soluzione 9.13 Per calcolare la temperatura al centro della piastra, risolviamo il corrispondente problema di Poisson per vari valori di $h = h_x = h_y$ dando le seguenti istruzioni:

```
k=0; fun=@(x,y) 25+0*x+0*y;
bound=@(x,y) (x==1);
for N = [10,20,40,80,160]
[xh,yh,uh]=poisson_fd_2d(0,1,0,1,N,N,fun,bound);
k=k+1; uc(k) = uh(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 =	2.0168	2.0616	2.0789	2.0859	2.089
------	--------	--------	--------	--------	-------

Possiamo quindi ritenere che la temperatura della piastra al centro sia di circa 2.09 °C. In Figura 10.25 riportiamo le linee di livello della soluzione calcolata per due diversi valori di h .

Soluzione 10.14 Per semplicità di scrittura poniamo $u_t = \partial u / \partial t$ e $u_x = \partial u / \partial x$. Moltiplichiamo l'equazione (9.101) con $f \equiv 0$ per u_t , integriamo su (a, b) e utilizziamo la formula di integrazione per parti sul secondo termine:

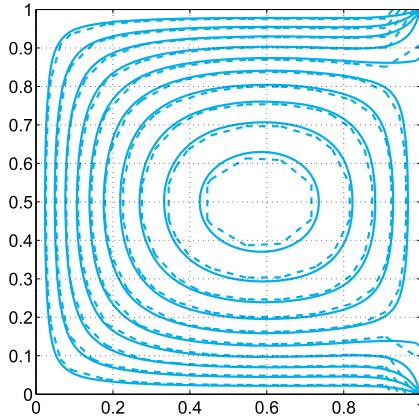


Figura 10.25. Le isoline della temperatura (dell'Esercizio 9.13) calcolate per $h = 1/10$ (linea tratteggiata) tive a $h = 1/80$ (linea continua)

$$\int_a^b u_{tt}(x, t)u_t(x, t)dx + c \int_a^b u_x(x, t)u_{tx}(x, t)dx - c[u_x(x, t)u_t(x, t)]_a^b = 0. \quad (10.11)$$

A questo punto integriamo in tempo sull'intervallo $(0, t)$ l'equazione (10.11). Osservando che $u_{tt}u_t = \frac{1}{2}(u_t^2)_t$ e $u_xu_{xt} = \frac{1}{2}(u_x^2)_t$, applicando il teorema fondamentale del calcolo integrale e ricordando le condizioni iniziali (9.103) (per cui $u_t(x, 0) = v_0(x)$ e $u_x(x, 0) = u_{0x}(x)$), otteniamo

$$\begin{aligned} & \int_a^b u_t^2(x, t)dx + c \int_a^b u_x^2(x, t)dx \\ &= \int_a^b v_0^2(x)dx + c \int_a^b u_{0x}^2(x)dx + 2c \int_0^t (u_x(b, s)u_t(b, s) - u_x(a, s)u_t(a, s))ds. \end{aligned}$$

D'altra parte, integrando per parti e ricordando che la soluzione u soddisfa condizioni al bordo di Dirichlet omogenee per $t > 0$ e la condizione iniziale $u_t(x, 0) = v_0(x)$, si ottiene

$$\int_0^t (u_x(b, s)u_t(b, s) - u_x(a, s)u_t(a, s))ds = 0.$$

Quindi, (9.114) è dimostrata.

Soluzione 9.15 Per la definizione (9.92) basta verificare che

$$\sum_{j=-\infty}^{\infty} |u_j^{n+1}|^2 \leq \sum_{j=-\infty}^{\infty} |u_j^n|^2. \quad (10.12)$$

Consideriamo la formula (9.90), portiamo tutti i termini a primo membro e moltiplichiamo per u_j^{n+1} . Grazie all'identità $2(a-b)a = a^2 - b^2 + (a-b)^2$ abbiamo

$$|u_j^{n+1}|^2 - |u_j^n|^2 + |u_j^{n+1} - u_j^n|^2 + \lambda a(u_{j+1}^{n+1} - u_{j-1}^{n+1})u_j^{n+1} = 0,$$

quindi sommiamo su j e osservando che $\sum_{j=-\infty}^{\infty} (u_{j+1}^{n+1} - u_{j-1}^{n+1})u_j^{n+1} = 0$ abbiamo

$$\sum_{j=-\infty}^{\infty} |u_j^{n+1}|^2 \leq \sum_{j=-\infty}^{\infty} |u_j^{n+1}|^2 + \sum_{j=-\infty}^{\infty} |u_j^{n+1} - u_j^n|^2 \leq \sum_{j=-\infty}^{\infty} |u_j^n|^2.$$

Soluzione 9.16 Lo schema *upwind* (9.87) può essere riscritto nella forma semplificata

$$u_j^{n+1} = \begin{cases} (1 - \lambda a)u_j^n + \lambda a u_{j-1}^n & \text{se } a > 0 \\ (1 + \lambda a)u_j^n - \lambda a u_{j+1}^n & \text{se } a < 0. \end{cases}$$

Consideriamo dapprima il caso $a > 0$. Se è soddisfatta la condizione CFL, allora entrambi i coefficienti $(1 - \lambda a)$ e λa sono positivi e minori di 1.

Questo implica

$$\min\{u_{j-1}^n, u_j^n\} \leq u_j^{n+1} \leq \max\{u_{j-1}^n, u_j^n\}$$

e, procedendo per ricorsione, anche

$$\inf_{l \in \mathbb{Z}}\{u_l^0\} \leq u_j^{n+1} \leq \sup_{l \in \mathbb{Z}}\{u_l^0\} \quad \forall n \geq 0,$$

da cui si ottiene la stima (9.116).

Quando $a < 0$, sempre grazie alla condizione CFL, entrambi i coefficienti $(1 + \lambda a)$ e $-\lambda a$ sono positivi e minori di 1. Procedendo analogamente a quanto fatto sopra si deduce ancora la stima (9.116).

Soluzione 9.17 Per risolvere numericamente il problema (9.75) possiamo utilizzare il Programma 10.8 sotto riportato. Osserviamo che la soluzione esatta del problema assegnato è l'onda viaggiante di velocità $a = 1$, ovvero $u(x, t) = 2 \cos(4\pi(x - t)) + \sin(20\pi(x - t))$. Essendo fissato il numero CFL pari a 0.5, i parametri di discretizzazione h e Δt saranno legati dalla proporzione $\Delta t = CFL \cdot h$ e quindi potremo scegliere arbitrariamente solo uno dei due parametri. La verifica dell'ordine di accuratezza rispetto a Δt potrà essere effettuata mediante le seguenti istruzioni:

```
xspan=[0,0.5]; tspan=[0,1]; a=1; cfl=0.5;
u0=@(x) 2*cos(4*pi*x)+sin(20*pi*x);
uex=@(x,t) 2*cos(4*pi*(x-t))+sin(20*pi*(x-t));
ul=@(t) 2*cos(4*pi*t)-sin(20*pi*t);
DT=[1.e-2,5.e-3,2.e-3,1.e-3,5.e-4,2.e-4,1.e-4];
e_lw=[]; e_up=[];
for deltat=DT
    h=deltat*a/cfl;
    [xx,tt,u_lw]=hyper(xspan,tspan,u0,ul,2,... 
        cfl,h,deltat);
    [xx,tt,u_up]=hyper(xspan,tspan,u0,ul,3,... 
        cfl,h,deltat);
    U=uex(xx,tt(end));
    [Nx,Nt]=size(u_lw);
    e_lw=[e_lw sqrt(h)*norm(u_lw(Nx,:)-U,2)];
    e_up=[e_up sqrt(h)*norm(u_up(Nx,:)-U,2)];
end
```

```

p_lw=log(abs(e_lw(1:end-1)./e_lw(2:end)))./...
    log(DT(1:end-1)./DT(2:end))
p_up=log(abs(e_up(1:end-1)./e_up(2:end)))./...
    log(DT(1:end-1)./DT(2:end))

p_lw =
0.1939 1.8626 2.0014 2.0040 2.0112 2.0239
p_up =
0.2272 0.3604 0.5953 0.7659 0.8853 0.9475

```

Operando analogamente un ciclo al variare di h , si verifica l'ordine di accuratezza rispetto alla discretizzazione in spazio. In particolare, facendo variare h tra 10^{-4} e 10^{-2} otteniamo

```

p_lw =
1.8113 2.0235 2.0112 2.0045 2.0017 2.0007
p_up =
0.3291 0.5617 0.7659 0.8742 0.9407 0.9734

```

Programma 10.8. hyper: gli schemi Lax-Friedrichs, Lax-Wendroff e upwind

```

function [xh,th,uh]=hyper(xspan,tspan,u0,ul, ...
    scheme,cfl,h,deltat)
% HYPER risolve un'eqz scalare iperbolica, a>0
% [XH,TH,UH]=HYPER(XSPAN,TSPAN,U0,UL,SCHEME,CFL, ...
% H,DELTAT)
% risolve l'equazione differenziale iperbolica scalare
% DU/DT+ A * DU/DX=0
% in (XSPAN(1),XSPAN(2))x(TSPAN(1),TSPAN(2))
% con condizione iniziale U(X,0)=U0(X) e
% condizione al bordo U(T)=UL(T) assegnata in XSPAN(1)
% con vari schemi alle differenze finite.
% scheme = 1 Lax - Friedrichs
%         2 Lax - Wendroff
%         3 Upwind
% La velocita' di propagazione A non e' richiesta
% esplicitamente, essendo CFL = A * DELTAT / DELTAX
% In output XH e' il vettore della discretizzazione
% in x; TH e' il vettore della discretizzazione in t
% UH e' una matrice che contiene la soluzione numerica:
% UH(n,:) contiene la sol all'istante temporale TT(n)
% U0 e UL possono essere inline o anonymous function o
% function definite tramite M-file.

Nt=(tspan(2)-tspan(1))/deltat+1;
th=linspace(tspan(1),tspan(2),Nt);
Nx=(xspan(2)-xspan(1))/h+1;
xh=linspace(xspan(1),xspan(2),Nx);
u=zeros(Nt,Nx); cfl2=cfl*0.5; cfl21=1-cfl^2;
cflp1=cfl+1; cflm1=cfl-1;
uh(1,:)=u0(xh);
for n=1:Nt-1
    uh(n+1,:)=ul(th(n+1));
    if scheme == 1
        % Lax Friedrichs
        for j=2:Nx-1
            uh(n+1,j)=0.5*(-cflm1*uh(n,j+1)+cflp1*uh(n,j-1));
        end
    else
        % Lax-Wendroff
        for j=2:Nx-1
            uh(n+1,j)=(1-cfl)*uh(n,j)+(cfl/2)*(uh(n,j+1)-uh(n,j-1));
        end
    end
end

```

```

    end
    j=Nx;
    uh(n+1,j)=0.5*(-cflm1*(2*uh(n,j)-uh(n,j-1))+...
        cflp1*uh(n,j-1));
elseif scheme == 2
% Lax-Wendroff
    for j=2:Nx-1
        uh(n+1,j)=cfl21*uh(n,j)+...
            cfl2*(cflm1*uh(n,j+1)+cflp1*uh(n,j-1));
    end
    j=Nx;
    uh(n+1,j)=cfl21*uh(n,j)+...
        cfl2*(cflm1*(2*uh(n,j)-uh(n,j-1))+cflp1*uh(n,j-1));
elseif scheme == 3
% Upwind
    for j=2:Nx
        uh(n+1,j)=-cflm1*uh(n,j)+cfl*uh(n,j-1);
    end
end

```

Soluzione 9.18 Osserviamo che la soluzione esatta del problema è la somma di due armoniche semplici, una a bassa frequenza e l'altra ad alta frequenza. Avendo scelto $\Delta t = 5 \cdot 10^{-2}$, poiché $a = 1$ e numero CFL = 0.8, si ha $h = 6.25e-3$ e quindi i due angoli di fase associati alle due armoniche sono $\phi_{k_1} = 4\pi \cdot 6.25e-3 \simeq 0.078$ e $\phi_{k_2} = 20\pi \cdot 6.25e-3 \simeq 0.393$. Dalla Figura 9.20 è evidente che lo schema *upwind* è più dissipativo dello schema di Lax-Wendroff. Ciò è confermato dall'andamento del coefficiente di dissipazione (si veda il grafico in basso a destra della Figura 9.16), infatti la curva associata allo schema di Lax-Wendroff si mantiene più vicina a 1 rispetto a quella associata allo schema *upwind* per i valori di ϕ_k corrispondenti alle armoniche in esame.

Per quanto riguarda il coefficiente di dispersione, dalla Figura 9.20 emerge che lo schema di Lax-Wendroff ha un ritardo di fase, mentre lo schema *upwind* ha un leggero anticipo di fase. Analizzando il grafico in basso a destra della Figura 9.17 troviamo conferma di ciò e possiamo anche concludere che il ritardo dello schema di Lax-Wendroff è maggiore di quanto non sia l'anticipo dello schema *upwind*.

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., Sorensen, D.: LAPACK User's Guide, 3rd edn. SIAM, Philadelphia (1999)
- [Ada90] Adair, R.: The Physics of Baseball. Harper and Row, New York (1990)
- [Arn73] Arnold, V.: Ordinary Differential Equations. MIT Press, Cambridge (1973)
- [Atk89] Atkinson, K.: An Introduction to Numerical Analysis, 2nd edn. Wiley, New York (1989)
- [Att16] Attaway, S.: MATLAB: A Practical Introduction to Programming and Problem Solving, 4th edn. Butterworth-Heinemann/Elsevier, Oxford/Waltham (2016)
- [Axe94] Axelsson, O.: Iterative Solution Methods. Cambridge University Press, Cambridge (1994)
- [BB96] Brassard, G., Bratley, P.: Fundamentals of Algorithmics. Prentice Hall, Englewood Cliffs (1996)
- [BC98] Bernasconi, A., Codenotti, B.: Introduzione Alla Complessità Computazionale. Springer-Verlag Italia, Milano (1998)
- [BDF⁺10] Bomze, I., Demyanov, V., Fletcher, R., Terlaky, T., Polik, I.: Nonlinear Optimization. Di Pillo, G., Schoen, F. (eds.) Lecture Notes in Mathematics, vol. 1989. Springer, Berlin (2010). Lectures given at the C.I.M.E. Summer School held in Cetraro, July 2007
- [Bec71] Beckmann, P.: A History of π , 2a edn. The Golem Press, Boulder (1971)
- [Ber82] Bertsekas, D.: Constrained Optimization and Lagrange Multipliers Methods. Academic Press, San Diego (1982)
- [BGL05] Benzi, M., Golub, G., Liesen, J.: Numerical solution of saddle point problems. *Acta Numer.* **14**, 1–137 (2005)

- [BM92] Bernardi, C., Maday, Y.: Approximations Spectrales des Problèmes aux Limites Elliptiques. Springer, Paris (1992)
- [Bom10] Bomze, M.: Global optimization: a quadratic programming perspective. In: Di Pillo, G., Schoen, F. (eds.) Lecture Notes in Mathematics, vol. 1989, pp. 1–53. Springer, Berlin (2010). Lectures given at the C.I.M.E. Summer School held in Cetraro, July 2007
- [BP98] Brin, S., Page, L.: The anatomy of a large-scale hypertextual web search engine. *Comput. Netw. ISDN Syst.* **33**, 107–117 (1998)
- [Bra97] Braess, D.: Finite Elements: Theory, Fast Solvers and Applications in Solid Mechanics. Cambridge University Press, Cambridge (1997)
- [Bre02] Brent, R.: Algorithms for Minimization Without Derivatives. Dover, Mineola (2002). Reprint of the 1973 original, Prentice-Hall, Englewood Cliffs
- [BS89] Bogacki, P., Shampine, L.: A 3(2) pair of Runge-Kutta formulas. *Appl. Math. Lett.* **2**(4), 321–325 (1989)
- [BS01] Babuska, I., Strouboulis, T.: The Finite Element Method and its Reliability. Numerical Mathematics and Scientific Computation. Clarendon Press/Oxford University Press, New York (2001)
- [BS08] Brenner, S., Scott, L.: The Mathematical Theory of Finite Element Methods, 3rd edn. Texts in Applied Mathematics, vol. 15. Springer, New York (2008)
- [BT04] Berrut, J.-P., Trefethen, L.-N.: Barycentric Lagrange interpolation. *SIAM Rev.* **46**(3), 501–517 (2004)
- [But87] Butcher, J.: The Numerical Analysis of Ordinary Differential Equations: Runge-Kutta and General Linear Methods. Wiley, Chichester (1987)
- [CFL28] Courant, R., Friedrichs, K., Lewy, H.: Über die partiellen Differenzengleichungen der mathematischen Physik. *Math. Ann.* **100**(1), 32–74 (1928)
- [Che04] Chen, K.: Matrix Preconditioning Techniques and Applications. Cambridge University Press, Cambridge (2004)
- [CHQZ06] Canuto, C., Hussaini, M.Y., Quarteroni, A., Zang, T.A.: Spectral Methods: Fundamentals in Single Domains. Scientific Computation. Springer, Berlin (2006)
- [CHQZ07] Canuto, C., Hussaini, M.Y., Quarteroni, A., Zang, T.A.: Spectral Methods. Evolution to Complex Geometries and Applications to Fluid Dynamics. Scientific Computation. Springer, Heidelberg (2007)
- [CL96a] Coleman, T., Li, Y.: An interior trust region approach for nonlinear minimization subject to bounds. *SIAM J. Optim.* **6**(2), 418–445 (1996)

- [CL96b] Coleman, T., Li, Y.: A reflective Newton method for minimizing a quadratic function subject to bounds on some of the variables. *SIAM J. Optim.* **6**(4), 1040–1058 (1996)
- [CLW69] Carnahan, B., Luther, H., Wilkes, J.: *Applied Numerical Methods*. Wiley, New York (1969)
- [Com95] Comincioli, V.: *Analisi Numerica Metodi Modelli Applicazioni*, 2a edn. McGraw-Hill Libri Italia, Milano (1995)
- [Dav63] Davis, P.: *Interpolation and Approximation*. Blaisdell/Ginn, Toronto/New York (1963)
- [Dav06] Davis, T.: *Direct Methods for Sparse Linear Systems*. SIAM, Philadelphia (2006)
- [dB01] de Boor, C.: *A Practical Guide to Splines*. Applied Mathematical Sciences. Springer, New York (2001)
- [DD99] Davis, T., Duff, I.: A combined unifrontal/multifrontal method for unsymmetric sparse matrices. *ACM Trans. Math. Software* **25**(1), 1–20 (1999)
- [Dem97] Demmel, J.: *Applied Numerical Linear Algebra*. SIAM, Philadelphia (1997)
- [Deu04] Deuflhard, P.: *Newton Methods for Nonlinear Problems. Affine Invariance and Adaptive Algorithms*. Springer Series in Computational Mathematics. Springer, Berlin (2004)
- [Die93] Dierckx, P.: *Curve and Surface Fitting with Splines*. Monographs on Numerical Analysis. Clarendon Press/Oxford University Press, New York (1993)
- [DL92] DeVore, R., Lucier, B.: Wavelets. *Acta Numer.* **1992**, 1–56 (1992)
- [DP80] Dormand, J., Prince, P.: A family of embedded Runge-Kutta formulae. *J. Comput. Appl. Math.* **6**(1), 19–26 (1980)
- [DR75] Davis, P., Rabinowitz, P.: *Methods of Numerical Integration*. Academic Press, New York (1975)
- [DS96] Dennis, J., Schnabel, R.: Numerical methods for unconstrained optimization and nonlinear equations. Classics in Applied Mathematics, vol. 16. Society for Industrial and Applied Mathematics, Philadelphia (1996)
- [EBHW15] Eaton, J.W., Bateman, D., Hauberg, S., Wehbring, R.: *GNU Octave version 4.0.0 manual: a high-level interactive language for numerical computations*. CreateSpace Independent Publishing Platform <http://www.gnu.org/software/octave/doc/interpreter> (2015)
- [EEHJ96] Eriksson, K., Estep, D., Hansbo, P., Johnson, C.: *Computational Differential Equations*. Cambridge University Press, Cambridge (1996)

- [EG04] Ern, A., Guermond, J.-L.: Theory and Practice of Finite Elements. Applied Mathematical Sciences., vol. 159. Springer, New York (2004)
- [Eva98] Evans, L.: Partial Differential Equations. American Mathematical Society, Providence (1998)
- [FGN92] Freund, R., Golub, G., Nachtigal, N.: Iterative solution of linear systems. *Acta Numer.* **1992**, 57–100 (1992)
- [Fle76] Fletcher, R.: Conjugate gradient methods for indefinite systems. In: Numerical Analysis, Proc. 6th Biennial Dundee Conf., Univ. Dundee, Dundee, 1975. Lecture Notes in Math., vol. 506, pp. 73–89. Springer, Berlin (1976)
- [Fle10] Fletcher, R.: The sequential quadratic programming method. In: Di Pillo, G., Schoen, F. (eds.) Lecture Notes in Mathematics vol. 1989, pp. 165–214. Springer, Berlin (2010). Lectures given at the C.I.M.E. Summer School held in Cetraro, July 2007
- [Fun92] Funaro, D.: Polynomial Approximation of Differential Equations. Springer, Berlin (1992)
- [Gau97] Gautschi, W.: Numerical Analysis. An Introduction. Birkhäuser Boston, Boston (1997)
- [Gea71] Gear, C.: Numerical Initial Value Problems in Ordinary Differential Equations. Prentice-Hall, Upper Saddle River (1971)
- [GI04] George, A., Ikramov, K.: Gaussian elimination is stable for the inverse of a diagonally dominant matrix. *Math. Comp.* **73**(246), 653–657 (2004)
- [GL96] Golub, G., Loan, C.V.: Matrix Computations, 3rd edn. John Hopkins University Press, Baltimore (1996)
- [GM72] Gill, P., Murray, W.: Quasi-Newton methods for unconstrained optimization. *J. Inst. Math. Appl.* **9**, 91–108 (1972)
- [GN06] Giordano, N., Nakanishi, H.: Computational Physics, 2nd edn. Prentice-Hall, Upper Saddle River (2006)
- [GOT05] Gould, N., Orban, D., Toint, P.: Numerical methods for large-scale nonlinear optimization. *Acta Numer.* **14**, 299–361 (2005)
- [GR96] Godlewski, E., Raviart, P.-A.: Hyperbolic Systems of Conservation Laws. Springer, New York (1996)
- [Hac85] Hackbusch, W.: Multigrid Methods and Applications. Springer Series in Computational Mathematics. Springer, Berlin (1985)
- [Hac16] Hackbusch, W.: Iterative Solution of Large Sparse Systems of Equations. Applied Mathematical Sciences. Springer, Switzerland (2016)

- [Hen79] Henrici, P.: Barycentric formulas for interpolating trigonometric polynomials and their conjugate. *Numer. Math.* **33**, 225–234 (1979)
- [Hes98] Hesthaven, J.: From electrostatics to almost optimal nodal sets for polynomial interpolation in a simplex. *SIAM J. Numer. Anal.* **35**(2), 655–676 (1998)
- [HH17] Higham, D., Higham, N.: MATLAB Guide, 3rd edn. SIAM, Philadelphia (2017)
- [Hig02] Higham, N.: Accuracy and Stability of Numerical Algorithms, 2nd edn. SIAM, Philadelphia (2002)
- [Hig04] Higham, N.-J.: The numerical stability of barycentric Lagrange interpolation. *IMA J. Numer. Anal.* **24**(4), 547–556 (2004)
- [Hir88] Hirsh, C.: Numerical Computation of Internal and External Flows. Wiley, Chichester (1988)
- [HLR14] Hunt, B., Lipsman, R., Rosenberg, J.: A Guide to MATLAB. For Beginners and Experienced Users, 3rd edn. Cambridge University Press, Cambridge (2014)
- [HRK04] Halliday, D., Resnick, R., Krane, K.: Fisica 2. Casa Editrice Ambrosiana, Milano (2004)
- [IK66] Isaacson, E., Keller, H.: Analysis of Numerical Methods. Wiley, New York (1966)
- [Joh90] Johnson, C.: Numerical Solution of Partial Differential Equations by the Finite Element Method. Cambridge University Press, Cambridge (1990)
- [Krö98] Kröner, D.: Finite volume schemes in multidimensions. In: Numerical Analysis 1997 (Dundee). Pitman Research Notes in Mathematics Series, pp. 179–192. Longman, Harlow (1998)
- [KS99] Karniadakis, G., Sherwin, S.: Spectral/hp Element Methods for CFD. Oxford University Press, New York (1999)
- [KW08] Kalos, M., Whitlock, P.: Monte Carlo Methods, 2nd edn. Wiley, New York (2008)
- [Lam91] Lambert, J.: Numerical Methods for Ordinary Differential Systems. Wiley, Chichester (1991)
- [Lan03] Langtangen, H.: Advanced Topics in Computational Partial Differential Equations: Numerical Methods and Diffpack Programming. Springer, Berlin (2003)
- [LeV02] LeVeque, R.: Finite Volume Methods for Hyperbolic Problems. Cambridge University Press, Cambridge (2002)
- [LM06] Langville, A., Meyer, C.: Google’s PageRank and Beyond: The Science of Search Engine Rankings. Princeton University Press, Princeton (2006)

- [LRWW99] Lagarias, J., Reeds, J., Wright, M., Wright, P.: Convergence properties of the Nelder-Mead simplex method in low dimensions. *SIAM J. Optim.* **9**(1), 112–147 (1999)
- [Mei67] Meinardus, G.: Approximation of Functions: Theory and Numerical Methods. Springer Tracts in Natural Philosophy. Springer, New York (1967)
- [MH03] Marchand, P., Holland, O.: Graphics and GUIs with MATLAB, 3rd edn. Chapman & Hall/CRC, London/New York (2003)
- [Mun07] Munson, T.: Mesh shape-quality optimization using the inverse mean-ratio metric. *Math. Program. A* **110**(3), 561–590 (2007)
- [Nat65] Natanson, I.: Constructive Function Theory, vol. III. Interpolation and Approximation Quadratures. Ungar, New York (1965)
- [NM65] Nelder, J., Mead, R.: A simplex method for function minimization. *Comput. J.* **7**, 308–313 (1965)
- [Noc92] Nocedal, J.: Theory of algorithms for unconstrained optimization. *Acta Numer.* **1992**, 199–242 (1992)
- [NW06] Nocedal, J., Wright, S.: Numerical Optimization, 2nd edn. Springer Series in Operations Research and Financial Engineering. Springer, New York (2006)
- [OR70] Ortega, J., Rheinboldt, W.: Iterative Solution of Nonlinear Equations in Several Variables. Academic Press, New York (1970)
- [Pal08] Palm, W.: A Concise Introduction to Matlab. McGraw-Hill, New York (2008)
- [Pan92] Pan, V.: Complexity of computations with matrices and polynomials. *SIAM Rev.* **34**(2), 225–262 (1992)
- [Pap87] Papoulis, A.: Probability, Random Variables, and Stochastic Processes. McGraw-Hill, New York (1987)
- [PBP02] Prautzsch, H., Boehm, W., Paluszny, M.: Bezier and B-Spline Techniques. Mathematics and Visualization. Springer, Berlin (2002)
- [PdDKÜK83] Piessens, R., de Doncker-Kapenga, E., Überhuber, C., Kahaner, D.: QUADPACK: A Subroutine Package for Automatic Integration. Springer Series in Computational Mathematics. Springer, Berlin (1983)
- [Pra16] Pratap, R.: Getting Started with MATLAB: A Quick Introduction for Scientists and Engineers, 7th edn. Oxford University Press, Oxford (2016)
- [QSS07] Quarteroni, A., Sacco, R., Saleri, F.: Numerical Mathematics, 2nd edn. Texts in Applied Mathematics. Springer, Berlin (2007)

- [QSSG14] Quarteroni, A., Sacco, R., Saleri, F., Gervasio, P.: *Matematica Numerica*, 4a edn. Springer, Milano (2014)
- [Qua16] Quarteroni, A.: *Modellistica Numerica per Problemi Differenziali*, 6a edn. Springer, Milano (2016)
- [QV94] Quarteroni, A., Valli, A.: *Numerical Approximation of Partial Differential Equations*. Springer, Berlin (1994)
- [QV99] Quarteroni, A., Valli, A.: *Domain Decomposition Methods for Partial Differential Equations*. Oxford University Press, London (1999)
- [Ros61] Rosenbrock, H.: An automatic method for finding the greatest or least value of a function. *Comput. J.* **3**, 175–184 (1960/1961)
- [RR01] Ralston, A., Rabinowitz, P.: *A First Course in Numerical Analysis*, 2nd edn. Dover, Mineola (2001)
- [RT83] Raviart, P., Thomas, J.: *Introduction à l'Analyse Numérique des Équations aux Dérivées Partielles*. Masson, Paris (1983)
- [Saa92] Saad, Y.: *Numerical Methods for Large Eigenvalue Problems*. Manchester University Press/Halsted/Wiley, Manchester/New York (1992)
- [Saa03] Saad, Y.: *Iterative Methods for Sparse Linear Systems*, 2nd edn. SIAM, Philadelphia (2003)
- [Sal10] Salsa, S.: *Equazioni a Derivate Parziali. Metodi, Modelli e Applicazioni*, 3a edn. Springer, Milano (2010)
- [SM03] Süli, E., Mayers, D.: *An Introduction to Numerical Analysis*. Cambridge University Press, Cambridge (2003)
- [SR97] Shampine, L., Reichelt, M.: The MATLAB ODE suite. *SIAM J. Sci. Comput.* **18**(1), 1–22 (1997)
- [SS86] Saad, Y., Schultz, M.: GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Statist. Comput.* **7**(3), 856–869 (1986)
- [SSB85] Shultz, G., Schnabel, R., Byrd, R.: A family of trust-region-based algorithms for unconstrained minimization with strong global convergence properties. *SIAM J. Numer. Anal.* **22**(1), 47–67 (1985)
- [Ste83] Steihaug, T.: The conjugate gradient method and trust regions in large scale optimization. *SIAM J. Numer. Anal.* **20**(3), 626–637 (1983)
- [Str07] Stratton, J.: *Electromagnetic Theory*. Wiley/IEEE Press, Hoboken/New Jersey (2007)
- [SY06] Sun, W., Yuan, Y.-X.: *Optimization Theory and Methods. Nonlinear Programming*. Springer Optimization and Its Applications, vol. 1. Springer, New York (2006).
- [Ter10] Pólik, I., Terlaky, T.: Interior point methods for nonlinear optimization. In: Di Pillo, G., Schoen, F. (eds.) *Lecture Notes in Mathematics*, vol. 1989, pp. 215–276. Springer, Berlin

- (2010). Lectures given at the C.I.M.E. Summer School held in Cetraro, July 2007
- [Tho06] Thomée, V.: Galerkin Finite Element Methods for Parabolic Problems, 2nd edn. Springer Series in Computational Mathematics, vol. 25. Springer, Berlin (2006)
- [TW98] Tveito, A., Winther, R.: Introduction to Partial Differential Equations. A Computational Approach. Springer, Berlin (1998)
- [TW05] Toselli, A., Widlund, O.: Domain Decomposition Methods – Algorithms and Theory. Springer Series in Computational Mathematics, vol. 34. Springer, Berlin (2005)
- [Übe97] Überhuber, C.: Numerical Computation: Methods, Software, and Analysis. Springer, Berlin (1997)
- [Urb02] Urban, K.: Wavelets in Numerical Simulation. Lecture Notes in Computational Science and Engineering. Springer, Berlin (2002)
- [vdV03] van der Vorst, H.: Iterative Krylov Methods for Large Linear Systems. Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, Cambridge (2003)
- [VGCN05] Valorani, M., Goussis, D., Creta, F., Najm, H.: Higher order corrections in the approximation of low-dimensional manifolds and the construction of simplified problems with the CSP method. *J. Comput. Phys.* **209**(2), 754–786 (2005)
- [Wes04] Wesseling, P.: An Introduction to Multigrid Methods. Edwards, Philadelphia (2004)
- [Wil88] Wilkinson, J.: The Algebraic Eigenvalue Problem. Monographs on Numerical Analysis. Clarendon Press/Oxford University Press, New York (1988)
- [Zha99] Zhang, F.: Matrix Theory. Universitext. Springer, New York (1999)

Indice analitico

- `;`, 11
- `abs`, 9
- accuratezza, 105
- adattività, 108, 140, 327, 333, 334, 341
- algoritmo, 32
 - della fattorizzazione LU, 160
 - delle sostituzioni all'indietro, 159
 - delle sostituzioni in avanti, 159
 - di divisione sintetica, 75
 - di Gauss, 160
 - di Strassen, 33
 - di Thomas, 178, 379
 - di Winograd e Coppersmith, 33
- `aliasing`, 106
- `angle`, 9
- `anonymous function`, 19
- `ans`, 36
- approssimazione
 - di Galerkin, 386
- aritmetica
 - esatta, 8, 99, 197, 318
 - floating-point, 8, 99
- `array` di Butcher, 339
- arrotondamento, 4
- attesa, 144
- autovalore, 17, 221
- autovettore, 17, 221
- `axis`, 234
- banda
 - larghezza di, 168
- base, 5
- `bfgsmin`, 270
- `bicgstab`, 205
- `bim`, 430
- `broyden`, 309
- cancellazione, 7
- `cell`, 18
- `chol`, 167
- cifre significative, 5
- `clear`, 37
- `clock`, 35
- coefficiente
 - di amplificazione, 418
 - di dispersione, 419
 - di dissipazione, 418
 - di Fourier, 418
 - di viscosità artificiale, 415
- `compass`, 9
- complessità, 33
- `complex`, 9
- `cond`, 177
- `condestd`, 177
- condizione
 - CFL, 417, 429
 - delle radici, 342
 - di compatibilità, 377
 - di stabilità, 322
- condizioni
 - al bordo
 - di Dirichlet, 376
 - di Neumann, 377, 431

- di Karush–Kuhn–Tucker, 288
- di Lagrange, 289
- di ottimalità, 248, 288
- di Wolfe, 262, 263
- LICQ, 288
- conj**, 10
- consistenza, 312, 343, 381
 - di un metodo iterativo, 183
 - ordine di, 312
- contour**, 491
- conv**, 25
- convergenza, 30, 381
 - del metodo delle potenze, 229
 - del metodo di Eulero, 311
 - del metodo di Richardson, 189
 - di secanti, 59
 - di un metodo iterativo, 183, 184
 - globale, 258
 - lineare, 67
 - locale, 258
 - ordine di, 59
 - quadratica, 55
 - super-lineare, 59, 253
- cos**, 37
- costante
 - di Lebesgue, 95, 96, 98
 - di Lipschitz, 307, 314
- costo computazionale, 32
 - della fattorizzazione LU, 163
 - della regola di Cramer, 156
- cputime**, 34
- cross**, 16
- cumtrapz**, 134
- curve caratteristiche, 412
- curve Fitting**, 119
- Dahlquist
 - barriera di, 343, 346
- dblquad**, 146
- decomposizione in valori singolari, 117, 179, 180
- deconv**, 25
- deflazione, 76, 78, 240
- derivata
 - approssimazione di, 126
 - parziale, 59, 373
- det**, 13, 163, 217
- determinante, 13
 - calcolo del, 163
- diag**, 14
- diagonale principale, 12, 14
- diff**, 27
- differenze divise di Newton, 253
- differenze finite
 - all'indietro, 127
 - centrate, 127
 - in avanti, 126
 - in dimensione 1, 378, 383, 401, 413
 - in dimensione 2, 393
 - schema a 5 punti, 394
- differenziazione numerica, 126
- direzione di discesa, 191, 259
 - del gradiente, 260
 - del gradiente coniugato, 260
 - di Newton, 259
 - quasi-Newton, 260
- disp**, 38, 438
- dominio di dipendenza, 425
- dot**, 16
- double**, 129, 460
- drop tolerance*, 198
- eig**, 237
- eigs**, 239
- end**, 35
- eps**, 6, 7
- epsilon macchina, 6, 7, 439
- equazione
 - alle derivate parziali, 303
 - del calore, 374, 401, 407
 - del telegrafo, 376
 - delle onde, 374, 423
 - di Burgers, 413
 - di diffusione-trasporto, 383, 391
 - di diffusione-trasporto-reazione, 379
 - di Poisson, 373, 376
 - di trasporto, 411, 413, 422
 - differenziale ordinaria, 303
- equazioni
 - di Lotka–Volterra, 304, 351
 - normali, 116, 179
- errore
 - assoluto, 6
 - computazionale, 29
 - di arrotondamento, 5, 8, 29, 98, 170, 231, 318
 - di perturbazione, 329

di troncamento, 29, 399
 globale, 312
 locale, 311, 312, 343, 381, 403, 416
 relativo, 6, 208, 209
 stimatore dell', 31, 56, 140
 a posteriori, 336
 esponente, 5
 estrapolazione
 di Aitken, 71
 di Richardson, 148
etime, 34
exit, 36
exp, 37
eye, 11
fattore di convergenza asintotico, 67
fattorizzazione
 di Cholesky, 166, 232
 di Gauss, 161
 incompleta
 di Cholesky, 198
 LU, 206
 LU, 158, 232
 QR, 62, 179, 237, 269
fem-fenics, 399
FFT, 101
fft, 104
FFT, 104
fftshift, 105
figure, 234
fill-in, 167, 172
find, 51, 130, 494
fix, 438
floating point, 6
flusso
 di diffusione artificiale, 415
 numerico, 414
fminbnd, 253
fminsearch, 256
fminunc, 270, 279
for, 35, 39
format, 4
formula di Eulero, 9
formula di quadratura, 130
 aperta, 136, 462
 chiusa, 136
 composita
 del punto medio, 131
 del trapezio, 133
di Simpson, 134
di Gauss-Legendre, 137
di Gauss-Legendre-Lobatto, 138
di Newton-Cotes, 145
di Simpson adattiva, 140, 142
grado di esattezza di una, 132
interpolatoria, 135
ordine di accuratezza, 131
semplice
 del punto medio, 131
 del trapezio, 133
 di Simpson, 134
formulazione debole, 385
fplot, 20
fsolve, 18, 80, 81
full, 177
function, 20
 user-defined, 20
function, 40
function_handle, 89, 94
function handle, 19, 21
funtool, 28
funzione
 convessa, 190, 248
 costo, 243
 derivabile, 26
 derivata di, 26
 di forma, 387
 di incremento, 320, 340
 di iterazione, 65
 di penalizzazione, 291
 di Runge, 93
 fortemente convessa, 287
 grafico di, 20
 Lagrangiana, 288
 aumentata, 296
 lipschitziana, 248, 271, 307, 320
 obiettivo, 243
 primitiva, 26
 reale, 19
funzioni
 di base, 387
fzero, 22, 23, 79, 81
gallery, 214
Gershgorin
 cerchi di, 234
gmres, 205
gradiente, 248

grid, 20
griddata, 118
griddata3, 118
griddatan, 118

help, 37, 42
hold off, 234
hold on, 234

ichol, 199
ifft, 104
ilu, 206
imag, 10
image, 239
imread, 239
Inf, 6
instabilità, 95
int, 27
integrazione numerica, 130

- multidimensionale, 146
- su intervalli illimitati, 146

interp1, 108
interp1q, 108
interp3, 118
interp2, 118
interpft, 105
interpolatore, 89

- di Lagrange, 90, 91
- polinomiale, 89
- razionale, 89
- trigonometrico, 89, 101, 105

interpolazione

- baricentrica, 98
- composita, 109
- con funzioni *spline*, 108
- di Hermite, 112
- formula baricentrica, 98
- lineare composita, 107
- nodi di, 89
- polinomiale di Lagrange, 90

inv, 13

LAPACK, 182
larghezza di banda, 168
line search

- cubica, 264
- quadratica, 264

linspace, 21
load, 37
loglog, 30

logspace, 438
lsode, 347
ltfat, 119
lu, 163

m-file, 39
magic, 217
mantissa, 5
mass-lumping, 409
MAT&OCT, 2
matlabFunction, 89, 94
MAT||OCT, 2
matrice, 11

- a banda, 168, 211–213
- a dominanza diagonale, 165
- a dominanza diagonale stretta, 186, 188
- a rango pieno, 179
- ben condizionata, 177, 211
- bidiagonale, 178
- definita positiva, 166, 188
- di Google, 226
- di Hilbert, 174
- di iterazione, 184
- di Leslie, 223
- di massa, 408, 500
- di permutazione, 170
- di Riemann, 214
- di Vandermonde, 163
- di Wilkinson, 241
- diagonale, 14
- diagonalizzabile, 221
- hermitiana, 15
- Hessiana, 248
- identità, 12
- inversa, 13, 466
- invertibile, 13
- Jacobiana, 59, 359
- mal condizionata, 177, 209
- non singolare, 13
- ortogonale, 180
- pattern di, 168
- pseudoinversa, 181
- quadrata, 11
- radice quadrata di, 470
- rango di, 179
- semi definita positiva, 166
- simile, 17, 190
- simmetrica, 15, 166

- singolare, 13
- somma, 12
- sparsa, 168, 182, 211, 215, 226, 396
- spettro di, 226
- trasposta, 15
- triangolare
 - inferiore, 14
 - superiore, 14
- tridiagonale, 178, 379
- unitaria, 180
- media, 121
- media statistica, 144
- mesh**, 396
- meshgrid**, 118, 491
- metodi
 - di discesa, 192
 - di Krylov, 204
 - iterativi, 183
 - multigrid*, 217
 - multistep*, 342, 343
 - predictor-corrector*, 348
 - spettrali, 213, 430
 - metodo
 - θ -, 402
 - A-stabile, 325, 345
 - ad un passo, 308
 - assolutamente stabile
 - condizionatamente, 325
 - incondizionatamente, 325
 - backward difference formula* o BDF, 345
 - BFGS, 268
 - Bi-CGSTab, 205, 290
 - consistente, 312, 340, 343
 - convergente, 380, 399
 - degli elementi finiti, 385, 407, 422
 - dei minimi quadrati, 113, 114
 - del gradiente, 193, 200, 262
 - precondizionato, 199, 200
 - del gradiente coniugato, 196, 201, 213, 262
 - precondizionato, 201
 - delle iterazioni QR, 237
 - delle potenze, 226
 - delle potenze inverse, 231
 - delle potenze inverse con *shift*, 232
 - delle secanti, 58, 61
 - derivative free, 249
 - di Adams-Bashforth, 344
 - di Adams-Moulton, 344
 - di Aitken, 70–72
 - di Bairstow, 80
 - di barriera, 300
 - di bisezione, 50
 - di Broyden, 61
 - di Crank-Nicolson, 308, 403, 406
 - di Dekker-Brent, 79
 - di discesa, 249, 265
 - di eliminazione di Gauss, 162
 - di Eulero
 - all'indietro (o implicito), 308, 406
 - all'indietro/centrato, 415
 - in avanti adattivo, 323, 335
 - in avanti (o esplicito), 308
 - in avanti/centrato, 414
 - in avanti/decentrato, 415, 429
 - migliorato, 348
 - modificato, 371
 - di Galerkin, 386, 392, 407, 422
 - di Gauss-Newton, 281
 - damped, 282
 - di Gauss-Seidel, 188
 - di Heun, 348
 - di Hörner, 75
 - di interpolazione quadratica, 252
 - di Jacobi, 185
 - di Lax-Friedrichs, 414
 - di Lax-Wendroff, 414, 429
 - di Levenberg-Marquardt, 284
 - di Müller, 80
 - di Newmark, 355, 426
 - di Newton, 54, 60, 65
 - di Newton-Hörner, 77
 - di punto fisso, 65
 - di Richardson, 185
 - dinamico, 185
 - stazionario, 185
 - di rilassamento, 188
 - di Runge-Kutta, 339
 - adattivo, 341
 - stadi del, 339
 - di Steffensen, 71
 - differenze finite, 270
 - esplicito, 308
 - GMRES, 205, 214, 290
 - implicito, 308
 - L-stabile, 327

leap-frog, 354, 427
line search, 249, 259
 Monte Carlo, 143
 multifrontale, 216
predictor-corrector, 347
 quasi-Newton, 62
 SOR, 218
trust region, 249, 272, 284
upwind, 415, 429
mkpp, 110
 modello
 di Leontief, 153
 di Lotka e Leslie, 223
 moltiplicatori, 161
 di Lagrange, 274, 288
NaN, 8
nargin, 42
nargout, 42
nchoosek, 437
nnz, 17
 nodi
 di Chebyshev-Gauss, 97, 98
 di Chebyshev-Gauss-Lobatto, 96, 454
 di Gauss-Legendre, 137, 138
 di Gauss-Legendre-Lobatto, 138, 139
 di quadratura, 136
 d'interpolazione, 89
norm, 16, 383
 norma
 del massimo discreta, 381, 388
 dell'energia, 189, 192
 di Frobenius, 246
 di matrice, 177
 euclidea, 16, 174
 integrale, 390
Not-a-knot condition, 110
 numeri
 complessi, 9
 floating-point, 3, 5
 macchina, 3, 4
 reali, 3
 numero
 CFL, 417, 418
 di condizionamento
 dell'interpolazione, 95
 di matrice, 176, 177, 208, 211
 di Péclét
 globale, 383
 locale, 384
 nurbs, 119
ode13, 347
ode113, 350
ode15s, 347, 359, 361
ode23, 341, 342, 494
ode23s, 359, 361, 368
ode23tb, 359
ode45, 341, 342
ode54, 342
odepkg, 359
ones, 15
 operatore
 di Laplace, 373, 394
 gradiente, 430
 operatori
 booleani, 37, 38
 logici, 37, 38
 relazionali, 37
 short-circuit, 38
 operazioni
 elementari, 37
 punto, 16, 21
optimset, 18, 254
 ordinamento lessicografico, 394
 ordine di convergenza
 lineare, 67
 quadratica, 55, 258
 super-lineare, 59, 268
overflow, 6–8, 64
 passo di discretizzazione, 307
 adattivo, 327, 333–335
patch, 234
path, 39
pcg, 201
pchip, 112
pde, 399, 430
pdetool, 118, 215
 pesi
 di Gauss-Legendre, 137, 138
 di Gauss-Legendre-Lobatto, 138, 139
 di quadratura, 136
 piano
 complesso o di Gauss, 10, 75
 delle fasi, 351

- pivot*, 161
- pivoting*, 169
 - per righe, 170, 171
 - totale, 171, 173, 467
- plot**, 21, 30
- polinomi, 22, 23
 - caratteristici di Lagrange, 91, 136
 - di Legendre, 137
 - di Taylor, 26, 87
 - divisione di, 25, 76
 - nodali, 137
 - radici di, 24, 74
- polinomio
 - caratteristico
 - di matrice, 221
 - di un'equazione differenziale, 342
 - di interpolazione di Lagrange, 91
- poly**, 44, 94
- polyder**, 25, 95
- polyfit**, 25, 115
- polyint**, 25
- polyval**, 91
- ppval**, 110
- precondizionatore, 184, 197, 213
 - destro, 198
 - sinistro, 198
- pretty**, 436
- problema
 - ai limiti, 373
 - ai minimi quadrati
 - lineari, 179
 - di Cauchy, 306
 - lineare, 307, 356
 - lineare modello, 322
 - di diffusione-trasporto, 383, 391
 - di diffusione-trasporto-reazione, 379
 - di Dirichlet, 376
 - di Neumann, 377
 - di Poisson, 378, 385, 393
 - ellittico, 374
 - iperbolico, 374
 - mal condizionato, 79
 - minimi quadrati
 - non lineari, 280
 - parabolico, 374
 - stiff*, 356, 358
- prod**, 438
- prodotto
 - di matrici, 12
 - scalare, 16
 - vettore, 16
- programmazione quadratica, 289
- proiezione
 - di un vettore lungo una direzione, 192, 196
- prompt, 2
- punto
 - ammissibile, 286
 - di Cauchy, 277
 - di equilibrio, 351
 - di Karush–Kuhn–Tucker, 288
 - di minimo
 - globale, 247
 - globale vincolato, 286
 - locale, 247
 - locale vincolato, 286
 - regolare, 248
 - stazionario o critico, 248
- punto fisso, 64
 - iterazione di, 65
- quadl**, 138
- quit**, 36
- quiver**, 16
- quiver3**, 16
- radice
 - di una funzione, 22
 - multipla, 22, 24
 - semplice, 22, 55
- raggio spettrale, 184, 211
- rand**, 35
- real**, 10
- realmax**, 6
- realmin**, 6
- regola
 - di Armijo, 262
 - di Cartesio, 74
 - di Cramer, 156
 - di Laplace, 13
- residuo, 56, 177, 179, 185, 191, 208, 281
 - precondizionato, 185, 199
 - relativo, 193, 197
- retta di regressione, 115
- return**, 41, 438
- rmfield**, 18
- roots**, 24, 80

rpmak, 119
rsmak, 119
save, 37
scala
 lineare, 30, 32
 logaritmica, 30
 semilogaritmica, 31, 32
semi-discretizzazione, 401, 407
semilogx, 438
semilogy, 32
sequential Quadratic Programming,
 300
serie discreta di Fourier, 103
sezione aurea, 250
Shift, 232
simbolo di Kronecker, 90
simplesso, 254
simplify, 27, 469
sin, 37
sistema
 iperbolico, 424
 lineare, 151
 precondizionato, 197
 sottodeterminato, 160, 179
 sovradeterminato, 179
stiff, 358
 triangolare, 158
soluzione
 debole, 413
 ottimale, 192
sort, 229
sottospazio di Krylov, 204
sparse, 168
spdiags, 168, 178
spettro di una matrice, 226
Spline, 108
 cubica naturale, 109
spline, 110
spy, 168, 211, 396
sqrt, 37
stabilità, 313, 381
 A-, 345
 asintotica, 402
 assoluta, 319, 322
 regione di, 325, 346
 dell'interpolazione, 95
 L-, 327
 zero-, 313
statement
 case, 39
 ciclo *for*, 39
 ciclo *while*, 39
 condizionale, 37
 otherwise, 39
 switch, 39
stimator dell'errore, 31, 56, 140
strategia di backtracking, 263
struct, 18
successione di Fibonacci, 39, 45
successioni di Sturm, 80, 240
suitesparse, 182
sum, 437
SVD, 117, 179, 180
svd, 181
svds, 181
syms, 27
taylor, 27
taylortool, 88
tempo di CPU, 34
teorema
 degli zeri di una funzione
 continua, 50, 66
 del valor medio, 26, 319
 della divergenza (o di Gauss), 496
 della media integrale, 26
 di Abel, 74
 di Cauchy, 75
 di Lagrange, 26
 di Ostrowski, 67
 fondamentale del calcolo integrale,
 26
test d'arresto, 56, 238
 sul residuo, 57
 sull'incremento, 56
title, 234
toolbox, 2, 37
trapz, 134
trasformata rapida di Fourier, 104
tril, 14
triu, 14
UMFPACK, 182, 215
underflow, 6, 7
unità di arrotondamento, 6
valori singolari, 180
vander, 163

- `varargin`, 51
- variabili caratteristiche, 424
- varianza, 121, 456
- vettore
 - colonna, 11
 - riga, 11
 - trasposto coniugato, 16
- vettori
 - A-coniugati, 196
 - A-ortogonali, 196
 - linearmente indipendenti, 15
- vincolo
 - attivo, 286
 - di disuguaglianza, 286
 - di uguaglianza, 286
- viscosità artificiale, 391, 392
- wavelet*, 119
- `wavelet`, 119
- `while`, 39
- `wilkinson`, 242
- workspace base*, 37
- `xlabel`, 234
- `ylabel`, 234
- zero
 - di una funzione, 22
 - multiplo, 22
 - semplice, 22, 55
- `zeros`, 11, 15