9

Metodi numerici per problemi ai limiti stazionari ed evolutivi

I *problemi 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 *problemi ai limiti e ai valori iniziali*, o anche *problemi 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*

$$-u''(x) = f(x), \quad x \in (a, b), \tag{9.1}$$

o, in più dimensioni,

$$-\Delta u(\mathbf{x}) = f(\mathbf{x}), \quad \mathbf{x} = (x_1, \ldots, x_d)^T \in \Omega, \tag{9.2}$$

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 $\mathbf{x}^0$

A. Quarteroni et al., *Calcolo Scientifico*, 6a edizione,
UNITEXT – La Matematica per il 3+2 105, DOI 10.1007/978-88-470-3953-7_9

$$\frac{\partial u}{\partial x_i}(\mathbf{x}^0) = \lim_{h\to 0} \frac{u(\mathbf{x}^0 + h\mathbf{e}_i) - u(\mathbf{x}^0)}{h}, \tag{9.3}$$

dove $\mathbf{e}_i$ è l'i-esimo vettore unitario di $\mathbb{R}^d$.

2. L'*equazione del calore*

$$\frac{\partial u(x,t)}{\partial t} - \mu \frac{\partial^2 u(x,t)}{\partial x^2} = f(x,t), \quad x \in (a,b),\ t > 0, \tag{9.4}$$

o, in più dimensioni,

$$\frac{\partial u(\mathbf{x},t)}{\partial t} - \mu \Delta u(\mathbf{x},t) = f(\mathbf{x},t), \quad \mathbf{x} \in \Omega,\ t > 0, \tag{9.5}$$

dove $\mu > 0$ è un coefficiente assegnato che rappresenta la diffusività termica e f è nuovamente una funzione assegnata.

3. L'*equazione delle onde*

$$\frac{\partial^2 u(x,t)}{\partial t^2} - c \frac{\partial^2 u(x,t)}{\partial x^2} = 0, \quad x \in (a,b),\ t > 0, \tag{9.6}$$

o, in più dimensioni,

$$\frac{\partial^2 u(\mathbf{x},t)}{\partial t^2} - c \Delta u(\mathbf{x},t) = 0, \quad \mathbf{x} \in \Omega,\ t > 0, \tag{9.7}$$

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

$$\mathbf{q} = -K\nabla\phi, \tag{9.8}$$

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 $\mathrm{div}\mathbf{q} = 0$, dove $\mathrm{div}\mathbf{q}$ è la *divergenza* del vettore $\mathbf{q}$ ed è definita come

$$\mathrm{div}\mathbf{q} = \sum_{i=1}^{3} \frac{\partial q_i}{\partial x_i}$$

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 eguagli la velocità con cui il calore fluisce nel volume stesso, otteniamo l'equazione del calore

$$\rho c\frac{\partial T}{\partial t} = k\Delta T, \tag{9.9}$$

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

$$\rho c\frac{\partial T}{\partial t} = k\Delta T + f. \tag{9.10}$$

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

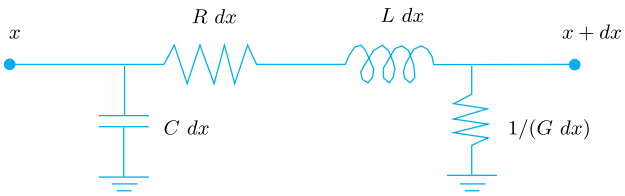

Figura 9.1. Un elemento del cavo di lunghezza dx

una capacità C ed una conduttanza G per unità di lunghezza (si veda la Figura 9.1), l'equazione per il potenziale v è

$$\frac{\partial^2 v}{\partial t^2} - c\frac{\partial^2 v}{\partial x^2} = -\alpha\frac{\partial v}{\partial t} - \beta v, \tag{9.11}$$

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

$$\begin{aligned} &-u''(x) = f(x) \quad \text{per } x \in (a,b), \\ &u(a) = \alpha, \qquad u(b) = \beta \end{aligned} \tag{9.12}$$

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

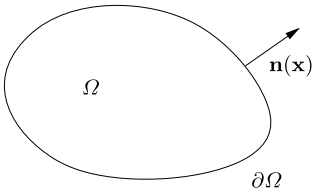

Figura 9.2. Un dominio bidimensionale Ω ed il versore normale al bordo con verso uscente

di compatibilità

$$\gamma - \delta = \int_a^b f(x)dx, \tag{9.13}$$

il *problema ai limiti di Neumann* è

$$\boxed{\begin{aligned} &-u''(x) = f(x) \quad \text{per } x \in (a,b), \\ &u'(a) = \gamma, \qquad u'(b) = \delta \end{aligned}} \tag{9.14}$$

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

$$\boxed{\begin{aligned} &-\Delta u(\mathbf{x}) = f(\mathbf{x}) \quad \text{per } \mathbf{x} \in \Omega, \\ &u(\mathbf{x}) = g_D(\mathbf{x}) \qquad \text{per } \mathbf{x} \in \partial\Omega \end{aligned}} \tag{9.15}$$

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

$$\int_{\partial\Omega} g_N \; d(\partial\Omega) = -\int_{\Omega} f d\Omega, \tag{9.16}$$

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 $j = 0, \ldots, N$ 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

$$-u''(x_j) = f(x_j), \quad j = 1, \ldots, N.$$

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 $u : [a, b] \to \mathbb{R}$ sufficientemente regolare in un intorno di un generico punto $\bar{x} \in (a, b)$, la quantità

$$\delta^2 u(\bar{x}) = \frac{u(\bar{x} + h) - 2u(\bar{x}) + u(\bar{x} - h)}{h^2} \tag{9.17}$$

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

$$\boxed{-\frac{u_{j+1} - 2u_j + u_{j-1}}{h^2} = f(x_j), \quad j = 1, \ldots, N} \tag{9.18}$$

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

$$\mathrm{A}_{\mathrm{fd}}\mathbf{u} = \mathbf{f}, \tag{9.19}$$

dove $\mathbf{u} = (u_1, \ldots, u_N)^T$ è il vettore delle incognite, $\mathbf{f} = (f(x_1) + \alpha/h^2, f(x_2), \ldots, f(x_{N-1}), f(x_N) + \beta/h^2)^T$ 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).

$$\mathrm{A}_{\mathrm{fd}} = h^{-2}\widehat{\mathrm{A}}, \tag{9.20}$$

essendo $\widehat{\mathrm{A}}$ la matrice tridiagonale

$$\widehat{\mathrm{A}} = \mathrm{tridiag}(-1, 2, -1) = \begin{bmatrix} 2 & -1 & 0 & \dots & 0 \\ -1 & 2 & \ddots & & \vdots \\ 0 & \ddots & \ddots & -1 & 0 \\ \vdots & & -1 & 2 & -1 \\ 0 & \dots & 0 & -1 & 2 \end{bmatrix}. \tag{9.21}$$

Tale sistema ammette un'unica soluzione in quanto 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 A_{fd} è malcondizionata. Infatti $K(\mathrm{A}_{\mathrm{fd}}) = \lambda_{\max}(\mathrm{A}_{\mathrm{fd}})/\lambda_{\min}(\mathrm{A}_{\mathrm{fd}}) = Ch^{-2}$, 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*:

$$\begin{cases} -\mu u''(x) + \eta u'(x) + \sigma u(x) = f(x) \quad \text{per } x \in (a, b), \\ u(a) = \alpha, \qquad u(b) = \beta, \end{cases} \tag{9.22}$$

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:

$$\begin{cases} -\mu \dfrac{u_{j+1} - 2u_j + u_{j-1}}{h^2} + \eta \dfrac{u_{j+1} - u_{j-1}}{2h} + \sigma u_j = f(x_j), \quad j = 1, \dots, N, \\ u_0 = \alpha, \qquad u_{N+1} = \beta. \end{cases} \tag{9.23}$$

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 `xh` dei nodi di discretizzazione e la soluzione calcolata `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

$$\max_{j=0,\ldots,N+1} |u(x_j) - u_j| \leq C(h) \quad \text{quando } h \to 0 \tag{9.24}$$

essendo $C(h)$ un infinitesimo rispetto a h, ovvero $\lim_{h\to 0} C(h) = 0$, 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

$$\max_{j=0,\dots,N+1} |u(x_j) - u_j| \leq \frac{(b-a)^2}{96} h^2 \max_{x\in[a,b]} |f''(x)| \tag{9.25}$$

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

$$\tau_h(x_j) = -\frac{u(x_{j+1}) - 2u(x_j) + u(x_{j-1})}{h^2} - f(x_j), \quad j = 1,\dots,N. \tag{9.26}$$

Quindi diciamo che un metodo è *consistente* se

$$\|\tau_h\| \to 0 \quad \text{quando } h \to 0 \tag{9.27}$$

e *consistente di ordine* q (con q un intero positivo) se $\|\tau_h\| = \mathcal{O}(h^q)$ per $h \to 0$, 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

$$\tau_h(x_j) = \frac{h^2}{12} u^{(4)}(\eta_j) + \mathcal{O}(h^4), \tag{9.28}$$

per un opportuno η_j in un intorno di x_j. Definendo la norma

$$\|\tau_h\| = \max_{j=1,\dots,N} |\tau_h(x_j)|, \tag{9.29}$$

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 discreta*.

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

$$\max_{j=1,\dots,N} |u_j| \leq C \max_{j=1,\dots,N} |f(x_j)| + |\alpha| + |\beta|. \tag{9.30}$$

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

$$e_j = u(x_j) - u_j, \quad j = 1, \dots, N \tag{9.31}$$

nei nodi e analizziamo le equazioni dell'errore

$$-\frac{e_{j+1} - 2e_j + e_{j-1}}{h^2} = \tau_h(x_j), \quad j = 1, \dots, N \tag{9.32}$$

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

$$\max_{j=0,\dots,N+1} |e_j| \leq C \max_{j=1,\dots,N} |\tau_h(x_j)| \leq \frac{C}{12} h^2 \max_{x\in[a,b]} |f''(x)|, \tag{9.33}$$

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 MATLAB. Infine, con le istruzioni:

`norm(v,inf)`

```
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 centrate. ■

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)

$$\boxed{\begin{aligned} &-\mu u''(x) + \eta u'(x) = f(x) \quad \text{per } x \in (a,b), \\ &u(a) = \alpha, \qquad u(b) = \beta \end{aligned}} \tag{9.34}$$

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

$$\mathbb{P}\mathrm{e}_{gl} = \frac{\eta(b-a)}{2\mu}, \tag{9.35}$$

rappresenta una misura di quanto il termine di trasporto domini quello diffusivo e diremo a trasporto dominante un problema in cui $\mathbb{P}\mathrm{e}_{gl} \gg 1$.

Una possibile discretizzazione di (9.34) è:

$$\begin{cases} -\mu\dfrac{u_{j+1} - 2u_j + u_{j-1}}{h^2} + \eta\dfrac{u_{j+1} - u_{j-1}}{2h} = f(x_j), \quad j = 1, \ldots, N, \\ u_0 = \alpha, \qquad u_{N+1} = \beta, \end{cases} \tag{9.36}$$

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

$$\max_{j=0,\dots,N+1} |u(x_j) - u_j| \leq Ch^2 \max_{x\in[a,b]} |f''(x)|. \tag{9.37}$$

La costante C, che è di fatto la costante di stabilità del metodo (9.36) (si veda (9.30)), è ora proporzionale a $\mathbb{Pe}_{gl}$, 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 Péclet locale* (o "di griglia")

$$\mathbb{Pe} = \frac{\eta h}{2\mu}. \tag{9.38}$$

Si può dimostrare che la soluzione del problema discreto (9.36) è priva di oscillazioni quando $\mathbb{Pe} < 1$ (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:

$$\begin{cases} -\mu\dfrac{u_{j+1} - 2u_j + u_{j-1}}{h^2} + \eta\dfrac{u_j - u_{j-1}}{h} = f(x_j), \quad j = 1, \dots, N, \\ u_0 = \alpha, \qquad u_{N+1} = \beta, \end{cases} \tag{9.39}$$

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].

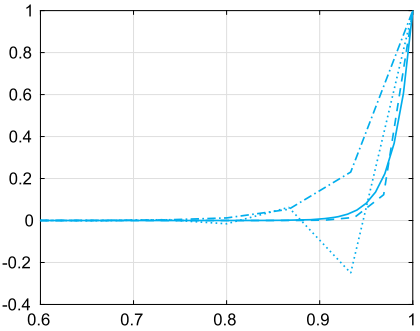

Figura 9.3. Soluzione esatta (*linea continua*), approssimazione con differenze finite centrate con $h = 1/15$ ($\mathbb{P}\mathrm{e} > 1$) (*linea punteggiata*), approssimazione con differenze finite centrate con $h = 1/32$ ($\mathbb{P}\mathrm{e} < 1$) (*linea tratteggiata*), approssimazione con differenze 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

$$\int_a^b u'(x)v'(x)dx = \int_a^b f(x)v(x)dx \tag{9.40}$$

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

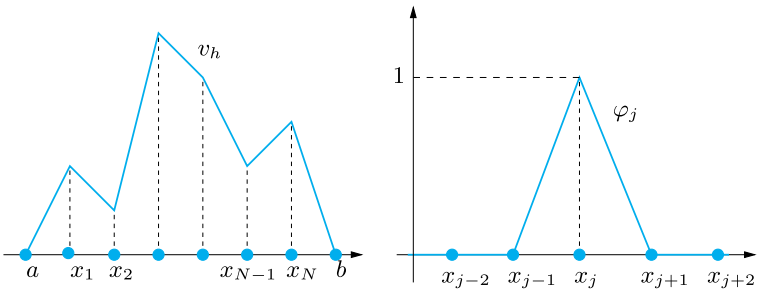

Figura 9.4. A sinistra, una generica funzione $v_h \in V_h^0$. A destra, la funzione di base per V_h^0 associata al nodo j-esimo

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:

$$\begin{aligned}&\text{trovare } u_h \in V_h \text{ tale che } u_h(a) = \alpha, u_h(b) = \beta \text{ e}\\ &\sum_{j=0}^{N} \int_{x_j}^{x_{j+1}} u_h'(x) v_h'(x) dx = \int_a^b f(x) v_h(x) dx \quad \forall v_h \in V_h^0\end{aligned} \tag{9.41}$$

dove

$$V_h = \left\{ v_h \in C^0([a,b]) : \ v_{h|I_j} \in \mathbb{P}_1, j = 0, \ldots, N \right\}, \tag{9.42}$$

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

$$\begin{aligned}&\text{trovare } u_h \in V_h^0 \text{ tale che}\\ &\sum_{j=0}^{N} \int_{x_j}^{x_{j+1}} u_h'(x) v_h'(x) dx = \int_a^b f(x) v_h(x) dx \quad \forall v_h \in V_h^0\end{aligned} \tag{9.43}$$

ed è noto come *approssimazione di Galerkin* di (9.40) con $\alpha = \beta = 0$.

Le funzioni di V_h^0 sono polinomi compositi 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 $j = 1, \ldots, N$, $v_j = v_h(x_j)$ e

$$\varphi_j(x) = \begin{cases} \frac{x - x_{j-1}}{x_j - x_{j-1}} & \text{se } x \in I_{j-1}, \\ \frac{x - x_{j+1}}{x_j - x_{j+1}} & \text{se } x \in I_j, \\ 0 & \text{altrimenti.} \end{cases} \tag{9.44}$$

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, $j = 1, \ldots, N$ 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, $j = 1, \ldots, N$. Sfruttando la proprietà che φ_j si annulla al di fuori degli intervalli I_{j-1} e I_j, dalla (9.41) otteniamo

$$\int_{I_{j-1} \cup I_j} u_h'(x) \varphi_j'(x) dx = \int_{I_{j-1} \cup I_j} f(x) \varphi_j(x) dx, \quad j = 1, \ldots, N. \tag{9.45}$$

D'altra parte, ponendo $u_j = u_h(x_j)$, possiamo scrivere

$$u_h(x) = \sum_{j=0}^{N+1} u_j \varphi_j(x) = \sum_{j=1}^{N} u_j \varphi_j(x) + \alpha \varphi_0(x) + \beta \varphi_{N+1}(x), \tag{9.46}$$

dove $\varphi_0(x) = (x_1 - x)/(x_1 - a)$ per $a \le x \le x_1$ e $\varphi_0(x) = 0$ altrove, $\varphi_{N+1}(x) = (x - x_N)/(b - x_N)$ per $x_N \le x \le b$ e $\varphi_{N+1}(x) = 0$ altrove.

Sostituendo (9.46) in (9.45), troviamo:

$$\begin{aligned}
&u_1 \int_{I_0 \cup I_1} \varphi_1'(x) \varphi_1'(x) dx + u_2 \int_{I_1} \varphi_2'(x) \varphi_1'(x) dx \\
&\quad = \int_{I_0 \cup I_1} f(x) \varphi_1(x) dx - \alpha \int_{I_0} \varphi_0'(x) \varphi_1'(x) dx \\
&u_{j-1} \int_{I_{j-1}} \varphi_{j-1}'(x) \varphi_j'(x) dx + u_j \int_{I_{j-1} \cup I_j} \varphi_j'(x) \varphi_j'(x) dx \\
&\qquad + u_{j+1} \int_{I_j} \varphi_{j+1}'(x) \varphi_j'(x) dx \\
&\quad = \int_{I_{j-1} \cup I_j} f(x) \varphi_j(x) dx, \quad j = 2, \ldots, N-1, \\
&u_{N-1} \int_{I_{N-1}} \varphi_{N-1}'(x) \varphi_N'(x) dx + u_N \int_{I_{N-1} \cup I_N} \varphi_N'(x) \varphi_N'(x) dx \\
&\quad = \int_{I_{N-1} \cup I_N} f(x) \varphi_j(x) dx - \beta \int_{I_N} \varphi_{N+1}'(x) \varphi_N'(x) dx.
\end{aligned}$$

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

$$\begin{aligned} \frac{2u_1 - u_2}{h} &= \int_{I_0 \cup I_1} f(x)\varphi_1(x)dx + \frac{\alpha}{h}, \\ \frac{-u_{j-1} + 2u_j - u_{j+1}}{h} &= \int_{I_{j-1} \cup I_j} f(x)\varphi_j(x)dx, \\ & \qquad j = 2, \ldots, N-1, \\ \frac{-u_{N-1} + 2u_N}{h} &= \int_{I_{N-1} \cup I_N} f(x)\varphi_N(x)dx + \frac{\beta}{h}. \end{aligned} \tag{9.47}$$

Si trova dunque il sistema lineare

$$\mathrm{A}_{\mathrm{fe}}\mathbf{u} = \mathbf{f}_{\mathrm{fe}} \tag{9.48}$$

nell'incognita $\mathbf{u} = (u_1, \ldots, u_N)^T$. 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 $\mathrm{A}_{\mathrm{fe}} = h\mathrm{A}_{\mathrm{fd}}$) e il termine noto $\mathbf{f}_{\mathrm{fe}}$ 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 $h \to 0$, 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 $j = 1, \ldots, N$. Abbiamo

$$\begin{aligned}(\mathbf{f}_{\text{fe}})_j &= \int_{I_{j-1} \cup I_j} f(x)\varphi_j(x)dx \\ &= hf\left(\frac{x_{j-1}+x_j}{2}\right)\varphi_j\left(\frac{x_{j-1}+x_j}{2}\right) \\ &\quad + hf\left(\frac{x_j+x_{j+1}}{2}\right)\varphi_j\left(\frac{x_j+x_{j+1}}{2}\right) \\ &= \frac{h}{2}\left[f\left(\frac{x_{j-1}+x_j}{2}\right) + f\left(\frac{x_j+x_{j+1}}{2}\right)\right]. \end{aligned} \tag{9.49}$$

Nel caso in cui $f \in C^2([a,b])$, la si può sostituire con l'interpolatore composito lineare di Lagrange $\Pi_1^h f(x)$ (si veda la Sezione 3.4) che interpola 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 compositi 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

$$u(x) = \begin{cases} -x/24 - 1/64 & \text{se } x < -1/2, \\ x^4/12 & \text{se } -1/2 \le x \le 1/2 \\ x/24 - 1/64 & \text{se } x > 1/2 \end{cases}$$

valutiamo l'errore nella norma integrale

$$\|u - u_h\| = \left(\int_a^b (u(x) - u_h(x))^2 dx \right)^{1/2}. \tag{9.50}$$

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 MATLAB/OCTAVE:

```
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_gl4c(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_gl4c.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 $N \in \{199, 399, 799, 999\}$, scegliessimo $N \in \{200, 400, 800, 1000\}$, 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. ■

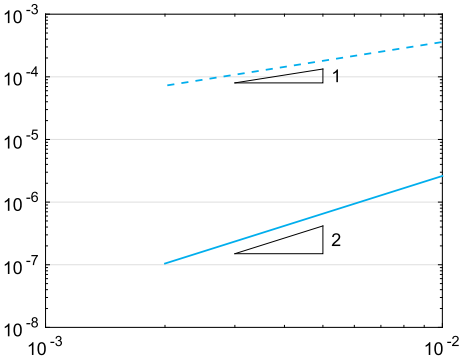

Figura 9.5. Gli errori (9.50) al variare di $h = 2/(N+1)$ con $N \in \{199, 399, 799, 999\}$ (*linea continua*) e con $N \in \{200, 400, 800, 1000\}$ (*linea tratteggiata*)

9.6 Approssimazione agli elementi finiti di un problema di diffusioni-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

$$\int_a^b \eta u_h'(x)\varphi_j(x)dx = \eta \frac{u_{j+1} - u_{j-1}}{2}, \quad j = 1, \ldots, N.$$

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

$$-\mu_h u''(x) + \eta u'(x) = f(x), \tag{9.51}$$

dove $\mu_h = (1 + \mathbb{P}\mathrm{e})\mu$. Il nuovo termine $\mathbb{P}\mathrm{e}\mu = \eta h/2$ 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.

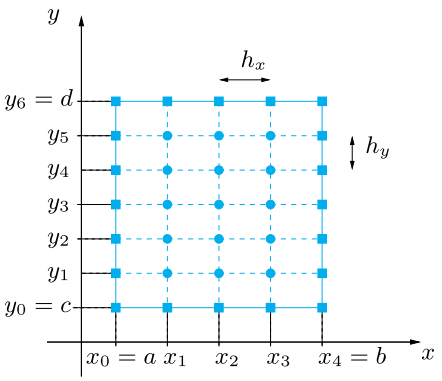

Figura 9.6. Griglia di calcolo Δ_h di 15 nodi interni su un dominio rettangolare

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 $i = 0, \ldots, N_x$, con $x_0 = a$ e $x_{N_x+1} = b$. Indichiamo con $\Delta_x = \{x_0, \ldots, x_{N_x+1}\}$ l'insieme degli estremi di tali intervalli e con $h_x = \max_{i=0,\ldots,N_x}(x_{i+1} - x_i)$ la loro massima lunghezza.

In modo del tutto analogo introduciamo una discretizzazione lungo l'asse y: $\Delta_y = \{y_0, \ldots, y_{N_y+1}\}$, con $y_0 = c$ e $y_{N_y+1} = d$ e $h_y = \max_{j=0,\ldots,N_y}(y_{j+1} - y_j)$. 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 $i = 0, \ldots, N_x + 1$ e $y_j = y_0 + jh_y$ per $j = 0, \ldots, N_y + 1$.

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

$$\begin{aligned}\delta_x^2 u_{i,j} &= \frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{h_x^2},\\ \delta_y^2 u_{i,j} &= \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{h_y^2}.\end{aligned} \tag{9.52}$$

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

$$-(\delta_x^2 u_{i,j} + \delta_y^2 u_{i,j}) = f(x_i, y_j), \quad i = 1, \ldots, N_x,\ j = 1, \ldots, N_y. \tag{9.53}$$

Ad esse vanno aggiunte le equazioni che impongono il dato di Dirichlet sul bordo $\partial\Omega$

$$u_{i,j} = g_D(x_i, y_j) \quad \forall i, j \text{ tale che } (x_i, y_j) \in \partial\Delta_h, \tag{9.54}$$

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

$$\begin{aligned}-\frac{1}{h^2}(u_{i-1,j} + u_{i,j-1} - 4u_{i,j} + u_{i,j+1} + u_{i+1,j}) = f(x_i, y_j),\\ i = 1, \ldots, N_x,\ j = 1, \ldots, N_y\end{aligned} \tag{9.55}$$

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 $\mathrm{A}_{\mathrm{fd}} \in \mathbb{R}^{N\times N}$ ha la seguente forma (tridiagonale a blocchi)

$$\mathrm{A}_{\mathrm{fd}} = \mathrm{tridiag}\,(\mathrm{D}, \mathrm{T}, \mathrm{D})\,. \tag{9.56}$$

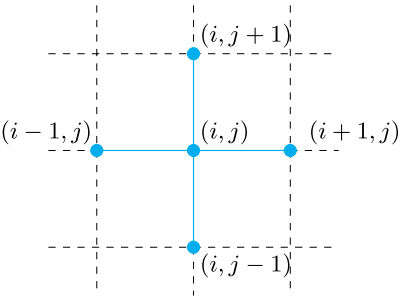

Figura 9.7. Supporto dello schema a 5 punti per l'operatore di Laplace

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, $\mathrm{D} \in \mathbb{R}^{N_x \times N_x}$ è la matrice diagonale i cui coefficienti diagonali valgono $-1/h_y^2$, mentre $\mathrm{T} \in \mathbb{R}^{N_x \times N_x}$ è la seguente matrice tridiagonale simmetrica

$$\mathrm{T} = \mathrm{tridiag}\left(-\frac{1}{h_x^2}, \frac{2}{h_x^2} + \frac{2}{h_y^2}, -\frac{1}{h_x^2}\right).$$

La matrice $\mathrm{A_{fd}}$ è simmetrica in quanto tutti i blocchi diagonali sono simmetrici. Verifichiamo che è anche definita positiva dimostrando che $\mathbf{v}^T \mathrm{A_{fd}} \mathbf{v} > 0$ 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

$$\mathbf{v}^T \mathrm{A_{fd}} \mathbf{v} = \sum_{k=1}^{N_y} \mathbf{v}_k^T \mathrm{T} \mathbf{v}_k - \frac{2}{h_y^2} \sum_{k=1}^{N_y - 1} \mathbf{v}_k^T \mathbf{v}_{k+1}. \tag{9.57}$$

Possiamo scrivere $\mathrm{T} = 2/h_y^2 \mathrm{I} + 1/h_x^2 \widehat{\mathrm{A}}$, dove $\widehat{\mathrm{A}}$ 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

$$\begin{aligned}
\mathbf{v}^T \mathrm{A_{fd}} \mathbf{v} &= \frac{1}{h_x^2} \sum_{k=1}^{N_y - 1} \mathbf{v}_k^T \widehat{\mathrm{A}} \mathbf{v}_k \\
&\quad + \frac{1}{h_y^2} \left(\mathbf{v}_1^T \mathbf{v}_1 + \mathbf{v}_{N_y}^T \mathbf{v}_{N_y} + \sum_{k=1}^{N_y - 1} (\mathbf{v}_k - \mathbf{v}_{k+1})^T (\mathbf{v}_k - \mathbf{v}_{k+1}) \right) \\
&\geq \frac{1}{h_x^2} \sum_{k=1}^{N_y - 1} \mathbf{v}_k^T \widehat{\mathrm{A}} \mathbf{v}_k
\end{aligned}$$

che è un numero reale strettamente positivo in quanto $\widehat{\mathrm{A}}$ è definita positiva ed almeno uno dei vettori $\mathbf{v}_k$ è non nullo.

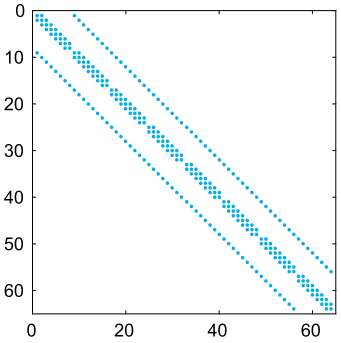

Figura 9.8. *Pattern* della matrice associata allo schema a 5 punti usando un ordinamento lessicografico delle incognite

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 MATLAB 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 $h \to 0$ 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 in *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
mesh 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`)

$$\texttt{error} = \max_{i,j} |u(x_i, y_j) - u_{i,j}| \Big/ \max_{i,j} |u(x_i, y_j)|. \tag{9.58}$$

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,uex,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;
aii = 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{:});
```

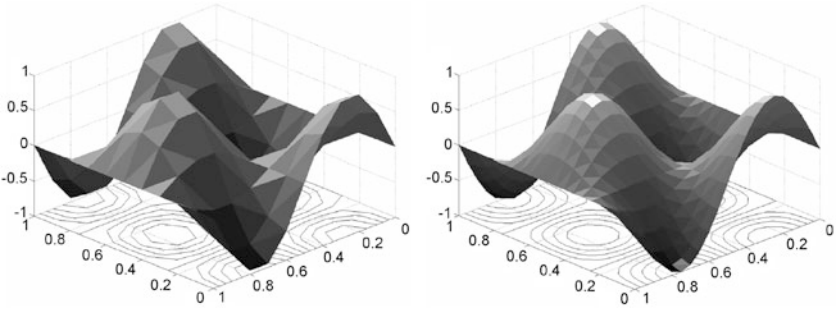

Figura 9.9. Spostamento trasversale di una membrana calcolato con una griglia più rada (*a sinistra*) e con una più fitta (*a destra*). Sul piano orizzontale vengono riportate le isolinee della soluzione. La decomposizione in triangoli che compare nelle figure tridimensionali serve per la sola visualizzazione

```
        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 $\Omega = (0,1)^2$, soggetta 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` di MATLAB o il *package* `fem-fenics` di Octave.

`pdetool`

`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

$$\max_{i,j} |u(x_i, y_j) - u_{i,j}| \leq C(h) \quad \text{quando } h \to 0 \tag{9.59}$$

essendo $C(h)$ un infinitesimo rispetto a h, ovvero $\lim_{h\to 0} C(h) = 0$, 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*

$$\max_{i,j} |u(x_i, y_j) - u_{i,j}| \leq CMh^2 \tag{9.60}$$

dove 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

$$\tau_h(x_i, y_j) = -f(x_i, y_j) - \frac{u(x_{i-1}, y_j) + u(x_i, y_{j-1}) - 4u(x_i, y_j) + u(x_i, y_{j+1}) + u(x_{i+1}, y_j)}{h^2} \tag{9.61}$$

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 $h \to 0$, 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]:

$$\max_{(i,j)\in D_h} |u_{i,j}| \leq \max_{(i,j)\in C_h} |g_D(x_i, y_j)| + C \max_{(i,j)\in D_h} |f(x_i, y_j)|, \tag{9.62}$$

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

$$\max_{(i,j)\in D_h\cup C_h} |e_{i,j}| \leq C \max_{(i,j)\in D_h} |\tau_h(x_i, y_j)| \leq \widetilde{C} h^2 \max_{(i,j)\in D_h} |f(x_i, y_j)|. \tag{9.63}$$

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 MATLAB/OCTAVE:

```
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 $h \to 0$. ■

Si veda l'Esercizio 9.13.

9.8 Approssimazione dell'equazione del calore monodimensionale

Consideriamo l'equazione del calore monodimensionale

$$\begin{cases} \dfrac{\partial u}{\partial t}(x,t) - \mu \dfrac{\partial^2 u}{\partial x^2}(x,t) = f(x,t) & x \in (a,b),\ t > 0, \\ u(a,t) = u(b,t) = 0 & t > 0, \\ u(x,0) = u^0(x) & x \in (a,b) \end{cases} \tag{9.64}$$

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 $j = 0, \ldots, N+1$, e approssimiamo il problema (9.64) con il seguente schema: per ogni $t > 0$:

$$\begin{cases} \dfrac{du_j}{dt}(t) - \dfrac{\mu}{h^2}(u_{j-1}(t) - 2u_j(t) + u_{j+1}(t)) = f_j(t), \quad j = 1, \ldots, N, \\ u_0(t) = u_{N+1}(t) = 0, \end{cases}$$

dove $f_j(t) = f(x_j,t)$, mentre per $t = 0$ imponiamo la condizione iniziale

$$u_j(0) = u^0(x_j), \quad j = 0, \ldots, N+1.$$

Questa è in realtà una *semi-discretizzazione* dell'equazione del calore, che produce un sistema di equazioni differenziali ordinarie del tipo:

$$\begin{cases} \dfrac{d\mathbf{u}}{dt}(t) = -\mu \mathrm{A}_{\mathrm{fd}}\mathbf{u}(t) + \mathbf{f}(t) \quad \forall t > 0, \\ \mathbf{u}(0) = \mathbf{u}^0, \end{cases} \tag{9.65}$$

dove $\mathbf{u}(t) = (u_1(t), \ldots, u_N(t))^T$ è il vettore colonna delle incognite, $\mathbf{f}(t) = (f_1(t), \ldots, f_N(t))^T$, $\mathbf{u}^0 = (u^0(x_1), \ldots, u^0(x_N))^T$ 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 temporare $t^k = k\Delta t$. Il θ-metodo prende la seguente forma:

$$\begin{aligned} &\frac{\mathbf{u}^{k+1} - \mathbf{u}^k}{\Delta t} = -\mu \mathrm{A}_{\mathrm{fd}}(\theta \mathbf{u}^{k+1} + (1-\theta)\mathbf{u}^k) + \theta \mathbf{f}^{k+1} + (1-\theta)\mathbf{f}^k, \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad k = 0, 1, \ldots \\ &\mathbf{u}^0 \text{ assegnato} \end{aligned} \tag{9.66}$$

o, equivalentemente,

$$(\mathrm{I} + \mu\theta\Delta t \mathrm{A}_{\mathrm{fd}})\,\mathbf{u}^{k+1} = (\mathrm{I} - \mu\Delta t(1-\theta)\mathrm{A}_{\mathrm{fd}})\,\mathbf{u}^k + \mathbf{g}^{k+1}, \tag{9.67}$$

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

$$\mathrm{K} = \mathrm{I} + \mu\theta\Delta t \mathrm{A}_{\mathrm{fd}}.$$

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 $t \to \infty$. 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

$$\mathbf{u}^k = (\mathrm{I} - \mu \Delta t \mathrm{A}_{\mathrm{fd}})^k \mathbf{u}^0, \quad k = 1, 2, \ldots$$

pertanto $\mathbf{u}^k \to \mathbf{0}$ per $k \to \infty$ se e solo se

$$\rho(\mathrm{I} - \mu \Delta t \mathrm{A}_{\mathrm{fd}}) < 1. \tag{9.68}$$

D'altro canto, gli autovalori λ_j di A_{fd} sono, per $j = 1, \ldots, N$ (si veda l'Esercizio 9.4):

$$\lambda_j = \frac{2}{h^2}\left(1 - \cos(j\pi/(N+1))\right) = \frac{4}{h^2}\sin^2(j\pi/(2(N+1))).$$

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 asintoticamante stabile sotto la condizione che Δt decresca come il quadrato del parametro h di discretizzazione spaziale quando $h \to 0$.

Nel caso del metodo di Eulero all'indietro ($\theta = 1$), da (9.67) otteniamo

$$\mathbf{u}^k = \left[(\mathrm{I} + \mu \Delta t \mathrm{A}_{\mathrm{fd}})^{-1}\right]^k \mathbf{u}^0, \quad k = 1, 2, \ldots$$

Poiché tutti gli autovalori della matrice $(\mathrm{I}+\mu\Delta t \mathrm{A}_{\mathrm{fd}})^{-1}$ 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 A_{fd} definita in (9.20) con la matrice 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 variable `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; u = 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 $\max_{i=0,\ldots,N} |u(x_i, 1) - u_i^M|$ 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

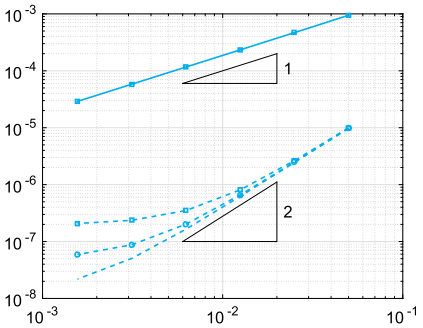

Figura 9.10. L'errore al variare di Δt per il θ-metodo (per $\theta = 1$ *in linea continua*, e $\theta = 0.5$ *in linea tratteggiata*), per tre differenti valori di h: 0.008 (□), 0.004 (○) e 0.002 (*nessun simbolo*)

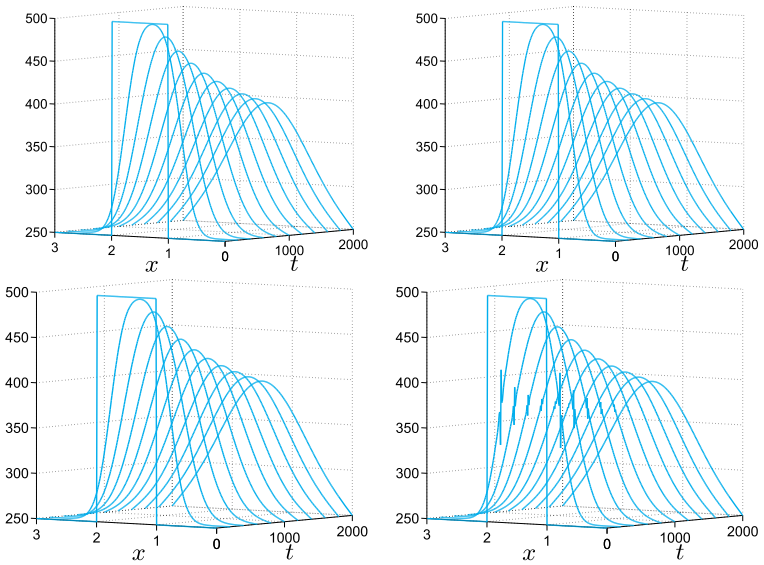

Figura 9.11. Profili di temperatura in una barra di alluminio a differenti istanti temporali (da $t = 0$ a $t = 2000$ secondi con passo di integrazione Δt di 0.25 secondi (*in alto*) e 20 secondi (*in basso*), ottenuti utilizzando gli schemi di Eulero all'indietro (*a sinistra*) e Crank-Nicolson (*a 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/m^3 e una capacità di calore specifico $c = 897$ J/(kg K), di conseguenza la diffusività termica è $\mu = 9.786 \cdot 10^{-5}$ m^2/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 $t \to \infty$. ■

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

$$\int_a^b \frac{\partial u}{\partial t}(x,t)v(x)dx + \int_a^b \mu \frac{\partial u}{\partial x}(x,t)\frac{dv}{dx}(x)dx \tag{9.69}$$

$$= \int_a^b f(x)v(x)dx \quad \forall v \in C^1([a,b]),$$

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

$$\int_a^b \frac{\partial u_h}{\partial t}(t)v_h dx + \int_a^b \mu \frac{\partial u_h}{\partial x}(t)\frac{dv_h}{dx}dx$$

$$= \int_a^b f(t)v_h dx \quad \forall v_h \in V_h^0, \tag{9.70}$$

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 $i = 1, \ldots, N$, si ottiene

$$\int_a^b \sum_{j=1}^N \frac{du_j}{dt}(t)\varphi_j\varphi_i dx + \mu \int_a^b \sum_{j=1}^N u_j(t)\frac{d\varphi_j}{dx}\frac{d\varphi_i}{dx}dx$$

$$= \int_a^b f(t)\varphi_i dx, \quad i = 1, \ldots, N$$

ossia

$$\sum_{j=1}^N \frac{du_j}{dt}(t)\int_a^b \varphi_j\varphi_i dx + \mu \sum_{j=1}^N u_j(t)\int_a^b \frac{d\varphi_j}{dx}\frac{d\varphi_i}{dx}dx$$

$$= \int_a^b f(t)\varphi_i dx, \quad i = 1, \ldots, N.$$

Utilizzando la medesima notazione impiegata nella (9.65), otteniamo

$$\mathrm{M}\frac{d\mathbf{u}}{dt}(t) + \mathrm{A_{fe}}\mathbf{u}(t) = \mathbf{f}_{\mathrm{fe}}(t), \tag{9.71}$$

dove $(\mathrm{A_{fe}})_{ij} = \mu\int_a^b \frac{d\varphi_j}{dx}\frac{d\varphi_i}{dx}dx$, $(\mathbf{f}_{\mathrm{fe}}(t))_i = \int_a^b f(t)\varphi_i dx$ e M è la *matrice di massa* i cui elementi sono $\mathrm{M}_{ij} = \int_a^b \varphi_j(x)\varphi_i(x)dx$. Osserviamo che, a differenza della formulazione alle differenze finite, qui il coefficiente μ è stato inglobato nella definizione della matrice $\mathrm{A_{fe}}$.

Per risolvere (9.71) in modo approssimato possiamo ancora ricorrere al θ-metodo, ottenendo:

$$\mathrm{M}\frac{\mathbf{u}^{k+1}-\mathbf{u}^k}{\Delta t} + \mathrm{A_{fe}}\left[\theta\mathbf{u}^{k+1} + (1-\theta)\mathbf{u}^k\right] = \theta\mathbf{f}_{\mathrm{fe}}^{k+1} + (1-\theta)\mathbf{f}_{\mathrm{fe}}^k, \quad k = 0, 1, \ldots$$
$$\mathbf{u}^0 \text{ assegnato} \tag{9.72}$$

o, equivalentemente,

$$\left(\frac{1}{\Delta t}\mathrm{M} + \theta\mathrm{A_{fe}}\right)\mathbf{u}^{k+1} = \left(\frac{1}{\Delta t}\mathrm{M} - (1-\theta)\mathrm{A_{fe}}\right)\mathbf{u}^k + \mathbf{g}^{k+1}, \tag{9.73}$$

avendo posto $\mathbf{g}^{k+1} = \theta\mathbf{f}_{\mathrm{fe}}^{k+1} + 1 - \theta\mathbf{f}_{\mathrm{fe}}^k$.

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

$$\mathrm{K} = \frac{1}{\Delta t}\mathrm{M} + \theta\mathrm{A_{fe}}.$$

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:

$$\begin{cases} R^T \mathbf{y} = \left[\dfrac{1}{\Delta t} M - (1-\theta) A_{fe}\right] \mathbf{u}^k + \theta \mathbf{f}_{fe}^{k+1} + (1-\theta)\mathbf{f}_{fe}^k, \\ R\mathbf{u}^{k+1} = \mathbf{y}. \end{cases}$$

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 $\widetilde{M}$.

Nel caso di elementi finiti di grado 1 per problemi monodimensionali, $\widetilde{M}$ 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 $\tilde{m}_{ij} = h\delta_{ij}$, $i, j = 1, \ldots, N$.

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

$$0 < \Delta t \leq C_1(\theta) h^2, \tag{9.74}$$

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);
```

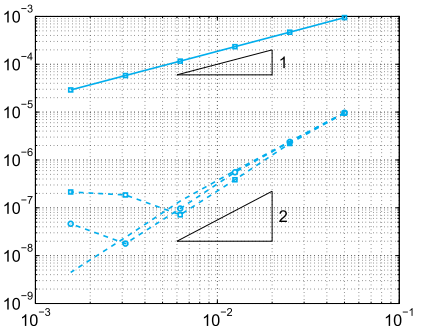

Figura 9.12. L'errore al variare di Δt per il θ-metodo e approssimazione agli elementi finiti in spazio (per $\theta = 1$ *in linea continua*, e $\theta = 0.5$ *in linea tratteggiata*), per tre differenti valori di h: 0.008 (□), 0.004 (○) e 0.002 (*nessun simbolo*)

```
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 $\max_{i=0,\ldots,N} |u(x_i,1) - u_i^M|$ 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

$$\begin{cases} \dfrac{\partial u}{\partial t} + a\dfrac{\partial u}{\partial x} = 0, & x \in \mathbb{R},\ t > 0, \\ u(x,0) = u^0(x), & x \in \mathbb{R}, \end{cases} \tag{9.75}$$

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

$$\begin{cases} \dfrac{dx}{dt}(t) = a, & t > 0, \\ x(0) = x_0, \end{cases} \tag{9.76}$$

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

$$\begin{cases} \dfrac{\partial u}{\partial t} + a\dfrac{\partial u}{\partial x} + a_0 u = f, & x \in \mathbb{R},\ t > 0, \\ u(x,0) = u^0(x), & x \in \mathbb{R}, \end{cases} \tag{9.77}$$

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 $[\alpha, \beta]$

$$\begin{cases} \dfrac{\partial u}{\partial t} + a\dfrac{\partial u}{\partial x} = 0, & x \in (\alpha, \beta),\ t > 0, \\ u(x,0) = u^0(x), & x \in (\alpha, \beta). \end{cases} \tag{9.78}$$

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 $t > 0$.

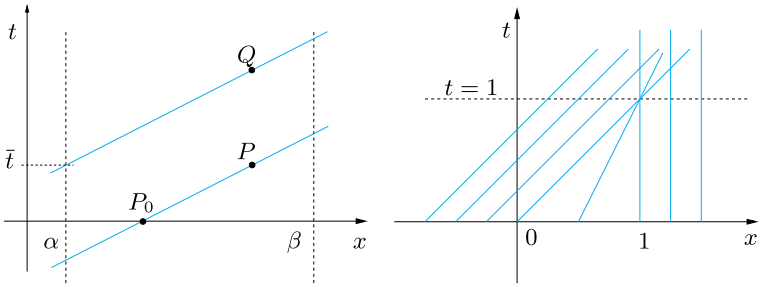

Figura 9.13. Esempi di caratteristiche che sono rette spiccate dai punti P e Q (*a sinistra*). Rette caratteristiche per l'equazione di Burgers (9.79) (*a destra*)

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

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = 0, \qquad x \in \mathbb{R},\ t > 0, \tag{9.79}$$

che rappresenta il più semplice esempio di equazione iperbolica non lineare. Prendendo come condizione iniziale

$$u(x,0) = u^0(x) = \begin{cases} 1, & x \le 0, \\ 1-x, & 0 < x \le 1, \\ 0, & x > 1, \end{cases}$$

la linea caratteristica spiccata dal punto $(x_0, 0)$ è data da

$$x(t) = x_0 + tu^0(x_0) = \begin{cases} x_0 + t, & x_0 \le 0, \\ x_0 + t(1-x_0), & 0 < x_0 \le 1, \\ x_0, & x_0 > 1. \end{cases}$$

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

$$x_j = jh, \quad j \in \mathbb{Z}, \qquad t^n = n\Delta t, \quad n \in \mathbb{N}.$$

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

$$u_j^{n+1} = u_j^n - \lambda(H_{j+1/2}^n - H_{j-1/2}^n), \tag{9.80}$$

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 espliciti per l'approssimazione del problema (9.75):

1. *Eulero in avanti/centrato*

$$u_j^{n+1} = u_j^n - \frac{\lambda}{2}a(u_{j+1}^n - u_{j-1}^n) \tag{9.81}$$

che può essere scritto nella forma (9.80) ponendo

$$H_{j+1/2}^n = \frac{1}{2}a(u_{j+1}^n + u_j^n); \tag{9.82}$$

2. *Lax-Friedrichs*

$$u_j^{n+1} = \frac{1}{2}(u_{j+1}^n + u_{j-1}^n) - \frac{\lambda}{2}a(u_{j+1}^n - u_{j-1}^n) \tag{9.83}$$

che risulta della forma (9.80) ponendo

$$H_{j+1/2}^n = \frac{1}{2}[a(u_{j+1}^n + u_j^n) - \lambda^{-1}(u_{j+1}^n - u_j^n)]; \tag{9.84}$$

3. *Lax-Wendroff*

$$u_j^{n+1} = u_j^n - \frac{\lambda}{2}a(u_{j+1}^n - u_{j-1}^n) + \frac{\lambda^2}{2}a^2(u_{j+1}^n - 2u_j^n + u_{j-1}^n) \tag{9.85}$$

che può essere scritto nella forma (9.80) con la scelta

$$H_{j+1/2}^n = \frac{1}{2}[a(u_{j+1}^n + u_j^n) - \lambda a^2(u_{j+1}^n - u_j^n)]; \tag{9.86}$$

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	$H^{diff}_{j+1/2}$	$\tau(\Delta t, h)$
Lax-Friedrichs	h^2	$-\frac{1}{2\lambda}(u_{j+1}-u_j)$	$\mathcal{O}\left(h^2/\Delta t+\Delta t+h^2\right)$
Lax-Wendroff	$a^2\Delta t^2$	$-\frac{\lambda a^2}{2}(u_{j+1}-u_j)$	$\mathcal{O}\left(\Delta t^2+h^2+\Delta t h^2\right)$
upwind	$\lvert a\rvert h\Delta t$	$-\frac{\lvert a\rvert}{2}(u_{j+1}-u_j)$	$\mathcal{O}(\Delta t+h)$

4. *Upwind (o Eulero in avanti/decentrato)*

$$u_j^{n+1}=u_j^n-\frac{\lambda}{2}a(u_{j+1}^n-u_{j-1}^n)+\frac{\lambda}{2}|a|(u_{j+1}^n-2u_j^n+u_{j-1}^n) \quad (9.87)$$

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

$$H_{j+1/2}=H^{EA}_{j+1/2}+H^{diff}_{j+1/2},$$

dove $H^{EA}_{j+1/2}$ è il flusso numerico relativo al metodo di Eulero in avanti/centrato (che è dato dalla (9.82)) mentre il *flusso di diffusione artificiale* $H^{diff}_{j+1/2}$ 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

$$\lim_{\Delta t, h \to 0} \max_{j,n} |u(x_j, t^n) - u_j^n| = 0.$$

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

$$\|\mathbf{u}^n\|_\Delta \leq C_T \|\mathbf{u}^0\|_\Delta, \tag{9.91}$$

per ogni n tale che $n\Delta t \leq T$ e per ogni Δt, 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:

$$\|\mathbf{v}\|_{\Delta,p} = \left(h \sum_{j=-\infty}^{\infty} |v_j|^p \right)^{\frac{1}{p}} \quad \text{per } p = 1, 2, \qquad \|\mathbf{v}\|_{\Delta,\infty} = \sup_j |v_j|. \tag{9.92}$$

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

$$|a\lambda| \leq 1, \quad \text{ovvero } \Delta t \leq \frac{h}{|a|} \tag{9.93}$$

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

$$\Delta t \leq \frac{h}{\sup\limits_{x\in\mathbb{R},\ t>0} |a(x,t)|}.$$

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 disuguaglianza 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

$$\alpha_k = \frac{1}{2\pi} \int_0^{2\pi} u^0(x) e^{-ikx} \mathrm{d}x$$

è il k-simo coefficiente di Fourier. La soluzione esatta u del problema (9.75) soddisfa (formalmente) le condizioni nodali

$$u(x_j, t^n) = \sum_{k=-\infty}^{\infty} \alpha_k e^{ikjh} (g_k)^n, \quad j \in \mathbb{Z},\ n \in \mathbb{N} \tag{9.94}$$

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

$$u_j^n = \sum_{k=-\infty}^{\infty} \alpha_k e^{ikjh} (\gamma_k)^n, \quad j \in \mathbb{Z},\ n \in \mathbb{N}. \tag{9.95}$$

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 disuguaglianza di stabilità (9.91).

Il rapporto

$$\epsilon_a(k) = |\gamma_k|/|g_k| = |\gamma_k| \tag{9.96}$$

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

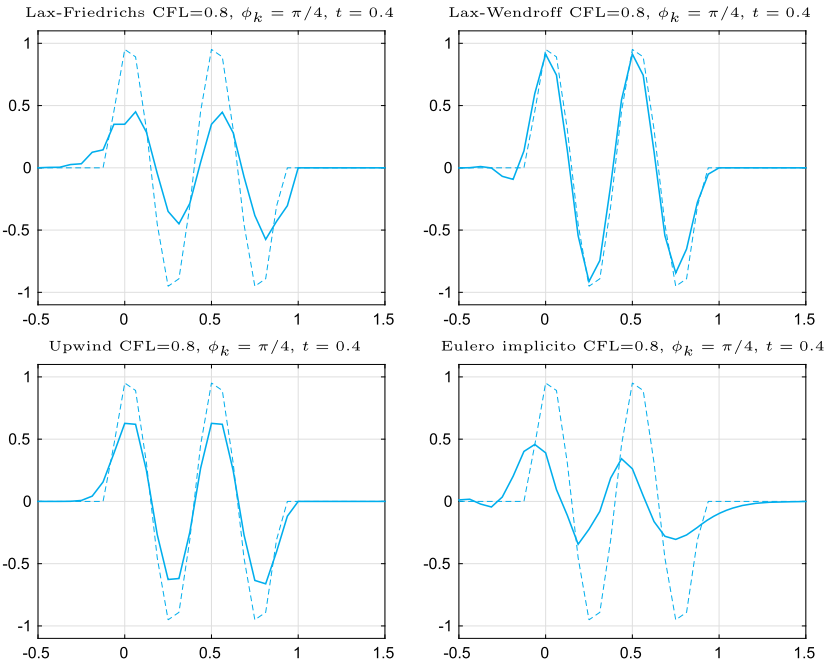

Figura 9.14. Soluzione esatta (*linea tratteggiata*) e soluzione numerica (*linea continua*), all'istante $t = 0.4$ e al variare di x, del problema (9.78) per $a = 1$ e dato iniziale definito in (9.98) con lunghezza d'onda $\ell = 1/2$

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:

$$g_k = e^{-ia\lambda\phi_k}, \qquad \gamma_k = |\gamma_k| e^{-i\omega\Delta t} = |\gamma_k| e^{-i\frac{\omega}{k}\lambda\phi_k},$$

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

$$\epsilon_d(k) = \frac{\omega}{ak} = \frac{\omega\Delta t}{\phi_k a\lambda}. \tag{9.97}$$

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

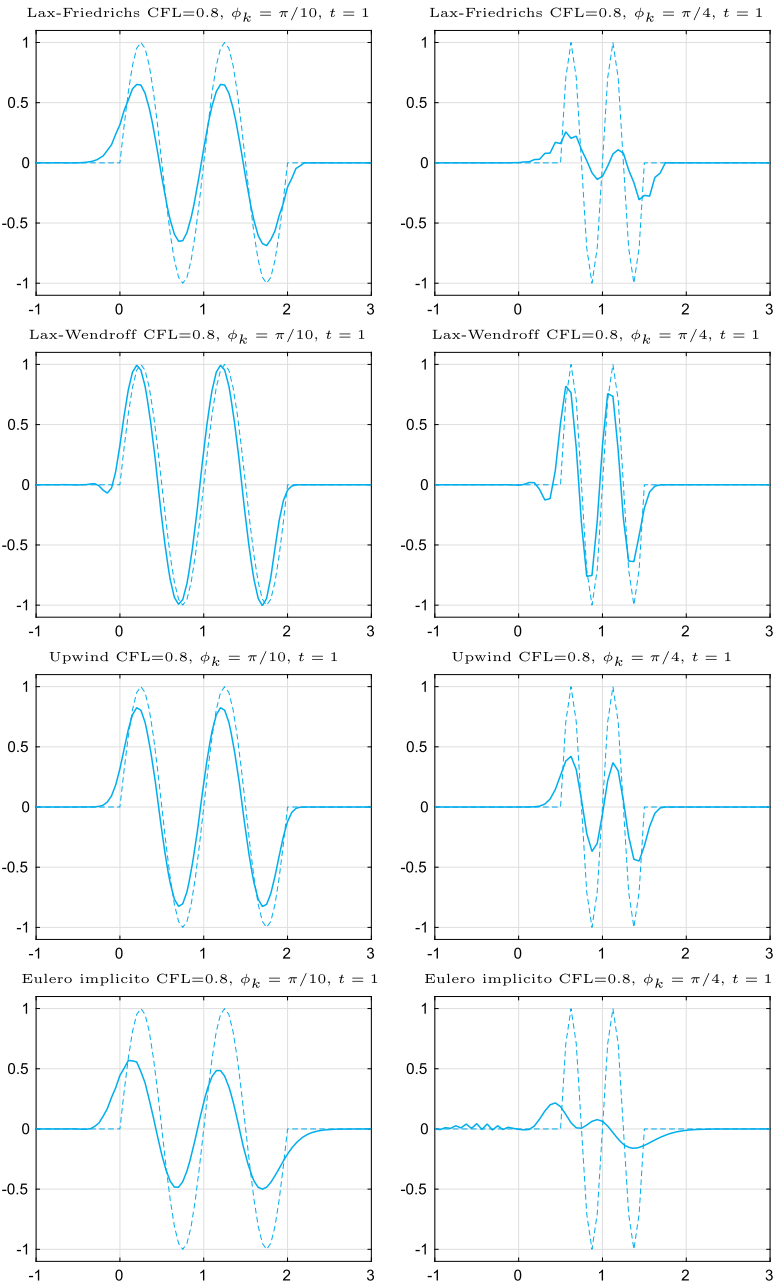

Figura 9.15. Soluzione esatta (*in linea tratteggiata*) e soluzione numerica (*in linea continua*), all'istante $t = 1$ e al variare di x, del problema (9.78) per $a = 1$ e dato iniziale definito in (9.98) con lunghezza d'onda $\ell = 1$ (*grafici a sinistra*) e $\ell = 1/2$ (*grafici a destra*)

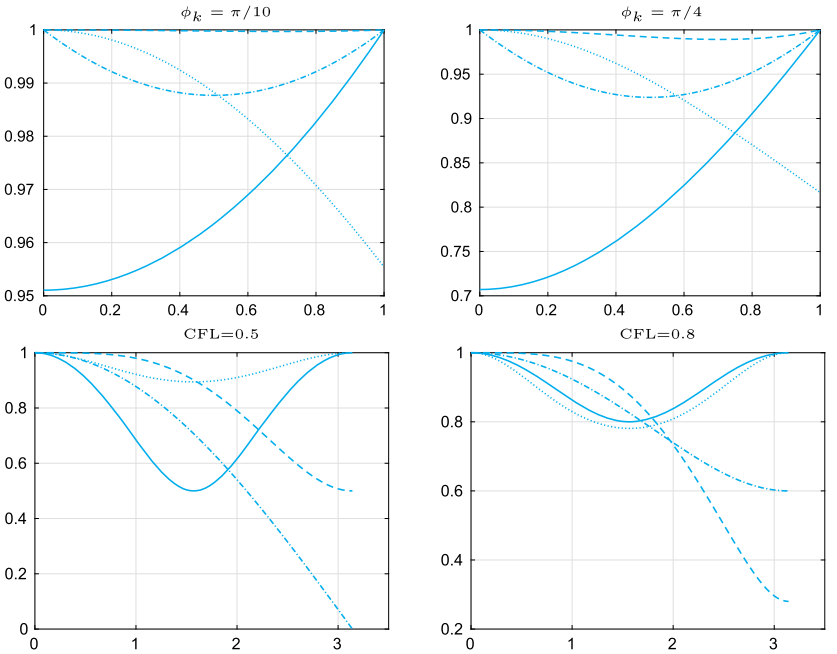

Figura 9.16. Coefficienti di dissipazione $\epsilon_a(k)$ per i metodi Lax-Friedrichs (*linea continua*), Lax-Wendroff (*linea tratteggiata*), Upwind (*linea tratto punto*) ed Eulero all'indietro/centrato (*linea punteggiata*), al variare del numero CFL (*in alto*) ed al variare di $\phi_k = kh$ (*in basso*)

schemi della sezione 9.9.1. Il dato iniziale è

$$u^0(x) = \begin{cases} \sin(2\pi x/\ell) & -1 \le x \le \ell \\ 0 & \ell < x < 3, \end{cases} \tag{9.98}$$

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

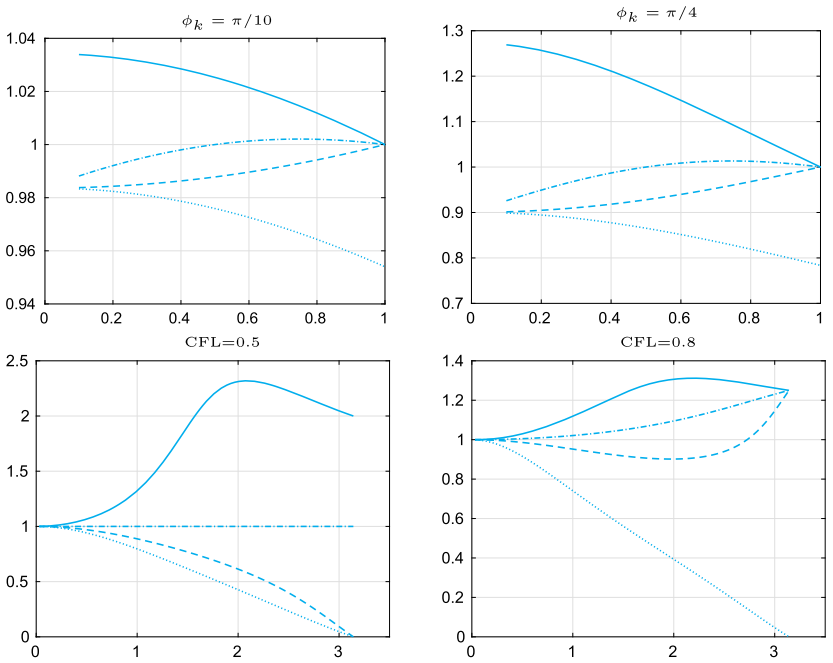

Figura 9.17. Coefficienti di dispersione $\epsilon_d(k)$ al variare del numero CFL (*in alto*) ed al variare di $\phi_k = kh$ (*in basso*). Si veda la legenda della Figura 9.16 per la descrizione dei tipi di linea

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

$$u(\alpha, t) = \varphi(t), \quad t > 0. \tag{9.99}$$

Dopo aver definito gli spazi

$$V_h = \{v_h \in C^0([\alpha,\beta]) : \ v_{h|I_j} \in \mathbb{P}_1, j = 1,\dots,N\},$$
$$V_h^{in} = \{v_h \in V_h : \ v_h(\alpha) = 0\},$$

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

$$\begin{cases} \displaystyle\int_\alpha^\beta \frac{\partial u_h(t)}{\partial t} v_h dx + \int_\alpha^\beta a \frac{\partial u_h(t)}{\partial x} v_h dx = 0 & \forall v_h \in V_h^{in}, \\ u_h(t) = \varphi(t) & \text{in } x = \alpha, \end{cases} \tag{9.100}$$

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

$$\|u_h^n\|_{\mathrm{L}^2(\alpha,\beta)} \leq \|u_h^0\|_{\mathrm{L}^2(\alpha,\beta)} \quad \forall n \geq 0,$$

ovvero lo schema di Eulero all'indietro è incondizionatamente stabile rispetto alla norma $\|v\|_{\mathrm{L}^2(\alpha,\beta)} := (\int_\alpha^\beta v^2(x)dx)^{1/2}$.

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

$$\boxed{\frac{\partial^2 u}{\partial t^2} - c\frac{\partial^2 u}{\partial x^2} = f} \tag{9.101}$$

dove c è una costante positiva assegnata. Quando $f = 0$, la soluzione generale di (9.101) è la cosiddetta onda viaggiante di d'Alembert

$$u(x,t) = \psi_1(\sqrt{c}t - x) + \psi_2(\sqrt{c}t + x), \tag{9.102}$$

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

$$u(x,0) = u_0(x) \text{ e } \frac{\partial u}{\partial t}(x,0) = v_0(x), \quad x \in (a,b), \tag{9.103}$$

ed i dati al bordo

$$u(a,t) = 0 \text{ e } u(b,t) = 0, \ t > 0. \tag{9.104}$$

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

$$\omega_1 = \frac{\partial u}{\partial x}, \qquad \omega_2 = \frac{\partial u}{\partial t},$$

trasforma (9.101) nel sistema del primo ordine

$$\boxed{\frac{\partial \boldsymbol{\omega}}{\partial t} + \mathrm{A}\frac{\partial \boldsymbol{\omega}}{\partial x} = \mathbf{f}, \quad x \in (a,b), \ t > 0} \tag{9.105}$$

dove

$$\boldsymbol{\omega} = \begin{bmatrix} \omega_1 \\ \omega_2 \end{bmatrix}, \qquad \mathrm{A} = \begin{bmatrix} 0 & -1 \\ -c & 0 \end{bmatrix}, \qquad \mathbf{f} = \begin{bmatrix} 0 \\ f \end{bmatrix},$$

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 $\boldsymbol{\omega}$, $\mathbf{f} : \mathbb{R} \times [0,\infty) \to \mathbb{R}^p$ sono funzioni vettoriali assegnate e $\mathrm{A} \in \mathbb{R}^{p\times p}$ 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 $\mathrm{T} \in \mathbb{R}^{p\times p}$ tale che

$$\mathrm{A} = \mathrm{T}\Lambda\mathrm{T}^{-1},$$

dove $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_p)$ è la matrice diagonale degli autovalori reali di A, mentre $\mathrm{T} = (\mathbf{v}^1, \mathbf{v}^2, \ldots, \mathbf{v}^p)$ è la matrice i cui vettori colonna sono gli autovettori destri di A. Così

$$\mathrm{A}\mathbf{v}^k = \lambda_k \mathbf{v}^k, \quad k = 1, \ldots, p.$$

Introducendo le *variabili caratteristiche* $\mathbf{w} = \mathrm{T}^{-1}\boldsymbol{\omega}$, il sistema (9.105) diventa

$$\frac{\partial \mathbf{w}}{\partial t} + \Lambda \frac{\partial \mathbf{w}}{\partial x} = \mathbf{g},$$

dove $\mathbf{g} = \mathrm{T}^{-1}\mathbf{f}$. Questo è un sistema di p equazioni scalari independenti della forma

$$\frac{\partial w_k}{\partial t} + \lambda_k \frac{\partial w_k}{\partial x} = g_k, \quad k = 1, \ldots, p. \tag{9.106}$$

Quando $g_k = 0$, la sua soluzione è data da $w_k(x,t) = w_k(x - \lambda_k t, 0)$, $k = 1, \ldots, p$ e la soluzione $\boldsymbol{\omega} = \mathrm{T}\mathbf{w}$ 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 $\boldsymbol{\omega}(\overline{x}, \overline{t})$ dipende solo dal dato iniziale nei punti $\overline{x} - \lambda_k \overline{t}$.

Per questa ragione l'insieme dei p punti che formano i piedi delle caratteristiche uscenti dal punto $(\overline{x}, \overline{t})$,

$$D(\overline{t}, \overline{x}) = \{x \in \mathbb{R} \ : \ x = \overline{x} - \lambda_k \overline{t} \ , \ k = 1, \ldots, p\}, \tag{9.107}$$

è detto *dominio di dipendenza* della soluzione $\boldsymbol{\omega}(\overline{x}, \overline{t})$.

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 A è diagonalizzabile mediante la matrice

$$\mathrm{T} = \begin{bmatrix} -\frac{1}{\sqrt{c}} & \frac{1}{\sqrt{c}} \\ 1 & 1 \end{bmatrix}$$

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 $x_j = x_0 + jh$, $j = 0, \ldots, N+1$, $x_0 = a$ e $x_{N+1} = b$, lo schema di Newmark per (9.101) è: per ogni $n \geq 1$ si cercano $\{u_j^n, v_j^n, j = 1, \ldots, N\}$ tali che

$$\begin{aligned} u_j^{n+1} &= u_j^n + \Delta t v_j^n \\ &\quad + \Delta t^2 \left[\zeta(cw_j^{n+1} + f(x_j, t^{n+1})) + (1/2 - \zeta)(cw_j^n + f(x_j, t^n))\right], \\ v_j^{n+1} &= v_j^n + \Delta t \left[(1-\theta)(cw_j^n + f(x_j, t^n)) + \theta(cw_j^{n+1} + f(x_j, t^{n+1}))\right], \end{aligned} \tag{9.108}$$

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)`=ζ, `param(2)`=θ). 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,V0,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)=V0(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), V0(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,V0,GD,F,P1,P2,...) passa i parametri addizio-
%   nali P1,P2,... alle funzioni U0,V0,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),

$$u_j^{n+1} - 2u_j^n + u_j^{n-1} = c\left(\frac{\Delta t}{h}\right)^2 (u_{j+1}^n - 2u_j^n + u_{j-1}^n) \tag{9.109}$$

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),

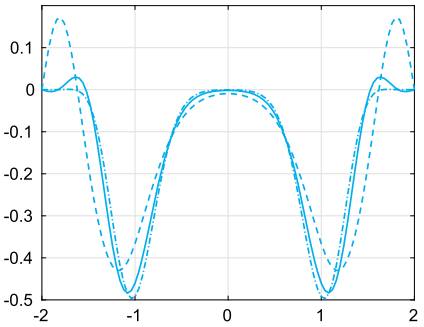

Figura 9.18. Confronto tra le soluzioni ottenute con il metodo di Newmark per una discretizzazione con $h = 0.04$ e $\Delta t = 0.15$ (*linea tratteggiata*), $\Delta t = 0.075$ (*linea continua*) e $\Delta t = 0.0375$ (*linea tratto-punto*)

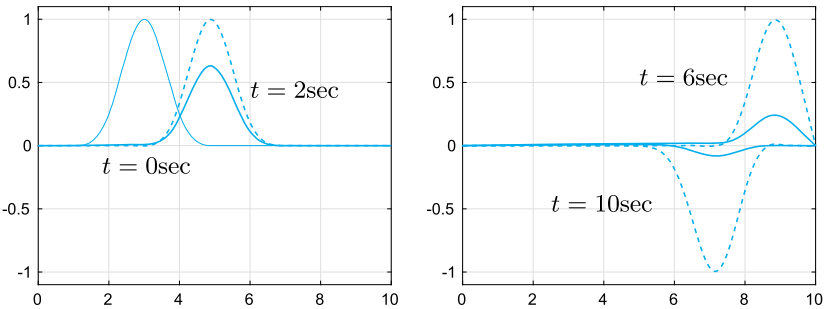

Figura 9.19. Propagazione di un impulso di potenziale ottenuto risolvendo l'equazione delle onde (*linea tratteggiata*) e l'equazione del telegrafo (*linea continua*). A sinistra, la linea continua sottile descrive la condizione iniziale $u_0(x)$

$\Delta t = 0.075$ (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*

$$\begin{aligned}\boldsymbol{\omega}_j^{n+1} = \boldsymbol{\omega}_j^n - \frac{\lambda}{2}\mathrm{A}(\boldsymbol{\omega}_{j+1}^n - \boldsymbol{\omega}_{j-1}^n) \\ + \frac{\lambda^2}{2}\mathrm{A}^2(\boldsymbol{\omega}_{j+1}^n - 2\boldsymbol{\omega}_j^n + \boldsymbol{\omega}_{j-1}^n);\end{aligned} \tag{9.110}$$

2. *metodo upwind (o Eulero in avanti/decentrato)*

$$\begin{aligned}\boldsymbol{\omega}_j^{n+1} = \boldsymbol{\omega}_j^n - \frac{\lambda}{2}\mathrm{A}(\boldsymbol{\omega}_{j+1}^n - \boldsymbol{\omega}_{j-1}^n) \\ + \frac{\lambda}{2}|\mathrm{A}|(\boldsymbol{\omega}_{j+1}^n - 2\boldsymbol{\omega}_j^n + \boldsymbol{\omega}_{j-1}^n),\end{aligned} \tag{9.111}$$

avendo posto $|\mathrm{A}| = \mathrm{T}|\Lambda|\mathrm{T}^{-1}$ ed essendo $|\Lambda|$ 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:

$$\Delta t < \frac{h}{\rho(\mathrm{A})}, \tag{9.112}$$

dove, come al solito, $\rho(\mathrm{A})$ 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 fomulazione 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 realizzi 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 implicito 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* `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 $\mathrm{div}\nabla\phi = \Delta\phi$, 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. Analogmanete, si mostri che la codizione 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 $\mathrm{A_{fd}}$ definita in (9.20) è simmetrica e definita positiva.

Esercizio 9.4 Si verifichi che la matrice $\mathrm{A_{fd}}$ definita in (9.20) ha autovalori pari a

$$\lambda_j = \frac{2}{h^2}(1 - \cos(j\theta)), \quad j = 1, \ldots, N,$$

e corrispondenti autovettori dati da

$$\mathbf{q}_j = (\sin(j\theta), \sin(2j\theta), \ldots, \sin(Nj\theta))^T,$$

dove $\theta = \pi/(N+1)$. Si concluda che $K(\mathrm{A_{fd}})$ 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

$$\delta^2 u(\bar{x}) - u''(\bar{x}) = \mathcal{O}(h^2) \quad \text{quando } h \to 0. \tag{9.113}$$

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

$$\begin{cases} -u'' + \dfrac{k}{T}u = \dfrac{w}{T} & \text{in } (0,1), \\ u(0) = u(1) = 0, \end{cases}$$

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 MATLAB 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 MATLAB 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 $j = 1, \ldots, N$.

Esercizio 9.12 Si scriva una *function* MATLAB 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 conduttività termica $k = 0.2$ cal/(sec·cm·C). Denotiamo con $Q = 5$ cal/(cm^3·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

$$\int_a^b (u_t(x,t))^2 dx + c\int_a^b (u_x(x,t))^2 dx \qquad (9.114)$$
$$= \int_a^b (v_0(x))^2 dx + c\int_a^b (u_{0,x}(x))^2 dx,$$

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 $\forall \Delta t > 0$,

$$\|\mathbf{u}^n\|_{\Delta,2} \leq \|\mathbf{u}^0\|_{\Delta,2} \quad \forall n \geq 0. \qquad (9.115)$$

Esercizio 9.16 Si dimostri che la soluzione fornita dallo schema *upwind* (9.87) soddisfa la stima

$$\|\mathbf{u}^n\|_{\Delta,\infty} \leq \|\mathbf{u}^0\|_{\Delta,\infty} \quad \forall n \geq 0 \qquad (9.116)$$

a condizione che venga verificata la condizione CFL. La relazione (9.116) è detta *principio del massimo discreto*.

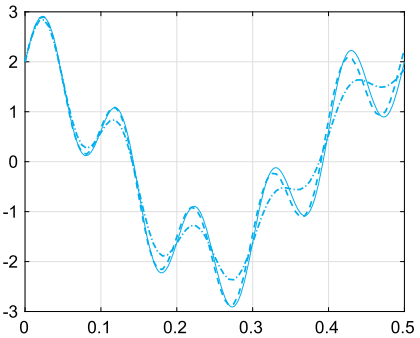

Figura 9.20. La soluzione esatta (*linea continua*) e le soluzioni numeriche del problema (9.75) con i dati dell'esercizio 9.17 al tempo $t = 5$ ottenute con lo schema di Lax-Wendroff (*linea tratteggiata*) e lo schema Upwind (*linea tratto punto*). Il numero CFL è 0.8

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 $\Delta t = 5.\text{e}{-3}$, commentare i risultati ottenuti (in termini di coefficiente di dissipazione e di dispersione).

10

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 $a_2, \ldots, a_t$ e $\beta - 1$ per la cifra a_1 (che non può assumere il valore 0). In tutto, avremo perciò $(\beta - 1)\beta^{t-1}$ numeri rappresentabili a meno del segno e per esponente fissato. D'altra parte, l'esponente può assumere $U - L + 1$ valori e quindi, complessivamente, l'insieme $\mathbb{F}(\beta, t, L, U)$ è costituito da $2(\beta - 1)\beta^{t-1}(U - L + 1)$ elementi.

Soluzione 1.3 Per la formula di Eulero si ha $i = e^{i\pi/2}$ e quindi $i^i = e^{-\pi/2}$ che è un numero reale. In MATLAB/OCTAVE

```
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)
```


A. Quarteroni et al., *Calcolo Scientifico*, 6a edizione,
UNITEXT – La Matematica per il 3+2 105, DOI 10.1007/978-88-470-3953-7_10

Soluzione 1.5 Per scambiare la terza con la settima riga della matrice triangolare inferiore costruita in precedenza, basta porre `r=[1:10]; r(3)=7; r(7)=3; Lr=L(r,:)`. Si noti l'uso dei due punti in `L(r,:)` che indica che tutte le colonne di `L` devono essere percorse nell'ordine usuale. Per scambiare l'ottava colonna con la quarta, basta invece porre: `c=[1:10]; c(8)=4; c(4)=8; Lc=L(:,c)`. Un analogo discorso vale per la matrice triangolare superiore.

L(r,:)

Soluzione 1.6 Si costruisce la matrice `A = [v1;v2;v3;v4]` dove `v1`, `v2`, `v3` e `v4` sono i vettori riga MATLAB corrispondenti ai 4 vettori dati. Siccome `det(A)=0` i 4 vettori sono linearmente dipendenti.

Soluzione 1.7 Utilizziamo i seguenti comandi per introdurre l'espressione simbolica delle funzioni f e g:

```
syms x
f=sqrt(x^2+1); pretty(f)
```

$$(x^2+1)^{1/2}$$

```
g=sin(x^3)+cosh(x); pretty(g)
```

$$\sin(x^3)+\cosh(x)$$

pretty

Si noti il comando `pretty` con il quale è possibile avere una versione più leggibile delle funzioni introdotte. A questo punto per calcolare le derivate prima e seconda, nonché l'integrale indefinito di f basta scrivere:

```
diff(f,x)
ans =
1/(x^2+1)^(1/2)*x
diff(f,x,2)
ans =
-1/(x^2+1)^(3/2)*x^2+1/(x^2+1)^(1/2)
int(f,x)
ans =
1/2*x*(x^2+1)^(1/2)+1/2*asinh(x)
```

Analoghe istruzioni valgono per la funzione g.

Soluzione 1.8 L'accuratezza delle radici calcolate degrada rapidamente al crescere del grado del polinomio. Questo risultato ci deve mettere in guardia sull'accuratezza del calcolo delle radici di un polinomio di grado elevato.

Soluzione 1.9 Una possibile implementazione è la seguente:

```
function 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) <= 1` per ogni componente del vettore `z`. Se la disuguaglianza 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. sum

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 MATGOCT) 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`): 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
prod *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 nu-
semilogx meri varia notevolmente. Il comando `semilogx(x,y)` crea un grafico con scala logaritmica in x e lineare in y.

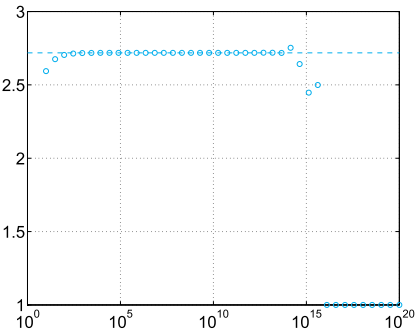

Figura 10.1. Alcuni valori della successione $a_n = (1 + 1/n)^n$ calcolati in MATLAB per il problema dell'Esercizio 1.15

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 $fl_t(1 + 1/n)$ risulta pari a 1 e, di conseguenza, anche $(fl_t(1 + 1/n))^n$ rimarrà uguale a 1.

Poiché si ha

```
1/(eps/2)
ans =
     9.007199254740992e+15
```

avremo $fl_t(1 + 1/n) = 1$ 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 disuguaglianza (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 `tol=1.e-10`, il metodo di Newton (richiamato con il Programma 2.2) converge al valore $\widehat{\alpha} = 1.4961 \cdot 10^{-4}$ in 31 iterazioni in MATLAB e ad $\widehat{\alpha} = 2.4295 \cdot 10^{-4}$ 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,

$$x^{(k+1)} = \frac{1}{2}\left(x^{(k)} + \frac{a}{x^{(k)}}\right), \quad k \geq 0 \qquad \text{per la radice quadrata,}$$
$$x^{(k+1)} = \frac{1}{3}\left(2x^{(k)} + \frac{a}{(x^{(k)})^2}\right), \quad k \geq 0 \quad \text{per la radice cubica.}$$

Soluzione 2.8 Poniamo $e^{(k)} = x^{(k)} - \alpha$. Allora, sviluppando f in serie di Taylor in un intorno del punto $x^{(0)}$ si trova:

$$0 = f(\alpha) = f(x^{(k)}) - e^{(k)} f'(x^{(k)}) + \frac{1}{2}(e^{(k)})^2 f''(x^{(k)}) + \mathcal{O}((e^{(k)})^3). \tag{10.1}$$

Per il metodo di Newton si ha

$$e^{(k+1)} = e^{(k)} - f(x^{(k)})/f'(x^{(k)}) \tag{10.2}$$

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 $k \to \infty$, 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 $k = 0, \ldots, 100$ (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);
```

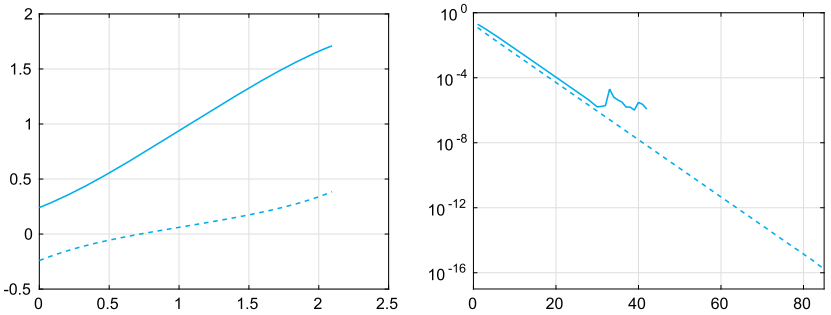

Figura 10.2. A sinistra: le due curve rappresentano le due possibili configurazioni del sistema di aste (precisamente l'angolo α) al variare di $\beta \in [0, 2\pi/3]$ (*Soluzione 2.9*). A destra: andamento dell'errore nel metodo di Newton per il calcolo dello zero di $f(x) = x^3 - 3x^2 2^{-x} + 3x4^{-x} - 8^{-x}$ (*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`$= \epsilon_M$. 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) -l2*cos(g+a)/sin(g+a)^2-l1*cos(a)/sin(a)^2;
df=@(a) [l2/sin(g+a)+2*l2*cos(g+a)^2/sin(g+a)^3+...
         l1/sin(a)+2*l1*cos(a)^2/sin(a)^3];
[zero,res,niter]=newton(f,df,pi/4,1.e-15,100)
L=l2/sin(2*pi/5-zero)+l1/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 `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 si 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).

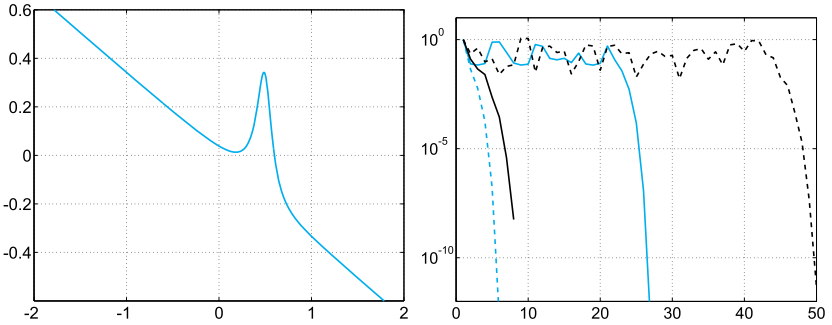

Figura 10.3. La funzione della Soluzione 2.15 (*a sinistra*). Le storie di convergenza dei metodi utilizzati per lo svolgimento dell'esercizio 2.15 (*a destra*): Newton con $x^{(0)} = 0.4$ (*azzurro in linea tratteggiata*), Newton con $x^{(0)} = 0.55$ (*azzurro in linea continua*), secanti con $x^{(0)} = 0.45$ 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 MAT&OCT 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 MAT&OCT 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');
```

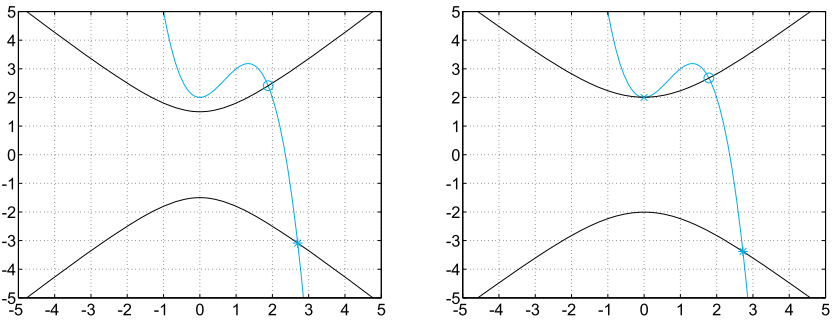

Figura 10.4. La rappresentazione grafica del sistema della Soluzione 2.16, $\gamma = 2$ (*a sinistra*) e $\gamma = 3.75$ (*a destra*)

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 $\mathbf{J_f}(\mathbf{x})$ 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 $\lim_{x\to\alpha} \phi_N'(x) = 1 - 1/m$, 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:

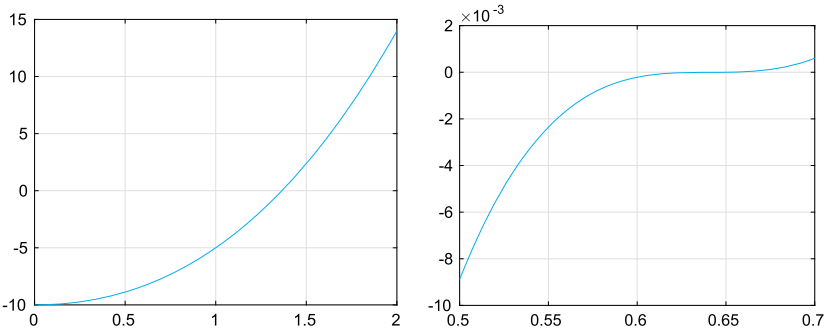

Figura 10.5. A sinistra il grafico di $f(x) = x^3 + 4x^2 - 10$ per $x \in [0, 2]$ (*Soluzione 2.18*). A destra il grafico di $f(x) = x^3 - 3x^2 2^{-x} + 3x4^{-x} - 8^{-x}$ per $x \in [0.5, 0.7]$ (*Soluzione 2.20*)

$$\begin{aligned}\phi(x) &= \frac{2x^3+4x^2+10}{3x^2+8x} = -\frac{f(x)}{3x^2+8x}+x,\\ \phi'(x) &= \frac{(6x^2+8x)(3x^2+8x)-(6x+8)(2x^3+4x^2+10)}{(3x^2+8x)^2}\\ &= \frac{(6x+8)f(x)}{(3x^2+8x)^2},\end{aligned}$$

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
```

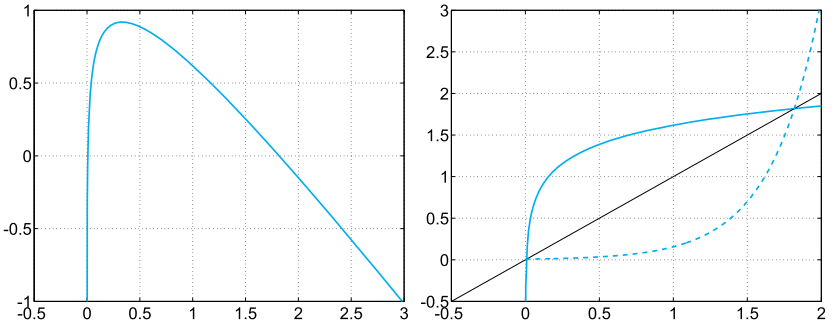

Figura 10.6. La funzione della Soluzione 2.21 (*a sinistra*). Le funzioni di punto fisso $\phi_1(x)$ (*linea continua*) e $\phi_2(x)$ (*linea tratteggiata*) (*a destra*)

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 - 0.000000000000000i
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 $\phi_1(x) = \log(x/\pi)/3 + 2$. Rappresentandola 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 MATLAB/OCTAVE:

```
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 MATLAB/OCTAVE:

```
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 MATGOCT 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 indicheremo 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 = cubicspline(anno,produzione,z);
s3d = cubicspline(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);
```

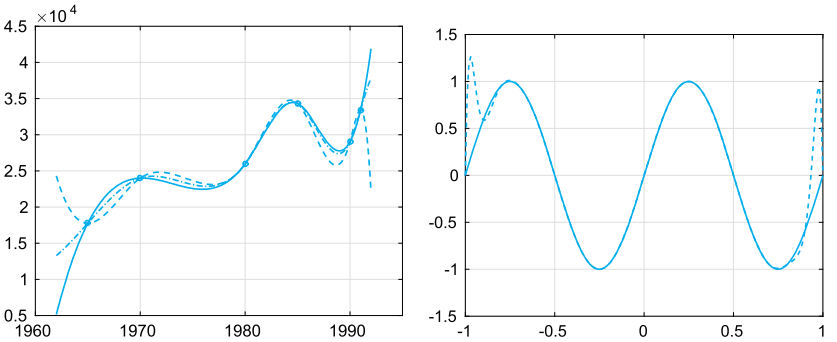

Figura 10.7. A sinistra: confronto fra i grafici delle *spline* cubiche generate nel corso della Soluzione 3.7: `s3` (*linea continua*), `s3d` (*linea tratteggiata*), `s3n` (*linea tratto-punto*). I cerchietti rappresentano i valori interpolati. A destra: il polinomio interpolatore (*linea tratteggiata*) 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 globale 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 $n \to \infty$.

Grazie alla Proposizione 3.3 possiamo concludere che anche l'interpolatore lineare composito $\Pi_1^H f$ convergerà ad f per $H \to 0$ 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 $H \to 0$ 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];
```

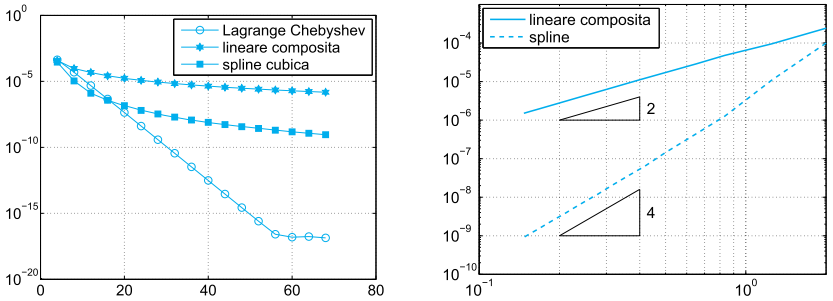

Figura 10.8. Gli errori di interpolazione per la Soluzione 3.9: a sinistra al variare di n, a destra al variare di H

```
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 MATLAB, 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

$$H \leq \left(\frac{8}{\varepsilon \max_{x\in[x_0,x_n]}|f''(x)|}\right)^{1/2}. \tag{10.3}$$

Rappresentando graficamente la funzione $f''(x) = 2e^{-x^2}(2x^2-1)$, deduciamo che $\max_{[-2,2]}|f''(x)| = 2$, quindi dalla disuguaglianza (10.3) ricaviamo

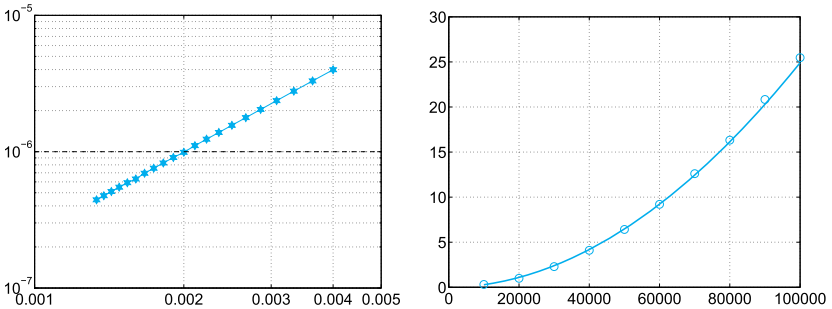

Figura 10.9. Gli errori di interpolazione per la Soluzione 3.10 (*a sinistra*). Approssimazione nel senso dei minimi quadrati per i dati della Soluzione 3.17 (*a destra*)

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];
```

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 `9.900e-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`):

$K = 0.67$, $a_4 = 7.211 \cdot 10^{-8}$, $a_3 = -6.088 \cdot 10^{-7}$, $a_2 = -2.988 \cdot 10^{-4}$, $a_1 = 1.650 \cdot 10^{-3}$, $a_0 = -3.030$;
$K = 1.5$, $a_4 = -6.492 \cdot 10^{-8}$, $a_3 = -7.559 \cdot 10^{-7}$, $a_2 = 3.788 \cdot 10^{-4}$, $a_1 = 1.67310^{-3}$, $a_0 = 3.149$;
$K = 2$, $a_4 = -1.050 \cdot 10^{-7}$, $a_3 = 7.130 \cdot 10^{-8}$, $a_2 = 7.044 \cdot 10^{-4}$, $a_1 = -3.828 \cdot 10^{-4}$, $a_0 = 4.926$;
$K = 3$, $a_4 = -2.319 \cdot 10^{-7}$, $a_3 = 7.740 \cdot 10^{-7}$, $a_2 = 1.419 \cdot 10^{-3}$, $a_1 = -2.574 \cdot 10^{-3}$, $a_0 = 7.315$.

A sinistra di Figura 10.10 riportiamo il polinomio ottenuto per i dati della colonna relativa a $K = 0.67$ nella Tabella 3.1.

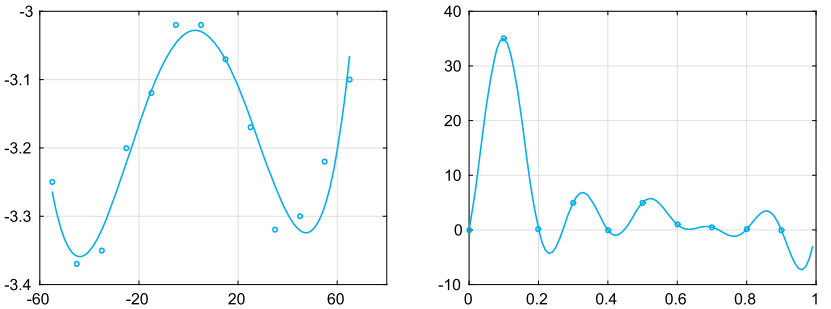

Figura 10.10. A sinistra: polinomio di grado 4 dei minimi quadrati (*linea continua*) a confronto con i dati della colonna di Tabella 3.1 per $K = 0.67$ (Soluzione 3.12). A destra: l'interpolatore trigonometrico ottenuto con le istruzioni della Soluzione 3.16. I pallini si riferiscono ai dati sperimentali 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 $i = 1, \ldots, N$. Le istruzioni MATLAB 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

$$f(x_1) = f(x_0) + hf'(x_0) + \frac{h^2}{2}f''(x_0) + \frac{h^3}{6}f'''(\xi_1),$$

$$f(x_2) = f(x_0) + 2hf'(x_0) + 2h^2 f''(x_0) + \frac{4h^3}{3}f'''(\xi_2),$$

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:

$$\frac{1}{2h}\left[-3f(x_0) + 4f(x_1) - f(x_2)\right] = f'(x_0) + \frac{h^2}{3}[f'''(\xi_1) - 2f'''(\xi_2)],$$

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 $f'(\overline{x})$.

Soluzione 4.3 Eseguendo operazioni analoghe a quelle indicate nella Soluzione 4.2, si trovano i seguenti errori (supponendo $f \in C^4$):

$$a.\quad -\frac{1}{4}f^{(4)}(\xi)h^3, \qquad b.\quad -\frac{1}{12}f^{(4)}(\xi)h^3, \qquad c.\quad \frac{1}{6}f^{(4)}(\xi)h^3.$$

Soluzione 4.4 Usiamo l'approssimazione (4.11). Si trovano i seguenti valori

t (Mesi)	0	0.5	1	1.5	2	2.5	3
δn	–	78	45	19	7	3	–
n'	–	77.91	39.16	15.36	5.91	1.99	–

che, come risulta dal confronto con i valori esatti di $n'(t)$ calcolati negli stessi istanti, sono abbastanza accurati.

Soluzione 4.5 L'errore di quadratura commesso con la formula composita del punto medio può essere maggiorato con

$$\frac{(b-a)^3}{24M^2}\max_{x\in[a,b]}|f''(x)|,$$

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, $k = 1,\ldots,M$, 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:

$$I(x) = \int_a^b x dx = \frac{b^2-a^2}{2}, \qquad I_{PM}(x) = (b-a)\frac{b+a}{2}.$$

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

$$I(f) = I_1 + CH_1^2, \qquad I(f) = I_2 + CH_2^2, \tag{10.4}$$

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 MATLAB/OCTAVE:

```
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 MATLAB/OCTAVE 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.

double

Soluzione 4.14 Si noti che, essendo la funzione integranda non negativa, allora $I_k > 0\ \forall k$. La formula ricorsiva proposta risulta però instabile a causa degli errori di arrotondamento come si vede dando i seguenti comandi MATLAB/OCTAVE:

```
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:

$$I_1 = 1.19616568040561, \qquad I_2 = 1.19628173356793,$$
$$I_R = 1.19628947044542,$$

con un errore $I(f) - I_R = -1.4571\text{e}{-06}$ 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):

$$I_1 = 1.19637085545393 \quad (-8.2842\text{e}{-05}),$$
$$I_2 = 1.19629221796844 \quad (-4.2046\text{e}{-06}),$$
$$I_R = 1.19628697546941 \quad (1.0379\text{e}{-06}).$$

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 $k = 1,\ldots,10$ 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 disuguaglianza è allora dato, al variare di $r = k/10$ con $k = 1,\ldots,10$, 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 $\lim_{x\to 0^+} \frac{\sin x}{x} = 1$, possiamo estendere la funzione $\lambda(x)$ ad una funzione $\widetilde{\lambda}(x)$ tale che $\widetilde{\lambda}(0) = 10^{-4}$ e $\widetilde{\lambda}(x) = \lambda(x)$ 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

$$\left| \int_0^2 \lambda(x)dx - I^c_{pm}(\lambda) \right| \leq \frac{L}{24} H^2 \max_{0\leq x\leq L} |\lambda''(x)| < 10^{-8}.$$

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 MATLAB/OCTAVE:

```
a=0; b=1; A=1.e-4;
lambda=@(x)A*sin(x)./x;
maxl2=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 `massa=9.461702244627381e-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 $\overline{x} = 4.858509412435690\text{e}{-01}$ 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

$$\left| \int_0^{2\pi} f(t)dt - I^c_T(f) \right| \leq \frac{2\pi}{12} H^2 \max_{0\leq t\leq 2\pi} |f''(t)| < 10^4.$$

La funzione integranda è $f(t) = \sqrt{a^2\cos^2(t) + b^2\sin^2(t)}$, calcoliamo la derivata seconda in MATLAB/OCTAVE 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

$$H \leq \sqrt{12 \cdot 10^4/(2\pi \cdot \texttt{f2max})} \simeq 0.67618,$$

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 $k = 0, \ldots, 4$, sappiamo che gli integrali esatti sono $\int_a^b x^k dx = (b^{k+1} - a^{k+1})/(k+1)$. Dobbiamo verificare che $\widetilde{I}(x^k) = \int_a^b x^k dx$ per $k = 0, 1, 2, 3$ e che $\widetilde{I}(x^4) \neq \int_a^b x^4 dx$. 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 $\widetilde{I}(f)$:

$$\widetilde{I}(x) = \frac{b-a}{8}\left(a + 3\frac{2a+b}{3} + 3\frac{a+2b}{3} + b\right) = \frac{b^2-a^2}{2}$$

$$\widetilde{I}(x^2) = \frac{b-a}{8}\left(a^2 + 3\frac{(2a+b)^2}{9} + 3\frac{(a+2b)^2}{9} + b^2\right) = \frac{b^3-a^3}{3}$$

$$\widetilde{I}(x^3) = \frac{b-a}{8}\left(a^3 + 3\frac{(2a+b)^3}{27} + 3\frac{(a+2b)^3}{27} + b^3\right) = \frac{b^4-a^4}{4}$$

$$\widetilde{I}(x^4) = \frac{b-a}{8}\left(a^4 + 3\frac{(2a+b)^4}{81} + 3\frac{(a+2b)^4}{81} + b^4\right) = \frac{b^5-a^5}{5} + \frac{(b-a)^5}{270}$$

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

$$\widetilde{I}^c(f) = \sum_{k=1}^{M} \frac{x_{k+1} - x_k}{8} \left[f(x_k) + 3f(\overline{x}_k) + 3f(\widehat{x}_k) + f(x_{k+1}) \right]$$

con $\overline{x}_k = (2x_k + x_{k+1})/3$ e $\widehat{x}_k = (x_k + 2x_{k+1})/3$. La function MATLAB 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 MATLAB:

```
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 MATLAB è:

```
P=log(e(2:end)./e(1:end-1))./log(H(2:end)./H(1:end-1))
```

Otteniamo:

```
P =
    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 MATGOCT 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: $\det \mathrm{A}_1 = 1$, $\det \mathrm{A}_2 = \varepsilon$, $\det \mathrm{A}_3 = \det \mathrm{A} = 2\varepsilon + 12$. 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

$$\mathrm{L} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 1.25 & 1 \end{bmatrix}, \qquad \mathrm{U} = \begin{bmatrix} 1 & 7 & 3 \\ 0 & -12 & -4 \\ 0 & 0 & 0 \end{bmatrix}.$$

Tuttavia, poiché U è singolare (com'era del resto da attendersi essendo A singolare), il sistema triangolare superiore $\mathrm{U}\mathbf{x} = \mathbf{y}$ 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 $i = 2, \ldots, n$. Quindi servono $(n-1)^2$ moltiplicazioni e $(n-1)^2$ addizioni per costruire i valori $a_{ij}^{(2)}$, per $i, j = 2, \ldots, n$. 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

$$\sum_{s=1}^{q} s = \frac{q(q+1)}{2}, \qquad \sum_{s=1}^{q} s^2 = \frac{q(q+1)(2q+1)}{6}, \quad q \geq 1,$$

possiamo concludere che per realizzare la fattorizzazione di Gauss serve il seguente numero di operazioni

$$\sum_{k=1}^{n-1} \sum_{i=k+1}^{n} \left(1 + \sum_{j=k+1}^{n} 2 \right) = \sum_{k=1}^{n-1} (n-k)(1+2(n-k))$$
$$= \sum_{j=1}^{n-1} j + 2\sum_{j=1}^{n-1} j^2 = \frac{(n-1)n}{2} + 2\frac{(n-1)n(2n-1)}{6} = \frac{2}{3}n^3 - \frac{n^2}{2} - \frac{n}{6}.$$

Soluzione 5.5 Per definizione, l'inversa X di una matrice $\mathrm{A} \in \mathbb{R}^{n\times n}$ è tale che $\mathrm{XA} = \mathrm{AX} = \mathrm{I}$. Di conseguenza, per ogni $j = 1, \ldots, n$ il vettore colonna $\mathbf{x}_j$ di X risolve il sistema lineare $\mathrm{A}\mathbf{x}_j = \mathbf{e}_j$ 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:

$$\mathrm{L} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & -3.38 \cdot 10^{15} & 1 \end{bmatrix}, \qquad \mathrm{U} = \begin{bmatrix} 1 & 1 & 3 \\ 0 & -8.88 \cdot 10^{-16} & 14 \\ 0 & 0 & 4.73 \cdot 10^{16} \end{bmatrix},$$

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

$$\mathrm{L} = \begin{bmatrix} 1 & 0 & 0 \\ (\varepsilon-2)/2 & 1 & 0 \\ 0 & -1/\varepsilon & 1 \end{bmatrix}, \qquad \mathrm{U} = \begin{bmatrix} 2 & -2 & 0 \\ 0 & \varepsilon & 0 \\ 0 & 0 & 3 \end{bmatrix},$$

cosicché $l_{32} \to \infty$, quando $\varepsilon \to 0$. Se noi scegliamo $\mathbf{b} = (0, \varepsilon, 2)^T$, è facile verificare che $\mathbf{x} = (1,1,1)^T$ è la soluzione esatta di $\mathrm{A}\mathbf{x} = \mathbf{b}$. Per analizzare l'errore rispetto alla soluzione esatta per $\varepsilon \to 0$, prendiamo $\varepsilon = 10^{-k}$, per $k = 0, \ldots, 9$. 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 $k = 0, \ldots, 9$, 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 $K(\mathrm{A}) = C/\varepsilon$) 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 $\mathrm{A}\mathbf{v} = \lambda\mathbf{v}$ segue $\mathrm{A}^2\mathbf{v} = \lambda\mathrm{A}\mathbf{v} = \lambda^2\mathbf{v}$. Di conseguenza, $K(\mathrm{A}^2) = (K(\mathrm{A}))^2$.

Soluzione 5.11 La matrice di iterazione del metodo di Jacobi è:

$$\mathrm{B}_J = \begin{bmatrix} 0 & 0 & -\alpha^{-1} \\ 0 & 0 & 0 \\ -\alpha^{-1} & 0 & 0 \end{bmatrix}$$

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

$$\mathrm{B}_{GS} = \begin{bmatrix} 0 & 0 & -\alpha^{-1} \\ 0 & 0 & 0 \\ 0 & 0 & \alpha^{-2} \end{bmatrix}$$

con autovalori dati da $\{0, 0, \alpha^{-2}\}$. Il metodo è quindi convergente se $|\alpha| > 1$. Si noti che, avendosi $\rho(\mathrm{B}_{GS}) = [\rho(\mathrm{B}_J)]^2$, 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 limitazione (necessaria e sufficiente) $|\beta| < 25$ per entrambi gli schemi.

Soluzione 5.13 Il metodo del rilassamento può essere scritto nella seguente forma vettoriale

$$(\mathrm{I} - \omega\mathrm{D}^{-1}\mathrm{E})\mathbf{x}^{(k+1)} = [(1-\omega)\mathrm{I} + \omega\mathrm{D}^{-1}\mathrm{F}]\mathbf{x}^{(k)} + \omega\mathrm{D}^{-1}\mathbf{b}$$

dove $\mathrm{A} = \mathrm{D} - (\mathrm{E} + \mathrm{F})$, essendo D la diagonale di A, $-\mathrm{E}$ e $-\mathrm{F}$ la parte triangolare inferiore e superiore di A, rispettivamente. Si ricava allora che la matrice di iterazione è:

$$\mathrm{B}(\omega) = (\mathrm{I} - \omega\mathrm{D}^{-1}\mathrm{E})^{-1}[(1-\omega)\mathrm{I} + \omega\mathrm{D}^{-1}\mathrm{F}].$$

Possiamo a questo punto osservare che, se denotiamo con λ_i gli autovalori di $\mathrm{B}(\omega)$, abbiamo

$$\begin{aligned}\left|\prod_{i=1}^{n}\lambda_i\right| &= |\mathrm{detB}(\omega)| \\ &= |\det[(I-\omega D^{-1}E)^{-1}]|\cdot|\det[(1-\omega)I+\omega D^{-1}F)]|.\end{aligned}$$

Osservando ora che, date due matrici A e B con $\mathrm{A}=\mathrm{I}+\alpha\mathrm{B}$, per ogni $\alpha\in\mathbb{R}$ vale $\lambda_i(\mathrm{A})=1+\alpha\lambda_i(\mathrm{B})$, e che gli autovalori di $\mathrm{D}^{-1}E$ e di $\mathrm{D}^{-1}F$ sono tutti nulli, abbiamo

$$\left|\prod_{i=1}^{n}\lambda_i\right| = \left|\prod_{i=1}^{n}\frac{(1-\omega)+\omega\lambda_i(\mathrm{D}^{-1}F)}{1-\omega\lambda_i(\mathrm{D}^{-1}E)}\right| = |1-\omega|^n.$$

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 $\mathrm{A}=\left[\begin{smallmatrix}3 & 2\\ 2 & 6\end{smallmatrix}\right]$ è a dominanza diagonale stretta per righe, una condizione sufficiente affinché il metodo di Gauss Seidel converga. La matrice $\mathrm{A}=\left[\begin{smallmatrix}1 & \frac{1}{2}\\ 1 & 2\end{smallmatrix}\right]$ non è a dominanza stretta per righe, tuttavia è simmetrica e quindi verifichiamo se essa è anche definita positiva, ovvero che $\mathbf{z}^T\mathrm{A}\mathbf{z}>0$ per ogni $\mathbf{z}\neq\mathbf{0}$ di $\mathbb{R}^2$. Eseguiamo i calcoli con MATLAB 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 `(z1+z2)^2+z2^2`. 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

$$\mathbf{r}^{(0)}=\mathbf{b}-\mathrm{A}\mathbf{x}^{(0)}=\begin{bmatrix}1\\0\end{bmatrix}-\begin{bmatrix}2 & 1\\ 1 & 3\end{bmatrix}\mathbf{x}^{(0)}=\begin{bmatrix}-3/2\\ -5/2\end{bmatrix}.$$

A questo punto, avendosi

$$\mathrm{P}^{-1} = \begin{bmatrix} 1/2 & 0 \\ 0 & 1/3 \end{bmatrix},$$

si può calcolare $\mathbf{z}^{(0)} = \mathrm{P}^{-1}\mathbf{r}^{(0)} = (-3/4, -5/6)^T$. Di conseguenza,

$$\alpha_0 = \frac{(\mathbf{z}^{(0)})^T\mathbf{r}^{(0)}}{(\mathbf{z}^{(0)})^T\mathrm{A}\mathbf{z}^{(0)}} = \frac{77}{107},$$

e

$$\mathbf{x}^{(1)} = \mathbf{x}^{(0)} + \alpha_0\mathbf{z}^{(0)} = (197/428, -32/321)^T.$$

Soluzione 5.16 Gli autovalori della matrice $\mathrm{B}_\alpha = \mathrm{I} - \alpha\mathrm{P}^{-1}\mathrm{A}$ sono $\lambda_i(\alpha) = 1 - \alpha\mu_i$, essendo μ_i l'i-esimo autovalore di $\mathrm{P}^{-1}\mathrm{A}$. Allora

$$\rho(\mathrm{B}_\alpha) = \max_{i=1,\ldots,n} |1 - \alpha\mu_i| = \max(|1 - \alpha\mu_{\min}|, |1 - \alpha\mu_{\max}|).$$

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 $\rho(\mathrm{B}_{\alpha_{opt}})$.

Soluzione 5.17 Per ogni matrice P simmetrica definita positiva, la matrice simmetrica e definita positiva $\mathrm{P}^{1/2}$ è l'unica soluzione dell'equazione matriciale $\mathrm{X}^2 = \mathrm{P}$ ed è la cosiddetta radice quadrata di P (si veda, ad esempio [QV94, Sect. 2.5]). Indicato con $\mathrm{P}^{-1/2}$ l'inversa di $\mathrm{P}^{1/2}$, si osserva che $\mathrm{P}^{-1}\mathrm{A} = \mathrm{P}^{-1/2}(\mathrm{P}^{-1/2}\mathrm{A}\mathrm{P}^{-1/2})\mathrm{P}^{1/2}$, quindi $\mathrm{P}^{-1}\mathrm{A}$ è simile alla matrice $\widetilde{\mathrm{A}} = \mathrm{P}^{-1/2}\mathrm{A}\mathrm{P}^{-1/2}$. Dobbiamo allora dimostrare che $\widetilde{\mathrm{A}}$ è simmetrica e definita positiva.

La simmetria di $\widetilde{\mathrm{A}}$ segue immediatamente dalla simmetria di A e di $\mathrm{P}^{1/2}$.

Per dimostrare che $\widetilde{\mathrm{A}}$ è definita positiva denotiamo con μ un generico autovalore di $\widetilde{\mathrm{A}}$ e con $\mathbf{y}(\neq \mathbf{0})$ un autovettore associato a μ, per cui abbiamo $\widetilde{\mathrm{A}}\mathbf{y} = \mu\mathbf{y}$. Premoltiplicando per $\mathrm{P}^{1/2}$ entrambe i membri di $\widetilde{\mathrm{A}}\mathbf{y} = \mu\mathbf{y}$ e introducendo il vettore $\tilde{\mathbf{y}} = \mathrm{P}^{-1/2}\mathbf{y}$, abbiamo $\mathrm{A}\tilde{\mathbf{y}} = \mu\mathrm{P}\tilde{\mathbf{y}}$, da cui segue $\tilde{\mathbf{y}}^T A\tilde{\mathbf{y}} = \mu\tilde{\mathbf{y}}^T\mathrm{P}\tilde{\mathbf{y}}$. Poiché sia A che P sono simmetriche definite positive e $\tilde{\mathbf{y}} \neq \mathbf{0}$ otteniamo $\mu = (\tilde{\mathbf{y}}^T A\tilde{\mathbf{y}})/(\tilde{\mathbf{y}}^T\mathrm{P}\tilde{\mathbf{y}}) > 0$. Dall'arbitrarietà di μ segue che tutti gli autovalori di $\widetilde{\mathrm{A}}$ sono strettamente positivi e quindi $\widetilde{\mathrm{A}}$ 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

$$\begin{aligned}\tilde{\varphi}(\alpha) &= \frac{1}{2}(\mathbf{x}^{(k)} + \alpha\mathbf{d}^{(k)})^T\mathrm{A}(\mathbf{x}^{(k)} + \alpha\mathbf{d}^{(k)}) - (\mathbf{x}^{(k)} + \alpha\mathbf{d}^{(k)})^T\mathbf{b} \\ &= \frac{1}{2}(\mathbf{x}^{(k)})^T(\mathrm{A}\mathbf{x}^{(k)} - \mathbf{b}) + \alpha(\mathbf{d}^{(k)})^T(\mathrm{A}\mathbf{x}^{(k)} - \mathbf{b}) + \frac{\alpha^2}{2}(\mathbf{d}^{(k)})^T\mathrm{A}\mathbf{d}^{(k)}.\end{aligned}$$

Il minimo di $\tilde{\varphi}(\alpha)$ si trova in corrispondenza di α_k tale che $\tilde{\varphi}'(\alpha_k) = 0$, ovvero

$$-(\mathbf{d}^{(k)})^T\mathbf{r}^{(k)} + \alpha_k(\mathbf{d}^{(k)})^T\mathrm{A}\mathbf{d}^{(k)} = 0,$$

dunque (5.70) è soddisfatta.

Minimizziamo ora la funzione $\varphi(\alpha) = \|\mathbf{e}^{(k+1)}\|_{\mathrm{A}}^2$ 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

$$\varphi(\alpha) = \|\mathbf{e}^{(k+1)}\|_{\mathrm{A}}^2 = \|\mathbf{e}^{(k)}\|_{\mathrm{A}}^2 + \alpha^2\|\mathbf{d}^{(k)}\|_{\mathrm{A}}^2 - 2\alpha(\mathrm{A}\mathbf{e}^{(k)}, \mathbf{d}^{(k)}).$$

Il minimo di $\varphi(\alpha)$ si trova in corrispondenza di α_k tale che $\varphi'(\alpha_k) = 0$, ovvero

$$\alpha_k\|\mathbf{d}^{(k)}\|_{\mathrm{A}}^2 - (\mathrm{A}\mathbf{e}^{(k)}, \mathbf{d}^{(k)}) = 0,$$

dunque $\alpha_k = (\mathrm{A}\mathbf{e}^{(k)}, \mathbf{d}^{(k)})/\|\mathbf{d}^{(k)}\|_{\mathrm{A}}^2$. Infine la formula (5.70) segue osservando che $\mathrm{A}\mathbf{e}^{(k)} = \mathbf{r}^{(k)}$.

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

$$\mathbf{r}^{(k+1)} = \mathbf{b} - \mathrm{A}\mathbf{x}^{(k+1)} = \mathbf{b} - \mathrm{A}(\mathbf{x}^{(k)} + \alpha_k\mathbf{d}^{(k)}) = \mathbf{r}^{(k)} - \frac{(\mathbf{d}^{(k)})^T\mathbf{r}^{(k)}}{(\mathbf{d}^{(k)})^T\mathrm{A}\mathbf{d}^{(k)}}\mathrm{A}\mathbf{d}^{(k)},$$

quindi

$$(\mathbf{d}^{(k)})^T\mathbf{r}^{(k+1)} = (\mathbf{d}^{(k)})^T\mathbf{r}^{(k)} - \frac{(\mathbf{d}^{(k)})^T\mathbf{r}^{(k)}}{(\mathbf{d}^{(k)})^T\mathrm{A}\mathbf{d}^{(k)}}(\mathbf{d}^{(k)})^T\mathrm{A}\mathbf{d}^{(k)} = 0.$$

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 $\mathrm{A}^T\mathrm{A}\mathbf{x} = \mathrm{A}^T\mathbf{b}$ che ha matrice simmetrica e definita positiva. Risolviamo tale sistema richiedendo una tolleranza sul residuo relativo pari a 10^{-10} e partendo dal dato iniziale $\mathbf{x}^{(0)} = \mathbf{0}^T$:

```
b = [1:20]';  AA=A'*A; b=A'*b; x0 = zeros(20,1);
[x,iter]=itermeth(AA,b,x0,100,1.e-10);
```

Il metodo converge in 15 iterazioni. Facciamo notare che un problema di questo approccio risiede nel fatto che la matrice $\mathrm{A}^T\mathrm{A}$ 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 $\mathrm{B} = (\mathrm{I} - \alpha\mathrm{A})$ e $\mathbf{g} = \alpha\mathbf{b}$. Il metodo converge se e solo se $\rho(\mathrm{B}) < 1$. Poiché

$$\rho(\mathrm{B}) = \max_i |\lambda_i(\mathrm{B})| = \max_i |\lambda_i(\mathrm{I} - \alpha\mathrm{A})| = \max_i |1 - \alpha\lambda_i(\mathrm{A})|,$$

la convergenza è garantita se $|1 - \alpha\lambda_i(\mathrm{A})| < 1$ per ogni $\lambda_i(\mathrm{A})$. 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 disuguaglianza richiesta è soddisfatta se $0 < \alpha < 2/\lambda_i(A)$ per ogni $i = 1, \ldots, 4$ 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 ouput: `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 $\delta\mathbf{b} = [10^{-3}, 0, \ldots, 0]^T$, otteniamo

$$\frac{\|\mathbf{x} - \widehat{\mathbf{x}}\|}{\|\mathbf{x}\|} \leq K_2(\mathrm{A}) \frac{\|\delta\mathbf{b}\|}{\|\mathbf{b}\|}.$$

Eseguendo i seguenti comandi MATLAB:

```
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 $\|\mathbf{x} - \widehat{\mathbf{x}}\|/\|\mathbf{x}\| \leq 0.0018$, a fronte di un errore relativo sui dati $\|\delta\mathbf{b}\|/\|\mathbf{b}\| \simeq 3.46 \cdot 10^{-6}$, 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 MATLAB 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 $\|\mathbf{x} - \widehat{\mathbf{x}}\|/\|\mathbf{x}\| \simeq 1.23 \cdot 10^{-4}$ (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 ρ(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. Peraltro, 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 $\mathbf{x}^{(0)} = \mathbf{0}$, 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 MATLAB, 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 A_1: il metodo converge in 34 passi al valore 2.00000000004989. A_2: a partire dallo stesso vettore iniziale servono ora 457 iterazioni per ottenere il valore 1.99999999990611. Il peggioramento nella velocità di convergenza è dovuto al fatto che i due autovalori più grandi in modulo sono molto vicini tra loro. Infine, per A_3 il metodo non converge in quanto questa matrice ha come autovalori di modulo massimo i e $-i$.

Soluzione 6.2 La matrice di Leslie associata ai valori riportati in tabella è

$$A = \begin{bmatrix} 0 & 0.5 & 0.8 & 0.3 \\ 0.2 & 0 & 0 & 0 \\ 0 & 0.4 & 0 & 0 \\ 0 & 0 & 0.8 & 0 \end{bmatrix}.$$

Con il metodo delle potenze si trova che $\lambda_1 \simeq 0.5353$ e la distribuzione per fasce d'età è data dalle componenti dell'autovettore di norma unitaria associato cioè $\mathbf{x}_1 \simeq (0.8477, 0.3167, 0.2367, 0.3537)^T$.

Soluzione 6.3 Riscriviamo il generico vettore iniziale come

$$\mathbf{y}^{(0)} = \beta^{(0)} \left(\alpha_1 \mathbf{x}_1 + \alpha_2 \mathbf{x}_2 + \sum_{i=3}^{n} \alpha_i \mathbf{x}_i \right),$$

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 $k \to \infty$ 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.

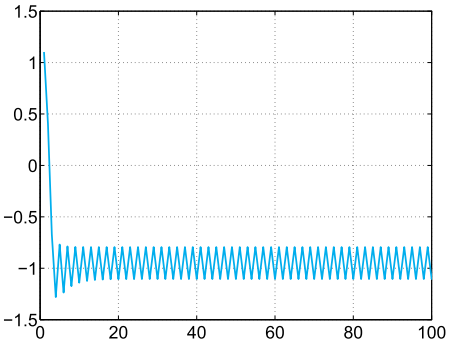

Figura 10.11. Approssimazioni dell'autovalore di modulo massimo calcolate dal metodo delle potenze al variare del numero di iterazioni per la matrice della Soluzione 6.5

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 `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 `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

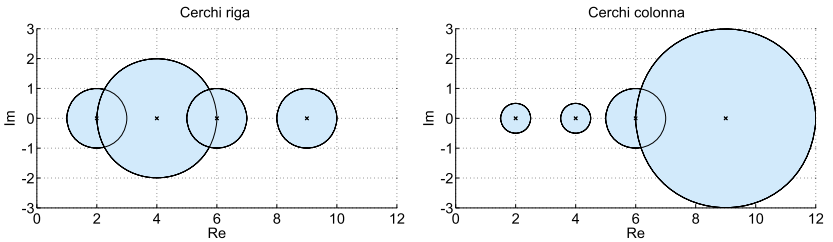

Figura 10.12. Cerchi riga (*a sinistra*) e colonna (*a destra*) per la matrice A della Soluzione 6.7

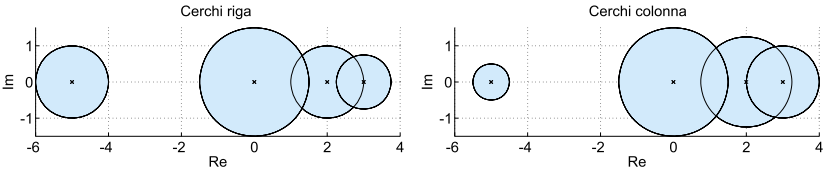

Figura 10.13. Cerchi riga (*a sinistra*) e colonna (*a destra*) per la matrice B della Soluzione 6.7

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

$$\mathrm{A}^{(k)} = \mathrm{Q}^{(k+1)}\mathrm{R}^{(k+1)} \quad \text{e} \quad \mathrm{A}^{(k+1)} = \mathrm{R}^{(k+1)}\mathrm{Q}^{(k+1)}$$

e quindi

$$(\mathrm{Q}^{(k+1)})^T \mathrm{A}^{(k)} \mathrm{Q}^{(k+1)} = \mathrm{R}^{(k+1)} \mathrm{Q}^{(k+1)} = \mathrm{A}^{(k+1)}.$$

Si conclude che, essendo $(\mathrm{Q}^{(k+1)})^T = (\mathrm{Q}^{(k+1)})^{-1}$ la matrice $\mathrm{A}^{(k)}$ è simile alla matrice $\mathrm{A}^{(k+1)}$ 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 `xmin=-3.660253989004456e-01` in 42 iterazioni ed il valore minimo della funzione è `fmin=-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 `xmin=-3.660254076197302e-01`.

Soluzione 7.2 Date $\boldsymbol{\gamma}_i(t) = (x_i(t), y_i(t))$, 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 `tmin=8.438731484275010`. Le due navi si trovano alla distanza minima pari a `dmin=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 `xm=[-2.1861e+00, 2.1861e+00]` è raggiunto in 181 iterazioni e sono state effettuate 353 valuazioni funzionali. Per simmetria, l'altro punto di minimo globale è $(-2.1861, 2.1861)$. Mettiamo in guardia il lettore che scegliendo `x0=[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 $x^{(k+1)} = x^{(k)} + \alpha_k d^{(k)}$ come

$$x^{(k+1)} = x^{(0)} + \sum_{\ell=0}^{k} \alpha_\ell d^{(\ell)},$$

abbiamo $x^{(0)} = 3/2$ e

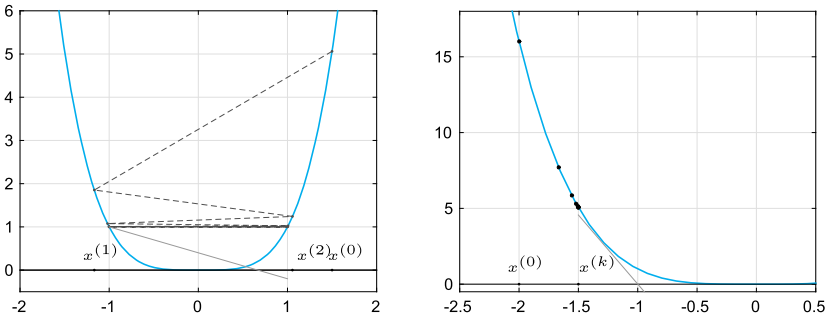

Figura 10.14. A sinistra: la successione del metodo di discesa della Soluzione 7.4. Prendendo $x^{(k)} \simeq -1$, il punto $(x^{(k+1)}, f(x^{(k+1)}))$ dovrebbe trovarsi sotto la retta azzurra per soddisfare la prima condizione di Wolfe con $\sigma = 0.2$, al contrario esso rimane abbondantemente sopra, 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 $(-1.5, 5.06)$

$$x^{(k+1)} = \frac{3}{2} + \left(2 + \frac{2}{3^{k+1}}\right)(-1)^{k+1} = \frac{3}{2} - 2\sum_{\ell=0}^{k}(-1)^{\ell} - \frac{1}{2} - \frac{1}{6}\left(-\frac{1}{3}\right)^{k}$$
$$= (-1)^{k+1}\left(1 + \frac{1}{6 \cdot 3^{k}}\right),$$

cioè la successione $x^{(k)}$ non converge a zero per $k \to \infty$ 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 $x^{(k+1)} = -2 + (1 - 3^{-k})/2 \to -3/2$ per $k \to \infty$. 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 graficamante 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

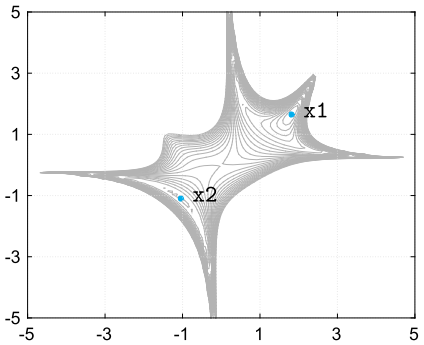

Figura 10.15. Linee di livello tra i valori 0 e 20 della funzione della Soluzione 7.7

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 `x1=(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 `x2=(-5.3282e-01, -5.8850e-01)`. Si ha $f(\mathtt{x1}) \simeq 3.6661$ e $f(\mathtt{x2}) \simeq 8.2226$. Prendendo $\mathbf{x}^{(0)} = (2, 1)$, entrambi i metodi convergono al punto di minimo globale `x1` in 11 iterazioni.

Soluzione 7.8 Calcolare i punti stazionari di $\tilde{f}_k(\mathbf{x}) = \frac{1}{2}\|\widetilde{\mathbf{R}}_k(\mathbf{x})\|^2$ equivale a risolvere il sistema non lineare

$$\nabla \tilde{f}_k(\mathbf{x}) = \mathrm{J}_{\widetilde{\mathbf{R}}_k}(\mathbf{x})^T \widetilde{\mathbf{R}}_k(\mathbf{x}) = \mathbf{0}. \tag{10.5}$$

Per la definizione (7.64) si ha $\mathrm{J}_{\widetilde{\mathbf{R}}_k}(\mathbf{x}) = \mathrm{J}_{\mathbf{R}}(\mathbf{x}^{(k)})$ per ogni $\mathbf{x} \in \mathbb{R}^n$ e il sistema (10.5) diventa

$$\mathrm{J}_{\mathbf{R}}(\mathbf{x}^{(k)})^T \mathbf{R}(\mathbf{x}^{(k)}) + \mathrm{J}_{\mathbf{R}}(\mathbf{x}^{(k)})^T \mathrm{J}_{\mathbf{R}}(\mathbf{x}^{(k)})(\mathbf{x} - \mathbf{x}^{(k)}) = \mathbf{0},$$

ovvero (7.63).

Soluzione 7.9 Dobbiamo dimostrare che $\boldsymbol{\delta}\mathbf{x}^{(k)}$ soddisfa le condizioni (5.67). Ricordiamo che per ogni matrice A con rango massimo, la matrice $\mathrm{A}^T\mathrm{A}$ è simmetrica e definita positiva.

Dimostriamo $(5.67)_2$. Dalla definizione $\nabla f(\mathbf{x}^{(k)}) = \mathrm{J}_{\mathbf{R}}(\mathbf{x}^{(k)})^T \mathbf{R}(\mathbf{x}^{(k)})$, segue che $\nabla f(\mathbf{x}^{(k)}) = \mathbf{0} \Leftrightarrow \mathbf{R}(\mathbf{x}^{(k)}) = \mathbf{0}$ (poiché $\mathrm{J}_{\mathbf{R}}(\mathbf{x}^{(k)})$ ha rango massimo) e quindi da $(7.63)_1$ segue $\boldsymbol{\delta}\mathbf{x}^{(k)} = \mathbf{0}$.

Supponiamo ora che $\mathbf{R}(\mathbf{x}^{(k)}) \neq \mathbf{0}$, si ha

$$\begin{aligned}
&(\boldsymbol{\delta}\mathbf{x}^{(k)})^T \nabla f(\mathbf{x}^{(k)}) \\
&\quad = -\left\{\left[\mathrm{J}_{\mathbf{R}}(\mathbf{x}^{(k)})^T \mathrm{J}_{\mathbf{R}}(\mathbf{x}^{(k)})\right]^{-1} \mathrm{J}_{\mathbf{R}}(\mathbf{x}^{(k)})^T \mathbf{R}(\mathbf{x}^{(k)})\right\}^T \mathrm{J}_{\mathbf{R}}(\mathbf{x}^{(k)})^T \mathbf{R}(\mathbf{x}^{(k)}) \\
&\qquad - \left(\mathrm{J}_{\mathbf{R}}(\mathbf{x}^{(k)})^T \mathbf{R}(\mathbf{x}^{(k)})\right)^T \left[\mathrm{J}_{\mathbf{R}}(\mathbf{x}^{(k)})^T \mathrm{J}_{\mathbf{R}}(\mathbf{x}^{(k)})\right]^{-1} \left(\mathrm{J}_{\mathbf{R}}(\mathbf{x}^{(k)})^T \mathbf{R}(\mathbf{x}^{(k)})\right) < 0,
\end{aligned}$$

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 $i = 1, \ldots, 8$, i coefficienti cercati $x_1, \ldots, x_5$, 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 $\mathrm{J}_{\mathbf{R}}(\mathbf{x})$ rispettivamente:

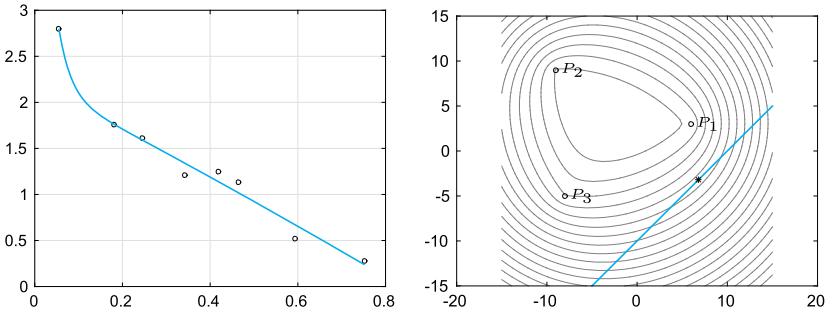

Figura 10.16. A sinistra: i dati e la soluzione dell'Esercizio 7.10. A destra: la soluzione dell'Esercizio 7.12. In grigio le curve di livello della funzione obiettivo. Il dominio Ω di ammissibilità è la parte illimitata del piano sottostante la retta azzurra.

```
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 $f(x) = 1.8428\text{e}{-01}$, 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 `x0 = [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 $x_1, \ldots, x_5$ 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

$$\tilde{f}_k(\mathbf{s}) = f(\mathbf{x}^{(k)}) + \mathbf{s}^T \nabla f(\mathbf{x}^{(k)}) + \frac{1}{2}\mathbf{s}^T \mathrm{H}_k \mathbf{s} \quad \forall \mathbf{s} \in \mathbb{R}^n,$$

dove H_k è l'Hessiana in $\mathbf{x}^{(k)}$ o una sua approssimazione. Ricordando $(7.62)_1$ e prendendo

$$\mathrm{H}_k = \mathrm{J}_{\mathbf{R}}(\mathbf{x}^{(k)})^T \mathrm{J}_{\mathbf{R}}(\mathbf{x}^{(k)}),$$

otteniamo

$$\begin{aligned}\tilde{f}_k(\mathbf{s}) &= \frac{1}{2}\|\mathbf{R}(\mathbf{x}^{(k)})\|^2 + \mathbf{s}^T \mathrm{J}_{\mathbf{R}}(\mathbf{x}^{(k)})^T \mathbf{R}(\mathbf{x}^{(k)}) + \frac{1}{2}\mathbf{s}^T \mathrm{J}_{\mathbf{R}}(\mathbf{x}^{(k)})^T \mathrm{J}_{\mathbf{R}}(\mathbf{x}^{(k)})\mathbf{s} \\ &= \frac{1}{2}\|\mathbf{R}(\mathbf{x}^{(k)}) + \mathrm{J}_{\mathbf{R}}(\mathbf{x}^{(k)})\mathbf{s}\|^2 \\ &= \frac{1}{2}\|\widetilde{\mathbf{R}}_k(\mathbf{x})\|^2,\end{aligned}$$

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 $\widetilde{\mathbf{R}}_k(\mathbf{x}) = \mathbf{R}(\mathbf{x}^{(k)}) + \mathrm{J}_{\mathbf{R}}(\mathbf{x})(\mathbf{x} - \mathbf{x}^{(k)})$.

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 $\ker \mathrm{C} = \{\mathbf{z} = \alpha[1,1,1]^T, \alpha \in \mathbb{R}\}$. 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 $\mathrm{M} = [\mathrm{A},\ \mathrm{C}^T; \mathrm{C},\ 0]$ 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];
xl=M\f;
```

ottenendo la soluzione

```
xl =
   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 `xl`, 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

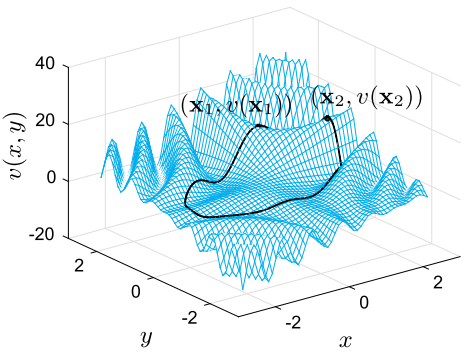

Figura 10.17. La funzione $v(x, y)$ dell'Esercizio 7.14 e i due massimi vincolati calcolati con il metodo della Lagrangiana aumentata

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 $L = \max_{0\le t\le 1} |L(t)|$ 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 $M = \max_{0\le t\le 1} |y''(t)|$ con $y'' = -e^{-y} - t^2e^{-2y}$, 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 $M = \max_{0\le t\le 1} |y''(t)|$ 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^2e^{-2y} e quindi la funzione $y'' = -e^{-y} - t^2e^{-2y}$ è negativa e decrescente. Questo ci permette di concludere che $M = \max_{0\le t\le 1} |y''(t)|$ è assunto in $t = 1$.

Dalla (8.29), per $h = 0.01$ si ricava allora

$$|u_{100} - y(1)| \le \frac{e^L - 1}{L}\frac{M}{200} \simeq 0.26$$

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(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:

$$y(t_{n+1}) - y(t_n) \simeq \frac{h}{2}\left[f(t_n, y(t_n)) + f(t_{n+1}, y(t_{n+1}))\right].$$

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

$$|z_n - u_n| \leq \overline{\rho}[a(h)]^n + h\overline{\rho} \sum_{k=0}^{n-1} [a(h)]^{n-k-1}$$

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:

$$\begin{aligned} &u_{n+1} = u_n + \frac{h}{2}(K_1 + K_2) \\ &K_1 = f(t_n, u_n), \qquad K_2 = f(t_{n+1}, u_n + hK_1). \end{aligned} \tag{10.6}$$

Abbiamo $h\tau_{n+1}(h) = y(t_{n+1}) - y(t_n) - h(\widehat{K}_1 + \widehat{K}_2)/2$, con $\widehat{K}_1 = f(t_n, y(t_n))$ e $\widehat{K}_2 = f(t_{n+1}, y(t_n) + h\widehat{K}_1)$. Poiché f è continua rispetto ad entrambi gli argomenti si ha

$$\lim_{h\to 0} \tau_{n+1} = y'(t_n) - \frac{1}{2}[f(t_n, y(t_n)) + f(t_n, y(t_n))] = 0.$$

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 MATLAB:

```
p=log(abs(e(1:end-1)./e(2:end)))/log(2);
p(1:2:end)
```

si trovano le seguenti stime per l'ordine:

```
p =
    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.'; % trasposta anche di variabili complesse
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:

$$h\tau_{n+1}(h) = y(t_{n+1}) - y(t_n) - \frac{h}{6}(\widehat{K}_1 + 4\widehat{K}_2 + \widehat{K}_3),$$

$$\widehat{K}_1 = f(t_n, y(t_n)), \qquad \widehat{K}_2 = f(t_n + \tfrac{h}{2}, y(t_n) + \tfrac{h}{2}\widehat{K}_1),$$

$$\widehat{K}_3 = f(t_{n+1}, y(t_n) + h(2\widehat{K}_2 - \widehat{K}_1)).$$

Essendo f continua rispetto ad entrambi gli argomenti, si ha

$$\lim_{h\to 0} \tau_{n+1} = y'(t_n) - \frac{1}{6}[f(t_n, y(t_n)) + 4f(t_n, y(t_n)) + f(t_n, y(t_n))] = 0,$$

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.'; % trasposta anche di variabili complesse
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+h*(2*k2-k1);
    k3=odefun(tt(n+1),y);
    w=wn+h6*(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

$$u_{n+1} = u_n[1 + h\lambda + \frac{1}{2}(h\lambda)^2 + \frac{1}{6}(h\lambda)^3] = u_n p_3(h\lambda).$$

Da uno studio grafico, effettuato con i seguenti comandi

```
c=[1/6 1/2 1 1]; z=[-3:0.01:1];
p=polyval(c,z); plot(z,abs(p))
```

si deduce che, 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.

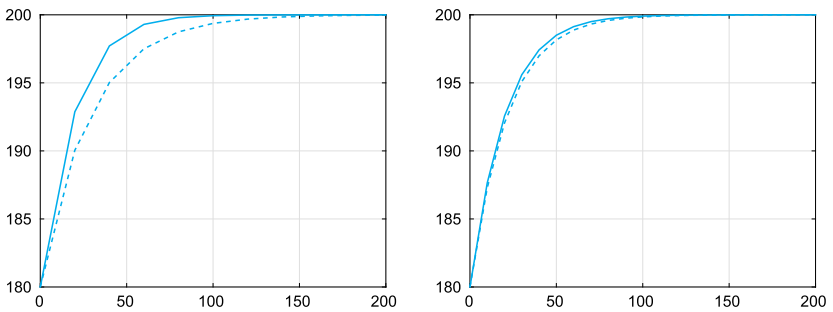

Figura 10.18. Soluzioni calcolate con $N = 10$ (*a sinistra*) e $N = 20$ (*a destra*) per il problema di Cauchy della Soluzione 8.15: in linea continua le soluzioni ottenute con il metodo di Crank-Nicolson, in linea tratteggiata quelle ricavate con il metodo RK2 o di Heun

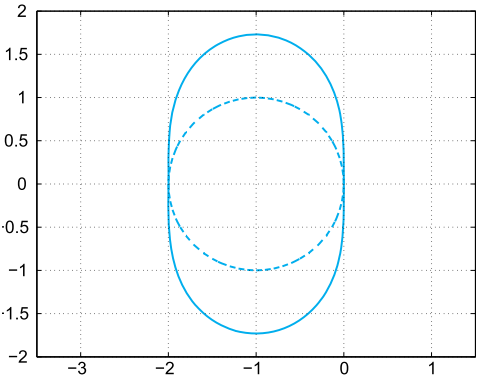

Figura 10.19. Bordo delle regioni di assoluta stabilità per i metodi di Eulero esplicito (*linea tratteggiata*) e di Heun (*linea continua*). Le regioni si estendono all'interno delle aree delimitate dalle rispettive curve

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.

contour

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:

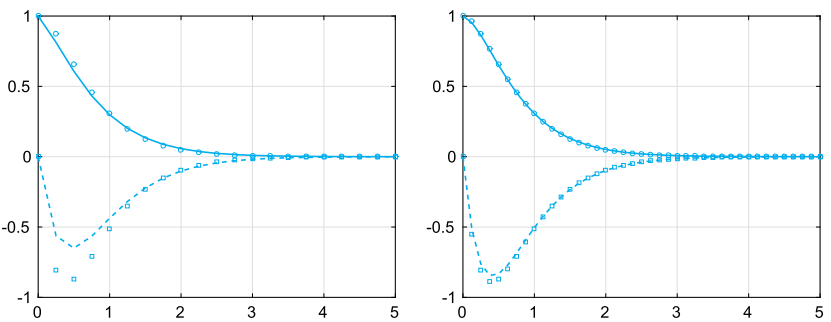

Figura 10.20. Approssimazioni di $x(t)$ (*linea continua*) e $x'(t)$ (*linea tratteggiata*) calcolate in corrispondenza di $N = 20$ (*a sinistra*) e $N = 40$ (*a destra*). I cerchietti ed i quadratini si riferiscono alle quantità esatte $x(t)$ e $x'(t)$, rispettivamente

$$\begin{cases} x'(t) = z(t), \\ y'(t) = v(t), \\ z'(t) = 2\omega \sin(\Psi) v(t) - k^2 x(t), \\ v'(t) = -2\omega \sin(\Psi) z(t) - k^2 y(t). \end{cases} \tag{10.7}$$

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:

$$x(0) = 1, \qquad y(0) = 0, \qquad z(0) = 0, \qquad v(0) = 0.$$

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.

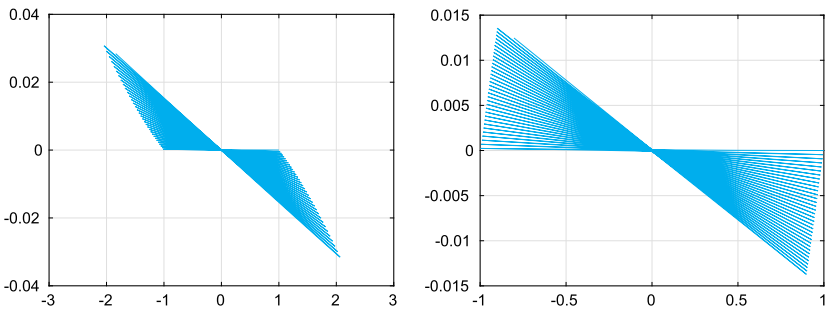

Figura 10.21. Traiettorie nel piano delle fasi per il pendolo di Focault della Soluzione 8.19, ottenute con il metodo di Eulero esplicito (*a sinistra*) e con un metodo Runge-Kutta adattivo (*a destra*)

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
```

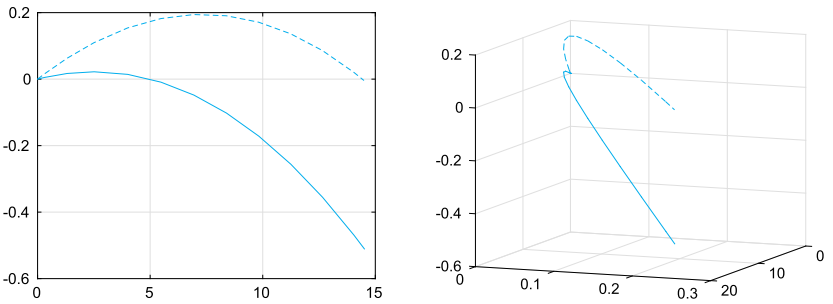

Figura 10.22. Le traiettorie seguite da una palla da baseball lanciata con angolo iniziale pari a 1 grado (*linea continua*) e 3 gradi (*linea tratteggiata*)

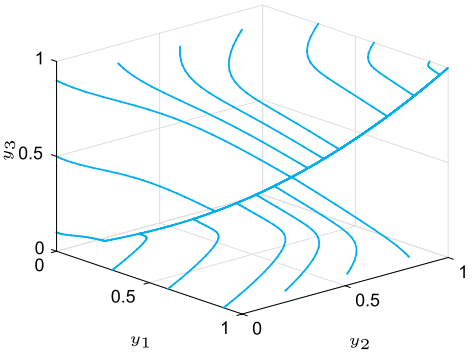

Figura 10.23. Le traiettorie del modello (8.117) per vari dati iniziali ed $\varepsilon = 10^{-2}$

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, $\mathrm{div}\nabla\phi = \partial^2\phi/\partial x^2 + \partial^2\phi/\partial y^2$ 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

$$\int_\Omega \nabla \cdot \mathbf{v} d\Omega = \int_{\partial\Omega} \mathbf{v} \cdot \mathbf{n} d(\partial\Omega). \tag{10.8}$$

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 $\mathrm{A_{fd}} = h^{-2}\widehat{\mathrm{A}}$, verifichiamo la proprietà richiesta mostrando che $\mathbf{x}^T\widehat{\mathrm{A}}\mathbf{x} > 0$ per ogni $\mathbf{x} \neq \mathbf{0}$. (È evidente che $\mathrm{A_{fd}}$ è simmetrica.) Abbiamo

$$[x_1 \quad x_2 \quad \dots \quad x_{N-1} \quad x_N] \begin{bmatrix} 2 & -1 & 0 & \dots & 0 \\ -1 & 2 & \ddots & & \vdots \\ 0 & \ddots & \ddots & -1 & 0 \\ \vdots & & -1 & 2 & -1 \\ 0 & \dots & 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{N-1} \\ x_N \end{bmatrix}$$

$$= 2x_1^2 - 2x_1x_2 + 2x_2^2 - 2x_2x_3 + \ldots - 2x_{N-1}x_N + 2x_N^2.$$

A questo punto basta raccogliere opportunamente i termini per concludere che l'ultima espressione trovata è equivalente a $(x_1 - x_2)^2 + \ldots + (x_{N-1} - x_N)^2 + x_1^2 + x_N^2$, che è evidentemente positiva.

Soluzione 9.4 Verifichiamo che $\mathrm{A_{fd}}\mathbf{q}_j = \lambda_j \mathbf{q}_j$. Eseguiamo il prodotto matrice-vettore $\mathbf{w} = \mathrm{A_{fd}}\mathbf{q}_j$ 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):

$$\begin{cases} 2\sin(j\theta) - \sin(2j\theta) = 2(1 - \cos(j\theta))\sin(j\theta), \\ -\sin(j(k-1)\theta) + 2\sin(jk\theta) - \sin(j(k+1)\theta) = 2(1 - \cos(j\theta))\sin(kj\theta), \\ \quad k = 2, \ldots, N-1 \\ 2\sin(Nj\theta) - \sin((N-1)j\theta) = 2(1 - \cos(j\theta))\sin(Nj\theta). \end{cases}$$

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 $\mathrm{A_{fd}}$ simmetrica e definita positiva, $K(\mathrm{A_{fd}}) = \lambda_{\max}/\lambda_{\min}$ ovvero $K(\mathrm{A_{fd}}) = \lambda_N/\lambda_1 = (1 - \cos(N\pi/(N+1)))/(1 - \cos(\pi/(N+1)))$. 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 $K(\mathrm{A_{fd}}) \simeq (N+1)^2$ cioè $K(\mathrm{A_{fd}}) \simeq h^{-2}$.

Soluzione 9.5 Basta osservare che:

$$u(\bar{x}+h) = u(\bar{x}) + hu'(\bar{x}) + \frac{h^2}{2}u''(\bar{x}) + \frac{h^3}{6}u'''(\bar{x}) + \frac{h^4}{24}u^{(4)}(\xi_+),$$
$$u(\bar{x}-h) = u(\bar{x}) - hu'(\bar{x}) + \frac{h^2}{2}u''(\bar{x}) - \frac{h^3}{6}u'''(\bar{x}) + \frac{h^4}{24}u^{(4)}(\xi_-),$$

dove $\xi_+ \in (x, x+h)$ e $\xi_- \in (x-h, x)$.

Sommando membro a membro le due espressioni si trova

$$u(\bar{x}+h) + u(\bar{x}-h) = 2u(\bar{x}) + h^2u''(\bar{x}) + \frac{h^4}{24}(u^{(4)}(\xi_+) + u^{(4)}(\xi_-)),$$

da cui la proprietà desiderata.

Soluzione 9.6 La matrice è ancora tridiagonale ed ha elementi $(\mathrm{A_{fd}})_{i,i-1} = -\mu/h^2 - \eta/(2h)$, $(\mathrm{A_{fd}})_{ii} = 2\mu/h^2 + \sigma$, $(\mathrm{A_{fd}})_{i,i+1} = -\mu/h^2 + \eta/(2h)$. Il termine noto, una volta incorporate le condizioni al contorno, diventa conseguentemente $\mathbf{f} = (f(x_1) + \alpha(\mu/h^2 + \eta/(2h)), f(x_2), \ldots, f(x_{N-1}), f(x_N) + \beta(\mu/h^2 - \eta/(2h)))^T$.

Soluzione 9.7 Con le seguenti istruzioni MATLAB 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-uhex(1:50:end),inf)
e3=norm(uh41-uhex(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 $\mathbf{u} = (u_0, \ldots, u_{N+1})^T$ e la matrice ha dimensione $N + 2$. Nella prima e nell'ultima equazione del sistema dobbiamo imporre le condizioni

$$\frac{3u_0 - 4u_1 + u_2}{2h} = \gamma, \qquad \frac{3u_{N+1} - 4u_N + u_{N-1}}{2h} = \delta.$$

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 equispa-
% 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 MATLAB:

```
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/OCTAVE:

```
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 $\mathbb{P}\mathrm{e} = 500/201 > 1$ 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 $\mathbb{P}\mathrm{e} = 500/401 > 1$ e $\mathbb{P}\mathrm{e} = 500/801 < 1$, rispettivamente.

Il Programma 10.7 implementa lo schema *upwind* (9.39). Gli input e gli output assumono lo stesso significato di quelli del Programm 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];
```

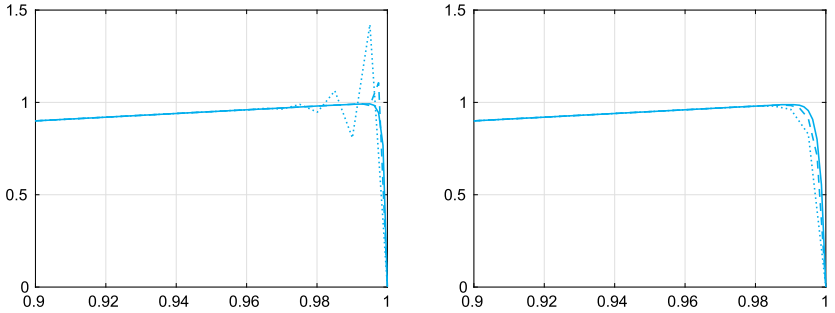

Figura 10.24. Le soluzioni dell'Esercizio 9.9 utilizzando lo schema centrato (9.36) (*a sinistra*) e lo schema *upwind* (9.39) (*a destra*). $N = 200$ (*linea continua*), $N = 500$ (*linea tratteggiata*) e $N = 800$ (*linea punteggiata*)

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 $j = 1, \ldots, N$, 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* $\mathrm{M} \in \mathbb{R}^{N\times N}$,

$$\mathrm{M}_{ji} = \int_a^b \varphi_i(x)\varphi_j(x)dx = \int_{I_{j-1}\cup I_j} \varphi_i(x)\varphi_j(x)dx, \quad i,j = 1,\ldots,N, \tag{9.9}$$

ovvero

$$\mathrm{M} = \frac{h}{6}\begin{bmatrix} 4 & 1 & 0 & \ldots & 0 \\ 1 & 4 & \ddots & & \vdots \\ 0 & \ddots & \ddots & 1 & 0 \\ \vdots & & 1 & 4 & 1 \\ 0 & \ldots & 0 & 1 & 4 \end{bmatrix}. \tag{9.10}$$

Infine il termine noto $\mathbf{f}_{\mathrm{fe}}$ di (9.48) può essere ottenuto con una operazione di prodotto matrice vettore tra la matrice di massa M ed il vettore $(f(x_1),\ldots,f(x_N))^T$.

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 M che abbiamo costruito in (9.10) coinvolge le sole funzioni di forma $\varphi_1,\ldots,\varphi_N$ 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:

$$\begin{aligned} err^2 &= \int_a^b (u(x)-u_h(x))^2dx = \sum_{j=0}^{N}\int_{x_j}^{x_{j+1}} (u(x)-u_h(x))^2dx \\ &= \sum_{j=0}^{N}\int_{x_j}^{x_{j+1}} \left(u(x)-\sum_{i=0}^{N} u_i\varphi_i(x)\right)^2 dx \\ &= \sum_{j=0}^{N}\int_{x_j}^{x_{j+1}} (u(x)-u_j\varphi_j(x)-u_{j+1}\varphi_{j+1}(x))^2dx \\ &= \sum_{j=0}^{N}\int_{x_j}^{x_{j+1}} \left(u(x)-u_j\frac{x-x_{j+1}}{x_j-x_{j+1}}-u_{j+1}\frac{x-x_j}{x_{j+1}-x_j})\right)^2 dx. \end{aligned}$$

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:

$$\begin{aligned} Int_j = \int_{x_j}^{x_{j+1}} e(x)dx &\simeq \sum_{i=0}^{n} e(y_i)\alpha_i \\ &= \sum_{i=0}^{n} e\left(\frac{x_{j+1}-x_j}{2}\overline{y}_i + \frac{x_{j+1}+x_j}{2}\right)\frac{x_{j+1}-x_j}{2}\overline{\alpha}_i \end{aligned}$$

Il calcolo di Int_j è realizzato nella seguente *function*:

```
function int=gl4c(u,uj,uj1,xj,xj1)
% int=gl4c(u,u1,u2,x1,x2) integrale con Gauss-Legendre
% calcola l'integrale di (u(x)-u_h(x))^2
% sull'intervallo [xj,xj1], dove
% uh(x)=uj*phi_j(x)+uj1*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 uh(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:

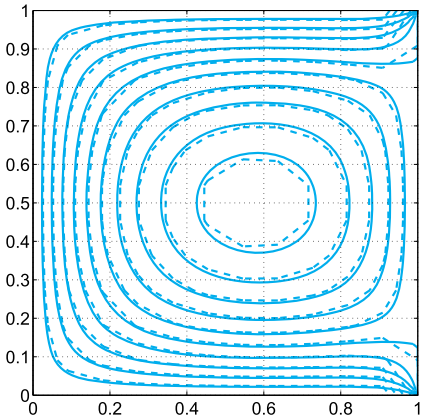

Figura 10.25. Le isolinee della temperatura (dell'Esercizio 9.13) calcolate per $h = 1/10$ (*linea tratteggiata*) tive a $h = 1/80$ (*linea continua*)

$$\int_a^b u_{tt}(x,t)u_t(x,t)dx + c\int_a^b u_x(x,t)u_{tx}(x,t)dx - c[u_x(x,t)u_t(x,t)]_a^b = 0. \tag{10.11}$$

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 $u_x u_{xt} = \frac{1}{2}(u_x^2)_t$, 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

$$\int_a^b u_t^2(x,t)dx + c\int_a^b u_x^2(x,t)dx$$
$$= \int_a^b v_0^2(x)dx + c\int_a^b u_{0x}^2(x)dx + 2c\int_0^t (u_x(b,s)u_t(b,s) - u_x(a,s)u_t(a,s))ds.$$

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

$$\int_0^t (u_x(b,s)u_t(b,s) - u_x(a,s)u_t(a,s))\mathrm{d}s = 0.$$

Quindi, (9.114) è dimostrata.

Soluzione 9.15 Per la definizione (9.92) basta verificare che

$$\sum_{j=-\infty}^{\infty} |u_j^{n+1}|^2 \le \sum_{j=-\infty}^{\infty} |u_j^n|^2. \tag{10.12}$$

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+1,1)=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
    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
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 $\phi_{k_1} = 4\pi \cdot 6.25\text{e}{-3} \simeq 0.078$ e $\phi_{k_2} = 20\pi \cdot 6.25\text{e}{-3} \simeq 0.393$. 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)

A. Quarteroni et al., *Calcolo Scientifico*, 6a edizione,
UNITEXT – La Matematica per il 3+2 105, DOI 10.1007/978-88-470-3953-7

[BM92] Bernardi, C., Maday, Y.: Approximations Spectrales des Problémes 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 Conservations 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

A. Quarteroni et al., *Calcolo Scientifico*, 6a edizione,
UNITEXT – La Matematica per il 3+2 105, DOI 10.1007/978-88-470-3953-7