Dispense del corso di Laboratorio di Calcolo Numerico

Dott. Marco Caliari

a.a. 2007/08

Questi appunti non hanno alcuna pretesa di completezza. Sono solo alcune note ed esercizi che affiancano il corso di Calcolo Numerico. Sono inoltre da considerarsi in perenne "under revision" e pertanto possono contenere discrepanze, inesattezze o errori.

Indice

Capitolo 1

Aritmetica floating-point

1.1 Overflow e underflow

Con i comandi `realmax` e `realmin` di GNU Octave, si ottengono il più grande numero rappresentabile e il più piccolo, rispettivamente. È importante tener presente questo fatto, per evitare i problemi di overflow e underflow, come mostrato nei seguenti esempi.

1. Si vuole trovare la media aritmetica di $a = 1.7 \cdot 10^{308}$ e $b = 1.6 \cdot 10^{308}$. Il calcolo $(a+b)/2$ produce `Inf`, mentre $b+(a-b)/2$ produce $1.6500 \cdot 10^{308}$.

2. Si vuole calcolare la norma euclidea del vettore $[v_1, v_2] = [3 \cdot 10^{307}, 4 \cdot 10^{307}]$. Il calcolo $\sqrt{v_1^2 + v_2^2}$ produce `Inf`, mentre $v_2 \cdot \sqrt{(v_1/v_2)^2 + 1}$ produce $5.0000 \cdot 10^{307}$.

3. Si vuole calcolare il prodotto tra gli elementi del vettore `v` definito da `v=[realmax,realmax,1/realmax,1/realmax]`. Il comando `prod(v)` produce `Inf`. Si perde anche la proprietà commutativa del prodotto (`prod(w)`=0 con `w=[1/realmax,1/realmax,realmax,realmax]`). Il comando `exp(sum(log(v)))` produce il risultato corretto.

4. Si vuole calcolare il coefficiente binomiale $\binom{n}{m}$, con $n = 181$ e $m = 180$. Il calcolo $n!/(m!(n-m)!)$ produce un overflow, mentre $\exp(\ln(n!) - \ln(m!) - \ln(n-m)) = \exp(\ln\Gamma(n+1)! - \ln\Gamma(m+1)! - \ln\Gamma(n-m+1)!)$ produce $1.81 \cdot 10^2$.

1.2 Cancellazione

Si ha cancellazione numerica quando si sottraggono due numeri con lo stesso segno e di modulo vicino. Quando i due numeri sono arrotondati, si possono avere risultati molto imprecisi, come mostrato nei seguenti esempi.

1. Dato $x = 8 \cdot 10^{-16}$, il calcolo di $((1 + x) - 1)/x$ produce 1.1102.

2. Dato $x = 1.005$, il calcolo di $(x - 1)^7$ produce $7.8125 \cdot 10^{-17}$, mentre il calcolo di $x^7 - 7x^6 + 21x^5 - 35x^4 + 35x^3 - 21x^2 + 7x - 1$ produce $-8.88178419700125 \cdot 10^{-16}$.

3. L'equazione $ax^2 + bx + c$, con $a = 1 \cdot 10^{-10}$, $b = 1$, $c = 1 \cdot 10^{-4}$ ha una soluzione data da
$$\frac{-b + \sqrt{b^2 - 4ac}}{2a} . \tag{1.1}$$
Il calcolo di $(-b + \sqrt{b^2 - 4ac})/(2a)$ produce $x_1 = -9.99200722162641 \cdot 10^{-5}$. Il calcolo di $ax_1^2 + bx_1 + c$ produce $7.99277837369190 \cdot 10^{-8}$. Razionalizzando (1.1) si ottiene
$$\frac{2c}{-b - \sqrt{b^2 - 4ac}}$$
il cui calcolo produce $\hat{x}_1 = -1.00000000000001 \cdot 10^{-4}$. Il calcolo di $a\hat{x}_1^2 + b\hat{x}_1 + c$ produce $0.00000000000000 \cdot 10^{0}$.

4. Il seguente codice

```
clear all
for i = 1:15
  x(i) = 10^(-i);
  y(i) = (exp(x(i))-1)/x(i);
end
semilogx(x,y,'-o')
```

produce un grafico del valore di $f(x) = (e^x - 1)/x$ calcolato per x sempre più prossimo a 0, mostrando che la successione $y^{(n)}$ non converge a $\lim_{x \to 0} f(x) = 1$.

1.3 Esercizi

1. Si calcoli $a^2 - b^2$ con $a = 1.4 \cdot 10^{154}$ e $b = 1.3 \cdot 10^{154}$.

2.? Dato $x = 8.88178419700125184 \cdot 10^{-16}$, si calcoli $((1+x)-1)/x$. Perché il risultato è molto più accurato che con $x = 8 \cdot 10^{-16}$?

3. Sia $s^{(n)}(x) = 1 + x + x^2/2 + x^3/3! + x^4/4! + \ldots + x^n/n!$ la troncata n-esima della serie esponenziale. Si prenda $x = -10$ e si produca un grafico semilogaritmico nelle ordinate, per $n = 1, 2, \ldots, 80$, dell'errore relativo $\left|s^{(n)}(x) - \exp(x)\right| / \exp(x)$. Si individui il problema.

4.? La successione definita per ricorrenza

$$\begin{aligned} z^{(1)} &= 1 \\ z^{(2)} &= 2 \\ z^{(n+1)} &= 2^{n-1/2}\sqrt{1 - \sqrt{1 - 4^{1-n} z^{(n)^2}}}, \quad n > 1 \end{aligned}$$

converge a π. La si implementi in GNU Octave e si produca un grafico semilogaritmico nelle ordinate dell'errore relativo $|z_n - \pi| / \pi$, per $n = 1, 2, \ldots, 30$. Si individui il problema e si proponga una strategia per risolverlo.

5.?! Sia

$$I^{(n)} = \frac{1}{e} \int_0^1 x^n e^x \mathrm{d}x \, . \tag{1.2}$$

È facile verificare che $I^{(1)} = 1/e$ e

$$I^{(n)} = 1 - nI^{(n-1)} \tag{1.3}$$

e $I^{(n)} > 0$ e $\lim_{n\to\infty} I^{(n)} = 0$. Si calcoli $I^{(n)}$, $n = 2, 3, \ldots, 30$ usando (1.3) e si confrontino i valori con la *soluzione di riferimento* data da `quad(@(x)integranda(x,n),0,1)`, ove `integranda` è una function che implementa la funzione integranda in (1.2). Si individui il problema e si proponga una strategia per risolverlo.

Capitolo 2

Equazioni non lineari

2.1 Metodo di bisezione

Data la successione $\{x^{(k)}\}_k$ prodotta dal metodo di bisezione convergente alla radice ξ di $f(x)$, il criterio d'arresto basato sul *residuo* $f(x^{(k)})$ (cioè dedurre $x^{(k)} \approx \xi$ se $f(x^{(k)}) \approx 0$) non è un buon criterio. Si ha infatti, sviluppando in serie di Taylor,

$$f(x) = f(x^{(k)}) + f'(x^{(k)})(x_\xi - x^{(k)})$$

da cui

$$\left| \frac{f(x^{(k)})}{f'(x^{(k)})} \right| = \left| x_\xi - x^{(k)} \right| \approx \left| \xi - x^{(k)} \right|$$

se $x_\xi \approx \xi$. È però certamente vero che se $f(x^{(k)}) = 0$ allora $x^{(k)} = \xi$ (nell'ipotesi di unicità della radice). Tuttavia, a causa degli errori di arrotondamento, l'ipotetica istruzione

```
if (feval(f,x(k)) == 0)
```

dovrebbe essere sostituita da

```
if (abs(feval(f,x(k))) < eps)
```

È anche ipotizzabile di moltiplicare `eps` per un opportuno parametro di *sicurezza*, per esempio 10.

2.2 Ordine dei metodi

Data la successione $\{x^{(k)}\}_k$ convergente a ξ, l'ordine di convergenza è p se

$$\lim_{k\to\infty} \frac{\left| x^{(k+1)} - \xi \right|}{\left| x^{(k)} - \xi \right|^p} = C \neq 0 \, .$$

Si ricava dunque

$$p = \lim_{k\to\infty} \frac{\log\left|x^{(k+1)} - \xi\right| - \log C}{\log|x^{(k)} - \xi|} \approx \lim_{k\to\infty} \frac{\log\left|x^{(k+1)} - \xi\right|}{\log|x^{(k)} - \xi|} \ . \qquad (2.1)$$

Se non si conosce ξ, si può usare una sua approssimazione $x^{(K)}$, con K sufficientemente maggiore di $k+1$.

2.3 Metodi di Newton modificati

I metodi delle corde e delle secanti possono essere interpretati come metodi di Newton *modificati*, in cui si sostituisce la derivata della funzione con una opportuna approssimazione. In particolare, per il metodo delle corde, si ha

$$f'(x^{(k)}) \approx \frac{f(b) - f(a)}{b - a} = c$$

e per il metodo delle secanti

$$f'(x^{(k)}) \approx \frac{f(x^{(k)}) - f(x^{(k-1)})}{x^{(k)} - x^{(k-1)}} \ .$$

2.4 Metodo di Newton–Hörner

2.4.1 Metodo di Hörner per la valutazione di un polinomio

Consideriamo il polinomio

$$p_{n-1}(x) = a_1 x^{n-1} + a_2 x^{n-2} + \ldots + a_{n-1}x + a_n \ .$$

Per la sua valutazione in un punto z occorrono $n-1$ addizioni e $2n-3$ moltiplicazioni. Se consideriamo invece la formulazione equivalente

$$p_{n-1}(x) = (((a_1 x + a_2)x + a_3)x + \ldots)x + a_n \ ,$$

per la stessa valutazione bastano $n-1$ addizioni e $n-1$ moltiplicazioni. Dati il vettore riga `a` dei coefficienti e un vettore colonna `z` di valori, il metodo di Hörner si implementa mediante l'algoritmo in Tabella 2.1, ove il vettore colonna `pnz=b(:,n)` contiene il valore del polinomio calcolato nei punti `z`.

Il polinomio (dipendente da z) definito da

$$q_{n-2}(x; z) = b_1 x^{n-2} + b_2 x^{n-3} + \ldots + b_{n-2}x + b_{n-1}$$

```
function [pnz,b] = horner(a,z)
%
% [pnz,b] = horner(a,z)
%
n = length(a);
m = length(z);
b = zeros(m,n);
b(:,1) = a(1);
for j = 2:n
  b(:,j) = a(j)+b(:,j-1).*z;
end
pnz = b(:,n);
```

Tabella 2.1: Metodo di Hörner

è chiamato *polinomio associato* a $p_{n-1}(x)$. Valgono le seguenti uguaglianze:

$$p_{n-1}(x) = b_n + (x - z)q_{n-2}(x; z)$$
$$p'_{n-1}(z) = q_{n-2}(z; z)$$

Se inoltre z è una radice di $p_{n-1}(x)$, allora $b_n = 0$ e $p_{n-1}(x) = (x-z)q_{n-2}(x; z)$ e dunque $q_{n-2}(x; z)$ ha per radici le restanti $n - 2$ radici di $p_{n-1}(x)$.

2.4.2 Metodo di Newton–Hörner per il calcolo delle radici di polinomi

Le considerazioni fatte al paragrafo precedente permettono di implementare in maniera efficiente il metodo di Newton per il calcolo di tutte le radici (anche complesse) dei polinomi. Infatti, dato il punto $x^{(k)}$ e $b_n = p_{n-1}(x^{(k)})$, l'algoritmo per calcolare $x_1^{(k+1)}$ (approssimazione $k + 1$-esima della prima radice di $p_{n-1}(x)$) è

$$x_1^{(k+1)} = x_1^{(k)} - b_n/q_{n-2}(x_1^{(k)}; x_1^{(k)})$$

ove $q_{n-2}(x_1^{(k)}; x_1^{(k)})$ è calcolato nuovamente mediante il metodo di Hörner. Una volta arrivati a convergenza, diciamo con il valore x_1^K, il polinomio in x $q_{n-2}(x; x_1^{(K)})$ conterrà le rimanenti radici di $p_{n-1}(x)$ e si applicherà nuovamente il metodo di Newton–Hörner a tale polinomio. Tale tecnica è detta *deflazione* e si arresterà a $q_1(x; x_{n-2}^{(K)})$, per il quale $x_{n-1} = -b_2/b_1$. Per assicurare la convergenza alle radici complesse, è necessario scegliere una soluzione iniziale con parte immaginaria non nulla.

2.5 Accelerazione di Aitken

Data l'equazione di punto fisso

$$x = g(x) \ ,$$

sappiamo che

$$\lim_{k\to\infty} \frac{\xi - x^{(k)}}{\xi - x^{(k-1)}} = g'(\xi)$$

e

$$\lim_{k\to\infty} \lambda^{(k)} = \lim_{k\to\infty} \frac{x^{(k)} - x^{(k-1)}}{x^{(k-1)} - x^{(k-2)}} = g'(\xi) \ .$$

Allora,

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

viene detta *estrapolazione di Aitken*. Si noti che il termine $\hat{x}^{(k)}$ non dipende dai termini $\hat{x}^{(k-1)}, \hat{x}^{(k-2)}, \ldots$ e la tecnica di accelerazione può essere usata ad una qualunque successione $\{x^{(k)}\}_k$ convergente. La successione $\{\hat{x}^{(k)}\}_k$ converge, in generale, a ξ più velocemente della successione $\{x^{(k)}\}_k$, cioè

$$\lim_{k\to\infty} \frac{\hat{x}^{(k)} - \xi}{x^{(k)} - \xi} = 0$$

2.5.1 Metodo di Steffensen

Una variante del metodo di Aikten è il metodo di Steffensen. Data l'equazione di punto fisso

$$x = g(x)$$

il metodo di Steffensen è definito da

$$x^{(k+1)} = x^{(k)} - \frac{(g(x^{(k)}) - x^{(k)})^2}{g(g(x^{(k)})) - 2g(x^{(k)}) + x^{(k)}} \ .$$

2.6 Metodo di Newton per radici multiple

2.6.1 Stima della molteplicità

Il metodo di Newton converge con ordine 1 alle radici multiple. Infatti, la funzione di iterazione g associata al metodo di Newton soddisfa

$$g'(\xi) = 1 - \frac{1}{m}$$

se ξ è una radice di molteplicità m. Tuttavia, spesso il valore di m non è noto a priori. Visto che comunque la successione $\{x^{(k)}\}_k$ converge (linearmente) a ξ, si ha

$$\lim_{k\to\infty} \frac{x^{(k)} - x^{(k-1)}}{x^{(k-1)} - x^{(k-2)}} = \lim_{k\to\infty} \frac{g(x_{k-1}) - g(x_{k-2})}{x^{(k-1)} - x^{(k-2)}} = g'(\xi) = 1 - \frac{1}{m}$$

e dunque

$$\lim_{k\to\infty} \frac{1}{1 - \frac{x^{(k)} - x^{(k-1)}}{x^{(k-1)} - x^{(k-2)}}} = m \ .$$

Per recuperare la convergenza quadratica è necessario considerare il metodo

$$x^{(k+1)} = x^{(k)} - m\frac{f(x^{(k)})}{f'(x^{(k)})} \tag{2.2}$$

ove si sostituisce a m la stima $m^{(k)}$ data da

$$m^{(k)} = \frac{1}{1 - \frac{x^{(k)} - x^{(k-1)}}{x^{(k-1)} - x^{(k-2)}}} = \frac{x^{(k-1)} - x^{(k-2)}}{2x^{(k-1)} - x^{(k)} - x^{(k-2)}} \ .$$

2.6.2 Accelerazione di Aitken nel metodo di Newton

Alternativamente, è possibile usare l'estrapolazione di Aitken nel metodo di Newton. Dato $x^{(k)}$, l'algoritmo per calcolare $x^{(k+1)}$ è

$$\begin{aligned} x^{(k+1/3)} &= x^{(k)} - \frac{f(x^{(k)})}{f'(x^{(k)})} \\ x^{(k+2/3)} &= x^{(k+1/3)} - \frac{f(x^{(k+1/3)})}{f'(x^{(k+1/3)})} \\ x^{(k+1)} &= x^{(k+2/3)} - \frac{(x^{(k+2/3)} - x^{(k+1/3)})^2}{x^{(k+2/3)} - 2x^{(k+1/3)} + x^{(k)}} \end{aligned}$$

2.7 Errore assoluto ed errore relativo

Si consideri una successione $\{x^{(k)}\}_k$ convergente a x e una stima dell'errore assoluto $\left|x - x^{(k)}\right| \lesssim e^{(k)}$. Nel caso vi usi una tolleranza per l'errore assoluto $\mathrm{tol_a}$, il criterio d'arresto sarà basato sulla disuguaglianza

$$e^{(k)} \leq \mathrm{tol_a}$$

mentre, nel caso di una tolleranza per l'errore relativo $\mathrm{tol_r}$, sulla disuguaglianza

$$e^{(k)} \leq \mathrm{tol_r} \cdot \left|x^{(k)}\right| \ . \tag{2.3}$$

Se $x = 0$, potrebbe succedere che $x^{(k)}$ è molto piccolo o, addirittura, assumere proprio il valore 0. La disuguaglianza (2.3) non ha problemi ad essere implementata (a differenza di $e^{(k)}/\left|x^{(k)}\right| \leq \mathrm{tol_r}$), ma probabilmente non verrebbe mai soddisfatta. Si può usare allora un criterio d'arresto *misto*

$$e^{(k)} \leq \mathrm{tol_r} \cdot \left|x^{(k)}\right| + \mathrm{tol_a} \tag{2.4}$$

che diventa, in pratica, un criterio d'arresto per l'errore assoluto quando $x^{(k)}$ è sufficientemente piccolo.

2.8 Esercizi

1.? Si implementi il metodo di bisezione con una function `[x,iter,delta] = bisezione(fun,a,b,tol,maxit,varargin)`.

2. Si testi la function `bisezione` per il calcolo della radice n-esima di 2, con $n = 2, 3, 4$.

3. Si testi l'uso della function di GNU Octave `fsolve`.

4.? Si implementi il metodo di iterazione di punto fisso con una function `[x,iter,errest] = puntofisso(g,x0,tol,maxit,varargin)`, con il criterio d'arresto basato sullo scarto, ove `g` è la funzione di iterazione.

5.?! Si implementi il metodo di Newton con una function `[x,iter,errest] = newton(fun,fun1,x0,tol,maxit,varargin)` con il criterio d'arresto basato sullo scarto, ove `fun` e `fun1` sono rispettivamente la funzione di cui si vuole calcolare la radice e la sua derivata.

6.?! Si confrontino i metodi di bisezione e di Newton per il calcolo della radice quadrata di 2. Si calcoli l'ordine dei metodi mediante la formula (2.1). Si riporti, infine, in un grafico semilogaritmico nelle ordinate, la stima d'errore relativo e l'errore relativo ad ogni iterazione per entrambi i metodi.

7. Si confrontino i metodi di bisezione, di Newton e di iterazione di punto fisso (mediante opportuna funzione di iterazione) per il calcolo dell'unica radice di

$$1 = \frac{g}{2x^2}(\sinh(x) - \mathrm{sen}(x)) \ ,$$

con $g = 9.81$.

8.? Si individui un metodo di iterazione di punto fisso che converga linearmente alla radice positiva dell'equazione

$$0 = x^2 - \operatorname{sen}(\pi x)e^{-x}$$

e un metodo di iterazione di punto fisso che converga quadraticamente alla radice nulla.

9. Si confrontino i metodi di Newton, delle corde e delle secanti per il calcolo della radice quadrata di 2. Si calcoli l'ordine dei metodi mediante la formula (2.1). Si riporti, infine, in un grafico semilogaritmico nelle ordinate, la stima d'errore relativo e l'errore relativo ad ogni iterazione per i tre metodi.

10. Si implementi il metodo di Newton–Hörner e lo si testi per il calcolo delle radici del polinomio $x^4 - x^3 + x^2 - x + 1$. Si confrontino i risultati con le radici ottenute dal comando di GNU Octave `roots`.

11. Sia data la successione $\{x^{(k)}\}_k$ di punti definita nella Tabella 2.2. Si applichi l'accelerazione di Aitken per ottenere la successione $\{\hat{x}^{(k)}\}_k$ e nuovamente l'accelerazione di Aitken per ottenere la successione $\{\hat{\hat{x}}^{(k)}\}_k$. Si produca un grafico logaritmico nelle ordinate dell'errore rispetto al limite $\xi = 1$.

12. Si implementi il metodo Steffensen. Lo si confronti con il metodo di punto fisso per il calcolo dell'unica radice della funzione $f(x) = (x - 1)e^x$. Si testino, come funzioni di iterazione, $g_1(x) = \log(xe^x)$, $g_2(x) = (e^x + x)/(e^x + 1)$ e $g_3(x) = (x^2 - x + 1)/x$, calcolando preventivamente la derivata della funzione di iterazione valutata nella radice. Si usi come valore iniziale $x_0 = 2$, una tolleranza pari a 10^{-10} e un numero massimo di iterazioni pari a 100.

13. Si implementi il metodo di Newton adattivo per radici multiple. Lo si testi per il calcolo della radice multipla della funzione $f(x) = (x^2 - 1)^p \log x$, con $p = 2, 4, 6$, $x_0 = 0.8$, tolleranza pari a 10^{-10} e numero massimo di iterazioni pari a 200. Lo si confronti con il metodo di Newton e con il metodo (2.2), ove m è la molteplicità (in questo caso nota a priori) della radice. Lo si confronti infine con il metodo di Newton accelerato mediante estrapolazione di Aitken.

$x^{(1)} = 1.11920292202212$
$x^{(2)} = 1.02934289154332$
$x^{(3)} = 1.00772338703086$
$x^{(4)} = 1.00206543159514$
$x^{(5)} = 1.00055464176131$
$x^{(6)} = 1.00014910566816$
$x^{(7)} = 1.00004009631930$
$x^{(8)} = 1.00001078324501$
$x^{(9)} = 1.00000290003838$
$x^{(10)} = 1.00000077993879$
$x^{(11)} = 1.00000020975773$
$x^{(12)} = 1.00000005641253$
$x^{(13)} = 1.00000001517167$
$x^{(14)} = 1.00000000408029$
$x^{(15)} = 1.00000000109736$
$x^{(16)} = 1.00000000029513$
$x^{(17)} = 1.00000000007937$
$x^{(18)} = 1.00000000002135$

Tabella 2.2: Successione

Capitolo 3

Sistemi lineari

3.1 Considerazioni generali

Consideriamo il sistema lineare (non singolare)

$$Ax = b, \quad A \in \mathbb{R}^{n\times n},\ b \in \mathbb{R}^{n\times 1}\ .$$

In GNU Octave il comando

```
x = A\b;
```

calcola la soluzione del sistema lineare, usando un opportuno metodo diretto. Nel caso generale, viene eseguita una fattorizzazione LU con pivoting parziale seguita da una coppia di sostituzioni in avanti e all'indietro. Sarebbe ovviamente possibile usare il comando `inv` per il calcolo dell'inversa A^{-1} e calcolare x tramite il comando `x = inv(A)*b`. Tuttavia, tale metodo risulta essere svantaggioso, sia dal punto di vista computazionale che dal punto di vista dell'accuratezza. Infatti, per il calcolo dell'inversa, occorre risolvere gli n sistemi lineari

$$AX = I\ .$$

È necessaria pertanto una fattorizzazione LU e n coppie di sostituzioni in avanti e all'indietro. Dal punto di vista dell'accuratezza, si consideri il sistema lineare formato dall'unica equazione

$$49x = 49\ .$$

L'applicazione del metodo di eliminazione gaussiana porta al comando `x = 49/49`, mentre l'inversione di matrice porta al comando `x = (1/49)*49`. È facile verificare che solo nel primo caso si ottiene esattamente $x = 1$. La regola fondamentale è: *mai calcolare l'inversa di una matrice*, a meno che

non sia esplicitamente richiesta. Dovendo valutare un'espressione del tipo $A^{-1}B$, si possono considerare i sistemi lineari $AX = B$ da risolvere con il comando `X = A\B`. Analogamente, un'espressione del tipo BA^{-1} può essere calcolata con il comando `B/A`.

3.1.1 Precedenza degli operatori

Il seguente comando

```
x = alpha*A\b
```

risolve il sistema lineare $\alpha Ax = b$ e non $x = \alpha A^{-1}b$.

3.2 Operazioni vettoriali in GNU Octave

Riportiamo in questo paragrafo alcuni comandi, sotto forma di esempi, utili per la manipolazione di matrici in GNU Octave.

3.2.1 Operazioni su singole righe o colonne

Data una matrice A, le istruzioni

```
for i = 1:n
  A(i,j) = A(i,j)+1;
end
```

possono essere sostituite da

```
A(1:n,j) = A(1:n,j)+1;
```

È possibile scambiare l'ordine di righe o colonne. Per esempio, l'istruzione

```
A = B([1,3,2],:);
```

crea una matrice A che ha per prima riga la prima riga di B, per seconda la terza di B e per terza la seconda di B. Analogamente, l'istruzione

```
A = B(:,[1:3,5:6]);
```

crea una matrice A che ha per colonne le prime tre colonne di B e poi la quinta e la sesta di B.

È possibile concatenare matrici. Per esempio, l'istruzione

```
U = [A,b];
```

crea una matrice U formata da A e da un'ulteriore colonna b (ovviamente le dimensioni di A e b devono essere compatibili).

È possibile assegnare lo stesso valore ad una sottomatrice. Per esempio, l'istruzione

```
A(1:3,5:7) = 0;
```

pone a zero la sottomatrice formata dalle righe dalla prima alla terza e le colonne dalla quinta alla settima.

Un altro comando utile per la manipolazione di matrici è `max`. Infatti, nella forma

```
[m,i] = max(A(2:7,j));
```

restituisce l'elemento massimo m nella colonna j-esima di A (dalla seconda alla settima riga) e la posizione di tale elemento nel vettore $[a_{2j}, a_{3j}, \ldots, a_{7j}]^{\mathrm{T}}$.

Tutte le istruzioni vettoriali che sostituiscono cicli `for` sono da preferirsi dal punto di vista dell'efficienza computazionale in GNU Octave.

3.3 Sostituzioni all'indietro

L'algoritmo delle sostituzioni all'indietro per la soluzione di un sistema lineare $Ux = b$, con U matrice triangolare superiore, può essere scritto

```
function x = bsdummy(U,b)
%
% x = bsdummy(U,b)
%
n = length(b);
x = b;
x(n) = x(n)./U(n,n);
for i = n-1:-1:1
  for j = i+1:n
    x(i) = x(i)-U(i,j)*x(j);
  end
  x(i) = x(i)/U(i,i);
end
```

Per quanto visto nel paragrafo precedente, le istruzioni

```
for j = i+1:n
  x(i) = x(i)-U(i,j)*x(j);
end
x(i) = x(i)/U(i,i);
```

possono essere sostituite da

```
x(i) = (x(i)-U(i,i+1:n)*x(i+1:n))/U(i,i);
```

ove, in questo caso, l'operatore `*` è il prodotto scalare tra vettori. Generalizzando al caso di più termini noti $b_1, b_2, \ldots, b_m$, la function

```
function X = bs(U,B)
%
% X = bs(U,B)
%
n = length(B);
X = B;
X(n,:) = X(n,:)./U(n,n);
for i = n-1:-1:1
  X(i,:) = (X(i,:)-U(i,i+1:n)*X(i+1:n,:))/U(i,i);
end
```

risolve

$$UX = B$$

cioè

$$Ux_i = b_i, \quad i = 1, 2, \ldots, m \; .$$

Il comando \ di GNU Octave usa automaticamente l'algoritmo delle sostituzioni all'indietro (in avanti) quando la matrice è triangolare superiore (inferiore).

3.4 Sistemi rettangolari

3.4.1 Sistemi sovradeterminati

Consideriamo il caso di un sistema *sovradeterminato*

$$Ax = b, \quad A \in \mathbb{R}^{n \times m}, \; n > m$$

con rango$(A|b) >$ rango$(A) = m$. Non esiste *la* soluzione: possiamo però considerare il *residuo*

$$r(\bar{x}) = b - A\bar{x}$$

associato al vettore $\bar{x}$ e minimizzare la sua norma euclidea. Si ha

$$\begin{aligned} \|r(x)\|_2^2 = (b^{\mathrm{T}} - x^{\mathrm{T}}A^{\mathrm{T}})(b - Ax) = \|b\|_2^2 - b^{\mathrm{T}}Ax - x^{\mathrm{T}}A^{\mathrm{T}}b + x^{\mathrm{T}}A^{\mathrm{T}}Ax = \\ = \|b\|_2^2 - 2x^{\mathrm{T}}A^{\mathrm{T}}b + x^{\mathrm{T}}A^{\mathrm{T}}Ax \end{aligned}$$

il cui gradiente rispetto a x

$$\nabla_x \|r(x)\|_2^2 = -2A^{\mathrm{T}}b + 2A^{\mathrm{T}}Ax$$

si annulla quando

$$A^{\mathrm{T}}Ax = A^{\mathrm{T}}b$$

La soluzione *ai minimi quadrati* coincide con la soluzione del sistema *normale*

$$A^{\mathrm{T}}Ax = A^{\mathrm{T}}b\,. \tag{3.1}$$

La matrice $A^{\mathrm{T}}A$ risulta essere simmetrica e definita positiva. Pertanto, è teoricamente possibile utilizzare la fattorizzazione di Cholesky per risolvere il sistema. In maniera più efficiente, si può usare la fattorizzazione $A = QR$, con $Q \in \mathbb{R}^{n\times n}$ ortogonale e $R \in \mathbb{R}^{n\times m}$ triangolare superiore, senza il bisogno di calcolare esplicitamente la matrice $A^{\mathrm{T}}A$. Si ha infatti

$$A^{\mathrm{T}}Ax = A^{\mathrm{T}}b \Leftrightarrow R^{\mathrm{T}}Q^{\mathrm{T}}QRx = R^{\mathrm{T}}Q^{\mathrm{T}}b \Leftrightarrow R^{\mathrm{T}}(Rx - Q^{\mathrm{T}}b) = 0.$$

Poichè le ultime $n - m$ colonne di R^{T} sono nulle, basta considerare le prime m righe del sistema (quadrato) $Rx = Q^{\mathrm{T}}b$.

In ogni caso, il numero condizionamento della matrice $A^{\mathrm{T}}A$ potrebbe essere molto elevato.

Si può ricorrere alla decomposizione SVD. Sia

$$A = USV^{\mathrm{T}}, \quad U \in \mathbb{R}^{n\times n},\ S \in \mathbb{R}^{n\times m},\ V \in \mathbb{R}^{m\times m}$$

con U e V matrici ortogonali e

$$S = \begin{bmatrix} s_1 & 0 & \dots & \dots & 0 \\ 0 & s_2 & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & s_{m-1} & 0 \\ 0 & \dots & \dots & 0 & s_m \\ 0 & \dots & \dots & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \dots & \dots & \dots & 0 \end{bmatrix}$$

La soluzione ai minimi quadrati è quella che minimizza $\|Ax - b\|_2$. Per l'ortogonalità di U e V, si ha

$$\|Ax - b\|_2 = \left\|U^{\mathrm{T}}(AV^{\mathrm{T}} - b)\right\|_2 = \|Sz - d\|_2$$

con $y = V^{\mathrm{T}}x$ e $d = U^{\mathrm{T}}b$. Il minimo di $\|Sy - d\|_2$ si ha per $y_i = d_i/s_i$, $i = 1, \dots, m$, da cui poi si ricava $x = Vy$.

Il comando `x=A\b` di GNU Octave usa automaticamente la decomposizione SVD per la risoluzione ai minimi quadrati di un sistema sovradeterminato.

3.4.2 Sistemi sottodeterminati

Consideriamo il caso di un sistema *sottodeterminato*

$$Ax = b, \quad A \in \mathbb{R}^{n \times m},\ n < m$$

con rango$(A) = n$. Tra le infinite soluzioni, possiamo considerare quella di norma euclidea minima. Consideriamo ancora la decomposizione SVD, ove

$$S = \begin{bmatrix} s_1 & 0 & \dots & \dots & 0 & 0 & \dots & 0 \\ 0 & s_2 & 0 & \dots & 0 & \vdots & \dots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots & \vdots & \dots & \vdots \\ 0 & \dots & 0 & s_{n-1} & 0 & \vdots & \dots & \vdots \\ 0 & \dots & \dots & 0 & s_n & 0 & \dots & 0 \end{bmatrix}$$

Da $Ax = b$ si ricava $SV^{\mathrm{T}}x = U^{\mathrm{T}}b$ e, per l'ortogonalità di V, minimizzare la norma euclidea di x equivale a minimizzare la norma euclidea di $y = V^{\mathrm{T}}x$. Il sistema $Sy = d$, ove $d = U^{\mathrm{T}}b$, ammette come soluzione a norma euclide minima il vettore $y_i = d_i/s_i$, $i = 1, \dots, n$, $y_i = 0$, $i = n+1, \dots, m$, da cui poi si ricava $x = Vy$.

Il comando `x=A\b` di GNU Octave usa automaticamente la decomposizione SVD per trovare la soluzione di norma euclidea minima di un sistema sottodeterminato.

Nel caso di sistemi sottoderminati, un'altra soluzione interessante è quella maggiormente sparsa, cioè con il maggior numero di elementi uguali a zero. Si considera la fattorizzazione QR con pivoting $AE = QR$, ove E è una matrice di permutazione (dunque ortogonale). Si trova

$$A^{\mathrm{T}}Ax = A^{\mathrm{T}}b \Leftrightarrow ER^{\mathrm{T}}(Ry - Q^{\mathrm{T}}b) = 0$$

ove $y = E^{\mathrm{T}}x$. Il sistema $Ry = Q^{\mathrm{T}}b$ è sottodeterminato, dunque si possono scegliere $y_i = 0$, $i = n+1, \dots, m$. Poi si ricava $x = Ey$ che, essendo una permutazione di y, mantiene la stessa sparsità. Un'implementazione efficiente dal punto di vista dell'occupazione di memoria è la seguente:

```
[Q,R,E] = qr(A,0);
x = R(:,1:size(A,1))\(Q'*b);
x(size(A,2)) = 0;
x(E) = x;
```

Il comando `x=A\b` di Matlab® usa automaticamente la fattorizzazione QR per trovare la soluzione più sparsa di un sistema sottodeterminato.

3.5 Memorizzazione di matrici sparse

Sia A una matrice sparsa di ordine n e con m elementi diversi da zero (una matrice si dice sparsa se il numero dei suoi elementi diversi da zero è $m = \mathcal{O}(n)$ invece che $m = \mathcal{O}(n^2)$). Esistono molti formati di memorizzazione di matrici sparse. Quello usato da GNU Octave è il Compressed Column Storage (CCS). Consiste di tre array: un primo, `data`, di lunghezza m contenente gli elementi diversi da zero della matrice, ordinati prima per colonna e poi per riga; un secondo, `ridx`, di lunghezza m contenente gli indici di riga degli elementi di `data`; ed un terzo, `cidx`, di lunghezza $n+1$, il cui primo elemento è 0 e l'elemento $i+1$-esimo è il numero totale di elementi diversi da zero nelle prime i colonne della matrice. Per esempio, alla matrice

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 3 & 0 \\ 4 & 0 & 5 & 6 \\ 0 & 0 & 0 & 7 \end{bmatrix}$$

corrispondono i vettori

$$\begin{aligned} \texttt{data} &= [1, 4, 2, 3, 5, 6, 7] \\ \texttt{ridx} &= [1, 3, 2, 2, 3, 3, 4] \\ \texttt{cidx} &= [0, 2, 3, 5, 7] \end{aligned}$$

Dato un vettore x, il prodotto matrice-vettore $y = Ax$ è implementato dall'algoritmo in Tabella 3.1.

```
function y = ccsmv(data,ridx,cidx,x)
n = length(x);
y = zeros(size(x));
for j = 1:n
  for i = cidx(j)+1:cidx(j+1)
    y(ridx(i)) = y(ridx(i))+data(i)*x(j);
  end
end
```

Tabella 3.1: Prodotto matrice-vettore in formato CCS.

In GNU Octave, il formato CCS e l'implementazione del prodotto matrice-vettore sono automaticamente usati dalla function `sparse` e dall'operatore `*`, rispettivamente. Un utile comando per la creazione di matrici sparse è `spdiags`.

3.6 Metodi iterativi per sistemi lineari

I metodi iterativi per sistemi lineari si usano quando si vogliono sfruttare alcune proprietà della matrice (per esempio la sparsità), oppure quando la soluzione del sistema è richiesta a meno di una certa tolleranza. I fattori da invertire nel calcolo delle matrici di iterazione sono "facilmente" invertibili, nel senso che se ne possono calcolare le inverse senza ricorrere al metodo di eliminazione di Gauss o alle sue varianti (neanche implicitamente con i comandi \ o `inv`).

Potrebbe essere necessario un pivoting parziale preventivo per poter applicare i metodi iterativi, come si vede nel caso di una matrice

$$A = \begin{bmatrix} 0 & 1 & 4 & 1 \\ 1 & 4 & 1 & 0 \\ 4 & 1 & 0 & 0 \\ 0 & 0 & 1 & 4 \end{bmatrix}$$

Nel seguito useremo la decomposizione $L + U = D - A$.

3.6.1 Metodo di Jacobi

Nel metodo di Jacobi, la matrice di iterazione B_{J} è

$$B_{\mathrm{J}} = D^{-1}(L + U)$$

e dunque

$$x^{(k+1)} = B_{\mathrm{J}} x^{(k)} + D^{-1} b$$

La matrice diagonale D può essere costruita, direttamente in formato sparso, con il comando

```
D = spdiags(diag(A),0,size(A,1),size(A,2))
```

mentre la matrice D^{-1} con il comando

```
D1 = spdiags(1./diag(A),0,size(A,1),size(A,2))
```

3.6.2 Metodo di Gauss–Seidel

Nel metodo di Gauss–Seidel, la matrice di iterazione B_{GS} è

$$B_{\mathrm{GS}} = (D - L)^{-1} U$$

e dunque

$$x^{(k+1)} = B_{\mathrm{GS}} x^{(k)} + (D - L)^{-1} b$$

La matrice $D-L$ è triangolare inferiore: dunque la matrice di iterazione B_{GS} può essere calcolata risolvendo il sistema lineare

$$(D-L)B_{\mathrm{GS}} = U$$

per mezzo dell'algoritmo delle sostituzioni in avanti.

3.6.3 Metodo SOR

Il metodo SOR può essere scritto come

$$\begin{cases} Dx^{(k+1/2)} = Lx^{(k+1)} + Ux^{(k)} + b \\ x^{(k+1)} = \omega x^{(k+1/2)} + (1-\omega)x^{(k)} \end{cases}$$

da cui

$$\begin{aligned} x^{(k+1)} &= \omega D^{-1}Lx^{(k+1)} + \omega D^{-1}Ux^{(k)} + \omega D^{-1}b + (1-\omega)x^{(k)} \\ Dx^{(k+1)} &= \omega Lx^{(k+1)} + \omega Ux^{(k)} + \omega b + (1-\omega)Dx^{(k)} \\ (D-\omega L)x^{(k+1)} &= [(1-\omega)D + \omega U]x^{(k)} + \omega b \\ x^{(k+1)} &= (D-\omega L)^{-1}[(1-\omega)D + \omega U]x^{(k)} + (D-\omega L)^{-1}\omega b \end{aligned}$$

Dunque la matrice di iterazione B_{SOR}

$$B_{\mathrm{SOR}} = (D-\omega L)^{-1}[(1-\omega)D + \omega U]$$

può essere calcolata risolvendo il sistema lineare

$$(D-\omega L)B_{\mathrm{SOR}} = [(1-\omega)D + \omega U]$$

per mezzo dell'algoritmo delle sostituzioni in avanti.

3.7 Metodo di Newton per sistemi di equazioni non lineari

Consideriamo il sistema di equazioni non lineari

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

che può essere riscritto, in forma compatta,

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

Dato $x^{(1)}$, il metodo di Newton per calcolare $x^{(k+1)}$ è

$$\begin{aligned} J^{(k)} \delta x^{(k)} &= -f(x^{(k)}) \\ x^{(k+1)} &= x^{(k)} + \delta x^{(k)} \end{aligned} \tag{3.2}$$

ove $J^{(k)}$ è la matrice Jacobiana, definita da

$$J_{ij}^{(k)} = \frac{\partial f_i(x^{(k)})}{\partial x_j^{(k)}} \, . \tag{3.3}$$

Il criterio d'arresto (basato sull'errore assoluto) solitamente usato è

$$\left\| \delta x^{(k)} \right\| \leq \text{tol} \, .$$

3.7.1 Metodo di Newton modificato

Il metodo di Newton (3.2) richiede il calcolo della matrice Jacobiana e la sua "inversione" ad ogni passo k. Questo potrebbe essere troppo oneroso. Una strategia per ridurre il costo computazionale è usare sempre la stessa matrice Jacobiana $J^{(1)}$, oppure aggiornarla solo dopo un certo numero di iterazioni. In tal modo, per esempio, è possibile usare la stessa fattorizzazione $L^{(k)}U^{(k)}$ per più iterazioni successive.

3.8 Esercizi

1. Si implementi una function `[U,b1] = meg(A,b)` per il metodo di eliminazione gaussiana con pivoting per righe.

2. Si risolvano, mediante il metodo di eliminazione gaussiana e l'algoritmo delle sostituzioni all'indietro, i sistemi lineari

$$A_i x_i = b_i, \quad A_i = (A_1)^i, \quad i = 1, 2, 3$$

con

$$A_1 = \begin{bmatrix} 15 & 6 & 8 & 11 \\ 6 & 6 & 5 & 3 \\ 8 & 5 & 7 & 6 \\ 11 & 3 & 6 & 9 \end{bmatrix}$$

e b_i scelto in modo che la soluzione esatta sia $x_i = [1, 1, 1, 1]^{\mathrm{T}}$. Per ogni sistema lineare si calcoli l'errore in norma euclidea e il numero di condizionamento della matrice (con il comando `cond`). Si produca infine un grafico logaritmico-logaritmico che metta in evidenza la dipendenza lineare dell'errore dal numero di condizionamento.

3. Si implementi una function `X = fs(L,B)` che implementa l'algoritmo delle sostituzioni in avanti generalizzato al caso di più termini noti.

4. Sia A una matrice di ordine 5 generata in maniera casuale. Se ne calcoli l'inversa X, per mezzo del comando `lu` e degli algoritmi delle sostituzioni in avanti e all'indietro generalizzati al caso di più termini noti, secondo lo schema

$$\begin{gathered} AX = I \\ PAX = P \\ LUX = P \\ \begin{cases} LY = P \\ UX = Y \end{cases} \end{gathered}$$

5. Verificare che la risoluzione di sistemi sovra- e sottodeterminati con in comando \ di GNU Octave coincide con la descrizione data nei capitoli 3.4.1 e 3.4.2.

6. Implementare le functions `[data,ridx,cidx] = full2ccs(A)` e `[A] = ccs2full(data,ridx,cidx)`.

7. Data una matrice A memorizzata in formato CCS e un vettore x, implementare la seguente formula: $y = A^{\mathrm{T}}x$.

8. Si implementi il metodo di Jacobi con una function `[x,iter,err] = jacobi(A,b,x0,tol,maxit)` che costruisce la matrice di iterazione per mezzo del comando `spdiags`. Lo si testi per la soluzione del sistema lineare $Ax = b$ con `A = toeplitz([4,1,0,0,0,0])` e b scelto in modo che la soluzione esatta sia $x = [2, 2, 2, 2, 2, 2]^{\mathrm{T}}$.

9. Si risolva il sistema non lineare

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

con il metodo di Newton (3.2). Si usi una tolleranza pari a 10^{-6}, un numero massimo di iterazioni pari a 150 e un vettore iniziale $x^{(1)} = [1, 1]^{\mathrm{T}}$.

10. Si risolva lo stesso sistema non lineare usando sempre la matrice Jacobiana relativa al primo passo e aggiornando la matrice Jacobiana ogni r iterazioni, ove r è il più piccolo numero che permette di ottenere la soluzione con la tolleranza richiesta calcolando solo due volte la matrice Jacobiana. La risoluzione dei sistemi lineari deve avvenire con il calcolo della fattorizzazione LU solo quando necessaria.

Capitolo 4

Autovalori

4.1 Metodo delle potenze

Consideriamo il metodo delle potenze per il calcolo degli autovalori di una matrice A di ordine n ove x_1 è l'autovettore associato all'autovalore di modulo massimo. Sia $x^{(1)}$ un vettore arbitrario tale che

$$x^{(1)} = \sum_{i=1}^{n} \alpha_i x_i \ ,$$

con $\alpha_1 \neq 0$.

> La condizione su α_1, seppur teoricamente impossibile da assicurare essendo x_1 una delle incognite del problema, non è di fatto restrittiva. L'insorgere degli errori di arrotondamento comporta infatti la comparsa di una componente nella direzione di x_1, anche se questa non era presente nel vettore iniziale $x^{(1)}$.
>
> [Quarteroni e Saleri, *Introduzione al Calcolo Scientifico*, 2006]

Si consideri allora la matrice

$$A = \begin{bmatrix} -2 & 1 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & 1 & -2 \end{bmatrix}$$

generata dal comando `A = toeplitz([-2,1,0,0])`, i cui autovalori (calcolati dal comando `eig`), arrotondati alla quinta cifra decimale, sono -3.61903, -2.61803, -1.38197 e -0.38197. Il metodo delle potenze con vettore iniziale $x^{(1)} = [3, 4, 4, 3]^{\mathrm{T}}$ converge al *secondo* autovalore più grande in modulo. È buona norma, allora, prendere un vettore generato in maniera casuale (`x1 = rand(n,1)`) come vettore iniziale $x^{(1)}$.

4.1.1 Stima della convergenza

Consideriamo, per semplicità, una matrice di ordine 2, con autovalori $|\lambda_1| > |\lambda_2|$ e autovettori x_1 e x_2 normalizzati. Dato allora

$$x^{(k)} = \lambda_1^k \alpha_1 x_1 + \lambda_2^k \alpha_2 x_2 \ ,$$

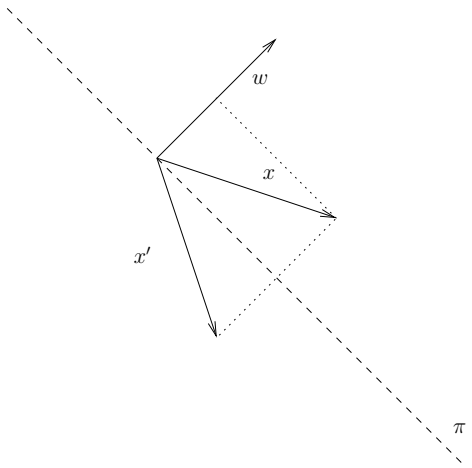
consideriamo la differenza tra il quoziente di Rayleigh e l'autovalore λ_1. Si ha

$$\frac{x^{(k)*} A x^{(k)}}{x^{(k)*} x^{(k)}} - \lambda_1 = \frac{\lambda_2^k \lambda_1^k (\lambda_2 - \lambda_1)\alpha_2 \left[\left(\frac{\lambda_2}{\lambda_1}\right)^k \alpha_2 + \alpha_1 x_1^* x_2 \right]}{\lambda_1^{2k} \left[\alpha_1^2 + \left(\frac{\lambda_2}{\lambda_1}\right)^{2k} \alpha_2^2 + 2 \left(\frac{\lambda_2}{\lambda_1}\right)^k \alpha_1 \alpha_2 x_1^* x_2 \right]}$$

da cui si deduce che la stima d'errore è proporzionale a $(\lambda_2/\lambda_1)^k$ (se la matrice A è simmetrica $(\lambda_2/\lambda_1)^{2k}$, perché, il tal caso, $x_1^* x_2 = 0$).

4.2 Matrici di Householder e deflazione

Dalla figura è chiaro che $x' - x = -2w/\left|w\right| \left(w^{\mathrm{T}} x/\left|w\right|\right)$.



Dunque la matrice

$$Q_w = I - 2\frac{w w^{\mathrm{T}}}{\|w\|_2^2}$$

è la matrice (simmetrica e ortogonale, cioè $Q_w = Q_w^{\mathrm{T}} = Q_w^{-1}$) che realizza la riflessione rispetto all'iperpiano π. Sia ora x un vettore di norma euclidea unitaria. Cerchiamo una matrice Q_w (cioè un vettore w) per cui $Q_w x = e_1$.

Deve essere allora

$$x_1 - 2\frac{w_1 \sum_{k=1}^{n} w_k x_k}{\sum_{k=1}^{n} w_k^2} = 1$$
$$x_j - 2\frac{w_j \sum_{k=1}^{n} w_k x_k}{\sum_{k=1}^{n} w_k^2} = 0, \quad j > 1$$

la cui soluzione è $w_1 = x_1 - 1$ e $w_j = x_j$, $j > 1$. Se x è un autovettore di una matrice A, allora $Ax = \lambda_1 x$, da cui $Q_w A Q_w^{-1} e_1 = Q_w \lambda_1 x = \lambda_1 e_1$. Dunque la matrice $B = Q_w A Q_w^{\mathrm{T}} = Q_w A Q_w$ (simile ad A) ha l'autovalore λ_1 corrispondente all'autovettore e_1 e dunque deve essere della forma

$$B = \begin{bmatrix} \lambda_1 & * & \dots & * \\ 0 & & & \\ \vdots & & A_1 & \\ 0 & & & \end{bmatrix} \tag{4.1}$$

Dunque A_1 ha per autovalori tutti gli autovalori di A, escluso λ_1.

4.3 Matrice companion

Per il calcolo degli zeri di un polinomio

$$a_1 x^{n-1} + a_2 x^{n-2} + \ldots + a_n$$

ci si può ricondurre al calcolo degli autovalori della matrice *companion*

$$F = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & 0 \\ 0 & & & 0 & 1 \\ -\frac{a_n}{a_1} & -\frac{a_{n-1}}{a_1} & \dots & -\frac{a_3}{a_1} & -\frac{a_2}{a_1} \end{bmatrix}$$

Si può dunque usare iterativamente il metodo delle potenze e la tecnica di deflazione.

4.4 Autovalori e autovettori in GNU Octave

Il comando principale in GNU Octave per il calcolo di autovalori e autovettori è `eig`. Nella forma `[V,D] = eig(A)`, restituisce la matrice V le cui colonne sono gli autovettori corrispondenti agli autovalori riportati sulla diagonale di D. Dunque, $AV = VD$.

4.5 Esercizi

1. Si implementi il metodo delle potenze per il calcolo dell'autovalore di modulo massimo e dell'autovettore corrispondente con una function `[lambda,y,iter,errest] = potenze (A,tol,maxit)`.

2. Si implementi una function `normestimate(A)` che stima la norma 2 di una matrice.

3. Si implementi una function `A1 = deflaziona(A,x)` che data una matrice A ed un suo autovettore x di norma unitaria, calcola la matrice A_1 in (4.1).

4. Sia A la matrice simmetrica di ordine N generata dal comando `A = toeplitz([-2,1,zeros(1,N-2)])*N^2`, con $N = 10$. Si calcoli, con il metodo delle potenze, l'autovalore di modulo massimo. Si consideri poi il risultato λ_1 ottenuto con una tolleranza pari a 10^{-14} come risultato di riferimento e si confronti, ad ogni iterazione k, il comportamento di $\left|\lambda_1^{(k)} - \lambda_1\right|$ con la stima $|\lambda_2/\lambda_1|^{2k}$, ove λ_2 è calcolato applicando la tecnica di deflazione alla matrice A.

5. Per la stessa matrice del punto precedente, si calcoli l'autovalore di modulo massimo prendendo come vettore iniziale $x^{(1)} = [1, 1, \ldots, 1]^{\mathrm{T}}$.

6. Si implementi il metodo delle potenze inverse con shift, mediante l'uso della fattorizzazione LU.

7. Si calcolino gli zeri del polinomio

$$13x^6 - 364x^5 + 2912x^4 - 9984x^3 + 16640x^2 - 13312x + 4096$$

   applicando il metodo delle potenze e la tecnica di deflazione alla matrice companion.

8. Si implementi l'algoritmo di iterazione QR per il calcolo degli autovalori di una matrice, usando il comando `qr` che realizza la fattorizzazione QR di una matrice. Si controlli la norma infinito della diagonale sotto la principale per terminare le iterazioni.

9. Si implementi il metodo di Givens basato su successioni di Sturm per il calcolo degli autovalori di matrici simmetriche.

10. Si implementi il metodo di Jacobi per il calcolo degli autovalori di matrici simmetriche.

Capitolo 5

Interpolazione ed approssimazione

5.1 Interpolazione polinomiale

Data una funzione $f\colon [a,b] \to \mathbb{R}$ e un insieme $\{x_i\}_{i=1}^n \subset [a,b]$, sia $L_{n-1}f(x)$ il polinomio di grado $n-1$ interpolatore di f nei punti x_i (cioè $L_{n-1}f(x_i) = f(x_i)$). Chiameremo i punti x_i *nodi di interpolazione* (o, più semplicemente, *nodi*). Un generico punto $\bar{x} \in [a,b]$ in cui si valuta $L_{n-1}f$ sarà chiamato *nodo target* (o, più semplicemente, *target*).

5.1.1 Nodi di Chebyshev

Si chiamano n *nodi di Chebyshev* gli zeri del polinomio di Chebyshev di grado n $T_n(x) = \cos(n \arccos(x))$. Dunque, $x_{j+1} = \cos\left(\frac{j\pi+\frac{\pi}{2}}{n}\right)$, $j = 0, \ldots, n-1$. Si chiamano n *nodi di Chebyshev estesi (o di Chebyshev–Lobatto)* i nodi $\bar{x}_{j+1} = \cos\left(\frac{j\pi}{n-1}\right)$, $j = 0, \ldots, n-1$. Tali nodi appartengono all'intervallo $[-1,1]$. I nodi di Chebyshev relativi ad un intervallo generico $[a,b]$ si ottengono semplicemente per traslazione e scalatura.

5.1.2 Interpolazione di Lagrange

Dato un insieme di n coppie di interpolazione $\{(x_i, y_i)\}_{i=1}^n$, il polinomio elementare di Lagrange i-esimo (di grado $n-1$) è

$$L_i(x) = \prod_{\substack{j=1 \\ j\neq i}}^{n} \frac{(x - x_j)}{x_i - x_j} = \\ = \frac{(x-x_1)\cdot(x-x_2)\cdot\ldots\cdot(x-x_{i-1})\cdot(x-x_{i+1})\cdot\ldots\cdot(x-x_n)}{(x_i-x_1)\cdot(x_i-x_2)\cdot\ldots\cdot(x_i-x_{i-1})\cdot(x_i-x_{i+1})\cdot\ldots\cdot(x_i-x_n)} .$$

L'algoritmo per il calcolo dei polinomi di Lagrange su vettori (colonna) target x è riportato in Tabella 5.1.

```
function y = lagrange(i,xbar,x)
%
% y = lagrange(i,xbar,x)
%
n = length(x);
m = length(xbar);
y = prod(repmat(xbar,1,n-1)-repmat(x([1:i-1,i+1:n]),m,1),2)/...
prod(x(i)-x([1:i-1,i+1:n]));
```

Tabella 5.1: Polinomio elementare di Lagrange.

Il polinomio di interpolazione si scrive dunque

$$p_{n-1}(x) = \sum_{i=1}^{n} y_i L_i(x) .$$

5.1.3 Sistema di Vandermonde

Dato il polinomio

$$p_{n-1}(x) = a_1 x^{n-1} + a_2 x^{n-2} + \ldots + a_{n-1}x + a_n$$

e n coppie di interpolazione $\{(x_i, y_i)\}_{i=1}^n$, il corrispondente sistema di Vandermonde si scrive

$$\begin{bmatrix} x_1^{n-1} & x_1^{n-2} & \ldots & x_1 & 1 \\ x_2^{n-1} & x_2^{n-2} & \ldots & x_2 & 1 \\ \vdots & \vdots & \ldots & \vdots & \vdots \\ x_{n-1}^{n-1} & x_{n-1}^{n-2} & \ldots & x_{n-1} & 1 \\ x_n^{n-1} & x_n^{n-2} & \ldots & x_n & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_{n-1} \\ a_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{n-1} \\ y_n \end{bmatrix} \tag{5.1}$$

L'implementazione dell'algoritmo per il calcolo della matrice di Vandermonde è riportata in Tabella 5.2. Alternativamente, si può usare la function di GNU Octave `vander`.

```
function V = vandermonde(x,varargin)
%
% V = vandermonde(x)
%
n = length(x);
if (nargin == 1)
  m = n;
else
  m = varargin{1};
end
V = repmat(x',1,m).^repmat([m-1:-1:0],n,1);
```

Tabella 5.2: Matrice di Vandermonde.

5.1.4 Interpolazione di Newton

Data una funzione f, definiamo le differenze divise nel seguente modo:

$$\begin{aligned} f[x] &= f(x) \\ f[x_1, x] &= \frac{f[x] - f[x_1]}{x - x_1} \\ \dots &= \dots \\ f[x_1, x_2, \dots, x_{k-1}, x_k, x] &= \frac{f[x_1, x_2, \dots, x_{k-1}, x] - f[x_1, x_2, \dots, x_{k-1}, x_k]}{x - x_k} \end{aligned}$$

L'algoritmo per il calcolo delle differenze divise è riportato in Tabella 5.3.

L'interpolazione nella forma di Newton si scrive dunque

$$\begin{aligned} L_0 f(x) &= d_1 \\ w &= (x - x_1) \\ \begin{cases} L_i f(x) = L_{i-1} f(x) + d_{i+1} w, \\ w = w \cdot (x - x_{i+1}) \end{cases} & \quad i = 1, \dots, n-1 \end{aligned}$$

ove

$$d_i = f[x_1, \dots, x_i] \, .$$

```
function d = diffdiv(x,y)
%
% d = diffdiv(x,y)
%
n = length(x);
for i = 1:n
  d(i) = y(i);
  for j = 1:i-1
    d(i) = (d(i)-d(j))/(x(i)-x(j));
  end
end
```

Tabella 5.3: Differenze divise.

Il calcolo delle differenze divise e la costruzione del polinomio di interpolazione possono essere fatti nel medesimo ciclo `for`.

Sfruttando la rappresentazione dell'errore

$$\begin{aligned} f(x) - L_{i-1}f(x) &= \left(\prod_{j=1}^{i}(x-x_j)\right) f[x_1,\ldots,x_i,x] \approx \\ &\approx \left(\prod_{j=1}^{i}(x-x_j)\right) f[x_1,\ldots,x_i,x_{i+1}] \end{aligned} \tag{5.2}$$

è possibile implementare un algoritmo per la formula di interpolazione di Newton adattativo, che si interrompa cioè non appena la stima dell'errore è più piccola di una tolleranza fissata.

Dato il polinomio interpolatore nella forma di Newton

$$p_{n-1}(x) = d_1 + d_2(x-x_1) + \ldots + d_n(x-x_1)\cdot\ldots\cdot(x-x_{n-1}) \ ,$$

si vede che le differenze divise soddisfano il sistema lineare

$$\begin{bmatrix} 1 & 0 & \ldots & \ldots & 0 \\ 1 & (x_2-x_1) & 0 & \ldots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 1 & (x_{n-1}-x_1) & \ldots & \prod_{j=1}^{n-2}(x_{n-1}-x_j) & 0 \\ 1 & (x_n-x_1) & \ldots & \ldots & \prod_{j=1}^{n-1}(x_n-x_j) \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_{n-1} \\ d_n \end{bmatrix} = \begin{bmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_{n-1}) \\ f(x_n) \end{bmatrix}$$

5.2 Interpolazione polinomiale a tratti

Data una funzione $f\colon [a,b] \to \mathbb{R}$ e un'insieme $\{x_i\}_{i=1}^n \subset [a,b]$ di nodi ordinati $(x_{i-1} < x_i)$, consideriamo l'interpolante polinomiale a tratti $L_{k-1}^c f$ di grado

$k-1$. Su ogni intervallo $[x_i, x_{i+1}]$ di lunghezza $h_i = x_{i+1} - x_i$ essa coincide con il polinomio di grado $k-1$

$$a_{i,1}(x-x_i)^{k-1} + a_{i,2}(x-x_i)^{k-2} + \ldots + a_{i,k-1}(x-x_i) + a_{i,k} \; . \tag{5.3}$$

Dunque, l'interpolante polinomiale a tratti è completamente nota una volta noti i nodi e i coefficienti di ogni polinomio.

5.2.1 Strutture in GNU Octave: `pp`

In GNU Octave è possibile definire delle *strutture*, cioè degli insiemi (non ordinati) di oggetti. Per esempio, le istruzioni

```
S.a = 1;
S.b = [1,2];
```

generano la struttura `S`

```
S =
{
  a =  1
  b =

     1   2

}
```

L'interpolazione polinomiale a tratti è definita mediante una struttura solitamente chiamata `pp` *(piecewise polynomial)*, che contiene gli oggetti `pp.x` (vettore colonna dei nodi), `pp.P` (matrice dei coefficienti), `pp.n` (numero di intervalli, cioè numero di nodi meno uno), `pp.k` (grado polinomiale più uno) e `pp.d` (numero di valori assunti dai polinomi). La matrice P ha dimensione $(n-1) \times k$ e, con riferimento a (5.3),

$$P_{ij} = a_{i,j} \; .$$

Nota una struttura `pp`, è possibile valutare il valore dell'interpolante in un generico target $\bar{x}$ con il comando `ppval(pp,xbar)`.

È possibile definire una struttura per l'interpolazione polinomiale a tratti attraverso il comando `mkpp(x,P)`.

5.2.2 Splines cubiche

Le splines cubiche sono implementate da GNU Octave con il comando `spline` che accetta in input il vettore dei nodi e il vettore dei valori e restituisce la struttura associata. La spline cubica costruita è nota come *not-a-knot*, ossia viene imposta la continuità della derivata terza (generalmente discontinua) nei nodi x_2 e x_{n-1}. Lo stesso comando permette di generare anche le splines *vincolate*: è sufficiente che il vettore dei valori abbia due elementi in più rispetto al vettore dei nodi. Il primo e l'ultimo valore verranno usati per imporre il valore della derivata alle estremità dell'intervallo. Se si usa un ulteriore vettore di input `xbar`, il comando restituisce il valore dell'interpolante sui nodi target `xbar`. Dunque, il comando

```
spline(x,y,xbar)
```

è equivalente ai comandi

```
pp = spline(x,y); ppval(pp,xbar)
```

Implementazione di splines cubiche in GNU Octave

Con le notazioni usate fino ad ora, si può costruire una spline cubica S a partire dalla sua derivata seconda nell'intervallo generico $[x_i, x_{i+1}]$

$$S''_{[x_i,x_{i+1}]}(x) = \frac{m_{i+1} - m_i}{h_i}(x - x_i) + m_i, \quad i = 1, \ldots, n-1 \tag{5.4}$$

ove $m_i = S''(x_i)$ sono incogniti. Integrando due volte la (5.4), si ottiene

$$S'_{[x_i,x_{i+1}]}(x) = \frac{m_{i+1} - m_i}{2h_i}(x - x_i)^2 + m_i(x - x_i) + a_i \tag{5.5}$$

$$S_{[x_i,x_{i+1}]}(x) = \frac{m_{i+1} - m_i}{6h_i}(x - x_i)^3 + \frac{m_i}{2}(x - x_i)^2 + a_i(x - x_i) + b_i \tag{5.6}$$

ove le costanti a_i e b_i sono da determinare. Innanzitutto, richiedendo la proprietà di interpolazione, cioè $S_{[x_i,x_{i+1}]}(x_j) = y_j$, $j = i, i+1$, si ottiene

$$\begin{aligned} b_i &= y_i, \\ a_i &= \frac{y_{i+1} - y_i}{h_i} - (m_{i+1} - m_i)\frac{h_i}{6} - m_i\frac{h_i}{2} = \\ &= \frac{y_{i+1} - y_i}{h_i} - m_{i+1}\frac{h_i}{6} - m_i\frac{h_i}{3} \end{aligned}$$

A questo punto, richiedendo la continuità della derivata prima nel nodo x_i, cioè $S'_{[x_{i-1},x_i]}(x_i) = S'_{[x_i,x_{i+1}]}(x_i)$ per $i = 2, \ldots, n-1$, si ottiene

$$\frac{h_{i-1}}{6}m_{i-1} + \frac{h_{i-1}+h_i}{3}m_i + \frac{h_i}{6}m_{i+1} = \frac{y_{i+1}-y_i}{h_i} - \frac{y_i - y_{i-1}}{h_{i-1}} \; . \tag{5.7}$$

Risulta chiaro che ci sono $n-2$ equazioni e n incognite m_i.

Splines cubiche *naturali* Si impone che il valore della derivata seconda agli estremi dell'intervallo sia 0. Da (5.4), si ricava dunque $m_1 = m_n = 0$. Il sistema lineare (5.7) diventa allora

$$\begin{bmatrix} 1 & 0 & \ldots & \ldots & \ldots & 0 \\ \frac{h_1}{6} & \frac{h_1+h_2}{3} & \frac{h_2}{6} & 0 & \ldots & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \ldots & 0 & \frac{h_{n-2}}{6} & \frac{h_{n-2}+h_{n-1}}{3} & \frac{h_{n-1}}{6} \\ 0 & \ldots & \ldots & \ldots & 0 & 1 \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ \vdots \\ \vdots \\ m_{n-1} \\ m_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ \vdots \\ d_{n-1} \\ d_n \end{bmatrix}$$

con $d_1 = d_n = 0$ e $d_i = \frac{y_{i+1}-y_i}{h_i} - \frac{y_i - y_{i-1}}{h_{i-1}}$, $i = 2, \ldots, n-1$. L'algoritmo per il calcolo della struttura associata ad una spline cubica naturale è riportato in Tabella 5.4.

Splines cubiche *vincolate* Si impongono due valori y'_1 e y'_n per la derivata $S'(x_1)$ e $S'(x_n)$, rispettivamente. Da (5.5) si ricava dunque

$$a_1 = y'_1$$
$$\frac{m_n - m_{n-1}}{2h_{n-1}}(x_n - x_{n-1})^2 + m_{n-1}(x_n - x_{n-1}) + a_{n-1} = y'_n$$

da cui

$$\frac{h_1}{3}m_1 + \frac{h_1}{6}m_2 = \frac{y_2 - y_1}{h_1} - y'_1$$
$$\frac{h_{n-1}}{6}m_{n-1} + \frac{h_{n-1}}{3}m_n = y'_n - \frac{y_n - y_{n-1}}{h_{n-1}}$$

Il sistema lineare da risolvere diventa dunque

$$\begin{bmatrix} \frac{h_1}{3} & \frac{h_1}{6} & 0 & \ldots & \ldots & 0 \\ \frac{h_1}{6} & \frac{h_1+h_2}{3} & \frac{h_2}{6} & 0 & \ldots & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \ldots & 0 & \frac{h_{n-2}}{6} & \frac{h_{n-2}+h_{n-1}}{3} & \frac{h_{n-1}}{6} \\ 0 & \ldots & \ldots & 0 & \frac{h_{n-1}}{6} & \frac{h_{n-1}}{3} \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ \vdots \\ \vdots \\ m_{n-1} \\ m_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ \vdots \\ d_{n-1} \\ d_n \end{bmatrix}$$

```
function pp = splinenaturale(x,y,varargin)
%
% function pp = splinenaturale(x,y)
%
n = length(x);
x = x(:);
y = y(:);
h = x(2:n)-x(1:n-1);
d1 = 0;
dn = 0;
diagup = [0;0;h(2:n-1)/6];
diagdown = [h(1:n-2)/6;0;0];
diag0 = [1;(h(1:n-2)+h(2:n-1))/3;1];
rhs = [d1;(y(3:n)-y(2:n-1))./h(2:n-1)-...
(y(2:n-1)-y(1:n-2))./h(1:n-2);dn];
S = spdiags([diagdown,diag0,diagup],[-1,0,1],n,n);
m = S\rhs;
a = (y(2:n)-y(1:n-1))./h(1:n-1)-...
h(1:n-1).*(m(2:n)/6+m(1:n-1)/3);
b = y(1:n-1);
P = [(m(2:n)-m(1:n-1))./(6*h),m(1:n-1)/2,a,b];
pp = mkpp(x,P);
if (nargin == 3)
  pp = ppval(pp,varargin{1});
end
```

Tabella 5.4: Spline cubica naturale.

con $d_1 = \frac{y_2 - y_1}{h_1} - y_1'$ e $d_n = y_n' - \frac{y_n - y_{n-1}}{h_{n-1}}$.

Splines cubiche *periodiche* Si impone la periodicità della derivata prima e seconda, cioè $S'(x_1) = S'(x_n)$ e $S''(x_1) = S''(x_n)$. Da (5.4) e (5.5) si ricava dunque

$$\begin{aligned} m_1 &= m_n \\ a_1 &= \frac{m_n - m_{n-1}}{2} h_{n-1} + m_{n-1} h_{n-1} + a_{n-1} \end{aligned}$$

da cui

$$\begin{aligned} m_1 - m_n &= 0 \\ \frac{h_1}{3}m_1 + \frac{h_1}{6}m_2 + \frac{h_{n-1}}{6}m_{n-1} + \frac{h_{n-1}}{3}m_n &= \frac{y_2 - y_1}{h_1} + \\ &\quad - \frac{y_n - y_{n-1}}{h_{n-1}} \end{aligned}$$

Il sistema lineare da risolvere diventa dunque

$$\begin{bmatrix} 1 & 0 & \dots & \dots & \dots & 0 & -1 \\ \frac{h_1}{6} & \frac{h_1+h_2}{3} & \frac{h_2}{6} & 0 & \dots & \dots & 0 \\ 0 & \frac{h_2}{6} & \frac{h_2+h_3}{3} & \frac{h_3}{6} & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & \dots & 0 & \frac{h_{n-2}}{6} & \frac{h_{n-2}+h_{n-1}}{3} & \frac{h_{n-1}}{6} \\ \frac{h_1}{3} & \frac{h_1}{6} & 0 & \dots & 0 & \frac{h_{n-1}}{6} & \frac{h_{n-1}}{3} \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ \vdots \\ \vdots \\ m_{n-1} \\ m_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ \vdots \\ d_{n-1} \\ d_n \end{bmatrix}$$

con $d_1 = 0$ e $d_n = \frac{y_2 - y_1}{h_1} - \frac{y_n - y_{n-1}}{h_{n-1}}$.

Splines cubiche *not-a-knot* Si impone la continuità della derivata terza in x_2 e x_{n-1}. Derivando (5.4) si ricava dunque

$$\begin{aligned} \frac{m_2 - m_1}{h_1} &= \frac{m_3 - m_2}{h_2} \\ \frac{m_{n-1} - m_{n-2}}{h_{n-2}} &= \frac{m_n - m_{n-1}}{h_{n-1}} \end{aligned}$$

da cui

$$\begin{aligned} \frac{1}{h_1}m_1 - \left(\frac{1}{h_1} + \frac{1}{h_2}\right)m_2 + \frac{1}{h_2}m_3 &= 0 \\ \frac{1}{h_{n-2}}m_{n-2} - \left(\frac{1}{h_{n-2}} + \frac{1}{h_{n-1}}\right)m_{n-1} + \frac{1}{h_{n-1}}m_n &= 0 \end{aligned}$$

Il sistema lineare da risolvere diventa dunque

$$\begin{bmatrix} \frac{1}{h_1} & -\frac{1}{h_1} - \frac{1}{h_2} & \frac{1}{h_2} & 0 & \dots & 0 \\ \frac{h_1}{6} & \frac{h_1+h_2}{3} & \frac{h_2}{6} & 0 & \dots & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \frac{h_{n-2}}{6} & \frac{h_{n-2}+h_{n-1}}{3} & \frac{h_{n-1}}{6} \\ 0 & \dots & 0 & \frac{1}{h_{n-2}} & -\frac{1}{h_{n-2}} - \frac{1}{h_{n-1}} & \frac{1}{h_{n-1}} \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ \vdots \\ \vdots \\ m_{n-1} \\ m_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ \vdots \\ d_{n-1} \\ d_n \end{bmatrix}$$

con $d_1 = d_n = 0$.

Derivazione delle splines cubiche

Attraverso la struttura `pp` è possibile calcolare la derivata prima di una spline. Infatti, i valori `pp.x`, `pp.n` e `pp.d` sono gli stessi. Il valore `pp.k` deve essere diminuito di uno. La matrice dei coefficienti `pp.P` diventa

$$\begin{bmatrix} 3a_{1,1} & 2a_{1,2} & a_{1,3} \\ 3a_{2,1} & 2a_{2,2} & a_{2,3} \\ \vdots & \vdots & \vdots \\ 3a_{n,1} & 2a_{n,2} & a_{n,3} \end{bmatrix}$$

Può essere utile utilizzare il comando `unmkpp`.

5.2.3 Rappresentazione dell'errore

Supponiamo di usare un metodo di interpolazione polinomiale a tratti di grado $k-1$ in un intervallo $[a, b]$ e consideriamo due diverse discretizzazioni, rispettivamente con n_1 e n_2 nodi, con intervalli di lunghezza media $h_1 = (b-a)/(n_1-1)$ e $h_2 = (b-a)/(n_2-1)$. Gli errori di approssimazione saranno verosimilmente $\mathrm{err}_1 = Ch_1^k$ e $\mathrm{err}_2 = Ch_2^k$. Si ha dunque

$$\frac{\mathrm{err}_2}{\mathrm{err}_1} = \left(\frac{h_2}{h_1}\right)^k$$

da cui

$$\log(\mathrm{err}_2) - \log(\mathrm{err}_1) = k(\log h_2 - \log h_1) = -k(\log(n_2 - 1) - \log(n_1 - 1)) \ .$$

Dunque, rappresentando in un grafico logaritmico-logaritmico l'errore in dipendenza dal numero di nodi, la pendenza della retta corrisponde al grado di approssimazione del metodo, cambiato di segno.

5.2.4 Compressione di dati

Supponiamo di avere un insieme di coppie di nodi/valori $\{(x_i, y_i)\}_{i=1}^N$ con N molto grande e di non conoscere la funzione che associa il valore al nodo corrispondente. Ci poniamo il problema di *comprimere* i dati, ossia memorizzare il minor numero di coefficienti pur mantenendo un sufficiente grado di accuratezza. Una prima idea potrebbe essere quella di selezionare alcuni dei nodi, diciamo n, e di costruire la spline cubica su quei nodi. Il costo di memorizzazione, oltre ai nodi, sarebbe dunque pari a $4(n-1)$. Rimarrebbe il problema di scegliere i nodi da memorizzare, visto che non si suppone siano equispaziati.

Si potrebbe ridurre il costo di memorizzazione (a n) usando un unico polinomio interpolatore: rimarrebbe il problema della scelta dei nodi e, probabilmente, si aggiungerebbe un problema di mal condizionamento sempre dovuto alla scelta dei nodi.

Un'idea che combina le tecniche discusse è la seguente: si usa una interpolazione a tratti (anche lineare) per ricostruire i valori della funzione sconosciuta in corrispondenza di n nodi di Chebyshev. Si usa poi un unico polinomio interpolatore su quei nodi. Il rapporto di compressione è $2N/n$, considerando che non è necessario memorizzare i nodi di Chebyshev, ma solo i coefficienti del polinomio interpolatore (e trascurando i due estremi dell'intervallo).

5.2.5 Il comando `find`

Supponiamo di voler implementare una function per la valutazione della funzione

$$f(x) = \begin{cases} \operatorname{sen}(x) & \text{se } x \geq 0 \\ -\operatorname{sen}(x) & \text{se } x < 0 \end{cases}$$

Una implementazione potrebbe essere

```
function y = f(x)
if (x >= 0)
   y =  sin(x);
else
   y = -sin(x);
end
```

La forte limitazione di questa implementazione è l'impossibilità di avere un risultato corretto quando `x` è un vettore anziché uno scalare. Si può dunque ricorrere al comando `find`. Per esempio, dato il vettore `x=[-1,0,1]`, il comando

```
find(x >= 0)
```

restituisce il vettore `[2,3]`. Cioè, gli elementi 2 e 3 del vettore `x` soddisfano la relazione `x>=0`. Allora, una function che implementa correttamente la funzione data per un input vettoriale generico è

```
function y = f(x)
y = zeros(size(x));
index1 = find(x >= 0);
y(index1) = sin(x(index1));
index2 = find(x < 0);
y(index2) = -sin(x(index2));
```

5.3 Approssimazione polinomiale

5.3.1 Sistema normale per i minimi quadrati

Consideriamo un polinomio di grado $m-1$ che "interpoli" i dati $\{(x_i, y_i)\}_{i=1}^n$, $n > m$. Il sistema di Vandermonde da risolvere è

$$\begin{bmatrix} x_1^{m-1} & x_1^{m-2} & \dots & x_1 & 1 \\ x_2^{m-1} & x_2^{m-2} & \dots & x_2 & 1 \\ x_3^{m-1} & x_3^{m-2} & \dots & x_3 & 1 \\ \vdots & \vdots & \dots & \vdots & \vdots \\ x_{n-2}^{m-1} & x_{n-2}^{m-2} & \dots & x_{n-2} & 1 \\ x_{n-1}^{m-1} & x_{n-1}^{m-2} & \dots & x_{n-1} & 1 \\ x_n^{m-1} & x_n^{m-2} & \dots & x_n & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_{m-2} \\ a_{m-1} \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n-2} \\ y_{n-1} \\ y_n \end{bmatrix}$$

A meno di configurazioni particolari dei dati, il sistema risulta essere sovradeterminato. Se ne può cercare la soluzione ai minimi quadrati, secondo quanto descritto al Capitolo 3.4.1. Si noti che la norma euclidea al quadrato del residuo del sistema lineare è

$$\sum_{i=1}^{n} \left[y_i - \left(a_1 x_i^{m-1} + a_2 x_i^{m-2} + \ldots + a_{m-2} x_i + a_{m-1} \right) \right]^2$$

da cui il nome di "minimi quadrati".

5.3.2 Il comando `polyfit`

Nel caso di approssimazione ai minimi quadrati con polinomi, è possibile usare il comando `polyfit`. Dati due vettori di nodi e valori `x` e `y` di lunghezza `n`, il comando `polyfit(x,y,m-1)`, $m < n$, restituisce i coefficienti del polinomio di grado $m-1$ approssimante nel senso dei minimi quadrati. Ovviamente, se $m = n$, il polinomio è l'unico interpolante.

5.4 Esercizi

1. Si considerino i seguenti comandi per generare n nodi di Chebyshev–Lobatto:

   ```
   cos([0:n-1]*pi/(n-1))
   cos(linspace(0,pi,n))
   ```

   e si dica quale è preferibile.

2. Si implementi una function `y = lagrange(i,x,nodi)` che valuta il polinomio di Lagrange i-esimo sul vettore target x.

3. Si implementi una function `y = interplagrange(nodi,valori,x)` per la formula di interpolazione nella forma di Lagrange.

4. Si testi l'interpolazione nella forma di Lagrange della funzione di Runge nell'intervallo $[-5, 5]$ su nodi equispaziati. Si prendano rispettivamente $n = 11, 21, 31, 41, 51$ nodi di interpolazione e si valuti l'interpolante su $5(n-1)+1$ nodi target equispaziati. Si producano delle figure mettendo in evidenza i nodi di interpolazione, la funzione di Runge e l'interpolante.

5. Si implementi una function `y = chebyshev(n)` per il calcolo dei nodi di Chebyshev nell'intervallo $[-1, 1]$.

6. Si ripeta l'esercizio 4 usando nodi di interpolazione di Chebyshev anziché nodi equispaziati.

7. Si implementi una function `V = vandermonde(nodi)` per il calcolo della matrice di Vandermonde definita in (5.1).

8. Si implementi una function `y = interpvandermonde(nodi,valori,x)` per la formula di interpolazione mediante matrice di Vandermonde. Si spieghino i risultati ottenuti.

9. Si ripeta l'esercizio 4, usando la formula di interpolazione mediante matrice di Vandermonde. Si usi il metodo di Hörner implementato nel Capitolo 2, Tabella 2.1 per la valutazione del polinomio.

10. Si implementi una function `y = interpnewton(nodi,valori,x)` per il calcolo del polinomio di interpolazione nella forma di Newton.

11. Si ripeta l'esercizio 4, usando la formula di interpolazione di Newton.

12. Si modifichi l'implementazione dell'interpolazione nella forma di Newton, in modo da prevedere come parametro opzionale di input la tolleranza per l'errore (in norma infinito) di interpolazione, stimato come in (5.2). Nel caso la tolleranza non sia raggiunta, l'algoritmo si interrompe all'ultimo nodo di interpolazione. La function deve fornire in uscita il numero di iterazioni e la stima dell'errore.

13. Si considerino $n = 21$ nodi di interpolazione equispaziati nell'intervallo $[-5, 5]$. Si interpoli in forma di Newton la funzione $y = \cos(x)$ sull'insieme di nodi target $\{-2, 0, 1\}$ per diverse tolleranze e, successivamente, sull'insieme di nodi target $\{-\pi, \pi\}$. Si spieghino i risultati ottenuti.

14. Si calcolino i numeri di condizionamento della matrice di Vandermonde (5.1) e della matrice dei coefficienti dell'interpolazione di Newton, da ordine 2 a 20 (considerando nodi equispaziati in $[-1, 1]$) e se ne produca un grafico semilogaritmico nelle ordinate. Si discutano i risultati.

15. Si implementi una function `pp = lintrat(x,y)` per l'interpolazione lineare a tratti.

16. Si verifichi, mediante un grafico logaritmico-logaritmico, il grado di approssimazione (errore in norma infinito) delle splines cubiche naturali per la funzione di Runge nell'intervallo $[-5, 5]$. Si considerino un numero di nodi di interpolazione equispaziati nell'intervallo $[-5, 5]$ da $n = 90$ a $n = 150$ e 1000 nodi target equispaziati.

17. Si ripeta l'esercizio precedente con l'interpolazione lineare a tratti.

18. Data la struttura associata ad una spline cubica, si ricavi la corrispondente struttura per la derivata seconda.

19. Si ripeta l'esercizio 16, confrontando però la derivata seconda della funzione di Runge e la derivata seconda della spline cubica not-a-knot associata.

20. Si considerino le coppie $\{(x_i, y_i)\}$ ove gli x_i sono $N = 1001$ nodi equispaziati nell'intervallo $[0, 2\pi]$ e $y_i = \text{sen}(x_i)$. Mediante il procedimento descritto in § 5.2.4 (interpolazione lineare a tratti e interpolazione su nodi di Chebyshev estesi), si determini il minimo grado n necessario per comprimere i dati con un errore in norma infinito inferiore a 10^{-5}. Si determini poi l'errore in corrispondenza del rapporto di compressione 286. Infine, si giustifichi la stagnazione dell'errore di approssimazione per grado di interpolazione maggiore di 10.

21. Si implementino due functions per la risoluzione del sistema normale (3.1) tramite fattorizzazione QR e tramite decomposizione SVD. Si consulti l'help dei comandi `qr` e `svd`.

22. Si considerino il vettore `x = linspace(0,3*pi,101)` e il vettore `y = sin(x)+(rand(size(x))-0.5)/5`. Le coppie (x_i, y_i) possono essere interpretate come dati affetti da rumore. Si costruiscano i polinomi di

approssimazione ai minimi quadrati, con grado opportuno. Si determini sperimentalmente quale è il grado polinomiale che meglio ricostruisce la funzione seno.

Capitolo 6

FFT

6.1 Funzioni ortonormali

Sia $[a, b]$ un intervallo di $\mathbb{R}$, $N > 0$ pari e fissato e $M = N/2$. Consideriamo, per ogni $m \in \mathbb{Z}$,

$$\phi_m(x) = \frac{e^{i(m-1-M)2\pi(x-a)/(b-a)}}{\sqrt{b-a}} \ .$$

Allora,

$$\int_a^b \phi_n(x)\overline{\phi_m(x)}\mathrm{d}x = \delta_{nm} \ . \tag{6.1}$$

Infatti, se $n = m$ allora $\phi_n(x)\overline{\phi_m(x)} = 1/(b-a)$, altrimenti

$$\phi_n(x)\overline{\phi_m(x)} = \frac{e^{i2\pi(n-m)(x-a)/(b-a)}}{\sqrt{b-a}}$$

e quindi

$$\int_a^b \phi_n(x)\overline{\phi_m(x)}\mathrm{d}x = \int_0^1 \frac{e^{i2\pi(n-m)y}}{\sqrt{b-a}}(b-a)\mathrm{d}y = 0 \ ,$$

poiché l'integrale delle funzioni sen e cos in un intervallo multiplo del loro periodo è nullo. La famiglia di funzioni $\{\phi_m(x)\}_m$ si dice *ortonormali* nell'intervallo $[a, b]$.

Un risultato utile è il seguente

$$\sum_{n=1}^{N} e^{i(n-1)2\pi(k-m)/N} = N\delta_{km}, \quad -\frac{N}{2} \le k, m \le \frac{N}{2} - 1 \tag{6.2}$$

È ovvio per $k = m$; altrimenti

$$\sum_{n=1}^{N} e^{i(n-1)2\pi(k-m)/N} = \sum_{n=0}^{N-1} \left(e^{i2\pi(k-m)/N}\right)^n = \\ = \frac{1 - e^{i2\pi(k-m)}}{1 - e^{i2\pi(k-m)/N}} = \frac{1 - \cos(2\pi(k-m))}{1 - e^{i2\pi(k-m)/N}} = 0 \, .$$

6.2 Trasformata di Fourier discreta

Sia f una funzione da $[a, b]$ a $\mathbb{C}$ *periodica* ($f(a) = f(b)$). Supponiamo che f si possa scrivere (ciò è vero, per esempio, per funzioni di classe C^1) come

$$f(x) = \sum_{n=-\infty}^{\infty} f_n \phi_n(x), \quad f_n \in \mathbb{C} \, . \tag{6.3}$$

Fissato $m \in \mathbb{Z}$, moltiplicando entrambi i membri per $\overline{\phi_m(x)}$ e integrando nell'intervallo $[a, b]$, usando (6.1) si ottiene

$$\int_a^b f(x)\overline{\phi_m(x)}\mathrm{d}x = \int_a^b \left(\sum_{n=-\infty}^{\infty} f_n \phi_n(x)\overline{\phi_m(x)} \right) \mathrm{d}x = \\ = \sum_{n=-\infty}^{\infty} f_n \int_a^b \phi_n(x)\overline{\phi_m(x)}\mathrm{d}x = f_m \, . \tag{6.4}$$

Dunque, abbiamo un'espressione esplicita per i coefficienti f_m. Riportiamo per comodità la formula di quadratura trapezoidale a $N+1$ nodi equispaziati $x_k = (b-a)y_k + a$, ove $y_k = (k-1)/N$, $k = 1, \ldots, N+1$ per funzioni periodiche:

$$\int_a^b f(x)\mathrm{d}x \approx \frac{b-a}{2N} \left(f(x_1) + 2\sum_{k=2}^{N} f(x_k) + f(x_{N+1}) \right) = \frac{b-a}{N} \sum_{k=1}^{N} f(x_k)$$

Applicando tale formula di quadratura ai coefficienti (6.4) si ottiene

$$f_m = \int_a^b f(x) \frac{e^{-i(m-1-N/2)2\pi(x-a)/(b-a)}}{\sqrt{b-a}} \mathrm{d}x = \\ = \sqrt{b-a} \int_0^1 f((b-a)y + a) e^{-i(m-1)2\pi y} e^{iN\pi y} \mathrm{d}y \approx \\ \approx \frac{\sqrt{b-a}}{N} \boxed{\sum_{k=1}^{N} \left(f(x_k) e^{iN\pi y_k} \right) e^{-i(m-1)2\pi y_k}} = \hat{f}_m$$

ove $x = (b-a)y + a$.

La funzione

$$\tilde{f}(x) = \sum_{n=1}^{N} \hat{f}_n \phi_n(x) = \sum_{m=-M}^{M-1} \hat{f}_{m+1+M} \phi_{m+1+M}(x) =$$
$$= \sum_{m=-M}^{M-1} \hat{f}_{m+1+M} e^{im2\pi(x-a)/(b-a)}$$

è un polinomio trigonometrico che *approssima* $f(x)$ ed è *interpolante* nei nodi x_k. Si ha infatti, usando (6.2),

$$\begin{aligned}
\tilde{f}(x_k) &= \sum_{n=1}^{N} \hat{f}_n \phi_n(x_k) = \\
&= \sum_{n=1}^{N} \left(\frac{\sqrt{b-a}}{N} \sum_{m=1}^{N} \left(f(x_m) e^{iN\pi y_m} \right) e^{-i(n-1)2\pi y_m} \right) \frac{e^{i(n-1-N/2)2\pi(x_k-a)/(b-a)}}{\sqrt{b-a}} = \\
&= \frac{1}{N} \sum_{m=1}^{N} f(x_m) e^{iN\pi(m-1)/N} e^{-iN\pi(k-1)/N} \sum_{n=1}^{N} e^{-i(n-1)2\pi(m-1)/N} e^{i(n-1)2\pi(k-1)/N} = \\
&= \frac{1}{N} \sum_{m=1}^{N} f(x_m) e^{i(m-k)\pi} \sum_{n=1}^{N} e^{i(n-1)2\pi(k-m)/N} = \frac{1}{N} f(x_k) N = f(x_k) \, .
\end{aligned}$$

La trasformazione

$$[f(x_1), f(x_2), \ldots, f(x_N)]^{\mathrm{T}} \to [\hat{f}_1, \hat{f}_2, \ldots, \hat{f}_N]^{\mathrm{T}}$$

si chiama *trasformata di Fourier discreta* di f e $\hat{f}_1, \ldots, \hat{f}_N$ *coefficienti di Fourier* di f. Il vettore $N \cdot [\hat{f}_1, \hat{f}_2, \ldots, \hat{f}_N]^{\mathrm{T}} / \sqrt{b-a}$ può essere scritto come prodotto matrice-vettore $F[f(x_1)e^{iN\pi y_1}, f(x_2)e^{iN\pi y_2}, \ldots, f(x_N)e^{iN\pi y_N}]^{\mathrm{T}}$, ove

$$F = (f_{mn}), \quad f_{mn} = e^{-i(m-1)2\pi y_n} \, .$$

Alternativamente, si può usare la Fast Fourier Transform (FFT). Il comando `fft` applicato al vettore $[f(x_1)e^{iN\pi y_1}, f(x_2)e^{iN\pi y_2}, \ldots, f(x_N)e^{iN\pi y_N}]^{\mathrm{T}}$ produce il vettore $N \cdot [\hat{f}_1, \hat{f}_2, \ldots, \hat{f}_N]^{\mathrm{T}} / \sqrt{b-a}$, così come il comando `fftshift` applicato al risultato del comando `fft` applicato al vettore $[f(x_1), f(x_2), \ldots, f(x_N)]$.

Dati i coefficienti $\hat{u}_n$, $n = 1, \ldots, N$, si può considerare la funzione (periodica)

$$\sum_{n=1}^{N} \hat{u}_n \phi_n(x) \, .$$

La valutazione nei nodi x_k, $k = 1, \ldots, N$, porge

$$\hat{\hat{u}}_k = \sum_{n=1}^{N} \hat{u}_n \phi_n(x_k) = \sum_{n=1}^{N} \hat{u}_n \frac{e^{i(n-1-N/2)2\pi(x_k-a)/(b-a)}}{\sqrt{b-a}} =$$
$$= \frac{N}{\sqrt{b-a}} \boxed{\frac{1}{N} \left(\sum_{n=1}^{N} \hat{u}_n e^{i(n-1)2\pi y_k} \right)} e^{-iN\pi y_k} .$$

La trasformazione

$$[\hat{u}_1, \hat{u}_2, \ldots, \hat{u}_N]^{\mathrm{T}} \to [\hat{\hat{u}}_1, \hat{\hat{u}}_2, \ldots, \hat{\hat{u}}_N]^{\mathrm{T}}$$

si chiama *anti-trasformata di Fourier discreta*. Se gli $\hat{u}_n$ sono i coefficienti di Fourier di una funzione $u(x)$, la proprietà di interpolazione comporta $\hat{\hat{u}}_k = u(x_k)$.

Il vettore $\sqrt{b-a} \cdot [\hat{\hat{u}}_1 e^{iN\pi y_1}, \hat{\hat{u}}_2 e^{iN\pi y_2}, \ldots, \hat{\hat{u}}_N e^{iN\pi y_N}]^{\mathrm{T}}/N$ può essere scritto come prodotto matrice-vettore $F^*[\hat{u}_1, \hat{u}_2, \ldots, \hat{u}_N]^{\mathrm{T}}/N$. Alternativamente, il comando `ifft` applicato al vettore $[\hat{u}_1, \hat{u}_2, \ldots, \hat{u}_N]$ produce il vettore $\sqrt{b-a} \cdot [\hat{\hat{u}}_1 e^{iN\pi y_1}, \hat{\hat{u}}_2 e^{iN\pi y_2}, \ldots, \hat{\hat{u}}_N e^{iN\pi y_N}]/N$, mentre, se applicato al risultato del comando `fftshift` applicato al vettore $[\hat{u}_1, \hat{u}_2, \ldots, \hat{u}_N]$, produce il vettore $\sqrt{b-a} \cdot [\hat{\hat{u}}_1, \hat{\hat{u}}_2, \ldots, \hat{\hat{u}}_N]/N$.

6.2.1 Costi computazionali e stabilità

La Fast Fourier Transform di un vettore di lunghezza N ha costo $\mathcal{O}(N \log N)$, mentre il prodotto matrice-vettore $\mathcal{O}(N^2)$. Tali costi sono però asintotici e nascondono i fattori costanti. Inoltre, GNU Octave può far uso di implementazioni ottimizzate di algebra lineare (come, ad esempio, le librerie ATLAS). In pratica, dunque, esiste un N_0 sotto il quale conviene, dal punto di vista del costo computazionale, usare il prodotto matrice-vettore e sopra il quale la FFT.

Per quanto riguarda l'accuratezza, in generale la FFT è più precisa del prodotto matrice vettore. Poiché la trasformata di Fourier discreta comporta l'uso di aritmetica complessa (anche se la funzione da trasformare è reale), la sequenza trasformata/anti-trasformata potrebbe introdurre una quantità immaginaria spuria che può essere eliminata con il comando `real`.

Anche per la trasformata di Fourier vi possono essere problemi di stabilità simili al fenomeno di Runge (qui chiamato *fenomeno di Gibbs*). Una tecnica per "smussare" (in inglese "to smooth") eventuali oscillazioni, consiste nel moltiplicare opportunamente i coefficienti di Fourier $\hat{u}_n$ per opportuni termini

σ_n che decadono in n, per esempio

$$\sigma_n = \left(1 - \frac{|-M+n-1|}{M+1}\right) .$$

Questa scelta corrisponde alle *medie di Cesáro*.

6.2.2 Valutazione di un polinomio trigonometrico

Supponiamo di conoscere i coefficienti $\hat{u}_n$, $n = 1, \ldots, N$ e di voler valutare la funzione

$$u(x) = \sum_{n=1}^{N} \hat{u}_n \phi_n(x) .$$

su un insieme di nodi target x_j equispaziati, $x_j = (j-1)/J$, $j = 1, \ldots, J$, $J > N$, J pari. Si possono introdurre dei coefficienti fittizi $\hat{U}_J$

$$\begin{aligned}
&\hat{U}_j = 0 && j = 1, \ldots, \frac{J-N}{2} \\
&\hat{U}_j = \hat{u}_{j-\frac{J-N}{2}} && j = \frac{J-N}{2}+1, \ldots, N - \frac{J-N}{2} \\
&\hat{U}_j = 0 && j = N - \frac{J-N}{2} + 1, \ldots, J
\end{aligned}$$

Si avrà

$$\hat{\hat{u}}_j = \sum_{n=1}^{N} \hat{u}_n \phi_n(x_j) = \sum_{n=1}^{J} \hat{U}_n \frac{e^{i(n-1-J/2)2\pi(x_j-a)/(b-a)}}{\sqrt{b-a}} =$$

$$= \frac{J}{\sqrt{b-a}} \boxed{\frac{1}{J}\left(\sum_{n=1}^{J} \hat{U}_n e^{i(n-1)2\pi y_j}\right)} e^{-iJ\pi y_j} .$$

6.3 Norme

Data una funzione $f(x)$ e due diverse discretizzazioni (su nodi equispaziati o meno) $[\tilde{f}_1, \tilde{f}_2, \ldots, \tilde{f}_N] \approx [f(\tilde{x}_1), f(\tilde{x}_2), \ldots, f(\tilde{x}_N)]$ e $[\hat{f}_1, \hat{f}_2, \ldots, \hat{f}_M] \approx [f(\hat{x}_1), f(\hat{x}_2), \ldots, f(\hat{x}_M)]$, con $N \neq M$, non ha senso confrontare gli errori in norma 2 rispetto a f, $\left\|[f(\tilde{x}_1) - \tilde{f}_1, f(\tilde{x}_2) - \tilde{f}_2, \ldots, f(\tilde{x}_N) - \tilde{f}_N]\right\|_2$ e $\left\|[f(\hat{x}_1) - \hat{f}_1, f(\hat{x}_2) - \hat{f}_2, \ldots, f(\hat{x}_M) - \hat{f}_M]\right\|_2$. Si preferisce usare la norma infinito, oppure la norma $\|[f_1, f_2, \ldots, f_N]\|_{L^2} = \sqrt{(b-a)/N}\,\|f\|_2$, che risulta

essere una approssimazione mediante quadratura trapezoidale della norma

$$\|f\|_{L^2} = \left(\int_a^b |f(x)|^2 \, \mathrm{d}x \right)^{1/2} . \tag{6.5}$$

Spesso si tralascia il fattore $\sqrt{b-a}$.

6.4 Esercizi

1. Si confrontino i tempi di calcolo per le tre implementazioni della trasformata di Fourier proposte, con $N = 2, 4, 8, \ldots, 2048$. Si confrontino gli errori, in norma L^2, tra i valori di una funzione in N nodi equispaziati e della sua approssimazione mediante trasformata di Fourier.

2. Si consideri la funzione periodica $f(x) = \mathrm{sen}(x) + \mathrm{sen}(5x)$ nell'intervallo $[0, 2\pi]$ e la sua ricostruzione mediante polinomio trigonometrico a partire da $N + 1 = 7$ nodi equispaziati. Si produca un grafico della funzione e della sua approssimazione su $J + 1 = 129$ nodi equispaziati, mettendo in evidenza i nodi di interpolazione. Si ripeta con $N + 1 = 9$, 11 e 13 nodi equispaziati.

3. Si vuole approssimare con polinomi trigonometrici la funzione a gradino implementata in Tabella 6.1. Si considerino le ricostruzioni con $N = 8$, 16, 32, 64, 128, 256 e 512 su $J + 1 = 1025$ nodi equispaziati, confrontando l'errore in norma infinito e l'errore in norma L^2. Si ripeta l'esercizio usando le medie di Cesáro.

```
function y = stepfun(x,a,b)
y = zeros(size(x));
index1 = find(x <= a+(b-a)/4);
y(index1) = -1;
index2 = find(a+(b-a)/4 < x & x < a+3*(b-a)/4);
y(index2) = 1;
index3 = find(a+3*(b-a)/4 <= x);
y(index3) = -1;
```

Tabella 6.1: Funzione a gradino.

Capitolo 7

Quadratura

7.1 Formula dei trapezi composita

Sia $f\colon [a,b] \to \mathbb{R}$ una funzione di classe $\mathcal{C}^2$ di cui vogliano calcolare l'integrale

$$I(f) = \int_a^b f(x)\mathrm{d}x \ .$$

Usando un passo $h = (b-a)/(n-1)$, la formula dei trapezi composita si scrive

$$I(f) = I_h(f) - \frac{b-a}{12}h^2 f''(\xi_h) \tag{7.1}$$

ove

$$I_h(f) = \frac{h}{2}\left(f(a) + 2\sum_{j=1}^{n-2} f(a+jh) + f(b)\right) \tag{7.2}$$

e ξ_h è un punto opportuno in (a,b). Consideriamo ora la formula dei trapezi composita con passo $h/2$ ($2n-1$ nodi). Si ha

$$I(f) = I_{h/2}(f) - \frac{b-a}{12}\frac{h^2}{4}f''(\xi_{h/2}) \ . \tag{7.3}$$

ove

$$I_{h/2} = \frac{h}{4}\left(f(a) + 2\sum_{j=1}^{2n-3} f(a+jh/2) + f(b)\right) \ . \tag{7.4}$$

Confrontando le due formule (7.1) e (7.3) e *supponendo* $f''(\xi_h) \approx f''(\xi_{h/2})$, si ottiene

$$I_h(f) - I_{h/2}(f) \approx \frac{b-a}{12}\frac{3h^2}{4}f''(\xi_{h/2})$$

da cui, ricavando $f''(\xi_{h/2})$ da (7.3), si ha

$$I(f) \approx I_{h/2}(f) - \left(\frac{I_h(f) - I_{h/2}(f)}{3} \right) \tag{7.5}$$

e, ricavando $f''(\xi_h)$ da (7.1), si ha

$$I(f) \approx I_h(f) - 4 \left(\frac{I_h(f) - I_{h/2}(f)}{3} \right) . \tag{7.6}$$

Visto che $I_{h/2}$ deve essere calcolato in ogni caso, conviene usare l'espressione $\left| I_h(f) - I_{h/2}(f) \right| /3$ come stima *a posteriori* dell'errore nell'approssimazione $I(f) \approx I_{h/2}(f)$. Nel caso invece in cui si conosca o si riesca a stimare il massimo della derivata seconda (per esempio, calcolandone il massimo su un insieme abbastanza numeroso di nodi) l'espressione

$$\frac{b-a}{12} h^2 \left\| f'' \right\|_\infty$$

è una stima *a priori* dell'errore $|I(f) - I_h(f)|$, che permette di determinare il numero minimo di nodi (equispaziati) da usare per soddisfare una tolleranza prefissata.

7.1.1 Dettagli implementativi

1. Confrontando le formule (7.2) e (7.4), si vede che la metà circa delle valutazioni della funzione integranda sono comuni. Si può allora riscrivere
$$I_{h/2} = \frac{I_h + h \sum_{j=1}^{n-1} f(a + h/2 + (j-1)h)}{2} .$$
Inoltre, la stima dell'errore (7.5) si basa sull'assunzione che le derivate seconde calcolate in due punti in generale diversi coincidano. Si può usare una stima più conservativa, moltiplicando per esempio la stima per 3/2. Un'implementazione della formula dei trapezi composita è riportata in Tabella 7.1.

2. Dopo aver applicato la formula dei trapezi composita a n nodi e a $2n-1$ nodi, si ha a disposizione una stima dell'errore. Si può introdurre allora una semplice strategia adattiva: se la tolleranza richiesta non è soddisfatta, si applica nuovamente la formula dei trapezi composita a $2n-1$ nodi e a $2(2n-1)-1$ nodi. Tale procedura può essere implementata molto facilmente tramite chiamate ricorsive, come in Tabella 7.2.

```
function [I,stimaerr,x] = trapadatt1(func,a,b,tol,varargin)
if (nargin == 4)
  n = 2;
else
  n = varargin{1};
end
h = (b-a)/(n-1);
x = linspace(a,b,n)';
weight = h/2*[1,2*ones(1,length(x)-2),1];
Iold = weight*feval(func,x);
x = linspace(a,b,2*n-1)';
weight = h*ones(1,n-1);
Inew = weight*feval(func,x(2:2:end-1));
I = (Inew+Iold)/2;
stimaerr = abs(I-Iold)/2;
if (stimaerr > tol)
  warning('Impossibile raggiungere la tolleranza richiesta')
  warning('con il numero di nodi di quadratura consentito.')
end
```

Tabella 7.1: Formula dei trapezi composita.

3. Una strategia adattiva più efficace è la seguente: se la formula trapezoidale composita a n nodi (e a $2n-1$ nodi) nell'intervallo $[a,b]$ non ha raggiunto la tolleranza richiesta tol, si può applicare la formula trapezoidale composita a n nodi all'intervallo $[a,(a+b)/2]$ e all'intervallo $[(a+b)/2,b]$, con tolleranza richiesta tol/2. La procedura può essere implementata con chiamate ricorsive, come in Tabella 7.3.

   Si osservi che se

$$\left| \int_a^{\frac{a+b}{2}} f(x)\mathrm{d}x - I_1(f) \right| \leq \frac{\mathrm{tol}}{2}$$

$$\left| \int_{\frac{a+b}{2}}^b f(x)\mathrm{d}x - I_2(f) \right| \leq \frac{\mathrm{tol}}{2}$$

```
function [I,stimaerr,x] = trapadatt2(func,a,b,tol,varargin)
if (nargin == 4)
  n = 3;
else
  n = varargin{1};
end
h = (b-a)/(n-1);
x = linspace(a,b,n)';
weight = h/2*[1,2*ones(1,length(x)-2),1];
Iold = weight*feval(func,x);
x = linspace(a,b,2*n-1)';
weight = h*ones(1,n-1);
Inew = weight*feval(func,x(2:2:end-1));
I = (Inew+Iold)/2;
stimaerr = abs(I-Iold)/2;
if (stimaerr > tol)
  [I,stimaerr,x] = trapadatt2(func,a,b,tol,2*n-1);
end
```

Tabella 7.2: Formula dei trapezi composita adattiva a passo costante.

allora

$$\left|\int_a^b f(x)\mathrm{d}x - (I_1(f) + I_2(f))\right| =$$
$$= \left|\int_a^{\frac{a+b}{2}} f(x)\mathrm{d}x - I_1(f) + \int_{\frac{a+b}{2}}^b f(x)\mathrm{d}x - I_2(f)\right| \leq$$
$$\leq \left|\int_a^{\frac{a+b}{2}} f(x)\mathrm{d}x - I_1(f)\right| + \left|\int_{\frac{a+b}{2}}^b f(x)\mathrm{d}x - I_2(f)\right| \leq \mathrm{tol}$$

Tale procedura darà luogo ad una formula dei trapezi composita a passo variabile.

4. La stima (7.5) può fallire clamorosamente. Si consideri, per esempio, $f(x) = x\,\mathrm{sen}(2x)$. Si ha

$$\int_0^{2\pi} f(x)\mathrm{d}x = -\pi\ .$$

Preso $h = 2\pi$, allora $I_h(f) = 0$, così come $I_{h/2}(f) = 0$ e $I_{h/4}(f) = 0$.

```
function [I,stimaerr,x] = trapadatt3(func,a,b,tol,varargin)
if (nargin == 4)
  n = 3;
else
  n = varargin{1};
end
h = (b-a)/(n-1);
x = linspace(a,b,n)';
weight = h/2*[1,2*ones(1,length(x)-2),1];
Iold = weight*feval(func,x);
x = linspace(a,b,2*n-1)';
weight = h*ones(1,n-1);
Inew = weight*feval(func,x(2:2:end-1));
I = (Inew+Iold)/2;
stimaerr = abs(I-Iold)/2;
if (stimaerr > tol)
  [Il,stimaerrl,xl] = trapadatt3(func,a,a+(b-a)/2,tol/2,n);
  [Ir,stimaerrr,xr] = trapadatt3(func,a+(b-a)/2,b,tol/2,n);
  I = Il+Ir;
  stimaerr = stimaerrl+stimaerrr;
  x = union(xl,xr);
end
```

Tabella 7.3: Formula dei trapezi composita adattiva a passo variabile.

7.2 Formula di Simpson composita

Tutto quanto detto per la formula dei trapezi composita si può ripetere con la formula di Simpson composita con passo $h = (b-a)/(2n-2)$ ($2n-1$ nodi)

$$I(f) = I_h(f) - \frac{b-a}{180} h^4 f^{(4)}(\xi_h)$$

ove

$$I_h(f) = \frac{h}{3}\left(f(a) + 4\sum_{j=1}^{n-1} f(a+(2j-1)h) + 2\sum_{j=1}^{n-2} f(a+2jh) + f(b) \right) .$$

7.3 Formula di Gauss–Legendre

Si consideri la formula di Gauss–Legendre per l'approssimazione dell'integrale

$$\int_{-1}^{1} f(x)\mathrm{d}x \approx \sum_{j=1}^{n} w_j f(x_j) \ .$$

Come noto, il grado di esattezza di tale formula è $2n-1$. Consideriamo allora il seguente integrale

$$\int_a^b p_{2n-1}(y)\mathrm{d}y = \sum_{j=1}^{n} v_j p_{2n-1}(y_j)$$

ove $p_{2n-1}(y)$ è un polinomio di grado $2n-1$ e i v_j e i y_j sono pesi e nodi di quadratura da determinare. Si ha

$$\begin{aligned}\sum_{j=1}^{n} v_j p_{2n-1}(y_j) &= \int_a^b p_{2n-1}(y)\mathrm{d}y = \\ &= \int_{-1}^{1} p_{2n-1}(a+(b-a)(1+x)/2)\frac{b-a}{2}\mathrm{d}x = \\ &= \frac{b-a}{2}\int_{-1}^{1} q_{2n-1}(x)\mathrm{d}x = \frac{b-a}{2}\sum_{j=1}^{n} w_j q_{2n-1}(x_j)\end{aligned}$$

con $x = 2(y-a)/(b-a) - 1$, da cui

$$\begin{aligned}y_j &= a + (b-a)(1+x_j)/2 \\ v_j &= (b-a)w_j/2\end{aligned}$$

I nodi e i pesi di quadratura per l'intervallo standard $[-1,1]$ si possono ottenere con le functions `r_jacobi` e `gauss` di W. Gautschi riportate in Tabella 7.4 e 7.5. L'uso di tali functions è mostrato in Tabella 7.6. Le functions necessarie per le altre formule di quadratura gaussiana si trovano all'indirizzo `http://www.cs.purdue.edu/archives/2002/wxg/codes/OPQ.html`.

7.4 Esercizi

1. Si consideri la formula dei trapezi composita per l'approssimazione dell'integrale
$$\int_{-\pi}^{\pi} x^2 e^{-x^2}\mathrm{d}x \ .$$

```
% R_JACOBI Recurrence coefficients for monic Jacobi polynomials.
%
%    ab=R_JACOBI(n,a,b) generates the first n recurrence
%    coefficients for monic Jacobi polynomials with parameters
%    a and b. These are orthogonal on [-1,1] relative to the
%    weight function w(t)=(1-t)^a(1+t)^b. The n alpha-coefficients
%    are stored in the first column, the n beta-coefficients in
%    the second column, of the nx2 array ab. The call ab=
%    R_JACOBI(n,a) is the same as ab=R_JACOBI(n,a,a) and
%    ab=R_JACOBI(n) the same as ab=R_JACOBI(n,0,0).
%
%    Supplied by Dirk Laurie, 6-22-1998; edited by Walter
%    Gautschi, 4-4-2002.
%
function ab=r_jacobi(N,a,b)
if nargin<2, a=0; end;  if nargin<3, b=a; end
if((N<=0)|(a<=-1)|(b<=-1)) error('parameter(s) out of range'), end
nu=(b-a)/(a+b+2);
mu=2^(a+b+1)*gamma(a+1)*gamma(b+1)/gamma(a+b+2);
if N==1, ab=[nu mu]; return, end
N=N-1; n=1:N; nab=2*n+a+b;
A=[nu (b^2-a^2)*ones(1,N)./(nab.*(nab+2))];
n=2:N; nab=nab(n);
B1=4*(a+1)*(b+1)/((a+b+2)^2*(a+b+3));
B=4*(n+a).*(n+b).*n.*(n+a+b)./((nab.^2).*(nab+1).*(nab-1));
ab=[A' [mu; B1; B']];
```

Tabella 7.4: Formula di quadratura di Gauss–Legendre.

Si mostri, con un grafico logaritmico-logaritmico, l'andamento dell'errore calcolato rispetto alla soluzione di riferimento data dal comando `quad` di GNU Octave e della stima dell'errore data da (7.5) in funzione del numero di nodi di quadratura n, con $n = 10, 11, \ldots, 200$.

2. Si utilizzi la stima a priori dell'errore per determinare il numero minimo di nodi necessari per approssimare l'integrale

$$\int_{-3}^{3} \frac{\operatorname{sen} x}{1 + e^x} \mathrm{d}x$$

con la formula dei trapezi composita con una precisione pari a 10^{-2}.

```
% GAUSS Gauss quadrature rule.
%
%    Given a weight function w encoded by the nx2 array ab of the
%    first n recurrence coefficients for the associated orthogonal
%    polynomials, the first column of ab containing the n alpha-
%    coefficients and the second column the n beta-coefficients,
%    the call xw=GAUSS(n,ab) generates the nodes and weights xw of
%    the n-point Gauss quadrature rule for the weight function w.
%    The nodes, in increasing order, are stored in the first
%    column, the n corresponding weights in the second column, of
%    the nx2 array xw.
%
function xw=gauss(N,ab)
N0=size(ab,1); if N0<N, error('input array ab too short'), end
J=zeros(N);
for n=1:N, J(n,n)=ab(n,1); end
for n=2:N
  J(n,n-1)=sqrt(ab(n,2));
  J(n-1,n)=J(n,n-1);
end
[V,D]=eig(J);
[D,I]=sort(diag(D));
V=V(:,I);
xw=[D ab(1,2)*V(1,:)'.^2];
```

Tabella 7.5: Formula di quadratura di Gauss–Legendre.

```
clear all
I = quad(@integranda1,-1,1);
N = 12;
ab = r_jacobi(N);
xw = gauss(N,ab);
nodes = xw(:,1);
weights = xw(:,2);
Igauss = weights'*integranda1(nodes);
error = abs(I-Igauss)
```

Tabella 7.6: Formula di quadratura di Gauss–Legendre.

Lo si confronti con il numero di nodi dato dall'uso della formula dei trapezi composita adattiva a passo costante.

3. Si implementi la formula di Simpson composita.

4. Si ripeta l'esercizio 1 per la formula di Simpson composita.

5. Si implementino le formule di Simpson composite adattive, a passo costante e a passo variabile.

6. Si ripeta l'esercizio 1 per la formula di Gauss–Legendre.

Appendice A

Soluzioni di alcuni esercizi

In questo capitolo sono presenti le risoluzioni (cioè i codici in GNU Octave, di proposito *non* commentati) di alcuni degli esercizi proposti.

```
clear all
close all
x = -10;
s(1) = 1;
for i=1:80
  s(i+1) = s(i)+x^i/factorial(i);
end
semilogy(abs(s-exp(x))/exp(x))
```

Tabella A.1: Serie esponenziale, Capitolo 1, Esercizio 3.

```
clear all
close all
I(1) = 1/e;
Irif(1) = 1/e;
N = 30;
for n = 2:N
  I(n) = 1-n*I(n-1);
  Irif(n) = quad(@(x)integranda(x,n),0,1);
end
Iback(2*N) = 0;
for n = 2*N-1:-1:1
  Iback(n) = (1-Iback(n+1))/(n+1);
end
semilogy([1:30],abs(I),'*',[1:30],Iback(1:30),'o',[1:30],Irif)
legend('successione','successione indietro','quadratura')
```

Tabella A.2: Successione ricorrente, Capitolo 1, Esercizio 1.3.

```
function [x,iter,stimaerr] = bisezione(fun,a,b,tol,maxit,varargin)
%
% [x,iter,stimaerr] = bisezione(fun,a,b,tol,maxit,varargin)
%
iter = 0;
x = a;
stimaerr = (b-a);
while ((stimaerr > tol) & (iter < maxit))
  stimaerr = stimaerr/2;
  x = a+stimaerr;
  iter = iter+1;
  if (feval(fun,a,varargin{:})*feval(fun,x,varargin{:}) < 0)
    b = x;
  else
    a = x;
  end
end
if (stimaerr > tol)
  warning('Impossibile raggiungere la tolleranza richiesta')
  warning('entro il numero massimo di iterazioni consentito.')
end
```

Tabella A.3: Metodo di bisezione, Capitolo 2, Esercizio 2.

```
function [x,iter,errest] = puntofisso(fun,x0,tol,maxit,varargin)
%
% [x,iter,errest] = puntofisso(fun,x0,tol,maxit,varargin)
%
x = x0;
iter = 1;
errest = tol+1;
while ((errest > tol) & (iter < maxit))
  x = feval(fun,x0,varargin{:});
  iter = iter+1;
  errest = abs(x-x0);
  x0 = x;
end
if (errest > tol)
  warning('Impossibile raggiungere la tolleranza richiesta')
  warning('entro il numero massimo di iterazioni consentito.')
end
```

Tabella A.4: Metodo di punto fisso, Capitolo 2, Esercizio 4.

```
function [x,iter,stimaerr] = newton(fun,fun1,x,tol,maxit,varargin)
%
% [x,iter,stimaerr] = newton(fun,fun1,x,tol,maxit,varargin)
%
iter = 0;
stimaerr = -feval(fun,x,varargin{:})/feval(fun1,x,varargin{:});
while ((abs(stimaerr) > tol) & (iter < maxit))
  x = x+stimaerr;
  iter = iter+1;
  stimaerr = -feval(fun,x,varargin{:})/feval(fun1,x,varargin{:});
end
stimaerr = abs(stimaerr);
if (stimaerr > tol)
  warning('Impossibile raggiungere la tolleranza richiesta')
  warning('entro il numero massimo di iterazioni consentito.')
end
```

Tabella A.5: Metodo di Newton, Capitolo 2, Esercizio 5.

```
clear all
close all
irange = [1:20];
tol = 0;
for i = irange
   [xb(i),iter,errestb(i)] = bisezione(@root2,1,2,tol,i);
   [xn(i),iter,errestn(i)] = newton(@root2,@root2der,2,tol,i);
   if (i>1)
     ordineb = log(abs(xb(i)-sqrt(2)))/log(abs(xb(i-1)-sqrt(2)))
     ordinen = log(abs(xn(i)-sqrt(2)))/log(abs(xn(i-1)-sqrt(2)))
   end
end
semilogy(irange,abs(sqrt(2)-xb)/sqrt(2),'r',...
 irange,errestb/sqrt(2),'r*',...
 irange,abs(sqrt(2)-xn)/sqrt(2),'b',...
 irange,errestn/sqrt(2),'b*');
legend('errore bisezione','stima bisezione','errore Newton','stima Newton')
```

Tabella A.6: Radice di 2, Capitolo 2, Esercizio 6.

```
function x = aitken(x0,x1,x2)
%
% function x = aitken(x0,x1,x2)
%
x = x2-(x2-x1).^2./(x2-2*x1+x0);
```

Tabella A.7: Estrapolazione di Aitken, Capitolo 2, Esercizio 11.

```
clear all
close all
x = [1.11920292202212e+00;...
     1.02934289154332e+00;...
     1.00772338703086e+00;...
     1.00206543159514e+00;...
     1.00055464176131e+00;...
     1.00014910566816e+00;...
     1.00004009631930e+00;...
     1.00001078324501e+00;...
     1.00000290003838e+00;...
     1.00000077993879e+00;...
     1.00000020975773e+00;...
     1.00000005641253e+00;...
     1.00000001517167e+00;...
     1.00000000408029e+00;...
     1.00000000109736e+00;...
     1.00000000029513e+00;...
     1.00000000007937e+00;...
     1.00000000002135e+00];
xhat = aitken(x(1:end-2),x(2:end-1),x(3:end));
xhathat = aitken(xhat(1:end-2),xhat(2:end-1),xhat(3:end));
semilogy([1:length(x)],abs(1-x),[1:length(xhat)],abs(1-xhat),...
[1:length(xhathat)],abs(1-xhathat))
legend('x','xhat','xhathat')
```

Tabella A.8: Capitolo 2, Esercizio 11.

```
function [x,iter,errest,m] = newtonm(fun,fun1,x0,tol,maxit,varargin)
%
% [x,iter,errest] = newtonm(fun,fun1,x0,tol,maxit,varargin)
%
m0 = 1;
x = x0;
iter = 1;
errest = -feval(fun,x,varargin{:})/feval(fun1,x,varargin{:});
x = x+errest;
x1 = x;
iter = iter+1;
errest = -feval(fun,x,varargin{:})/feval(fun1,x,varargin{:});
m = m0+1;
while (abs(errest) > tol) & (iter < maxit) & (abs(m-m0) > m/100)
  m0 = m;
  x = x+errest;
  iter = iter+1;
  num = feval(fun,x,varargin{:});
  den = feval(fun1,x,varargin{:});
  errest = -feval(fun,x,varargin{:})/feval(fun1,x,varargin{:});
  m = (x1-x0)/(2*x1-x-x0);
  x0 = x1;
  x1 = x;
end
errest = errest*m;
while (abs(errest) > tol) & (iter < maxit)
  x = x+errest;
  iter = iter+1;
  num = feval(fun,x,varargin{:});
  den = feval(fun1,x,varargin{:});
  errest = -m*feval(fun,x,varargin{:})/feval(fun1,x,varargin{:});
end
errest = abs(errest);
if (errest > tol)
  warning('Impossibile raggiungere la tolleranza richiesta')
  warning('entro il numero massimo di iterazioni consentito.')
end
```

Tabella A.9: Metodo di Newton per radici multiple, Capitolo 2, Esercizio 13.

```
function [x,iter,errest] = newtonaitken(fun,fun1,x0,tol,maxit,varargin)
%
% [x,iter,errest] = newtonaitken(fun,fun1,x0,tol,maxit,varargin)
%
x = x0;
iter = 1;
errest = -feval(fun1,x,varargin{:})\feval(fun,x,varargin{:});
while ((norm(errest) > tol) & (iter+2 < maxit))
  x1 = x0+errest;
  iter = iter+1;
  errest = -feval(fun1,x1,varargin{:})\feval(fun,x1,varargin{:});
  x2 = x1+errest;
  iter = iter+1;
  errest = -feval(fun1,x2,varargin{:})\feval(fun,x2,varargin{:});
  x = aitken(x0,x1,x2);
  iter = iter+1;
  errest = -feval(fun1,x,varargin{:})\feval(fun,x,varargin{:});
  x0 = x;
end
errest = abs(errest);
if (errest > tol)
  warning('Impossibile raggiungere la tolleranza richiesta')
  warning('entro il numero massimo di iterazioni consentito.')
end
```

Tabella A.10: Metodo di Newton–Aitken, Capitolo 2, Esercizio 13.

```
function [U,varargout] = meg(A,varargin)
%
% function [U,b1,L,P] = meg(A,b)
%
% multi rhs
% optional LU factorization
% optional LU = PA
%
if (nargin == 2)
  b = varargin{1};
  [n,m] = size(b);
  A = [A,b];
else
  n = size(A,1);
  m = 0;
end
P = eye(n);
for i = 1:n-1
  [massimo,ii] = max(abs(A(i:n,i)));
  ii = ii+i-1;
  if (ii ~= i)
    A = A([1:i-1,ii,i+1:ii-1,i,ii+1:n],:);
    P = P([1:i-1,ii,i+1:ii-1,i,ii+1:n],:);
  end
  A(i+1:n,i) = A(i+1:n,i)/A(i,i);
  for j = i+1:n
    A(j,i+1:n+m) = A(j,i+1:n+m)-A(j,i)*A(i,i+1:n+m);
  end
end
if (nargin == 1)
% L,U,P
  A = A(:,1:n);
  U = triu(A);
  if (nargout == 2)
    varargout{1} = P'*(tril(A,-1)+eye(n));
  elseif (nargout == 3)
    varargout{1} = (tril(A,-1)+eye(n));
    varargout{2} = full(P);
  end
else
% b1
  varargout{1} = A(:,n+1:n+m);
  A = A(:,1:n);
  U = triu(A);
  if (nargout == 3)
    varargout{2} = P'*(tril(A,-1)+eye(n));
  elseif (nargout == 4)
    varargout{2} = (tril(A,-1)+eye(n));
    varargout{3} = P;
  end
```

```
function X = fs(L,B)
%
% X = fs(L,B)
%
n = length(B);
X = B;
X(1,:) = X(1,:)./L(1,1);
for i = 2:n
  X(i,:) = (X(i,:)-L(i,1:i-1)*X(1:i-1,:))/L(i,i);
end
```

Tabella A.12: Algoritmo delle sostituzioni in avanti: Capitolo 3, Esercizio 3.

```
clear all
close all
A = [15,6,8,11;6,6,5,3;8,5,7,6;11,3,6,9];
exact = ones(4,1);
for i = 1:3
  A = A^i;
  b = A*exact;
  [U,b1] = meg(A,b);
  x = bs(U,b1);
  condizionamento(i) = cond(A);
  errore(i) = norm(x-exact);
end
semilogy(condizionamento,errore)
```

Tabella A.13: Capitolo 3, Esercizio 2.

```
clear all
A = rand(5);
[L,U,P] = lu(A);
Y = fs(L,P);
X = bs(U,Y);
```

Tabella A.14: Capitolo 3, Esercizio 4.

```
function [x,iter,errest] = jacobi(A,b,x0,tol,maxit)
%
% [x,iter,errest] = jacobi(A,b,x0,tol,maxit)
%
n = length(A);
d = diag(A);
P = spdiags(1./d,0,n,n)*([spdiags(d,0,n,n)-A,b]);
g = P(:,n+1);
B = P(:,1:n);
x = x0;
iter = 0;
errest = tol+1;
while ((errest > tol) & (iter < maxit))
  x = B*x0+g;
  iter = iter+1;
  errest = norm(x-x0);
  x0 = x;
end
if (errest > tol)
  warning('Impossibile raggiungere la tolleranza richiesta')
  warning('entro il numero massimo di iterazioni consentito.')
end
```

Tabella A.15: Metodo di Jacobi: Capitolo 3, Esercizio 8.

```
function [lambda,y,iter,errest] = potenze(A,x,tol,maxit)
%
% [lambda,y,iter,errest] = potenze(A,x,tol,maxit)
%
if (length(A) == 1)
  lambda = A;
  y = 1;
  iter = 0;
  errest = 0;
  return
end
y = x/norm(x);
x = A*y;
lambda = y'*x;
iter = 1;
y = x/norm(x);
errest = tol+1;
lambda0 = lambda;
while ((errest > tol) & (iter < maxit))
  x = A*y;
  lambda = y'*x;
  iter = iter+1;
  errest = abs(lambda-lambda0);
  y = x/norm(x);
  lambda0 = lambda;
end
if (errest > tol)
  warning('Impossibile raggiungere la tolleranza richiesta')
  warning('entro il numero massimo di iterazioni consentito.')
end
```

Tabella A.16: Capitolo 4, Metodo delle potenze 1.

```
function y = normestimate(A)
y = sqrt(potenze(A*A',1e-6,100));
```

Tabella A.17: Capitolo 4, Stima della norma 2 2.

```
function A1 = deflaziona(A,x)
%
% A1 = deflaziona(A,x)
%
if (length(A) == 1)
  A1 = A;
  warning('Matrice di ordine 1.');
  return
end
w = x;
w(1) = w(1)-1;
beta = norm(w);
if (beta == 0)
  P = eye(size(A));
else
  P = (eye(size(A))-2*(w*w')/beta^2);
end
% B = P*A*P';
% P e' simmetrica
B = P*A*P;
A1 = B(2:end,2:end);
```

Tabella A.18: Capitolo 4, Deflazione 3.

```
close all
clear all
N = 10;
A = sparse(toeplitz([-2,1,zeros(1,N-2)])*N^2);
imax = 300;
tol = 1e-14;
x0 = rand(size(A,1),1);
% vedere cosa succede spostando il vettore iniziale random dentro il ciclo
for i = 1:imax
  [lambda1(i),x] = potenze(A,x0,tol,i);
end
B = deflaziona(A,x);
lambda2 = potenze(B,rand(size(B,1),1),tol,imax);
semilogy([1:(imax-1)],abs(lambda1(1:end-1)-lambda1(end)),'*',...
[1:(imax-1)],abs(lambda2/lambda1(end)).^(2*[1:(imax-1)]),...
[1:(imax-1)],tol*ones(1,imax-1));
legend('potenze','stima','tolleranza')
title('matrice simmetrica')
```

Tabella A.19: Capitolo 4, Esercizio 4.

```
function [lambda,y,iter,errest] = potenzeinverse(A,x,tol,maxit,varargin)
%
%  [lambda,y,iter,errest] = potenzeinverse(A,x,tol,maxit,varargin)
%
if (nargin == 5)
  mu = varargin{1};
else
  mu = 0;
end
if (length(A) == 1)
  lambda = A;
  y = 1;
  iter = 0;
  errest = 0;
  return
end
y = x/norm(x);
[L U P] = lu(A-mu*eye(size(A)));
z = L\(P*y);
x = U\z;
iter = 1;
lambda0 = y'*x;
y = x/norm(x);
errest = tol+1;
while (errest > tol) & (iter < maxit)
  z = L\(P*y);
  x = U\z;
  iter = iter+1;
  lambda = y'*x;
  errest = abs(lambda-lambda0);
  y = x/norm(x);
  lambda0 = lambda;
end
lambda = mu+1/lambda;
if (errest > tol)
  warning('Impossibile raggiungere la tolleranza richiesta')
  warning('entro il numero massimo di iterazioni consentito.')
end
```

Tabella A.20: Metodo delle potenze inverse con shift: Capitolo 4, Esercizio 6.

```
clear all
n = 6;
a = [13 -364 2912 -9984 16640 -13312 4096];
F = spdiags(ones(n,1),1,n,n);
F(n,:) = -a(n+1:-1:2)./a(1);
tol = 1e-12;
maxit = 200;
for i = 1:6
  [lambda(i) x] = potenze(F,tol,maxit);
  F = deflaziona(F,x);
end
errore = norm(lambda.'-roots(a),Inf)
```

Tabella A.21: Capitolo 4, Esercizio 7.

```
function [lambda,iter,errest] = autovaloriqr(A,tol,maxit,varargin)
%
% lambda = autovaloriqr (A,tol,maxit)
%
if (nargin == 3)
  mu = 0;
else
  mu = varargin{1};
end
mu = mu*eye(size(A));
iter = 0;
errest = tol+1;
while ((errest > tol) & (iter < maxit))
  iter = iter+1;
  [Q,R] = qr(A-mu);
  A = R*Q+mu;
  errest = norm(diag(A,-1),Inf);
end
if (errest > tol)
  warning('Impossibile raggiungere la tolleranza richiesta')
  warning('entro il numero massimo di iterazioni consentito.')
end
lambda = diag(A);
```

Tabella A.22: Capitolo 4, Esercizio 8.