Elaborato di Calcolo Numerico

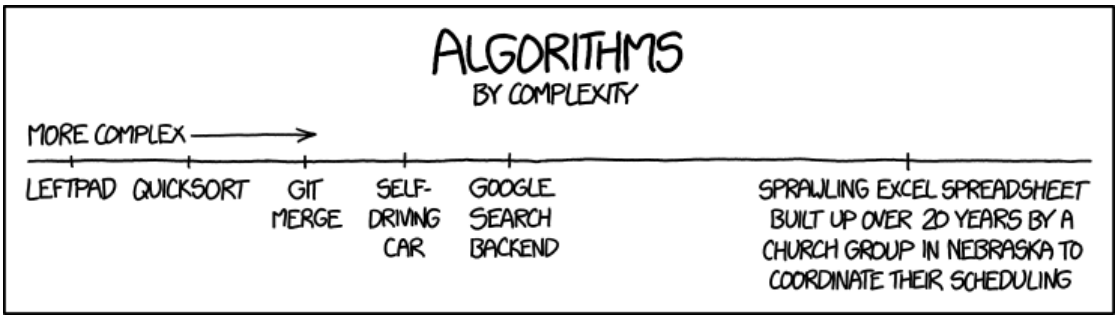
Università degli Studi di Firenze

Laurea in Informatica
Calcolo Numerico (2018-2019)

ALESSANDRO MONTAGHI

MATRICOLA: 6224741

email: *alessandro.montaghi@stud.unifi.it*



xkcd: A webcomic of romance, sarcasm, math, and language

Indice

1 Esercizio 1 **2**

2 Esercizio 2 **4**

3 Esercizio 3 **6**

4 Esercizio 4 **20**

5 Esercizio 5 **22**
5.1 Metodo di Bisezione 22
5.2 Metodo di Newton 26
5.3 Metodo delle Secanti 30
5.4 Metodo delle Corde 33

6 Esercizio 6 **36**
6.1 Metodo della Bisezione 38
6.2 Metodo di Newton 40
6.3 Metodo delle Secanti 42
6.4 Metodo delle Corde 44
6.5 Commenti finali 46

7 Esercizio 7 **48**
7.1 Calcolo della molteplicità della radice nulla per $f(x) = x^2 sin(x^2)$ 48
7.2 Confronto tra i metodi 52
7.3 Commenti finali 62

8 Esercizio 8 **64**
8.1 Algoritmo: *[LU, p] = palu(A)* 64
8.2 Esecuzione 67

9 Esercizio 9 **68**
9.1 Algoritmo: *x = lusolve(LU, b, p)* 68

9.2 Esecuzione 71

10 Esercizio 10 **72**

11 Esercizio 11 **76**
11.1 Algoritmo: *QR = myqr(A)* 76
11.2 Esecuzione 78

12 Esercizio 12 **80**
12.1 Algoritmo: *x = qrsolve(QR, b)* 80
12.2 Esecuzione 82

13 Esercizio 13 **84**

14 Esercizio 14 **88**
14.1 Forma di Lagrange 88
14.1.1 Algoritmo: *y = lagrange(xi, fi, x)* 89
14.1.2 Esecuzione 91
14.2 Forma di Newton 93
14.2.1 Algoritmo: *y = newton(xi, fi, x)* 93
14.2.2 Esecuzione 96

15 Esercizio 15 **98**
15.1 Algoritmo: *y = hermite(xi, fi, x)* 99
15.2 Esecuzione 100

16 Esercizio 16 **104**
16.1 Algoritmo: *s = naturalCubicSplinesSolver(xi, yi, xx)* 104
16.2 Esecuzione 108

17 Esercizio 17 (opzionale) **110**

18 Esercizio 18 **112**

19 Esercizio 19 **118**
19.1 Algoritmo: *x = chebyshev (a, b, n)* 119
19.2 Algoritmo: *[xp, cLeb, pleb] = lebesgue(xnodes, xpoints)* 120
19.3 Esecuzione 121
19.4 Esecuzione 124

20 Esercizio 20 **132**
20.1 Esecuzione 132

21 Esercizio 21 **136**
21.1 Algoritmo: *[res, coeff] = minimiQuadratiResidue (x, y, n)* 138

22 Esercizio 22 **142**
22.1 Algoritmo: *I2 = trapezi(fun, a, b, tol)* 143
22.2 Esecuzione 145
22.3 Algoritmo: *I2 = adapSimpson(fun, a, b, tol)* 146
22.4 Esecuzione 148

23 Esercizio 23 **150**
23.1 Algoritmo: *[I2, np] = adapTrapeziCounter(fun, a, b, tol)* 151
23.2 Esecuzione 152
23.3 Algoritmo: *[I2, np] = adapSimpsonCounter(fun, a, b, tol)* 154
23.4 Esecuzione 155

24 Esercizio 24 **158**
24.1 Algoritmo: *[lambda1, x1] = powerMethod(A, tol, itmax)* 159
24.2 Esecuzione 161

25 Esercizio 25 **162**
25.1 Algoritmo: *[lambda1, x1, numIte] = powerMethodCounter(S, tol)* . . . 162
25.2 Esecuzione 164

26 Esercizio 26 **168**
26.1 Algoritmo: *x = genericSplitting(A, b, msolve, tol)* 169

27 Esercizio 27 **172**
27.1 Algoritmo: *y = Jacobi(A, b)* 172
27.2 Esecuzione 173
27.3 Algoritmo: *y = GaussSeidel(A, b)* 174
27.4 Esecuzione 176

28 Esercizio 28 **178**
28.1 Algoritmo: *[x, nIte] = splittingSparseMatrix(b, matvec, msolve, tol)* . . 181
28.2 Esecuzione 185

1

Esercizio 1

Descrizione: *Verificare che, per h sufficientemente piccolo,* $\frac{3}{2}f(x) - 2f(x-h) + \frac{1}{2}(x - 2h) = hf'(x) + O(h^3)$.

Svolgimento:

Per completare la verifica assegnata, consideriamo lo *Sviluppo di Taylor* di secondo ordine nel punto x_0, ovvero:

$$f(x_0 - h) = f(x_0) - f^1(x_0)h + \frac{1}{2}f^2(x_0)h^2 + O(h^3) \tag{1.1}$$

Sostituendo nella funziona data

$$\frac{3}{2}f(x) - 2f(x-h) + \frac{1}{2}(x - 2h) = hf'(x) + O(h^3) \tag{1.2}$$

La $f(x_0 - h)$, si ottiene che:

$$\begin{aligned} &\frac{2}{3}f(x) - 2\left[f(x) - f^1(x)h + \frac{1}{2}f^2(x)h^2 + O(h^3)\right] + \\ &+ \frac{1}{2}\left[f(x) - f^1(x)2h + \frac{1}{2}f^2(x)4h^2 + O(h^3)\right] = \\ &= hf^1(x) + O(h^3) \end{aligned} \tag{1.3}$$

Svolgendo gli opportuni calcoli e semplificazioni, si ottiene infine:

$$\begin{aligned}
&\frac{3}{2}f(x) - 2f(x) + 2f^1(x)h - \\
&- f^2(x)h^2 + \frac{1}{2}f(x) - \\
&- f^1(x)h + f^2(x)h^2 + O(h^3) = \\
&= hf^1(x) + O(h^3)
\end{aligned} \tag{1.4}$$

Infine, per h sufficientemente piccoli, si completa la verifica nel seguente modo:

$$hf'(x) + O(h^3) = hf'(x) + O(h^3) \tag{1.5}$$

2

Esercizio 2

Descrizione: *Quanti sono i numeri di macchina normalizzati della doppia precisione IEEE? Argomentare la risposta.*

Svolgimento:

L'insieme del numeri di macchina $\mathcal{M}$ è un insieme finito di elementi, tramite i quali è possibile rappresentare un insieme $\mathcal{I}$, caratterizzato dall'essere denso (ossia un numero infinito di elementi) e che rappresenta un sottoinsieme dei numeri reali, ossia $\mathcal{I} \subset \mathbb{R}$.

$$\mathcal{I} = \left[-r_{max}, -r_{min}\right] \cup \{0\} \cup \left[r_{min}, r_{max}\right] \tag{2.1}$$

dove r_{max} e r_{min} rappresentano il più grande ed il più piccolo (in valore assoluto) tra i numeri macchina diversi da 0. Come descritto in precedenza, l'insieme $\mathcal{I}$ è caratterizzato da avere un numero *"infinito"* di elementi, mentre l'insieme $\mathcal{M}$ dei numeri di macchina è un insieme *"finito"* di elementi. Per tale ragione viene definita la funzione $fl : \mathcal{I} \to \mathcal{M}$ che associa ad ogni numero reale $x \in \mathcal{I}$ un corrispondente numero di macchina (o *"floating"*) $fl(x)$. Come conseguenza si ha un *errore di rappresentazione* dato che, in generale, $x \neq fl(x)$. Da questo si desume che, l'errore di rappresentazione è dato dalla differenza $x - fl(x)$.

Al fine di poter garantire che stessi programmi, anche se eseguiti su piattaforme di calcolo differenti, producono gli stessi risultati (principio di riproducibilità), è stato definito uno standard per la rappresentazione dei numeri reali su calcolatore. Tale standard, definito come standard *ANSI/IEEE 754-1985* (detto anche standard *IEEE 754*), e che utilizza la base binaria ($b = 2$), prevede due formati di base per i dati, la *"singola precisione"* e la doppia precisione. In particolare, il formato a *singola precisione* (*single precision*) prevede che un numero reale floating point sia rappresentato con *4 byte* (32

bit) di cui 1 *bit* per il segno (s), 8 *bit* per l'esponente (e) e 23 *bit* per la mantissa (m). Il formato a *doppia precisione* (*double precision*) prevede che un numero reale floating point sia rappresentato con *8 byte* (64 *bit*) di cui 1 *bit* per il segno (s), 11 *bit* per l'esponente (e) e 52 *bit* per la mantissa (m). Il numero di bit riservati all'esponente (e) determina l'intervallo dei numeri rappresentabili. Riguardo a quanti sono i numeri di macchina normalizzati che possono essere ottenuti della doppia precisione IEEE, questo valore ($TotM$) può essere calcolato nella seguente relazione

$$\begin{aligned} TotM &= 2^1 \cdot 2^{52} \cdot (2^{11} - 2^1) = \\ &= 2^{1+52} \cdot (2^{11} - 2^1) = \\ &= 2^{1+52+11} - 2^{1+52+1} = \\ &= 2^{64} - 2^{54} = 18,428,729,675,200,069,632 \end{aligned} \tag{2.2}$$

3

Esercizio 3

Descrizione: *Eseguire il seguente script Matlab e spiegare i risultati ottenuti:*

format long e
n=75;
*u=1e-300; for i=1:n, u=u*2; end, for i=1:n, u=u/2; end, u*
*u=1e-300; for i=1:n, u=u/2; end, for i=1:n, u=u*2; end, u*

Svolgimento:

Per lo svolgimento di questo quesito, è stato implementato, all'interno dell'ambiente Matlab, il seguente script:

```
% Largest positive floating-point number
ndouble_max = realmax;
% Smallest positive normalized floating-point number
ndouble_min = realmin;

format long e
n = 75;
u = 1e-300;
original_u = 1e-300;

disp('First Loop (u*2):')
for i = 1:n
    u = u*2;
```

```
    if u > ndouble_max
       disp('Largest positive floating-point number exceeded');
    end
    fprintf('Loop: %d, value: %e\n',i, u);
end

newline;
disp('Second Loop (u/2):')
for i = 1:n
    u = u/2;
    if u < ndouble_min
       disp('Smallest positive floating-point number exceeded');
    end
    fprintf('Loop: %d, value: %e\n',i, u);
end

fprintf('FIRST PART - Final value of u: %e\n',u);

if (u ~= original_u)
   disp('FIRST PART: the intial value of u is different to the final
      ↪ value')
else
   disp('FIRST PART: the intial value of u is equal to the final value')
end
```

Listing 3.1: Codice Matlab Esercizio 3 - prima parte.

Con la procedura appena utilizzata, il valore della variabile u è stato inizialmente moltiplicato per 2 (i.e., $u \cdot 2$), per un numero di volte pari a 75 (ossia $u \cdot 2^{75}$), e successivamente diviso per 2 (i.e., $u/2$), per un egual numero di volte (ovvero $u \cdot 2^{-75}$). Il valore finale della variabile u è di *"1.000000e-300"*, che corrisponde al valore di partenza. Infatti, sia l'operazione di moltiplicazione che di divisione sono operazioni algebriche elementari di tipo *"ben condizionate"*, dato che per entrambe queste due operazioni il loro *"numero di condizionamento"* è $k = 2$.

Di seguito, sono riportati i risultati dell'analisi effettuata.

```
First Loop (u*2):
Loop: 1, value: 2.000000e-300
Loop: 2, value: 4.000000e-300
Loop: 3, value: 8.000000e-300
Loop: 4, value: 1.600000e-299
Loop: 5, value: 3.200000e-299
Loop: 6, value: 6.400000e-299
Loop: 7, value: 1.280000e-298
Loop: 8, value: 2.560000e-298
Loop: 9, value: 5.120000e-298
Loop: 10, value: 1.024000e-297
Loop: 11, value: 2.048000e-297
Loop: 12, value: 4.096000e-297
Loop: 13, value: 8.192000e-297
Loop: 14, value: 1.638400e-296
Loop: 15, value: 3.276800e-296
Loop: 16, value: 6.553600e-296
Loop: 17, value: 1.310720e-295
Loop: 18, value: 2.621440e-295
Loop: 19, value: 5.242880e-295
Loop: 20, value: 1.048576e-294
Loop: 21, value: 2.097152e-294
Loop: 22, value: 4.194304e-294
Loop: 23, value: 8.388608e-294
Loop: 24, value: 1.677722e-293
Loop: 25, value: 3.355443e-293
Loop: 26, value: 6.710886e-293
Loop: 27, value: 1.342177e-292
Loop: 28, value: 2.684355e-292
Loop: 29, value: 5.368709e-292
Loop: 30, value: 1.073742e-291
Loop: 31, value: 2.147484e-291
Loop: 32, value: 4.294967e-291
Loop: 33, value: 8.589935e-291
Loop: 34, value: 1.717987e-290
Loop: 35, value: 3.435974e-290
Loop: 36, value: 6.871948e-290
Loop: 37, value: 1.374390e-289
```

```
Loop: 38, value: 2.748779e-289
Loop: 39, value: 5.497558e-289
Loop: 40, value: 1.099512e-288
Loop: 41, value: 2.199023e-288
Loop: 42, value: 4.398047e-288
Loop: 43, value: 8.796093e-288
Loop: 44, value: 1.759219e-287
Loop: 45, value: 3.518437e-287
Loop: 46, value: 7.036874e-287
Loop: 47, value: 1.407375e-286
Loop: 48, value: 2.814750e-286
Loop: 49, value: 5.629500e-286
Loop: 50, value: 1.125900e-285
Loop: 51, value: 2.251800e-285
Loop: 52, value: 4.503600e-285
Loop: 53, value: 9.007199e-285
Loop: 54, value: 1.801440e-284
Loop: 55, value: 3.602880e-284
Loop: 56, value: 7.205759e-284
Loop: 57, value: 1.441152e-283
Loop: 58, value: 2.882304e-283
Loop: 59, value: 5.764608e-283
Loop: 60, value: 1.152922e-282
Loop: 61, value: 2.305843e-282
Loop: 62, value: 4.611686e-282
Loop: 63, value: 9.223372e-282
Loop: 64, value: 1.844674e-281
Loop: 65, value: 3.689349e-281
Loop: 66, value: 7.378698e-281
Loop: 67, value: 1.475740e-280
Loop: 68, value: 2.951479e-280
Loop: 69, value: 5.902958e-280
Loop: 70, value: 1.180592e-279
Loop: 71, value: 2.361183e-279
Loop: 72, value: 4.722366e-279
Loop: 73, value: 9.444733e-279
Loop: 74, value: 1.888947e-278
Loop: 75, value: 3.777893e-278

Second Loop (u/2):
```

```
Loop: 1, value: 1.888947e-278
Loop: 2, value: 9.444733e-279
Loop: 3, value: 4.722366e-279
Loop: 4, value: 2.361183e-279
Loop: 5, value: 1.180592e-279
Loop: 6, value: 5.902958e-280
Loop: 7, value: 2.951479e-280
Loop: 8, value: 1.475740e-280
Loop: 9, value: 7.378698e-281
Loop: 10, value: 3.689349e-281
Loop: 11, value: 1.844674e-281
Loop: 12, value: 9.223372e-282
Loop: 13, value: 4.611686e-282
Loop: 14, value: 2.305843e-282
Loop: 15, value: 1.152922e-282
Loop: 16, value: 5.764608e-283
Loop: 17, value: 2.882304e-283
Loop: 18, value: 1.441152e-283
Loop: 19, value: 7.205759e-284
Loop: 20, value: 3.602880e-284
Loop: 21, value: 1.801440e-284
Loop: 22, value: 9.007199e-285
Loop: 23, value: 4.503600e-285
Loop: 24, value: 2.251800e-285
Loop: 25, value: 1.125900e-285
Loop: 26, value: 5.629500e-286
Loop: 27, value: 2.814750e-286
Loop: 28, value: 1.407375e-286
Loop: 29, value: 7.036874e-287
Loop: 30, value: 3.518437e-287
Loop: 31, value: 1.759219e-287
Loop: 32, value: 8.796093e-288
Loop: 33, value: 4.398047e-288
Loop: 34, value: 2.199023e-288
Loop: 35, value: 1.099512e-288
Loop: 36, value: 5.497558e-289
Loop: 37, value: 2.748779e-289
Loop: 38, value: 1.374390e-289
Loop: 39, value: 6.871948e-290
Loop: 40, value: 3.435974e-290
```

```
Loop: 41, value: 1.717987e-290
Loop: 42, value: 8.589935e-291
Loop: 43, value: 4.294967e-291
Loop: 44, value: 2.147484e-291
Loop: 45, value: 1.073742e-291
Loop: 46, value: 5.368709e-292
Loop: 47, value: 2.684355e-292
Loop: 48, value: 1.342177e-292
Loop: 49, value: 6.710886e-293
Loop: 50, value: 3.355443e-293
Loop: 51, value: 1.677722e-293
Loop: 52, value: 8.388608e-294
Loop: 53, value: 4.194304e-294
Loop: 54, value: 2.097152e-294
Loop: 55, value: 1.048576e-294
Loop: 56, value: 5.242880e-295
Loop: 57, value: 2.621440e-295
Loop: 58, value: 1.310720e-295
Loop: 59, value: 6.553600e-296
Loop: 60, value: 3.276800e-296
Loop: 61, value: 1.638400e-296
Loop: 62, value: 8.192000e-297
Loop: 63, value: 4.096000e-297
Loop: 64, value: 2.048000e-297
Loop: 65, value: 1.024000e-297
Loop: 66, value: 5.120000e-298
Loop: 67, value: 2.560000e-298
Loop: 68, value: 1.280000e-298
Loop: 69, value: 6.400000e-299
Loop: 70, value: 3.200000e-299
Loop: 71, value: 1.600000e-299
Loop: 72, value: 8.000000e-300
Loop: 73, value: 4.000000e-300
Loop: 74, value: 2.000000e-300
Loop: 75, value: 1.000000e-300

FIRST PART - Final value of u: 1.000000e-300
FIRST PART: the intial value of u is equal to the final value
```

Listing 3.2: Codice Matlab Esercizio 3 - Esecuzione prima parte.

```
1
2  u = 1e-300;
3
4  disp('First Loop (u/2):')
5  for i = 1:n
6      u = u/2;
7      if u < ndouble_min
8         disp('Smallest positive
9               floating-point number exceeded');
10     end
11     fprintf('Loop: %d, value: %e\n',i, u);
12 end
13
14 newline;
15 disp('Second Loop (u*2):')
16 for i=1:n
17     u = u*2;
18     if u > ndouble_max
19        disp('Largest positive
20              floating-point number exceeded');
21     end
22     fprintf("Loop: %d, value: %e\n",i, u);
23 end
24
25 fprintf('SECOND PART - Final value of u: %e\n', u);
26
27 if (u ~= original_u)
28    disp('SECOND PART: the intial value
29          of u is different to the final value')
30 else
31    disp('SECOND PART: the intial value
32          of u is equal to the final value')
33 end
```

Listing 3.3: Codice Matlab Esercizio 3 - seconda parte

Tramite questo secondo script, il valore della variabile u è stato inizialmente diviso per 2 (i.e., $u/2$), per un numero di volte pari a 75 (ossia $u \cdot 2^{75}$), e successivamente moltiplicato per 2 (i.e., $u \cdot 2$), per un egual numero di volte. In questo caso, il valore finale della variabile u risulta pari a *"1.119916e-300"*, che corrisponde ad un valore maggiore e

quindi diverso rispetto a quello di partenza. Questa differenza, come illustrato precedentemente, non può essere imputata al *condizionamento del problema*, dato che entrambe le operazioni usate sono *"ben condizionate"*.

Questo risultato è spiegabile tramite il sopraggiungere di una condizione di errore definita di *"Underflow"* ($0 < |x| < r_{min}$, dove r_{min} è il più piccolo, in valore assoluto, tra i numeri di macchina diversi da zero). La condizione di *"Underflow"* avviene durante il primo ciclo di divisioni della variabile u (i.e., $u/2$). Infatti, dividendo ripetutamente la variabile u, ad un certo punto, il suo valore diventa più piccolo di r_{min}, comportando errori di arrotondamento durante nelle successive divisioni. L'elaboratore, usando lo Standard lo *Standard IEEE 754*, utilizza una tecnica di *recovery* di tipo *gradual underflow*, tramite la quale la mantissa del numero rappresentato viene *denormalizzata* e l'insieme dei numeri macchina viene ampliato tramite i *numeri macchina denormalizzati*. Comunque, anche se è attuata questa recovery, la successiva moltiplicazione non garantisce di ottenere il valore di partenza. In Matlab, tramite la funzione *"realmin"* è possibile ottenere il minimo numero reale positivo rappresentabile (i.e., 2.225073858507201e-308). Di seguito, sono riportati i risultati dell'analisi effettuata.

```
First Loop (u/2):
Loop: 1, value: 5.000000e-301
Loop: 2, value: 2.500000e-301
Loop: 3, value: 1.250000e-301
Loop: 4, value: 6.250000e-302
Loop: 5, value: 3.125000e-302
Loop: 6, value: 1.562500e-302
Loop: 7, value: 7.812500e-303
Loop: 8, value: 3.906250e-303
Loop: 9, value: 1.953125e-303
Loop: 10, value: 9.765625e-304
Loop: 11, value: 4.882813e-304
Loop: 12, value: 2.441406e-304
Loop: 13, value: 1.220703e-304
Loop: 14, value: 6.103516e-305
Loop: 15, value: 3.051758e-305
Loop: 16, value: 1.525879e-305
Loop: 17, value: 7.629395e-306
Loop: 18, value: 3.814697e-306
Loop: 19, value: 1.907349e-306
Loop: 20, value: 9.536743e-307
Loop: 21, value: 4.768372e-307
Loop: 22, value: 2.384186e-307
```

```
Loop: 23, value: 1.192093e-307
Loop: 24, value: 5.960464e-308
Loop: 25, value: 2.980232e-308
Smallest positive floating-point number exceeded
Loop: 26, value: 1.490116e-308
Smallest positive floating-point number exceeded
Loop: 27, value: 7.450581e-309
Smallest positive floating-point number exceeded
Loop: 28, value: 3.725290e-309
Smallest positive floating-point number exceeded
Loop: 29, value: 1.862645e-309
Smallest positive floating-point number exceeded
Loop: 30, value: 9.313226e-310
Smallest positive floating-point number exceeded
Loop: 31, value: 4.656613e-310
Smallest positive floating-point number exceeded
Loop: 32, value: 2.328306e-310
Smallest positive floating-point number exceeded
Loop: 33, value: 1.164153e-310
Smallest positive floating-point number exceeded
Loop: 34, value: 5.820766e-311
Smallest positive floating-point number exceeded
Loop: 35, value: 2.910383e-311
Smallest positive floating-point number exceeded
Loop: 36, value: 1.455192e-311
Smallest positive floating-point number exceeded
Loop: 37, value: 7.275958e-312
Smallest positive floating-point number exceeded
Loop: 38, value: 3.637979e-312
Smallest positive floating-point number exceeded
Loop: 39, value: 1.818989e-312
Smallest positive floating-point number exceeded
Loop: 40, value: 9.094947e-313
Smallest positive floating-point number exceeded
Loop: 41, value: 4.547474e-313
Smallest positive floating-point number exceeded
Loop: 42, value: 2.273737e-313
Smallest positive floating-point number exceeded
Loop: 43, value: 1.136868e-313
Smallest positive floating-point number exceeded
```

```
Loop: 44, value: 5.684342e-314
Smallest positive floating-point number exceeded
Loop: 45, value: 2.842171e-314
Smallest positive floating-point number exceeded
Loop: 46, value: 1.421085e-314
Smallest positive floating-point number exceeded
Loop: 47, value: 7.105427e-315
Smallest positive floating-point number exceeded
Loop: 48, value: 3.552714e-315
Smallest positive floating-point number exceeded
Loop: 49, value: 1.776357e-315
Smallest positive floating-point number exceeded
Loop: 50, value: 8.881784e-316
Smallest positive floating-point number exceeded
Loop: 51, value: 4.440892e-316
Smallest positive floating-point number exceeded
Loop: 52, value: 2.220446e-316
Smallest positive floating-point number exceeded
Loop: 53, value: 1.110223e-316
Smallest positive floating-point number exceeded
Loop: 54, value: 5.551115e-317
Smallest positive floating-point number exceeded
Loop: 55, value: 2.775558e-317
Smallest positive floating-point number exceeded
Loop: 56, value: 1.387779e-317
Smallest positive floating-point number exceeded
Loop: 57, value: 6.938895e-318
Smallest positive floating-point number exceeded
Loop: 58, value: 3.469448e-318
Smallest positive floating-point number exceeded
Loop: 59, value: 1.734724e-318
Smallest positive floating-point number exceeded
Loop: 60, value: 8.673619e-319
Smallest positive floating-point number exceeded
Loop: 61, value: 4.336809e-319
Smallest positive floating-point number exceeded
Loop: 62, value: 2.168405e-319
Smallest positive floating-point number exceeded
Loop: 63, value: 1.084178e-319
Smallest positive floating-point number exceeded
```

```
Loop: 64, value: 5.420888e-320
Smallest positive floating-point number exceeded
Loop: 65, value: 2.710444e-320
Smallest positive floating-point number exceeded
Loop: 66, value: 1.355222e-320
Smallest positive floating-point number exceeded
Loop: 67, value: 6.778581e-321
Smallest positive floating-point number exceeded
Loop: 68, value: 3.389290e-321
Smallest positive floating-point number exceeded
Loop: 69, value: 1.694645e-321
Smallest positive floating-point number exceeded
Loop: 70, value: 8.497929e-322
Smallest positive floating-point number exceeded
Loop: 71, value: 4.248965e-322
Smallest positive floating-point number exceeded
Loop: 72, value: 2.124482e-322
Smallest positive floating-point number exceeded
Loop: 73, value: 1.086944e-322
Smallest positive floating-point number exceeded
Loop: 74, value: 5.434722e-323
Smallest positive floating-point number exceeded
Loop: 75, value: 2.964394e-323
Second Loop (u*2):
Loop: 1, value: 5.928788e-323
Loop: 2, value: 1.185758e-322
Loop: 3, value: 2.371515e-322
Loop: 4, value: 4.743030e-322
Loop: 5, value: 9.486060e-322
Loop: 6, value: 1.897212e-321
Loop: 7, value: 3.794424e-321
Loop: 8, value: 7.588848e-321
Loop: 9, value: 1.517770e-320
Loop: 10, value: 3.035539e-320
Loop: 11, value: 6.071079e-320
Loop: 12, value: 1.214216e-319
Loop: 13, value: 2.428431e-319
Loop: 14, value: 4.856863e-319
Loop: 15, value: 9.713726e-319
Loop: 16, value: 1.942745e-318
```

```
Loop: 17, value: 3.885490e-318
Loop: 18, value: 7.770981e-318
Loop: 19, value: 1.554196e-317
Loop: 20, value: 3.108392e-317
Loop: 21, value: 6.216785e-317
Loop: 22, value: 1.243357e-316
Loop: 23, value: 2.486714e-316
Loop: 24, value: 4.973428e-316
Loop: 25, value: 9.946855e-316
Loop: 26, value: 1.989371e-315
Loop: 27, value: 3.978742e-315
Loop: 28, value: 7.957484e-315
Loop: 29, value: 1.591497e-314
Loop: 30, value: 3.182994e-314
Loop: 31, value: 6.365987e-314
Loop: 32, value: 1.273197e-313
Loop: 33, value: 2.546395e-313
Loop: 34, value: 5.092790e-313
Loop: 35, value: 1.018558e-312
Loop: 36, value: 2.037116e-312
Loop: 37, value: 4.074232e-312
Loop: 38, value: 8.148464e-312
Loop: 39, value: 1.629693e-311
Loop: 40, value: 3.259386e-311
Loop: 41, value: 6.518771e-311
Loop: 42, value: 1.303754e-310
Loop: 43, value: 2.607508e-310
Loop: 44, value: 5.215017e-310
Loop: 45, value: 1.043003e-309
Loop: 46, value: 2.086007e-309
Loop: 47, value: 4.172013e-309
Loop: 48, value: 8.344027e-309
Loop: 49, value: 1.668805e-308
Loop: 50, value: 3.337611e-308
Loop: 51, value: 6.675222e-308
Loop: 52, value: 1.335044e-307
Loop: 53, value: 2.670089e-307
Loop: 54, value: 5.340177e-307
Loop: 55, value: 1.068035e-306
Loop: 56, value: 2.136071e-306
```

```
Loop: 57, value: 4.272142e-306
Loop: 58, value: 8.544284e-306
Loop: 59, value: 1.708857e-305
Loop: 60, value: 3.417713e-305
Loop: 61, value: 6.835427e-305
Loop: 62, value: 1.367085e-304
Loop: 63, value: 2.734171e-304
Loop: 64, value: 5.468342e-304
Loop: 65, value: 1.093668e-303
Loop: 66, value: 2.187337e-303
Loop: 67, value: 4.374673e-303
Loop: 68, value: 8.749346e-303
Loop: 69, value: 1.749869e-302
Loop: 70, value: 3.499739e-302
Loop: 71, value: 6.999477e-302
Loop: 72, value: 1.399895e-301
Loop: 73, value: 2.799791e-301
Loop: 74, value: 5.599582e-301
Loop: 75, value: 1.119916e-300
SECOND PART - Final value of u: 1.119916e-300
SECOND PART: the intial value of u is different to the final value
```

Listing 3.4: Codice Matlab Esercizio 3 - Esecuzione seconda parte

4

Esercizio 4

Descrizione: *Eseguire le seguenti istruzioni Matlab e spiegare i risultati ottenuti:*

format long e
a = 1.111111111111111
b = 1.11111111111111
a + b
a - b

Svolgimento:

Per lo svolgimento di questo quesito, è stato implementato, all'interno dell'ambiente Matlab, il seguente script:

```
format long e
a = 1.111111111111111;
b = 1.11111111111111;
sumResult = a + b;
differenceResult = a - b;
fprintf('a + b = %1.10e\n', sumResult);
fprintf('a - b = %1.10e\n', differenceResult);
```

Listing 4.1: Codice Matlab Esercizio 4

I risultai ottenuti sono di seguito riportati:

```
a + b = 2.2222222222e+00
a - b = 8.8817841970e-16
```

Listing 4.2: Risultato dell'esecuzione Esercizio 4

Il risultato che si ottiene dal codice è rispettivamente di *"2.222222222222221e+00"* per *sumResult* e *"8.881784197001252e-16"* per *differenceResult*.

Per quanto riguarda il primo risultato, nell'eseguire tale operazioni tramite un aritmetica ordinaria (o esatta), il risultato dell'operazione della somma algebrica $(a+b)$ è pari a *"2.222222222222221"*, il quale corrisponde al risultato fornito, utilizzando l'aritmetica finita, dall'elaboratore. Più in dettaglio e considerando lo studio del *"Condizionamento del problema"* per la somma algebrica, esprimibile tramite la seguente relazione:

$$\left|\varepsilon_y\right| \leq \frac{|a| + |b|}{|a + b|} \equiv k \cdot \varepsilon_x \tag{4.1}$$

Dove ε_y è l'*errore relativo* sui dati in uscita, ε_x è l'*errore relativo* sui dati in ingresso, dato da $\varepsilon_x = max\left\{\left|\varepsilon_a\right|, \left|\varepsilon_b\right|\right\}$, con ε_a e ε_b *errori relativi* sui dati iniziali. In particolare, k rappresenta il *"Numero di condizionamento"* del problema (per la somma).

$$k = \frac{|a| + |b|}{|a + b|} \tag{4.2}$$

Nel caso di $a \cdot b > 0$ (ossia tutti e due aventi lo stesso segno), $k = 1$ e quindi la somma di due numeri concordi è sempre ben condizionata.

Per quanto riguarda il risultato dell'operazione di sottrazione $(a - b)$, tramite aritmetica ordinaria, il risultato ottenuto è pari a "0.000000000000001" (ovvero $1 \cdot 10^{-16}$). Questo valore, si discosta dal risultato ottenuto in aritmetica finita a causa del cosiddetto fenomeno della *"cancellazione numerica"*. La cancellazione numerica è un fenomeno perdita di cifre significative che si verifica durante l'operazione di sottrazione tra due numeri "quasi uguali" (ossia i due operandi sono vicini tra loro). Infatti, riprendendo i concetti sopra discussi, nel caso in cui si abbia che $a \approx -b$, il numero di condizionamento più risultare arbitrariamente grande, risultando un'operazione di tipo mal condizionata.

5

Esercizio 5

Descrizione: *Scrivere function Matlab distinte che implementino efficientemente i seguenti metodi per la ricerca degli zeri di una funzione: (i) metodo di bisezione; (ii) metodo di Newton; (iii) metodo delle secanti e (iv) metodo delle corde. Detta x_i l'approssimazione al passo i-esimo, utilizzando come criterio di arresto $|\Delta x_i| \leq tol \cdot (1+|x_i|)$ essendo tol una opportuna tolleranza specificata in ingresso.*

Svolgimento:

5.1 Metodo di Bisezione

Sia $f(x)$ una funzione continua in un intervallo chiuso e limitato $[a, b]$, con $f(a)f(b) < 0$ e $a < b$. Allora esiste almeno una radice $\hat{x} \in (a, b)$ tale che $f(\hat{x}) = 0$. Tramite il metodo di Bisezione, implementato con l'algoritmo 5.1, è possibile in maniera iterativa determinare il valore di $\hat{x}$ per cui $f(\hat{x}) = 0$. L'algoritmo è implementato tenendo conto di un criterio di arresto basato su un valore di tolleranza (tol) definito in ingresso, che serve ad approssimare il calcolo della radice.

```
1 function [x, iter] = bisection (f, low, high, tol, setprint)
2     %
3     % function x = bisection (f,low,high,tol)
4     %
5     % Author : Alessandro Montaghi
6     % Email  : alessandro.montaghi@gmail.com
```

```
% Date   : Spring 2019
% Course : Calcolo Numerico 2018/2019
%
% Function   : Bisection method
%
% Description:  This code calculates roots of continuous
%               functions within a given interval and
%               uses the Bisection method.
%
% Parameters : f - input funtion
%              low, high - interval [low, high]
%              tol - tolerance
%              setprint - print output (1: yes)
%
% Return     : x-root and numeber of iteration (iter)
%
% Examples of Usage:
%
%    >> my_fun = @(x) 5*x^4 - 2.7*x^2 - 2*x + .5;
%    >> a = .1;
%    >> b = 0.5;
%    >> tol = .00001;
%    >> x = bisection(my_fun, a, b, tol);
%
%    >> Root at x = 0.200000
%

flow = feval(f, low);
fhigh = feval(f, high);
if ( (flow * fhigh) > 0 )
    error('The interval range [low, high] does not contain any roots
       ↪ ');
end
x = (low + high)/2;
fx = feval(f, x);
imax = ceil(log2(high - low) - log2(tol));
x_set = realmax;
if (setprint == 1)
    disp('Iter    low        high          x0');
end
```

```
46      for i = 2:imax
47          if (setprint == 1)
48              fprintf('%2i \t %e \t %e \t %e \n', i-1, low, high, x);
49          end
50          fder = abs( (fhigh - flow)/(high - low) );
51          delta = abs(x_set - x);
52          tolx = tol *( 1 + abs(x));
53          if (abs(fx) <= tolx * fder || delta <= tolx)
54              if (setprint == 1)
55                  fprintf('Root at x = %e\n', x);
56              end
57              break
58          elseif ( (flow * fx) < 0)
59              high = x;
60              fhigh = fx;
61          else
62              low = x;
63              flow = fx;
64          end
65          x_set = x;
66          x = (low + high)/2;
67          fx = feval(f, x);
68      end
69      if (setprint == 1)
70          % Show the last approximation considering the tolerance
71          froot = feval(f, x);
72          fprintf('\n x-root = %e produces f(x) = %e \n %i iterations\n',
                ↪ x, froot, i-1);
73          fprintf(' Approximation with tolerance = %e \n', tol);
74      end
75      iter = i+1;
76      return
```

Listing 5.1: Metodo di bisezione

La funzione *function x = bisection (f,low,high,tol)* sviluppata è stata testata su di una funzione $2.5x^2 - 3x + 0.5$ di cui si conosce una sua radice.

```
my_fun = @(x) 2.5*x^2 - 3*x + .5;
low = 0;
high = 0.5;
tolerance = .00001;
setprint = 1;

[x, iter] = bisection(my_fun, low, high, tolerance, setprint);

% Bisection's method
Iter    low             high            x0
 1       0.000000e+00    5.000000e-01    2.500000e-01
 2       0.000000e+00    2.500000e-01    1.250000e-01
 3       1.250000e-01    2.500000e-01    1.875000e-01
 4       1.875000e-01    2.500000e-01    2.187500e-01
 5       1.875000e-01    2.187500e-01    2.031250e-01
 6       1.875000e-01    2.031250e-01    1.953125e-01
 7       1.953125e-01    2.031250e-01    1.992188e-01
 8       1.992188e-01    2.031250e-01    2.011719e-01
 9       1.992188e-01    2.011719e-01    2.001953e-01
10       1.992188e-01    2.001953e-01    1.997070e-01
11       1.997070e-01    2.001953e-01    1.999512e-01
12       1.999512e-01    2.001953e-01    2.000732e-01
13       1.999512e-01    2.000732e-01    2.000122e-01
14       1.999512e-01    2.000122e-01    1.999817e-01
15       1.999817e-01    2.000122e-01    1.999969e-01
Root at x = 1.999969e-01

x-root = 1.999969e-01 produces f(x) = 6.103539e-06
15 iterations
Approximation with tolerance = 1.000000e-05
```

Listing 5.2: Risultato dell'esecuzione del Metodo di bisezione

5.2 Metodo di Newton

Il *metodo di Newton*, implementato con l'algoritmo 5.2, è un metodo iterativo a passo (funzionale) singolo. Supposta la funzione $f(x)$ da studiare continua e derivabile in un intervallo chiuso e limitato $[a, b]$, il valore di $\hat{x}$ per cui $f(\hat{x}) = 0$, è determinato partendo da un punto di innesco iniziale x_0 e risolvendo una successione di equazioni lineari. L'espressione funzionale del *metodo di Newton* è la seguente:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}, \quad per \quad i = 1, 2, \cdots, \quad e \quad con \quad f'(x_i) \neq 0 \tag{5.1}$$

Il successivo algoritmo descrive l'implementazione del *metodo di Newton* con l'utilizzo del parametro di tolleranza *tol* e come criterio di arresto $\left|\Delta x_i\right| \leq tol \cdot (1 + \left|x_i\right|)$.

```
function [x, iter] = newton(f, f1, x0, tol, nmax, setprint)
    %
    % function x = newton(f, f1, x0, tol, nmax, setprint)
    %
    % Author : Alessandro Montaghi
    % Email  : alessandro.montaghi@gmail.com
    % Date   : Spring 2019
    % Course : Calcolo Numerico 2018/2019
    %
    % Function   : Newton method
    %
    % Description:  This code calculates the 'Newton' method
    %               for calculating the root of the
    %               equation f(x)=0
    %
    % Parameters : f - input funtion f(x)
    %              f1 - first derivative of the function f(x)
    %              x0 - starting point
    %              tol - tolerance
    %              nmax - maximum number of interactions
    %              setprint - print output (1: yes)
    %
    % Return     : x-root and numeber of iteration (iter)
    %
    % Examples of Usage:
    %
    %    >> my_fun = @(x) 2.5*x^2 - 3*x + .5;
```

```
28      %    >> my_der = @(x) 5*x -3;
29      %    >> x0 = 0;
30      %    >> tolerance = .00001;
31      %    >> nmax = 20;
32      %    >> x = newton(my_fun, my_der, x0, tolerance, nmax);
33      %
34      %    >> Root at x = 0.200000
35      %
36
37      n = 1;
38      fx = feval(f, x0);
39      f1x = feval(f1, x0);
40      x = x0 - (fx/f1x);
41      if (setprint == 1)
42          fprintf('%2i \t %e \t %e \t %e  \t %e \n', n, x0, x, fx, feval(f
                ↪ , x));
43      end
44      delta = abs(x - x0);
45      tolx = tol * (1 + abs(x));
46
47      while(n < nmax && delta >= tolx)
48          n = n + 1;
49          x0 = x;
50          fx = feval(f, x0);
51          f1x = feval(f1, x0);
52          x = x0 - (fx/f1x);
53          if (setprint == 1)
54              fprintf('%2i \t %e \t %e \t %e  \t %e \n', n, x0, x, fx,
                    ↪ feval(f, x));
55          end
56          delta = abs(x - x0);
57          tolx = tol * (1 + abs(x));
58      end
59      if (delta >= tolx)
60          disp('Newton: the method does not converge');
61      end
62      if (setprint == 1)
63          % Show the last approximation considering the tolerance
64          froot = feval(f, x);
65          fprintf('\n x-root = %e produces f(x) = %e \n %i iterations\n',
```

```
            ↪ x, froot, n);
66          fprintf(' Approximation with tolerance = %e \n', tol);
67      end
68      iter = n;
69      return
```

Listing 5.3: Metodo di Newton

La funzione *function x = newton(f, f1, x0, tol, nmax)* sviluppata è stata testata su di una funzione $2.5x^2 - 3x + 0.5$, la cui derivata prima è $5x - 3$, di cui si conosce una sua radice.

```
my_fun = @(x) 2.5*x^2 - 3*x + .5;
my_der = @(x) 5*x -3;
x0 = -1;
nmax = 20;
setprint = 1;
tolerance = .00001;

[x, iter] = newton(my_fun, my_der, x0, tolerance, nmax, setprint);

% Newton's method
Iter    xi            x(i+1)      f(xi)        f(xi+1)
 1      -1.0000e+00   -2.5000e-01       6.0000e+00   1.4062e+00
 2      -2.5000e-01    8.0882e-02       1.4062e+00   2.7370e-01
 3       8.0882e-02    1.8633e-01       2.7371e-01   2.7799e-02
 4       1.8633e-01    1.9977e-01       2.7799e-02   4.5163e-04
 5       1.9977e-01    1.9999e-01       4.5163e-04   1.2733e-07
 6       1.9999e-01    2.0000e-01   1.2733e-07   1.0103e-14

x-root = 2.000000e-01 produces f(x) = 1.010303e-14
6 iterations
Approximation with tolerance = 1.000000e-05
```

Listing 5.4: Risultato dell'esecuzione del Metodo di Newton

5.3 Metodo delle Secanti

Il *metodo delle Secanti* appartiene ai metodi definiti di *"quasi-Newton"*, dato che rappresenta una variante del metodo di Newton che non richiede la valutazione della derivata prima della funzione ad ogni passo. Infatti, il metodo di Newton richiede ad ogni passo sia una valutazione funzionale della $f(x)$ e sia una valutazione della sua derivata prima $f^{'}(x)$, il quale comporta un ulteriore costo computazionale. Il metodo delle Secanti approssima la derivata prima della funzione nel modo seguente:

$$f^{'}(x) \approx \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}} \equiv \varphi_i \tag{5.2}$$

Generalizzando, l'iterazione i-esima assume la seguente forma:

$$x_{i+1} = \frac{f(x_i)x_{i-1} - f(x_{i-1})x_i}{f(x_i) - f(x_{i-1})}, \quad per \quad i = 1, 2, \cdots, \tag{5.3}$$

Il successivo algoritmo descrive l'implementazione del *metodo delle Secanti* con l'utilizzo del parametro di tolleranza *tol* e come criterio di arresto $\left|\Delta x_i\right| \leq tol \cdot (1 + \left|x_i\right|)$.

```
function [x, iter] = secanti(f, f1, x0, tol, nmax, setprint)
    %
    % function x = secanti(f, f1, x0, tol, nmax)
    %
    % Author : Alessandro Montaghi
    % Email  : alessandro.montaghi@gmail.com
    % Date   : Spring 2019
    % Course : Calcolo Numerico 2018/2019
    %
    % Function   : Secanti method
    %
    % Description:  This code calculates the 'Secanti' method
    %               for calculating the root of the
    %               equation f(x)=0
    %
    % Parameters : f - input funtion f(x)
    %              f1 - first derivative of the function f(x)
    %              x0 - starting point
    %              tol - tolerance
    %              nmax = maximum number of interactions
    %              setprint - print output (1: yes)
```

```
22      %
23      % Return     : x-root and numeber of iteration (iter)
24      %
25      % Examples of Usage:
26      %
27      %    >> my_fun = @(x) 2.5*x^2 - 3*x + .5;
28      %    >> my_der = @(x) 5*x -3;
29      %    >> x0 = 0;
30      %    >> tolerance = .00001;
31      %    >> nmax = 20;
32      %    >> x = secanti(my_fun, my_der, x0, tolerance, nmax);
33      %
34      %    >> Root at x = 0.200000
35      %
36
37      n = 0;
38      fx = feval(f, x0);
39      f1x = feval(f1, x0);
40      x = x0 -(fx/f1x);
41      delta = abs(x - x0);
42      tolx = tol * (1 + abs(x0));
43      if (setprint == 1)
44          disp('Iter    xi        x(i+1)      f(xi)       f(xi+1)');
45      end
46      while (n < nmax) && (delta > tolx)
47          n = n + 1;
48          fx0 = fx;
49          fx = feval(f, x);
50          x1 = (fx*x0 - fx0*x)/ (fx-fx0);
51          if (setprint == 1)
52              fprintf('%2i \t %e \t %e \t %e  \t %e \n', n, x, x1, fx,
                    ↪ feval(f, x1));
53          end
54          delta = abs(x - x1);
55          tolx = tol *(1 + abs(x1));
56          x0 = x;
57          x = x1;
58      end
59      if (delta > tolx)
60          disp('The method does not converge');
```

```
    end
    if (setprint == 1)
        % Show the last approximation considering the tolerance
        froot = feval(f, x);
        fprintf('\n x-root = %e produces f(x) = %e \n %i iterations\n',
            ↪ x, froot, n);
        fprintf(' Approximation with tolerance = %e \n', tol);
    end
    iter = n + 1;
    return
```

Listing 5.5: Metodo delle Secanti

La funzione *function x = secanti(f, f1, x0, tol, nmax)* sviluppata è stata testata su di una funzione $2.5x^2 - 3x + 0.5$, la cui derivata prima è $5x - 3$, di cui si conosce una sua radice.

```
my_fun = @(x) 2.5*x^2 - 3*x + .5;
my_der = @(x) 5*x -3;
x0 = 0;
nmax = 20;
tolerance = .00001;
setprint = 1;

[x, iter] = secanti(my_fun, my_der, x0, tolerance, nmax, setprint);

% Secanti's method
Iter    xi          x(i+1)      f(xi)         f(xi+1)
 1       1.6666e-01   1.9354e-01   6.9444e-02   1.3007e-02
 2       1.9354e-01   1.9974e-01   1.3007e-02   5.1232e-04
 3       1.9974e-01   1.9999e-01   5.1232e-04   4.0960e-06
 4       1.9999e-01   2.0000e-01   4.0960e-06   1.3107e-09

x-root = 2.000000e-01 produces f(x) = 1.310720e-09
4 iterations
Approximation with tolerance = 1.000000e-05
```

Listing 5.6: Risultato dell'esecuzione del Metodo delle Secanti

5.4 Metodo delle Corde

Il *metodo delle Corde* è anch'esso un metodo del tipo *"quasi-Newton"* che parte dal presupposto che, in presenza di una $f(x)$ sufficientemente regolare, in prossimità della radice $\hat{x}$, la derivata prima di $f(x)$ subisce basse variazioni. Quindi, se x_0 (ossia il punto di innesco) è prossimo alla radice si può convenientemente utilizzare la seguente approssimazione $f^{'}(x_i) \approx f^{'}(x_0) \approx \varphi_i$. Di conseguenza, l'iterazione i-esima assume la seguente forma:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_0)}, \quad per \quad i = 0, 1, 2, \cdots, \tag{5.4}$$

Il successivo algoritmo descrive l'implementazione del *metodo delle Corde* con l'utilizzo del parametro di tolleranza *tol* e come criterio di arresto $\left|\Delta x_i\right| \leq tol \cdot (1 + \left|x_i\right|)$.

```
function [x, iter] = corde(f, f1, x0, tol, nmax, setprint)
    %
    % function x = corde(f, f1, x0, tol, nmax)
    %
    % Author : Alessandro Montaghi
    % Email  : alessandro.montaghi@gmail.com
    % Date   : Spring 2019
    % Course : Calcolo Numerico 2018/2019
    %
    % Function   : Corde method
    %
    % Description:  This code calculates the 'corde' method for
    %               calculating the root of the equation f(x)=0
    %
    % Parameters : f - input funtion f(x)
    %              f1 - first derivative of the function f(x)
    %              x0 - starting point
    %              tol - tolerance
    %              nmax = maximum number of interactions
    %              setprint - print output (1: yes)
    %
    % Return     : x-root and numeber of iteration (iter)
    %
    % Examples of Usage:
    %
```

```
26      %    >> my_fun = @(x) 2.5*x^2 - 3*x + .5;
27      %    >> my_der = @(x) 5*x -3;
28      %    >> x0 = 0;
29      %    >> tolerance = .00001;
30      %    >> nmax = 20;
31      %    >> x = corde(my_fun, my_der, x0, tolerance, nmax);
32      %
33      %    >> Root at x = 0.200000
34      %
35
36      n = 0;
37      x = x0;
38      tolf = 0;
39
40      m = feval(f1, x);
41      tolx = tol *(1 + abs(x0));
42      delta = abs(realmax - x);
43      if (setprint == 1)
44          disp('Iter    xi        f(xi)');
45      end
46      while (n < nmax && delta > tolx)
47          fx = feval(f, x);
48          if (setprint == 1)
49              fprintf('%2i \t %e \t %e \n', n, x, fx);
50          end
51          tolf = tolx * abs(m);
52          if abs(fx) <= tolf
53              fprintf('Root at x = %e\n', x);
54              break
55          end
56          x1 = x-fx/m;
57          delta = abs(x - x1);
58          tolx = tol *(1 + abs(x1));
59          x = x1;
60          n = n + 1;
61      end
62      if (delta > tolx && abs(fx) > tolf)
63          disp('The method does not converge');
64      end
65      if (setprint == 1)
```

```
66          % Show the last approximation considering the tolerance
67          froot = feval(f, x);
68          fprintf('\n x = %e produces f(x) = %e \n %i iterations\n', x,
                ↪ froot, n);
69          fprintf(' Approximation with tolerance = %e \n', tol);
70      end
71      iter = n + 1;
72      return
```

Listing 5.7: Metodo delle Corde

La funzione *function x = corde(f, f1, x0, tol, nmax)* sviluppata è stata testata su di una funzione $2.5x^2 - 3x + 0.5$, la cui derivata prima è $5x - 3$, di cui si conosce una sua radice.

```
1  my_fun = @(x) 2.5*x^2 - 3*x + .5;
2  my_der = @(x) 5*x -3;
3  x0 = 0;
4  nmax = 20;
5  tolerance = .00001;
6  setprint = 1;
7
8  [x, iter] = corde(my_fun, my_der, x0, tolerance, nmax, setprint);
9
10 % Corde's method
11 Iter    xi               f(xi)
12  0       0.000000e+00    5.000000e-01
13  1       1.666667e-01    6.944444e-02
14  2       1.898148e-01    2.062972e-02
15  3       1.966914e-01    6.644594e-03
16  4       1.989063e-01    2.190488e-03
17  5       1.996364e-01    7.275025e-04
18  6       1.998789e-01    2.422070e-04
19  7       1.999597e-01    8.070308e-05
20  8       1.999866e-01    2.689741e-05
21 Root at x = 1.999866e-01
22
23 x = 1.999866e-01 produces f(x) = 2.689741e-05
24 8 iterations
25 Approximation with tolerance = 1.000000e-05
```

Listing 5.8: Risultato dell'esecuzione del Metodo delle Corde

6

Esercizio 6

Descrizione: *Utilizzare le function del precedente esercizio per determinare una approssimazione della radice della funzione* $f(x) = x - e^{-x}cos(x/100)$*, per* $tol = 10^{-i}, \quad i = 1, 2, ..., 12$*, partendo da* $x_0 = -1$*. Per il metodo di bisezione, utilizzare* $[-1, 1]$*, come intervallo di confidenza iniziale. Tabulare i risultati, in modo da confrontare le iterazioni richieste da ciascun metodo. Commentare il relativo costo computazionale.*

Svolgimento:

In questo esercizio si è approssimata la radice $\hat{x}$ della funzione $f(x) = x - e^{-x} \cdot cos(x/100)$ tramite il *metodo della Bisezione*, il *metodo di Newton*, il *metodo delle Secanti* e il *metodo delle Corde*, i cui algoritmi sono stati implementati tramite l'ambiente di sviluppo Matlab (vedere esercizio 5). Il criterio di arresto è definito come $|\Delta x_i| \leq tol \cdot (1 + |x_i|)$, con una tolleranza in ingresso pari a $tol = 10^{-1}, \quad i = 1, 2, ..., 12$, partendo da $x_0 = -1$.

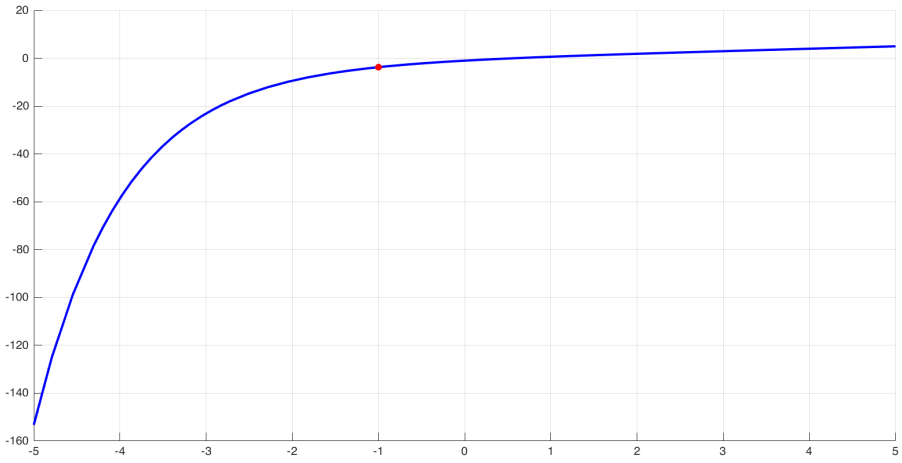


Figura 6.1: La funzione $f(x) = x - e^{-x}cos(x/100)$ è stata rappresentata usando l'intervallo $[-5, 5]$. In rosso è evidenziato il punto di innesco iniziale $x_0 = -1$.

Il codice in Matlab per plottare la funzione $f(x) = x - e^{-x}cos(x/100)$ è il seguente:

```
my_fun = @(x) x - exp(-x)*cos(x/100);
x0 = -1;
hold on
fplot(my_fun,[-5 5],'b', 'LineWidth',2)
plot(x0, feval(my_fun, x0), 'r.','MarkerSize',20)
hold off
grid on
```

Listing 6.1: Esercizio 6 - funzione

6.1 Metodo della Bisezione

Applicando il metodo della Bisezione (vedere *Listing 5.1*), con il punto di innesco collocato a $x_0 = -1$ e $[low, high] = [-1, 1]$, si sono ottenuti i seguenti risultati.

```
my_fun = @(x) x - exp(-x)*cos(x/100);
my_fun = @(x) x - exp(-x)*cos(x/100);
low = -1;
high = 1;
setprint = 0;

x_pos = [];
iter_pos = [];
for i = 1 : 12
    tol = 10^(-i);
    [x, iter] = bisection (my_fun, low, high, tol, setprint);
    x_pos = [x_pos x];
    iter_pos = [iter_pos iter];
    fprintf("tol: %e, x-root: %e \n", tol, x);
end

hold on
fplot(my_fun,[-1 1],'b')
for i = 1 : 12
    plot(x_pos(i), 0, 'r.','MarkerSize',20)
end
hold off
grid on
```

Listing 6.2: Esercizio 6 - Metodo della Bisezione

Di seguito sono riportati i risultati

Tolleranza iniziale (*tol*)	**Radice $\hat{x}$**
10^{-1}	5.000000000000000e-01
10^{-2}	5.625000000000000e-01
10^{-3}	5.664062500000000e-01
10^{-4}	5.671386718750000e-01
10^{-5}	5.671386718750000e-01
10^{-6}	5.671386718750000e-01
10^{-7}	5.671374797821045e-01
10^{-8}	5.671374797821045e-01
10^{-9}	5.671374704688787e-01
10^{-10}	5.671374702360481e-01
10^{-11}	5.671374702942558e-01
10^{-12}	5.671374702942558e-01

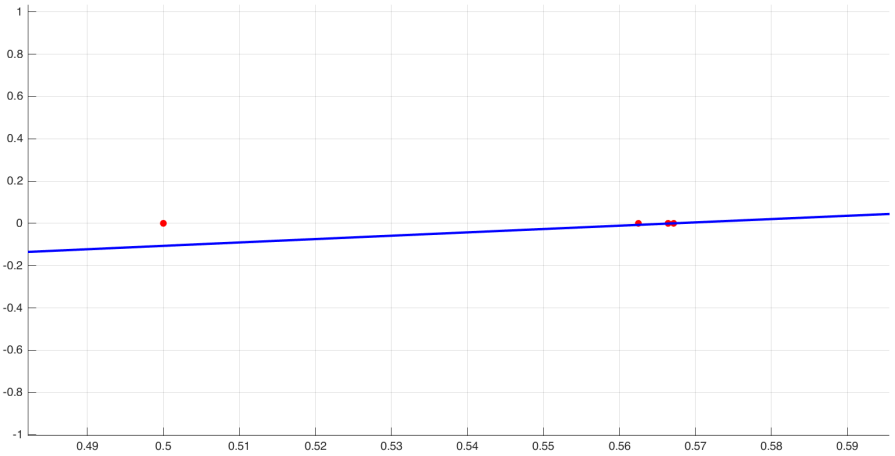


Figura 6.2: La figura riporta le varie approssimazioni della radice $\hat{x}$ trovate applicando il metodo di Bisezione per valori di $tol = 10^{-1}, \quad i = 1, 2, ..., 12$, partendo da $x_0 = -1$.

6.2 Metodo di Newton

Applicando il metodo di Newton (vedere *Listing 5.3*), con il punto di innesco collocato a $x_0 = -1$ per la funzione $f(x) = x - e^{-x}cos(x/100)$, la cui derivata prima è $(\sin\left(x \cdot 100^{-1}\right) \mathrm{e}^{-x}) \cdot 100^{-1} + \cos\left(x \cdot 100^{-1}\right) \mathrm{e}^{-x} + 1$, si sono ottenuti i seguenti risultati.

```
my_fun = @(x) x - exp(-x)*cos(x/100);
my_fun_1 = @(x) (sin(x/100)*exp(-x))/100 + cos(x/100)*exp(-x) + 1;
x0 = -1;
nmax = 20;
setprint = 0;

x_pos = [];
iter_pos = [];
for i = 1 : 12
    tol = 10^(-i);
    [x, iter] = newton(my_fun, my_fun_1, x0, tol, nmax, setprint);
    x_pos = [x_pos x];
    iter_pos = [iter_pos iter];
    fprintf("tol: %e, x-root: %e \n", tol, x);
end

hold on
fplot(my_fun,[-1 1],'b')
for i = 1 : 12
    plot(x_pos(i), 0, 'r.','MarkerSize',20)
end
hold off
grid on
```

Listing 6.3: Esercizio 6 - Metodo di Newton

Di seguito sono riportati i risultati

Tolleranza iniziale (*tol*)	**Radice $\hat{x}$**
10^{-1}	5.663058026182579e-01
10^{-2}	5.671373451065942e-01
10^{-3}	5.671373451065942e-01
10^{-4}	5.671374702931882e-01
10^{-5}	5.671374702931882e-01
10^{-6}	5.671374702931882e-01
10^{-7}	5.671374702931882e-01
10^{-8}	5.671374702931911e-01
10^{-9}	5.671374702931911e-01
10^{-10}	5.671374702931911e-01
10^{-11}	5.671374702931911e-01
10^{-12}	5.671374702931911e-01

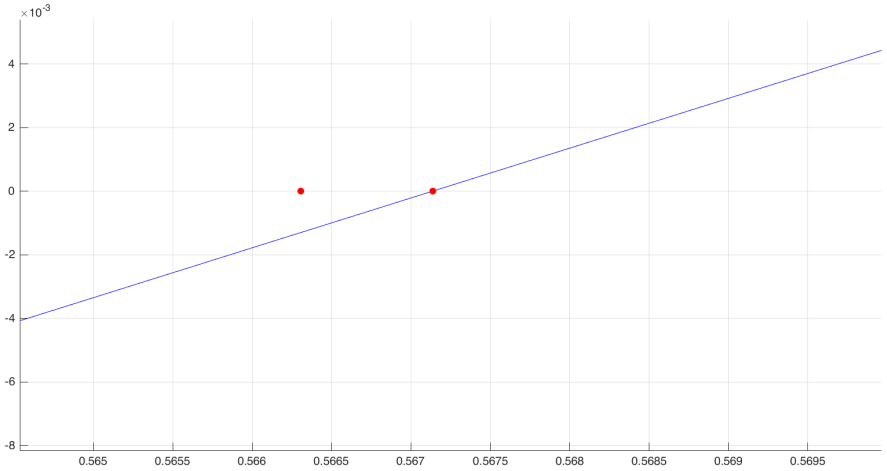


Figura 6.3: La figura riporta le varie approssimazioni della radice $\hat{x}$ trovate applicando il metodo di Newton per valori di $tol = 10^{-1}, \quad i = 1, 2, ..., 12$, partendo da $x_0 = -1$.

6.3 Metodo delle Secanti

Applicando il metodo delle Secanti (vedere *Listing 5.5*), con il punto di innesco collocato a $x_0 = -1$ per la funzione $f(x) = x - e^{-x}cos(x/100)$, la cui derivata prima è $(\sin\left(x \cdot 100^{-1}\right) \mathrm{e}^{-x}) \cdot 100^{-1} + \cos\left(x \cdot 100^{-1}\right) \mathrm{e}^{-x} + 1$, si sono ottenuti i seguenti risultati.

```
my_fun = @(x) x - exp(-x)*cos(x/100);
my_fun_1 = @(x) (sin(x/100)*exp(-x))/100 + cos(x/100)*exp(-x) + 1;
x0 = -1;
nmax = 20;
setprint = 0;

x_pos = [];
iter_pos = [];
for i = 1 : 12
    tol = 10^(-i);
    [x, iter] = secanti(my_fun , my_fun_1, x0, tol, nmax, setprint);
    x_pos = [x_pos x];
    iter_pos = [iter_pos iter];
    fprintf("tol: %e, x-root: %e \n", tol, x);
end

hold on
fplot(my_fun,[-1 1],'b')
for i = 1 : 12
    plot(x_pos(i), 0, 'r.','MarkerSize',20)
end
hold off
grid on
```

Listing 6.4: Esercizio 6 - Metodo delle Secanti

Di seguito sono riportati i risultati

Tolleranza iniziale (*tol*)	**Radice $\hat{x}$**
10^{-1}	5.662928457961099e-01
10^{-2}	5.671339926160822e-01
10^{-3}	5.671339926160822e-01
10^{-4}	5.671374697616252e-01
10^{-5}	5.671374697616252e-01
10^{-6}	5.671374702931907e-01
10^{-7}	5.671374702931907e-01
10^{-8}	5.671374702931907e-01
10^{-9}	5.671374702931907e-01
10^{-10}	5.671374702931911e-01
10^{-11}	5.671374702931911e-01
10^{-12}	5.671374702931911e-01

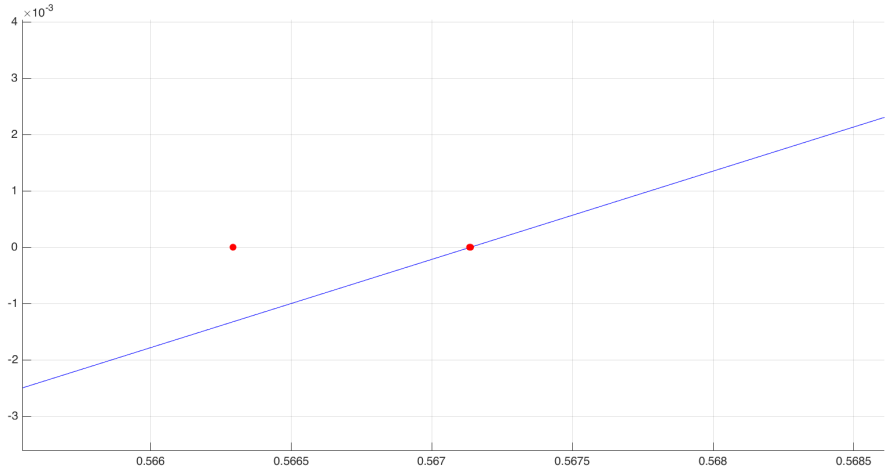


Figura 6.4: La figura riporta le varie approssimazioni della radice $\hat{x}$ trovate applicando il metodo delle Secanti per valori di $tol = 10^{-1}, \quad i = 1, 2, ..., 12$, partendo da $x_0 = -1$.

6.4 Metodo delle Corde

Applicando il metodo delle Corde (vedere *Listing 5.7*), con il punto di innesco collocato a $x_0 = -1$ per la funzione $f(x) = x - e^{-x}cos(x/100)$, la cui derivata prima è $(\sin\left(x \cdot 100^{-1}\right) \mathrm{e}^{-x}) \cdot 100^{-1} + \cos\left(x \cdot 100^{-1}\right) \mathrm{e}^{-x} + 1$, si sono ottenuti i seguenti risultati.

```
my_fun = @(x) x - exp(-x)*cos(x/100);
my_fun_1 = @(x) (sin(x/100)*exp(-x))/100 + cos(x/100)*exp(-x) + 1;
x0 = -1;
nmax = 20;
setprint = 0;

x_pos = [];
iter_pos = [];
for i = 1 : 12
    tol = 10^(-i);
    [x, iter] = corde(my_fun , my_fun_1, x0, tol, nmax, setprint);
    x_pos = [x_pos x];
    iter_pos = [iter_pos iter];
    fprintf("tol: %e, x-root: %e \n", tol, x);
end

hold on
fplot(my_fun,[-1 1],'b')
for i = 1 : 12
    plot(x_pos(i), 0, 'r.','MarkerSize',20)
end
hold off
grid on
```

Listing 6.5: Esercizio 6 - Metodo delle Corde

Di seguito sono riportati i risultati

Tolleranza iniziale (*tol*)	**Radice $\hat{x}$**
10^{-1}	4.021808606806709e-01
10^{-2}	5.365554984803566e-01
10^{-3}	5.637420065614376e-01
10^{-4}	5.669177360432487e-01
10^{-5}	5.671128657404283e-01
10^{-6}	5.671232371727831e-01
10^{-7}	5.671232371727831e-01
10^{-8}	5.671232371727831e-01
10^{-9}	5.671232371727831e-01
10^{-10}	5.671232371727831e-01
10^{-11}	5.671232371727831e-01
10^{-12}	5.671232371727831e-01

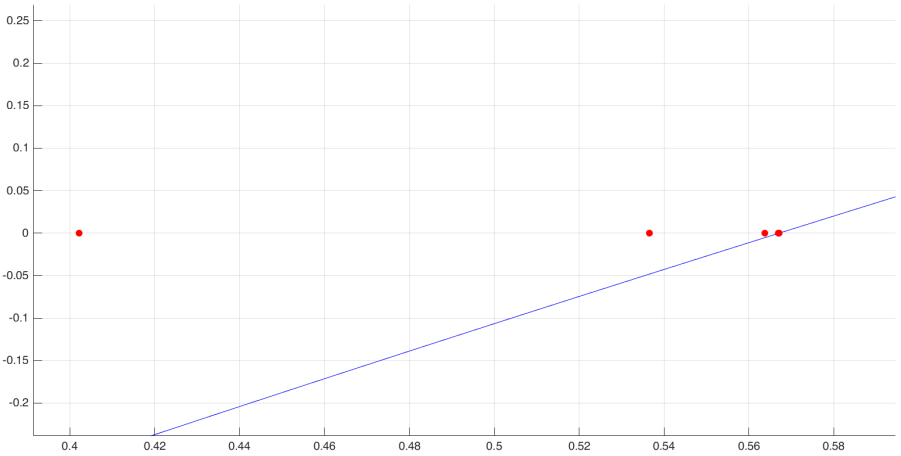


Figura 6.5: La figura riporta le varie approssimazioni della radice $\hat{x}$ trovate applicando il metodo delle Corde per valori di $tol = 10^{-1}, \quad i = 1, 2, ..., 12$, partendo da $x_0 = -1$.

6.5 Commenti finali

Le seguenti due tabelle riportando il risultato per le varie approssimazioni della radice $\hat{x}$, per le date tolleranze (*tol*), date tramite il *Metodo di Bisezione*, il *Metodo di Newton*, il *Metodo delle Secanti* ed il *Metodo delle corde*.

tol	**Bisezione**	**Newton**
10^{-1}	5.000000000000000e-01	5.663058026182579e-01
10^{-2}	5.625000000000000e-01	5.671373451065942e-01
10^{-3}	5.664062500000000e-01	5.671373451065942e-01
10^{-4}	5.671386718750000e-01	5.671374702931882e-01
10^{-5}	5.671386718750000e-01	5.671374702931882e-01
10^{-6}	5.671386718750000e-01	5.671374702931882e-01
10^{-7}	5.671374797821045e-01	5.671374702931882e-01
10^{-8}	5.671374797821045e-01	5.671374702931911e-01
10^{-9}	5.671374704688787e-01	5.671374702931911e-01
10^{-10}	5.671374702360481e-01	5.671374702931911e-01
10^{-11}	5.671374702942558e-01	5.671374702931911e-01
10^{-12}	5.671374702942558e-01	5.671374702931911e-01

tol	**Secanti**	**Corde**
10^{-1}	5.662928457961099e-01	4.021808606806709e-01
10^{-2}	5.671339926160822e-01	5.365554984803566e-01
10^{-3}	5.671339926160822e-01	5.637420065614376e-01
10^{-4}	5.671374697616252e-01	5.669177360432487e-01
10^{-5}	5.671374697616252e-01	5.671128657404283e-01
10^{-6}	5.671374702931907e-01	5.671232371727831e-01
10^{-7}	5.671374702931907e-01	5.671232371727831e-01
10^{-8}	5.671374702931907e-01	5.671232371727831e-01
10^{-9}	5.671374702931907e-01	5.671232371727831e-01
10^{-10}	5.671374702931911e-01	5.671232371727831e-01
10^{-11}	5.671374702931911e-01	5.671232371727831e-01
10^{-12}	5.671374702931911e-01	5.671232371727831e-01

La seguente tabella riporta il numero di iterazioni necessarie per le varie approssimazioni della radice $\hat{x}$, per le date tolleranze (*tol*), ottenute tramite il *Metodo di Bisezione*, il *Metodo di Newton*, il *Metodo delle Secanti* ed il *Metodo delle corde*.

tol	**Bisezione**	**Newton**	**Secanti**	**Corde**
10^{-1}	4	3	4	4
10^{-2}	7	4	5	7
10^{-3}	11	4	5	11
10^{-4}	15	5	6	16
10^{-5}	15	5	6	20
10^{-6}	15	5	7	21
10^{-7}	25	5	7	21
10^{-8}	25	6	7	21
10^{-9}	32	6	7	21
10^{-10}	35	6	8	21
10^{-11}	37	6	8	21
10^{-12}	37	6	8	21

La strategia del metodo di bisezione consiste nel dimezzare l'intervallo di partenza, selezionare tra i due sotto-intervalli ottenuti quello nel quale la funzione $f(x)$ cambia di segno agli estremi ed applicare ricorsivamente questa procedura all'ultimo intervallo selezionato. Quindi, questo metodo richiede ad ogni passo una sola valutazione della funzione $f(x)$, ma il suo ordine di convergenza risulta essere di tipo lineare. Per quanto riguarda il metodo di Newton, questo richiede ad ogni passo due valutazioni ossia una valutazione funzionale $f(x)$ ed un a valutazione della sua derivata prima $f^{'}(x)$. Comunque, anche se il metodo di Newton richiede un costo per iterazione più elevato, se la funzione $f(x)$ è sufficientemente regolare, questo metodo converge almeno quadraticamente verso le radici radici semplici. Comunque, il Metodo di Newton, nel caso in cui le radici siano multiple, subisce un peggioramento della convergenza. Il metodo delle Secanti ed il Metodo delle Corde risultano essere delle varianti del metodo di Newton che non richiedono la valutazione della derivata prima della funzione ad ogni passo e quindi risultano essere meno costosi (in termini computazionali) di quest'ultimo. Comunque, sia il Metodo delle Secanti che quello delle Corde risultano avere una convergenza di tipo lineare e quindi inferiore a quello di Newton.

7

Esercizio 7

Descrizione: *Calcolare la molteplicità della radice nulla della funzione* $f(x) = x^2 \cdot sin(x^2)$. *Confrontare, quindi, i metodi di Newton, Newton modificato, e di Aitken, per approssimarla per gli stessi valori di tol del precedente esercizio (ed utilizzando il medesimo criterio di arresto), partendo da* $x_0 = 1$. *Tabulare e commentare i risultati ottenuti.*

Svolgimento:

7.1 Calcolo della molteplicità della radice nulla per $f(x) = x^2 sin(x^2)$

In termini matematici, si definisce una radice di una funzione f un elemento $x \in \mathbb{R}$ (di solito indicata con il simbolo $\hat{x}$ oppure x^*), del dominio di f, tale che $f(\hat{x}) = 0$. Una radice $\hat{x}$ della equazione $f(\hat{x}) = 0$ ha *"molteplicità esatta"* $m \geq 1$, se:

$$f(\hat{x}) = f'(\hat{x}) = \cdots = f^{(m-1)}(\hat{x}) = 0, \quad se \quad f^{(m)}(\hat{x}) \neq 0. \tag{7.1}$$

Se $m = 1$ la radice $\hat{x}$ è definita *"semplice"*, mentre se $m \geq 2$, la radice si $\hat{x}$ si dice *"multipla"*.

Ritornando alla funzione assegnata dall'esercizio (il cui dominio è tutto l'insieme dei numeri Reali):

$$f(x) = x^2 sin(x^2), \quad con \quad x \in \mathbb{R} \tag{7.2}$$

Ponendo $x^2 sin(x^2) = 0$, la funzione presenta una radice per $\hat{x}_1 = 0$ e per i valori $\hat{x}_2 = \sqrt{\pi n}$ e $\hat{x}_3 = -\sqrt{\pi n}$, entrambe con $n \in \mathbb{N}\{0\}$. Tale risultato, si può facilmente evincere plottando la funzione ed i punti di intersezione di questa con l'asse delle x (ovvero dove vale $f(\hat{x}) = 0$).

```
my_fun = @(x) x.^2 * sin(x.^2);

x_pos = [];
for i = 1:31
    x = sqrt(pi*i);
    x_pos = [x_pos x];
end

x_neg = [];
for i = 1:31
    x = -sqrt(pi*i);
    x_neg = [x_neg x];
end

hold on
fplot(my_fun,[-10 10],'b')
plot(0, 0, 'r.','MarkerSize',12)
plot(x_pos,0,'r.','MarkerSize',12)
plot(x_neg,0,'r.','MarkerSize',12)
hold off
grid on
```

Listing 7.1: Codice per il plottaggio della funzione assegnata

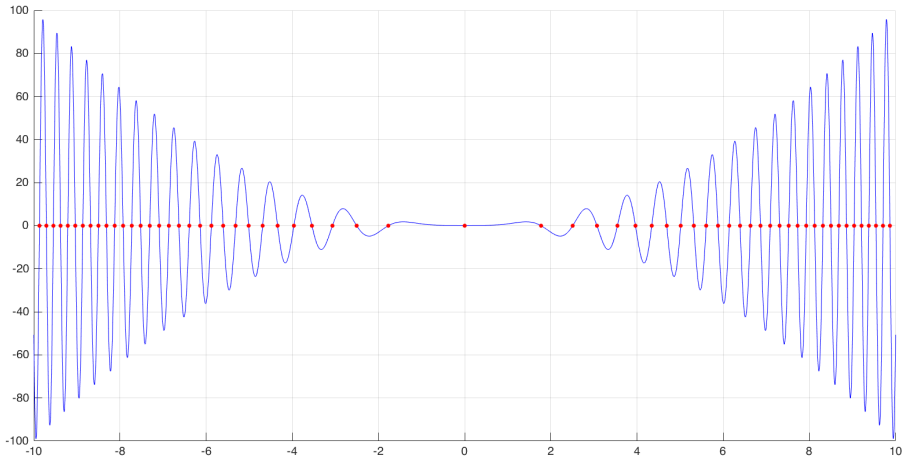


Figura 7.1: La funzione $f(x) = x^2 \cdot sin(x^2)$ è stata rappresentata usando l'intervallo $[-10, 10]$. In rosso sono evidenziati i punti dove la funzione interseca l'asse delle x.

Per il calcolo della molteplicità delle radici $\hat{x}_1 = 0$, $\hat{x}_2 = \sqrt{\pi n}$ e $\hat{x}_3 = -\sqrt{\pi n}$, entrambe con $n \in \mathbb{N}\,\{0\}$, calcoliamo la derivata prima, $f^1(x)$, della funzione:

$$f^1(x) = 2x \cdot (sin(x^2) + x^2 cos(x^2) \tag{7.3}$$

Quindi sostituendo i valori delle radici $\hat{x}$ di $f(x)$, si determina che:

$$f^1(\hat{x}_1) = 2 \cdot (0) \cdot (sin(0) + (0) \cdot cos(0)) = 0 \tag{7.4}$$

$$\begin{aligned} f^1(\hat{x}_2) &= 2 \cdot (\sqrt{\pi \cdot n}) \cdot (sin(\pi \cdot n) + (\pi \cdot n) \cdot cos(\pi \cdot n)) = \\ &= 2 \cdot (\sqrt{\pi \cdot n}) \cdot (0 + (\pi \cdot n)) = \\ &= 2 \cdot (\pi \cdot n) \cdot (\sqrt{\pi \cdot n}) \neq 0 \end{aligned} \tag{7.5}$$

$$\begin{aligned} f^1(\hat{x}_3) &= -2 \cdot (\sqrt{\pi \cdot n}) \cdot (sin(-\pi \cdot n) + (-\pi \cdot n) \cdot cos(-\pi \cdot n)) = \\ &= 2 \cdot (-\sqrt{\pi \cdot n}) \cdot (0 + (-\pi \cdot n)) = \\ &= 2 \cdot (\pi \cdot n) \cdot (\sqrt{\pi \cdot n}) \neq 0 \end{aligned} \tag{7.6}$$

Essendo $f^1(\hat{x}_2) \neq 0$ e $f^1(\hat{x}_3) \neq 0$, ossia $m = 1$, le radici corrispondenti ad $\hat{x}_2$ e $\hat{x}_3$ sono semplici. Proseguiamo con il calcolo della molteplicità della radice $\hat{x}_1$, derivando ulteriormente:

$$f^2(x) = \left(2 - 4x^4\right) \sin\left(x^2\right) + 10x^2 \cos\left(x^2\right) \tag{7.7}$$

Quindi, sostituendo nella $f^2(x)$ il valore di $\hat{x}_1$, si ottiene che:

$$\begin{aligned} f^2(\hat{x}_1) &= (2-0)\cdot sin(0) + 0 \cdot cos(0) = \\ &= 2 \cdot 0 + 0 \cdot 1 = 0 \end{aligned} \tag{7.8}$$

Derivando ulteriormente la $f^2(x)$, si ottiene:

$$f^3(x) = \left(24x - 8x^5\right)\cos\left(x^2\right) - 36x^3 \sin\left(x^2\right) \tag{7.9}$$

Sostituendo nella $f^3(x)$ il valore di $\hat{x}_1$, si ottiene che:

$$\begin{aligned} f^3(\hat{x}_1) &= (24 \cdot 0 - 8 \cdot 0) \cdot cos(o) - 36 \cdot 0 \cdot sin(0) = \\ &= 0 \cdot cos(0) - 36 \cdot 0 \cdot sin(0) = 0 \end{aligned} \tag{7.10}$$

Derivando ulteriormente la $f^3(x)$, si ottiene:

$$f^4(x) = \left(16x^6 - 156x^2\right)\sin\left(x^2\right) + \left(24 - 112x^4\right)\cos\left(x^2\right) \tag{7.11}$$

Infine, sostituendo nella $f^4(x)$ il valore di $\hat{x}_1$, si ottiene che:

$$\begin{aligned} f^4(\hat{x}_1) &= (16 \cdot 0 - 156 \cdot 0) \cdot sin(0) + (24 - 112 \cdot 0) \cdot cos(0) = \\ &= 0 \cdot 0 + 24 \cdot 1 = 24 \neq 0 \end{aligned} \tag{7.12}$$

Dato che $f^4(\hat{x}_1) = 24 \neq 0$, la molteplicità ($m$) di $\hat{x}_1$ risulta essere $m = 4$, da cui deriva che $\hat{x}_1$ è una radice multipla.

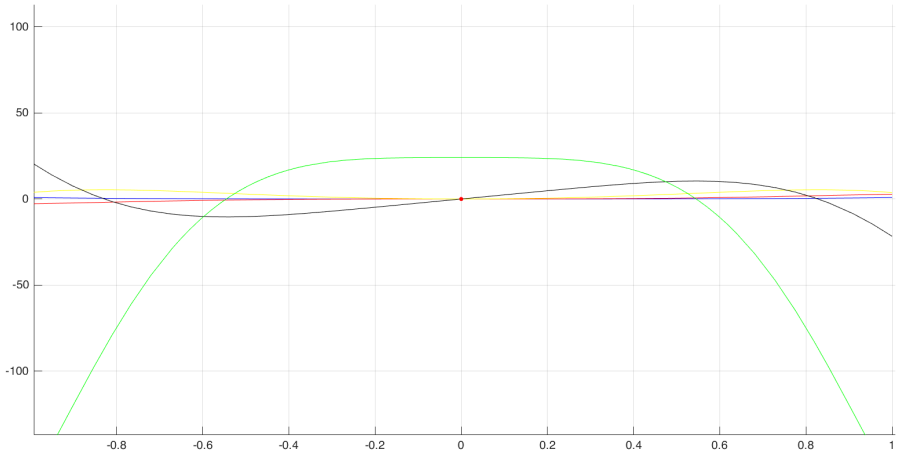


Figura 7.2: Nell'intervallo $[-1, 1]$ sono rappresentata la funzione $f(x)$ (blue) e le sue derivare $f^1(x)$ (red), $f^2(x)$ (yellow), $f^3(x)$ (Black) e $f^4(x)$ (green). In rosso è evidenziato la radice $\hat{x}_1$.

```
my_fun = @(x) x.^2 * sin(x.^2);
my_fun_1 = @(x) 2*x.*(sin(x.^2) + x.^2*cos(x.^2));
my_fun_2 = @(x) (2-4*x.^4)*sin(x.^2)+10*x.^2*cos(x.^2);
my_fun_3 = @(x) (24*x - 8*x.^5)*cos(x.^2)-36*x.^3*sin(x.^2);
my_fun_4 = @(x) (16*x.^6 - 156*x.^2)*sin(x.^2) + (24 - 112*x.^4)*cos(x
    ↪ .^2);

hold on
fplot(my_fun,[-1 1],'b','MarkerSize',20)
fplot(my_fun_1,[-1 1],'r','MarkerSize',20)
fplot(my_fun_2,[-1 1],'y','MarkerSize',20)
fplot(my_fun_3,[-1 1],'k','MarkerSize',20)
fplot(my_fun_4,[-1 1],'g','MarkerSize',20)
plot(0, 0, 'r.','MarkerSize',12)
hold off
grid on
```

Listing 7.2: Codice per il plottaggio della funzione e delle sue derivate

7.2 Confronto tra i metodi

Per il confronto dei metodi di Newton, Newton modificato, e di Aitken sulla funzione assegnata $f(x) = x^2 \cdot sin(x^2)$, partendo da $x_0 = 1$, si sono implementati i seguenti metodi in Matlab.

```
function x = newton(f, f1, x0, tol, nmax, setprint)
    %
    % function x = newton(f, f1, x0, tol, nmax, setprint)
    %
    % Author : Alessandro Montaghi
    % Email  : alessandro.montaghi@gmail.com
    % Date   : Spring 2019
    % Course : Calcolo Numerico 2018/2019
    %
    % Function   : Newton method
    %
    % Description:  This code calculates the 'Newton' method
    %               for calculating the root of the
```

```
14      %               equation f(x)=0
15      %
16      % Parameters : f - input funtion f(x)
17      %              f1 - first derivative of the function f(x)
18      %              x0 - starting point
19      %              tol - tolerance
20      %              nmax - maximum number of interactions
21      %              setprint - print output (1: yes, 0: no)
22      %
23      % Return     : x-root
24      %
25
26      n = 1;
27      fx = feval(f, x0);
28      f1x = feval(f1, x0);
29      x = x0 - (fx/f1x);
30      if (setprint == 1)
31          fprintf('%2i \t %e \t %e \t %e  \t %e \n', n, x0, x, fx, feval(f
                ↪ , x));
32      end
33      delta = abs(x - x0);
34      tolx = tol * (1 + abs(x));
35
36      while(n < nmax && delta >= tolx)
37          n = n + 1;
38          x0 = x;
39          fx = feval(f, x0);
40          f1x = feval(f1, x0);
41          x = x0 - (fx/f1x);
42          if (setprint == 1)
43              fprintf('%2i \t %e \t %e \t %e  \t %e \n', n, x0, x, fx,
                    ↪ feval(f, x));
44          end
45          delta = abs(x - x0);
46          tolx = tol * (1 + abs(x));
47      end
48      if (delta >= tolx)
49          disp('Newton: the method does not converge');
50      end
51      if (setprint == 1)
```

```
52          % Show the last approximation considering the tolerance
53          froot = feval(f, x);
54          fprintf('\n x-root = %e produces f(x) = %e \n %i iterations\n',
                ↪ x, froot, n);
55          fprintf(' Approximation with tolerance = %e \n', tol);
56      end
57      return
58  end
```

Listing 7.3: Newton

Eseguendo il *Metodo di Newton* per la funzione $f(x) = x^2 \cdot sin(x^2)$, la cui derivata prima è $f^1(x) = 2x \cdot (sin(x^2) + x^2 cos(x^2)$, secondo i parametri forniti dall'esercizio, si ottengono i seguenti risultati:

```
my_fun = @(x) x.^2 * sin(x.^2);
my_fun_1 = @(x) 2*x.*(sin(x.^2) + x.^2*cos(x.^2));
x0 = 1;
nmax = 20;
setprint = 0;
x_res = [];

% Newton
for i = 1 : 12
    tol = 10^(-i);
    x = newton(my_fun, my_fun_1, x0, tol, nmax, setprint);
    fprintf('x-root %e, tol: %e \n', x, tol);
    x_res = [x_res x];
end

x-root 3.843179e-01, tol: 1.000000e-01
x-root 2.880514e-02, tol: 1.000000e-02
x-root 2.883766e-03, tol: 1.000000e-03
x-root 2.883766e-03, tol: 1.000000e-04 (does not converge)
x-root 2.883766e-03, tol: 1.000000e-05 (does not converge)
x-root 2.883766e-03, tol: 1.000000e-06 (does not converge)
x-root 2.883766e-03, tol: 1.000000e-07 (does not converge)
x-root 2.883766e-03, tol: 1.000000e-08 (does not converge)
x-root 2.883766e-03, tol: 1.000000e-09 (does not converge)
x-root 2.883766e-03, tol: 1.000000e-10 (does not converge)
x-root 2.883766e-03, tol: 1.000000e-11 (does not converge)
x-root 2.883766e-03, tol: 1.000000e-12 (does not converge)
```

Listing 7.4: Esecuzione: Newton

Nel caso di radici multiple, la convergenza del *Metodo di Newton* risulta essere solo lineare. Per ripristinare la convergenza quadratica, nel caso in cui la molteplicità (m) della radice è nota, è possibile applicare il seguente metodo definito *Metodo di Newton modificato (per radici la cui molteplicità è nota)*:

```
function x = newton_modified(f, f1, x0, m, tol, nmax, setprint)
    %
    % function x = newton_modified(f, f1, x0, m, tol,
    %                              nmax, setprint)
    %
    % Author : Alessandro Montaghi
    % Email  : alessandro.montaghi@gmail.com
    % Date   : Spring 2019
    % Course : Calcolo Numerico 2018/2019
    %
    % Function   : newton modified method
    %
    % Description:  This code calculates the 'newton_modified'
    %               method for calculating the root of the
    %               equation f(x)=0 when the root has
    %               multiplicity known (m).
    %
    % Parameters : f - input funtion f(x)
    %              f1 - first derivative of the function f(x)
    %              x0 - starting point
    %              m - multiplicity of the root
    %              tol - tolerance
    %              nmax - maximum number of interactions
    %              setprint - print output (1: yes, 0: no)
    %
    % Return     : x-root
    %
    n = 1;
    fx = feval(f, x0);
    f1x = feval(f1, x0);
    x = x0 - m * (fx/f1x);
    if (setprint == 1)
        fprintf('%2i \t %e \t %e \t %e  \t %e \n', n, x0, x, fx, feval(f
            ↪ , x));
    end
```

```
35      delta = abs(x - x0);
36      tolx = tol * (1 + abs(x));
37
38      while(n < nmax && delta >= tolx)
39          n = n + 1;
40          x0 = x;
41          fx = feval(f, x0);
42          f1x = feval(f1, x0);
43          x = x0 - m * (fx/f1x);
44          if (setprint == 1)
45              fprintf('%2i \t %e \t %e \t %e  \t %e \n', n, x0, x, fx,
                    ↪ feval(f, x));
46          end
47          delta = abs(x - x0);
48          tolx = tol * (1 + abs(x));
49      end
50      if (delta >= tolx)
51          disp('Newton modified: the method does not converge');
52      end
53      if (setprint == 1)
54          % Show the last approximation considering the tolerance
55          froot = feval(f, x);
56          fprintf('\n x-root = %e produces f(x) = %e \n %i iterations\n',
                ↪ x, froot, n);
57          fprintf(' Approximation with tolerance = %e \n', tol);
58      end
59      return
60  end
```

Listing 7.5: Newton modificato

Eseguendo il *Metodo di Newton modificato* per la funzione $f(x) = x^2 \cdot sin(x^2)$, la cui derivata prima è $f^1(x) = 2x \cdot (sin(x^2) + x^2 cos(x^2))$, secondo i parametri forniti dall'esercizio, si ottengono i seguenti risultati:

```
my_fun = @(x) x.^2 * sin(x.^2);
my_fun_1 = @(x) 2*x.*(sin(x.^2) + x.^2*cos(x.^2));
x0 = 1;
m = 4;
nmax = 20;
setprint = 0;
x_res = [];

% Newton modified
for i = 1 : 12
    tol = 10^(-i);
    x = newton_modified(my_fun, my_fun_1, x0, m, tol, nmax, setprint);
    fprintf('x-root %e, tol: %e \n', x, tol);
    x_res = [x_res x];
end

x-root 0.000000e+00, tol: 1.000000e-01
x-root 0.000000e+00, tol: 1.000000e-02
x-root 0.000000e+00, tol: 1.000000e-03
x-root 0.000000e+00, tol: 1.000000e-04
x-root NaN, tol: 1.000000e-05
x-root NaN, tol: 1.000000e-06
x-root NaN, tol: 1.000000e-07
x-root NaN, tol: 1.000000e-08
x-root NaN, tol: 1.000000e-09
x-root NaN, tol: 1.000000e-10
x-root NaN, tol: 1.000000e-11
x-root NaN, tol: 1.000000e-12
```

Listing 7.6: Esecuzione: Newton modificato

Il *Metodo di accelerazione di Aitken* è una procedura a due livelli nel quale, nel livello interno vengono eseguiti due passi del *Metodo di Newton*, mentre nel livello esterno viene eseguito il passo di *accelerazione*, che consente di estrapolare una approssimazione più accurata della radice. Tale approssimazione, rappresenta a sua volta un nuovo punto iniziale per il livello interno. Applicando questo metodo, la successione delle approssimazioni converge quadraticamente verso la radice $\hat{x}$.

```
function x = aitken(f, f1, x0, tol, nmax)
    %
    % x = aitken(f, f1, x0, tol, nmax)
    %
    % Author : Alessandro Montaghi
    % Email  : alessandro.montaghi@gmail.com
    % Date   : Spring 2019
    % Course : Calcolo Numerico 2018/2019
    %
    % Function   : Aitken's method
    %
    % Description:  This code calculates the 'aitken' method
    %               for calculating the root of the
    %               equation f(x)= 0
    %
    % Parameters : f - input funtion f(x)
    %              f1 - first derivative of the function f(x)
    %              x0 - starting point
    %              tol - tolerance
    %              nmax = maximum number of interactions
    %
    % Return     : x-root
    %
    % Examples of Usage:
    %
    %    >> my_fun = @(x) 2.5*x^2 - 3*x + .5;
    %    >> my_der = @(x) 5*x -3;
    %    >> x0 = 0;
    %    >> tolerance = .00001;
    %    >> nmax = 20;
    %    >> x = aitken(f, f1, x0, tol, nmax);
    %
    %    >> Root at x = 0.200000
```

```
34      %
35
36      i = 0;
37      x = x0;
38      vai = 1;
39
40      while(i < nmax) && vai
41          i = i + 1;
42          x0 = x;
43          fx = feval(f, x0);
44          f1x = feval(f1, x0);
45          x1 = x0 -(fx/f1x);
46          fx = feval(f, x1);
47          f1x = feval(f1, x1);
48          x = x1 -(fx/f1x);
49          x = (x * x0 - x1^2)/(x -2*x1 + x0);
50          delta = abs(x0-x);
51          tolx = tol*(1 + abs(x));
52          vai = delta >= tolx;
53      end
54      if (vai)
55          disp('The method does not converge');
56      end
57      return
58  end
```

Listing 7.7: Aitken

Eseguendo il *Metodo di Aitken* per la funzione $f(x) = x^2 \cdot sin(x^2)$, la cui derivata prima è $f^1(x) = 2x \cdot (sin(x^2) + x^2 cos(x^2))$, secondo i parametri forniti dall'esercizio, si ottengono i seguenti risultati:

```
my_fun = @(x) x.^2 * sin(x.^2);
my_fun_1 = @(x) 2*x.*(sin(x.^2) + x.^2*cos(x.^2));
x0 = 1; nmax = 20;
for i = 1 : 12
    tol = 10^(-i);
    x = aitken(my_fun, my_fun_1, x0, tol, nmax);
    fprintf('root %e, tol: %e \n', x, tol);
end
x-root 6.492909e-19, tol: 1.000000e-01
x-root 6.492909e-19, tol: 1.000000e-02
x-root 6.492909e-19, tol: 1.000000e-03
x-root 0.000000e+00, tol: 1.000000e-04
x-root 0.000000e+00, tol: 1.000000e-05
x-root 0.000000e+00, tol: 1.000000e-06
x-root 0.000000e+00, tol: 1.000000e-07
x-root 0.000000e+00, tol: 1.000000e-08
x-root 0.000000e+00, tol: 1.000000e-09
x-root 0.000000e+00, tol: 1.000000e-10
x-root 0.000000e+00, tol: 1.000000e-11
x-root 0.000000e+00, tol: 1.000000e-12
```

Listing 7.8: Esecuzione: Aitken

7.3 Commenti finali

In tabella si riportano i vari risultati a seconda della tolleranza data,

tol	Newton	Newton Modificato	Aitken
10^{-1}	3.8431788060706e-01	0	6.4929086188118e-19
10^{-2}	2.8805139309384e-02	0	6.4929086188118e-19
10^{-3}	2.8837663030348e-03	0	6.4929086188118e-19
10^{-4}	2.8837663030348e-03	0	0
10^{-5}	2.8837663030348e-03	NaN	0
10^{-6}	2.8837663030348e-03	NaN	0
10^{-7}	2.8837663030348e-03	NaN	0
10^{-8}	2.8837663030348e-03	NaN	0
10^{-9}	2.8837663030348e-03	NaN	0
10^{-10}	2.8837663030348e-03	NaN	0
10^{-11}	2.8837663030348e-03	NaN	0
10^{-12}	2.8837663030348e-03	NaN	0

Essendo il caso di radici multiple ($m = 4$), il problema risulta essere malcondizionato e la convergenza del *Metodo di Newton* risulta essere solo lineare e non più almeno quadratica come nel caso di radici semplici ($m = 1$). Essendo la molteplicità della radice nota, è possibile ripristinare la convergenza quadratica del metodo di Netwon applicando il *Metodo di Newton modificato*, tramite il quale, da come si osserva in tabella, si ottiene con un solo passo la soluzione corretta. Infine, tramite il *Metodo di accellerazione di Aitken*, che al suo interno esegue una iterazione a due livelli, è possibile ripristinare la convergenza quadratica del metodo di Newton anche nel caso di radici multiple con molteplicità incognita.

8

Esercizio 8

Descrizione: *Scrivere una function Matlab che, data in ingresso una matrice* ***A****, restituisca una matrice,* ***LU****, che contenga l'informazione sui suoi fattori* ***L*** *ed* ***U****, ed un vettore* ***p*** *contenente la relativa permutazione, della fattorizzazione LU con pivoting parziale di A:* ***function [LU,p] = palu(A)****. Curare particolarmente la scrittura e l'efficienza della function.*

Svolgimento:

8.1 Algoritmo: *[LU, p] = palu(A)*

Il *"Metodo di fattorizzazione LU"* di una matrice quadrata $A \in \mathbb{R}^{n \times n}$, non singolare (ossia con $det(A) \neq 0$), presenta il limite di arrestarsi quando nella matrice A è presente un elemento $a_{k,k}^{(k)}$ uguale a zero, ossia quando la matrice ha un minore principale nullo e quindi non sono soddisfatte le ipotesi del *"Teorema dell'esistenza della fattorizzazione LU"* (cioè, se A è non singolare, la fattorizzazione LU esiste sse tutti i minori principali di A sono non nulli). Tramite la *"Tecnica del pivoting parziale"*, descritta dall'algoritmo 8.1, se A è una matrice nonsingolare, allora esiste una matrice di permutazione P (i cui elementi sono 0 e 1) tale che PA è fattorizzazibile LU. In particolare, L (*L-ower triangular matrix*) è rappresentata da una matrice triangolare inferiore a diagonale unitaria e U (*U-pper triangular matrix*) è una matrice triangolare superiore. Per rendere la fattorizzazione tramite il *"pivoting parziale"* efficiente da un punto di vista di occupazione di memoria, è possibile utilizzare un'unica matrice di uscita (ossia LU) mantenendo, nella parte strettamente inferiore, gli elementi appartenenti alla matrice L, ad esclusione degli elementi della diagonale unitaria, e nella parte superiore, gli elementi della matrice

U. Infine, invece di restituire una matrice di Permutazione P, l'algoritmo restituisce un vettore di permutazione p, nel quale è salvato l'ordine delle righe permutate.

```
function [LU, p] = palu(A)
    %
    % function [LU, p] = palu(A)
    %
    % Author : Alessandro Montaghi
    % Email  : alessandro.montaghi@gmail.com
    % Date   : Spring 2019
    % Course : Calcolo Numerico 2018/2019
    %
    % Function   : LU Factorization with Partial Pivoting
    %
    % Description: use Gaussian elimination with partial
    %              pivoting (the interchanging of rows)
    %              to find the LU decomposition
    %
    % Parameters : A - square matrix (n-by-n)
    %
    % Return     : the LU matrix and the
    %              associated permutation vector p
    %
    % Examples of Usage:
    %
    %    >> A = [1 2 3; 4 5 6; 7 8 0];
    %    >> [LU, p] = palu(A);
    %    LU =
    %        7.0000    8.0000         0
    %        0.1429    0.8571    3.0000
    %        0.5714    0.5000    4.5000
    %
    %    p =
    %        3     1     2
    %
    %

    LU = A;
    % get the number of rows and columns of the matrix.
    [m, n] = size(LU);
```

```
38      if abs(m - n) > 0
39          error('wrong data input: matrix is not square');
40      end
41      p = 1:n;
42      for i = 1:n-1
43          [mi, ki] = max(abs(A(i:n,i)));
44          if mi==0
45              error('Matrix is singular');
46          end
47          ki = ki+i-1;
48          if ki > i
49              LU([i, ki],:)= LU([ki, i],:);
50              p([i,ki]) = p([ki,i]);
51          end
52          LU(i+1:n,i) = LU(i+1:n,i)/LU(i,i);
53          LU(i+1:n,i+1:n) = LU(i+1:n,i+1:n) - LU(i+1:n,i) * LU(i,i+1:n);
54      end
55      return
56  end
```

Listing 8.1: Codice Matlab Esercizio 8

8.2 Esecuzione

La funzione *function [LU,p] = palu(A)* sviluppata è stata testata su di una matrice A di cui si conosce il risultato della fattorizzazione LU.

```
A = [1 2 3; 4 5 6; 7 8 0];
disp('The Input matrix is:')
disp(A);
[LU, p] = palu(A);
disp('The LU matrix is:')
disp(LU);
disp('The permutaion vector p is:')
disp(p);
The Input matrix is:
     1     2     3
     4     5     6
     7     8     0

The LU matrix is:
    7.0000    8.0000         0
    0.1429    0.8571    3.0000
    0.5714    0.5000    4.5000

The permutaion vector p is:
     3     1     2
```

Listing 8.2: Risultato dell'esecuzione Esercizio 8

9

Esercizio 9

Descrizione: *Scrivere una function Matlab che, data in ingresso la matrice* ***LU*** *ed il vettore* ***p*** *creati dalla function del precedente esercizio, ed il termine noto del sistema lineare* ***Ax = b****, ne calcoli la soluzione:* ***function x = lusolve(LU,p,b)****. Curare particolarmente la scrittura e l'efficienza della function.*

Svolgimento:

9.1 Algoritmo: *x = lusolve(LU, b, p)*

In questo esercizio, è stato sviluppato un metodo in Matlab che data in ingresso una matrice di tipo *LU* (dove *L* è una matrice quadrata triangolare inferiore non singolare a diagonale unitaria ed *U* è una matrice triangolare superiore non singolare), derivata da una fattorizzazione *LU* con pivoting parziale (lo scambio riguarda solo le righe della matrice), il vettore *p* delle permutazioni e il vettore *b* dei termini noti, calcola la soluziona del sistema lineare $Ax = b$.

```
1  function x = lusolve(LU, b, p)
2      %
3      % function x = lusolve(LU, b, p)
4      %
5      % Author : Alessandro Montaghi
6      % Email  : alessandro.montaghi@gmail.com
7      % Date   : Spring 2019
```

```
8   % Course : Calcolo Numerico 2018/2019
9   %
10  % Function    : solve the LU matrix derived by factorization
11  %               with partial Pivoting (the interchanging
12  %               of rows).
13  %
14  % Description: given a LU matrix n-by-n (where U is an upper
15  %               triangular matrix and L is a lower triangular
16  %               matrix with a unit diagonal), derived from a
17  %               LU Factorization with Partial Pivoting, the
18  %               'lusolve' function returns the solutions of
19  %               the linear system.
20  %
21  % Parameters : LU - square matrix (n-by-n)
22  %               b -  column vector of constant terms
23  %               p - permutation vector
24  %
25  % Return     : x vector of solutions.
26  %
27  % Examples of Usage:
28  %
29  %    >> LU = [7 8 0; 0.1429 0.8571 3; 3 0.5 4.5];
30  %    >> p = [3, 1, 2];
31  %    >> b = [5, 7, 8]';
32  %    >> x = lusolve(LU, b, p);
33  %    >> x =
34  %               -3.5556
35  %                4.1111
36  %                0.1111
37  %
38
39  A = LU;
40  % number of rows and columns of the matrix
41  [m, n] = size(A);
42  % length of the vector of constant terms
43  bLen = length(b);
44  % length of the permutation vector
45  pLen = length(p);
46  if abs(m-n) > 0 || abs(bLen-n) > 0 || abs(bLen-pLen) > 0
47      error('Wrong input data');
```

```
48      end
49      % reorder b in function of pivot
50      b = b(p);
51      x = b(:);
52      % Lower Triangular
53      for i = 1:n
54          x(i+1:n) = x(i+1:n) - A(i+1:n,i) * x(i);
55      end
56      % Upper Triangular
57      for i = n:-1:1
58          x(i) = x(i)/A(i,i);
59          x(1:i-1) = x(1:i-1)-A(1:i-1,i) * x(i);
60      end
61      return
```

Listing 9.1: Codice Matlab Esercizio 9

9.2 Esecuzione

La funzione *function x = lusolve(LU, b, p)* sviluppata è stata testata su di una matrice *LU* ed il vettore *p* creati dalla *function [LU,p] = palu(A)*, ed il vettore dei termini noti del sistema lineare $Ax = b$, di cui si conosce la soluzione (x).

```
A = [1 2 3; 4 5 6; 7 8 0];
b = [5, 7, 8]';
disp('The Input matrix is:')
disp(A);
[LU, p] = palu(A);
disp('The LU matrix is:')
disp(LU);
disp('The permutaion vector p is:')
disp(p);
x = lusolve(LU, b, p);
disp('The linear system solution is:')
disp(x);
The Input matrix is:
     1     2     3
     4     5     6
     7     8     0

The LU matrix is:
    7.0000    8.0000         0
    0.1429    0.8571    3.0000
    0.5714    0.5000    4.5000

The permutaion vector p is:
     3     1     2

The linear system solution is:
   -3.5556
    4.1111
    0.1111
```

Listing 9.2: Risultato dell'esecuzione Esercizio 9

10

Esercizio 10

Descrizione: *Scaricare la function cremat che crea sistemi lineari $n \times n$ la cui soluzione è il vettore $v = (1 \cdots n)^T$. Eseguire, quindi, lo script Matlab (vedere esercizio) per testare le function dei precedenti esercizi. Confrontare i risultati ottenuti con quelli attesi, e dare una spiegazione esauriente degli stessi.*

Svolgimento:

Nell'analisi di questo esercizio, si è stata utilizzata la funzione *function [A,b] = cremat(n,k,simme)* che crea sistemi lineari $n \times n$ la cui soluzione è il vettore $v = (1 \cdots n)^T$.

```
function [A,b] = cremat(n, k, simme)
    %
    %  [A,b] = cremat(n, k, simme)
    %

    if nargin == 1
        sigma = 1/n;
    else
        sigma = 10^(-k);
    end
    rng(0);
    [q1,r1] = qr(rand(n));
    if nargin==3
        q2 = q1';
    else
```

```
16          [q2,r1] = qr(rand(n));
17      end
18      A = q1*diag([sigma 2/n:1/n:1])*q2;
19      x = [1:n]';
20      b = A*x;
```

Listing 10.1: Funzione cremat

Il seguente script in Matlab è stato quindi eseguito e i risultati ottenuti, $v = (1, 2, \cdots, 15)^T$, per ogni $k = 1, 2, \cdots, 15$ della funzione *cremat* sono stati tabulati nella seguente pagina:

```
n = 10;
x = zeros(n,15);
for k = 1:15
    [A,b] = cremat(n, k);
    [LU,p] = palu(A);
    x(:,i) = lusolve(LU, b, p);
end
```

Listing 10.2: Esecuzione Esercizio 10

Dai risultati ottenuti si evince che per $k \geq 12$ si verificano delle *perturbazioni* sui dati in uscita. Infatti, considerando il *Condizionamento del problema* nel caso delle matrici, questo può essere espresso dalla seguente relazione:

$$\frac{\|\Delta x\|}{\|x\|} \leq \|A\| \cdot \left\|A^{-1}\right\| \cdot \left(\frac{\|\Delta b\|}{\|b\|} + \frac{\|\Delta A\|}{\|A\|}\right) \tag{10.1}$$

Dove

- $\|\Delta x\| \,/\, \|x\|$ rappresenta una sorta di *errore relativo* sui dati in uscita (risultato).
- $\|\Delta A\| \,/\, \|A\|$ e $\|\Delta b\| \,/\, \|b\|$ rappresentano ad una sorta di *errori relativi* sui dati in ingresso.
- $k(A) = \|A\| \cdot \left\|A^{-1}\right\|$ rappresenta il numero di condizione della matrice A. Se $k(A)$ è piccolo allora la matrice è *ben condizionata*, mentre se $k(A) >> 1$ allora la matrice è *malcondizionata*.

Quindi, per valori di $1 \leq k < 12$ si deduce che $k(A)$ è piccolo e quindi la matrice risulta *ben condizionata*. Invece all'aumentare di k (ossia per $k \geq 12$) si hanno valori $k(A) >> 1$ comportando delle perturbazioni sui dati in uscita (ossia il risultato) dato che la matrice risulta essere *malcondizionata*.

k:	1	2	3	4	5
1	1.0000	1.0000	1.0000	1.0000	1.0000
2	2.0000	2.0000	2.0000	2.0000	2.0000
3	3.0000	3.0000	3.0000	3.0000	3.0000
4	4.0000	4.0000	4.0000	4.0000	4.0000
5	5.0000	5.0000	5.0000	5.0000	5.0000
6	6.0000	6.0000	6.0000	6.0000	6.0000
7	7.0000	7.0000	7.0000	7.0000	7.0000
8	8.0000	8.0000	8.0000	8.0000	8.0000
9	9.0000	9.0000	9.0000	9.0000	9.0000
10	10.0000	10.0000	10.0000	10.0000	10.0000

k:	6	7	8	9	10
1	1.0000	1.0000	1.0000	1.0000	1.0000
2	2.0000	2.0000	2.0000	2.0000	2.0000
3	3.0000	3.0000	3.0000	3.0000	3.0000
4	4.0000	4.0000	4.0000	4.0000	4.0000
5	5.0000	5.0000	5.0000	5.0000	5.0000
6	6.0000	6.0000	6.0000	6.0000	6.0000
7	7.0000	7.0000	7.0000	7.0000	7.0000
8	8.0000	8.0000	8.0000	8.0000	8.0000
9	9.0000	9.0000	9.0000	9.0000	9.0000
10	10.0000	10.0000	10.0000	10.0000	10.0000

k:	11	12	13	14	15
1	1.0000	0.9999	1.0022	1.0060	0.8926
2	2.0000	2.0002	1.9935	1.9825	2.3131
3	3.0000	2.9999	3.0021	3.0056	2.8997
4	4.0000	4.0002	3.9931	3.9815	5.0000
5	5.0000	4.9995	5.0174	5.0466	4.1649
6	6.0000	6.0003	5.9914	5.9769	6.4133
7	7.0000	7.0002	6.9934	6.9823	7.3175
8	8.0000	8.0000	7.9994	7.9983	8.0300
9	9.0000	9.0002	8.9940	8.9839	9.2881
10	10.0000	10.0001	9.9976	9.9936	10.1138

11

Esercizio 11

Descrizione: *Scrivere una function Matlab che, data in ingresso una matrice* $A \in \mathbb{R}^{m \times n}$*, con* $m \geq n = rank(A)$*, restituisca una matrice,* ***QR****, che contenga l'informazione sui fattori* ***Q*** *ed* ***R*** *della fattorizzazione* ***QR*** *di* ***A****:* ***function QR = myqr(A)****. Curare particolarmente la scrittura e l'efficienza della function.*

Svolgimento:

11.1 Algoritmo: *QR = myqr(A)*

Una matrice A rettangolare $(m \times n)$ può essere fattorizzata della forma $A = QR$, dove la matrice $Q \in \mathbb{R}^{m \times m}$ è una matrice di tipo ortogonale (una matrice Q è detta ortogonale quando la sua matrice inversa Q^{-1} è uguale alla matrice trasposta A^T), mentre $\hat{R} \in \mathbb{R}^{n \times n}$ è una matrice triangolare superiore e non singolare.

La *fattorizzazione QR* trova riscontro in molte applicazioni quali ad esempio la ricerca di autovalori o nell'algoritmo dei minimi quadrati. La decomposizione $A = QR$ può essere ottenuta con differenti tecniche. L'algoritmo sviluppato in Matlab implementa il *Metodo di fattorizzazione QR di Householder*.

```
1 function QR = myqr(A)
2     %
3     % function QR = myqr(A)
4     %
5     % Author : Alessandro Montaghi
6     % Email  : alessandro.montaghi@gmail.com
```

```
% Date   : Spring 2019
% Course : Calcolo Numerico 2018/2019
%
% Function   : QR Factorization using
%              Householder's methods
%
% Description: The qr function performs the
%              orthogonal-triangular decomposition
%              of a matrix. This factorization is
%              useful for both square and rectangular
%              matrices.
%
% Parameters : matrix A (n-by-n or m-by-n
%              with m >= n = rank(A))
%
% Return     : the QR matrix.
%

QR = A;
% get the number of rows and columns of the matrix.
[m, n] = size(QR);
if m < n
    error('The number rows is not equal or greater than the number
        ↪ of columns');
end
for i = 1: n
    alpha = norm(QR(i:m, i));
    if alpha == 0
        error('Matrix has not max Rank');
    end
    if QR(i, i) >= 0
        alpha = -alpha;
    end
    V1 = QR(i,i) - alpha;
    QR(i, i) = alpha;
    QR(i+1:m, i) = QR(i+1:m, i)/V1;
    beta = V1/alpha;
    V = [1; QR(i+1:m,i)];
    QR(i:m, i+1:n) = QR(i:m,i+1:n)+(beta*V) * (V'*QR(i:m,i+1:n));
end
```

```
46      return
```

Listing 11.1: Codice Matlab Esercizio 11

11.2 Esecuzione

La funzione *function QR = myqr(A)* sviluppata è stata testata su di una matrice $A \in \mathbb{R}^{4\times3}$ di cui si conosce il risultato della *fattorizzazione QR*.

```
1  A = [1 3 1; 1 3 7; 1 -1 -4; 1 -1 2];
2  disp('The Input matrix is:')
3  disp(A);
4  QR = myqr(A);
5  disp('The QR matrix is:')
6  disp(QR);
7  The Input matrix is:
8       1     3     1
9       1     3     7
10      1    -1    -4
11      1    -1     2
12
13 The QR matrix is:
14     -200.e+00    -2.00e+00    -3.00e+00
15      3.33e-01    -4.00e+00    -4.99e+00
16      3.33e-01    -5.00e-01    -6.00e+00
17      3.33e-01    -5.00e-01     9.99e-01
```

Listing 11.2: Risultato dell'esecuzione Esercizio 11

12

Esercizio 12

Descrizione: *crivere una function Matlab che, data in ingresso la matrice* ***QR*** *creata dalla function del precedente esercizio, ed il termine noto del sistema lineare* $\boldsymbol{Ax = b}$*, ne calcoli la soluzione nel senso dei minimi quadrati: function* ***x = qrsolve(QR,b)****. Curare particolarmente la scrittura e l'efficienza della function.*

Svolgimento:

12.1 Algoritmo: *x = qrsolve(QR, b)*

In questo esercizio, è stato sviluppato una funzione in Matlab, definita come *function x = qrsolve(QR, b)*, che data in ingresso una matrice di tipo *QR* (dove $Q \in \mathbb{R}^{m \times m}$ è una matrice di tipo ortogonale e $\hat{R} \in \mathbb{R}^{n \times n}$ è una matrice triangolare superiore non singolare), derivata da una fattorizzazione *QR* tramite il *Metodo di Householder*, il vettore *b* dei termini noti, calcola la soluziona del sistema lineare $Ax = b$.

```
1  function x = qrsolve(A, b)
2      %
3      % function x = qrsolve(A,b)
4      %
5      % Author : Alessandro Montaghi
6      % Email  : alessandro.montaghi@gmail.com
7      % Date   : Spring 2019
8      % Course : Calcolo Numerico 2018/2019
9      %
```

```
10      % Function   : QR Factorization using
11      %              Householder's methods
12      %
13      % Description: The qrsolve solves the linear
14      %              system (QR)X = B
15      %
16      % Parameters : A - unitary matrix A = Q*R
17      %              b -  column vector of constant terms
18      %
19      % Return     : x vector of solutions.
20      %
21
22      % number of rows and columns of the matrix
23      [m,n] = size(A);
24      % length of the vector of constant terms
25          bLen = length(b);
26      if abs(bLen - m) > 0
27          error('Wrong input data');
28      end
29      x = b;
30      for i = 1:n
31          v = [1; A(i+1:m,i)];
32          beta = 2/(v'*v);
33          x(i:m) = x(i:m) -( beta*( v'*x(i:m) ) )*v;
34      end
35      x = x(1:n);
36      for i = n:-1:1
37          x(i) = x(i)/A(i,i);
38          if i > 1
39              x(1:i-1) = x(1:i-1)-x(i)*A(1:i-1,i);
40          end
41      end
42      return
```

Listing 12.1: Codice Matlab Esercizio 12

12.2 Esecuzione

La funzione *function x = qrsolve(A, b)* sviluppata è stata testata su di una matrice *QR*, ottenuta tramite la *QR = myqr(A)*, utilizzando il vettore dei termini noti del sistema lineare $Ax = b$, di cui si conosce la soluzione (x).

```
A = [1 3 1; 1 3 7; 1 -1 -4; 1 -1 2];
b = [1, 3, 4, 5]';
QR = myqr(A);
disp('The QR matrix is:')
disp(QR);
disp('The constant terms are:')
disp(b);
x = qrsolve(QR, b);
disp('The solution is:')
disp(x);

The QR matrix is:
   -2.0000   -2.0000   -3.0000
    0.3333   -4.0000   -5.0000
    0.3333   -0.5000   -6.0000
    0.3333   -0.5000    1.0000

The constant terms are:
     1
     3
     4
     5

The solution is:
    3.8125
   -0.9375
    0.2500
```

Listing 12.2: Risultato dell'esecuzione Esercizio 12

13

Esercizio 13

Descrizione: *Scaricare la function cremat1 che crea sistemi lineari $m \times n$, con $m \geq n$, la cui soluzione (nel senso dei minimi quadrati) è il vettore $x = (1 \cdots n)^T$. Eseguire, quindi, lo script Matlab (vedere esercizio) per testare le function dei precedenti esercizi.*

Svolgimento:

Nell'analisi di questo esercizio, si è stata utilizzata la funzione *function [A,b] = cremat1(m,n)* che crea sistemi lineari $m \times n$, con $m \geq n$, la cui soluzione (nel senso dei minimi quadrati) è il vettore $x = (1, 2, \cdots, n)^T$.

```
function [A,b] = cremat1(m, n)
    %
    %  [A,b] = cremat1(m, n)
    %
    rng(0);
    A = rand(m, n);
    [q, r] = qr(A);
    b = r * [1:n]';
    b(n + 1:m) = rand(m - n, 1);
    b = q*b;
    return
```

Listing 13.1: Funzione cremat1

Il seguente script in Matlab è stato quindi eseguito

```
format long e

for n = 5:10
    xx = [1:n]';
    for m = n:n+10
        [A,b] = cremat1(m,n);
        QR = myqr(A);
        x = qrsolve(QR,b);
        disp([m n norm(x-xx)])
    end
end
```

Listing 13.2: Esecuzione Esercizio 10

Ite:	m	n	norm(x-xx)	Ite:	m	n	norm(x-xx)
1	5	5	1.556072131157320e-13	**12**	6	6	5.387968501360199e-14
2	6	5	2.147557060165057e-14	**13**	7	6	1.288671977272211e-14
3	7	5	1.295540364239626e-14	**14**	8	6	2.605880519086710e-14
4	8	5	2.491752066750447e-14	**15**	9	6	1.700345617112888e-14
5	9	5	5.277940006822452e-15	**16**	10	6	1.374705388827034e-14
6	10	5	1.166526922752426e-14	**17**	11	6	2.100405820894741e-14
7	11	5	9.678699938266253e-15	**18**	12	6	1.282680016781883e-14
8	12	5	4.022931084256207e-15	**19**	13	6	1.057920795372844e-14
9	13	5	5.626103786990878e-15	**20**	14	6	5.621720378622258e-15
10	14	5	5.388872059480397e-15	**21**	15	6	8.358438485664404e-15
11	15	5	4.720733054568282e-15	**22**	16	6	6.620504818089212e-15

Ite:	m	n	norm(x-xx)	Ite:	m	n	norm(x-xx)
23	7	7	2.751494640974127e-14	**34**	8	8	9.405181470485344e-14
24	8	7	5.360434541403836e-14	**35**	9	8	2.270100685624472e-14
25	9	7	9.272854936269104e-15	**36**	10	8	4.137186180368597e-14
26	10	7	2.152458757787511e-14	**37**	11	8	3.244967483735887e-14
27	11	7	2.651551021153896e-14	**38**	12	8	2.895533164773297e-14
28	12	7	1.875488421897247e-14	**39**	13	8	1.401697789135048e-14
29	13	7	1.041954807470275e-14	**40**	14	8	1.209888431736885e-14
30	14	7	2.360888674514792e-14	**41**	15	8	3.775411299608581e-14
31	15	7	1.645528807760935e-14	**42**	16	8	1.512513347095137e-14
32	16	7	7.695055015927580e-15	**43**	17	8	1.154044659076355e-14
33	17	7	1.575108741300131e-14	**44**	18	8	1.360283742757016e-14

Ite:	m	n	norm(x-xx)	Ite:	m	n	norm(x-xx)
45	9	9	7.025738323617081e-14	**56**	10	10	6.126751452305436e-14
46	10	9	7.118321999924466e-14	**57**	11	10	1.234524572132209e-13
47	11	9	5.895901019066522e-14	**58**	12	10	6.826165435821775e-14
48	12	9	6.920671200143359e-14	**59**	13	10	2.702771770213499e-14
49	13	9	2.216000707355663e-14	**60**	14	10	4.810533718274760e-14
50	14	9	1.492826867807971e-14	**61**	15	10	3.951894959524226e-14
51	15	9	4.352357467474990e-14	**62**	16	10	5.550593293777077e-14
52	16	9	2.845984775847675e-14	**63**	17	10	2.355792802380179e-14
53	17	9	1.653896970447028e-14	**64**	18	10	1.929234137798180e-14
54	18	9	2.178301711174186e-14	**65**	19	10	2.189814655995394e-14
55	19	9	2.634014362416955e-14	**66**	20	10	1.879919374704706e-14

14

Esercizio 14

Descrizione: *Scrivere un programma che implementi efficientemente il calcolo del polinomio interpolante su un insieme di ascisse distinte.*

Svolgimento:

Sia $f : [a, b] \subset \mathbb{R} \to \mathbb{R}$ una funzione la cui forma funzionale è non nota e dato un insieme di ascisse tra loro distinte $a \leq x_0 < x_1 < \cdots < x_n \leq b$, note le seguenti coppie di dati (x_i, f_i) con $i = 0, 1, \cdots, n$ e $f_i \equiv f(x_i)$ esiste ed è unico il polinomio (di interpolazione) $p(x) \in \Pi_n$ (dove Π_n è l'insieme dei polinomi di grado al più n) che soddisfa il seguente vincolo di interpolazione $p(x_i) = f_i$ con $i = 0, 1, \cdots, n$.

$$\exists! p(x) \in \Pi_n : P(x_i) = fi \equiv f(x_i), \ con\ i = 0, 1, \cdots, n. \tag{14.1}$$

14.1 Forma di Lagrange

Considerano la *"Forma di Lagrange"* il seguente polinomio soddisfa i vincoli di interpolazione (14.1):

$$p(x) = \sum_{k=0}^{n} f_k \cdot Lkn(x) \tag{14.2}$$

Dove la *"Base di Lagrange"* è costituita dai seguenti polinomi:

$$L_{kn}(x) = \prod_{j=0, j\neq k}^{n} \frac{x - x_j}{x_k - x_j}, \; con\; k = 0, 1, \cdots, n. \tag{14.3}$$

14.1.1 Algoritmo: *y = lagrange(xi, fi, x)*

Il polinomio interpolante nella *"Forma di Lagrange"* appena descritta, può essere tradotta nel seguente algoritmo in Matlab:

```
function y = lagrange(xi,fi, x)
    %
    % y = lagrange(xi, fi, x)
    %
    % Author : Alessandro Montaghi
    % Email  : alessandro.montaghi@gmail.com
    % Date   : Spring 2019
    % Course : Calcolo Numerico 2018/2019
    %
    % Function   : Lagrange Interpolator Polynomial
    %
    % Description:  This code calculates interpolator Polynomial
    %               N-1th order with the Lagrange's method
    %
    % Parameters : xi - vector of abscissas.
    %              fi - matching vector of ordinates.
    %              x  - the target to be interpolated.
    %
    % Return     : solution from the Lagrange interpolation
    %
    % Examples of Usage:
    %
    %    given the function myFun = @(x) x^3 - 3*x + 3;
    %    >> xi = [-3 -2 -1 0 1 2 3];
    %    >> fi = [-15 1 5 3 1 5 21];
    %    >> x1 = -1.65;
    %    >> x2 = .2;
    %    >> y1 = lagrange(xi, fi, x1);
    %    >> y2 = lagrange(xi, fi, x2);
    %    The results are:
```

```
31      %    >> y1 = 3.4579
32      %    >> y2 = 2.4080
33      %
34
35      n = length(xi);
36      m = length(fi);
37      if (n ~= m)
38          error('inconsistent data');
39      end
40      for i = 1 : n-1
41          for j = i+1 : n
42              if xi(i) == xi(j)
43                  error('abscissas not distinct');
44              end
45          end
46      end
47      y = zeros(size(x));
48      for i = 1:n
49          if fi(i) ~= 0
50              Li = 1;
51              for k = [1:i-1 i+1:n]
52                  Li = Li.*(x -xi(k))/(xi(i)-xi(k));
53              end
54              y = y + fi(i) * Li;
55          end
56      end
57      return
```

Listing 14.1: Codice Matlab per l'interpolazione polinomiale mediante la Forma di Lagrange

14.1.2 Esecuzione

La funzione $y = newton(xi, fi, x)$ sviluppata è stata testata su dei dati (x_i, f_i) con $i = 0, 1, \cdots, n$ di una funzione di cui non si conosce la forma.

```
xi = [-3 -2 -1 0 1 2 3];
fi = [-15 1 5 3 1 5 21];

x1 = -1.65;
y1 = lagrange(xi,fi,x1);

x2 = .2;
y2 = lagrange(xi,fi,x2);

myFun = @(x) x^3 - 3*x + 3;
hold on
fplot(myFun,[-4 23],'--r','LineWidth',1)
plot(xi, fi, 'ro', x1, y1, 'bo', x2, y2, 'bo','MarkerSize',9)
axis([-4 4 -17 23])
title('y = x^3 - 3x + 3')
xlabel('x')
ylabel('y')
grid on
```

Listing 14.2: Esecuzione del codice

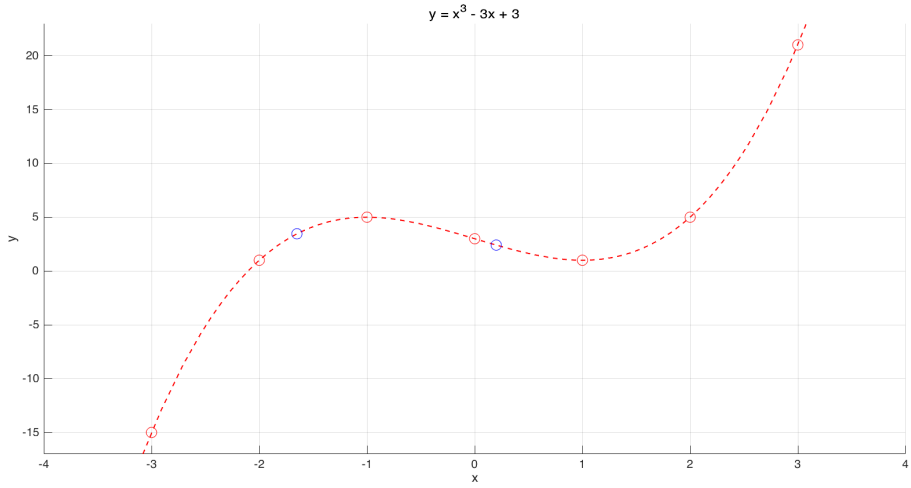


Figura 14.1: La funzione $f(x) = x^3 - 3 * x + 3$ è stata interpolata mediante il Metodo di Lagrange. In blue sono evidenziati i punti interpolati.

14.2 Forma di Newton

Considerano la *"Forma di Netwon"* il seguente polinomio soddisfa i vincoli di interpolazione (14.1):

$$p(x) \equiv p_n(x) = \sum_{r=0}^{n} f\left[x_0, x_1, \cdots, x_r\right] w_r(x) \tag{14.4}$$

Dove la quantità $f\left[x_0, x_1, \cdots, x_r\right]$ è detta *"differenza divisa di ordine r"* della funzione $f(x)$ sulle ascisse $x_0, x_1, \cdots, x_r$ ed è definita nel modo seguente:

$$f\left[x_0, x_1, \cdots, x_r\right] = \sum_{k=0}^{r} \frac{f_k}{\prod_{j=0, j\neq k}^{r}(x_k - x_j)} \tag{14.5}$$

La *"Base di Netwon"*, $w_r(x)$, è definita nel seguente modo ricorsivo:

$$\begin{aligned}
&w_0(x) \equiv 1, \\
&w_1(x) = (x - x0)w_0(x) = (x - x_0), \\
&w_2(x) = (x - x1)w_1(x) = (x - x_1)(x - x_0), \\
&\cdots \\
&w_{k+1}(x) = (x - x_k)w_k(x), \ con\ k = 0, 1, 2, \cdots
\end{aligned}$$

14.2.1 Algoritmo: *y = newton(xi, fi, x)*

Il polinomio interpolante nella *"Forma di Newton"* appena descritta, può essere tradotta nel seguente algoritmo in Matlab:

```
function y = newton(xi, fi, x)
    %
    % y = newton(xi, fi, x)
    %
    % Author : Alessandro Montaghi
    % Email  : alessandro.montaghi@gmail.com
    % Date   : Spring 2019
    % Course : Calcolo Numerico 2018/2019
```

```
9   %
10  % Function   : Newton Interpolator Polynomial
11  %
12  % Description:  This code calculates interpolator Polynomial
13  %               N-1th order with the Newton's method
14  %
15  % Parameters : xi - vector of abscissas.
16  %              fi - matching vector of ordinates.
17  %              x  - the target to be interpolated.
18  %
19  % Return     : solution from the Newton interpolation
20  %
21  % Examples of Usage:
22  %
23  %    given the function f = @(x) x^3 - 3*x + 3;
24  %    >> xi = [-3 -2 -1 0 1 2 3];
25  %    >> fi = [-15 1 5 3 1 5 21];
26  %    >> x1 = -1.65;
27  %    >> x2 = .2;
28  %    >> y1 = newton(xi, fi, x1);
29  %    >> y2 = newton(xi, fi, x2);
30  %    The results are:
31  %    >> y1 = 3.4579
32  %    >> y2 = 2.4080
33  %
34
35  n = length(xi);
36  m = length(fi);
37  if (n ~= m)
38      error('inconsistent data');
39  end
40  for i = 1 : n-1
41      for j = i+1 : n
42          if xi(i) == xi(j)
43              error('abscissas not distinct');
44          end
45      end
46  end
47  f = divdif(xi, fi);
48  y = f(n);
```

```
49      for i = n-1: -1: 1
50          y = y.* ( x - xi(i)) + f(i);
51      end
52      return
53
54  function f = divdif(xi, fi)
55      % Newton's divided difference
56      n = length(xi);
57      f = fi;
58      for i = 1: n-1
59          for j = n : -1 : i+1
60              f(j) = ( f(j) - f(j - 1) ) / ( xi(j) - xi(j - i) );
61          end
62      end
63      return
```

Listing 14.3: Codice Matlab per l'interpolazione polinomiale mediante la Forma di Newton

14.2.2 Esecuzione

La funzione $y = newton(xi, fi, x)$ sviluppata è stata testata su dei dati (x_i, f_i) con $i = 0, 1, \cdots, n$ di una funzione di cui non si conosce la forma.

```
xi = [-3 -2 -1 0 1 2 3];
fi = [-15 1 5 3 1 5 21];

x1 = -1.65;
y1 = newton(xi,fi,x1);

x2 = .2;
y2 = newton(xi,fi,x2);

myFun = @(x) x^3 - 3*x + 3;
hold on
fplot(myFun,[-4 23],'--r','LineWidth',1)
plot(xi, fi, 'ro', x1, y1, 'bo', x2, y2, 'bo','MarkerSize',9)
axis([-4 4 -17 23])
title('y = x^3 - 3x + 3')
xlabel('x')
ylabel('y')
grid on
```

Listing 14.4: Esecuzione del codice

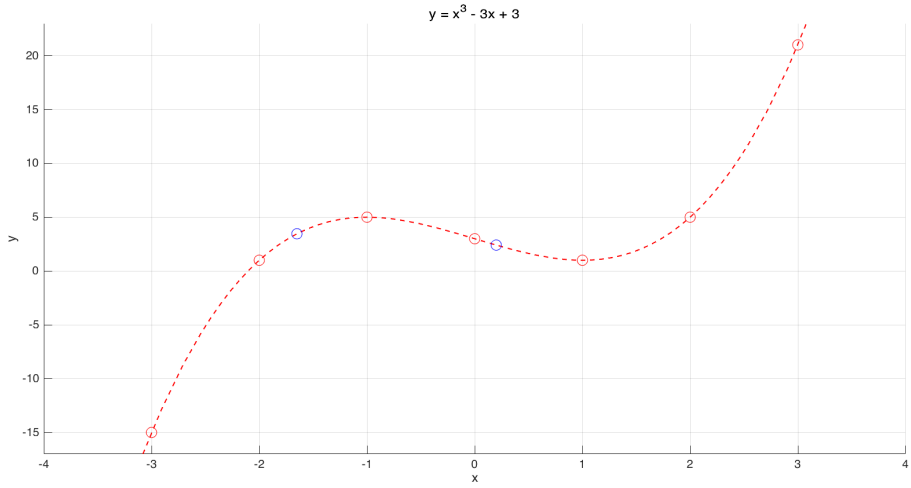


Figura 14.2: La funzione $f(x) = x^3 - 3 * x + 3$ è stata interpolata mediante il Metodo di Newton. In blue sono evidenziati i punti interpolati.

15

Esercizio 15

Descrizione: *Scrivere un programma che implementi efficientemente il calcolo del polinomio interpolante di Hermite su un insieme di ascisse distinte.*

Svolgimento:
Sia $f : [a, b] \subset \mathbb{R} \to \mathbb{R}$ una funzione la cui forma funzionale è non nota e dato un insieme di ascisse tra loro distinte nella forma $a \leq x_0 = x_0 < x_1 = x_1 < \cdots < x_n = x_n \leq b$, esiste ed è unico il *"polinomio interpolante di Hermite"* $p(x) \in \Pi_{2n+1}$ (dove Π_{2n+1} è l'insieme dei polinomi di grado al più $2n + 1$) che soddisfa il seguenti vincoli di interpolazione $p(x_i) = f(x_i)$ e $p^{'}(x_i) = f^{'}(x_i)$ con $i = 0, 1, \cdots, n$.

$$\exists! p(x) \in \Pi_{2n+1} : P(x_i) = f(x_i) \wedge p^{'}(x_i) = f^{'}(x_i), \ con\ i = 0, 1, \cdots, n. \tag{15.1}$$

In generale, il polinomio di Hermite sarà dato dalla seguente forma:

$$\begin{aligned} P_H(x) = {} & f[x_0] + f[x_0, x_0](x - x_0) + \\ & f[x_0, x_0, x_1](x - x_0)^2 + \\ & f[x_0, x_0, x_1, x_1](x - x_0)^2(x - x_1) + \\ & f[x_0, x_0, x_1, x_1, x_2](x - x_0)^2(x - x_1)^2 + \\ & + \cdots + \\ & f[x_0, x_0, x_1, x_1, x_2, \cdots, x_n, x_n](x - x_0)^2(x - x_1)^2 \cdots (x - x_{n-1})^2(x - x_n) \end{aligned}$$

15.1 Algoritmo: *y = hermite(xi, fi, x)*

Il *"polinomio interpolante di Hermite"* appena descritto, può essere tradotto nel seguente algoritmo in Matlab:

```
function y = hermite(xi, ff1, x)
    %
    % y = hermite(xi, ff1, x)
    %
    % Author : Alessandro Montaghi
    % Email  : alessandro.montaghi@gmail.com
    % Date   : Spring 2019
    % Course : Calcolo Numerico 2018/2019
    %
    % Function   : Hermite interpolation.
    %
    % Description: function that computes the coefficient of
    %              the hermite interpolation.
    %
    % Parameters : xi - set of x values.
    %              ff1 - set of f(x) and first derivative
    %                    values.
    %              x -  point where to evaluate the polynomial.
    %
    n = length(xi);
    % even index values
    f = ff1( 2 : 2 : end);
    % odd index values
    f1 = ff1(1 : 2 : end);
    if n ~= length(f) || n ~= length(f1)
        error('Inconsistent data');
    end
    for i = 1 : n-1
        for j= i + 1  :n
            if xi(i) == xi(j)
                error('Abscissas not distinct');
            end
        end
    end
    % Repeat copies of array elements
```

```
    xixi = repelem(xi, 2);
    ff = hermiteDividedDifferences(xixi, ff1);
    n = length(xixi);
    % generalized Horner
    y = ff(n);
    for i = n-1 : -1 : 1
        y = y.*(x-xixi(i)) + ff(i);
    end
    return

function [f] = hermiteDividedDifferences(x, f)
    % This function creates a vector of
    % divided differences for Hermite polynomials
    len = length(x);
    for i = len-1:-2:3
        f(i) = (f(i) - f(i-2))/(x(i) - x(i-2));
    end
    for i=2:len-1
        for j=len:-1:i+1
            f(j) = (f(j) - f(j-1))/(x(j) - x(j-i));
        end
    end
    return
```

Listing 15.1: Codice Matlab per l'interpolazione polinomiale mediante la Hermite

15.2 Esecuzione

Si riporta un caso di applicazione della funzione di *hermite* su di una funzione nota, ossia $f(x) = sin(x)$ nei punti $x = 0, \pi/2, \pi, 3 * \pi/2e2 * \pi$.

```
xi = [0 pi/2 pi 3*pi/2 2*pi];
ff1 = [0 1 1 0 0 -1 -1 0 0 1];

y1 = hermite(xi, ff1, pi/6);
y2 = hermite(xi, ff1, pi/5);
y3 = hermite(xi, ff1, pi/4);
y4 = hermite(xi, ff1, pi/3);
y5 = hermite(xi, ff1, 3*pi/4);
y6 = hermite(xi, ff1, 5*pi/6);
```

```
10
11 xRes = [pi/6 pi/5 pi/4 pi/3 3*pi/4 5*pi/6];
12 yRes = [y1 y2 y3 y4 y5 y6];
13
14 for i = 1: length(xRes)
15    fprintf('x: %f - f(x): %f\n', xRes(i), yRes(i));
16 end
17
18 fun = @(x) sin(x);
19 fun1 = @(x) cos(x);
20
21 hold on
22 fplot(fun,[-1,4],'b','LineWidth',1);
23 plot(xRes, yRes, 'r.','MarkerSize',24);
24 grid on
25 hold off
26
27 >> x: 0.523599 - f(x): 0.500064
28 >> x: 0.628319 - f(x): 0.587846
29 >> x: 0.785398 - f(x): 0.707155
30 >> x: 1.047198 - f(x): 0.866047
31 >> x: 2.356194 - f(x): 0.707110
32 >> x: 2.617994 - f(x): 0.500001
```

Listing 15.2: Esecuzione della funzione di hermite

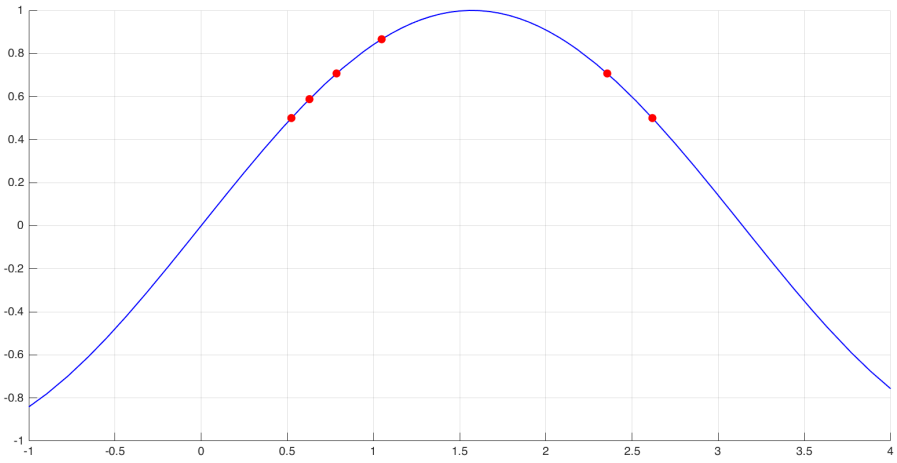


Figura 15.1: La funzione $f(x) = sin(x)$ è stata interpolata mediante il Metodo di Hermite. In rosso sono evidenziati i punti interpolati.

16

Esercizio 16

Descrizione: *Scrivere un programma che implementi efficientemente il calcolo di una spline cubica naturale interpolante su una partizione assegnata.*

Svolgimento:

Una spline di grado m e nodi $(x_0, x_1, \cdots, x_n)$ è una funzione polinomiale a tratti del tipo $S : [a, b] \subset \mathbb{R} \to \mathbb{R}$. In dettaglio e fissata una partizione Δ dell'intervallo originario $[a, b]$ del tipo $\Delta = \{a = x_0 < x_1 < \cdots < x_n = b\}$, diremo che $S_m(x)$ è una spline di grado m che interpola la funzione $f(x)$ nei nodi della partizione Δ, se valgono le seguenti proprietà:

1. $S_m(x) \in C^{m-1}$ sull'intervallo $[a, b]$.
2. $S|_{[x_{i-1}, x_i]}(x) \in \Pi_m$ con $i = 1, 2, ..., n$
3. $S_m(x_i) = f_i$ con $i = 1, 2, ..., n$

Una spline si dice lineare se $m = 1$, quadratica se $m = 2$ mentre cubica se $m = 3$. Nel caso delle Spline cubiche $S_3(x)$, questa si definisce naturale fissata la seguente ulteriore condizione:

4. $S_3''(a = x_0) = S_3''(b = x_n) = 0$

16.1 Algoritmo: *s = naturalCubicSplinesSolver(xi, yi, xx)*

La Spline cubica naturale è stata implementata, insieme alle sue funzioni accessorie, nel seguente codice di Matlab,

```
function s = naturalCubicSplinesSolver(xi, yi, xx)
    %
    % s = naturalCubicSplinesSolver(xi, yi, xx)
    %
    % Author : Alessandro Montaghi
    % Email  : alessandro.montaghi@gmail.com
    % Date   : Spring 2019
    % Course : Calcolo Numerico 2018/2019
    %
    % Function   : Natural cubic spline data interpolation
    %
    % Description: returns the returns a vector of
    %              interpolated values s corresponding to
    %              the query points (xx). The values of s
    %              are determined by natural cubic spline
    %              interpolation of a given data (xi, yi).
    %
    % Parameters : xi - x-coordinates
    %              yi - Function values at x-coordinates
    %              xx - Query points
    %
    %
    len = length(xi);
    hi = xi(2 : len) - xi(1 : len-1); % I-th interval width
    [subDiagCoeff, superDiagCoeff] =...
        getCoefficientDiagTriangularMatrix(xi, hi);
    divdiff = (yi(2 : len) - yi(1 : len-1))./hi;
    divdiff = 6*((divdiff(2 : end) - divdiff(1 : end-1)) ./...
        (xi(3 : end)-xi(1 : end-2)));
    m = getFactorM(subDiagCoeff, superDiagCoeff, divdiff);
    [ri, qi] = getIntegrationConstants(m, yi, xi, hi);
    s = evaluateSpline(ri, qi, xx, xi, m, hi);
return

function [subDiagCoeff, superDiagCoeff] =
    ↪ getCoefficientDiagTriangularMatrix(xi, hi)
    % returns the subdiagonal matrix and the superdiagonal matrix
    subDiagCoeff = (hi(1 : end-1))./...
        (hi(1 : end-1) + hi(2 : end));
```

```
39      superDiagCoeff = (hi(2 : end))./...
40          (hi(1 : end-1) + hi(2 : end));
41      return
42
43  function m = getFactorM(subDiagCoeff, superDiagCoeff, divdiff)
44      % returns the factor m of the spline
45      n = length(superDiagCoeff) + 1;
46      u(1) = 2;
47      l = zeros(1, n-2);
48      for i = 2 : n-1
49          l(i) = subDiagCoeff(i)/u(i - 1);
50          u(i) = 2-l(i)*superDiagCoeff(i - 1);
51      end
52      y(1) = divdiff(1);
53      for i = 2:n-1
54          y(i) = divdiff(i) - l(i) * y(i - 1);
55      end
56      m = zeros(1, n - 1);
57      m(n-1) = y(n - 1) / u(n - 1);
58      for j = n-2 : -1 : 1
59          m(j) = ( y(j) - superDiagCoeff(j + 1) * m(j + 1) )/u(j);
60      end
61      m = [0 m 0];
62      return
63
64  function [ri, qi] = getIntegrationConstants(m, yi, xi, hi)
65      % returns the integration constants of the spline
66      n = length(xi);
67      ri = zeros(1, n-1);
68      qi = ri;
69      for i = 2 : n
70          ri(i - 1) = yi(i - 1) - ( hi(i - 1)^2)/6 * m(i - 1);
71          qi(i - 1) = ( yi(i) - yi(i - 1)) /...
72              hi(i - 1) - hi(i - 1) /...
73              6*(m(i) - m(i - 1));
74      end
75      return
76
77  function s = evaluateSpline(ri, qi, xx, xi, m, hi)
78      % returns the evaluetion of the spline
```

```
79      n = length(xx);
80      s = zeros(n,1)';
81      for j = 1 : n
82          i = getRange(xi, xx(j));
83          s(j) = ( ((xx(j) - xi(i - 1))^3) * m(i) +...
84                   ((xi(i) - xx(j))^3) * m(i - 1) ) /...
85                   ( 6*hi(i - 1) ) + qi(i - 1) *...
86                   ( xx(j) - xi(i - 1) ) + ri(i - 1);
87      end
88      return
89
90  function i = getRange(xi, xx)
91      % retuns the index of the interpolation abscissas
92      for i = 2 : length(xi)
93          if xx <= xi(i)
94              return;
95          end
96      end
97      return
```

Listing 16.1: Codice Matlab che implementa la Spline cubica naturale e le funzioni accessorie.

16.2 Esecuzione

La funzione *naturalCubicSplinesSolver* sviluppata è stata testata utilizzando la funzione $f(x) = sin(x)$,

```
xi = [0 1 2.5 3.6 5 7 8.1 10];
yi = sin(xi);
a = min(xi);
b = max(x);
xx = linspace(a, b, 10001);
hold on
grid on
s = naturalCubicSplinesSolver(xi, yi, xx);
plot(xi, yi,'o',xx,s);
hold off
```

Listing 16.2: Esecuzione di *naturalCubicSplinesSolver* utilizzando come esempio la funzione $f(x) = sin(x)$.

La figura seguente riporta il risultato ottenuto,

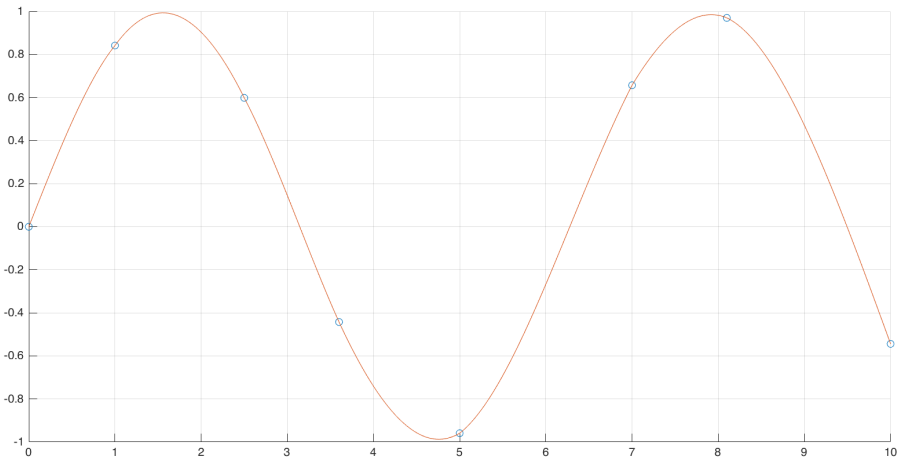


Figura 16.1: La figura riporta il risultato dell'interpolazione utilizzando il metodo della SPline cubica naturale implementato dalla funzione *naturalCubicSplinesSolver*.

17

Esercizio 17 (opzionale)

Descrizione: *Scrivere un programma che implementi il calcolo di una spline cubica not-a-knot interpolante su una partizione assegnata.*

Svolgimento:

18

Esercizio 18

Descrizione: *Confrontare i codici degli esercizi* $14 - 17$ *per approssimare la funzione* $f(x) = sin(x)$ *sulle ascisse* $x_i = i\pi/n, i = 0, 1, \cdots, n$, *per* $n = 1, 2, \cdots, 10$. *Graficare l'errore massimo di approssimazione verso n (in semilogy), calcolato su una griglia uniforme di 10001 punti nell'intervallo* $[0, \pi]$.

Svolgimento:

Il seguente codice di Matlab effettua il confronto tra i metodi di interpolazione di Newton, di Lagrange, di Hermite e la Spline Cubica Naturale.

```
format long e

f = @(x) sin(x);
f1 = @(x) cos(x);
a = 0; % left-endpoint of the closed interval.
b = pi; % right-endpoint of the closed interval.

xValutation = linspace(a, b, 10001);
yTrue = feval(f, xValutation);

NewtonResult = [];
LagrangeResult = [];
HermiteResult = [];
SplineResult = [];

```

```
for n = 2 : 10
    xi = (0:n) * pi/n;
    fi = feval(f, xi);
    ff1 = feval(f1, xi);
    % combine fi and ff1 for hermite
    len = 2 * length(xi)-1;
    fif1(1:2:len) = fi;
    fif1(2:2:len+1) = ff1;
    % Evaluate methods
    yNewton = newton(xi, fi, xValutation);
    yLagrange = lagrange(xi,fi, xValutation);
    yHermite = hermite(xi, fif1, xValutation);
    ySpline =  naturalCubicSplinesSolver(xi, fi, xValutation);
    % Computer Max Error
    maxErrorNewton = max(abs(yTrue - yNewton));
    maxErrorLagrange = max(abs(yTrue - yLagrange));
    maxErrorHermite = max(abs(yTrue - yHermite));
    maxErrorSpline = max(abs(yTrue - ySpline));
    % Print result
    fprintf('n: %d\n', n);
    fprintf('\t Newton Interpolation Max Error: %e\n', maxErrorNewton);
    fprintf('\t Lagrange Interpolation Max Error: %e\n',
        ↪ maxErrorLagrange);
    fprintf('\t Hermite Interpolation Max Error: %e\n', maxErrorHermite)
        ↪ ;
    fprintf('\t Spline Interpolation Max Error: %e\n', maxErrorSpline);
    % Save result
    NewtonResult = [NewtonResult maxErrorNewton];
    LagrangeResult = [LagrangeResult maxErrorLagrange];
    HermiteResult = [HermiteResult maxErrorNewton];
    SplineResult = [SplineResult maxErrorSpline];
end

x = (2:10);

semilogy(x, NewtonResult);
grid on
title('Netwon Interpolation max error')

semilogy(x, LagrangeResult);
```

```
54 grid on
55 title('Lagrange Interpolation max error')
56
57 semilogy(x, HermiteResult);
58 grid on
59 title('Hermite Interpolation max error')
60
61 semilogy(x, SplineResult);
62 grid on
63 title('Natural Cubic Spline Interpolation max error')
```

Listing 18.1: Codice Matlab per il confronto tra i metodi di interpolazione di Newton, Lagrange, Hermite e la Spline Cubica Naturale.

Di seguito si riportano i grafici ottenuti,

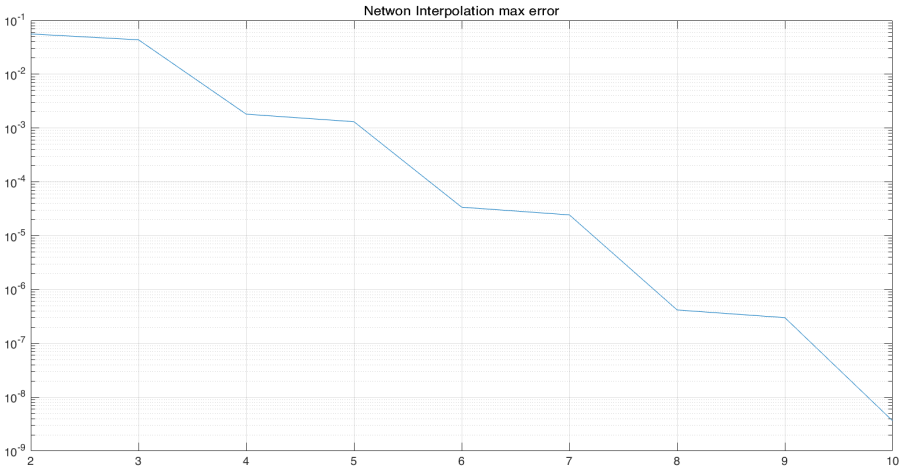


Figura 18.1: Grafico che riporta i valori della dell'errore massimo ottenuti valutando la funzione $f(x)$ utilizzando il Metodo di Newton.

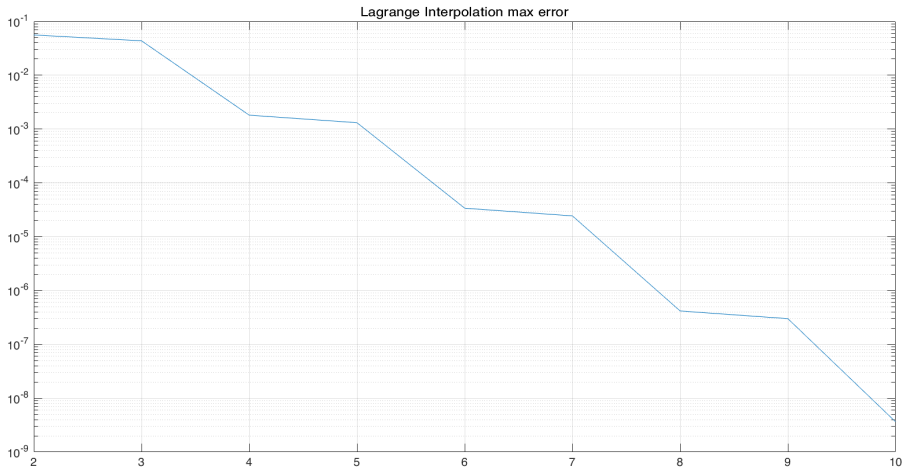


Figura 18.2: Grafico che riporta i valori della dell'errore massimo ottenuti valutando la funzione $f(x)$ utilizzando il Metodo di Lagrange.

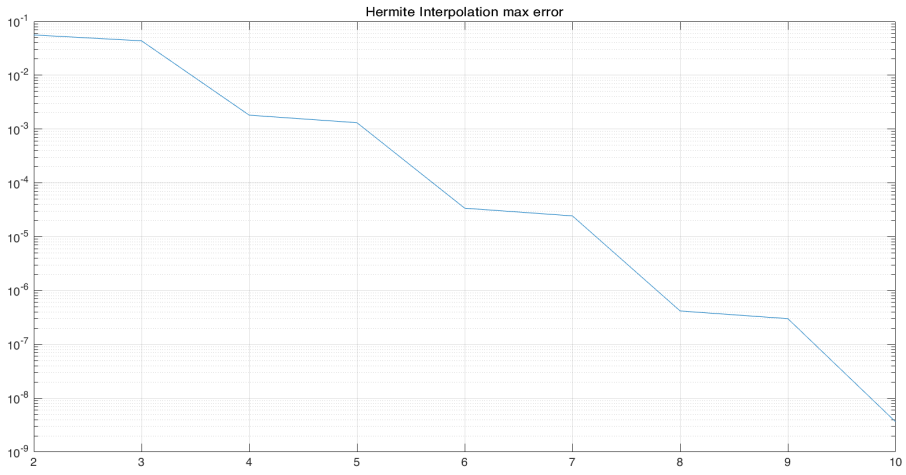


Figura 18.3: Grafico che riporta i valori della dell'errore massimo ottenuti valutando la funzione $f(x)$ utilizzando il Metodo di Hermite.

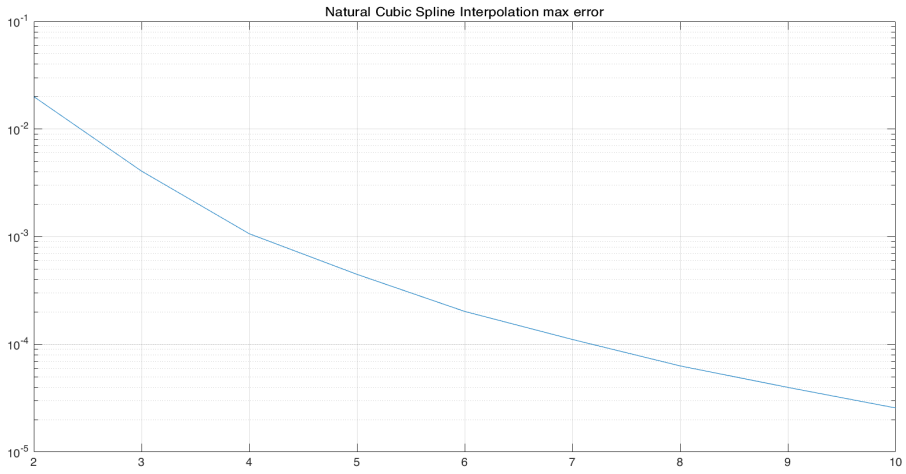


Figura 18.4: Grafico che riporta i valori della dell'errore massimo ottenuti valutando la funzione $f(x)$ utilizzando la Spline Cubica Naturale.

19

Esercizio 19

Descrizione: *Calcolare (numericamente) la costante di Lebesgue per i polinomi interpolanti di grado* $n = 2, 4, 8, \cdots, 40$*, sia sulle ascisse equidistanti che su quelle di Chebyshev (utilizzare 10001 punti equispaziati per valutare la funzione di Lebesgue). Graficare convenientemente i risultati ottenuti. Spiegare, quindi, i risultati ottenuti approssimando la funzione:*

$$f(x) = \frac{1}{1 + x^2} \quad x \in [-5, 5], \tag{19.1}$$

utilizzando le ascisse equidistanti e di Chebyshev precedentemente menzionate (tabulare il massimo errore valutato su una gliglia 10001 punti equidistanti nell'intervallo [-5, 5]).

Svolgimento:

Data una funziona $f : [a, b] \subset \mathbb{R} \to \mathbb{R}$ e le seguenti ascisse di interpolazioni distinte $a \leq x_0 < x_1 < \cdots < x_n \leq b$. Consideriamo il polinomio di interpolazione $p(x)$ di grado n della funzione $f(x)$ che soddisfa i vincoli di interpolazione $p(x_i) = f_i \equiv f(x_i)$ con $i = 0, 1, \cdots, n$ ed il polinomio di interpolazione $\tilde{p}(x)$, sempre di grado n, che interpola la funzione perturbata $\tilde{f}(x)$ di $f(x)$. Definiamo il *"condizionamento del problema"* (per l'interpolazione) nel seguente modo:

$$\|p - \tilde{p}\| \equiv \Lambda_n \cdot \left\|f - \tilde{f}\right\| \tag{19.2}$$

Dove $\|p - \tilde{p}\|$ misura la perturbazione sui dati in uscita, $\left\|f - \tilde{f}\right\|$ misura la perturbazione dei dati in entrata e la *"costante di Lebesgue"* Λ_n misura il condizionamento del problema dell'interpolazione lineare. In altre parole, Λ_n è un coefficiente che misura una *"amplificazione"* dell'errore e dipende dalla scelta delle ascisse di interpolazione.

In generale, posto n il grado del polinomio interpolante, abbiamo che $\Lambda_n \geq logn$, ovvero il problema diviene progressivamente malcodizionato con crescita logaritmica. Comunque, scegliendo ascisse di interpolazioni equidistanti, si ottiene che $\Lambda_n \approx 2^n$ e quindi il problema diviene velocemente malcondizionato (essendo di tipo esponenziale) al crescere del grado n del polinomio interpolante. Al fine di ottenere una crescita moderata di Λ_n ($\Lambda_n \approx 2/\pi \cdot log(n)$) si utilizzano le *"ascisse di Chebyshev"*. Assumendo un intervallo $[a, b] = [-1, 1]$, le ascisse di Chebyshev sono espresse come:

$$x_{n-i} = cos\left(\frac{(2 \cdot i + 1)\pi}{2 \cdot n + 2}\right), \; con \; i = 0, 1, \cdots, n \tag{19.3}$$

Nel caso di un intervallo $[a, b] \neq [-1, 1]$, occorre applicare la seguente trasformazione lineare:

$$x_i = \frac{a+b}{2} + \frac{b-a}{2} \cdot cos\left(\frac{(2 \cdot i + 1)\pi}{2 \cdot n + 2}\right), \; con \; i = 0, 1, \cdots, n \tag{19.4}$$

19.1 Algoritmo: *x = chebyshev (a, b, n)*

La funzione *chebyshev* implementa il calcolo dei nodi di Chebyshev in un arbitrario intervallo $[a, b]$, con $a < b$, per un polinomio interpolatore di grado n,

```
function x = chebyshev (a, b, n)
    %
    % x = chebyshev (a, b, n)
    %
    % Author : Alessandro Montaghi
    % Email  : alessandro.montaghi@gmail.com
    % Date   : Spring 2019
    % Course : Calcolo Numerico 2018/2019
    %
    % Function   : Chebyshev nodes
    %
    % Description:  This function calculates the Chebyshev
    %               nodes (i.e., abscissas).
    %
    % Parameters : a - left-endpoint of the closed interval.
    %              b - right-endpoint of the closed interval.
    %              n - Polynomial degree.
    %
    % Return     : Chebyshev nodes
```

```
    %
    if a >= b || n ~= fix(n) || n < 0
       error('Inconsistent data');
    end
        x = (a+b)/2 + ((b-a)/2)*cos((2*(0:n)+1)*(pi/(2*n+2)));
    return
```

Listing 19.1: Codice Matlab per l'implementazione delle ascisse di Chebyshev

19.2 Algoritmo: *[xp, cLeb, pleb] = lebesgue(xnodes, xpoints)*

La funzione *lebesgue* implementa il calcolo della *"costante di Lebesgue"* Λ_n ricevendo in entrata le ascisse (o nodi) di tipo equidistante oppure le ascisse di Chebyshev. La funzione di Lebesgue è valutata utilizzando 10001 punti equispaziati.

```
function [xpoints, MaxfunLeb, pointMaxfunleb] = lebesgue(xnodes, xpoints
    ↪ )
    %
    % [xpoints, MaxfunLeb, pointMaxfunleb] = lebesgue(xnodes [, xpoints
        ↪ ])
    %
    % Author : Alessandro Montaghi
    % Email  : alessandro.montaghi@gmail.com
    % Date   : Spring 2019
    % Course : Calcolo Numerico 2018/2019
    %
    % Function    :  Lebesgue constant
    %
    % Description:  This function estimates the Lebesgue
    %               constant for a set of points and the
    %               node where the Lebesgue function
    %               assumes the maximum value.
    %
    % Parameters : xnodes - interpolating nodes.
    %              xpoints - evaluating points (optional).
    %
    % Return     : xpoints - evaluating points.
    %              MaxfunLeb - Lebesgue constant.
```

```
22      %              pointMaxfunleb - node where the Lebesgue
23      %                               function assumes the
24      %                               maximum value.
25      %
26      if length(xnodes) == 1
27          error('Incorrect number of nodes');
28      end
29      if nargin == 1
30         xpoints = linspace(xnodes(1), xnodes(end), 10001);
31      end
32      len_xnodes = length(xnodes);
33      len_xpoints = length(xpoints);
34      for i = 1 : len_xpoints
35          for n = 1 : len_xnodes
36              plag(i,n) = prod(xpoints(i) - xnodes(1 : n-1)) *...
37                          prod(xpoints(i) - xnodes(n+1 : len_xnodes))/...
38                          (prod(xnodes(n) - xnodes(1 : n-1)) *...
39                          prod(xnodes(n) - xnodes(n+1 : len_xnodes)));
40          end
41      end
42      fun_leb = sum(abs(plag'));
43      [MaxfunLeb, p_Maxfunleb] = max(fun_leb);
44      pointMaxfunleb = xpoints(p_Maxfunleb);
45  return
```

Listing 19.2: Codice Matlab per il calcolo della costante di Lebesgue

19.3 Esecuzione

Il seguente codice di Matlab calcola (numericamente) la costante di Lebesgue per i polinomi interpolanti di grado $n = 2, 4, 6, \cdots, 40$ sia sulle ascisse equidistanti che su quelle di Chebyshev,

```
1  a = -5; % left-endpoint of the closed interval.
2  b = 5; % right-endpoint of the closed interval.
3  % initialize your vector(S)
4  equiLeb = [];
5  chebyLeb = [];
6  fprintf('Interval [%i, %i]\n', a, b);
7  for n = 2:2:40
```

```
    fprintf('Polynomial degree: %i\n', n);
    equiXnodes = linspace(a, b, n+1);
    g = sprintf('%d ', equiXnodes);
    fprintf('Equidistant nodes: %s\n', g);
    [equiXpoints, equiCostLeb, equiPointCostleb] = lebesgue(equiXnodes);
    fprintf('Lebesgue constant (equidistant nodes): %f\n', equiCostLeb);
    equiLeb = [equiLeb, equiCostLeb];

    chebyXnodes = chebyshev(a, b, n);
    g = sprintf('%d ', chebyXnodes);
    fprintf('Chebyshev nodes: %s\n', g);
    [chebyXpoints, chebyCostLeb, chebyPointCostleb] = lebesgue(
        ↪ chebyXnodes);
    fprintf('Lebesgue constant (Chebyshev nodes): %f\n', chebyCostLeb);
    chebyLeb =[chebyLeb, chebyCostLeb];
end

x = 2:2:40;

hold on
xlabel('Polynomial degree (n)')
ylabel('Lebesgue constant (equidistant nodes)')
plot(x, equiLeb,'-bs','MarkerSize',10)
grid on
hold off


hold on
xlabel('Polynomial degree (n)')
ylabel('Lebesgue constant (Chebyshev nodes)')
plot(x, chebyLeb,'-r*','MarkerSize',10)
grid on
hold off
```

Listing 19.3: Codice Matlab per il calcolo della costante di Lebesgue per i polinomi interpolanti di grado $n = 2, 4, 6, \cdots, 40$ sia sulle ascisse equidistanti che su quelle di Chebyshev.

Nei seguenti grafici si riportando i risultati ottenuti.

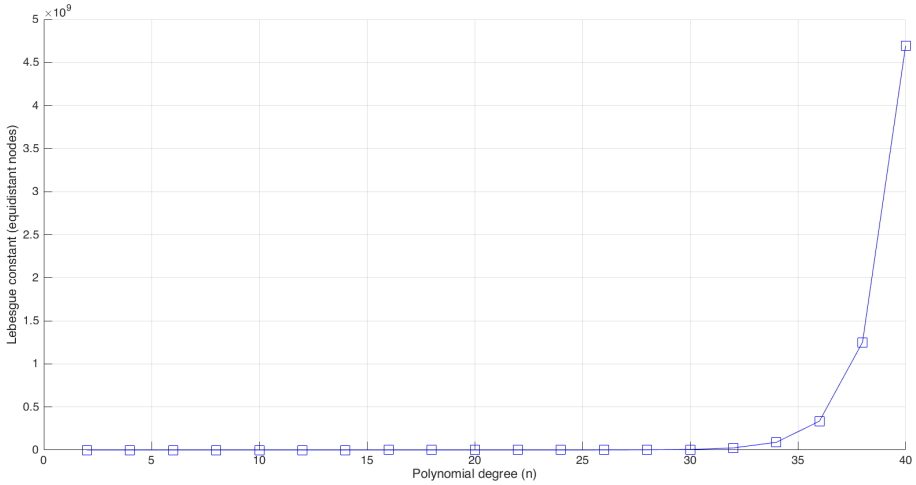


Figura 19.1: Grafico che riporta i valori della costante di Lebesgue ottenuti tramite la funzione *lebesgue* nel caso di ascisse equidistanti i polinomi interpolanti di grado $n = 2, 4, 6, \cdots, 40$.

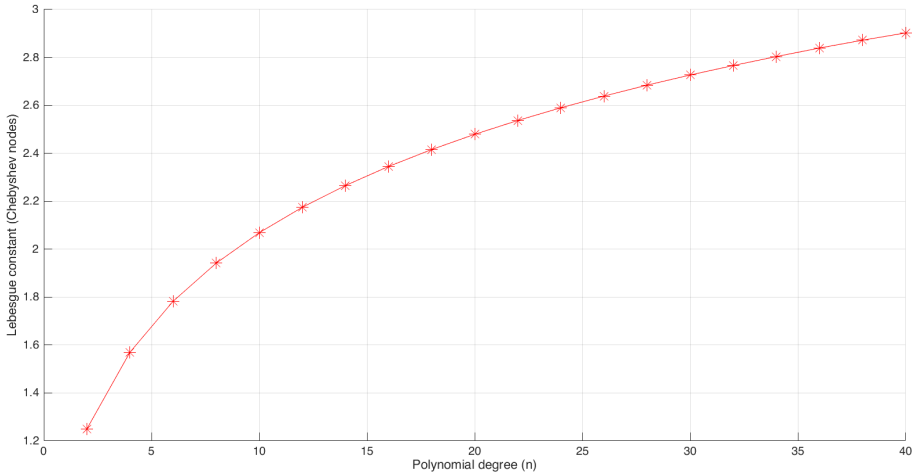


Figura 19.2: Grafico che riporta i valori della costante di Lebesgue ottenuti tramite la funzione *lebesgue* nel caso di ascisse di Chebyshev per i polinomi interpolanti di grado $n = 2, 4, 6, \cdots, 40$.

19.4 Esecuzione

Il seguente codice di Matlab calcola il *massimo errore* (valutato su una griglia 10001 punti equidistanti nell'intervallo $[-5, 5]$) approssimando la funzione $f(x)$ utilizzando le ascisse equidistanti e di Chebyshev precedentemente menzionate.

$$f(x) = \frac{1}{1+x^2} \quad x \in [-5, 5], \tag{19.5}$$

La seguente figura mostra la funzione $f(x)$ nell'intervallo $[-5, 5]$,

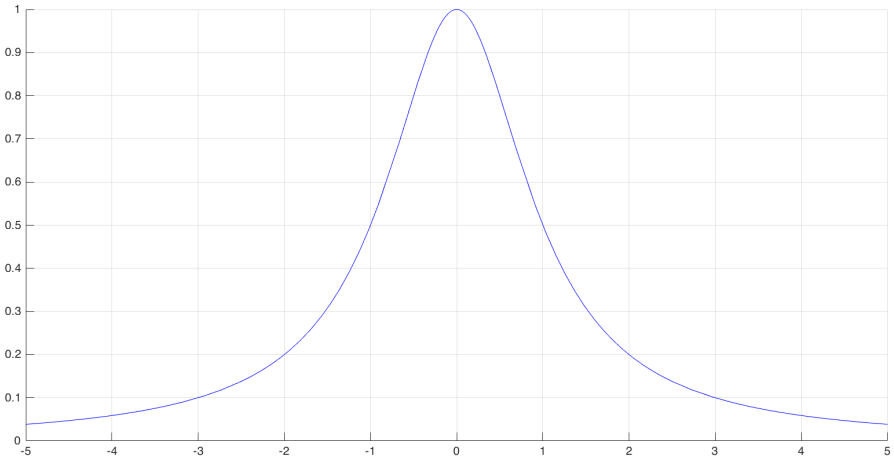


Figura 19.3: Andamento della funzione $f(x)$ nell'intervallo $[-5, 5]$.

Il codice utilizzato nella valutazione è il seguente,

```
1
2  fun = @(x) 1./(1+x.^2);
3  a = -5; % left-endpoint of the closed interval.
4  b = 5; % right-endpoint of the closed interval.
5
6  % valutation points
7  xValutation = linspace(a, b, 10001);
8
9  hold on
10 grid on
11 fplot(fun,[a b],'b');
```

```
12 hold off
13
14
15 % initialize vector(S)
16 equiMaxErrorLagrange = [];
17 equiMaxErrorNewton = [];
18 chebyMaxErrorLagrange = [];
19 chebyMaxErrorNewton = [];
20
21 fprintf('Interval [%i, %i]\n', a, b);
22 for n = 2:2:40
23     fprintf('Polynomial degree: %i\n', n);
24     % equidistant nodes
25     equiXnodes = linspace(a, b, n+1);
26     equiFi = feval(fun, equiXnodes);
27     % Chebyshev nodes
28     chebyXnodes = chebyshev(a, b, n);
29     chebyFi = feval(fun, chebyXnodes);
30
31     % true values
32     trueY = feval(fun, xValutation);
33
34     % Lagrange equidistant
35     equiYLagrange = lagrange(equiXnodes, equiFi, xValutation);
36     equiErrorLagrange = abs(equiYLagrange - trueY);
37     maxEquiErrorLagrange = max(equiErrorLagrange);
38     % Newton equidistant
39     equiYNewton = newton(equiXnodes, equiFi, xValutation);
40     equiErrorNewton = abs(equiYNewton - trueY);
41     maxEquiErrorNewton = max(equiErrorNewton);
42     % Lagrange Chebyshev
43     chebyYLagrange = lagrange(chebyXnodes, chebyFi, xValutation);
44     chebyErrorLagrange = abs(chebyYLagrange - trueY);
45     maxChebyErrorLagrange = max(chebyErrorLagrange);
46     % Newton Chebyshev
47     chebyYNewton = newton(chebyXnodes, chebyFi, xValutation);
48     chebyErrorNewton = abs(chebyYNewton - trueY);
49     maxChebyErrorNewton = max(chebyErrorNewton);
50     % print results
51     fprintf('Maxim error (equidistant nodes) with Lagrange: %f\n',
```

```
        ↪ maxEquiErrorLagrange);
    fprintf('Maxim error (equidistant nodes) with Newton: %f\n',
        ↪ maxEquiErrorNewton);
    fprintf('Maxim error (Chebyshev nodes) with Lagrange: %f\n',
        ↪ maxChebyErrorLagrange);
    fprintf('Maxim error (Chebyshev nodes) with Newton: %f\n',
        ↪ maxChebyErrorNewton);
    % store results
    equiMaxErrorLagrange = [equiMaxErrorLagrange maxEquiErrorLagrange];
    equiMaxErrorNewton = [equiMaxErrorNewton maxEquiErrorNewton];
    chebyMaxErrorLagrange = [chebyMaxErrorLagrange maxChebyErrorLagrange
        ↪ ];
    chebyMaxErrorNewton = [chebyMaxErrorNewton maxChebyErrorNewton];
end

x = 2:2:40;

% plot Maximum Error Lagrange with equidistant nodes
hold on
grid on
xlabel('Polynomial degree (n)')
ylabel('Maxim error (equidistant nodes) with Lagrange')
plot(x, equiMaxErrorLagrange,'-r*','MarkerSize',10)
title('Lagrange with equidistant nodes')
hold off


% plot Maximum Error Newton with equidistant nodes
hold on
grid on
xlabel('Polynomial degree (n)')
ylabel('Maxim error (equidistant nodes) with Newton')
plot(x, equiMaxErrorNewton,'-rS','MarkerSize',10)
title('Newton with equidistant nodes')
hold off


% plot Maximum Error Lagrange with Chebyshev nodes
hold on
grid on
```

```
xlabel('Polynomial degree (n)')
ylabel('Maxim error (Chebyshev nodes) with Lagrange')
plot(x, chebyMaxErrorLagrange,'-b*','MarkerSize',10)
title('Lagrange with Chebyshev nodes')
hold off


% plot Maximum Error Newton with Chebyshev nodes
hold on
grid on
xlabel('Polynomial degree (n)')
ylabel('Maxim error (Chebyshev nodes) with Newton')
plot(x, chebyMaxErrorLagrange,'-bS','MarkerSize',10)
title('Newton with Chebyshev nodes')
hold off


yZero = zeros(1, length(chebyXnodes));
hold on
grid on
xlabel('x-axis')
ylabel('y-axis')
fplot(fun,[a b],'b');
plot(chebyXnodes, yZero, 'rx', 'MarkerSize',14)
title('Chebyshev nodes Polynomial degree 40')
hold off
```

Listing 19.4: Codice Matlab per il calcolo dell'errore massimo sulla funzione $f(x)$ utilizzando le ascisse equidistanti che su quelle di Chebyshev.

La seguente figura mostra la funzione $f(x)$ nell'intervallo $[-5, 5]$,

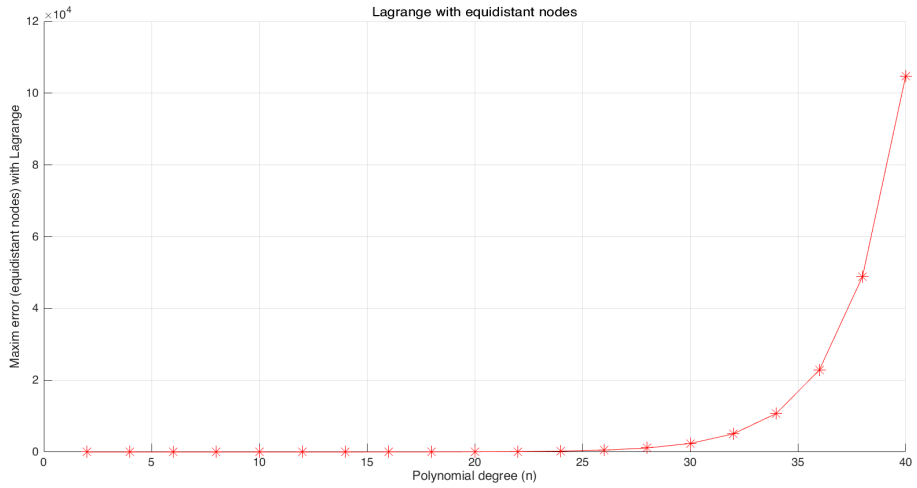


Figura 19.4: Grafico che riporta i valori della dell'errore massimo ottenuti valutando la funzione $f(x)$ nel caso di ascisse equidistanti utilizzando i polinomi interpolanti di Lagrange di grado $n = 2, 4, 6, \cdots, 40$.

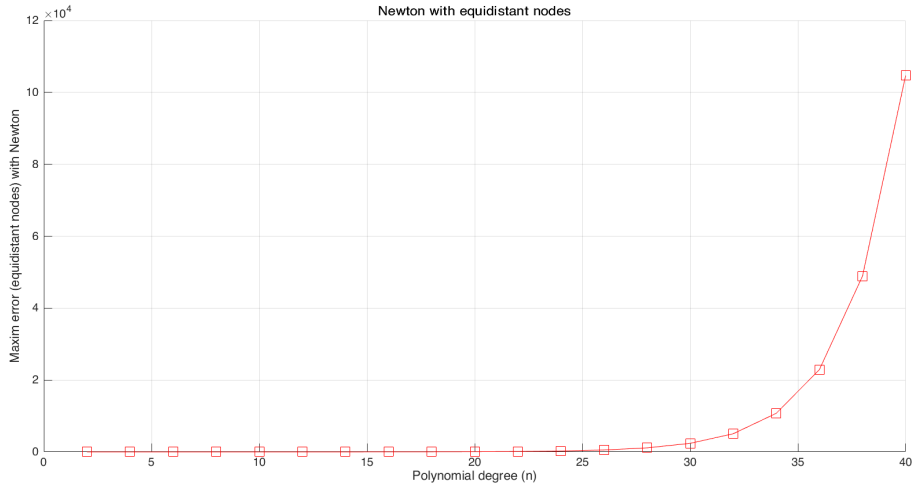


Figura 19.5: Grafico che riporta i valori della dell'errore massimo ottenuti valutando la funzione $f(x)$ nel caso di ascisse equidistanti utilizzando i polinomi interpolanti di Newton di grado $n = 2, 4, 6, \cdots, 40$.

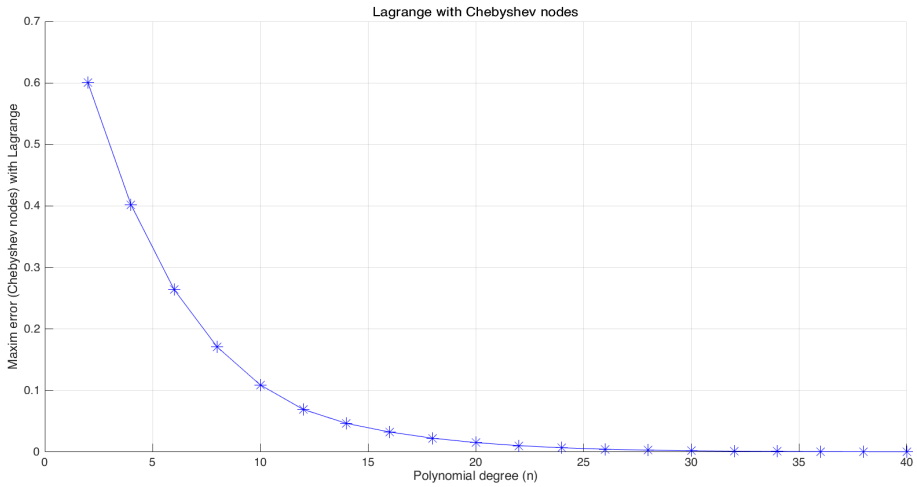


Figura 19.6: Grafico che riporta i valori della dell'errore massimo ottenuti valutando la funzione $f(x)$ nel caso di ascisse di Chebyshev utilizzando i polinomi interpolanti di Lagrange di grado $n = 2, 4, 6, \cdots, 40$.

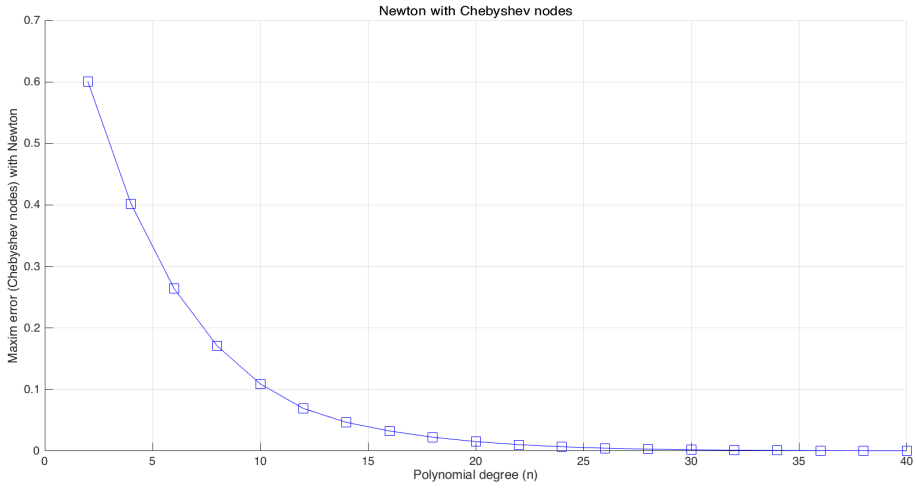


Figura 19.7: Grafico che riporta i valori della dell'errore massimo ottenuti valutando la funzione $f(x)$ nel caso di ascisse di Chebyshev utilizzando i polinomi interpolanti di Newton di grado $n = 2, 4, 6, \cdots, 40$.

Per concludere, risulta evidente come al crescere del grado, utilizzando le ascisse di Chebyshev (che si addensano vicino agli estremi dell'intervallo) l'approssimazione divenga

sempre più accurata, rispetto ai nodi equispaziati. Possiamo dedurre che l'interpolazione su nodi di Chebyshev è molto meno sensibile alla propagazione degli errori di arrotondamento di quanto non lo sia l'interpolazione su nodi equispaziati.

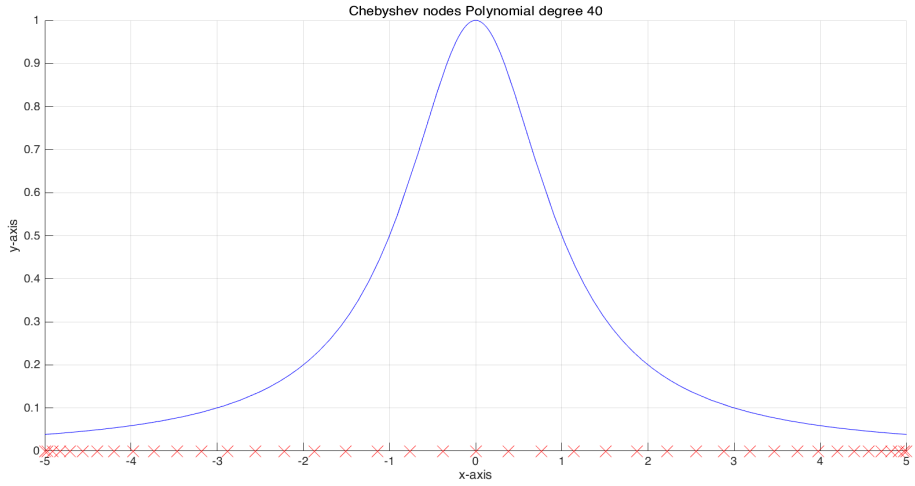


Figura 19.8: Distribuzione dei nodi di Chebyshev (in rosso) le quali si addensano vicino agli estremi dell'intervallo.

20

Esercizio 20

Descrizione: *Con riferimento al precedente esercizio, tabulare il massimo errore di approssimazione (calcolato come sopra indicato), sia utilizzando le ascisse equidistanti che quelle di Chebyshev su menzionate, relativo alla spline cubica naturale interpolante $f(x)$ su tali ascisse.*

Svolgimento:

20.1 Esecuzione

Il seguente codice di Matlab calcola il *massimo errore* (valutato su una griglia 10001 punti equidistanti nell'intervallo $[-5, 5]$) interpolando la funzione $f(x)$ tramite la spline cubica naturale interpolante, il cui algoritmo è stato implementato nell'Esercizio 16. Sono state valutate le ascisse equidistanti che su quelle di Chebyshev.

$$f(x) = \frac{1}{1+x^2} \quad x \in [-5, 5], \tag{20.1}$$

```
fun = @(x) 1./(1+x.^2);
a = -5; % left-endpoint of the closed interval.
b = 5; % right-endpoint of the closed interval.

% valutation points
xValutation = linspace(a, b, 10001);

```

```
% initialize vector(S)
equiMaxErrorSpline = [];
chebyMaxErrorSpline = [];

fprintf('Interval [%i, %i]\n', a, b);
for n = 2:2:40
    fprintf('Polynomial degree: %i\n', n);
    % equidistant nodes
    equiXnodes = linspace(a, b, n+1);
    equiFi = feval(fun, equiXnodes);
    % Chebyshev nodes
    chebyXnodes = chebyshev(a, b, n);
    chebyFi = feval(fun, chebyXnodes);

    % true values
    trueY = feval(fun, xValutation);

    % Spline equidistant
    equiYSpline = naturalCubicSplinesSolver(equiXnodes, equiFi,
        ↪ xValutation);
    equiError = abs(equiYSpline - trueY);
    maxEquiError = max(equiError);
    % Spline Chebyshev
    chebyYSpline = naturalCubicSplinesSolver(chebyXnodes, chebyFi,
        ↪ xValutation);
    chebyError = abs(chebyYSpline - trueY);
    maxChebyError = max(chebyError);
    % print results
    fprintf('Maxim error (equidistant nodes) with Spline: %f\n',
        ↪ maxEquiError);
    fprintf('Maxim error (Chebyshev nodes) with Lagrange: %f\n',
        ↪ maxChebyError);
    % store results
    equiMaxErrorSpline = [equiMaxErrorSpline maxEquiError];
    chebyMaxErrorSpline = [chebyMaxErrorSpline maxChebyError];
end

x = 2:2:40;

% plot Maximum Error Spline with equidistant nodes
```

```
hold on
grid on
xlabel('Polynomial degree (n)')
ylabel('Maxim error (equidistant nodes) with Natural Cubic Spline')
plot(x, equiMaxErrorSpline,'-r*','MarkerSize',10)
title('Natural Cubic Spline with equidistant nodes')
hold off


% plot Maximum Error Spline with Chebyshev nodes
hold on
grid on
xlabel('Polynomial degree (n)')
ylabel('Maxim error (Chebyshev nodes) with Natural Cubic Spline')
plot(x, chebyMaxErrorSpline,'-b*','MarkerSize',10)
title('Natural Cubic Spline with Chebyshev nodes')
hold off
```

Listing 20.1: Codice Matlab per il calcolo dell'errore massimo sulla funzione $f(x)$ utilizzando le ascisse equidistanti che su quelle di Chebyshev utilizzando la Spline Cubica Naturale.

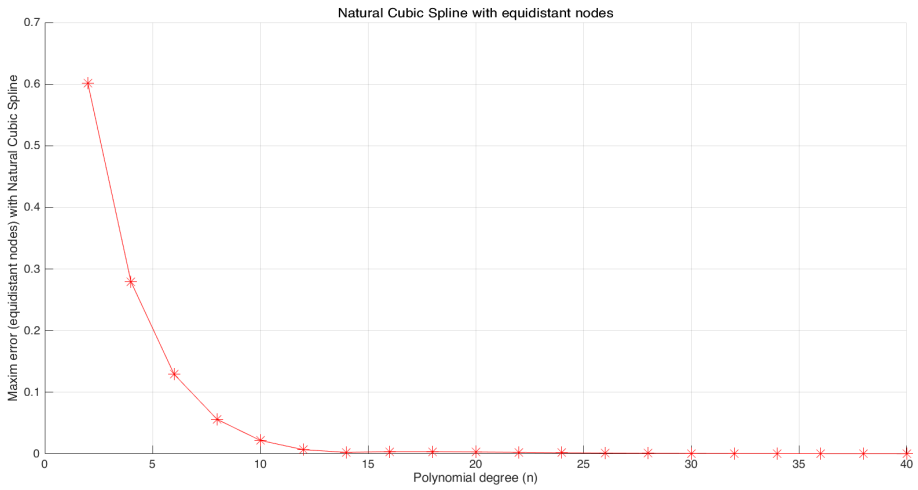


Figura 20.1: Grafico che riporta i valori della dell'errore massimo ottenuti valutando la funzione $f(x)$ nel caso di ascisse equidistanti utilizzando la Spline Cubica Naturale.

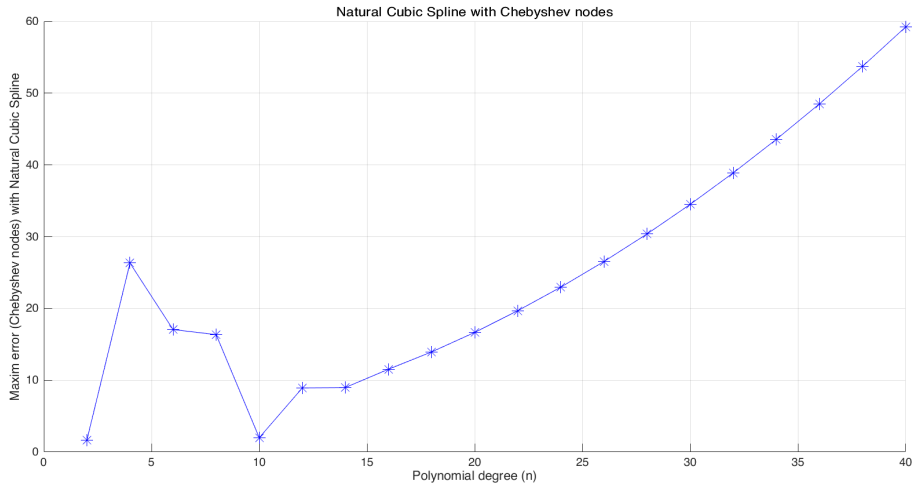


Figura 20.2: Grafico che riporta i valori della dell'errore massimo ottenuti valutando la funzione $f(x)$ nel caso di ascisse di Chebyshev utilizzando la Spline Cubica Naturale.

21

Esercizio 21

Descrizione: *Uno strumento di misura ha una accuratezza di* 10^{-6} *(in opportune unità di misura). I dati misurati nelle posizioni* x_i *sono dati da* y_i*, come descritto in tabella. Calcolare il grado minimo, ed i relativi coefficienti, del polinomio che meglio approssima i precedenti dati nel senso dei minimi quadrati con una adeguata accuratezza. Graficare convenientemente i risultati ottenuti.*

Svolgimento:

Consideriamo le seguenti misure sperimentali ottenute da uno strumento che ha un accuratezza di 10^{-1} (in opportune unità di misura),

i	x_i	y_i
0	0.010	1.003626
1	0.098	1.025686
2	0.127	1.029512
3	0.278	1.029130
4	0.547	0.994781
5	0.632	0.990156
6	0.815	1.016687
7	0.906	1.057382
8	0.913	1.061462
9	0.958	1.091263
10	0.965	1.096476

La seguente figura riporta l'andamento delle misure sperimentali (x_i, y_i),

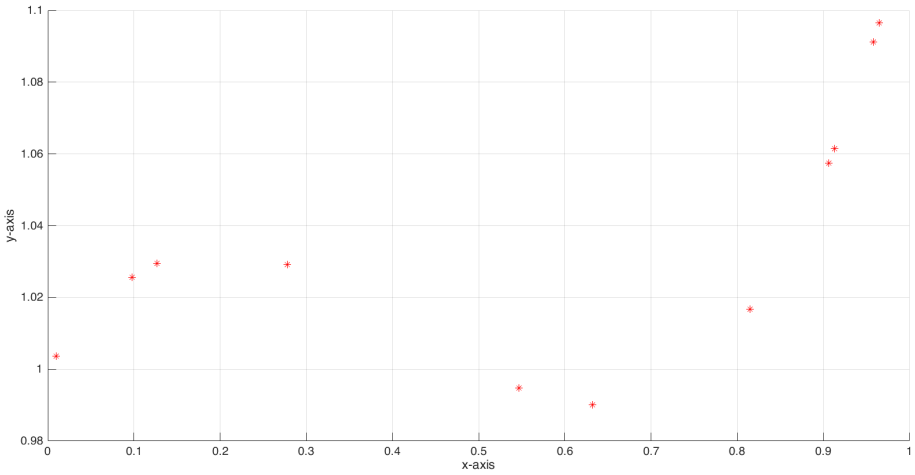


Figura 21.1: Grafico che riporta l'andamento delle coppie (x_i, y_i) delle misure sperimentali.

Più in generale, sapendo che la legge che descrive questo fenomeno è di tipo polinomiale, con un polinomio (incognito) di grado m,

$$y = \sum_{k=0}^{m} a_k \cdot x^k = a_0 \cdot x^0 + a_1 \cdot x^1 + \cdots + a_m \cdot x^m, \tag{21.1}$$

Quindi date le seguenti misure sperimentali (x_i, y_i), con $i = 0, 1, \cdots, n$ e $n \geq m$ (in genere, m sarà decisamente minore di n) si deve determinare il polinomio (21.1) che meglio approssima (ossia di migliore approssimazione) queste coppie di dati. Per definire il polinomio di migliore approssimazione dei dati, consideriamo il seguente sistema di equazioni lineari,

$$V \cdot \underline{c} = \underline{z}, \tag{21.2}$$

Dove $\underline{c} = (c_0, c_1, c_2, \cdots, c_m)^T$ è il vettore dei coefficienti, $\underline{z} = (z_0, z_1, z_2, \cdots, z_m)^T$ è il vettore dei valori previsti in corrispondenza delle ascisse $x_0, x_1, \cdots, x_n$ e la matrice $V \in \mathbb{R}^{(n+1)x(m+1)}$ è una *matrice di tipo Vandermonde*.

$$V = \begin{pmatrix} x_0^0 & x_0^1 & \cdots & x_0^m \\ x_1^0 & x_1^1 & \cdots & x_1^m \\ \vdots & \vdots & & \vdots \\ x_n^0 & x_n^1 & \cdots & x_n^m \end{pmatrix} \in \mathbb{R}^{(n+1)x(m+1)} \tag{21.3}$$

In generale, se il polinomio che vogliamo determinare ha grado m, supporremo che almeno $m + 1$ delle ascisse x_i di (x_i, y_i) siano tra loro distinte. Da questo deriva che

(teorema), la matrice V ha rango massimo (pari a $m+1$). Quindi, considerando il vettore $\underline{y} = (y_0, y_1, \cdots, y_n)^T$ dei valori misurati ed il vettore $\underline{z} = (z_0, z_1, \cdots, z_n)^T$ dei valori previsti, il polinomio di migliore approssimazione dei dati è quello che minimizza la quantità,

$$\left\| \underline{y} - \underline{z} \right\|_2^2 = \sum_{i=0}^{n} \left| y_i - z_i \right|^2, \tag{21.4}$$

21.1 Algoritmo: *[res, coeff] = minimiQuadratiResidue (x, y, n)*

Il calcolo del residuo e dei rispettivi coefficienti è stato implementato nella seguente funzione di matlab,

```
function [res, coeff] = minimiQuadratiResidue (x, y, n)
    %
    % [res, coeff] = minimiQuadratiResidue (x, y, n)
    %
    % Author : Alessandro Montaghi
    % Email  : alessandro.montaghi@gmail.com
    % Date   : Spring 2019
    % Course : Calcolo Numerico 2018/2019
    %
    % Function   : least squares (minimi quadraati) residue
    %              and coefficient  in a least-squares sense
    %
    % Description: this function returns for a polynomial
    %              p(x) of degree n the residual (res) and
    %              the coefficient (coeff) in a least-squares sense.
    %
    % Parameters : x - query points.
    %              y - fitted values at query points.
    %              n - Polynomial degree.
    %
    %
    V = fliplr(vander(x));
    A = V(1:end,1:n+1);
    QR = myqr(A);
    % 'a' is the vector of the coefficients
```

```
26      a = qrsolve(QR,y');
27      diff = (A*a)- y';
28      res = norm(diff,2)^2;
29      coeff = a;
30      return
```

Listing 21.1: Codice in Matlab della funzione *minimiQuadratiResidue* per il calcolo del residuo e dei rispettivi coefficienti utlizzando il metodo dei Minimi quadrati per un dato polinomio di grado n.

Per il calcolo del grado minimo, ed i relativi coefficienti, del polinomio che meglio approssima i precedenti dati nel senso dei minimi quadrati con una adeguata accuratezza, si è provveduto a realizzare il seguente script che utilizza la funzione *minimiQuadratiResidue*.

```
1  xi = [0.010 0.098 0.127 0.278 0.547 0.632 0.815 0.906 0.913 0.958
       ↪ 0.965];
2  yi = [1.003626 1.025686 1.029512 1.029130 0.994781 0.990156 1.016687
       ↪ 1.057382 1.061462 1.091263 1.096476];
3
4  residueList = zeros(length(1:10),2);
5
6  tol = 1E-6;
7  flag = 0;
8  for n = 1 : 10
9      [res, coeff] = minimiQuadratiResidue (xi, yi, n);
10     fprintf('Polynomial degree: %i - norma2: %e\n',n, res);
11     if res <= tol && flag == 0
12         fprintf('\t Mimimum Polynomial degree: %i - Residue: %e\n',n,
               ↪ res);
13         mimimumDegree = 3;
14         Mincoeff = coeff;
15         g = sprintf('%d ', Mincoeff);
16         fprintf('\t Coeff: %s\n', g)
17         flag = 1;
18     end
19     residueList(n,:)=[n, res];
20 end
21 >> Polynomial degree: 1 - Residue: 8.438988e-03
22 >> Polynomial degree: 2 - Residue: 3.656440e-03
23 >> Polynomial degree: 3 - Residue: 2.495722e-13
```

```
>>           Mimimum Polynomial degree: 3 - Residue: 2.495722e-13
>>           Coeff: 9.999999e-01 3.750012e-01 -1.250003e+00 1.000002e+00
>> Polynomial degree: 4 - Residue: 1.272099e-13
>> Polynomial degree: 5 - Residue: 1.207970e-13
>> Polynomial degree: 6 - Residue: 1.557540e-14
>> Polynomial degree: 7 - Residue: 1.495126e-14
>> Polynomial degree: 8 - Residue: 1.495091e-14
>> Polynomial degree: 9 - Residue: 6.757740e-15
>> Polynomial degree: 10 - Residue: 1.917918e-28
```

Listing 21.2: Codice in Matlab che utilizza la funzione *minimiQuadratiResidue* per determinare polinomio che meglio approssima coppie di dati (x_i, y_i), con $i = 0, 1, \cdots, n$, con una adeguata accuratezza.

La seguente tabella riporta il residuo per i vari gradi del polinomio,

Degree	*Residue*
1	8.438988e-03
2	3.656440e-03
3	2.495722e-13
4	1.272099e-13
5	1.207970e-13
6	1.557540e-14
7	1.495126e-14
8	1.495091e-14
9	6.757740e-15
10	1.917918e-28

Come si evidenzia dall'esecuzione del codice (21.2) e da quanto riportato in tabella il grado minimo del polinomio che meglio approssima le coppie di dato (x_i, y_i) con $i = 0, 1, \cdots, n$ nel senso dei minimi quadrati con una adeguata accuratezza corrisponde al grado 3. Di conseguenza, i coefficienti sono: 1.0000, 0.3750, -1.2500 e 1.0000.

$$P_3(x) = x^3 + 0.3750 \cdot x_2 - 1.2500 \cdot x + 1, \tag{21.5}$$

22

Esercizio 22

Descrizione: *Scrivere due functions che implementino efficientemente le formule adattattive dei trapezi e di Simpson.*

Svolgimento:

Consideriamo una funzione integranda $f(x)$ (abbastanza) regolare e continua nell'intervallo $[a, b]$ con $a < b$ ed il suo polinomo interpolante, nella forma di Lagrange, $P_n \in \Pi_n$ (dove Π_n rappresenta l'insieme dei polinomi di grado al più n) su n+1 ascisse equidistanti $a \leq x_0 < x_1 < \cdots < x_n \leq b$ dove $x_i = a + i \cdot h$ con $i = 0, 1, \cdots, n$ e $h = (b - a)/n$. La generica *"formula di quadratura di Newton-Cotes"* risulta essere:

$$I_n(f) \equiv \frac{b-a}{n} \cdot \sum_{i=0}^{n} C_{in} \cdot f_i, \tag{22.1}$$

in cui

$$C_{in} = \int_0^n \prod_{j=0, j\neq i}^{n} \frac{t-j}{i-j} dt, \ i = 0, 1, \cdots, n \tag{22.2}$$

Suddividendo l'intervallo $[a, b]$ in più sottointervalli di uguale ampiezza e utilizzando, in ciascuno di essi, la stessa formula di Newton-Cotes si ottengono le le cosiddette formule di Newton-Cotes composite. In particolare, possiamo ricordare la *formula dei Trapezi composita* e la *formula di Simposon composita*:

$$I_1^{(n)}(f) = \frac{b-a}{2n} \cdot \left(f_0 + 2 \cdot \sum_{i=1}^{n-1} f_i + f_n \right), \tag{22.3}$$

$$I_2^{(n)}(f) = \frac{b-a}{3n} \cdot \left(4 \cdot \sum_{i=1}^{n/2} f_{2i-1} + 2 \cdot \sum_{i=0}^{n/2} f_{2i} - f_0 - f_n \right), \tag{22.4}$$

Nel caso in in cui la funzione integranda f(x) abbia delle rapide variazioni in piccole porzioni dell'intervallo $[a, b]$, mentre risulta essere lentamente variabile al di fuori di queste. In questa evenienza, le formule composite di Newton-Cotes si prestano agevolmente ad essere implementate come formule adattative, ovvero consentendo ai nodi della partizione dell'intervallo [a, b], $a = x_0 < x_1 < \cdots < x_n = b$ di essere scelti in modo opportuno, in base al comportamento locale della funzione $f(x)$.

22.1 Algoritmo: *I2 = trapezi(fun, a, b, tol)*

La formula adattativa dei trapezi è implementata tramite il seguente codice in Matlab:

```
function I2 = adapTrapezi(fun, a, b, tol, fa, fb)
    %
    % I2 = trapezi(fun, a, b [, tol])
    %
    % Author : Alessandro Montaghi
    % Email  : alessandro.montaghi@gmail.com
    % Date   : Spring 2019
    % Course : Calcolo Numerico 2018/2019
    %
    % Function   : adaptative Trapezi formula
    %
    % Description: recursive function that computes the integral
    %              of the fuction f(x) in the closed interval [a, b],
    %              with a < b, using the "trapezi adattativa"
    %              formula and tollerance tol (default: 1E-5).
    %
    % Parameters : fun - continuous function f(x)
    %              a - point a of interval [a,b]
    %              b - point b of interval [a,b]
    %              tol - tollerance
    %              fa - y-value of point a
    %              fb - y-value of point b
    %
    %
    if nargin <= 4
```

```
26          fa = feval(fun, a);
27          fb = feval(fun, b);
28          if nargin == 3
29              tol = 1E-5;
30          end
31      end
32      x1 = (a+b)/2;
33      f1 = feval(fun, x1);
34      h = (b-a)/2;
35      I1 = h * (fa + fb);
36      I2 = 0.5 * h * (fa + 2*f1 + fb);
37      err = abs(I2 - I1) / 3;
38      if err > tol
39          I2 = adapTrapezi(fun, a, x1, tol/2, fa, f1) + ...
40              + adapTrapezi(fun, x1, b, tol/2, f1, fb);
41      end
42      return
```

Listing 22.1: Codice Matlab che implementa la formula adattativa dei trapezi in modo ricorsivo.

22.2 Esecuzione

Il seguente codice riporta un esecuzione della funzione *adapTrapezi* per il calcolo del seguente integrale definito.

$$I(f) = \int_a^b f(x)dx = \int_{\pi/5}^{\frac{3\pi}{10}} sin(5 \cdot x - \pi)dx = \frac{1}{5} \tag{22.5}$$

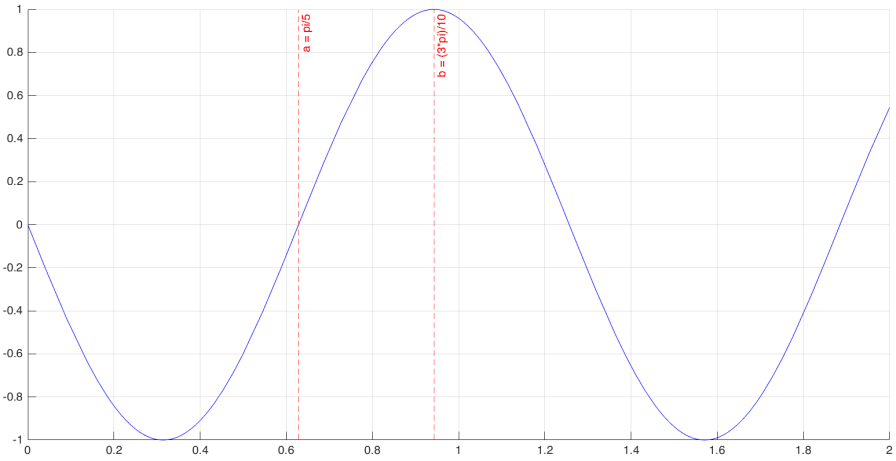


Figura 22.1: La funzione $f(x) = sin(5 \cdot x - \pi)$ è stata rappresentata usando l'intervallo $[0, 2]$. In rosso sono evidenziati i punti dove è stato calcolato l'integrale definito.

```
fun = @(x) sin(5*x - pi);
a = pi/5;
b = (3*pi)/10;

hold on
fplot(fun,[0,2],'b')
xline(a,'--r','a = (3*pi)/10')
xline(b,'--r','b = pi/5')
hold off
grid on

I2 = adapTrapezi(fun, a, b);
```

```
disp('Risultato della funzione adapTrapezi:');
disp(I2);

>> Risultato della funzione adapTrapezi:
>> 0.2000
```

Listing 22.2: Esempio di uso della funzione adapTrapezi.

22.3 Algoritmo: *I2 = adapSimpson(fun, a, b, tol)*

```
function I2 = adapSimpson(fun, a, b, tol, fa, fb, f1)
    %
    % I2 = adapSimpson(fun, a, b [, tol])
    %
    % Author : Alessandro Montaghi
    % Email  : alessandro.montaghi@gmail.com
    % Date   : Spring 2019
    % Course : Calcolo Numerico 2018/2019
    %
    % Function   : adaptative Simpson formula
    %
    % Description: recursive function that computes the integral
    %              of the fuction f(x) in the closed interval [a, b],
    %              with a < b, using the "Simpson adattativa"
    %              formula and tollerance tol (default: 1E-5).
    %
    % Parameters : fun - continuous function f(x)
    %              a - point a of interval [a,b]
    %              b - point b of interval [a,b]
    %              tol - tollerance
    %              fa - y-value of point a
    %              fb - y-value of point b
    %
    %
    x1 = (a+b)/2;
    if nargin <= 4
        fa = feval(fun, a);
        fb = feval(fun, b);
        f1 = feval(fun, x1);
         if nargin == 3
```

```
31            tol = 10E-5;
32        end
33    end
34    h = (b - a)/6;
35    x2 = ( a + x1 )/2;
36    x3 = ( x1 + b )/2;
37    f2 = feval(fun, x2);
38    f3 = feval(fun, x3);
39    I1 = h * (fa + 4* f1 + fb);
40    I2 = 0.5 * h * (fa + 4*f2 + 2*f1 + 4*f3 + fb);
41    err = abs(I2 - I1)/15;
42    if err > tol
43        I2 = adapSimpson(fun, a, x1, tol/2, fa, f1, f2) + ...
44            + adapSimpson(fun, x1, b, tol/2, f1, fb, f3);
45    end
46    return
```

Listing 22.3: Codice Matlab che implementa la formula adattativa di Simpson in modo ricorsivo.

22.4 Esecuzione

Il seguente codice riporta un esecuzione della funzione *adapSimpson* per il calcolo del seguente integrale definito.

$$I(f) = \int_a^b f(x)dx = \int_{\pi/5}^{\frac{3\pi}{10}} sin(5 \cdot x - \pi)dx = \frac{1}{5} \tag{22.6}$$

```
fun = @(x) sin(5*x - pi);
a = pi/5;
b = (3*pi)/10;

hold on
fplot(fun,[0,2],'b')
xline(a,'--r','a = (3*pi)/10')
xline(b,'--r','b = pi/5')
hold off
grid on

I2 = adapSimpson(fun, a, b);
disp('Risultato della funzione adapTrapezi:');
disp(I2);

>> Risultato della funzione adapSimpson:
>> 0.2000
```

Listing 22.4: Esempio di uso della funzione adapSimpson.

23

Esercizio 23

Descrizione: *Sapendo che*

$$I(x) = \int_0^{atan(30)} (1 + tan^2(x))dx = 30 \tag{23.1}$$

tabulare il numero dei punti richiesti dalle formule composite dei trapezi e di Simpson per approssimare $I(f)$ *con tolleranze* $tol = 10^{-i}$, *con* $i = 2, \cdots, 8$

Svolgimento:

Nell'esercizio proposto, sono stati applicati il metodo dei trapezi di tipo adattativo ed il metodo di Simpson di tipo adattativo per il calcolo dell'integrale definito nell'intervallo $[0, atan(30)]$ del seguente integrale

$$I(x) = \int_0^{atan(30)} (1 + tan^2(x))dx = 30 \tag{23.2}$$

```
1 fun =  @(x) (1 + (tan(x)).^2);
2 a = 0; b = atan(30);
3 hold on
4 grid on
5 fplot(fun,[-3,4],'b')
6 xline(a,'--r','a = 0')
7 xline(b,'--r','b = atan(30)')
8 hold off
```

Listing 23.1: Codice Matlab che visualizza la funzione f(x) data.

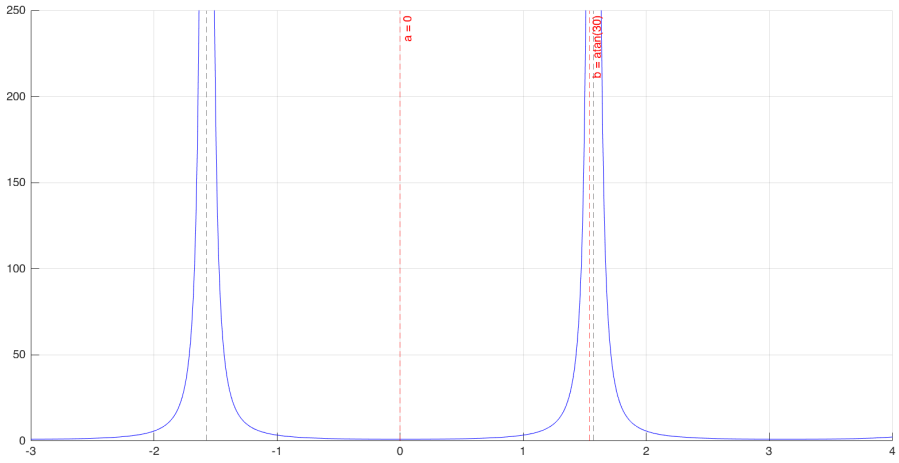


Figura 23.1: La funzione $f(x) = 1 + tan^2(x)$ è stata rappresentata usando l'intervallo $[-3, 4]$. In rosso sono evidenziati i punti dove è stato calcolato l'integrale definito.

23.1 Algoritmo: *[I2, np] = adapTrapeziCounter(fun, a, b, tol)*

Nello svolgimento di questo esercizio si è modificate la funzione *adapTrapezi* al fine di aggiungere un contatore di tipo *persistent* al fine di tabulare i numeri di punti richiesti.

```
1  function [I2, numberPoints] = adapTrapeziCounter(fun, a, b, tol, fa, fb)
2      %
3      % I = trapezi(fun, a, b [, tol])
4      %
5      % Author : Alessandro Montaghi
6      % Email  : alessandro.montaghi@gmail.com
7      % Date   : Spring 2019
8      % Course : Calcolo Numerico 2018/2019
9      %
10     % Function   : adaptative Trapezi formula with counter
11     %
12     % Description: recursive function that computes the integral
13     %              of the fuction f(x) in the closed interval [a, b],
14     %              with a < b, using the "trapezi adattativa"
15     %              formula and tollerance tol (default: 1E-5).
```

```
16      %
17      % Parameters : fun - continuous function f(x)
18      %               a - point a of interval [a,b]
19      %               b - point b of interval [a,b]
20      %               tol - tollerance
21      %               fa - y-value of point a
22      %               fb - y-value of point b
23      %
24      %
25      persistent counter
26      if nargin <= 4
27          fa = feval(fun, a);
28          fb = feval(fun, b);
29          counter = 1;
30          if nargin == 3
31              tol = 1E-5;
32          end
33      else
34          counter = counter + 1;
35      end
36      x1 = (a+b)/2;
37      f1 = feval(fun, x1);
38      h = (b-a)/2;
39      I1 = h * (fa + fb);
40      I2 = 0.5 * h * (fa + 2*f1 + fb);
41      err = abs(I2 - I1) / 3;
42      if err > tol
43          I2 = adapTrapeziCounter(fun, a, x1, tol/2, fa, f1) + ...
44              + adapTrapeziCounter(fun, x1, b, tol/2, f1, fb);
45      end
46      numberPoints = counter + 2;
47      return
```

Listing 23.2: Codice Matlab che implementa la formula adattativa dei trapezi in modo ricorsivo con contatore per il numero dei punti richiesti.

23.2 Esecuzione

Di seguito si riporta il codice dell'esecuzione della funzione *adapTrapeziCounter*

```
fun =  @(x) 1 + (tan(x)).^2;
a = 0;
b = atan(30);
i = [2,3,4,5,6,7,8];
disp('Trapezi adattativa formula')
for j = 1:length(i)
    fprintf('Value tollerance: 10 to -%i\n', i(j));
    [I2, numberPoints] = adapTrapeziCounter(fun, a, b, 10^-i(j));
    fprintf('I1(f): %f\n', I2);
    err = abs(30 - I2);
    fprintf('Error: %f\n', err);
    fprintf('number of points: %d\n', numberPoints);
end
```

Listing 23.3: Esecuzione della funzione adapTrapeziCounter.

I risultati dell'esecuzione sono riportati nella seguente tabella:

Tolleranza iniziale (*tol*)	$I_1^n(f)$	$E_1^n(f)$	**Numero di punti**
10^{-2}	30.004765	0.004765	375
10^{-3}	30.000574	0.000574	1181
10^{-4}	30.000053	0.000053	3687
10^{-5}	30.000005	0.000005	11883
10^{-6}	30.000001	0.000001	37273
10^{-7}	30.000000	0.000000	116747
10^{-8}	30.000000	0.000000	375793

23.3 Algoritmo: *[I2, np] = adapSimpsonCounter(fun, a, b, tol)*

Nello svolgimento di questo esercizio si è modificate la funzione *adapSimpson* al fine di aggiungere un contatore di tipo *persistent* al fine di tabulare i numeri di punti richiesti.

```
function [I2, numberPoints] = adapSimpsonCounter(fun, a, b, tol, fa, fb,
    ↪  f1)
    %
    % I2 = adapSimpsonCounter(fun, a, b [, tol])
    %
    % Author : Alessandro Montaghi
    % Email  : alessandro.montaghi@gmail.com
    % Date   : Spring 2019
    % Course : Calcolo Numerico 2018/2019
    %
    % Function   : adaptative Simpson formula with counter
    %
    % Description: recursive function that computes the integral
    %              of the fuction f(x) in the closed interval [a, b],
    %              with a < b, using the "Simpson adattativa"
    %              formula and tollerance tol (default: 1E-5).
    %
    % Parameters : fun - continuous function f(x)
    %              a - point a of interval [a,b]
    %              b - point b of interval [a,b]
    %              tol - tollerance
    %              fa - y-value of point a
    %              fb - y-value of point b
    %
    %
    persistent counter
    x1 = (a+b)/2;
    if nargin <= 4
        fa = feval(fun, a);
        fb = feval(fun, b);
        f1 = feval(fun, x1);
        counter = 3;
        if nargin == 3
            tol = 1E-5;
```

```
        end
    else
        counter = counter + 3;
    end
    h = (b - a)/6;
    x2 = ( a + x1 )/2;
    x3 = ( x1 + b )/2;
    f2 = feval(fun, x2);
    f3 = feval(fun, x3);
    I1 = h * (fa + 4* f1 + fb);
    I2 = 0.5 * h * (fa + 4*f2 + 2*f1 + 4*f3 + fb);
    err = abs(I2 - I1)/15;
    if err > tol
        I2 = adapSimpsonCounter(fun, a, x1, tol/2, fa, f1, f2) + ...
            + adapSimpsonCounter(fun, x1, b, tol/2, f1, fb, f3);
    end
    numberPoints = counter + 2;
    return
```

Listing 23.4: Codice Matlab che implementa la formula adattativa di Simpson in modo ricorsivo con contatore per il numero dei punti richiesti.

23.4 Esecuzione

Di seguito si riporta il codice dell'esecuzione della funzione *adapSimpsonCounter*

```
fun =  @(x) 1 + (tan(x)).^2;
a = 0; b = atan(30);
i = [2,3,4,5,6,7,8];
disp('Simpson adattativa formula')
for j = 1:length(i)
    fprintf('Value tollerance: 10 to -%i\n', i(j));
    [I2, numberPoints] = adapSimpsonCounter(fun, a, b, 10^-i(j));
    fprintf('I1(f): %f\n', I2);
    err = abs(30 - I2);
    fprintf('E1(f): %f\n', err);
    fprintf('number of points: %d\n', numberPoints);
end
```

Listing 23.5: Esecuzione della funzione adapSimpsonCounter.

I risultati dell'esecuzione sono riportati nella seguente tabella:

Tolleranza iniziale (*tol*)	$I_2^n(f)$	$E_2^n(f)$	**Numero di punti**
10^{-2}	30.002373	0.002373	71
10^{-3}	30.000617	0.000617	113
10^{-4}	30.000050	0.000050	203
10^{-5}	30.000002	0.000002	371
10^{-6}	30.000000	0.000000	635
10^{-7}	30.000000	0.000000	1145
10^{-8}	30.000000	0.000000	2045

24

Esercizio 24

Descrizione: *Scrivere una function che implementi efficientemente il metodo delle potenze.*

Svolgimento:

Data una matrice $A \in \mathbb{R}^{nxn}$, il Metodo delle potenze si applica per determinare una approssimazione dell'autovalore di modulo massimo (ossia autovalore dominante) e del corrispondente autovettore. In dettaglio, definendo con λ_1 l'autovalore dominate (e semplice) e con $\underline{V}_1$ il corrispondente autovettore, il metodo delle potenze si applica quando per $\{\lambda_1\}\ \sigma(A)$ con $i = 1, 2, \cdots, n$ si ha $|\lambda_1| = \rho(A) > |\lambda_2| \geq \cdots \geq |\lambda_n|$ con gli autovalori contati con la loro molteplicità, essendo λ_1 semplice ed il suo modulo strettamente maggiore di quello degli altri autovalori. In generale, partendo da una data approssimazione iniziale $\underline{X}_0 \in \mathbb{R}^n$, per $k = 1, 2, \cdots, n$, il Metodo delle potenze si può esprimere come:

$$\underline{X}_{k+1} = A \cdot \underline{X}_k \tag{24.1}$$

di cui

$$\lambda = \frac{\underline{X}_k^T \cdot \underline{X}_{k+1}}{\underline{X}_k^T \cdot \underline{X}_k} \tag{24.2}$$

24.1 Algoritmo: *[lambda1, x1] = powerMethod(A, tol, itmax)*

Il Metodo delle potenze è implementato tramite il seguente codice di Matlab:

```
function [lambda1, x1] = powerMethod(A, tol, itmax)
    %
    % [lambda1, x1] = powerMethod(A [, tol [, itmax]])
    %
    %
    % Author : Alessandro Montaghi
    % Email  : alessandro.montaghi@gmail.com
    % Date   : Spring 2019
    % Course : Calcolo Numerico 2018/2019
    %
    % Function   : power method
    %
    % Description: function that implements the power method
    %              in order to compute the dominant eigenvalue
    %              (i.e., lambda1) and the correspondent
    %              eigenvector (i.e. X1) with a given tollerance
    %              tol (default: 1E-6) and maximum number of iterations
    %
    % Parameters : A - matrix R^nxn
    %              tol - tollerance
    %              itmax - maximum number of iterations
    %
    %
    [m, n] = size(A);
    if n ~= m
        error('inconsistent data');
    end
    if nargin <= 2
        if nargin <= 1
            tol = 1E-6;
        else
            if tol >= 0.1 || tol <= 0
                error('incorrect tolerance');
            end
        end
```

```
36          itmax = ceil(-log10(tol)) * n;
37      end
38      x = rand(n, 1);
39      lambda1 = 0;
40      for k = 1: itmax
41          x1 = x/norm(x);
42          x = A * x1;
43          lambda0 = lambda1;
44          lambda1 = x' * x1;
45          err = abs(lambda1 - lambda0);
46          if err <= tol * (1+abs(lambda1))
47              break;
48          end
49      end
50      if err > tol * (1 + abs(lambda1))
51          warning('convergence not obtained');
52      end
53      return
```

Listing 24.1: Codice Matlab che implementa il Metodo delle potenze.

24.2 Esecuzione

Di seguito si riporta il codice dell'esecuzione della funzione *powerMethod*.

```
A = [1 1 0 0; 1 2 0 1; 0 0 3 3; 0 1 3 2];
disp('Matrix: ');
disp(A);
tol = 10E-5;
[lambda1, x1] = powerMethod(A, tol);
disp('Eigenvalue (lambda1): ');
disp(lambda1);
disp('Eigenvector (x1): ');
disp(x1);
>> Matrix:
     1     1     0     0
     1     2     0     1
     0     0     3     3
     0     1     3     2
>> Eigenvalue (lambda1):
    5.6657
>> Eigenvector (x1):
    0.0438
    0.1943
    0.7316
    0.6519
```

Listing 24.2: Esecuzione del Metodo delle potenze.

25

Esercizio 25

Descrizione: *Sia data la matrice di Toeplitz simmetrica A_N in cui le extra-diagonali più esterne sono le none. Partendo dal vettore $\underline{u}_0 = (1, \cdots, 1)^T \in \mathbb{R}^N$, applicare il metodo delle potenze con tolleranza tol $= 10^{-10}$ per $N = 10 : 10 : 500$, utilizzando la function dell'esercizio 24. Graficare il valore dell'autovalore dominante, e del numero di iterazioni necessarie per soddisfare il criterio di arresto, rispetto ad N. Utilizzare la funzione spdiags di Matlab per creare la matrice e memorizzarla come matrice sparsa.*

Svolgimento:

25.1 Algoritmo: *[lambda1, x1, numIte] = powerMethod-Counter(S, tol)*

Consideriamo la seguente matrice di *Toeplitz* simmetrica (in cui le extra-diagonali più esterne sono le none),

$$A_N = \begin{vmatrix} 4 & -1 & & -1 & & \\ -1 & \ddots & \ddots & & \ddots & \\ & \ddots & \ddots & \ddots & & -1 \\ -1 & & \ddots & \ddots & \ddots & \\ & \ddots & & \ddots & \ddots & -1 \\ & & -1 & & -1 & 4 \end{vmatrix} \in \mathbb{R}^{NxN},\ N \geq 10, \tag{25.1}$$

Per lo svolgimento di questo esercizio è stata modificata la funzione *powerMethod* dell'esercizio 24 al fine di gestire una matrice di tipo sparsa (ossia una matrice il cui numero

di elementi non nulla è proporzionale alla sua dimensione) e ritornare oltre che il valore dell'autovalore dominante (*lambda1*) anche il numero iterazioni necessarie (*numberIteration*) per soddisfare il criterio di arresto per una data tolleranza (ossia $tol = 10^{-10}$).

```
function [lambda1, x1, numberIteration] = powerMethodCounter(S, tol)
    %
    % [lambda1, x1, numberIteration] = powerMethodCounter(S, tol)
    %
    % Author : Alessandro Montaghi
    % Email  : alessandro.montaghi@gmail.com
    % Date   : Spring 2019
    % Course : Calcolo Numerico 2018/2019
    %
    % Function   : power method Counter
    %
    % Description: function that implements the power method
    %              in order to compute the dominant eigenvalue
    %              (i.e., lambda1) and the correspondent
    %              eigenvector (i.e. x1) and the number of
    %              iterations needed to satisfy the
    %              stopping criterion (i.e. numberIteration)
    %
    % Parameters : S - Sparse matrix R^NxN
    %              tol - tollerance
    %
    persistent counter
    counter = 0;
    if ~issparse(S)
        error('Matrix is not sparse');
    end
    [m, n] = size(S);
    if n ~= m
        error('inconsistent data');
    end
    itmax = 10 + max(1, -ceil(log10(tol))) * n * n;
    x = ones(n, 1);
    lambda1 = 0;
    for k = 1: itmax
        counter = counter + 1;
```

```
        x1 = x/norm(x);
        x = S * x1;
        lambda0 = lambda1;
        lambda1 = x' * x1;
        err = abs(lambda1 - lambda0);
        if err <= tol * ( 1 + abs(lambda1))
            break;
        end
    end
    if err > tol * (1 + abs(lambda1))
        warning('convergence not obtained');
    end
    numberIteration = counter;
    return
```

Listing 25.1: Codice Matlab che implementa il Metodo delle potenze modificato (powerMethodCounter).

25.2 Esecuzione

Di seguito si riporta il codice dell'esecuzione della funzione *powerMethodCounter* per la matrice di Toeplitz simmetrica A_N, con $N = 10, 20, 30, \cdots, 500$.

```
tol = 1E-10;
len = length(10:10:500);
lambdaResult = zeros(1,len);
numIter = zeros(1,len);
index = 1;
for N = 10:10:500
    subd = -1 * ones(N, 1);
    diag = 4 * ones(N, 1);
    S = spdiags([subd subd diag subd subd], [-9 -1 0 1 9], N, N);
    % full(S) % Convert sparse matrix to full storage
    [lambda1, x1, numberIteration] = powerMethodCounter(S, tol);
    fprintf('N: %i - Lambda1: %f - number of iteration: %i\n', N,
        ↪ lambda1, numberIteration);
    lambdaResult(index) = lambdaResult(index) + lambda1;
    numIter(index) = numIter(index) + numberIteration;
    index = index + 1;;
end
```

```
17
18 x = linspace(10,500,50);
19
20 hold on
21 xlabel('Square Matrix dimension (N)')
22 ylabel('Dominant eigenvalue value (Lamda1)')
23 plot(x, lambdaResult,'-r*','MarkerSize',10)
24 grid on
25 hold off
26
27 hold on
28 xlabel('Square Matrix dimension (N)')
29 ylabel('Number of iteration')
30 plot(x, numIter,'-r*','MarkerSize',10)
31 grid on
32 hold off
33
34 >> N: 10 - Lambda1: 2.000000 - number of iteration: 2
35 >> N: 20 - Lambda1: 6.571812 - number of iteration: 51
36 >> N: 30 - Lambda1: 6.978650 - number of iteration: 96
37 >> ..................................................
38 >> ..................................................
39 >> N: 470 - Lambda1: 7.985825 - number of iteration: 1367
40 >> N: 480 - Lambda1: 7.986400 - number of iteration: 1417
41 >> N: 490 - Lambda1: 7.986940 - number of iteration: 1467
42 >> N: 500 - Lambda1: 7.987449 - number of iteration: 1518
```

Listing 25.2: Esecuzione della funzione powerMethodCounter.

I risultati ottenuti sono graficati di seguito.

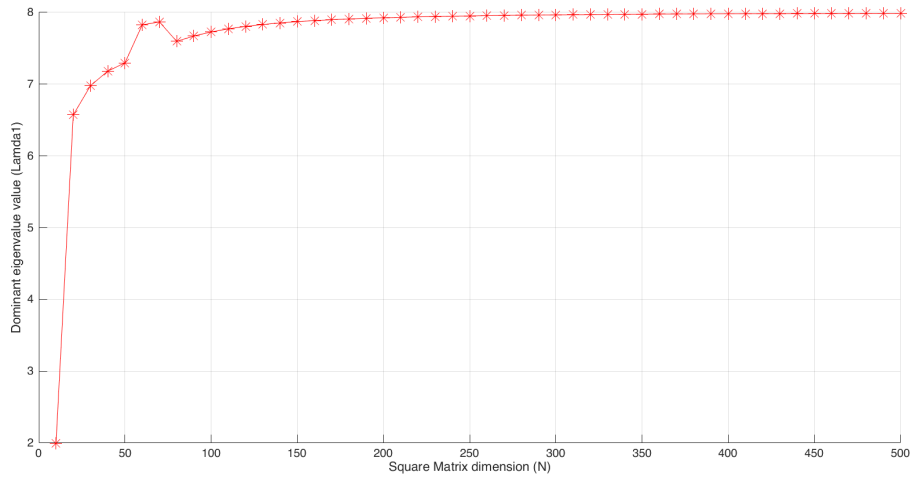


Figura 25.1: Grafico che riporta i valori dell'autovalore dominante (e semplice) ottenuti tramite la funzione *powerMethodCounter* nei punti $N = 10, 20, 30, \cdots, 500$.

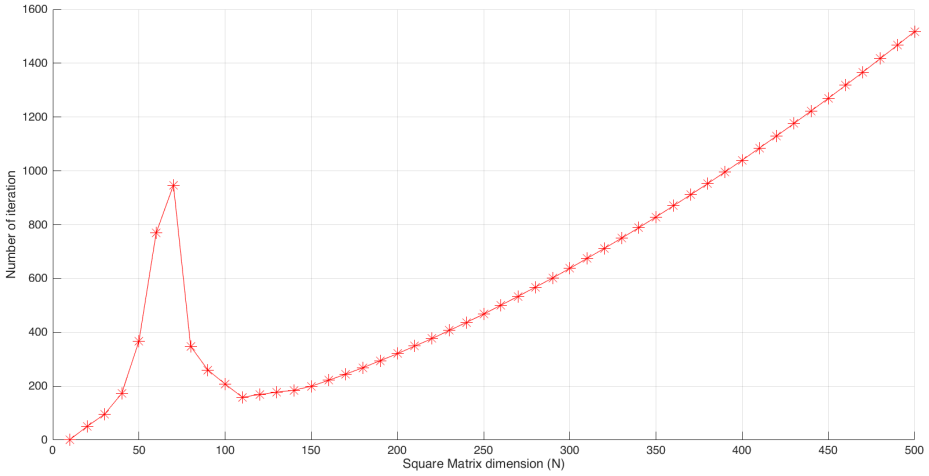


Figura 25.2: Grafico che riporta numero di iterazioni necessarie per soddisfare il criterio di arresto della funzione *powerMethodCounter*, rispetto ad N e dove $N = 10, 20, 30, \cdots, 500$.

26

Esercizio 26

Descrizione: *Scrivere una function che implementi efficientemente un metodo iterativo, per risolvere un sistema lineare, definito da un generico splitting della matrice dei coefficienti.*

Svolgimento:

Consideriamo una matrice $A \in \mathbb{R}^{nxn}$ non singolare (ossia $det(A) \neq 0$ e quindi esiste una soluzione e questa è unica) che supponiamo essere di tipo sparsa (ossia il cui numero di elementi non nulli è proporzionale alla dimensione di A) e che quindi la fattorizzazione di A per risolvere il sistema lineare $A \cdot \underline{X} = \underline{b}$ risulta non praticabile. Se la matrice A è partizionabile come:

$$A = M - N, \ (con \ det(M) \neq 0) \tag{26.1}$$

Il sistema lineare $A \cdot \underline{X} = \underline{b}$ diviene $(M - N) \cdot \underline{X} = \underline{b}$, ossia $M \cdot \underline{X} = N \cdot \underline{X} + \underline{b}$. Da questo, si deriva il seguente metodo iterativo per la risoluzione del sistema lineare $A \cdot \underline{X} = \underline{b}$ a partire da una data approssimazione lineare $\underline{X}_0$

$$M \cdot \underline{X}_k = N \cdot \underline{X}_{k-1} + \underline{b}, \ con \ k \geq 1 \tag{26.2}$$

La matrice M è scelta in modo tale che i sistemi lineari con M siano risolvibili direttamente senza la fattorizzazione di M. A tale proposito, supponiamo che la matrice A sia partizionabile come

$$A = D - L - U, \tag{26.3}$$

Dove D è una matrice che contiene gli elementi diagonali di A, la matrice L contiene gli opposti degli elementi strettamente triangonali inferiori di A e la matrice U contiene

gli opposti degli elementi strettamente triangonali superiori di A. A seconda del tipo di *splitting*, possiamo definire i seguenti due metodi iterativi:

$$A = M_J - N_J = D - (L + U),\ (metodo\ iterativo\ di\ Jacobi) \tag{26.4}$$

$$A = M_{GS} - N_{GS} = (D - L) - U,\ (metodo\ iterativo\ di\ Gauss - Seidel) \tag{26.5}$$

Sostituendo tali partizioni nel metodo iterativo $M \cdot \underline{X}_k = N \cdot \underline{X}_{k-1} + \underline{b}$ si definiscono i seguenti metodi iterativi:

$$D \cdot \underline{X}_k = (L + U) \cdot \underline{X}_{k-1} + \underline{b},\ (Metodo\ iterativo\ di\ Jacobi) \tag{26.6}$$

Dove ogni iterazione del metodo di Jacobi richiede la risoluzione di un sistema diagonale.

$$(D - L) \cdot \underline{X}_k = U \cdot \underline{X}_{k-1} + \underline{b},\ (Metodo\ iterativo\ di\ Gauss - Seidel) \tag{26.7}$$

Dove ogni iterazione del metodo di Gauss-Seidel richiede la risoluzione di un sistema triangolare inferiore.

Per concludere si ricordi che (*Teorema 1*) se la matrice A è *"diagonale dominante"* (*d.d.*), per riga o per colonna, allora i metodi di *Jacobi* e di *Gauss-Seidel* sono convergenti; mentre (*Teorema 2*) se la matrice A è una matrice *"simmetrica definita positiva"*, (*s.d.p.*) allora il metodo di *Gauss-Seidel* è convergente.

26.1 Algoritmo: *x = genericSplitting(A, b, msolve, tol)*

Una function che implementa efficientemente un metodo iterativo, per risolvere un sistema lineare, definito da un generico splitting della matrice dei coefficienti è implementato dal seguente codice di Matlab.

```
function x = genericSplitting(A, b, msolve, tol)
    %
    % x = genericSplitting(A, b, msolve, tol)
    %
    % Author : Alessandro Montaghi
    % Email  : alessandro.montaghi@gmail.com
    % Date   : Spring 2019
    % Course : Calcolo Numerico 2018/2019
    %
```

```
10      % Function   : generic splitting matrix solver
11      %
12      % Description: compute the solution of a linear system
13      %              of equations Ax = b with a given method
14      %              (msolve) and tollerance tol
15      %
16      % Parameters : A - R^nxn matrix
17      %              b -  column vector with n entries
18      %              msolve - Jacobi iterative method or
19      %                       Gauss-Seidel iterative method
20      %              tol - tollerance
21      %
22      [m, n] = size(A);
23      if n ~= m || m ~= length(b)
24          error('inconsistent data');
25      end
26      itmax = ceil( -log10(tol) ) * n;
27      x = zeros(n, 1);
28      tolb = tol * norm(b, Inf);
29      for i = 1 : itmax
30          r = A * x - b;
31          err = norm(r, Inf);
32          if err <= tolb
33              break;
34          end
35          u = msolve(A, r);
36          x = x - u;
37      end
38      if err > tolb
39          warning('Requested convergence not obtained');
40      end
41  return
```

Listing 26.1: Codice Matlab che implementa un generico splitting della matrice dei coefficienti.

27

Esercizio 27

Descrizione: *Scrivere le function ausiliarie, per la function del precedente esercizio, che implementano i metodi iterativi di Jacobi e Gauss-Seidel.*

Svolgimento:

La funzione *"genericSplitting*" implementata nell'esercizio 26 risolve un sistema lineare, definito da un generico splitting della matrice dei coefficienti, tramite il metodo iterativo di Jacobi ed il metodo iterativo di Gauss-Seidel. Questi due metodo sono implementati tramite le seguenti due funzioni ausiliarie.

27.1 Algoritmo: *y = Jacobi(A, b)*

La funzione ausiliaria *"Jacobi"* implementa il metodo iterativo di Jacobi.

```
function y = Jacobi(A, b)
    %
    % y = Jacobi(A, b)
    %
    % Author : Alessandro Montaghi
    % Email  : alessandro.montaghi@gmail.com
    % Date   : Spring 2019
    % Course : Calcolo Numerico 2018/2019
    %
    % Function   : Jacobi iterative method
```

```
11      %
12      % Description:  Method to solve a linear system
13      %               via Jacobi iteration. This is an
14      %               auxilary function of "genericSplitting"
15      %
16      % Parameters : A - R^nxn matrix of in Ax = b
17      %              b -  column vector with n entries
18      %                   in Ax = b
19      %
20      %
21      D = diag(A);
22      y = b./D;
23      return
```

Listing 27.1: Codice Matlab che implementa il metodo iterativo di Jacobi.

27.2 Esecuzione

Si riporta un esecuzione della funzione *"genericSplitting*" utilizzando il metodo iterativo di Jacobi.

```
1  A = [4 -1 -1; -2 6 1; -1 1 7];
2  disp('Matrix A of linear equatio Ax=b');
3  disp(A);
4  disp('Vector b of linear equatio Ax=b');
5  b = [3; 9; -6];
6  disp(b);
7  msolve = @Jacobi;
8  tol = 1E-5;
9  x = genericSplitting(A, b, msolve, tol);
10 disp('Result Jacobi');
11 disp(x);
```

Listing 27.2: Codice Matlab che esegue la funzione genericSplitting con il metodo iterativo di Jacobi.

Il corrispondente output usando la funzione *"genericSplitting*" con il metodo iterativo di Jacobi è il seguente.

```
Matrix A of linear equatio Ax=b
     4    -1    -1
    -2     6     1
    -1     1     7

Vector b of linear equation Ax=b
     3
     9
    -6

Result Jacobi
    1.0000
    2.0000
   -1.0000
```

Listing 27.3: Risultato della funzione genericSplitting con il metodo iterativo di Jacobi.

27.3 Algoritmo: *y = GaussSeidel(A, b)*

La funzione ausiliaria *"GaussSeidel"* implementa il metodo iterativo di Gauss-Seidel.

```
function x = GaussSeidel(A, b)
    %
    % y = GaussSeidel(A, b)
    %
    % Author : Alessandro Montaghi
    % Email  : alessandro.montaghi@gmail.com
    % Date   : Spring 2019
    % Course : Calcolo Numerico 2018/2019
    %
    % Function   : Gauss-Seidel iterative method
    %
    % Description:  Method to solve a linear system
    %               via Gauss-Seidel iteration. This is an
    %               auxilary function of "genericSplitting"
    %
```

```
16      % Parameters : A - R^nxn matrix of in Ax = b
17      %              b - column vector with n entries
18      %                  in Ax = b
19      %
20      %
21      n = length(b);
22      x = b;
23      for i = 1 : n
24          x(i) = x(i)/A(i,i);
25          x(i+1:n) = x(i+1:n) - x(i)*A(i+1:n,i);
26      end
27      return
```

Listing 27.4: Codice Matlab che implementa il metodo iterativo di Gauss-Seidel.

27.4 Esecuzione

Si riporta un esecuzione della funzione *"genericSplitting"* utilizzando il metodo iterativo di Gauss-Seidel.

```
A = [4 -1 -1; -2 6 1; -1 1 7];
disp('Matrix A of linear equatio Ax=b');
disp(A);
disp('Vector b of linear equatio Ax=b');
b = [3; 9; -6];
disp(b);
msolve = @GaussSeidel;
tol = 1E-5;
x = genericSplitting(A, b, msolve, tol);
disp('Result Gauss-Seidel');
disp(x);
```

Listing 27.5: Codice Matlab che esegue la funzione genericSplitting con il metodo iterativo di Gauss-Seidel.

Il corrispondente output usando la funzione *"genericSplitting"* con il metodo iterativo di Gauss-Seidel è il seguente.

```
Matrix A of linear equatio Ax=b
     4    -1    -1
    -2     6     1
    -1     1     7

Vector b of linear equation Ax=b
     3
     9
    -6

Result Gauss-Seidel
    1.0000
    2.0000
   -1.0000
```

Listing 27.6: Risultato della funzione genericSplitting con il metodo iterativo di Gauss-Seidel.

28

Esercizio 28

Descrizione: *Con riferimento alla matrice A_N definita in* (1)*, risolvere il sistema lineare $A_N x = (1, \cdots, 1)^T \in \mathbb{R}^N$ con i metodi di Jacobi e Gauss-Seidel, per $N = 10 : 10 : 500$, partendo dalla approssimazione nulla della soluzione, ed imponendo la norma del residuo sia minore di 10^{-8}. Utilizzare, a tale fine, la function dell'esercizio 26, scrivendo function ausiliare ad hoc (vedi esercizio 27) che sfruttino convenientemente la struttura di sparsità (nota) della matrice A_N. Graficare il numero delle iterazioni richieste dai due metodi iterativi, rispetto ad N, per soddisfare il criterio di arresto prefissato.*

Svolgimento:

Consideriamo il seguente sistema lineare

$$A_N \cdot \underline{x} = \underline{b} \tag{28.1}$$

Dove la matrice quadrata A_N non singolare ($det(A) \neq 0$), con dimensioni $N = 10, 20, 30, \cdots, 500$, rappresenta una matrice di *Toeplitz* simmetrica di tipo sparsa (ossia una matrice il cui numero di elementi non nulli è proporzionale alla dimensione della matrice), con il seguente formato (in cui le extra-diagonali più esterne sono le none),

$$A_N = \begin{vmatrix} 4 & -1 & & -1 & & \\ -1 & \ddots & \ddots & & \ddots & \\ & \ddots & \ddots & \ddots & & -1 \\ -1 & & \ddots & \ddots & \ddots & \\ & \ddots & & \ddots & \ddots & -1 \\ & & -1 & & -1 & 4 \end{vmatrix} \in \mathbb{R}^{NxN},\ N \geq 10, \tag{28.2}$$

Mentre il vettore $\underline{b}$ ha la seguente forma,

$$\underline{b} = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \in \mathbb{R}^N \tag{28.3}$$

Data la dimensione della matrice A_N e della sua caratteristica di sparsità, l'utilizzo di un metodo diretta di fattorizzazione per risolvere il sistema lineare $A_N \cdot \underline{x} = \underline{b}$ risulta non praticabile. Se la matrice A_N può essere scritta come,

$$A_N = M_N - N_N, \; con \; det(M_N) \neq 0 \tag{28.4}$$

Risulta possibile definire il seguente metodo iterativo in quanto, la matrice M_N è definita in modo tale che i sistemi lineari con la matrice M_N siano direttamente risolvibili senza richiedere la fattorizzazione di tale matrice.

$$M_N \cdot \underline{x}_k = N_N \cdot \underline{x}_{k-1} + \underline{b}, \; con \; k \geq 1 \tag{28.5}$$

Quando si parla di metodi iterativi, assume particolare importanza la definizione di un opportuno criterio di arresto per l'iterazione. Tra i vari criteri di arresto, possiamo ricordare quello che si basa sul controllo della norma del vettore residuo,

$$\underline{r}_k = A_N \cdot \underline{x}_k - \underline{b} = M_N \cdot \underline{x}_k - N_N \cdot \underline{x}_k - \underline{b}, \tag{28.6}$$

Qualora il criterio di arresto si basi sulla norma del vettore residuo, al fine di evitare operazioni ridondanti l'iterazione $M_N \cdot \underline{x}_{k+1} = N_N \cdot \underline{x}_k + \underline{b}$, per $k = 0, 1, \cdots,$, è convenientemente riformulata come segue:

$$\begin{aligned}
&M_N \cdot \underline{x}_{k+1} = N_N \cdot \underline{x}_k + \underline{b} \\
&M_N \cdot \underline{x}_{k+1} = -(-N_N \cdot \underline{x}_k - \underline{b}) \\
&M_N \cdot \underline{x}_{k+1} = M_N \cdot \underline{x}_k - (M_N \cdot \underline{x}_k - N_N \cdot \underline{x}_k - \underline{b}) \\
&M_N \cdot \underline{x}_{k+1} - M_N \cdot \underline{x}_k = -(M_N \cdot \underline{x}_k - N_N \cdot \underline{x}_k - \underline{b}) \\
&M_N \cdot (\underline{x}_{k+1} - \underline{x}_k) = -(M_N \cdot \underline{x}_k - N_N \cdot \underline{x}_k - \underline{b}) \\
&M_N \cdot \underline{u}_k = -\underline{r}_k
\end{aligned}$$

Quindi si controlla,

$$\left\| \underline{r}_k \right\| \leq tol \cdot \left\| \underline{b} \right\| \tag{28.7}$$

Se si, si interrompe l'iterazione e si esce dal metodo iterativo, altrimenti si risolve,

$$M_N \cdot \underline{u}_k = -\underline{r}_k$$
$$\underline{x}_{k+1} = \underline{x}_k + \underline{u}_k$$

Infine, supponiamo che la matrice A_N sia partizionabile come

$$A_N = D_N - L_N - U_N, \tag{28.8}$$

Dove D_N è una matrice che contiene gli elementi diagonali di A_N, la matrice L_N contiene gli opposti degli elementi strettamente triangonali inferiori di A_N e la matrice U_N contiene gli opposti degli elementi strettamente triangonali superiori di A_N. A seconda del tipo di *splitting*, possiamo definire i seguenti due metodi iterativi:

$$A_N = M_{N_J} - N_{N_J} = D_N - (L_N + U_N),\ (metodo\ di\ Jacobi) \tag{28.9}$$

$$A_N = M_{N_{GS}} - N_{N_{GS}} = (D_N - L_N) - U_N,\ (metodo\ di\ Gauss - Seidel) \tag{28.10}$$

Sostituendo tali partizioni nel metodo iterativo $M_N \cdot \underline{X}_k = N_N \cdot \underline{X}_{k-1} + \underline{b}$ si definiscono i seguenti metodi iterativi:

$$D_N \cdot \underline{X}_k = (L_N + U_N) \cdot \underline{X}_{k-1} + \underline{b},\ (Metodo\ iterativo\ di\ Jacobi) \tag{28.11}$$

Dove ogni iterazione del metodo di Jacobi richiede la risoluzione di un sistema diagonale.

$$(D_N - L_N) \cdot \underline{X}_k = U_N \cdot \underline{X}_{k-1} + \underline{b},\ (Metodo\ iterativo\ di\ Gauss - Seidel) \tag{28.12}$$

Dove ogni iterazione del metodo di Gauss-Seidel richiede la risoluzione di un sistema triangolare inferiore.

Per concludere si ricordi che (*Teorema 1*) se la matrice A_N è *"diagonale dominante"* (*d.d.*), per riga o per colonna, allora i metodi di *Jacobi* e di *Gauss-Seidel* sono convergenti; mentre (*Teorema 2*) se la matrice A è una matrice *"simmetrica definita positiva"*, (*s.d.p.*) allora il metodo di *Gauss-Seidel* è convergente.

28.1 Algoritmo: *[x, nIte] = splittingSparseMatrix(b, matvec, msolve, tol)*

La funzione *splittingSparseMatrix*, gestisce insieme alle funzioni accessorie matvecSparse, jacobiSparse e gaussSeidelSparse la risoluzione del sistema lineare $A_N \cdot \underline{x} = \underline{b}$, dove la matrice A_N corrisponde alla matrice descritta dall'esercizio 25.

```
function [x, numberIteration] = splittingSparseMatrix(b, matvec, msolve,
    ↪ tol)
    %
    % [x, numberIteration] = splittingSparseMatrix(b, matvec, msolve [,
        ↪ tol])
    %
    % Author : Alessandro Montaghi
    % Email  : alessandro.montaghi@gmail.com
    % Date   : Spring 2019
    % Course : Calcolo Numerico 2018/2019
    %
    % Function   :  Linear equation  solver
    %               for a given sparse matrix.
    %
    % Description: function that implements the solution
    %              of a linear equation system (Ax = b), where A
    %              is a given (Toeplitz) sparse matrix, using
    %              the Jacobi method or the Gauss-Seidel Method.
    %
    % Parameters : b -  column vector with n entries.
    %              matvec - matrix-vector product, for
    %                       the Toeplitz matrix provided
    %                       by exercise 25.
    %              msolve - Jacobi iterative method or
    %                       Gauss-Seidel iterative method.
    %              tol - tollerance (default: 1E-8).
    %
    n = length(b);
    if nargin == 3
        tol = 1E-8;
    end
    if tol >= 0.1 || tol <= 0
        error('Inconsistent Tolerance value');
```

```
32      end
33      x = zeros(n, 1);
34      itmax = 10 + max(1,-ceil(log10(tol)))* n * n ;
35      tolb = tol * norm(b, inf);
36      numberIteration = 0;
37      for i = 1 : itmax
38          numberIteration = numberIteration + 1;
39          r = matvec(x) - b;
40          err = norm(r, inf);
41          if err <= tolb
42              break;
43          end
44          r = msolve(r);
45          x = x - r;
46      end
47      if err > tolb
48          str = func2str(msolve);
49          %fprintf('%s',str);
50          fprintf('Method: %s - Tolerance %g not reached in %i iterations\
                ↪ n', str, tolb, itmax);
51      end
52      return
```

Listing 28.1: Codice Matlab che implementa la risoluzione del sistema lineare $A_N \cdot \underline{x} = \underline{b}$.

Di seguito si riportanto le funzioni accessorie,

```
1   function y = matvecSparse(x)
2       %
3       % y = matvecSparse(x)
4       %
5       %
6       % Author : Alessandro Montaghi
7       % Email  : alessandro.montaghi@gmail.com
8       % Date   : Spring 2019
9       % Course : Calcolo Numerico 2018/2019
10      %
11      % Function   :  matrix-vector product
12      %
13      % Description: function that implements the matrix-vector
```

```
    %               product, for the Toeplitz matrix provided
    %               by exercise 25.
    %
    %
    y = 4 * x;
    y(9 : end) = y(9 : end) - x(1 : end - 8);
    y(2 : end) = y(2 : end) - x(1 : end - 1);
    y(1 : end - 1) = y(1 : end - 1) - x(2 : end);
    y(1 : end - 8) = y(1 : end - 8) - x(9 : end);
    return
```

Listing 28.2: Codice Matlab che implementa la funzione matvecSparse, accessoria a splittingSparseMatrix.

```
function y = jacobiSparse(b)
    %
    % y = jacobiSparse(b)
    %
    % Author : Alessandro Montaghi
    % Email  : alessandro.montaghi@gmail.com
    % Date   : Spring 2019
    % Course : Calcolo Numerico 2018/2019
    %
    % Function   :  Jacobi method
    %
    % Description: function that implements the Jacobi method.
    %              This function associated with the function
    %              splittingSparseMatrix.
    %
    y = b./4;
    return
```

Listing 28.3: Codice Matlab che implementa la funzione jacobiSparse, accessoria a splittingSparseMatrix.

```
function y = gaussSeidelSparse(b)
    %
    % y = gaussSeidelSparse(b)
    %
    % Author : Alessandro Montaghi
    % Email  : alessandro.montaghi@gmail.com
    % Date   : Spring 2019
    % Course : Calcolo Numerico 2018/2019
    %
    % Function   :  Gauss-Seidel method
    %
    % Description: function that implements the Gauss-Seidel method.
    %              This function associated with the function
    %              splittingSparseMatrix.
    %
    n = length(b);
    y = b;
    y(1) = y(1)/4;
    for i = 2 : n
        y(i) = (y(i) + y(i-1))/4;
    end
    for j = 9:n
        y(j) = (y(j) + y(j-1))/4;
    end
    return
```

Listing 28.4: Codice Matlab che implementa la funzione gaussSeidelSparse, accessoria a splittingSparseMatrix.

28.2 Esecuzione

Il seguente codice di Matlab riporta il risultato dell'esecuzione della funzione *splitting-SparseMatrix* utilizzando sia il metodo di Jacobi che quello di Gauss-Seidel per la data matrice A_N con dimensioni $N = 10, 20, 30, \cdots, 500$.

```
len = length(10:10:500);
jacobiResultIteration = zeros(1, len);
gaussSeidelResultIteration = zeros(1, len);
matvec = @matvecSparse;
index = 1;

for N = 10:10:500
    fprintf('Matrix dimension %i x %i\n', N, N);
    b = ones(N, 1);
    % Jacobi Method
    msolve = @jacobiSparse;
    [x, iterJacobi] = splittingSparseMatrix(b, matvec, msolve);
    fprintf('Jacobi method number of iteration: %i\n', iterJacobi);
    jacobiResultIteration(index) =...
        jacobiResultIteration(index) + iterJacobi;
    % Gauss-Seidel Method
    msolve = @gaussSeidelSparse;
    [x, iterGaussSeidel] = splittingSparseMatrix(b, matvec, msolve);
    fprintf('Gauss-Seidel method number of iteration: %i\n',
        ↪ iterGaussSeidel);
    gaussSeidelResultIteration(index) =...
        gaussSeidelResultIteration(index) + iterGaussSeidel;
    index = index + 1;
end

x = linspace(10,500,50);

hold on
xlabel('Square Matrix dimension (N)')
ylabel('Number of Iteration')
plot(x, jacobiResultIteration,'-r*','MarkerSize',10)
plot(x, gaussSeidelResultIteration,'-bs','MarkerSize',10)
grid on
hold off
```

Listing 28.5: Codice Matlab per l'esecuzione della funzione splittingSparseMatrix, usando il metodo di Jacobi e di Gauss-Seidel per la data matrice A_N con dimensioni $N = 10, 20, 30, \cdots, 500$.

I risultati del numero di iterazioni richieste dai due metodi iterativi, rispetto ad N, per soddisfare il criterio di arresto prefissato sono rappresentati dal seguente grafico.

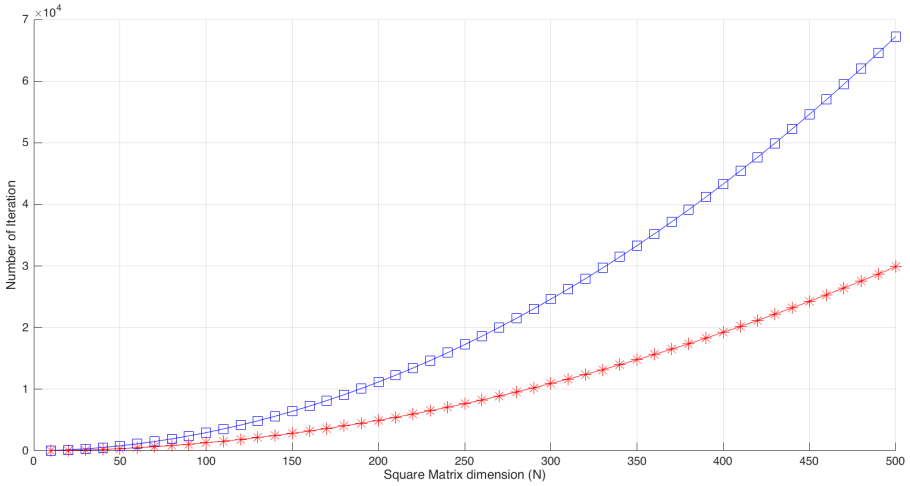


Figura 28.1: Grafico che riporta il numero di iterazioni necessarie al metodo di Jacobi (in rosso) e al metodo di Gauss-Seidel (in blue) per soddisfare il criterio di arresto prefissato, in funzione delle dimensioni della data matrice A_N, con $N = 10, 20, 30, \cdots, 500$.