ELABORATO DI CALCOLO NUMERICO

Filippo Mameli

15 marzo 2016

Indice

Capitolo 1

Errori ed aritmetica finita

1.1 Esercizi del libro

1.1.1 Esercizio 1.1

Sia $x = \pi \approx 3.1415 = \tilde{x}$. Calcolare il corrispondente errore relativo ϵ_x. Verificare che il numero di cifre decimali corrette nella rappresentazione approssimata di x mediante $\tilde{x}$ è all'incirca dato da

$$-\log_{10}|\epsilon_x|$$

Soluzione:

$$x = \pi \approx 3.1415, \qquad \tilde{x} = 3.1415$$
$$\Delta_x = x - \tilde{x} = 0,0000926$$
$$\epsilon_x = \tfrac{\Delta_x}{x} = 0,3 \cdot 10^{-4}$$
$$-log_{10}|\epsilon_x| = -log_{10}|0,3 \cdot 10^{-4}| \approx 4,5$$

Arrotondando $4,5$ si nota che il numero delle cifre decimali corrette nella rappresentazione è 4

1.1.2 Esercizio 1.2

Dimostrare che, se $f(x)$ è sufficientemente regolare e $h > 0$ è una quantità "piccola", allora:

$$\tfrac{f(x_0+h)-f(x_0-h)}{2h} = f'(x_0) + O(h^2)$$

$$\tfrac{f(x_0+h)-2f(x_0)+f(x_0-h)}{h^2} = f''(x_0) + O(h^2)$$

1.1.3 Esercizio 1.3

Dimostrare che il metodo iterativo (1.1), *convergente a x^* (vedi*(1.2)), *deve verificare la condizione di consistenza*

$$x^* = \Phi(x^*)$$

Ovvero la soluzione cercata deve essere un punto fisso per la funzione di iterazione che definisce il metodo.

1.1.4 Esercizio 1.4

Il metodo iterativo

$$x_{n+1} = \frac{x_n x_{n-1} + 2}{x_n + x_{n-1}}, \qquad n = 1, 2, ..., x_0 = 2, \qquad x_1 = 1.5$$

definisce una successione di approssimazioni convergente a $\sqrt{2}$*. Calcolare a quale valore di* n *bisogna arrestare l'iterazione, per avere un errore di convergenza* $\approx 10^{-22}$ *(comparare con i risultati in Tabella 1.1)*

1.1.5 Esercizio 1.5

Il codice Fortran

```fortran
program INTERO
c---variabili intere da 2 byte
integer*2 numero, i
numero = 32765
do i = 1, 10
write(*,*) i, numero
numero = numero +1
end do
end
```

Produce il seguente output:

1. 32765
2. 32766
3. 32767
4. -32768
5. -32767
6. -32766
7. -32765
8. -32764
9. -32763
10. -32762

Spiegarne il motivo

1.1.6 Esercizio 1.6

Dimostrare i teoremi 1.1 e 1.3

1.1.7 Esercizio 1.7

Completare la dimostrazione del teorema 1.4

1.1.8 Esercizio 1.8

Quante cifre binarie sono utilizzate per rappresentare, mediante arrotondamento, la mantissa di un numero, sapendo che la precisione di machina è $u \approx 4.66 \cdot 10^{-10}$

1.1.9 Esercizio 1.9

Dimostrare che, detta u *la precisione di macchina utilizzata,*

$$-log_{10}u$$

fornisce, approssimativamente, il numero di cifre decimali correttamente rappresentate dalla mantissa.

1.1.10 Esercizio 1.10

Con riferimento allo standard IEEE 754 determinare, relativamente alla doppia precisione:

1. il più grande numero di macchina
2. il più piccolo numero di macchina normalizzato positivo
3. il più piccolo numero di macchina denormalizzato positivo
4. la precisione di macchina

Confrontare le risposte ai primi due quesiti col risultato fornito dalle `function` $Matlab$ `realmax` e `realmin`

1.1.11 Esercizio 1.11

Eseguire le seguenti istruzioni $Matlab$*:*

```
x = 0; delta = 0.1;
while x \tilde= 1, x = x+delta end
```

Spiegarne il (non) funzionamento

1.1.12 Esercizio 1.12

Individuare l'algoritmo più efficace per calcolare, in aritmetica finita, l'espressione $\sqrt{x^2 + y^2}$

1.1.13 Esercizio 1.13

Eseguire le seguenti istruzioni $Matlab$*:*

```
help eps
((eps/2+1)-1)*(2/eps)
(eps/2+(1-1))*(2/eps)
```

Concludere che la somma algebrica non gode, in aritmetica finita, della proprietà associativa.

1.1.14 Esercizio 1.14

Eseguire e discutere il risultato delle seguenti istruzioni Matlab

$(1e300 - 1e300) * 1e300, \qquad (1e300 * 1e300) - (1e300 * 1e300)$

1.1.15 Esercizio 1.15

Eseguire l'analisi dell'errore (relativo), dei due seguenti algoritmi per calcolare la somma di tre numeri:

$$1)\ (x \oplus y) \oplus z), \qquad 2)\ x \oplus (y \oplus z)$$

1.1.16 Esercizio 1.17

(Cancellazione numerica) Si supponga di dover calcolare l'espressione

$$y = 0.12345678 - 0.12341234 \equiv 0.0000444,$$

utilizzando una rappresentazione decimale con arrotondamento alla quarta cifra significativa. Comparare il risultato esatto con quello ottenuto in aritmetica finita, e determinare la perdita di cifre significative derivante dalla operazione effettuata. Verificare che questo risultato è in accordo con l'analisi di condizionamento.

1.1.17 Esercizio 1.18

(Cancellazione Numerica) Eseguire le seguenti istruzioni Matlab

```
format long e
a = 0.1
b = 0.099999999999
a-b
```

Valutare l'errore relativo sui dati di ingresso e l'errore relativo sul risultato ottenuto.

1.2 Esercizi integrativi

1.2.1 Esercizio 1

un'approssimazione del secondo ordine di $f'(x_0)$ utilizzando il passo di discretizzazione h e i seguenti tre valori di funzione $f(x_0), f(x_0+h), f(x_0+2h)$ (molecola a tre punti in avanti).

1.2.2 Esercizio 2

Dimostrare che se un numero reale x viene approssimato da $\tilde{x}$ con un certo errore relativo ϵ_x, la quantità $-log_{10}|\epsilon_x|$ fornisce approssimativamente il numero di cifre decimali esatte di $\tilde{x}$.

1.2.3 Esercizio 3

Calcolare il più grande e il più piccolo numero reale di macchina positivo normalizzato che si può rappresentare utilizzando lo standard IEEE 754 nel formato della singola precisione e in quello della doppia precisione.
Soluzione:
Il numero più piccolo è uguale a:

$$R_1 = b^{\nu}$$

Per cui nel caso di questo esercizio:

$$2^{-126}$$

Il numero più grande è uguale a

$$R_2 = (1 - 2^{-24})2^{\varphi}$$

con

$$\varphi = 2^8 - 127$$

quindi

$$R_2 = 6.805646933 \cdot 10^{38}$$

.

1.2.4 Esercizio 4

Siano $x = 2.7352 \cdot 10^2, y = 4.8017 \cdot 10^{-2}$ *e* $z = 3.6152 \cdot 10^{-2}$. *Utilizzando un'aritmetica finita che lavora in base 10 con arrotondamento e che riserva* $m = 4$ *cifre alla mantissa, confrontare gli errori assoluti* $R_1 - R$ *e* $R_2 - R$*, dove* $R = x + y + z$ *e*

$$R_1 = (x \oplus y) \oplus z, \qquad R_2 = x \oplus (y \oplus z).$$

Soluzione:
Per facilitare i calcoli si portano x e y da 10^{-2} a 10^2:

$$y = 4.8017 \cdot 10^{-2} = 0.00048017 \cdot 10^2$$

$$z = 3.6152 \cdot 10^{-2} = 0.00031652 \cdot 10^2$$

Successivamente si calcola R_1, R_2 ed R:

$$R_1 = (x \oplus y) \oplus z) = (2.7352 + 0.00048017) + 0.00031652 = 2,7359 \cdot 10^2$$

$$R_2 = x \oplus (y \oplus z) = 2.7352 + (0.00048017 + 0.00031652) = 2.7360 \cdot 10^2$$

$$R = x + y + z = 2.73604169$$

Attraverso R si calcolano gli errori assoluti su R_1 ed R_2:

$$R_1 - R = -0.000014169 \cdot 10^2$$

$$R_2 - R = 0.000004169 \cdot 10^2$$

1.2.5 Esercizio 5

Un'aritmetica finita utilizza la base 10, l'arrotondamento, $m = 5$ cifre per la mantissa, $s = 2$ cifre per l'esponente e lo shift $\nu = 50$. Per gli interi esso utilizza $N = 7$ cifre decimali. Dire se il numero intero $x = 136726$ è un numero intero di macchina e come viene convertito in reale di macchina. Dire quindi se il numero intero $x = 78345 \cdot 10^{40}$ un reale di macchina e/o se è un intero di macchina.
Soluzione:
Si verifica facilmente che il numero $x = 136726$ è un intero di macchina.
Il numero x viene convertito in reale di macchina in questo modo: $x_r = 1,36726 \cdot 10^5$
Invece il numero $x = 78345 \cdot 10^{40}$ è un reale di macchina in quanto non bastano 5 cifre per rappresentarlo.
Per un maggiore sicurezza si calcola il più grande reale di macchina:
$R_2 = (1 - 10^{-5})10^{50} = 9.9999 \cdot 10^{49}$
Essendo $x < R_2$ è confermato che è un reale di macchina.

1.2.6 Esercizio 6

Dimostrare che il numero di condizionamento del problema di calcolo $\sqrt[n]{x}$ è $\frac{1}{n}$.
Soluzione:
Per prima cosa si riscrive la funzione: $f(x) = x^{\frac{1}{n}}$
Il numero di condizionamento è uguale a $k = |f'(x)\frac{x}{y}|$
Si calcola la derivata di $f(x)$ che è uguale a $f'(x) = \frac{x^{\frac{1}{n}-1}}{n}$
Quindi: $k = |\frac{x^{\frac{1}{n}-1}}{n} \frac{x}{\sqrt[n]{x}}|$
Da cui con passaggi algebrici: $|\frac{\sqrt[n]{x}\ x^{-1}}{n} \frac{x}{\sqrt[n]{x}}| = |\frac{1}{n}| = \frac{1}{n}$.

1.2.7 Esercizio 7

Individuare l'algoritmo più efficace per valutare, in aritmetica finita, la funzione $f(x) = \ln x^4$
Soluzione:
Bisogna semplicemente riscrivere la funzione per evitare problemi di overflow e underflow e poi valutarla. $f(x) = \ln x^4 = 4 \cdot \ln x$
L'algoritmo è questo:
Result $= 4 \cdot \ln x$
Return Result

1.2.8 Esercizio 8

Individuare una forma algebrica equivalente ma preferibile in aritmetica finita per il calcolo dell'espressione $(x + 2)^3 - x^3$
Soluzione:

$$(x+2)^3-x^3 = (x+2)(x+2)^2-x^3 = (x+2)(x^2+4x+4)-x^3 = x^3+4x^2+4x+2x^2+8x+8-x^3 = 6x^2+12x+8$$

1.2.9 Esercizio 9

Si calcoli l'approssimazione $\tilde{y}$ della differenza tra y fra $x_2 = 3.5555$ e $x_1 = 3.5554$ utilizzando un'aritmetica finita che lavora con arrotondamento in base 10 con 4 cifre per la mantissa

normalizzata. Se ne calcoli quindi il corrispondente errore relativo e la maggiorazione di esso che si ottiene utilizzando il numero di condizionamento della somma algebrica.
Soluzione:
Prima si calcola un'approssimazione di x_1 e x_2 poi si calcola un approssimazione $\tilde{y}$ della differenza y. $\tilde{x_1} = 3.555$
$\tilde{x_2} = 3.556$
$\tilde{y} = \tilde{x_1} - \tilde{x_2} = 0.001$ $y = x_2 - x_1 = 0.0001$
L'errore relativo è uguale a: $\varepsilon_y = \frac{0.001-0.0001}{0.0001} = 9$
Per la maggiorazione serve $max\{|\varepsilon_{x_1}|, |\varepsilon_{x_2}|\}$
$\varepsilon_{x_1} = \frac{\tilde{x_1}-x_1}{x_1} = -1.125 \cdot 10^{-4}$
$\varepsilon_{x_2} = \frac{\tilde{x_2}-x_2}{x_2} = 1.406 \cdot 10^{-4}$
Quindi: $k = \frac{|-3.5554|+|3.5555|}{|3.5555-3.5554|} \cdot 1.406 \cdot 10^{-4} = 9.9979254$

1.2.10 Esercizio 10

Dimostrare che il numero razionale 0.1 *(espresso in base 10) non può essere un numero di macchina in un'aritmetica finita che utilizza la base 2 indipendentemente da come viene fissato il numero m di bit riservati alla mantissa. Dare una maggiorazione del corrispondente errore relativo di rappresentazione supponendo di utilizzare l'aritmetica finita binaria che utilizza l'arrotondamento e assume $m = 7$.*

Capitolo 2

Radici di una equazione

2.1 Esercizio 2.1

Definire una procedura iterativa basata sul metodo di Newton per determinare $\sqrt{a}$, per un assegnato $a > 0$. Costruire una tabella dell'approssimazioni relativa al caso $a = x_0 = 2$ (Comparare con la tabella 1.1)

2.2 Esercizio 2.2

Generalizzare il risultato del precedente esercizio, derivando una procedura iterativa basata sul metodo di Newton per determinare $\sqrt[n]{a}$ per un assegnato $a > 0$

2.3 Esercizio 2.3

In analogia con quanto visto nell'Esercizio 2.1, definire una procedura iterativa basata sul metodo delle secanti per determinare $\sqrt{a}$. Confrontare con l'esercizio 1.4.

2.4 Esercizio 2.4

Discutere la convergenza del metodo di Newton, applicato per determinare le radici dell'equazione (2.11) in funzione della scelta del punto iniziale x_0

2.5 Esercizio 2.5

Comparare il metodo di Newton (2.9), il metodo di Newton modificato (2.16) ed il metodo di accellerazione di Aitken (2.17), per approsimare gli zeri delle funzioni:

$$f_1(x) = (x-1)^{10}, \qquad f_2(x) = (x-1)^{10}e^x$$

per valori decrescenti della tolleranza `tolx`. Utilizzare, in tutti i casi, il punto iniziale $x_0 = 10$.

2.6 Esercizio 2.6

È possibile, nel caso delle funzioni del precedente esercizio utilizzare il metodo di bisezione per determinare lo zero?

2.7 Esercizio 2.7

Costruire una tabella in cui si comparano, a partire dallo stesso punto iniziale $\texttt{x}_0 = \texttt{0}$, e per valori decrescenti della tolleranza `tolx`, il numero di iterazioni richieste per la convergenza dei metodi di Newton, corde e secanti, utilizzati per determinare lo zero della funzione

$$f(x) = x - \cos x$$

2.8 Esercizio 2.8

Completare i confronti del precedente esercizio inserendo quelli con il metodo di bisezione, con intervallo di confidenza iniziale $[0, 1]$.

2.9 Esercizio 2.9

Quali controlli introdurreste, negli algoritmi 2.4-2.6, al fine di rendere più "robuste" le corrispondenti iterazioni?

Capitolo 3

Capitolo 3

3.1 Esercizio 3.1

Riscrivere gli Algoritmi 3.1-3.4 in modo da controllare che la matrice dei coefficenti sia non singolare.

3.2 Esercizio 3.2

Dimostrare che la somma ed il prodotto di matrici triangolari inferiori (superiori), è una matrice triangolare inferiore (superiore).

3.3 Esercizio 3.3

Dimostrare che il prodotto di due matrici triangolari inferiori a diagonale unitaria è a sua volta una matrice triangolare inferiore a diagonale unitaria.

3.4 Esercizio 3.4

Dimostrare che la matrice inversa di una matrice triangolare inferiore è as ua volta triangolare inferiore. Dimostrare inoltre che, se la matrice ha diagonale unitaria, tale è anche la diagonale della sua inversa.

3.5 Esercizio 3.5

Dimostrare i lemmi 3.2 e 3.3.

3.6 Esercizio 3.6

Dimostrare che il numero di flop richiesti dall'algoritmo 3.5 è dato da (3.25)

3.7 Esercizio 3.7

Scrivere una function matlab che implementi efficientemente l'algoritmo 3.5 per calcolare la fattorizzazione LU di una matrice.

3.8 Esercizio 3.8

Scrivere una function Matlab che, avendo in ingresso la matrice A riscritta dall'algoritmo 3.5, ed un vettore x contenente i termini noti del sistema lineare (3.1), ne calcoli efficientemente la soluzione.

3.9 Esercizio 3.9

Dimostrare i lemmi 3.4 e 3.5

3.10 Esercizio 3.10

Completare la dimostrazione del Teorema 3.6

3.11 Esercizio 3.11

Dimostrare che , se A è non singolare, le matrici A^TA e AA^T sono sdp.

3.12 Esercizio 3.12

Dimostrare che se $A \in \mathbb{R}^{m\times n}$ con $m \geq n = rank(A)$, allora la matrice A^TA è sdp.

3.13 Esercizio 3.13

Data una matrice $A \in \mathbb{R}^{n\times n}$, dimostrare che essa può essere scritta come

$$A = \frac{1}{2}(A + A^T) + \frac{1}{2}(A - A^T) \equiv A_s + A_a$$

dove $A_s = A_s^T$ è detta parte simmetrica di A, mentre $A_a = -A_a^T$ è detta parte asimmetrica di A. Dimostrare inoltre che, dato un generico vettore $x \in \mathbb{R}^n$, risulta

$$x^TAx = x^TA_sx$$

3.14 Esercizio 3.14

Dimostrare la consistenza delle formule (3.29).

3.15 Esercizio 3.15

Dimostrare che il numero di flop richiesti dall'algoritmo 3.6 è dato da 3.30

3.16 Esercizio 3.16

Scrivere una function Matlab che implementi efficientemente l'algoritmo di fattorizzazione LDL^T *per matrici sdp*

3.17 Esercizio 3.17

Scrivere una funzione Matlab che, avendo in ingresso la matrice A *prodotta dalla precedente funziona, contenente la fattorizzazione* LDL^T *della matrice sdp originaria, ed un vettore di termini noti,* x*, calcoli efficientemente la soluzione del corrispondente sistema lineare.*

3.18 Esercizio 3.18

Utilizzare la function dell'esercizio 3.16 per verificare che la matrice

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 2 & 2 \\ 1 & 2 & 1 & 1 \\ 1 & 2 & 1 & 2 \end{pmatrix}$$

non è sdp.

3.19 Esercizio 3.19

Dimostrare che, al passo i-esimo di eliminazione di Gauss con pivoting parziale, si ha $a_{k_i i}^{(i)} \neq 0$*, se* A *è non singolare.*

3.20 Esercizio 3.20

Con riferimento alla matrice A *definita nella (3.24), qual è la matrice di permutazione* P *che rende* PA *fattorizzatile* LU*? Chi sono, in tal caso, i fattori* L *e* U*?*

3.21 Esercizio 3.21

Scrivere una function Matlab che implementi efficientemente l'algoritmo di fattorizzazione LU *con pivoting parziale.*

3.22 Esercizio 3.22

Scrivere una funzione matlab che, avendo in ingresso la matrice A *prodotta dalla precedente function, contenente la fattorizzazione* LU *della matrice permutata, il vettore* p *contenente l'informazione relativa alla corrispondente matrice di permutazione, ed un vettore di termini noti,* x*, calcoli efficientemente la soluzione del corrispondente sistema lineare.*

3.23 Esercizio 3.23

Costruire alcuni esempi di applicazione delle funzioni degli esercizi 3.21 e 3.22

3.24 Esercizio 3.24

Le seguenti istruzioni MATLAB *sono equivalenti a risolvere il dato sistema 2×2 con l'utilizzo del pivoting e non, rispettivamente. Spiegarne il differente risultato ottenuto. Concludere che l'utilizzo del pivoting migliora, in generale, la prognosi degli errori in aritmetica finita.*

```
A = [eps 1;1 0], b = [1; 1/4]
A\b
L = [1 0; 1/eps 1], U = [eps 1; 0 -1/eps]
L*U-A
U\(L\b)
```

3.25 Esercizio 3.25

Si consideri la seguente matrice bidiagonale inferiore

$$A = \begin{pmatrix} 1 & & & \\ 100 & 1 & & \\ & \ddots & \ddots & \\ & & 100 & 1 \end{pmatrix}_{10\times 10}.$$

Calcolare $k_\infty(A)$. Confrontate il risultato con quello fornito dalla `function cond` *di* MATLAB*. Dimostrare, e verificare, che $k_\infty(A) = k_1(A)$.*

3.26 Esercizio 3.26

Si consideri i seguenti vettori di $\mathbb{R}^{10}$,

$$\underline{b} = \begin{pmatrix} 1 \\ 101 \\ \vdots \\ 101 \end{pmatrix}, \quad \underline{c} = 0.1 \cdot \begin{pmatrix} 1 \\ 101 \\ \vdots \\ 101 \end{pmatrix},$$

ed i seguenti sistemi lineari

$$A\underline{x} = \underline{b}, \qquad A\underline{y} = \underline{c},$$

in cui A è la matrice definita nel precedente Esercizio **??**. *Verificare che le soluzioni di questi sistemi lineari sono, rispettivamente, date da:*

$$\underline{x} = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}, \qquad \underline{y} = \begin{pmatrix} 0.1 \\ \vdots \\ 0.1 \end{pmatrix}.$$

Confrontare questi vettori con quelli calcolato dalle seguenti due serie di istruzioni MATLAB*,*

```
b=[1 101*ones(1,9)]';
x(1)=b(1); for i=2:10, x(i)=b(i)-100*x(i-1), end
x=x(:)

c=0.1+[1 101*ones(1,9)]';
y(1)=c(1); for i=2:10, y(i)=c(i)-100*y(i-1); end
y=y(:)
```

che implementano, rispettivamente, le risoluzioni dei due sistemi lineari. Spiegare i risultati ottenuti, alla luce di quanto visto in Sezione 3.7.

3.27 Esercizio 3.27

Dimostrare che il numero di `flop` *richiesti dall'Algoritmo di fattorizzazione QR di Householder è dato da 3.60.*

3.28 Esercizio 3.28

Definendo il vettore $\underline{\hat{v}} = \frac{v}{v_1}$*, verificare che* `beta`*, come definito nel seguente algoritmo per la fattorizzazione QR di Householder, corrisponde alla quantità* $\frac{2}{\underline{\hat{v}}^T \underline{\hat{v}}}$

3.29 Esercizio 3.29

Scrivere una `function` MATLAB *che implementi efficientemente l'algoritmo di fattorizzazione QR, mediante il metodo di Householder (vedi l'Algoritmo descritto nell'Esercizio precedente).*

3.30 Esercizio 3.30

Scrivere una `function` MATLAB *che, avendo in ingresso la matrice A prodotta dalla* `function` *del precedente Esercizio, contenente la fattorizzazione QR della matrice originaria, e un corrispondente vettore di termini noti $\underline{b}$, calcoli efficientemente la soluzione del sistema lineare sovradeterminato.*

3.31 Esercizio 3.31

Utilizzare le `function` *degli Esercizi 3.29 e 3.30 per calcolare la soluzione ai minimi quadrati di (3.51), ed il corrispondente residuo, nel caso in cui*

$$A = \begin{pmatrix} 3 & 2 & 1 \\ 1 & 2 & 3 \\ 1 & 2 & 1 \\ 2 & 1 & 2 \end{pmatrix}, \qquad \underline{b} = \begin{pmatrix} 10 \\ 10 \\ 10 \\ 10 \end{pmatrix}.$$

3.32 Esercizio 3.32

Calcolare i coefficienti della equazione della retta

$$r(x) = a_1 x + a_2,$$

che meglio approssima i dati prodotti dalle seguenti istruzioni MATLAB*:*

```
x = linspace(0,10,101)'
gamma = 0.1
y = 10*x+5+(sin(x*pi))*gamma
```

Riformulare il problema come minimizzazione della norma Euclidea di un corrispondente vettore residuo. Calcolare la soluzione utilizzando le `function` *sviluppate negli Esercizi 3.29 e 3.30, e confrontarla con quella ottenuta dalle seguenti istruuzioni* MATLAB*:*

```
A = [x x.^0]
a = A\y
r = A*a-y
```

Confrontare i risultati che si ottengono per i seguenti valori del parametro `gamma`*:*

$$0.5, 0.1, 0.05, 0.01, 0.005, 0.001.$$

Quale è la soluzione che si ottiene nel limite `gamma` $\to 0$*?*

3.33 Esercizio 3.33

Determinare il punto di minimo della funzione $f(x_1, x_2) = x_1^4 + x_1(x_1 + x_2) + (1 - x_2)^2$*, utilizzando il metodo di Newton (3.63) per calcolarne il punto stazionario.*

3.34 Esercizio 3.34

Uno dei metodi di base per la risoluzione di equazioni differenziali ordinarie, $y'(t) = f(t, y(t))$*,* $t \in [t_0, T]$*,* $y(t_0) = y_0$ *(problema di Cauchy di prim'ordine), è il metodo di Eulero implicito,*

$$y_n = y_{n-1} + hf_n, \qquad n = 1, 2, \ldots, N \equiv \frac{T - t_0}{h},$$

in cui $y_n \approx y(t_n)$*,* $f_n = f(t_n, y_n)$*,* $t_n = t_0 + nh$*. Utilizzare questo metodo nel caso in cui* $t_0 = 0$*,* $T = 10$*,* $N = 100$*,* $y_0 = (1, 2)^T$ *e, se* $y = (x_1, x_2)^T$*,*
$f(t, y) = (-10^3 x_1 + \sin x_1 \cos x_2, -2x_2 + \sin x_1 \cos x_2)^T$*. Utilizzare il metodo (3.64) per la risoluzione dei sistemi nonlineari richiesti.*

Capitolo 4

Approssimazione di funzioni

4.1 Esercizio 1

Sia $f(x) = 4x^2 - 12x + 1$. *Determinare* $p(x) \in \Pi_4$ *che interpola* $f(x)$ *sulle ascisse* $x_i = i, i = 0, \dots 4$.

Soluzione.
Caso Lagrange:
Per prima cosa si calcolano gli $f(x_i)$ per ogni i=0,...,4:
$f(0) = 1$
$f(1) = -7$
$f(2) = -7$
$f(3) = 1$
$f(4) = 17$
Adesso calcoliamo $L_{kn}(x)$ con $k = 0, ..., 4$ e $n = 4$:
$L_{04}\frac{(x-1)(x-2)(x-3)(x-4)}{24}$
$L_{14}\frac{(-x)(x-2)(x-3)(x-4)}{6}$
$L_{24}\frac{(x)(x-1)(x-3)(x-4)}{4}$
$L_{34}\frac{(-x)(x-1)(x-2)(x-4)}{6}$
$L_{44}\frac{(x)(x-1)(x-2)(x-3)}{24}$
A questo punto possiamo scrivere $p(x) \in \Pi_4 = \frac{(x-1)(x-2)(x-3)(x-4)}{24}-7\frac{(-x)(x-2)(x-3)(x-4)}{6}-7\frac{(x)(x-1)(x-3)(x-4)}{4} + \frac{(-x)(x-1)(x-2)(x-4)}{6}$+17$\frac{(x)(x-1)(x-2)(x-3)}{24}$

Eseguendo i calcoli, si ottiene il polinomio $p(x) = 4x^2 - 12x + 1$ □

4.2 Esercizio 2

Dimostrare che il seguente algoritmo,

```
p = a(n+1)
for k = n:-1:1
  p = p*x +a(k)
end
```

valuta il polinomio (4.4) nel punto x, se il vettore a contiene i coefficienti del polinomio p(x) (Osservare che in MATLAB *i vettori hanno indice che parte da 1, invece che da 0).*

Soluzione.
Un polinomio nella base delle potenze è scritto come $\sum_{k=0}^{n} a_k x^k$; raccogliendo di volta in volta una x, si ottiene:

$$a_0 + a_1x + a_2x^2 + ... + a_nx^n = a_0 + x(a_1 + a_2x + ... + a_nx^{n-1}) = ... =$$
$$= a_0 + x(a_1 + x(a_2 + ... + x(a_{n-1} + a_nx)))$$

L'algoritmo di Horner esegue una somma e un prodotto ad ognuno degli n passi, il costo complessivo è, quindi, di $2n$ flops contro i $\sum_{i=0}^{n}(i+1) = \frac{(n+1)(n+2)}{2} \approx \frac{n^2}{2}$ flops della valutazione iniziale. Nell'implementazione è possibile valutare il polinomio in un insieme di punti, passando un vettore di ascisse. □

4.3 Esercizio 3

Dimostrare il lemma 4.1

Soluzione.
Si distingono due casi. Se $k = i :\Rightarrow$

$$L_{kn}(x_i) = \prod_{j=0, j\neq k}^{n} \frac{x_i - x_j}{x_i - x_j} = 1$$

Se $k \neq i :\Rightarrow$

$$L_{kn}(x_i) = \prod_{j=0, j\neq k}^{n} \frac{x_i - x_j}{x_k - x_j} = \frac{x_i - x_0}{x_k - x_0} ... \frac{x_i - x_i}{x_k - x_i} ... \frac{x_i - x_n}{x_k - x_n} =$$
$$= \frac{x_i - x_0}{x_k - x_0} ...0... \frac{x_i - x_n}{x_k - x_n} = 0$$

1. Appare inoltre evidente che il polinomio ha grado esatto n: è il prodotto di (n+1)-1 fattori $x - x_j$ al numeratore (mentre al denominatore abbiamo una quantità costante, scelto k). Per quanto riguarda il coefficiente principale, osserviamo che:

$$L_{kn}(x) = \prod_{j=0, j\neq k}^{n} \frac{(x - x_j)}{(x_k - x_j)} = \frac{\prod_{j=0, j\neq k}^{n} (x - x_j)}{\prod_{j=0, j\neq k}^{n} (x_k - x_j)} = c_{kn} \prod_{j=0, j\neq k}^{n} (x - x_j)$$

   con

$$c_{kn} = \frac{1}{\prod_{j=0, j\neq k}^{n} (x_k - x_j)}$$

   costante e diverso da zero per ipotesi.

2. Gli n polinomi (al variare di k = 1...n) sono tra loro linearmente indipendenti, infatti:

$$a_1 L_{1n}(x) + ... + a_n L_{nn}(x)$$

si ha che per $x = x_i$ tutti i termini della somma si annullano (dalle dimostrazioni precedenti) e rimane soltanto $a_i L_{in}(x_i) = 0 \Rightarrow a_i 1 = 0$ e quindi estendendo il ragionamento su tutti gli i = 1...n abbiamo che devono essere nulli i vari a_i.

□

4.4 Esercizio 4

Dimostrare il lemma 4.2

Soluzione.

1. $w_k(x) \in {\Pi_k}^{'}$ è di grado k. Infatti:

$$w_k(x) = (x - x_{k-1})w_{k-1}(x) = (x - x_{k-1})(x - x_{k-2})w_{k-2}(x) =$$

...

$$= (x - x_{k-1})(x - x_{k-2})...(x - x_1)w_0(x) = (x - x_{k-1})(x - x_{k-2})...(x - x_1)1$$

Sicuramente il termine di grado più alto del polinomio risultante dal precedente prodotto sarà k. Inoltre, tale termine avrà come coefficiente 1.

2. Il prodotto del punto 1 può anche essere scritto come produttoria, ottenendo, quindi, la tesi:

$$w_k(x) = \prod_{j=0}^{k} (x - x_j)$$

3. è sufficiente verificare quanto segue:

$$w_{k+1}(x_j) = (x_j - x_k)(x_j - x_k)...(x_j - x_j)...(x_j - x_1)1 =$$

$$= (x_j - x_k)(x_j - x_k)...0...(x_j - x_1)1 = 0$$

4. Il generico $w_i(x)$ è un polinomio monico di grado i e appartiene a Π_i, per cui i polinomi per i = 1...k generano lo spazio Π_k.

   Sia $a_1 w_0(x) + ... + a_k w_k(x)$ una combinazione lineare. Questa deve essere banale, poichè, se così non fosse, ci sarebbero alcuni $w_i(x) = 0$ per ogni ascissa data $x_0, ..., x_n$; ma, quindi, nella produttoria $\prod_{j=0}^{k} (x - x_j)$ ci sarebbe un termine nullo per qualsiasi ascissa che consideriamo, ciò comporterebbe che i prodotti sono tutti nulli, ma tale asserzione è assurda.

□

4.5 Esercizio 5

Dimostrare il lemma 4.4

Soluzione.

1. Linearità delle differenze divise:

$$\begin{aligned}(\alpha f+\beta g)[x_0,\dots,x_r]=&\sum_{k=0}^{r}\frac{\alpha f_k+\beta g_k}{\prod_{j=0,j\neq k}^{r}(x_k-x_j)}=\\ =&\sum_{k=0}^{r}\frac{\alpha f_k}{\prod_{j=0,j\neq k}^{r}(x_k-x_j)}+\sum_{k=0}^{r}\frac{\beta g_k}{\prod_{j=0,j\neq k}^{r}(x_k-x_j)}=\\ =&\alpha\sum_{k=0}^{r}\frac{f_k}{\prod_{j=0,j\neq k}^{r}(x_k-x_j)}+\beta\sum_{k=0}^{r}\frac{g_k}{\prod_{j=0,j\neq k}^{r}(x_k-x_j)}=\\ =&\alpha f[x_0,\dots,x_r]+\beta g[x_0,\dots,x_r].\end{aligned}$$

2. La permutazione degli indici è possibile poichè somme e prodotti sono operazioni commutative.

3. Differenze divise e polinomi: dato che $f(x)$ è un polinomio ed il polinomio interpolante è unico,

$$f(x)=\sum_{i=0}^{k}a_ix^i=\sum_{i=0}^{k}f[x_0,\dots,x_k]w_k(x)=p_k(x).$$

   Il termine k-esimo è $a_kx^k=f[x_0,\dots,x_k]w_k(x)$ quindi $f[x_0,\dots,x_k]=a_k$ poiché $w_k(x)\in\Pi'_k$. I termini con $r>k$ non compaiono nella funzione dunque hanno coefficiente $f[x_0,\dots,x_r]=0$.

4. Derivabilità: sia $p(x)$ il polinomio interplante allora $e(x)=f(x)-p(x)\in C^{(r)}([a,b])$ ed $e(x_i)=0$ per $i=0,\dots,r$. Per il teorema di Rolle sulle funzioni continue, $\exists\xi_i^{(1)}\in[x_i,x_{i+1}]$ tali che $e'(\xi_i^{(1)})=0$ per $i=0,\dots,r-1$. Reiterando, $\exists\xi_i^{(2)}\in\left[\xi_i^{(1)},\xi_{i+1}^{(1)}\right]$ tali che $e''(\xi_i^{(2)})=0$ per $i=0,\dots,r-2$ e così via; infine $\exists\xi^{(r)}\in[a,b]$ tali che $e^{(r)}(\xi^{(r)})=0$. Segue $e^{(r)}(x)=f^{(r)}(x)-p^{(r)}(x)=f^{(r)}(x)-r!f[x_0,\dots,x_r]=0$ ovvero $f[x_0,\dots,x_r]=\frac{f^{(r)}(\xi^{(r)})}{r!}$.

5. Ricorsività:

$$
\begin{aligned}
&\frac{f[x_1,\dots,x_r]-f[x_0,\dots,x_{r-1}]}{x_r-x_0} = \\
=&\left[\sum_{k=1}^{r}\frac{f_k}{\prod_{j=1,j\neq k}^{r}(x_k-x_j)}+\sum_{k=0}^{r-1}\frac{f_k}{\prod_{j=0,j\neq k}^{r-1}(x_k-x_j)}\right]\frac{1}{x_r-x_0} = \\
=&\left[\sum_{k=1}^{r}\frac{f_k(x_j-x_0)}{\prod_{j=0,j\neq k}^{r}(x_k-x_j)}-\sum_{k=0}^{r-1}\frac{f_k(x_j-x_k)}{\prod_{j=0,j\neq k}^{r}(x_k-x_j)}\right]\frac{1}{x_r-x_0} = \\
=&\left[\sum_{k=0}^{r}\frac{f_k(x_j-x_0)}{\prod_{j=0,j\neq k}^{r}(x_k-x_j)}-\sum_{k=0}^{r}\frac{f_k(x_j-x_k)}{\prod_{j=0,j\neq k}^{r}(x_k-x_j)}\right]\frac{1}{x_r-x_0} = \\
=&\left[\sum_{k=0}^{r}\frac{f_k(x_j-x_0)-f_k(x_j-x_k)}{\prod_{j=0,j\neq k}^{r}(x_k-x_j)}\right]\frac{1}{x_r-x_0} = \\
=&\left[\sum_{k=0}^{r}\frac{f_k(x_r-x_0)}{\prod_{j=0,j\neq k}^{r}(x_k-x_j)}\right]\frac{1}{x_r-x_0} = \\
=&(x_r-x_0)\left[\sum_{k=0}^{r}\frac{f_k}{\prod_{j=0,j\neq k}^{r}(x_k-x_j)}\right]\frac{1}{x_r-x_0} =
\end{aligned}
$$

$$= \sum_{k=0}^{r} \frac{f_k}{\prod_{j=0,j\neq k}^{r}(x_k - x_j)} = f[x_0, \dots, x_r].$$

□

4.6 Esercizio 6

Costruire una `function` MATLAB *che implementi in modo efficiente l'Algoritmo 4.1*

Soluzione.

```
% f = differenzeDivise(x, f)
% Calcola le differenze divise che costituiscono i coefficienti della
% forma di Newton (rispetto alla base di Newton).
%
% Input:
%   -f: vettore contenente i valori della funzione sulle ascisse di
%   interpolazione;
%   -x: vettore contenente le n+1 ascisse di interpolazione.
% Output:
%   -f: vettore contenente le n+1 differenze divise.
%


function [f] = differenzeDivise(x, f)
    for i=1:length(x)-1
        for j=length(x):-1:i+1
            f(j) = (f(j) - f(j-1))/(x(j)-x(j-i));
        end
    end
end
```

Listing 4.1: Calcolo delle differenze divise.

□

4.7 Esercizio 7

Dimostrare che il seguente algoritmo, che riceve in ingresso i vettori x e f prodotti dalla `function` *dell'Esercizio 4.6, valuta il corrispondente polinomio interpolante di Newton in un punto xx assegnato.*

```
  p = f(n+1)
  for k = n:-1:1
    p = p*(xx-x(k)) +f(k)
  end
```

Quale è il suo costo computazionale? Confrontarlo con quello dell'Algoritmo 4.1. Costruire, quindi, una corrispondente `function` MATLAB che lo implementi efficientemente (complementare la possibilità che xx sia un vettore)

Soluzione.
Essendo un polinomio in forma di Newton esprimibile come

$$p_n(x) = \sum_{k=0}^{n} f[x_0, \ldots, x_k] w_k(x)$$

ed essendo il $(k+1)$-esimo polinomio della base di Newton esprimibile come

$$w_{k+1} = \prod_{j=0}^{k} (x - x_j),$$

vediamo che, come nel caso dell'Esercizio **??**, possiamo raggruppare i vari $(x - x_i)$ e considerare le differenze divise come coefficienti rispetto alla base di Newton. Si ottiene quindi:

$$\sum_{k=0}^{n} f[x_0, \ldots, x_k] w_k(x) = f[x_0] + (x - x_0)(f[x_0, x_1] + \ldots$$
$$\cdots + (x - x_{n-2})(f[x_0, \ldots, x_{n-1}] + f[x_0, \ldots, x_n](x - x_{n-1})) \ldots).$$

Quindi, eseguendo le operazioni nell'ordine indicato, otteniamo l'algoritmo di Horner generalizzato, il cui costo risulta essere di $3n$ `flop`, in quanto ad ogni iterazione vengono eseguiti una sottrazione, un prodotto ed una somma. Il costo dell'Algoritmo per il calcolo delle differenze divise risulta invece essere pari a $(\frac{3}{2}n^2 - 3n)$ `flop`, in quanto per calcolare ogni differenza divisa vengono eseguite due sottrazioni ed una divisione (3 `flop`) ed il numero di differenze divise da calcolare è pari al numero di elementi triangolari inferiori di una matrice $n \times n$ (quindi $\frac{1}{2}n^2$) meno gli elementi della prima colonna (che sono n) che sono dati.
Nell'implementazione MATLAB, proposta di seguito, si è considerato il caso in cui `xx` sia un vettore di punti semplicemente reiterando m volte, con m dimensione del vettore `xx`, l'algoritmo di Horner generalizzato. in questo caso il costo computazionale risulta essere $3mn$ `flop`.

```
% p = hornerGeneralizzato(x, f, xx)
% Algoritmo di Horner generalizzato che valuta un polinomio in forma di
% Newton su un vettore di punti.
%
% Input:
%   -x: vettore contenente le ascisse di interpolazione;
%   -f: vettore contenente le differenze divise;
%   -xx: vettore di punti sui quali valutare il polinomio.
% Output:
%   -p: vettore contenente le valutazioni del polinomio sui punti in
%   xx.
%


function [p] = hornerGeneralizzato(x, f, xx)
    n = length(x)-1;
    p = zeros(length(xx), 1);
    for i=1:length(xx)
```

```
        p(i) = f(n+1);
        for k=n:-1:1
            p = p*(xx(i)-x(k)) +f(k);
        end
    end
end
```

Listing 4.2: Algoritmo di Horner generalizzato.

□

4.8 Esercizio 8

Costruire una `function` MATLAB *che implementi in modo efficiente l'algoritmo del calcolo delle differenze divise per il polinomio di Hermite.*

Soluzione.

```
% f = differenzeDiviseHermite(x, f)
% Calcola le differenze divise che costituiscono i coefficienti della
% forma di Newton (rispetto alla base di Newton) nel caso di ascisse di
% Hermite.
%
% Input:
%   -f: vettore contenente i valori della funzione e la derivata prima
%   sulle ascisse di interpolazione, nella forma [f0, f'0, f1, f'1,
%   ..., fn, f'n];
%   -x: vettore contenente le 2n+2 ascisse di interpolazione di
%   Hermite.
% Output:
%   -f: vettore contenente le 2n+2 differenze divise.
%


function [f] = differenzeDiviseHermite(x, f)
    for i = length(x)-1:-2:3
        f(i) = (f(i)-f(i-2))/(x(i)-x(i-2));
    end
    for i=2:length(x)-1
        for j=length(x):-1:i+1
            f(j) = (f(j)-f(j-1))/(x(j)-x(j-i));
        end
    end
end
```

Listing 4.3: Calcolo delle differenze divise per il polinomio di Hermite.

□

4.9 Esercizio 9

Si consideri la funzione

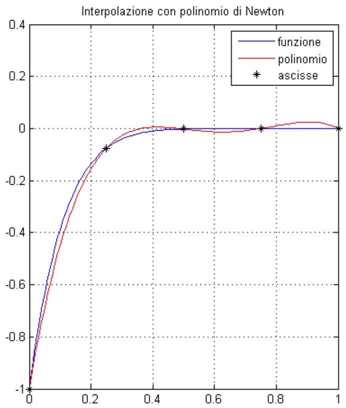
$$f(x) = (x - 1)^9.$$

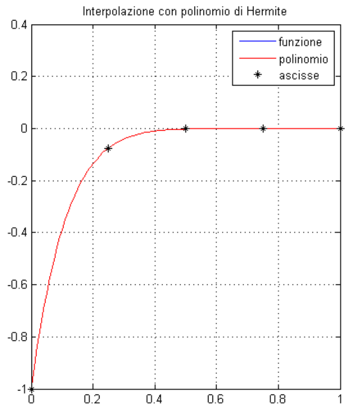
Utilizzando le `function` *degli Esercizi 4.6 e 4.8, valutare i polinomi interpolanti di Newton e di Hermite sulle ascisse*

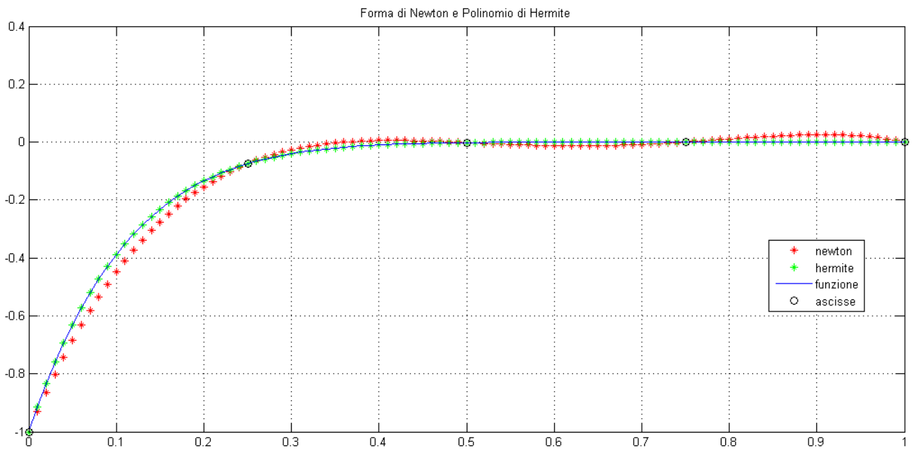
$$0, 0.25, 0.5, 0.75, 1,$$

per x=linspace(0,1,101). Raffigurare, quindi, (e spiegare) i risultati.

Soluzione.
I seguenti grafici mostrano i risultati ottenuti interpolando la funzione sulle ascisse indicate con i polinomi di Newton e di Hermite:







Dai grafici ottenuti si vede che l'interpolazione tramite polinomio di Newton approssima abbastanza bene la funzione, mentre utilizzando il polinomio di Hermite si ha una totale coincidenza tra la funzione originaria ed il polinomio interpolante. Questo è dovuto al fatto che il polinomio di Hermite interpola la funzione su un totale di 10 ascisse, il che lo rende a tutti gli effetti un polinomio di grado 9, esattamente come la funzione originaria. Il polinomio di newton, invece, interpolando la funzione su 5 sole ascisse, risulta essere un polinomio di quarto grado.

```
% Esercizio 4.9
%

format short

f = inline('(x-1).^9');
f1 = inline('9.*(x-1).^8');
ascisse = [0; 0.25; 0.5; 0.75; 1];
fi = f(ascisse);
fiHermite = zeros(2*length(ascisse), 1);
fiHermite(1:2:length(fiHermite)-1) = fi;
fiHermite(2:2:length(fiHermite)) = f1(ascisse);

pn = formaNewton(ascisse, fi);
ph = hermite(ascisse, fiHermite);

xtest = linspace(0,1,101);
figure (1)
h = subplot(1,2,1);
plot(xtest, f(xtest), 'b', xtest, pn(xtest), 'r', ascisse, fi, 'black *')
    ;
grid on
legend('funzione', 'polinomio', 'ascisse')
title('Interpolazione con polinomio di Newton')
subplot(1,2,2);
plot(xtest, f(xtest), 'b', xtest, ph(xtest), 'r', ascisse, fi, 'black *')
    ;
grid on
axis ([get(h, 'XLim') get(h, 'YLim')])
legend('funzione', 'polinomio', 'ascisse')
title('Interpolazione con polinomio di Hermite')

figure (2)
plot(xtest, pn(xtest), 'r *', xtest, ph(xtest), 'g *', xtest, f(xtest), '
    b', ascisse, fi, 'black O');
grid on
legend('newton', 'hermite', 'funzione', 'ascisse')
title('Forma di Newton e Polinomio di Hermite')
```

Listing 4.4: Esercizio ??.

□

4.10 Esercizio 10

Quante ascisse di interpolazione equidistanti sono necessarie per approssimare la funzione $\sin(x)$ *sull'intervallo* $[0, 2\pi]$, *con un errore di interpolazione inferiore a* 10^{-6}*?*

Soluzione.
Sapendo che

$$e(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} w_{n+1}(x)$$

vengono fatte le seguenti maggiorazioni:

$$w_{n+1}(x) = \prod_{i=1}^{n}(x - x_i) \leq \prod_{i=1}^{n}(b - a) = (b - a)^{\mathrm{n}}$$

con $f^{(\mathrm{i})}(x) \leq 1 \forall i = 1, ...n + 1$. Da cui segue:

$$10^{-6} \leq \frac{1(b-a)^{\mathrm{n}}}{(n-1)!} = \frac{2\pi^{\mathrm{n}}}{(n-1)!}$$

Dopo una serie tentativi su Matlab è stato ottenuto il seguente risultato n = 24.

```
% Esercizio 4.10
%


e= inline ('((2* pi ).^( n +1))./( factorial (n +1)) ');
n=0;
while (e(n)>= 10^ -6)
n=n+1;
end
fprintf ('e(%d) = %6.3 e\n', n, e(n));
```

Listing 4.5: Esercizio **??**.

□

4.11 Esercizio 11

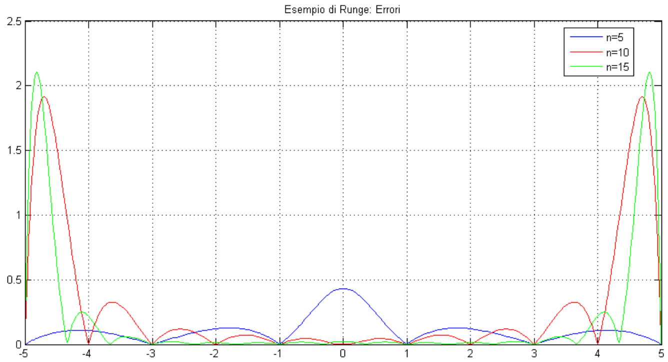
*Verificare sperimentalmente che, considerando le ascisse di interpolazione equidistanti (**??**) su cui si definisce il polinomio $p(x)$ interpolante $f(x)$, l'errore $||f - p||$ diverge, al crescere di n, nei seguenti due casi:*

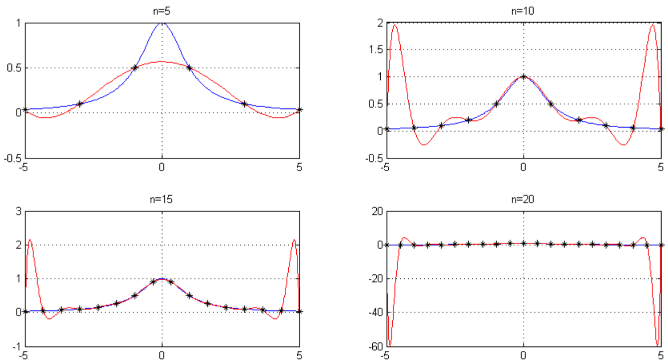
1. *esempio di Runge*:
$$f(x) = \frac{1}{1 + x^2}, \qquad [a, b] \equiv [-5, 5];$$

2. *esempio di Bernstein*:
$$f(x) = |x|, \qquad [a, b] \equiv [-1, 1].$$

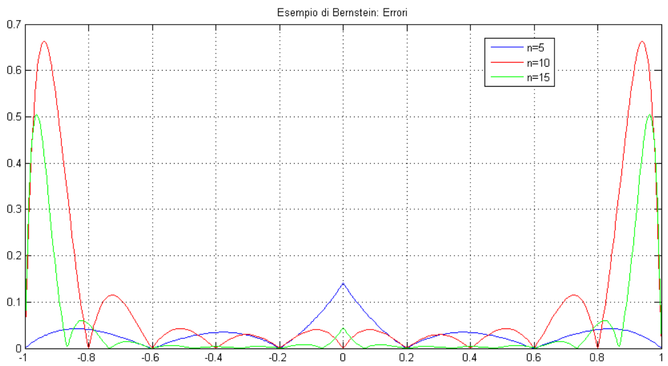
Soluzione.

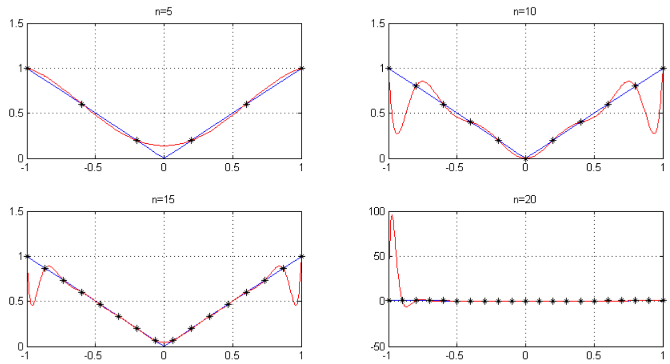
- Esempio di Runge:
Osserviamo i seguenti grafici che mostrano, rispettivamente, l'errore commesso con $n = 5, 10, 15$ e l'interpolazione della funzione per $n = 5, 10, 15, 20$:





- Esempio di Bernstein: Come per l'esempio di Runge, proponiamo i grafici dell'errore e dell'interpolazione per l'esempio di Bernstein:





Da i grafici proposti per le due funzioni d'esempio deduciamo che l'errore commesso tende a divergere all'aumentare di n con le ascisse di interpolazione scelte equidistanti, ovvero uniformemente distribuite sull'intervallo di interpolazione. Vediamo infine i valori dell'errore massimo di interpolazione commesso nei due esempi per $n = 5, 10, 15, 20$:

n	**Errore**	
	Runge	**Bernstein**
5	0.4327	0.0422
10	0.3276	0.1149
15	2.1076	0.5054
20	59.8223	95.1889

```
% Esercizio 4.11
%


fRunge = inline('1./(1.+x.^2)');
fBernstein = inline('abs(x)');
e = inline('abs( feval(f,x) - feval(p,x) )');

disp('Esempio di Runge:')
a=-5; b=5;
n = 5;
ascisse = ascisseEquidistanti(a, b, n);
pRunge = formaNewton(ascisse, fRunge(ascisse));
figure(1)
fplot(@(x)(e(fRunge, pRunge, x)), [a b], 'b');
hold on
grid on
legend('n=5')
title('Esempio di Runge: Errori')
maxErr = e(fRunge, pRunge, fminbnd(@(x)(-e(fRunge, pRunge, x)), a, b));
str = sprintf('Errore massimo con n = %d: %5.4f', n, maxErr); disp(str);
figure (2)
subplot(2,2,1)
fplot(fRunge, [a b], 'b')
hold on
grid on
```

```
fplot(pRunge, [a b], 'r')
plot(ascisse, fRunge(ascisse), 'k *')
title(sprintf('n=%d', n))

n = 10;
ascisse = ascisseEquidistanti(a, b, n);
pRunge = formaNewton(ascisse, fRunge(ascisse));
figure(1)
fplot(@(x)(e(fRunge, pRunge, x)), [a b], 'r');
legend('n=5', 'n=10')
maxErr = e(fRunge, pRunge, fminbnd(@(x)(-e(fRunge, pRunge, x)), a, b));
str = sprintf('Errore massimo con n = %d: %5.4f', n, maxErr); disp(str);
figure (2)
subplot(2,2,2)
fplot(fRunge, [a b], 'b')
hold on
grid on
fplot(pRunge, [a b], 'r')
plot(ascisse, fRunge(ascisse), 'k *')
title(sprintf('n=%d', n))

n = 15;
ascisse = ascisseEquidistanti(a, b, n);
pRunge = formaNewton(ascisse, fRunge(ascisse));
figure(1)
fplot(@(x)(e(fRunge, pRunge, x)), [a b], 'g');
legend('n=5', 'n=10', 'n=15')
maxErr = e(fRunge, pRunge, fminbnd(@(x)(-e(fRunge, pRunge, x)), a, b));
str = sprintf('Errore massimo con n = %d: %5.4f', n, maxErr); disp(str);
figure (2)
subplot(2,2,3)
fplot(fRunge, [a b], 'b')
hold on
grid on
fplot(pRunge, [a b], 'r')
plot(ascisse, fRunge(ascisse), 'k *')
title(sprintf('n=%d', n))

n = 20;
ascisse = ascisseEquidistanti(a, b, n);
pRunge = formaNewton(ascisse, fRunge(ascisse));
maxErr = e(fRunge, pRunge, fminbnd(@(x)(-e(fRunge, pRunge, x)), a, b));
str = sprintf('Errore massimo con n = %d: %5.4f', n, maxErr); disp(str);
figure (2)
subplot(2,2,4)
fplot(fRunge, [a b], 'b')
hold on
grid on
fplot(pRunge, [a b], 'r')
plot(ascisse, fRunge(ascisse), 'k *')
title(sprintf('n=%d', n))

disp(' '), disp('Esempio di Bernstein:')
```

```
a=-1; b=1;
n = 5;
ascisse = ascisseEquidistanti(a, b, n);
pBernstein = formaNewton(ascisse, fBernstein(ascisse));
figure(3)
fplot(@(x)(e(fBernstein, pBernstein, x)), [a b], 'b');
hold on
grid on
legend('n=5')
title('Esempio di Bernstein: Errori')
maxErr = e(fBernstein, pBernstein, fminbnd(@(x)(-e(fBernstein, pBernstein, x)), a, b));
str = sprintf('Errore massimo con n = %d: %5.4f', n, maxErr); disp(str);
figure (4)
subplot(2,2,1)
fplot(fBernstein, [a b], 'b')
hold on
grid on
fplot(pBernstein, [a b], 'r')
plot(ascisse, fBernstein(ascisse), 'k *')
title(sprintf('n=%d', n))

n = 10;
ascisse = ascisseEquidistanti(a, b, n);
pBernstein = formaNewton(ascisse, fBernstein(ascisse));
figure(3)
fplot(@(x)(e(fBernstein, pBernstein, x)), [a b], 'r');
legend('n=5', 'n=10')
maxErr = e(fBernstein, pBernstein, fminbnd(@(x)(-e(fBernstein, pBernstein, x)), a, b));
str = sprintf('Errore massimo con n = %d: %5.4f', n, maxErr); disp(str);
figure (4)
subplot(2,2,2)
fplot(fBernstein, [a b], 'b')
hold on
grid on
fplot(pBernstein, [a b], 'r')
plot(ascisse, fBernstein(ascisse), 'k *')
title(sprintf('n=%d', n))

n = 15;
ascisse = ascisseEquidistanti(a, b, n);
pBernstein = formaNewton(ascisse, fBernstein(ascisse));
figure(3)
fplot(@(x)(e(fBernstein, pBernstein, x)), [a b], 'g');
legend('n=5', 'n=10', 'n=15')
maxErr = e(fBernstein, pBernstein, fminbnd(@(x)(-e(fBernstein, pBernstein, x)), a, b));
str = sprintf('Errore massimo con n = %d: %5.4f', n, maxErr); disp(str);
figure (4)
subplot(2,2,3)
fplot(fBernstein, [a b], 'b')
hold on
```

```
grid on
fplot(pBernstein, [a b], 'r')
plot(ascisse, fBernstein(ascisse), 'k *')
title(sprintf('n=%d', n))

n = 20;
ascisse = ascisseEquidistanti(a, b, n);
pBernstein = formaNewton(ascisse, fBernstein(ascisse));
maxErr = e(fBernstein, pBernstein, fminbnd(@(x)(-e(fBernstein, pBernstein
    , x)), a, b));
str = sprintf('Errore massimo con n = %d: %5.4f', n, maxErr); disp(str);
figure (4)
subplot(2,2,4)
fplot(fBernstein, [a b], 'b')
hold on
grid on
fplot(pBernstein, [a b], 'r')
plot(ascisse, fBernstein(ascisse), 'k *')
title(sprintf('n=%d', n))
```

Listing 4.6: Esercizio **??**.

□

4.12 Esercizio 12

Dimostrare che, se $x \in [-1, 1]$*, allora:*

$$\tilde{x} \equiv \frac{a+b}{2} + \frac{b-a}{2} x \in [a, b].$$

Viceversa, se $\tilde{x} \in [a, b]$*, allora:*

$$x \equiv \frac{2\tilde{x} - a - b}{b - 1} \in [-1, 1].$$

Concludere che è sempre possibile trasformare il problema di interpolazione (4.1)-(4.2) in uno definito sull'intervallo $[-1, 1]$*, e viceversa.*

Soluzione.

- $x \in [-1, 1] \Rightarrow \tilde{x} \in [a, b]$:
  - se $x = -1$, $\tilde{x} = \frac{a+b}{2} - \frac{b-a}{2} = a$;
  - se $x = 1$, $\tilde{x} = \frac{a+b}{2} + \frac{b-a}{2} = b$.
- $\tilde{x} \in [a, b] \Rightarrow x \in [-1, 1]$:
  - se $\tilde{x} = a$, $x = \frac{2a-a-b}{b-a} = -1$;
  - se $\tilde{x} = b$, $x = \frac{2b-a-b}{b-a} = 1$;

□

4.13 Esercizio 13

Dimostrare le proprietà dei polinomi di Chebyshev di I specie (4.24) elencate nel Teorema 4.9.

Soluzione.
Prime proprietà

1. $T_k(x)$ è un polinomio di grado esatto k:
   - $k = 0$: $T_0(x) = 1$ polinomio di grado 0;
   - $k > 0$: $T_{k+1}(x) = 2xT_k(x) - T_{k-1}(x)$ dove, per ipotesi induttiva, $T_k(x)$ è un polinomio di grado k quindi $T_{k+1}(x)$ è un polinomio di grado $k + 1$.
2. Il coefficiente principale di $T_k(x)$ è 2^{k-1}, $k = 1, 2, \ldots$:
   - $k = 1$: $T_1(x) = x$ e $2^{1-1} = 1$;
   - $k > 1$: $T_{k+1}(x) = 2xT_k(x) - T_{k-1}(x)$ dove, per ipotesi induttiva, il coefficiente principale di $T_k(x)$ è 2^{k-1} quindi il coefficiente principale di $T_{k+1}(x)$ è $2 \cdot 2^{k-1} = 2^k$.
3. La famiglia di polinomi $\{\hat{T}_k\}$, in cui
$$\hat{T}_0(x) = T_0(x), \quad \hat{T}_k(x) = 2^{1-k}T_k(x), \quad k = 1, 2, \ldots,$$
   è una famiglia di polinomi monici di grado k, $k = 1, 2, \ldots$:
   - grado k: il grado di $\hat{T}_k$ coincide con il grado di $T_k(x)$ che è k;
   - monici: il coefficiente principale di $\hat{T}_k$ è 2^{1-k} dunque $2^{1-k}2^{k-1} = 1$ il polinomio è monico.
4. Ponendo $x = \cos\theta$, $\theta \in [0, \pi]$, si ottiene $T_k(x) = T_k(\cos\theta) = \cos k\theta, \quad k = 0, 1, \ldots$:
   - $k = 0$: $T_0(\cos\theta) = \cos 0\theta = 1 = T_0(x)$;
   - $k = 1$: $T_0(\cos\theta) = \cos\theta = x = T_1(x)$:
   - $k > 1$: $T_{k+1}(\cos\theta) = 2\cos\theta T_k(\cos\theta) - T_{k-1}(\cos\theta)$ per ipotesi induttiva, $T_k(\cos\theta) = \cos k\theta$ e $T_{k-1}(\cos\theta) = \cos(k-1)\theta$ dunque $T_{k+1}(\cos\theta) = 2\cos\theta\cos k\theta - \cos(k-1)\theta =$ $\cos(k+1)\theta + \cos(k-1)\theta - \cos(k-1)\theta = \cos(k+1)\theta = T_{k+1}(x)$.

Teorema 4.9

- Radici del polinomio: $T_k(x) = T_k(\cos\theta) = \cos k\theta = 0$ se $\cos k\theta = 0$ ovvero per $k\theta = \frac{\pi}{2} + i\pi$; segue $\theta_i = \frac{\frac{\pi}{2}+i\pi}{k} = \frac{(2i+1)\pi}{2k}$ cioè gli zeri del polinomio sono dati da $x_i^{(k)} = \cos\frac{(2i+1)\pi}{2k}$.
- Estremi: Agli estremi $\cos k\theta = \pm 1$ ovvero $k\theta = i\pi$; segue $\theta_i = \frac{i}{k}\pi$ dunque gli estremi sono assunti in $\xi_i^{(k)} = \cos\frac{i}{k}\pi$; in tali punti, poiché $\cos k\pi = (-1)^i$ la funzione vale $T_k(\xi_i^{(k)}) = (-1)^i$.
- Norme: dal punto precedente segue inoltre $||T_k|| = 1$ e $||\hat{T}_k|| = ||2^{1-k}T_k|| = ||2^{1-k}||$.

- Minima norma per $\hat{T}_k(x)$: supponiamo per assurdo che $\exists p \neq \hat{T}_k(x)$ monico tale che $||p|| < 2^{1-k} = ||\hat{T}_k||$, quindi $g(x) = \hat{T}_k - p(x) \in \Pi_{k-1}$ poiché entrambi monici di grado K. Studiando il segno di g si nota $sign(g(x_i)) = (-1)^i$ per $i = 0, \ldots, k$ ovvero ci sono k cambiamenti di segno quindi k radici cioè $g(x) \in \Pi_k$ ma, per ipotesi, $g(x) \in \Pi_{k-1}$ quindi $g(x) = 0$.

□

4.14 Esercizio 14

Quali diventano le ascisse di Chebyshev (4.26), per un problema definito su un generico intervallo $[a, b]$*?*

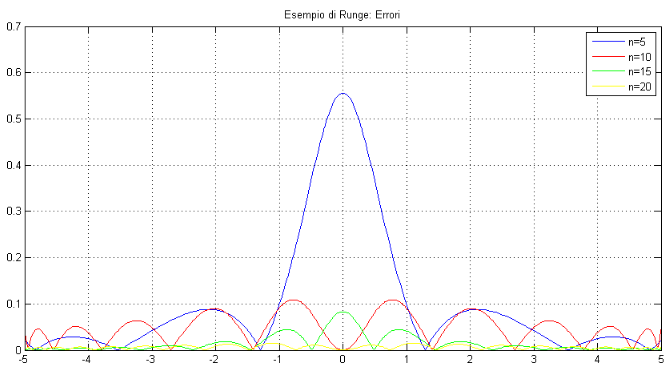
Soluzione.
Nel caso a=-1 e b=1, la formula per il calcolo delle ascisse di Chebyshev è: $x_i^{(k)} = cos(\frac{(2i+1)\pi}{2k})$ con k grado del polinomio e $i = 0, ..., k$.
Nel caso generico, la formula diventa: $x_i^{(k)} = \frac{a+b}{2} + \frac{b-a}{2} cos(\frac{(2i+1)\pi}{2k})$ con k grado del polinomio e $i = 0, ..., k$. □

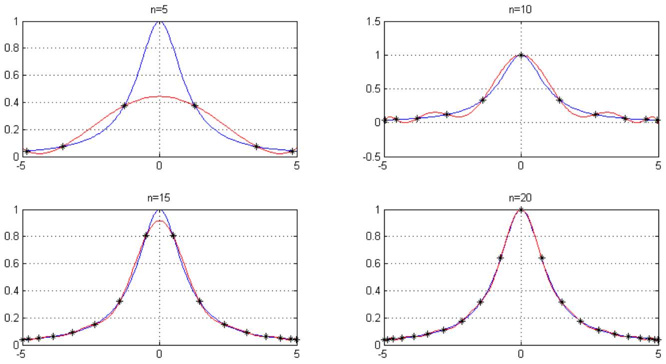
4.15 Esercizio 15

Utilizzare le ascisse di Chebyshev (4.26) per approssimare gli esempi visti nell'Esercizio 4.11, per $n = 2, 4, 6, \ldots, 40$.

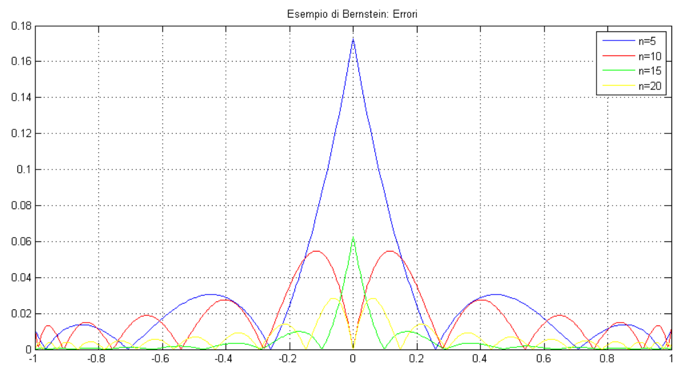
Soluzione.
Similmente a quanto visto nell'Esercizio 4.11 mostriamo di seguito i grafici degli errori e di interpolazione per $n = 5, 10, 15, 20$, rispetto agli esempi di Runge e Bernstein.

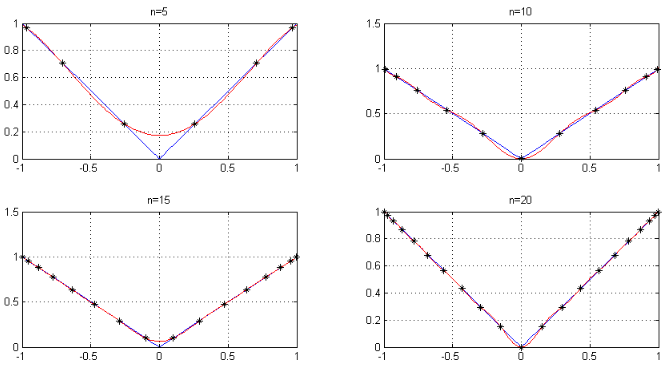
- Esempio di Runge:





- Esempio di Bernstein:





Di seguito proponiamo anche gli errori di interpolazione massimi commessi nei due esempi per $n = 5, 10, 15, 20$:

n	Errore	
	Runge	**Bernstein**
5	0.0881	0.0305
10	0.0897	0.0275
15	0.0831	0.0628
20	0.0153	0.0140

Si evince quindi, dai grafici e dai valori riportati, che la scelta delle ascisse di Chebyshev al posto di ascisse equidistanti è estremamente più conveniente, in quanto evita la divergenza dell'errore di interpolazione all'aumentare di n. Infatti l'errore commesso, all'aumentare di n risulta essere in generale molto stabile e, molto spesso, risulta anche in diminuzione.

```
% Esercizio 4.15
%


fRunge = inline('1./(1.+x.^2)');
fBernstein = inline('abs(x)');
e = inline('abs( feval(f,x) - feval(p,x) )');

disp('Esempio di Runge:')
a=-5; b=5;
n = 5;
ascisse = ascisseChebyshev(a, b, n);
pRunge = formaNewton(ascisse, fRunge(ascisse));
figure(1)
fplot(@(x)(e(fRunge, pRunge, x)), [a b], 'b');
hold on
grid on
legend('n=5')
title('Esempio di Runge: Errori')
maxErr = e(fRunge, pRunge, fminbnd(@(x)(-e(fRunge, pRunge, x)), a, b));
str = sprintf('Errore massimo con n = %d: %5.4f', n, maxErr); disp(str);
figure (2)
subplot(2,2,1)
fplot(fRunge, [a b], 'b')
hold on
grid on
fplot(pRunge, [a b], 'r')
plot(ascisse, fRunge(ascisse), 'k *')
title(sprintf('n=%d', n))

n = 10;
ascisse = ascisseChebyshev(a, b, n);
pRunge = formaNewton(ascisse, fRunge(ascisse));
figure(1)
fplot(@(x)(e(fRunge, pRunge, x)), [a b], 'r');
legend('n=5', 'n=10')
maxErr = e(fRunge, pRunge, fminbnd(@(x)(-e(fRunge, pRunge, x)), a, b));
str = sprintf('Errore massimo con n = %d: %5.4f', n, maxErr); disp(str);
figure (2)
subplot(2,2,2)
fplot(fRunge, [a b], 'b')
hold on
```

```
grid on
fplot(pRunge, [a b], 'r')
plot(ascisse, fRunge(ascisse), 'k *')
title(sprintf('n=%d', n))

n = 15;
ascisse = ascisseChebyshev(a, b, n);
pRunge = formaNewton(ascisse, fRunge(ascisse));
figure(1)
fplot(@(x)(e(fRunge, pRunge, x)), [a b], 'g');
legend('n=5', 'n=10', 'n=15')
maxErr = e(fRunge, pRunge, fminbnd(@(x)(-e(fRunge, pRunge, x)), a, b));
str = sprintf('Errore massimo con n = %d: %5.4f', n, maxErr); disp(str);
figure (2)
subplot(2,2,3)
fplot(fRunge, [a b], 'b')
hold on
grid on
fplot(pRunge, [a b], 'r')
plot(ascisse, fRunge(ascisse), 'k *')
title(sprintf('n=%d', n))

n = 20;
ascisse = ascisseChebyshev(a, b, n);
pRunge = formaNewton(ascisse, fRunge(ascisse));
figure(1)
fplot(@(x)(e(fRunge, pRunge, x)), [a b], 'y');
legend('n=5', 'n=10', 'n=15', 'n=20')
maxErr = e(fRunge, pRunge, fminbnd(@(x)(-e(fRunge, pRunge, x)), a, b));
str = sprintf('Errore massimo con n = %d: %5.4f', n, maxErr); disp(str);
figure (2)
subplot(2,2,4)
fplot(fRunge, [a b], 'b')
hold on
grid on
fplot(pRunge, [a b], 'r')
plot(ascisse, fRunge(ascisse), 'k *')
title(sprintf('n=%d', n))

disp(' '), disp('Esempio di Bernstein:')
a=-1; b=1;
n = 5;
ascisse = ascisseChebyshev(a, b, n);
pBernstein = formaNewton(ascisse, fBernstein(ascisse));
figure(3)
fplot(@(x)(e(fBernstein, pBernstein, x)), [a b], 'b');
hold on
grid on
legend('n=5')
title('Esempio di Bernstein: Errori')
maxErr = e(fBernstein, pBernstein, fminbnd(@(x)(-e(fBernstein, pBernstein
    , x)), a, b));
str = sprintf('Errore massimo con n = %d: %5.4f', n, maxErr); disp(str);
```

```
figure (4)
subplot(2,2,1)
fplot(fBernstein, [a b], 'b')
hold on
grid on
fplot(pBernstein, [a b], 'r')
plot(ascisse, fBernstein(ascisse), 'k *')
title(sprintf('n=%d', n))

n = 10;
ascisse = ascisseChebyshev(a, b, n);
pBernstein = formaNewton(ascisse, fBernstein(ascisse));
figure(3)
fplot(@(x)(e(fBernstein, pBernstein, x)), [a b], 'r');
legend('n=5', 'n=10')
maxErr = e(fBernstein, pBernstein, fminbnd(@(x)(-e(fBernstein, pBernstein
    , x)), a, b));
str = sprintf('Errore massimo con n = %d: %5.4f', n, maxErr); disp(str);
figure (4)
subplot(2,2,2)
fplot(fBernstein, [a b], 'b')
hold on
grid on
fplot(pBernstein, [a b], 'r')
plot(ascisse, fBernstein(ascisse), 'k *')
title(sprintf('n=%d', n))

n = 15;
ascisse = ascisseChebyshev(a, b, n);
pBernstein = formaNewton(ascisse, fBernstein(ascisse));
figure(3)
fplot(@(x)(e(fBernstein, pBernstein, x)), [a b], 'g');
legend('n=5', 'n=10', 'n=15')
maxErr = e(fBernstein, pBernstein, fminbnd(@(x)(-e(fBernstein, pBernstein
    , x)), a, b));
str = sprintf('Errore massimo con n = %d: %5.4f', n, maxErr); disp(str);
figure (4)
subplot(2,2,3)
fplot(fBernstein, [a b], 'b')
hold on
grid on
fplot(pBernstein, [a b], 'r')
plot(ascisse, fBernstein(ascisse), 'k *')
title(sprintf('n=%d', n))

n = 20;
ascisse = ascisseChebyshev(a, b, n);
pBernstein = formaNewton(ascisse, fBernstein(ascisse));
figure(3)
fplot(@(x)(e(fBernstein, pBernstein, x)), [a b], 'y');
legend('n=5', 'n=10', 'n=15', 'n=20')
maxErr = e(fBernstein, pBernstein, fminbnd(@(x)(-e(fBernstein, pBernstein
    , x)), a, b));
```

```
str = sprintf('Errore massimo con n = %d: %5.4f', n, maxErr); disp(str);
figure (4)
subplot(2,2,4)
fplot(fBernstein, [a b], 'b')
hold on
grid on
fplot(pBernstein, [a b], 'r')
plot(ascisse, fBernstein(ascisse), 'k *')
title(sprintf('n=%d', n))
```

Listing 4.7: Esercizio **??**.

□

4.16 Esercizio 16

Verificare che la fattorizzazione LU della matrice dei coefficienti del sistema tridiagonale (4.40) è dato da:

$$L = \begin{pmatrix} 1 & & & \\ l_2 & 1 & & \\ & \ddots & \ddots & \\ & & l_{n-1} & 1 \end{pmatrix}, \qquad U = \begin{pmatrix} u_1 & \xi_1 & & \\ & u_2 & \ddots & \\ & & \ddots & \xi_{n-2} \\ & & & u_{n-1} \end{pmatrix},$$

con

$$\begin{aligned} u_1 &= 2, \\ l_i &= \frac{\varphi_i}{u_{i-1}}, \\ u_i &= 2 - l_i \xi_{i-1}, \qquad i = 2, \dots, n-1. \end{aligned}$$

Scrivere una `function` MATLAB *che implementi efficientemente la risoluzione della (4.40).*

Soluzione.
La matrice dei coefficienti

$$A = \begin{pmatrix} 2 & \xi_1 & & & \\ \varphi_2 & 2 & \xi_2 & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & \xi_{n-2} \\ & & & \varphi_{n-1} & 2 \end{pmatrix}$$

si vede essere *dominante per righe* in quanto, essendo

$$\varphi_i = \frac{h_i}{h_i + h_{i+1}}, \qquad \xi_i = \frac{h_{i+1}}{h_i + h_{i+1}}, \qquad i = 1, \dots, n-1,$$

si vede facilmente che $\varphi_i + \xi_i = 1$. Quindi la matrice è fattorizzabile LU.
Moltiplichiamo adesso i termini L ed U e vediamo che il risultato sono proprio i termini della matrice A:

- $a_{11} = 1 \cdot u_1 = u_1 = 2$;
- $a_{12} = 1 \cdot \xi_1 + 0 \cdot u_2 = \xi_1$;
- $a_{i,i-1} = 0 \cdot \xi_{i-2} + l_i \cdot u_{i-1} + 1 \cdot 0 = l_i \cdot u_{i-1} = \frac{\varphi_i}{u_{i-1}} u_{i-1} = \varphi_i$, per $i > 1$;
- $a_{ii} = l_i \cdot \xi_{i-1} + 1 \cdot u_i = l_i \xi_{i-1} + 2 - l_i \xi_{i-1} = 2$, per $i > 1$;
- $a_{i,i+1} = l_i \cdot 0 + 1 \cdot \xi_i + 0 \cdot u_{i+1} = \xi_i$, per $i > 1$.

Quindi il sistema $A\underline{m} = \underline{d}$ si risolve come segue:

- si risolve $L\underline{y} = 6\underline{d}$:
  - $y_1 = 6d_1$,
  - $y_i = 6d_i - l_i y_{i-1}$ per $i = 2, \ldots, n-1$;
- si risolve il sistema $U\underline{m} = \underline{y}$:
  - $m_{n-1} = \frac{y_{n-1}}{u_{n-1}}$,
  - $m_i = \frac{y_i - \xi_i m_{i+1}}{u_i}$, per $i = n-2, \ldots, 1$.

```
% m = risolviSistemaSplineNaturale(phi, xi, dd)
% Risoluzione del sistema lineare di una spline cubica naturale per la
% determinazione dei fattori m_i necessari per la costruzione
% dell'espressione della spline cubica naturale.
%
% Input:
%   -phi: vettore dei fattori phi che definiscono la matrice dei
%   coefficienti (lunghezza n-1);
%   -xi: vettore dei fattori xi che definiscono la matrice dei
%   coefficienti (lunghezza n-1);
%   -dd: vettore delle differenze divise (lunghezza n-1).
% Output:
%   -m: vettore contenente gli n-1 fattori m_i calcolati.
%


function [m] = risolviSistemaSplineNaturale(phi, xi, dd)
    dd = 6*dd;
    n = length(xi)+1;
    u = zeros(n-1, 1);
    l = zeros(n-2, 1);
    u(1)=2;
    for i=2:n-1
        l(i)=phi(i)/u(i-1);
        u(i)=2-l(i)*xi(i-1);
    end
    y = zeros(n-1, 1);
    y(1)=dd(1);
    for i=2:n-1
        y(i)=dd(i)-l(i)*y(i-1);
    end
    m = zeros(n-1, 1);
```

```
    m(n-1)=y(n-1)/u(n-1);
    for i=n-2:-1:1
        m(i)=(y(i)-xi(i)*dd(i+1))/u(i);
    end
    m = [0; m; 0];
end
```

Listing 4.8: Calcolo dei fattori $\{m_i\}$ per una *spline* cubica naturale.

□

4.17 Esercizio 17

Generalizzare la fattorizzazione del precedente Esercizio 4.16 al caso della matrice dei coefficienti del sistema lineare (4.41). Scrivere una corrispondente `function` MATLAB *che risolva efficientemente questo sistema.*

Soluzione.
Generalizzando il risultato ottenuto nell'Esercizio 4.16, la fattorizzazione è della forma

$$L = \begin{pmatrix} 1 & & & \\ l_2 & 1 & & \\ & \ddots & \ddots & \\ & & l_{n+1} & 1 \end{pmatrix}, \qquad U = \begin{pmatrix} u_1 & w_1 & & \\ & u_2 & \ddots & \\ & & \ddots & w_n \\ & & & u_{n+1} \end{pmatrix}.$$

Se come prima moltiplichiamo i fattori L ed U ricaviamo le espressioni degli l_i, u_i e w_i:

- per $i = 4, \dots, n-1$:
$$\begin{cases} u_i = 2 - l_i w_{i-1} \\ w_{i-1} = \xi_{i-2} \\ l_i = \frac{\varphi_{i-1}}{u_{i-1}} \end{cases};$$

- per $i = 3$:
$$\begin{cases} u_3 = 2 - l_3 w_2 \\ w_2 = \xi_1 - \varphi_1 \\ l_3 = \frac{\varphi_2}{u_2} \end{cases};$$

- per $i = 2$:
$$\begin{cases} u_2 = 2 - \varphi_1 \\ w_1 = 0 \\ l_2 = \frac{\varphi_1}{u_1} \end{cases};$$

- per $i = 1$:
$$u_1 = 1;$$

- per $i = n$:
$$\begin{cases} u_n = 2 - \xi_{n-1} - l_n w_{n-1} \\ w_{n-1} = \xi_{n-2} \\ l_n = \frac{\varphi_{n-1} - \xi_{n-1}}{u_{n-1}} \end{cases};$$
- per $i = n+1$:
$$\begin{cases} u_{n+1} = 1 \\ w_n = \xi_{n-1} \\ l_{n+1} = 0 \end{cases}$$
.

Quindi come prima avremo:

- risoluzione del sistema $L\underline{y} = 6\underline{d}$:
  - $y_1 = 6d_1$,
  - $y_i = 6d_i - l_i y_{i-1}$ per $i = 2, \ldots, n+1$;
- risoluzione del sistema $U\underline{\hat{m}} = \underline{y}$:
  - $\hat{m}_{n+1} = \frac{y_{n+1}}{u_{n+1}}$,
  - $\hat{m}_i = \frac{y_i - w_i \hat{m}_{i+1}}{u_i}$, per $i = n, \ldots, 1$;
- calcolo della soluzione:
  - $m_1 = \hat{m}_1 - \hat{m}_2 - \hat{m}_3$,
  - $m_i = \hat{m}_i$, per $i = 2, \ldots, n$,
  - $m_{n+1} = \hat{m}_{n+1} - \hat{m}_n - \hat{m}_{n-1}$.

Qui si propone il codice:

```
% m = risolviSistemaSplineNotAKnot(phi, xi, dd)
% Risoluzione del sistema lineare di una spline cubica con condizioni
% not-a-knot per la determinazione dei fattori m_i necessari per la
% costruzione dell'espressione della spline cubica not-a-knot.
%
% Input:
%   -phi: vettore dei fattori phi che definiscono la matrice dei
%   coefficienti (lunghezza n-1);
%   -xi: vettore dei fattori xi che definiscono la matrice dei
%   coefficienti (lunghezza n-1);
%   -dd: vettore delle differenze divise (lunghezza n-1).
% Output:
%   -m: vettore riscritto con gli n+1 fattori m_i calcolati.
%


function [m] = risolviSistemaSplineNotAKnot(phi, xi, dd)
    dd=[6*dd(1); 6*dd; 6*dd(length(dd))];
    l=zeros(length(xi)+1, 1);
    u=zeros(length(xi)+2, 1);
    w=zeros(length(xi)+1, 1);
```

```
    u(1)=1;
    w(1)=0;
    l(1)=phi(1)/u(1);
    u(2)=2-phi(1);
    w(2)=xi(1)-phi(1);
    l(2)=phi(2)/u(2);
    u(3)=2-l(2)*w(2);
    for i=4:length(xi)
        w(i-1)=xi(i-2);
        l(i-1)=phi(i-1)/u(i-1);
        u(i)=2-l(i-1)*w(i-1);
    end
    w(length(xi))=xi(length(xi)-1);
    l(length(xi))=(phi(length(xi))-xi(length(xi)))/u(length(xi));
    u(length(xi)+1)=2-xi(length(xi))-l(length(xi)-1)*w(length(xi)-1);
    w(length(xi)+1)=xi(length(xi));
    l(length(xi)+1)=0;
    u(length(xi)+2)=1;

    y=zeros(length(xi)+2, 1);
    y(1)=dd(1);
    for i=2:length(xi)+2
        y(i)=dd(i)-l(i-1)*y(i-1);
    end
    m=zeros(length(xi)+2, 1);
    m(length(xi)+2)=y(length(xi)+2)/u(length(xi)+2);
    for i=length(xi)+1:-1:1
        m(i)=(y(i)-w(i)*m(i+1))/u(i);
    end
    m(1)=m(1)-m(2)-m(3);
    m(length(xi)+2)=m(length(xi)+2)-m(length(xi)+1)-m(length(xi));
end
```

Listing 4.9: Calcolo dei fattori $\{m_i\}$ per una *spline* cubica *not-a-knot*.

□

4.18 Esercizio 18

Scrivere una `function` MATLAB *che, noti gli* $\{m_i\}$ *in (4.32), determini l'espressione, polinomiale a tratti, della spline cubica (4.36).*

Soluzione.

```
% s = espressioniSplineCubica(ptx, fi, mi)
% Calcola le espressioni degli n polinomi costituenti una spline
% cubica.
%
% Input:
%   -ptx: vettore contenente gli n+1 nodi di interpolazione;
%   -fi: vettore contenente i valori assunti dalla funzione da
%   approssimare nei nodi in ptx;
```

```
%   -mi: fattori m_i calcolati risolvendo il sistema lineare
%   corrispondente.
% Output:
%   -s: vettore contenente le espressioni degli n polinomi che
%   definiscono la spline cubica.
%


function [s] = espressioniSplineCubica(ptx, fi, mi)
    s = sym('x', [length(ptx)-1 1]);
    syms x;
    for i=2:length(ptx)
        hi = ptx(i)-ptx(i-1);
        ri = fi(i-1)-((hi^2)/6)*mi(i-1);
        qi = (fi(i)-fi(i-1))/hi -(hi/6)*(mi(i)-mi(i-1));
        s(i-1)=(((x - ptx(i-1))^3)*mi(i) + ((ptx(i) - x)^3)*mi(i-1))/(6*hi) +qi*(x - ptx(i-1)) +ri;
    end
end
```

Listing 4.10: Calcolo delle espressioni di una *spline* (noti i fattori $\{m_i\}$).

□

4.19 Esercizio 19

Costruire una `function` MATLAB *che implementi le spline cubiche naturali e quelle definite dalle condizioni not-a-knot.*

Soluzione.

```
% s = splineCubica(ptx, fi, nak)
% Determina le espressioni degli n polinomi che formano una spline
% cubica naturale o con condizioni not-a-knot.
%
% Input:
%   -ptx: vettore contenente gli n+1 nodi di interpolazione;
%   -fi: vettore contenente i valori assunti dalla funzione da
%   approssimare nei nodi in ptx;
%   -nak: true se la spline implementa condizioni not-a-knot, false se
%   invece e' una spline naturale.
% Output:
%   -s: il vettore contenente le n espressioni dei polinomi costituenti
%   la spline.
%


function [s] = splineCubica(ptx, fi, nak)
    phi = zeros(length(ptx)-2, 1);
    xi = zeros(length(ptx)-2, 1);
    dd = zeros(length(ptx)-2, 1);
    for i=2:length(ptx)-1
```

```
        hi = ptx(i) - ptx(i-1);
        hi1 = ptx(i+1) - ptx(i);
        phi(i) = hi/(hi+hi1);
        xi(i) = hi1/(hi+hi1);
        dd(i) = differenzaDivisa(ptx(i-1:i+1), fi(i-1:i+1));
    end
    if nak
        mi = risolviSistemaSplineNotAKnot(phi, xi, dd);
    else
        mi = risolviSistemaSplineNaturale(phi, xi, dd);
    end
    s = espressioniSplineCubica(ptx, fi, mi);
end
```

Listing 4.11: Calcolo delle espressioni di una *spline* (naturale o con condizioni *not-a-knot*).

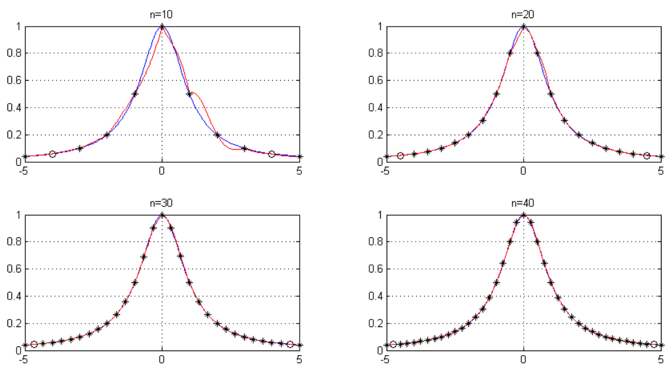
□

4.20 Esercizio 20

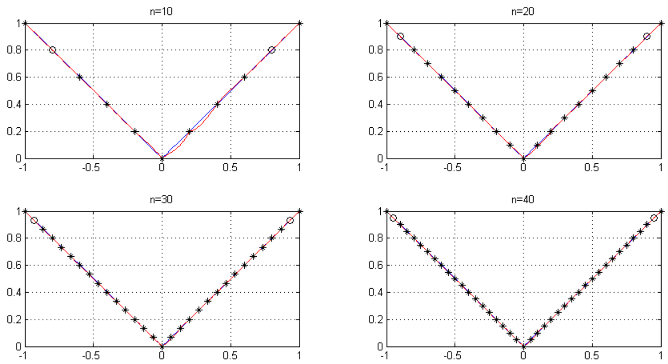
Utilizzare la `function` *dell'Esercizio 4.19 per approssimare, su partizioni (4.28) uniformi con* $n = 10, 20, 30, 40$*, gli esempi proposti nell'Esercizio 4.11.*

Soluzione.
I grafici ottenuti mostrano l'interpolazione delle due funzioni d'esempio tramite spline cubiche con condizioni *not-a-knot* (si osservi che nei grafici i due punti definiti, appunto, *not-a-knot*, sono indicati da un cerchio nero al posto di un asterisco come per tutti gli altri nodi). Non sono stati riportati i grafici riguardanti l'interpolazione tramite spline cubiche naturali in quanto tra i due tipi di grafici non sono presenti sostanziali differenze (eseguendo il file MATLAB relativo vengono disegnati entrambi i grafici per i due esempi).

- Esempio di Runge:



- Esempio di Bernstein:



Qui si propone il codice:

```
% Esercizio 4.20
%


fRunge = inline('1./(1.+x.^2)');
fBernstein = inline('abs(x)');

for i=1:4
    n=i*10;

    a=-5; b=5;
    ascisse = ascisseEquidistanti(a, b, n);
    fi = fRunge(ascisse);
    sN = splineCubica(ascisse, fi, false);
    sNaK = splineCubica(ascisse, fi, true);
    figure (1)
    h = subplot(2,2,i);
    fplot(fRunge, [a b], 'b')
    lims = [get(h, 'XLim') get(h, 'YLim')];
    hold on
    grid on
    for j=1:n
        fplot(inline(sN(j)), [ascisse(j) ascisse(j+1)], 'r')
    end
    axis(lims)
    plot(ascisse, fi, 'k *')
    title(sprintf('n=%d', n))
    figure (2)
    h = subplot(2,2,i);
    fplot(fRunge, [a b], 'b')
    lims = [get(h, 'XLim') get(h, 'YLim')];
    hold on
    grid on
    for j=1:n
        fplot(inline(sNaK(j)), [ascisse(j) ascisse(j+1)], 'r')
    end
    axis(lims)
    plot(ascisse(1), fi(1), 'k *', ascisse(n+1), fi(n+1), 'k *')
```

```
    plot(ascisse(2), fi(2), 'k O', ascisse(n), fi(n), 'k O')
    plot(ascisse(3:n-1), fi(3:n-1), 'k *')
    title(sprintf('n=%d', n))

    a=-1; b=1;
    ascisse = ascisseEquidistanti(a, b, n);
    fi = fBernstein(ascisse);
    sN = splineCubica(ascisse, fi, false);
    sNaK = splineCubica(ascisse, fi, true);
    figure (3)
    h = subplot(2,2,i);
    fplot(fBernstein, [a b], 'b')
    lims = [get(h, 'XLim') get(h, 'YLim')];
    hold on
    grid on
    for j=1:n
        fplot(inline(sN(j)), [ascisse(j) ascisse(j+1)], 'r')
    end
    axis(lims)
    plot(ascisse, fi, 'k *')
    title(sprintf('n=%d', n))
    figure (4)
    h = subplot(2,2,i);
    fplot(fBernstein, [a b], 'b')
    lims = [get(h, 'XLim') get(h, 'YLim')];
    hold on
    grid on
    for j=1:n
        fplot(inline(sNaK(j)), [ascisse(j) ascisse(j+1)], 'r')
    end
    axis(lims)
    plot(ascisse(1), fi(1), 'k *', ascisse(n+1), fi(n+1), 'k *')
    plot(ascisse(2), fi(2), 'k O', ascisse(n), fi(n), 'k O')
    plot(ascisse(3:n-1), fi(3:n-1), 'k *')
    title(sprintf('n=%d', n))
end
```

Listing 4.12: Esercizio ??.

□

4.21 Esercizio 21

Interpretare la retta dell'Esercizio 3.32 come retta di approssimazione ai minimi quadrati dei dati.

Soluzione.
Il problema ai minimi quadrati è dato da

$$\min_{a_1,a_2\in\mathbb{R}} \sum_{k=0}^{n} |y_i - a_1 x_i - a_2|^2$$

dove $\underline{y}$ è il vettore dei valori previsti per la retta. Il problema è esprimibile in forma matriciale come

$$\begin{pmatrix} 1 & x_0 \\ 1 & x_1 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix} \begin{pmatrix} a_2 \\ a_1 \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{pmatrix};$$

si tratta di risolvere un sistema sovradeterminato che ha soluzione se almeno 2 ascisse sono distinte in quanto $r(x) \in \Pi_1$. □

4.22 Esercizio 22

È noto che un fenomeno ha un decadimento esponenziale, modellizzato come

$$y = \alpha \cdot e^{-\lambda t},$$

in cui α e λ sono parametri positivi e incogniti. Riformulare il problema in modo che il modello sia di tipo polinomiale. Supponendo inoltre di disporre delle seguenti misure,

t_1	0	1	2	3	4	5	6	7	8	9	10
y_i	5.22	4.00	4.28	3.89	3.53	3.12	2.73	2.70	2.20	2.08	1.94

calcolare la stime ai minimi quadrati dei due parametri incogniti. Valutare il residuo e raffigurare, infine, i risultati ottenuti.

Soluzione.
Il problema in forma polinomiale è

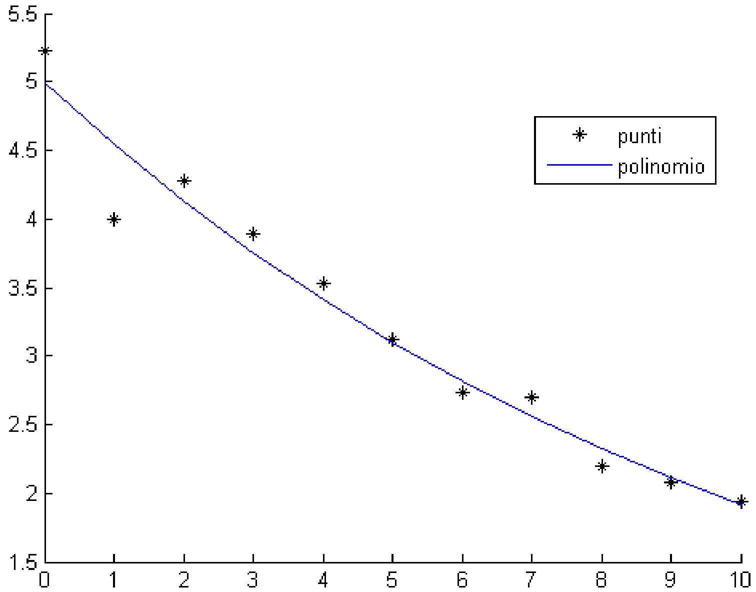
$$\bar{y} = \bar{\alpha} + \bar{\lambda} t, \qquad \text{con } \bar{y} = \log y,\ \bar{\alpha} = \log \alpha \text{ e } \bar{\lambda} = -\lambda$$

infatti si ha $y = \alpha \cdot e^{-\lambda t} \Rightarrow \log y = \log\left(\alpha \cdot e^{-\lambda t}\right) \Rightarrow \log y = \log \alpha - \lambda t \Rightarrow \bar{y} = \bar{\alpha} + \bar{\lambda} t$. Si tratta di risolvere il sistema lineare sovradeterminato

$$\begin{pmatrix} 1 & t_0 \\ 1 & t_1 \\ \vdots & \vdots \\ 1 & t_10 \end{pmatrix} \begin{pmatrix} \bar{\alpha} \\ \bar{\lambda} \end{pmatrix} = \begin{pmatrix} \bar{y}_0 \\ \bar{y}_1 \\ \vdots \\ \bar{y}_n \end{pmatrix}$$

mediante fattorizzazione QR (possibile poiché tutte le ascisse sono distinte) e ricavare $\begin{pmatrix} \bar{\alpha} \\ \bar{\lambda} \end{pmatrix}$ da qui $\begin{pmatrix} \alpha \\ \lambda \end{pmatrix} = \begin{pmatrix} e^{\bar{\alpha}} \\ -\bar{\lambda} \end{pmatrix}$.

I risultati ottenuti dallo `scritp` MATLAB sono $\begin{pmatrix} \alpha \\ \lambda \end{pmatrix} = \begin{pmatrix} 5.008 \\ 0.0959 \end{pmatrix}$ con un residuo $r = 0.1708$.



Qui si propone il codice:

```
% Esercizio 4.22
%


hold on
y = [5.22; 4.00; 4.28; 3.89; 3.53; 3.12; 2.73; 2.70; 2.20; 2.08; 1.94];
plot ((0:10) ,y, 'black *')
y = log(y);

A = [ ones(1 ,11)' , (0:10)'];
[y, r] = SistemaQR ( FattQR (A),y);
y (1)= exp(y (1));
y(2)= -y (2);
disp ('Soluzione :'), disp (y)
disp ('Residuo :'), disp (r)

plot ((0:10) , y (1)* exp(-y (2)*(0:10)))

legend ('punti ','polinomio ','Location ','Best ')
```

Listing 4.13: Esercizio ??.

□

Capitolo 5

Formule di quadratura

5.1 Esercizio 1

Calcolare il numero di condizionamento dell'integrale

$$\int_0^{e^{21}} \sin\sqrt{x}\, dx.$$

Questo problema è ben condizionato o è malcondizionato?

Soluzione.
κ definisce il numero di condizionamento ed è dato da $\kappa = b - a$
per cui abbiamo $\kappa = b - a = e^{21} - 0 = e^{21} > 10^9$,il problema è, quindi, mal condizionato. $\square$

5.2 Esercizio 2

Derivare, dalla (5.5), i coefficienti della formula dei trapezi (5.6) e della formula di Simpson (5.7).

Soluzione.

- Formula dei trapezi, $n = 1$:
  - $c_{1,1} = \int_0^1 \frac{t-0}{1-0}\, dt = \int_0^1 t = \left.\frac{t^2}{2}\right|_0^1 = \frac{1}{2}$,
  - $c_{0,1} = 1 - c_{1,1} = \frac{1}{2}$
    per la formula dei coefficienti di Newton Cotes

$$c_{k,n} = \int_0^n \prod_{\substack{j=0\\ j\neq k}}^{n} \frac{t-j}{k-j} dt, \qquad k = 0, \ldots, n, \tag{5.1}$$

  Segue $I_1(f) = (b-a)\left(\frac{1}{2}f(a) + \frac{1}{2}f(b)\right) = \frac{b-a}{2}\left(f(a) + f(b)\right)$.

- Formula di Simpson, $n = 2$:

  - $c_{1,2} = \int_0^2 \frac{t-0}{1-0}\frac{t-2}{1-2}\,dt = \int_2^0 t(t-2) = \left(\frac{t^3}{3} - t^2\right)\Big|_0^2 = \frac{4}{3}$,
  - $c_{2,2} = \int_0^2 \frac{t-0}{2-0}\frac{t-1}{2-1}\,dt = \int_2^0 \frac{t(t-1)}{2} = \left(\frac{t^3}{6} - \frac{t^2}{4}\right)\Big|_0^2 = \frac{1}{3}$,
  - $c_{0,2} = 1 - c_{1,2} - c_{2,2} = \frac{1}{3}$
    per la formula dei coefficienti di Newton Cotes

  Segue $I_2(f) = \frac{b-a}{2}\left(\frac{1}{3}f(a) + \frac{4}{3}f\left(\frac{a+b}{2}\right) + \frac{1}{2}f(b)\right) =$
  $= \frac{b-a}{6}\left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right)$.

□

5.3 Esercizio 3

Verificare, utilizzando il risultato del Teorema (5.2) le (5.9) e (5.10).

Soluzione.

- Formula dei trapezi, $n = 1$:

  - $k = 1$,
  - $\nu_1 = \int_0^1 \prod_{j=0}^1 (t-j)\,dt = \int_0^1 (t(t-1))\,dt = \left(\frac{t^3}{3} - \frac{t^2}{2}\right)\Big|_0^1 = -\frac{1}{6}$.

  Segue $E_1(f) = \nu_1 \frac{f^{(2)}(\xi)}{2!}\left(\frac{b-a}{1}\right)^3 = -\frac{1}{12}f^{(2)}(\xi)(b-a)^3$.

- Formula di Simpson, $n = 2$:

  - $k = 2$,
  - $\nu_2 = \int_0^2 t\prod_{j=0}^2 (t-j)\,dt = \int_0^1 (t^2(t-1)(t-2))\,dt =$
    $= \left(\frac{t^5}{5} - 3\frac{t^4}{4} + 2\frac{t^3}{3}\right)\Big|_0^1 = -\frac{4}{15}$.

  Segue $E_2(f) = \nu_2 \frac{f^{(4)}(\xi)}{4!}\left(\frac{b-a}{2}\right)^5 = -\frac{1}{90}f^{(4)}(\xi)\left(\frac{b-a}{2}\right)^5$.

□

5.4 Esercizio 4

Scrivere una `function` MATLAB *che implementi efficientemente la formula dei trapezi composita (5.11).*

Soluzione.

```
% int = trapeziComposita(f, a, b, n)
% Formula dei trapezi composita per l'approssimazione dell'integrale
% definito di una funzione.
%
% Input:
%   -funzione: la funzione di cui si vuol calcolare l'integrale;
%   -a: estremo sinistro dell'intervallo di integrazione;
%   -b: estremo destro dell'intervallo di integrazione;
%   -n: numero di sottointervalli sui quali applicare la formula dei
%   trapezi semplice.
% Output:
%   -int: l'approssimazione dell'integrale definito della funzione.


function [int] = trapeziComposita(funzione, a, b, n)
    h = (b-a)/n;
    int = 0;
    for i=1:n-1
        int = int+funzione(a+i*h);
    end
    int = (h/2)*(2*int + f(a) + f(b));
end
```

Listing 5.1: Formula dei trapezi composita.

□

5.5 Esercizio 5

Scrivere una `function` MATLAB *che implementi efficientemente la formula di Simpson composita (5.13).*

Soluzione.

```
% int = simpsonComposita(funzione, a, b, n)
% Formula di Simpson composita per l'approssimazione dell'integrale
% definito di una funzione.
%
% Input:
%   -funzione: la funzione di cui si vuol calcolare l'integrale;
%   -a: estremo sinistro dell'intervallo di integrazione;
%   -b: estremo destro dell'intervallo di integrazione;
%   -n: numero, pari, di sottointervalli sui quali applicare la formula
%   di Simpson semplice.
% Output:
%   -int: l'approssimazione dell'integrale definito della funzione.
```

```

function [int] = simpsonComposita(funzione, a, b, n)
    h = (b-a)/n;
    int = funzione(a)-funzione(b);
    for i=1:n/2
        int = int + 4*funzione(a+(2*i-1)*h)+2*funzione((a+2*i*h));
    end
    int = int*(h/3);
end
```

Listing 5.2: Formula di Simpson composita.

L'espressione implementata è

$$I_2^{(n)}(f) = \frac{b-a}{3n}(\sum_{i=1}^{n/2}(4f_{2i-1} + 2f_{2i}) + f_0 - f_n).$$

$\square$

5.6 Esercizio 6

Implementare efficientemente in MATLAB *la formula adattativa dei trapezi.*

Soluzione.

```
% [int, pt] = trapeziAdattativaRicorsiva(funzione, a, b, tol, pt)
% Formula dei trapezi adattativa per l'approssimazione dell'integrale
% definito di una funzione.
%
% Input:
%   -funzione: la funzione di cui si vuol calcolare l'integrale;
%   -a: estremo sinistro dell'intervallo di integrazione;
%   -b: estremo destro dell'intervallo di integrazione;
%   -tol: la tolleranza entro la quale si richiede debba rientrare la
%   soluzione approssimata.
% Output:
%   -int: l'approssimazione dell'integrale definito della funzione;
%   -pt: numero di punti generati ricorsivamente.
%

function [int, pt] = trapeziAdattativa(funzione, a, b, tol)
    [int, pt] = trapeziAdattativaRicorsiva(funzione, a, b, tol, 3);
end

function [int, pt] = trapeziAdattativaRicorsiva(funzione, a, b, tol, pt)
    h = (b-a)/2;
    m = (a+b)/2;
    int1 = h*(feval(funzione, a) + feval(funzione, b));
    int = int1/2 + h*feval(funzione, m);
    err = abs(int-int1)/3;
    if err>tol
```

```
        [intSx, ptSx] = trapeziAdattativaRicorsiva(funzione, a, m, tol/2, 1);
        [intDx, ptDx] = trapeziAdattativaRicorsiva(funzione, m, b, tol/2, 1);
        int = intSx+intDx;
        pt = pt+ptSx+ptDx;
    end
end
```

Listing 5.3: Formula dei trapezi adattativa.

□

5.7 Esercizio 7

Implementare efficientemente in MATLAB *la formula adattativa di Simpson.*

Soluzione.

```
% [int, pt] = simpsonAdattativa(funzione, a, b, tol)
% Formula di Simpson adattativa per l'approssimazione dell'integrale
% definito di una funzione.
%
% Input:
%   -funzione: la funzione di cui si vuol calcolare l'integrale;
%   -a: estremo sinistro dell'intervallo di integrazione;
%   -b: estremo destro dell'intervallo di integrazione;
%   -tol: la tolleranza entro la quale si richiede debba rientrare la
%   soluzione approssimata.
% Output:
%   -int: l'approssimazione dell'integrale definito della funzione;
%   -pt: numero di punti generati ricorsivamente.


function [int, pt] = simpsonAdattativa(funzione, a, b, tol)
    [int, pt] = simpsonAdattativaRicorsiva(funzione, a, b, tol, 5);
end

function [int, pt] = simpsonAdattativaRicorsiva(funzione, a, b, tol, pt)
    h = (b-a)/6;
    m = (a+b)/2;
    m1 = (a+m)/2;
    m2 = (m+b)/2;
    int1 = h*(feval(funzione, a) +4*feval(funzione, m) + feval(funzione, b));
    int = int1/2 + h*(2*feval(funzione, m1) + 2*feval(funzione, m2) - feval(funzione, m));
    err = abs(int-int1)/15;
    if err>tol
        [intSx, ptSx] = simpsonAdattativaRicorsiva(funzione, a, m, tol/2, 1);
```

```
        [intDx, ptDx] = simpsonAdattativaRicorsiva(funzione, m, b, tol/2,
    1);
        int = intSx+intDx;
        pt = pt+ptSx+ptDx;
    end
end
```

Listing 5.4: Formula di Simpson adattativa.

□

5.8 Esercizio 8

Come è classificabile, dal punto di vista del condizionamento, il seguente problema?

$$\int_{\frac{1}{2}}^{100} -2x^{-3}\cos\left(x^{-2}\right)\, dx \equiv \sin\left(10^{-4}\right) - \sin(4)$$

Soluzione.

```
% Esercizio 5.08
%


f = inline (' -2.*x .^( -3).* cos(x .^( -2)) ');
subplot (2 ,2 ,1)
axis ([ -5 105 -2 14])
title ('Trapezi composita ');
TrapeziComposita (f, 0.5 , 100 , 50, true );

subplot (2 ,2 ,2)
axis ([ -5 105 -2 14])
title ('Simpson composita ');
SimpsonComposita (f, 0.5 , 100 , 50, true );
subplot (2 ,2 ,3)
axis ([ -5 105 -2 14])
title ('Trapezi adattativa ');
TrapeziAdattativa (f, 0.5 , 100 , 10^ -3 , true );

subplot (2 ,2 ,4)
axis ([ -5 105 -2 14])
title ('Simpson adattativa ');
SimpsonAdattativa (f, 0.5 , 100 , 10^ -3 , true );
```

Listing 5.5: Esercizio **??**.

Per la **??** risulta $\kappa = b - a = 100 - 1/2 = 99.5$, il problema è dunque mal condizionato. Visto che la funzione ha rapide variazioni in $(1, 10) \subset [1/2, 100]$ è preferibile utilizzare le formule di adattative invece che le formule composite; □

5.9 Esercizio 9

Utilizzare le `function` degli Esercizi 5.4 e 5.6 per il calcolo dell'integrale

$$\int_{\frac{1}{2}}^{100} -2x^{-3}\cos\left(x^{-2}\right)\ \mathrm{d}x \equiv \sin\left(10^{-4}\right) - \sin(4),$$

indicando gli errori commessi. Si utilizzi $n = 1000, 2000, \dots, 10000$ per la formula dei trapezi composita e tol $= 10^{-1}, 10^{-2}, \dots, 10^{-5}$ per la formula dei trapezi adattativa (indicando anche il numero di punti).

Soluzione.

```
% Esercizio 5.9
%


f = inline (' -2*x^( -3)* cos(x^ -2) ');

fprintf ('\n\ tFormula composita dei trapezi \n')
for n =1000:1000:10000
  I = TrapeziComposita (f, 1/2 , 100 , n, false );
  fprintf ('n = %d \t I = %5.4 e \t E = %5.4 e\n', n, I, abs(I -( sin
    (10^ -4) - sin (4))));
end
fprintf ('\n\n\ tFormula dei trapezi adattativa \n')
for i =1:4
  tol = 10^ -i;
  [I, p] = TrapeziAdattativa (f, 1/2 , 100 , tol , false );
  fprintf ('tol = %1.1 e \t I = %5.4 e \t E = %5.4 e \t punti = %d\n',
    tol , I, abs(I -( sin (10^ -4) - sin (4))) , p);
end
```

Listing 5.6: Esercizio 5.9.

Formula composita dei trapezi		
n	I	$E_1^{(n)}$
1000	6.6401e-001	9.2897e-002
2000	7.3077e-001	2.6131e-002
3000	7.4507e-001	1.1836e-002
4000	7.5020e-001	6.7007e-003
5000	7.5260e-001	4.3009e-003
6000	7.5391e-001	2.9914e-003
7000	7.5470e-001	2.1998e-003
8000	7.5522e-001	1.6853e-003
9000	7.5557e-001	1.3321e-003
10000	7.5582e-001	1.0793e-003

Formula dei trapezi adattativa			
tol	I	$E_1^{(n)}$	punti
1.0e-001	7.5143e-001	5.4696e-003	159
1.0e-002	7.5563e-001	1.2676e-003	471
1.0e-003	7.5657e-001	3.3005e-004	1567
1.0e-004	7.5684e-001	6.5936e-005	4851

□

5.10 Esercizio 10

Utilizzare le `function` degli Esercizi 5.5 e 5.7 per il calcolo dell'integrale

$$\int_{\frac{1}{2}}^{100} -2x^{-3}\cos\left(x^{-2}\right)\,\mathrm{d}x \equiv \sin\left(10^{-4}\right) - \sin(4),$$

indicando gli errori commessi. Si utilizzi $n = 1000, 2000, \ldots, 10000$ per la formula di Simpson composita e tol $= 10^{-1}, 10^{-2}, \ldots, 10^{-5}$ per la formula di Simpson adattativa (indicando anche il numero di punti).

Soluzione.

```
% Esercizio 5.10
%


f = inline(' -2*x^( -3)* cos(x^ -2) ');

fprintf ('\n\ tFormula composita di Simpson \n')
for n =1000:1000:10000
  I = SimpsonComposita (f, 1/2 , 100 , n, false );
  fprintf ('n = %d \t I = %5.4 e \t E = %5.4 e\n', n, I, abs(I -( sin
    (10^ -4) - sin (4))));
end
fprintf ('\n\n\ tFormula di Simpson adattativa \n')
for i =1:5
  tol = 10^ -i;
  [I, p] = SimpsonAdattativa (f, 1/2 , 100 , tol , false );
  fprintf ('tol = %1.1 e \t I = %5.4 e \t E = %5.4 e \t punti = %d\n',
    tol , I, abs(I -( sin (10^ -4) - sin (4))) , p);
end
```

Listing 5.7: Esercizio 5.10

Formula composita di Simpson		
n	I	$E_1^{(n)}$
1000	7.0132e-001	5.5580e-002
2000	7.5303e-001	3.8753e-003
3000	7.5617e-001	7.2977e-004
4000	7.5668e-001	2.2403e-004
5000	7.5681e-001	9.0209e-005
6000	7.5686e-001	4.3062e-005
7000	7.5688e-001	2.3094e-005
8000	7.5689e-001	1.3479e-005
9000	7.5689e-001	8.3892e-006
10000	7.5690e-001	5.4921e-006

Formula di Simpson adattativa			
tol	I	$E_1^{(n)}$	punti
1.0e-001	7.5701e-001	1.1164e-004	49
1.0e-002	7.5671e-001	1.9384e-004	65
1.0e-003	7.5690e-001	4.8068e-006	93
1.0e-004	7.5688e-001	1.7808e-005	181
1.0e-005	7.5690e-001	4.8337e-006	309

□

Capitolo 6

Calcolo del Google Pagerank

6.1 Esercizio 1

[Teorema di Gershgorin] Dimostrare che gli autovalori di una matrice $A = (a_{ij}) \in \mathbb{C}^{nn}$ *sono contenuti nell'insieme*

$$\mathcal{D} = \bigcup_{i=1}^{n} \mathcal{D}_i, \qquad \mathcal{D}_i = \left\{ \lambda \in \mathbb{C} : |\lambda - a_{ii}| \leq \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}| \right\}, \quad i = 1, \ldots, n.$$

Soluzione.
Sia $\lambda \in \sigma A$ ed $\underline{x}$ il corrispondente autovettore ($\underline{x} \neq \underline{0}$ e $A\underline{x} = \lambda\underline{x}$) quindi $\underline{e}_i^T A\underline{x} = \lambda \underline{e}_i^T \underline{x}$ per $i = 1, \ldots, n$ cioè $\sum_{j=1}^{n} a_{i,j} x_j = \lambda x_i$;
posto i tale che $|x_i| \geq |x_j|$ ($x_i \neq 0$) risulta

$$\lambda = \frac{\sum_{j=1}^{n} a_{i,j} x_j}{x_i} = a_{i,i} + \sum_{j=1\ j\neq i}^{n} a_{i,j} \frac{x_j}{x_i}$$

ovvero $\lambda - a_{i,i} = \sum_{j=1\ j\neq i}^{n} a_{i,j} \frac{x_j}{x_i}$.
Mettendo poi i valori assuluti

$$|\lambda - a_{i,i}| = \left| \sum_{j=1\ j\neq i}^{n} a_{i,j} \frac{x_j}{x_i} \right| \leq \sum_{j=1\ j\neq i}^{n} \left| a_{i,j} \frac{x_j}{x_i} \right| \leq \sum_{j=1\ j\neq i}^{n} |a_{i,j}|$$

poiché $\left| \frac{x_j}{x_i} \right| \leq 1$ in quanto $|x_i| \geq |x_j|$. Segue $\lambda \in \bigcup_{i=1}^{n} \mathcal{D}_i$ da cui $\sigma A \subseteq \mathcal{D}$. □

6.2 Esercizio 2

Utilizzare il metodo delle potenze per approssimare l'autovalore dominante della matrice

$$A_n = \begin{pmatrix} 2 & -1 & & \\ -1 & 2 & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & 2 \end{pmatrix} \in \mathbb{R}^{n\times n},$$

per valori crescenti di n. Verificare numericamente che questo è dato da $2\left(1+\cos\frac{\pi}{n+1}\right)$.

Soluzione.
A_n è una M-matrice,in quanto può essere scritta come : $A_n = 2(I_n - B_n)$ dove

$$B_n = \begin{pmatrix} 0 & 1/2 & & \\ 1/2 & 0 & \ddots & \\ & \ddots & \ddots & 1/2 \\ & & 1/2 & 0 \end{pmatrix} \in \mathbb{R}^{n\times n} \quad I_n = \begin{pmatrix} 2 & 0 & & \\ 0 & 2 & \ddots & \\ & \ddots & \ddots & 0 \\ & & 0 & 2 \end{pmatrix} \in \mathbb{R}^{n\times n}.$$

Risulta $\lambda_j(B_n) = \cos\frac{j\pi}{n+1}$, $|\lambda_j| \le 1$, per $j = 1, \ldots, n$; segue
$\lambda_j(A_n) = 2(\lambda_j(I_n) - \lambda_j(B_n)) = 2(1 - \cos\frac{j\pi}{n+1})$; il massimo è $\lambda = 2(1+\cos\frac{\pi}{n+1})$. La verifica numerica di questo risultato è riportata nelle seguenti tabelle. Codice:

```
% Esercizio 6.2
%


format short

for n =5:5:50
  A=2* eye(n);
  for i=2:n
    A(i,i -1)= -1;
    A(i -1,i)= -1;
  end
  lambda = MetodoPotenze (A, 10^ -5);
  approx = 2*(1+ cos(pi /(n +1)));
  fprintf ('n = %d\ tlambda = %5.4 f\ tapprox = %5.4 f\ terr = %5.4 f \n'
    , n, lambda , approx , abs(lambda - approx ));
end
```

Listing 6.1: Esercizio 6.2

Tolleranza $tol = 10^{-5}$			
n	λ_1	$2\left(1+\cos\frac{\pi}{n+1}\right)$	scostamento
10	3.9189	3.9190	0.0001
15	3.9614	3.9616	0.0002
20	3.9774	3.9777	0.0003
25	3.9853	3.9854	0.0002
30	3.9891	3.9897	0.0006
35	3.9921	3.9924	0.0003
40	3.9929	3.9941	0.0012
45	3.9938	3.9953	0.0015
50	3.9947	3.9962	0.0015

Tolleranza $tol = 10^{-7}$			
n	λ_1	$2\left(1+\cos\frac{\pi}{n+1}\right)$	scostamento
5	3.7321	3.7321	0.0000
10	3.9190	3.9190	0.0000
15	3.9616	3.9616	0.0000
20	3.9777	3.9777	0.0000
25	3.9854	3.9854	0.0000
30	3.9897	3.9897	0.0000
35	3.9924	3.9924	0.0000
40	3.9941	3.9941	0.0000
45	3.9953	3.9953	0.0000
50	3.9962	3.9962	0.0000

□

6.3 Esercizio 3

Dimostrare i Corollari 6.2 e 6.3.

Soluzione.
<u>Corollario 6.2</u>
Se $A = \alpha(I - B)$ è una M-matrice e $A = M - N$, con $0 \le N \le \alpha B$, allora M è nonsingolare e lo splitting è regolare. Pertanto, il metodo iterativo per calcolare un'approssimazione del vettore di pagerank è convergente.
<u>Dimostrazione</u>
Sia $A = \alpha(I - B) = M - N$ con $0 \le N \le \alpha B$ quindi
$M = \alpha I - \alpha B + N = \alpha I - \alpha B + \alpha Q = \alpha(I - (B - Q))$ con $\alpha Q = N \le \alpha B$ e $0 \le Q \le B$. Dato che $0 \le B - Q \le B$, per il Lemma 6.2, $\rho(B - Q) \le \rho(B) \le 1$. Quindi M è una M-matrice; lo splitting è regolare infatti $M^{-1} \ge 0$ (M è una M-matrice) e $B - Q \ge 0$.

<u>Corollario 6.3</u>
Se A è una M-matrice e la matrice M in ($A = M - N$) è ottenuta ponendo a 0 gli elementi extradiagonali di A, allora lo splitting ($A = M - N$) è regolare. Pertanto il metodo iterativo è convergente.

Dimostrazione
Sia $A = M - N$ M-matrice con $M = diag(A)$ allora la matrice $A - M = B$ avrà elementi nulli sulla diagonale e, altrove, gli elementi di A. Poiché $A = \alpha(I - B) = \alpha I - \alpha B = M - N$, risulta $\alpha B = N$ quindi, per il Corollario 6.2, lo splitting è regolare ed il metodo è convergente. $\square$

6.4 Esercizio 4

Dimostrare il Teorema 6.9.

Soluzione.
Teorema 6.9
Se la matrice A in $(A = D - L - U)$ è una M-matrice, allora $D, L, U \geq 0$. In particolare D ha elementi diagonali positivi $(D > 0)$.
Dimistrazione
Poiché A è una M-matrice, essa può essere scritta nella forma $A = \alpha(I - B)$ con $B \geq 0$ e $\rho(B) < 1$; inoltre, per ipotesi, $A = D - L - U$ da cui si deduce che $D = \alpha(I - diag(B))$ e $(L + U) = \alpha(B - diag(B))$.
La matrice $(L + U) = \alpha(B - diag(B))$ risulta maggiore di zero in quanto $B > 0 \rightarrow (B - diag(B)) > 0$. La matrice D ha elementi positivi sulla diagonale: supponiamo per assurdo $a_{i,i} \leq 0$: l'i-esima riga di A, negativa, è data da $A\underline{e_i} \leq \underline{0}$ dato che A è una M-matrice ed è monotona si può scrivere $\underline{e_i} \leq A^{-1}\underline{0} = \underline{0}$ assurdo poiché $\underline{e_i} \geq \underline{0}, \forall i$. $\square$

6.5 Esercizio 5

Tenendo conto della (6.10), riformulare il metodo delle potenze (6.11) per il calcolo del Google pagerank come metodo iterativo definito da uno splitting regolare.

Soluzione.
Il problema del *Google pagerank* è $S(p)\underline{\hat{x}} = \underline{\hat{x}}$ dove $S(p) = pS + (1-p)\underline{v}\,\underline{e}^T$. Sostituendo, risulta

$$(pS + (1-p)\underline{v}\,\underline{e}^T)\underline{\hat{x}} = \underline{\hat{x}} \quad \Rightarrow \quad (I - pS)\underline{\hat{x}} = (1-p)\underline{v}\,\underline{e}^T\,\underline{\hat{x}}$$

Dato che $\underline{v} = \frac{1}{n}\underline{e}$ ed $e^T\underline{\hat{x}} = 1$ si ricava

$$(I - pS)\underline{\hat{x}} = \frac{1-p}{n}\underline{e}.$$

Si può infine definire il seguente metodo iterativo:

$$I\underline{x}_{k+1} = pS\underline{x}_k + \frac{1-p}{n}\underline{e}.$$

Il metodo è convergente in quanto la matrice di iterazione ha raggio spettrale minore di 1: $\rho(I^{-1}pS) = \rho(pS) < 1$ dato che $p \in (0,1)$ e $\rho(S) = 1$. $\square$

6.6 Esercizio 6

Dimostrare che il metodo di Jacobi converge asintoticamente in un numero minore di iterazioni, rispetto al metodo delle potenze (6.11) per il calcolo del Google pagerank.

Soluzione.
La matrice di iterazione di Jacobi ha raggio spettrale minore di 1 mentre, nel calcolo del *Google pagerank*, l'autovalore dominante è $\lambda = 1$ quindi il raggio spettrale di tale matrice è esattamente 1. Poiché il raggio spettrale della matrice di iterazione del metodo di Jacobi è minore del raggio spettrale della matrice del *Google pagerank*, il metodo di Jacobi converge in asintoticamente in un numero minore di iterazioni, rispetto al metodo delle potenze per il calcolo del *Google pagerank*. $\square$

6.7 Esercizio 7

Dimostrare che, se A è diagonale dominante, per riga o per colonna, il metodo di Jacobi è convergente.

Soluzione.
Nel metodo di Jacobi si ha $A = M - N = D - (L + U)$ quindi

$$M = \begin{pmatrix} a_{1,1} & & & \\ & \ddots & & \\ & & \ddots & \\ & & & a_{n,n} \end{pmatrix} \text{ e } N = \begin{pmatrix} 0 & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & a_{n-1,n} \\ a_{n,1} & \dots & a_{n,n-1} & 0 \end{pmatrix}.$$

Si ha

$$B = M^{-1}N = \begin{pmatrix} \frac{1}{a_{1,1}} & & & \\ & \ddots & & \\ & & \ddots & \\ & & & \frac{1}{a_{n,n}} \end{pmatrix} \begin{pmatrix} 0 & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & a_{n-1,n} \\ a_{n,1} & \dots & a_{n,n-1} & 0 \end{pmatrix} =$$

$$= \begin{pmatrix} 0 & \frac{a_{1,2}}{a_{1,1}} & \dots & \frac{a_{1,n}}{a_{1,1}} \\ \frac{a_{2,1}}{a_{2,2}} & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \frac{a_{n-1,n}}{a_{n-1,n-1}} \\ \frac{a_{n,1}}{a_{n,n}} & \dots & \frac{a_{n,n-1}}{a_{n,n}} & 0 \end{pmatrix},$$

per il Teorema di Gershgorin risulta

$$\mathcal{D}_i = \left\{ \lambda \in \mathbb{C} \,:\, |\lambda - b_{i,i}| \le \sum_{j=1\, j\neq i}^{n} |b_{i,j}| \right\} = \left\{ \lambda \in \mathbb{C} \,:\, |\lambda| \le \sum_{j=1\, j\neq i}^{n} \left| \frac{a_{i,j}}{a_{i,i}} \right| \right\};$$

supposta A a diagonale dominante per righe, $|a_{i,i}| \ge \sum_{j=1,\, j\neq i}^{n} |a_{i,j}|$, risulta $|\lambda| \le \frac{1}{|a_{i,i}|} \sum_{j=1\, j\neq i}^{n} a_{i,j} < 1$. Ogni $\mathcal{D}_i$ è centrato in 0 e ha raggio minore di 1 quindi

$\mathcal{D} = \bigcup_{i=1}^{n} \mathcal{D}_i$ è centrato in 0 e ha raggio pari al raggio massimo dei $\mathcal{D}_i$ ma sempre minore di 1. Dato che $\lambda(A) = \lambda(A^T)$, il risultato vale anche se A è a diagonale dominante per colonne; il metodo di Jacobi è dunque convergente per matrici a diagonale dominante. $\square$

6.8 Esercizio 8

Dimostrare che, se A è diagonale dominante, per riga o per colonna, il metodo di Gauss-Seidel è convergente.

Soluzione.
Nel metodo di Gauss-Seidel si ha $A = M - N = (D - L) - U$; sia $\lambda \in \sigma(M^-1N)$ quindi λ è tale che $\det(M^{-1}N - \lambda I) = \det(M^{-1}(N - \lambda M)) = \det(M^{-1})\det(N - \lambda M) = 0$. Dato che, per definizione di splitting, $\det(M^{-1}) \neq 0$ deve risultare $\det(N - \lambda M) = 0 = \det(\lambda M - N)$; sia $H = \lambda M - N$ matrice singolare e supponiamo, per assurdo, $|\lambda| \geq 1$. Risulta

$$H = \begin{cases} \lambda a_{i,j} & \text{se } i \geq j, \\ a_{i,j} & \text{altrimenti,} \end{cases};$$

quindi H è a diagonale dominante ma

$$\sum_{j=1\ j\neq i}^{n} |h_{i,j}| \leq |\lambda| \sum_{j=1\ j\neq i}^{n} |a_{i,j}| < |\lambda||a_{i,i}| = |h_{i,i}|.$$

Si ha una contraddizione poiché le matrici a diagonale dominanti sono non singolari, dunque $|\lambda| < 1$ e quindi il metodo di Gauss-Seidel è convergente per matrici a diagonale dominante. $\square$

6.9 Esercizio 9

Se A è sdp, il metodo di Gauss-Seidel risulta essere convergente. Dimostrare questo risultato nel caso (assai più semplice) in cui l'autovalore di massimo modulo della matrice di iterazione sia reale.
(<u>Suggerimento:</u> considerare il sistema lineare equivalente

$$(D^{-\frac{1}{2}}AD^{-\frac{1}{2}})(D^{\frac{1}{2}}\underline{x}) = (D^{-\frac{1}{2}}\underline{b}), \qquad D^{\frac{1}{2}} = diag(\sqrt{a_{11}}, \ldots, \sqrt{a_{nn}}),$$

la cui matrice dei coefficienti è ancora sdp ma ha diagonale unitaria, ovvero del tipo $I - L - L^T$. Osservare quindi che, per ogni vettore reale $\underline{v}$ di norma 1,si ha:
$\underline{v}^T L\underline{v} = \underline{v}^T L^T \underline{v} = \frac{1}{2}\underline{v}^T(L + L^T)\underline{v} < \frac{1}{2}$.)

Soluzione.
Scriviamo il sistema $A\underline{x} = \underline{b}$ nella forma equivalente

$$\left(D^{-1/2}AD^{-1/2}\right)\left(D^{1/2}\underline{x}\right) = \left(D^{-1/2}\underline{b}\right)$$

con $D = diag(\sqrt{a_{1,1}}, \ldots, \sqrt{a_{n,n}})$. La matrice $C = \left(D^{-1/2}AD^{-1/2}\right)$ ha diagonale unitaria:

$$c_{i,i} = d_i^{-1}a_{i,i}d_i^{-1} = \frac{1}{\sqrt{a_{i,i}}}a_{i,i}\frac{1}{\sqrt{a_{i,i}}} = a_{i,i};$$

inoltre è ancora sdp e scrivibile come $C = I - L - L^T$.
Poiché C è sdp risulta $\underline{v}^T A\underline{v} > 0, \forall \underline{v} \neq \underline{0}$ cioè

$$\underline{v}^T\underline{v} > \underline{v}^T L\underline{v} + \underline{v}^T L^T\underline{v} \quad \Rightarrow \quad \underline{v}^T L\underline{v} = \underline{v}^T L^T\underline{v} < \frac{1}{2}.$$

Sia $|\lambda| = \rho(M_{GS}^{-1}N_{GS}) = \rho\left((I-L)^{-1}L^T\right)$ assunto reale e $\underline{v}$ il corrispondente autovettore, dunque

$$(I-L)^{-1}L^T = \lambda\underline{v} \quad \Rightarrow \quad \lambda\underline{v} = L^T\underline{v} + \lambda L\underline{v}$$

ovvero $\lambda = \underline{v}^T L\underline{v} + \lambda\underline{v}^T L\underline{v} = (1+\lambda)\underline{v}^T L)\underline{v}$ da cui

$$\frac{\lambda}{1+\lambda} = \underline{v}^T L\underline{v} < \frac{1}{2} \quad \Rightarrow \quad -1 < \lambda < 1.$$

Segue $\rho(M_{GS}^{-1}N_{GS}) = |\lambda| < 1$. □

6.10 Esercizio 10

Con riferimento ai vettori errore (6.16) e residuo (6.17) dimostrare che, se

$$||\underline{r}_k|| \leq \varepsilon||\underline{b}||, \tag{6.1}$$

allora

$$||\underline{e}_k|| \leq \varepsilon k(A)||\underline{\hat{x}}||,$$

dove $k(A)$ denota, al solito, il numero di condizionamento della matrice A. Concludere che, per sistemi lineari malcondizionati, anche la risoluzione iterativa (al pari di quella diretta) risulta essere più problematica.

Soluzione.
Posto $\underline{e}_k = \underline{x}_k - \underline{\hat{x}}$ e $\underline{r}_k = A\underline{x}_k - \underline{b}$, risulta

$$A\underline{e}_k = A(\underline{x}_k - \underline{\hat{x}}) = A\underline{x}_k - A\underline{\hat{x}} = A\underline{x}_k - b = \underline{r}_k.$$

Segue, passando alle norme,

$$\begin{aligned} ||\underline{e}_k|| &= \left|\left|A^{-1}\underline{r}_k\right|\right| \leq \left|\left|A^{-1}\right|\right| \cdot ||\underline{r}_k|| \leq \left|\left|A^{-1}\right|\right| \cdot \varepsilon||\underline{b}|| \leq \\ &\leq \left|\left|A^{-1}\right|\right| \cdot ||A|| \cdot ||\underline{\hat{x}}|| = \varepsilon\kappa(A)||\underline{\hat{x}}||, \end{aligned} \tag{6.2}$$

ovvero

$$\frac{||\underline{e}_k||}{||\underline{\hat{x}}||} \leq \varepsilon\kappa(A).$$

La risoluzione iterativa, come la risoluzione diretta, di sistemi lineari è ben condizionata per $\kappa(A) \approx 1$ mentre risulta malcondizionata per $\kappa(A) \gg 1$. □

6.11 Esercizio 11

Calcolare il polinomio caratteristico della matrice

$$\begin{pmatrix} 0 & \dots & 0 & \alpha \\ 1 & \ddots & & 0 \\ & \ddots & \ddots & \vdots \\ 0 & & 1 & 0 \end{pmatrix} \in \mathbb{R}^{n\times n}.$$

Soluzione.
Il polinomio caratteristico è dato del determinante della matrice $A - \lambda I$:

$$\begin{aligned} \det(A-\lambda I) = &\det \begin{pmatrix} -\lambda & \dots & 0 & \alpha \\ 1 & \ddots & & 0 \\ & \ddots & \ddots & \vdots \\ 0 & & 1 & -\lambda \end{pmatrix} = \\ =&(-1)^n\lambda^n + (-1)^{n+1}\alpha = (-1)^n(\lambda^n - \alpha). \end{aligned} \tag{6.3}$$

Le radici di tale polinomio sono $\lambda = \sqrt[n]{\alpha}$. □

6.12 Esercizio 12

Dimostrare che i metodi di Jacobi e Gauss-Seidel possono essere utilizzati per la risoluzione del sistema lineare (gli elementi non indicati sono da intendersi nulli)

$$\begin{pmatrix} 1 & & & -\frac{1}{2} \\ -1 & 1 & & \\ & \ddots & \ddots & \\ & & -1 & 1 \end{pmatrix} \underline{x} = \begin{pmatrix} \frac{1}{2} \\ 0 \\ \vdots \\ 0 \end{pmatrix} \in \mathbb{R}^n,$$

la cui soluzione è $\underline{x} = (1,\dots,1)^T \in \mathbb{R}^n$. *Confrontare il numero di iterazioni richieste dai due metodi per soddisfare lo stesso criterio di arresto (6.19), per valori crescenti di* n *e per tolleranze* ε *decrescenti. Riportare i risultati ottenuti in una tabella* (n/ε).

Soluzione.
La matrice è una M-matrice:

$$A = \begin{pmatrix} 1 & & & -1/2 \\ -1 & 1 & & \\ & \ddots & \ddots & \\ & & -1 & 1 \end{pmatrix} = I - B \text{ con } B = \begin{pmatrix} 0 & & & 1/2 \\ 1 & \ddots & & \\ & \ddots & \ddots & \\ & & 1 & 0 \end{pmatrix} > 0.$$

Si dimostra che $\rho(B) < 1$ calcolando gli autovalori della matrice B:

$$\det(B - \lambda I) =$$

$$\det\begin{pmatrix} -\lambda & & & 1/2 \\ 1 & \ddots & & \\ & \ddots & \ddots & \\ & & 1 & -\lambda \end{pmatrix} = -\lambda \det\begin{pmatrix} -\lambda & & & \\ 1 & \ddots & & \\ & \ddots & \ddots & \\ & & 1 & -\lambda \end{pmatrix}_{(n-1)\times(n-1)} +$$

$$+(-1)^{n+1}\frac{1}{2}\det\begin{pmatrix} 1 & -\lambda & & \\ & \ddots & \ddots & \\ & & \ddots & -\lambda \\ & & & 1 \end{pmatrix}_{(n-1)\times(n-1)} = -\lambda(-\lambda)^{n-1} + (-1)^{n+1}\frac{1}{2} =$$

$$= (-1)^{n-2}\lambda^n + (-1)^{n+1}\frac{1}{2} = (-1)^n\frac{1}{2}(2\lambda^n - 1).$$

Quindi $\det(B - \lambda I) = 0$ se e solo se $2\lambda^n - 1 = 0$ se e solo se $\lambda = 2^{-1/n}$. Poiché $|\lambda| < 1$, $\rho(B) < 1$ quindi A è una M-matrice ed è possibile risolvere il sistema lineare tramite i metodo iterativi di Jacobi e Gauss-Seidel in quando lo splitting è regolare ed A converge.
Implementando il criterio d'arresto $||\underline{r}_k|| \leq \varepsilon||\underline{b}||$, si ha convergenza nel numero di iterazioni riportate nelle seguenti tabelle: Codice:

```
% Esercizio 6.12
%


format short
fprintf ('\ nMETODO DI JACOBI \n')
for n =5:5:50
  tol =10^ -1;
  while (tol >10^ -10)
    A=eye(n);
    for i=2:n

    end
    A(1,n )= -1/2;
    b= zeros (n ,1);
    b (1)=1/2;
    [b,i]= Jacobi ( A, b, tol );
    fprintf ('n = %d \t tol = %5.4 e \t i = %d \n', n, tol , i);
    tol=tol /10;
  end
end
fprintf ('\nMETODO DI GAUSS - SEIDEL \n')
for n =5:5:50
  tol =10^ -1;
  while (tol >10^ -10)
    A=eye(n);
    for i=2:n
      A(i,i -1)= -1;
    end
    A(1,n )= -1/2;
    b= zeros (n ,1);
    b (1)=1/2;
    [b,i]= GaussSeidel ( A, b, tol );
```

```
    fprintf ('n = %d \t tol = %5.4 e \t i = %d \n', n, tol , i);
    tol=tol /10;
  end
  fprintf ('\n')
end
```

Listing 6.2: Esercizio 6.12.

Iterazioni del metodo di Jacobi

$n\backslash\varepsilon$	10^{-1}	10^{-2}	10^{-3}	10^{-14}	10^{-5}	10^{-6}	10^{-7}	10^{-8}	10^{-9}	10^{-10}
5	25	34	54	74	85	105	124	139	157	171
10	50	72	116	145	181	211	250	276	304	342
15	87	125	180	230	284	326	379	430	465	524
20	95	175	239	298	370	433	512	573	628	702
25	128	217	299	375	468	549	643	720	800	885
30	162	265	378	475	570	659	762	870	964	1066
35	199	311	436	533	662	775	905	1014	1120	1249
40	214	384	494	635	755	889	1037	1157	1299	1429
45	252	423	556	703	853	1020	1165	1327	1448	1607
50	299	458	622	787	963	1108	1290	1473	1630	1789

Iterazioni del metodo di Gauss-Seidel

$n\backslash\varepsilon$	10^{-1}	10^{-2}	10^{-3}	10^{-14}	10^{-5}	10^{-6}	10^{-7}	10^{-8}	10^{-9}	10^{-10}
5	3	7	10	13	17	20	22	26	30	33
10	1	6	10	13	16	20	23	26	27	33
15	4	6	9	12	17	19	21	27	27	33
20	2	5	10	12	15	20	23	27	28	33
25	4	4	10	12	16	20	24	23	28	33
30	1	7	7	13	17	19	23	27	29	33
35	2	7	10	11	17	19	23	26	30	32
40	1	7	9	12	17	20	18	27	30	25
45	1	3	9	13	15	20	23	27	30	32
50	2	3	10	7	15	19	23	27	27	31

Si nota come il numero di iterazioni richieste dal metodo di Gauss-Seidel sia molto minore del numero richiesto dal metodo di Jacobi, per ogni valore della tolleranza. □