

# Elaborato di Calcolo Numerico A.A 2023-2024

Autore	Matricola	e-mail
Ede Boanini	7054924	ede.boanini@edu.unifi.it
Ginevra Lavacchi	7079429	ginevra.lavacchi@edu.unifi.it



# 1 Esercizio 1

Dimostrare che:

$$\frac{25f(x) - 48f(x-h) + 36f(x-2h) - 16f(x-3h) + 3f(x-4h)}{12h} = f'(x) + O(h^4)$$

## 1.1 Soluzione

Sviluppando le singole funzioni con il polinomio di Taylor centrato in x otteniamo:

$$f(x-h) = \frac{f^{(0)}(x)}{0!} \cdot (x-h-x)^0 + \frac{f'(x)}{1!} \cdot (x-h-x)^1 + \frac{f''(x)}{2!} \cdot (x-h-x)^2 + \frac{f'''(x)}{3!} \cdot (x-h-x)^3 + \frac{f^{(4)}(x)}{4!} \cdot (x-h-x)^4 + O(h^5)$$

$$f(x-2h) = \frac{f^{(0)}(x)}{0!} \cdot (x-2h-x)^0 + \frac{f'(x)}{1!} \cdot (x-2h-x)^1 + \frac{f''(x)}{2!} \cdot (x-2h-x)^2 + \frac{f'''(x)}{3!} \cdot (x-2h-x)^3 + \frac{f^{(4)}(x)}{4!} \cdot (x-2h-x)^4 + O(h^5)$$

$$f(x-3h) = \frac{f^{(0)}(x)}{0!} \cdot (x-3h-x)^0 + \frac{f'(x)}{1!} \cdot (x-3h-x)^1 + \frac{f''(x)}{2!} \cdot (x-3h-x)^2 + \frac{f'''(x)}{3!} \cdot (x-3h-x)^3 + \frac{f^{(4)}(x)}{4!} \cdot (x-3h-x)^4 + O(h^5)$$

$$f(x-4h) = \frac{f^{(0)}(x)}{0!} \cdot (x-4h-x)^0 + \frac{f'(x)}{1!} \cdot (x-4h-x)^1 + \frac{f''(x)}{2!} \cdot (x-4h-x)^2 + \frac{f'''(x)}{3!} \cdot (x-4h-x)^3 + \frac{f^{(4)}(x)}{4!} \cdot (x-4h-x)^4 + O(h^5)$$

Semplificando ogni singola funzione otteniamo:

$$f(x-h) = f(x) - hf'(x) + \frac{h^2 f''(x)}{2} - \frac{h^3 f'''(x)}{6} + \frac{h^4 f^{(4)}(x)}{24} + O(h^5)$$

$$f(x-2h) = f(x) - 2hf'(x) + \frac{4h^2 f''(x)}{2} - \frac{8h^3 f'''(x)}{6} + \frac{16h^4 f^{(4)}(x)}{24} + O(h^5)$$

$$f(x-3h) = f(x) - 3hf'(x) + \frac{9h^2 f''(x)}{2} - \frac{27h^3 f'''(x)}{6} + \frac{81h^4 f^{(4)}(x)}{24} + O(h^5)$$

$$f(x-4h) = f(x) - 4hf'(x) + \frac{16h^2 f''(x)}{2} - \frac{64h^3 f'''(x)}{6} + \frac{256h^4 f^{(4)}(x)}{24} + O(h^5)$$

Sostituendo gli sviluppi di Taylor appena ottenuti nel numeratore, avremo:

$$\begin{aligned} & 25f(x) - 48 \cdot \left( f(x) - hf'(x) + \frac{h^2 f''(x)}{2} - \frac{h^3 f'''(x)}{6} + \frac{h^4 f^{(4)}(x)}{24} + O(h^5) \right) \\ & + 36 \cdot \left( f(x) - 2hf'(x) + \frac{4h^2 f''(x)}{2} - \frac{8h^3 f'''(x)}{6} + \frac{16h^4 f^{(4)}(x)}{24} + O(h^5) \right) \\ & - 16 \cdot \left( f(x) - 3hf'(x) + \frac{9h^2 f''(x)}{2} - \frac{27h^3 f'''(x)}{6} + \frac{81h^4 f^{(4)}(x)}{24} + O(h^5) \right) \\ & + 3 \cdot \left( f(x) - 4hf'(x) + \frac{16h^2 f''(x)}{2} - \frac{64h^3 f'''(x)}{6} + \frac{256h^4 f^{(4)}(x)}{24} + O(h^5) \right) \end{aligned}$$

Proseguendo con lo sviluppo del numeratore:

$$\begin{aligned} & 25f(x) - 48f(x) + 48hf'(x) - 24h^2 f''(x) + 8h^3 f'''(x) - 2h^4 f^{(4)}(x) + O(h^5) \\ & + 36f(x) - 72hf'(x) + 72h^2 f''(x) - 48h^3 f'''(x) + 24h^4 f^{(4)}(x) + O(h^5) \\ & - 16f(x) + 48hf'(x) - 72h^2 f''(x) + 72h^3 f'''(x) - 54h^4 f^{(4)}(x) + O(h^5) \\ & + 3f(x) - 12hf'(x) + 24h^2 f''(x) - 32h^3 f'''(x) + 32h^4 f^{(4)}(x) + O(h^5) \\ & = 12hf'(x) + O(h^5) \end{aligned}$$

Pertanto, si dimostra che:

$$\frac{12hf'(x) + O(h^5)}{12h} = \frac{12h(f'(x) + O(h^4))}{12h} = f'(x) + O(h^4)$$

## 2 Esercizio 2

La funzione

$$f(x) = 1 + x^2 + \frac{\log(|3(1-x) + 1|)}{80}, \quad x \in [1, \frac{5}{3}]$$

ha un asintoto in  $x = \frac{4}{3}$ , in cui tende a  $-\infty$ . Graficarla in Matlab, utilizzando

$$x = \text{linspace}(1, \frac{5}{3}, 100001)$$

(in modo che il floating di  $\frac{4}{3}$ , in cui tende a  $-\infty$  sia contenuto in x) e vedere dove si ottiene il minimo. Commentare i risultati ottenuti.

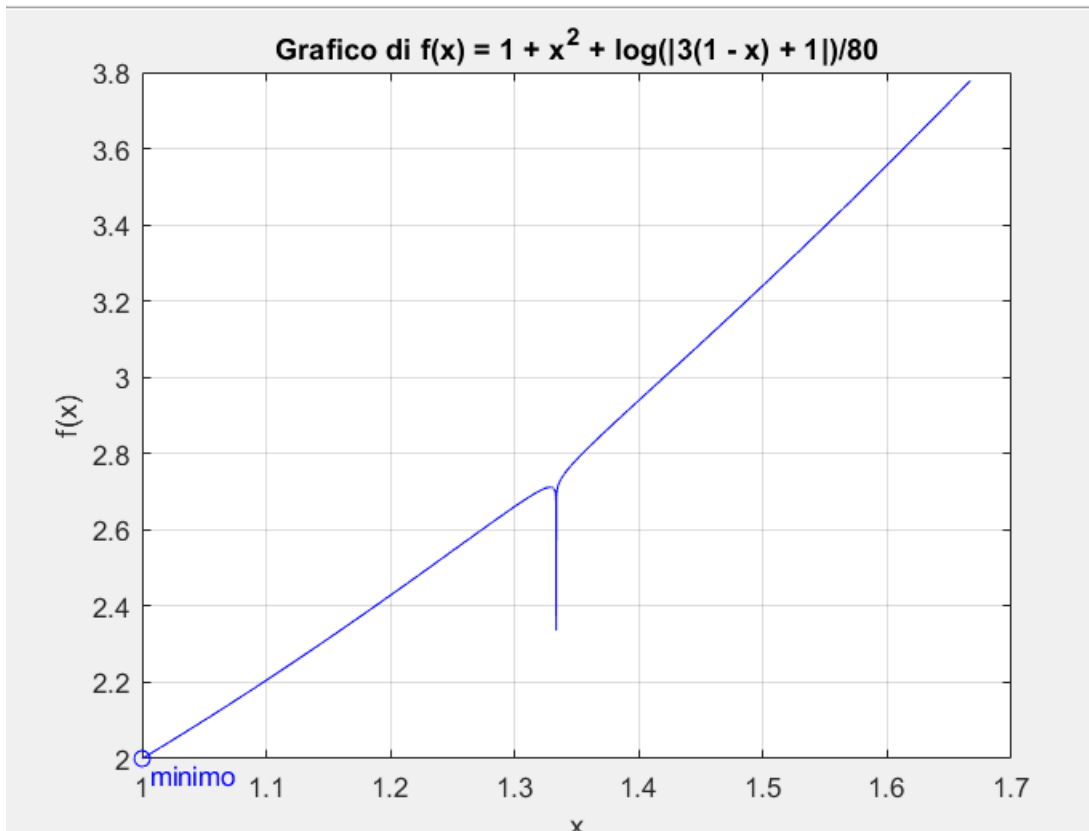
### 2.1 Soluzione

Il codice Matlab è:

```
1 % Definisco i limiti dell' intervallo x
2 x1 = 1; x2 = 4/3 - 0.01; % Intervallo a sinistra dell'asintoto
3 x3 = 4/3 + 0.01; x4 = 5/3; % Intervallo a destra dell'asintoto
4 % Uso linspace per generare i valori x
5 x=linspace(x1,x4,100001);
6 % Calcolo i corrispondenti valori y
7 y= 1 + x.^2 + (log(abs(3*(1 - x) + 1)))/80;
8 %calcolo i valori dei limiti della funzione in 4/3
9 x_sx=1 + x2.^2 + (log(abs(3*(1 - x2) + 1)))/80
10 x_dx=1 + x3.^2 + (log(abs(3*(1 - x3) + 1)))/80
11 % Creo il grafico
12 figure
13 plot(x,y,'b')
14 hold on
15 title('Grafico di f(x) = 1 + x^2 + log(|3(1 - x) + 1|)/80')
16 xlabel('x')
17 ylabel('f(x)')
18 grid on
19 % Trovo il minimo della funzione
20 [min_val, min_idx] = min(y);
21 min_x = x(min_idx);
22
23 plot(min_x,min_val,'bo')
24 text(min_x, min_val, ' minimo','VerticalAlignment','top', 'HorizontalAlignment','
    left', 'FontSize',10,'Color','b');
25 fprintf('Il minimo della funzione si ottiene per x = %f, dove f(x) = %f\n', min_x,
    min_val);
26 fprintf('Limite sinistro:%f',x_sx);
27 fprintf('Limite destro:%f',x_dx);
```

Listing 1: Codice per graficare f(x)

questo è il grafico che ne risulta:



Nel codice Matlab abbiamo verificato l'andamento della funzione e oltre alla ricerca del minimo abbiamo calcolato i valori che prende  $f(x)$  nel punto  $x_0 = \frac{4}{3}$ . Il grafico mostra una funzione crescente con minimo in  $X = 1$  dove  $f(x) = 2$ . I valori del limite sinistro e del destro che la funzione prende nell'avvicinarsi al punto  $x_0 = \frac{4}{3}$  sono rispettivamente  $f(x_0)_{sx} = 2.7074$  e  $f(x_0)_{dx} = 2.7607$ . Il risultato ottenuto è dovuto all'utilizzo di una aritmetica finita da parte del calcolatore che quindi genera un errore di rappresentazione.

### 3 Esercizio 3

Spiegare in modo esaustivo il fenomeno della *cancellazione numerica*. Fare un esempio che la illustri, spiegandone i dettagli.

#### 3.1 Soluzione

La cancellazione numerica è la perdita di cifre significative nella somma, in aritmetica finita, di numeri quasi opposti. Questo fenomeno è dovuto al mal condizionamento della somma algebrica quando i due numeri da sommare sono di segno discorde.

Infatti, sappiamo che per la somma il numero  $K$  di condizionamento è dato da:

$$K = \frac{|x| + |y|}{|x + y|}$$

È evidente che:

- Se  $x, y > 0$  ne segue che  $|x| + |y| = |x + y|$  e quindi  $K = 1$
- Se  $x \approx -y$  allora  $|x + y| \ll |x| + |y|$  e quindi  $K \gg 1$ ; Pertanto, la somma è mal condizionata.

Un esempio che illustra il malcondizionamento della somma:

```
format long e
(1+(1e-14-1))*1e14
```

Eseguendo lo script si ottiene:

$$ans = 9.992007221626409e - 01 \approx 0.9992007221626$$

Non è corretto considerando che il risultato atteso è 1. Il motivo di questa discordanza è dato dal mal condizionamento, spiegato sopra.

## 4 Esercizio 4

Scrivere una *function* Matlab che implementi in modo efficiente il metodo di bisezione.

### 4.1 Soluzione

```
1 function [x,count] = bise( a, b, f, tolx )
2 %
3 % x = bise( a, b, f, tolx ) Metodo di bisezione per calcolare
4 % una radice di f(x), interna ad [a,b],
5 % con tolleranza tolx. Il metodo conta anche il numero di iterazioni
6 % (count)
7 %
8 count=0;
9 if a>=b, error('estremi intervallo errati'), end
10 if tolx<=0; error('tolleranza non appropriata'), end
11 fa = feval(f,a);
12 fb = feval(f,b);
13 if fa*fb>=0, error('intervallo di confidenza non appropriato'), end
14 imax = ceil( log2(b-a)-log2(tolx) );
15 if imax<1, x = (a+b)/2; return, end
16 for i = 1:imax
17 x = (a+b)/2;
18 fx = feval( f, x );
19 f1x = abs(fb-fa)/(b-a);
20 count=count+1;
21 if abs(fx)<=tolx*f1x
22 break
23 elseif fa*fx<0
24 b = x; fb = fx;
25 else
26 a = x; fa = fx;
27 end
28 end
29 return
```

Listing 2: Codice metodo di bisezione

## 5 Esercizio 5

Scrivere *function* Matlab distinte che implementino efficientemente i metodi di Newton e delle secanti per la ricerca degli zeri di una funzione  $f(x)$ .

## 5.1 Soluzione

L'implementazione del metodo di Newton tramite la function newton:

```
1 function [x,count] = newtonZeri(x0,f,df,tol,itmax)
2 %
3 % [x,it] = newtonZeri(x0,f,df,tol,itmax)
4 % funzione che ricerca gli zeri della funzione f tramite il metodo di Newton con
   approssimazione iniziale x0
5 %
6 % input: x0- approssimazione iniziale della radice
7 %       f- funzione per cui si ricercano gli zeri
8 %       df- derivata di f
9 %       tol- tolleranza
10 %       itmax- massimo numero di iterazioni
11 %
12 % output: x- radice
13 %         count- numero iterazioni
14 %
15 if nargin~=5,error("input errato"),end;
16 if tol<0,error('Errore tolleranza: la tolleranza deve essere maggiore di 0.'),end;
17 if itmax<=0
18 error('Errore numero iterazioni: le iterazioni devono essere maggiori di 0.');
```

```
19 end;
20 count=0;
21 x = x0;
22
23 for i=1:itmax
24 x0 = x;
25 fx0 = feval(f,x0);
26 dfx0 = feval(df,x0);
27 count=count+1;
28 if dfx0==0, error('Errore derivata prima uguale a 0'),end;
29 x = x0-(fx0/dfx0);
30 if abs(x-x0)<=tol
31     break
32 end
33 end
34 if abs(x-x0)>tol
35 disp("Il metodo non converge");
36 end
37 return
38 end
```

Listing 3: Codice del metodo di newton per la ricerca degli zeri della funzione

L'implementazione del metodo delle secanti tramite la function sec

```
1 function [x,count] = sec(x0,x1,f,tol,itmax)
2 %
3 % [x,it] = sec(x0,x1,f,tol,itmax)
4 % funzione che ricerca gli zeri della funzione f tramite il metodo delle secanti
   con le approssimazioni iniziale x0 e x1.
5 %
6 % input:x0- approssimazione iniziale della radice
7 %       x1- seconda approssimazione radice
8 %       f- funzione per cui si ricercano gli zeri
9 %       tol- tolleranza
10 %       itmax- massimo numero di iterazioni
11 %
12 % output:x- radice
```

```

13 %           count- numero iterazioni
14 %
15 if nargin~=5,error("input errato"),end;
16 if tol<=0,error("tolleranza non adeguata"),end;
17 if itmax<=0,error("num it non adeguato"),end;
18 count=0;
19 fx0=feval(f,x0);
20 fx1=feval(f,x1);
21
22 for i=1:itmax
23     if fx1==fx0
24         error("approssimazione errata");
25     end
26     x=(fx1*x0-fx0*x1)/(fx1-fx0);
27     count=count+1;
28     if abs(x-x1)<=tol
29         break
30     elseif i<itmax
31         x0=x1;
32         fx0=fx1;
33         x1=x;
34         fx1=feval(f,x1);
35     end
36 end
37 if abs(x-x1)>tol
38     error("tolleranza non rispettata");
39 end
40
41 return
42 end

```

Listing 4: Codice del metodo delle secanti

## 6 Esercizio 6

Utilizzare le *function* dei precedenti esercizi per determinare una approssimazione della radice della funzione

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

per  $tol = 10^{-3}, 10^{-6}, 10^{-9}, 10^{-12}$ , partendo da  $X_0 = 1$  ( $x_1 = 0.9$  per il metodo delle secanti). Per il metodo di bisezione, usare l'intervallo di confidenza iniziale  $[-0.1, 1]$ . Tabulare i risultati, in modo da confrontare il costo computazionale di ciascun metodo.

### 6.1 Soluzione

Di seguito è riportato il codice usato per calcolare le approssimazioni della funzione:

```

1 %tolleranze
2 toll=[1e-3;1e-6;1e-9;1e-12];
3 %funzione
4 f=@(x) exp(x) - cos(x);
5 %derivata della funzione
6 f1=@(x) exp(x) +sin(x);
7 %intervallo confidenza per metodo bisezione
8 a=-0.1;
9 b=1;
10 %approssimazioni iniziali
11 x0=1;

```

```

12 x1=0.9;
13
14 for i=1:length(toll)
15     [x_b,count_b]=bise(a, b, f, toll(i));
16     [x_n, count_n]=newtonZeri(x0,f,f1,toll(i),1000);
17     [x_s,count_s]=sec(x0,x1,f,toll(i),1000);
18
19     radici=[x_s;x_b;x_n];
20     countazioni=[count_s;count_b;count_n];
21     disp("tolleranza = "+toll(i));
22     metodo=["secanti";"bisezione";"newton"];
23
24     t=table(metodo,radici,countazioni);
25     disp(t);
26 end

```

Listing 5: Codice per calcolare le approssimazioni  $f(x)$

Di seguito abbiamo riportato i valori che prendono le radici della funzione  $f(x) = e^x - \cos x$  ottenuti tramite le function che implementano i metodi delle secanti, di bisezione e di newton precedentemente implementati, per il conteggio delle iterazioni il codice è stato modificato aggiungendo una variabile counter che ha il compito di contare le iterazioni effettuate.

Per la tolleranza= $10^{-3}$

<i>Metodo</i>	<i>Radici</i>	<i>Iterazioni</i>
<i>secanti</i>	1.1522e-06	6
<i>bisezione</i>	0.00097656	9
<i>newton</i>	2.8423e-09	5

Per la tolleranza= $10^{-6}$

<i>Metodo</i>	<i>Radici</i>	<i>Iterazioni</i>
<i>secanti</i>	2.0949e-16	8
<i>bisezione</i>	9.5367e-07	19
<i>newton</i>	3.5748e-17	6

Per la tolleranza= $10^{-9}$

<i>Metodo</i>	<i>Radici</i>	<i>Iterazioni</i>
<i>secanti</i>	2.0949e-16	8
<i>bisezione</i>	9.3132e-10	29
<i>newton</i>	3.5748e-17	7

Per la tolleranza= $10^{-12}$

<i>Metodo</i>	<i>Radici</i>	<i>Iterazioni</i>
<i>secanti</i>	-1.2557e-17	9
<i>bisezione</i>	9.0949e-13	39
<i>newton</i>	3.5748e-17	7

## 7 Esercizio 7

Applicare gli stessi metodi e dati del precedente esercizio, insieme al metodo di Newton modificato, per la funzione

$$f(x) = e^x - \cos x + \sin x - x(x+2)$$

Tabulare i risultati, in modo da confrontare il costo computazionale e l'accuratezza di ciascun metodo. Commentare i risultati ottenuti.



## 7.1 Soluzione

```
1 function [x, it] = newtonModificato(x0, f, f1, m, tolx, maxit)
2 %
3 % [x, it] = newtonModificato(x0, f, f1, m, tolx, maxit)
4 % Funzione che implementa il metodo di Newton modificato per determinare
5 % una approssimazione della radice
6 %
7 % Input: x0 - approssimazione iniziale della radice
8 %         f - funzione che implementa f(x)
9 %         f1 - funzione che implementa f'(x)
10 %         m - molteplicità della radice di f(x)
11 %         tolx - accuratezza richiesta (default = 1e-6)
12 %         maxit - numero massimo di iterazioni (default = 1000)
13 %
14 % Output: x - approssimazione della radice di f(x);
15 %         it - num iterazioni
16 %
17 if nargin < 4
18     error('Numero di parametri in ingresso errato');
19 elseif nargin == 4
20     tolx = 1e-6;
21     maxit = 1000;
22 elseif nargin == 5
23     maxit = 1000;
24 end
25 if tolx < 0 || maxit <= 0
26     error('Parametri in ingresso errati');
27 end
28
29 x = x0;
30 for it = 1:maxit
31     fx0 = feval(f, x);
32     f1x0 = feval(f1, x);
33     if f1x0 == 0
34         error('La derivata prima si annulla');
35     end
36     x = x - m * (fx0 / f1x0);
37     % Verifica della condizione di convergenza
38     if abs(x - x0) <= tolx * (1 + abs(x0))
39         break
40     else
41         x0 = x;
42     end
43     if it == maxit
44         disp('Il metodo non converge. ');
45     end
46 end
47 return
```

Listing 6: Codice del metodo newtonModificato

```
1 % Tolleranze
2 toll=[1e-3;1e-6;1e-9;1e-12];
3
4 % Funzione
5 f=@(x) exp(x) - cos(x) + sin(x) - x*(x+2);
6
7 % Derivata della funzione
```

```

8  f1=@(x) exp(x) + sin(x) + cos(x) -2*x -2;
9
10 % Intervallo confidenza per metodo bisezione
11 a=-0.1;
12 b=1;
13
14 % Approssimazioni iniziali
15 x0=1;
16 x1=0.9;
17
18 % Molteplicità della radice m=5
19 %Vettori di appoggio
20 radici=ones(4,1);
21 iterazioni=ones(4,1);
22
23 %Metodo Secanti
24 disp("Metodo Secanti");
25 for i=1:length(toll)
26     [x_sec,it_sec]=sec(x0,x1,f,toll(i),1000);
27     radici(i)=x_sec;
28     iterazioni(i)=it_sec;
29 end
30 t=table(radici,iterazioni,toll,'VariableNames',{'Radici','Iterazioni','Tolleranza'
    });
31 disp(t);
32
33 %Metodo Bisezione
34 disp("Metodo Bisezione");
35 for i=1:length(toll)
36     [x_b,it_b]=bise(a,b,f,toll(i));
37     radici(i)=x_b;
38     iterazioni(i)=it_b;
39 end
40 t=table(radici,iterazioni,toll,'VariableNames',{'Radici','Iterazioni','Tolleranza'
    });
41 disp(t);
42
43 %Metodo Newton
44 disp("Metodo Newton");
45 for i=1:length(toll)
46     try
47         [x_n, it_n]=newtonZeri(x0,f,f1,toll(i),1000);
48         radici(i)=x_n;
49         iterazioni(i)=it_n;
50     catch ME
51         % Stampa il messaggio di errore
52         disp(['Errore: ', ME.message]);
53         radici(i) = NaN;
54         iterazioni(i) = NaN;
55     end
56 end
57 t=table(radici,iterazioni,toll,'VariableNames',{'Radici','Iterazioni','Tolleranza'
    });
58 disp(t);
59
60 %Metodo Newton Modificato
61 disp("Metodo Newton Modificato");
62 for i=1:length(toll)
63     try

```

```

64         [x_nm,it_nm]=newtonModificato(x0,f,f1,5,toll(i),1000);
65         radici(i)=x_nm;
66         iterazioni(i)=it_nm;
67     catch ME
68         % Stampa il messaggio di errore
69         disp(['Errore: ', ME.message]);
70         radici(i) = NaN;
71         iterazioni(i) = NaN;
72     end
73 end
74 t=table(radici,iterazioni,toll,'VariableNames',{'Radici','Iterazioni','Tolleranza'
75         });
76 disp(t);

```

Listing 7: Codice per ottenere i risultati

Per la tolleranza= $10^{-3}$

Metodo	Radici	Iterazioni
<i>secanti</i>	0.005576	33
<i>bisezione</i>	0.0375	3
<i>newton</i>	0.0039218	25
<i>newton modificato</i>	<i>NaN</i>	<i>NaN</i>

Per la tolleranza= $10^{-6}$

Metodo	Radici	Iterazioni
<i>secanti</i>	-0.0010403	61
<i>bisezione</i>	0.003125	5
<i>newton</i>	<i>NaN</i>	<i>NaN</i>
<i>newton modificato</i>	<i>NaN</i>	<i>NaN</i>

Per la tolleranza= $10^{-9}$

Metodo	Radici	Iterazioni
<i>secanti</i>	-0.001075	89
<i>bisezione</i>	0.0011163	31
<i>newton</i>	<i>NaN</i>	<i>NaN</i>
<i>newton modificato</i>	<i>NaN</i>	<i>NaN</i>

Per la tolleranza= $10^{-12}$

Metodo	Radici	Iterazioni
<i>secanti</i>	-0.0010751	123
<i>bisezione</i>	0.0011163	32
<i>newton</i>	<i>NaN</i>	<i>NaN</i>
<i>newton modificato</i>	<i>NaN</i>	<i>NaN</i>

- **Costo computazionale:** Il metodo di Bisezione presenta il costo computazionale più elevato tra i metodi considerati. A causa del suo approccio di divisione dell'intervallo a metà, richiede un numero maggiore di iterazioni per garantire la convergenza. Il metodo di Newton è più efficiente del primo solo nel caso della tolleranza  $\text{tol} = 10^{-3}$ ; negli altri casi mostra un errore, poiché la derivata prima è uguale a zero. Infine, il metodo di Newton modificato, non ha prodotto risultati validi quando la derivata della funzione si è annullata vicino alla radice.
- **Accuratezza:** Tutti i metodi utilizzano lo stesso criterio di arresto, ma la soluzione varia a seconda del metodo utilizzato. Il metodo delle secanti trova valori diversi a seconda della tolleranza, il numero di iterazioni aumenta con la tolleranza, ma il metodo converge. Il metodo di bisezione ha un numero di iterazioni generalmente basso ma anche qui, il metodo converge. Il metodo di Newton non trova valori per le ultime tre tolleranze. Molto efficiente ma non converge negli ultimi tre casi. Il metodo di Newton non riporta nessun dato utile.

## 8 Esercizio 8

Scrivere una *function* Matlab, **function x= mialu(A,b)** che, data in ingresso una matrice A ed un vettore b, calcoli la soluzione del sistema lineare  $Ax=b$  con il metodo di fattorizzazione LU con *pivoting* parziale. Curare particolarmente la scrittura e l'efficienza della *function*, e valutarla su un congruo numero di esempi significativi, che evidenzino tutti i suoi possibili *output*.

### 8.1 Soluzione

```
1 function x = mialu(A, b)
2     % x=mialu(A,b) Metodo di fattorizzazione LU con pivoting parziale
3     % Input: -A: matrice nxn
4     %         -b: vettore dei termini noti
5     % Output: -x: soluzione del sistema Ax=b
6
7     %vari controlli
8     [m,n]=size(A);
9     if m~=n, error('La matrice deve essere quadrata'), end;
10    if length(b)~=n, error('Dimensione del vettore b sbagliata'), end;
11
12    for i=1:n-1
13        if A(i,i) == 0
14            error('Matrice singolare');
15        end
16        A(i+1:n,i) = A(i+1:n,i) / A(i,i);
17        A(i+1:n,i+1:n) = A(i+1:n,i+1:n) - A(i+1:n,i) * A(i,i+1:n);
18    end
19
20    x = b(:);
21    % Risoluzione per L
22    for i=2:n
23        x(i:n) = x(i:n) - A(i:n,i-1) * x(i-1);
24    end
25
26    % Risoluzione per U
27    for i=n:-1:1
28        x(i) = x(i) / A(i,i);
29        x(1:i-1) = x(1:i-1) - A(1:i-1,i) * x(i);
30    end
31
32 end
```

Listing 8: Codice del metodo mialu

Per testare il codice abbiamo creato una codice tester che prendesse in ingresso una matrice e un vettore e andasse a controllare se il valore restituito dalla funzione mialu(A,b) fosse corretto.

1. nel primo esempio abbiamo la matrice  $A = \begin{bmatrix} 4 & 3 & -1 \\ 2 & -1 & 1 \\ 1 & -1 & 2 \end{bmatrix}$  e il vettore  $b = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$  sappiamo che  $x = A^{-1}b$  che si può ottenere tramite l'operatore '/' in matlab.

Il risultato quindi viene  $x = \begin{bmatrix} 0.4167 \\ 0.2500 \\ 1.4167 \end{bmatrix}$

2. nel secondo esempio abbiamo preso come matrice  $A = \begin{bmatrix} 4 & 1 & 2 \\ 3 & 5 & 1 \\ 1 & 1 & 3 \end{bmatrix}$  e il vettore  $b = \begin{bmatrix} 7 \\ 14 \\ 8 \\ 40 \end{bmatrix}$

Il programma riconosce che il vettore  $b$  è di dimensione sbagliata e segnala l'errore.

3. nel terzo esempio abbiamo preso come matrice  $A = \begin{bmatrix} 2 & 4 & 6 \\ 1 & 1 & 3 \\ 3 & 6 & 9 \end{bmatrix}$  e vettore  $b = \begin{bmatrix} 7 \\ 14 \\ 8 \end{bmatrix}$ , il programma riconosce che la matrice  $A$  è una matrice singolare e segnala l'errore.

4. nel quarto esempio abbiamo preso come matrice  $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$  e come vettore  $b = \begin{bmatrix} 8 \\ 12 \\ 3 \end{bmatrix}$  il programma riconosce che la matrice  $A$  non è una matrice quadrata e segnala l'errore.

## 9 Esercizio 9

Scrivere una *function* Matlab, **function** `x=mialdl(A,b)` che, dati in ingresso una matrice  $sdp$   $A$  ed un vettore  $b$ , calcoli la soluzione del corrispondente sistema lineare utilizzando la fattorizzazione  $LDL^T$ . Curare particolarmente la scrittura e l'efficienza della *function*, e validarla su un congruo numero di esempi significativi, che evidenzino tutti i suoi possibili *output*.

### 9.1 Soluzione

```

1 function x = mialdl(A,b)
2 %
3 % x = mialdl(A,b)
4 %
5 % Dati in ingresso una matrice sdp A ed un vettore b, calcolare la
6 % soluzione del corrispondente sistema lineare utilizzando la
7 % fattorizzazione LDL^t
8 %
9 % input: - A: matrice dei coefficienti
10 %         - b: vettore dei termini noti
11 %
12 % output: x - soluzione del sistema Ax=b
13 %
14
15 [m,n] = size(A);
16 if m~=n
17 error('La matrice A deve essere quadrata. Verifica le dimensioni di A');
18 end
19 if m~=length(b)
20 error('Le dimensioni della matrice A e del vettore b non sono compatibili');
21 end
22 if A(1,1)<=0
23 error('La matrice A deve essere simmetrica e definita positiva');
24 end
25
26 % Fattorizzazione LDL^t
27 A(2:n,1)=A(2:n,1)/A(1,1);
28 for i=2:n
29 v= (A(i,1:i-1).')*. diag(A(1:i-1,1:i-1));
30 A(i,i) = A(i,i) - A(i,1:i-1)*v;
31 if A(i,i)<=0
32 error('La matrice A deve essere simmetrica e definita positiva');
33 end
34 A(i+1:n,i) = (A(i+1:n,i)- A(i+1:n,i:i-1)*v)/A(i,i);
35 end
36

```

```

37 %Risoluzione del sistema lineare
38 x=b;
39 for i=1:n
40 x(i+1:n) = x(i+1:n)-(A(i+1:n,i)*x(i));
41 end
42 x = x ./ diag(A);
43 for i=n:-1:2
44 x(1:i-1) = x(1:i-1)-A(i,1:i-1).'*x(i);
45 end
46 end

```

Listing 9: Codice del metodo mialdl

Per testare la funzione abbiamo creato un test che prendesse in input una matrice  $A$  e un vettore  $b$  e che richiamando la funzione `mialdl(A,b)` testasse i risultati ritornati.

1. Nel primo esempio abbiamo la matrice  $A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}$  e il vettore  $b = \begin{bmatrix} 7 \\ 14 \\ 8 \end{bmatrix}$

Il risultato quindi viene  $x = \begin{bmatrix} 14.2500 \\ 21.5000 \\ 14.7500 \end{bmatrix}$

2. Nel secondo esempio abbiamo la matrice  $A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}$  e il vettore  $b = \begin{bmatrix} 7 \\ 14 \\ 8 \\ 9 \end{bmatrix}$

Il programma riconosce che il vettore  $b$  è di dimensione sbagliata e segnala l'errore.

3. Nel terzo esempio abbiamo la matrice  $A = \begin{bmatrix} 2 & 4 & 6 \\ 1 & 1 & 3 \\ 3 & 6 & 9 \end{bmatrix}$  e il vettore  $b = \begin{bmatrix} 7 \\ 14 \\ 8 \end{bmatrix}$

Il programma riconosce che  $A$  non è sdp e segnala l'errore.

4. Nel quarto esempio abbiamo la matrice  $A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \end{bmatrix}$  e il vettore  $b = \begin{bmatrix} 7 \\ 14 \end{bmatrix}$

Il programma riconosce che  $A$  non è una matrice quadrata e segnala l'errore.

## 10 Esercizio 10

Scrivere una *function* Matlab, **function** `[x,nr]=miaqr(A,b)` che, data in ingresso la matrice  $A$   $m \times n$ , con  $m \geq n = \text{rank}(A)$ , ed un vettore  $b$  di lunghezza  $m$ , calcoli la soluzione del sistema lineare  $Ax=b$  nel senso dei minimi quadrati e, inoltre, la norma,  $nr$ , del corrispondente vettore residuo. Curare particolarmente la scrittura e l'efficienza della *function*, e validarla su un congruo numero di esempi significativi, che evidenzino tutti i suoi possibili *output*.

### 10.1 Soluzione

```

1 function [x,nr]=miaqr(A,b)
2 %
3 % [x,nr]=miaqr(A,b)
4 % Esegue la fattorizzazione QR di A e restituisce la soluzione
5 % ai minimi quadrati del sistema lineare e la
6 % norma del corrispondente vettore residuo
7 %
8 % Input: A- matrice mxn

```

```

9 %          b- vettore dei termini noti
10 % Output: x-soluzione del sistema Ax=b
11 %          nr- norma del vettore residuo
12 [m,n]=size(A);
13 for i=1:n
14     alfa=norm(A(i:m,i));
15     if alfa==0
16         error('Matrice non a rango massimo');
17     end
18     if length(b)~=m, error('Dimensione del vettore b sbagliata'), end;
19     if A(i,i)>=0
20         alfa=-alfa;
21     end
22     v1=A(i,i)-alfa;
23     A(i,i)=alfa;
24     A(i+1:m,i)=A(i+1:m,i)/v1;
25     beta=-v1/alfa;
26     A(i:m,i+1:n)=A(i:m,i+1:n)-(beta*[1;A(i+1:m,i)])*...
27         ([1 A(i+1:m,i)']*A(i:m,i+1:n));
28     b(i:m)=b(i:m)-(beta*[1 A(i+1:m,i)']*b(i:m))*...
29         [1;A(i+1:m,i)]);
30 end
31 %risoluzione sistema Ax=b
32 x=b(:);
33 for i=n:-1:1
34     x(i)=x(i)/A(i,i);
35     x(1:i-1)=x(1:i-1)-A(1:i-1,i)*x(i);
36 end
37 %norma del vettore residuo
38 nr=norm(x(n+1:m));
39
40 end

```

Listing 10: Codice del metodo miaqr

Per testare la funzione abbiamo creato un testr che prendesse in input una matrice  $A$  e un vettore  $b$  e che richiamando la funzione miaqr testasse i risultati ritornati.

1. Nel primo esempio abbiamo preso come matrice  $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$  e come vettore dei termini noti

$$b = \begin{bmatrix} 10 \\ 11 \\ 12 \end{bmatrix} \text{ sappiamo che } x = A^{-1}b \text{ che si può ottenere tramite l'operatore ' / ' in matlab.}$$

Il risultato quindi viene  $x = \begin{bmatrix} -3.3333 \\ -2.3333 \\ 6.0000 \end{bmatrix}$  Mentre la norma euclidea del vettore residuo viene 0.

2. Nel secondo esempio abbiamo preso come matrice  $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$  e come vettore dei termini noti

$$b = \begin{bmatrix} 10 \\ 11 \\ 12 \\ 13 \end{bmatrix} \text{ Il programma riconosce che la dimensione del vettore } b \text{ è sbagliata e segnala l'errore.}$$

3. Nel terzo esempio abbiamo preso come matrice  $A = \begin{bmatrix} 1 & 2 & 3 & 9 \\ 4 & 5 & 6 & 9 \\ 7 & 8 & 9 & 9 \end{bmatrix}$  e come vettore dei termini noti

$b = \begin{bmatrix} 10 \\ 11 \\ 12 \end{bmatrix}$  Il programma riconosce che la matrice non è a rango massimo e segnala l'errore.

## 11 Esercizio 11

Risolvere i sistemi lineari, di dimensione  $n$ ,

$$A_n x_n = b_n, \quad n = 1, \dots, 15$$

in cui

$$A_n = \begin{pmatrix} 1 & 1 & \dots & \dots & 1 \\ 10 & \ddots & \ddots & & \vdots \\ 10^2 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 1 & 1 \\ 10^{n-1} & \dots & 10^2 & 10 & 1 \end{pmatrix} \in \mathbb{R}^{n \times n}, \quad \mathbf{b}_n = \begin{pmatrix} n-1 + \frac{10^1-1}{9} \\ n-2 + \frac{10^2-1}{9} \\ n-3 + \frac{10^3-1}{9} \\ \vdots \\ 0 + \frac{10^n-1}{9} \end{pmatrix} \in \mathbb{R}^n,$$

la cui soluzione è il vettore  $\mathbf{x}_n = (1, \dots, 1)^T \in \mathbb{R}^n$ , utilizzando la *function* `mialu`. Tabulare e commentare l'accuratezza dei risultati ottenuti, dandone spiegazione esaustiva.

### 11.1 Soluzione

Dimensione	Condizionamento	Soluzione
1	1.0000e+00	1.0000
2	1.1356e+01	1.0000
3	1.9106e+02	1.0000
4	2.1679e+03	1.0000
5	2.2819e+04	1.0000
6	2.3418e+05	1.0000
7	2.3771e+06	1.0000
8	2.3995e+07	1.0000
9	2.4147e+08	1.0000
10	2.4254e+09	1.0000
11	2.4332e+10	1.0000
12	2.4391e+11	1.0000
13	2.4437e+12	1.0000
14	2.4473e+13	1.0000
15	2.4501e+14	1.0000

Il numero di condizionamento della matrice aumenta rapidamente con  $n$ , come è evidenziato dai valori crescenti nella tabella. Questo indica che le matrici diventano sempre più mal condizionate man mano che la dimensione  $n$  aumenta. Per tutte le dimensioni da  $n = 1$  a  $n = 15$ , la soluzione calcolata è esattamente 1.0000. Nonostante il crescente condizionamento, la *function* `mialu` implementata mantiene l'accuratezza nella soluzione del sistema per tutte le dimensioni testate.

## 12 Esercizio 12

Fattorizzare, utilizzando la *function* `mialdlt`, le matrici `sdp`

$$A_n = \begin{pmatrix} n & -1 & \dots & -1 \\ -1 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & -1 \\ -1 & \dots & -1 & n \end{pmatrix} \in \mathbb{R}^{n \times n}, \quad n = 1, \dots, 100.$$



Graficare, in un unico grafico, gli elementi diagonali del fattore  $D$ , rispetto all'indice diagonale.

## 12.1 Soluzione

```

1  valori_n = 1:100;
2  max=length(valori_n);
3  for k = 1:max
4      n = valori_n(k);
5      A = ones(n);
6      %eseguo 2 for annidati per poter scorrere tutta la matrice A
7      for i = 1:n
8          for j = 1:n
9              if (i ~= j)
10                 A(i,j) = -A(i,j);
11             else
12                 A(i,j) = n;
13             end
14         end
15     end
16 end
17 %con questo for modifico la matrice in modo da avere gli elementi diagonali
18 %con n e gli elementi che non sono diagonali li nego
19 A = mialdlt(A);
20 diagonal_elements = diag(A);
21
22 if n==1||mod(n,1)==0
23     plot(1:n, diagonal_elements, '.');
24     hold on;
25 end
26 end
27
28 title('Esercizio 12');
29 xlabel('Indice dellla diagonale');
30 ylabel('Valore dei fattori di D');
```

Listing 11: Codice del metodo per graficare gli elementi diagonali di  $D$

```

1  function A = mialdlt(A)
2  %
3  % A = mialdlt(A)
4  %
5  % Data in input una matrice A sdp restituisce A contenente gli elementi L
6  % e D
7
8  [m,n] = size(A);
9  if A(1,1)<=0
10     error('Errore: La matrice deve essere sdp');
11 end
12 if m~=n
13     error("Errore: La matrice deve essere quadrata");
14 end
15
16 A(2:n,1) = A(2:n,1)/A(1,1);
17
18 for j=2:n
19     v = (A(j,1:j-1)'.')*.diag(A(1:j-1,1:j-1));
20     A(j,j) = A(j,j)-A(j,1:j-1)*v;
21     if A(j,j)<=0,error('La matrice deve essere sdp'),end;
```

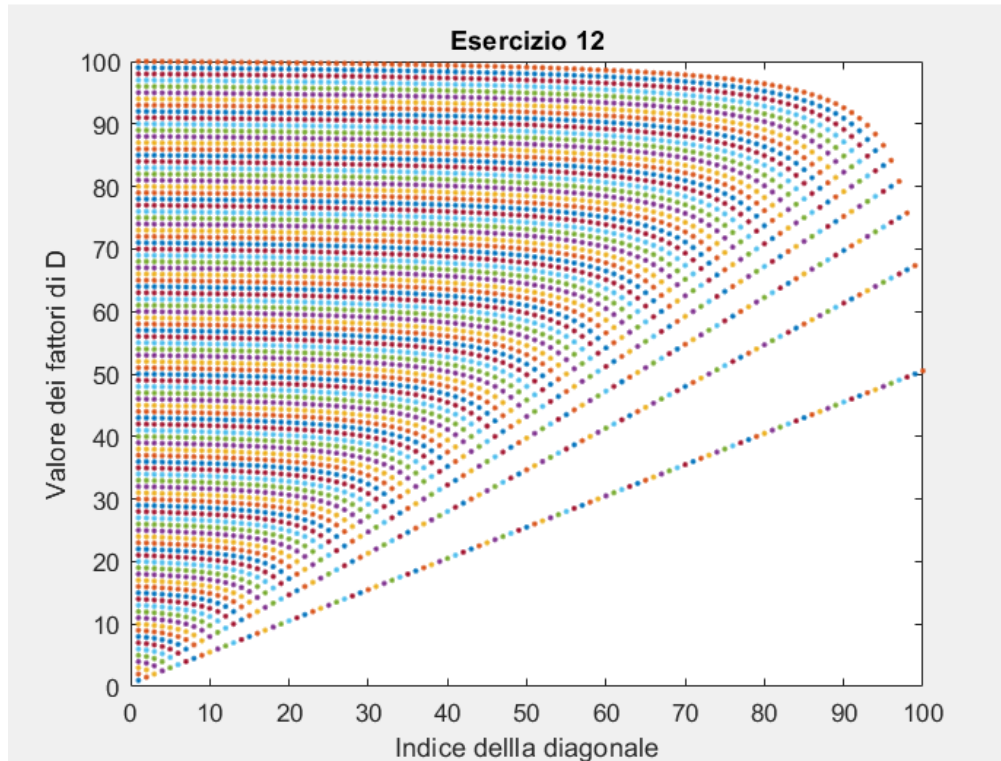
```

22     A(j+1:n,j) = (A(j+1:n,j)-A(j+1:n,1:j-1)*v)/A(j,j);
23 end
24
25 return

```

Listing 12: Codice del mialdt usato nel codice precedente

Questo è il grafico che ne risulta:



Dal grafico si può notare come gli elementi diagonali tendano a crescere man mano che ci si sposta verso l'alto lungo l'asse y. Questo comportamento è tipico nelle matrici sdp, dove, per la definizione di matrice simmetrica definita positiva, gli elementi della diagonale sono maggiori degli altri elementi della matrice.

## 13 Esercizio 13

Utilizzare la *function*, `miaqr` per risolvere, nel senso dei minimi quadrati, il sistema lineare sovradeterminato

$$Ax = b$$

in cui

$$A = \begin{pmatrix} 7 & 2 & 1 \\ 8 & 7 & 8 \\ 7 & 0 & 7 \\ 4 & 3 & 3 \\ 7 & 0 & 10 \end{pmatrix}, b = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{pmatrix},$$

dove viene minimizzata la seguente norma *pesata* del residuo  $\mathbf{r}=(r_1, \dots, r_5)^T$

$$p_w^2 := \sum_{i=1}^5 w_i r_i^2,$$

con

$$w_1 = w_2 = 0.5, w_3 = 0.75, w_4 = w_5 = 0.25$$

Dettagliare l'intero procedimento, calcolando, in uscita, anche  $p_w$ .

## 13.1 Soluzione

```
1 % Inizializzazione dei dati
2 A = [7 2 1; 8 7 8; 7 0 7; 4 3 3; 7 0 10];
3 b = [1; 2; 3; 4; 5];
4
5 % Vettore dei pesi
6 omega = [0.5; 0.5; 0.75; 0.25; 0.25];
7
8 % Normalizzazione dei pesi in modo che la somma sia 1
9 omega = omega ./ 2.25;
10
11 % Calcolo della matrice dei pesi B
12 B = diag(sqrt(omega));
13
14 % Risoluzione del sistema utilizzando QR
15 [x, nr] = miaqr(B*A, B*b);
16
17 disp("Soluzione trovata per il sistema sovradeterminato:");
18 disp(x);
19
20 disp("Norma del vettore residuo:");
21 disp(nr);
```

Listing 13: Codice che risolve il sistema lineare sovra-determinato

Utilizzando i pesi forniti, il nostro obbiettivo è minimizzare la norma pesata, che ci porta a risolvere il sistema sovra-determinato  $BAx = Bb$ , dove:

- $B$  è una matrice diagonale con le radici dei pesi come elementi
- $A$  è la matrice dei coefficienti
- $b$  è il vettore dei termini noti
- $x$  è il vettore soluzione del sistema

Una volta normalizzato i pesi, procediamo con la risoluzione del sistema mediante l'utilizzo della funzione `miaqr(BA, Bb)`, ottenendo come segue:

Soluzione trovata per il sistema sovradeterminato:

0.1531  
-0.1660  
0.3185  
1.0146  
0.3160

Norma del vettore residuo:

1.0627

## 14 Esercizio 14

Scrivere una *function* Matlab, `[x,nit]= newton(fun,x0,tol,maxit)` che implementi efficientemente il metodo di Newton per risolvere sistemi di equazioni nonlineari. Curare particolarmente il criterio di arresto. La seconda variabile, se specificata, ritorna il numero di iterazioni eseguite. Prevedere opportuni valori di *default* per gli ultimi due parametri in ingresso (rispettivamente, la tolleranza per il criterio di arresto, ed il massimo numero di iterazioni). La function `fun` deve avere sintassi: `[f,lacobian]=fun(x)`, se il sistema da risolvere è  $f(x)=0$ .

## 14.1 Soluzione

```
1
2 function [x, nit] = newton(fun,x0, tol, maxit)
3 %
4 % [x, nit] = newton(fun,x0, jacobian, tol, maxit)
5 %
6 % Metodo di newton per la risoluzione di sistemi di equazioni non lineari
7 % tramite il metodo di newton
8 %
9 % Input: fun-sistema di equazioni
10 %      x0- vettore valori iniziali
11 %      jacobian- matrice jacobiana del sistema
12 %      tol- tolleranza
13 %      maxit- numero massimo di iterazioni
14 %
15 % Output:X-soluzione del sistema
16 %      nit- numero di iterazioni eseguite
17 %Criterio d'arresto:  $|X_{n+1} - X_n| \leq \text{tol} * (1 + |X_n|)$ 
18 % Controlli di consistenza
19     if tol < 0
20         error('Tolleranza non valida');
21     end
22
23     %Valori di default per i parametri in ingresso
24     if nargin == 2
25         tol = 1e-3;
26         maxit = 1000;
27     else if nargin == 3
28         maxit = 1000;
29     else if nargin<2
30         error('numero di input errati');
31     end;
32     if maxit <= 0
33         error('Numero di iterazioni non valido');
34     end
35     x = x0;
36     for i=1:maxit
37         x0 = x;
38         [f,jacobiana]=feval(fun,x0);
39         x = x0 + mialu(jacobiana, -f); % Fattorizzazione e aggiornamento di xn+1
40         % controllo sul criterio di arresto
41         if abs(x - x0) <= tol * (1 + abs(x0))
42
43             break;
44         end
45     end
46     nit = i;
47     if abs(x - x0) > tol * (1 + abs(x0))
48         disp('Tolleranza non raggiunta');
49     end
50     return
51 end
52 end
```

Listing 14: Codice del metodo di newton per risolvere sistemi di equazioni non lineari

```
1 function [f,jacobian] = fun(x)
2
```

```

3 F = eye(50) * 4;
4 for i = 1:49
5     F(i, i+1) = 1;
6     F(i+1, i) = 1;
7 end
8 e = ones(50,1);
9 a = 2;
10 b = -1.1;
11 grad = @(x) F * x - a * e .* sin(a * x) - b * e .* exp(-x);
12 jacob = @(x) F - a^2 * diag(e .* cos(a * x)) + b * diag(e .* exp(-x));
13
14 f=grad(x);
15 jacobian=jacob(x);
16
17 end

```

Listing 15: Codice della funzione fun richiamata in newton

Il codice della funzione mialu si può trovare nell'esercizio 8

## 15 Esercizio 15

Usare la *function* del precedente esercizio per risolvere, a partire dal vettore iniziale nullo, il sistema nonlinear derivante dalla determinazione del punto stazionario della funzione:

$$f(x) = \frac{1}{2}x^T Qx + e^T [\cos(\alpha x) + \beta \exp(-x)], \quad e = (1, \dots, 1)^T \in \mathbb{R}^{50},$$

$$Q = \begin{pmatrix} 4 & 1 & & & \\ 1 & \ddots & \ddots & & \\ & \ddots & \ddots & 1 & \\ & & & 1 & 4 \end{pmatrix} \in \mathbb{R}^{50 \times 50}, \quad \alpha = 2, \quad \beta = -1.1,$$

utilizzando tolleranze `tol = 1e-3, 1e-8, 1e-13` (le *function* `cos` e `exp` sono da intendersi in modo vettoriale). Graficare la soluzione e tabulare in modo conveniente i risultati ottenuti.

### 15.1 Soluzione

Di seguito è riportato il codice usato per graficare la soluzione

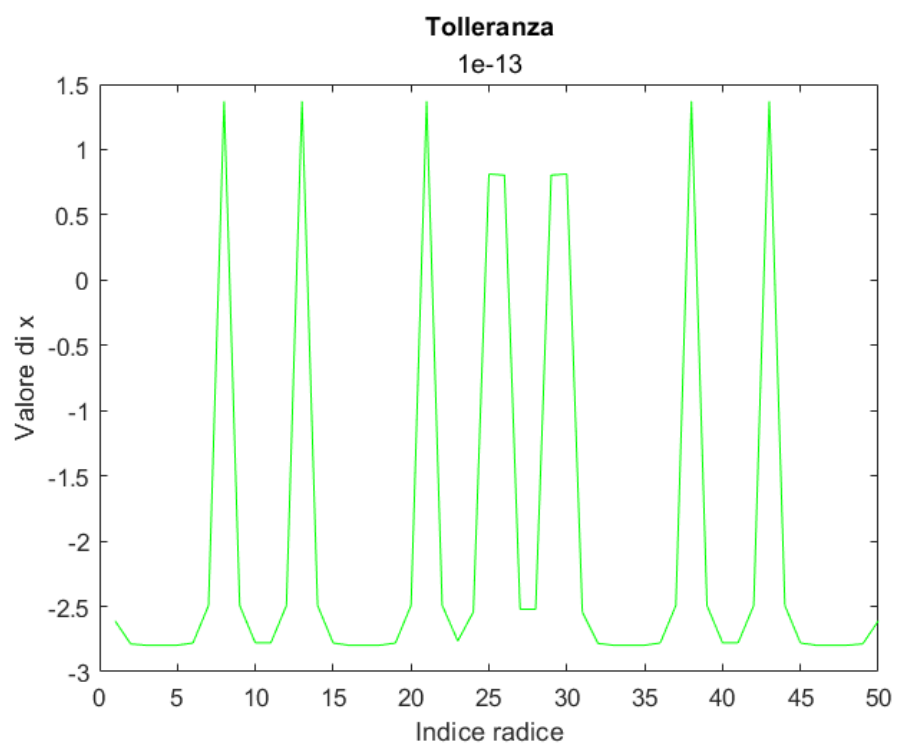
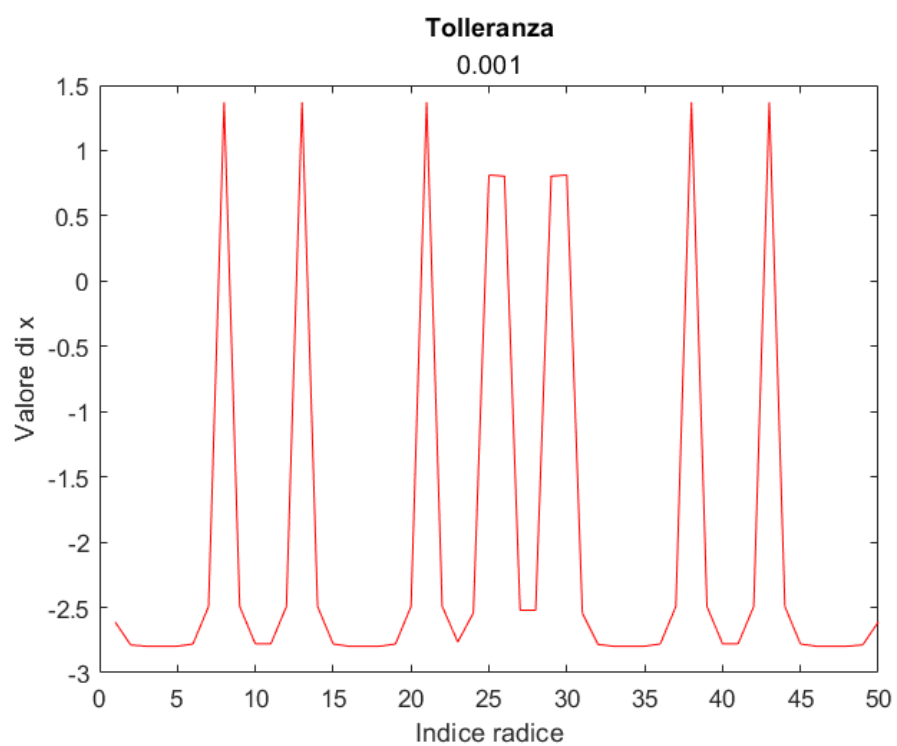
```

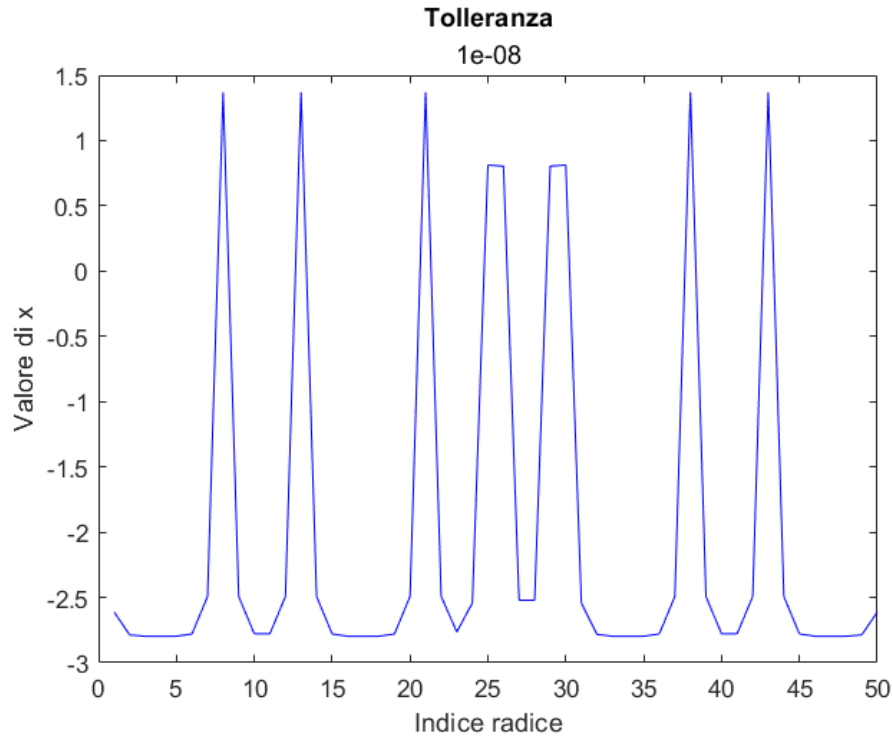
1 format long
2 n = 50;
3 x0 = zeros(n,1);
4 toll = [1e-3, 1e-8, 1e-13];
5 c = ['r', 'b', 'g'];
6 L = zeros(50, 3);
7 for i = 1:length(toll)
8     [x,nit] = newton(@fun, x0, toll(i),1000);
9     figure;
10    plot(1:n, x, '-', Color=c(i));
11    title('Tolleranza', num2str(toll(i)));
12    xlabel('Indice radice');
13    ylabel('Valore di x');
14    L(:,i) = x;
15 end
16 disp(L);

```

Listing 16: Codice per graficare soluzione

I rispettivi grafici sono:





## 16 Esercizio 16

Costruire una function **lagrange.m** avente la stessa sintassi della function **spline** di Matlab, che implementi, in modo vettoriale la forma di Lagrange del polinomio interpolante una funzione.

**N.B.:** il risultato dovrà avere le stesse dimensioni del dato in ingresso

### 16.1 Soluzione

```

1 function p = lagrange(X,Y,XQ)
2 % p = lagrange(X,Y,XQ)
3 %
4 % function che implementa la forma di lagrange del polinomio
5 % interpolante una funzione
6 %
7 % input:X- vettore delle coordinate x
8 %       Y- valori della funzione alle coordinate x
9 %       XQ- vettore dei punti in cui calcolare il polinomio
10 %
11 % output:p- polinomio interpolante in forma di Lagrange
12 %
13 n= length(X);
14 if length(Y)~= n
15     error('Le dimensioni di X e Y non sono corrette');
16 end
17 if length(unique(X))~= n
18     error('Presenza di ascisse uguali');
19 end
20 if nargin<3
21     error('numero paramentri errato');
22 end

```

```

23 p= zeros(size(XQ));
24 for i= 1:n
25     L= ones(size(XQ));
26     for j= 1:n
27         if j~= i
28             L= L .* (XQ - X(j)) / (X(i) - X(j));
29         end
30     end
31     p= p + Y(i) * L;
32 end
33 end

```

Listing 17: Codice della funzione per il calcolo del polinomio interpolante di lagrange

## 17 Esercizio 17

Costruire una function, `newton.m`, avente la stessa sintassi della function `spline` di Matlab, che implementi, in modo vettoriale, la forma di Newton del polinomio interpolante una funzione.

**N.B.:** il risultato dovrà avere le stesse dimensioni del dato in ingresso

### 17.1 Soluzione

```

1 function pn = newton(X, Y, XQ)
2 % pn = newton(X, Y, XQ)
3 %
4 % function che implementa la forma di Newton del polinomio
5 % interpolante una funzione
6 %
7 % Input: X - vettore delle ascisse
8 %         Y - vettore dei valori della funzione
9 %         XQ - vettore dei punti in cui calcolare il polinomio
10 %
11 % Output: pn - polinomio interpolante in forma di Newton
12 %
13 if length(X) ~= length(Y) || length(X) <= 0
14     error('Dati non corretti');
15 end
16
17 % Verifica che i valori in X siano distinti
18 if length(unique(X)) ~= length(X)
19     error('I valori di X devono essere distinti');
20 end
21
22 % Calcola le differenze divise usando la function differenzeDivise
23 dd = differenzeDivise(X, Y);
24 n = length(dd) - 1;
25
26 pn = dd(n+1) * ones(size(XQ));
27
28 % Costruisci il polinomio interpolante di Newton
29 for i = n:-1:1
30     pn = pn.*(XQ- X(i)) + dd(i);
31 end
32 return
33 end

```

Listing 18: Codice della funzione per il calcolo del polinomio interpolante di Newton



```

1 function dd = differenzeDivise(x, f)
2 % dd = differenzaDivise(x, f)
3 %
4 % function che calcola le differenze divise in (x_i, f_i)
5 %
6 % Input: x - vettore delle ascisse
7 %         f - vettore delle ordinate
8 %
9 % Output: dd - vettore delle differenze divise
10 %
11 % Numero di dati nel vettore x
12 k = length(x);
13 if length(f) ~= k
14     error('I vettori x e f devono avere la stessa dimensione');
15 end
16
17 k = k - 1;
18
19 % Inizializza il vettore delle differenze divise con i valori di f
20 dd = f;
21
22 % Calcola le differenze divise
23 for j = 1:k
24     for i = k+1:-1:j+1
25         % Formula per calcolare le differenze divise
26         dd(i) = (dd(i) - dd(i-1)) / (x(i) - x(i-j));
27     end
28 end

```

Listing 19: Codice della funzione per il calcolo delle differenze divise

## 18 Esercizio 18

Costruire una function `hermite.m`, avente sintassi

**`yy = hermite( xi, fi, fli, xx)`**

che implementi in modo vettoriale il polinomio interpolante di Hermite.

**N.B.:** il risultato dovrà avere le stesse dimensioni del dato in ingresso

### 18.1 Soluzione

```

1 function yy = hermite(xi,fi,fli,xx)
2 % yy = hermite(xi,fi,fli,xx)
3 % funzione che implementa la forma di hermite del polinomio interpolante una
4     funzione
5 %
6 % input:xi- vettore delle coordinate x
7 %         fi- valori della funzione alle coordinate x
8 %         fli- valori delle derivate della funzione nei punti x
9 %         xx- vettore dei punti in cui il polinomio interpolante va calcolato
10 %
11 % output:yy- polinomio interpolante in forma di Newton
12
13 n= length(xi);
14 if n~= length(fi) ||length(fli)~= n || n<= 0

```

```

14     error("Dimensioni dei dati in input errate");
15 end
16 if length(unique(xi))~=n
17 error("Valori delle ascisse non distinti tra di loro");
18 end
19 fi = repelem (fi,2);
20 for i= 1: length(fli)
21     fi(i*2) = fli(i);
22 end
23 %calcolo delle differenze divise
24 n=length(xi)-1;
25 for i=(2*n+1): -2: 3
26     fi(i)= (fi(i)- fi(i-2))/(xi((i+1)/2) - xi((i-1)/2));
27 end
28 for i= 2: 2*n+1
29     for i=(2*n+2): -1: i+1
30         fi(i)=(fi(i)- fi(i-1)) / ( xi(round(i/2))-xi(round((i-i)/2)));
31     end
32 end
33 %calcolo polinomio interpolante
34 n=length(fi)-1;
35 yy= fi(n+1)*ones(size(xx));
36 for i = n : -1:1
37     yy= yy.*(xx- xi(round(i/2)))+fi(i) ;
38 end
39 return
40 end

```

Listing 20: Codice della funzione per il calcolo del polinomio interpolante di Hermite

## 19 Esercizio 19

Si consideri la seguente base di Newton,

$$w_i(x) = \prod_{j=0}^{i-1} (x - x_j), \quad i = 0, \dots, n,$$

con  $x_0, \dots, x_n$  ascisse date (non necessariamente distinte tra loro), ed un polinomio rappresentato rispetto a tale base,

$$p(x) = \sum_{i=0}^n a_i w_i(x)$$

Derivare una modifica dell'algoritmo di Horner per calcolarne efficientemente la derivata prima.

### 19.1 Soluzione

Consideriamo il polinomio generico:

$$P(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \dots + a_n(x - x_0)(x - x_1) \dots (x - x_{n-1})$$

Per calcolare la sua derivata, impostiamo i valori iniziali come segue:

$$P = a_n, \quad derivataP = 0$$

Successivamente, iteriamo i coefficienti dal penultimo al primo, aggiornando P e derivataP ad ogni passo:

$$derivataP = derivataP \cdot (x - x_k) + P$$

$$P = P \cdot (x - x_k) + a_k$$

Alla fine derivataP conterrà il valore della derivata del polinomio P nel punto  $x$ .

```

1 function derivataP = horner(ascisse, coeff, xi)
2 %
3 % horner(ascisse, coeff, xi)
4 % Determina la derivata di un polinomio in forma di Newton in un punto dato
5 %
6 % Input:
7 %   ascisse - Vettore delle ascisse [x0,x1,...,xn]
8 %   coeff - Vettore dei coefficienti [a0,a1,...,an]
9 %   xi - punto in cui calcolare la derivata
10 % Output:
11 %   derivataP - Derivata del polinomio calcolata in xi
12 %
13 n = length(coeff) - 1;
14 %Imposta i valori iniziali per il polinomio P e la sua derivata derivataP
15 P = coeff(n+1);
16 derivataP = 0;
17
18 for k = n:-1:1
19     derivataP = derivataP * (xi - ascisse(k)) + P;
20     P = P * (xi - ascisse(k)) + coeff(k);
21 end
22 end

```

Listing 21: Codice della modifica dell'algoritmo di Horner per calcolare la derivata prima

## 20 Esercizio 20

Utilizzando le function degli esercizi 18 e 19, calcolare il polinomio interpolante di Hermite la funzione  $f(x) = e^{\frac{x}{2}} + e^{-x}$  sulle ascisse equidistanti  $[0, 2.5, 5]$ . Graficare il grafico della funzione interpolanda e del polinomio interpolante nell'intervallo  $[0, 5]$ , e quello della derivata prima della funzione interpolanda, e della derivata prima del polinomio interpolante, verificando graficamente le condizioni di interpolazione per entrambi.

### 20.1 Soluzione

Di seguito è mostrato il codice usato per graficare le funzioni

```

1 ascisse=[0,2.5,5];
2 a=0;
3 b=5;
4 f = @(x) exp(x/2 + exp(-x));
5 df = @(x) exp(x/2 + exp(-x)) .* (1/2 - exp(-x));
6 d2f = @(x) exp(x/2 + exp(-x)) .* ((1/2 - exp(-x)).^2 + exp(-x));
7
8 %genero i valori di x nell'intervallo [a,b] e calcolo i valori delle
9 %funzioni
10 x = linspace(a,b, 10001);
11 y = f(x);
12 dy = df(x);
13 d2fi = d2f(ascisse);
14
15 fi=f(ascisse);
16 dfi=df(ascisse);
17 %calcolo l'interpolazioni

```

```

18 yy=hermite(ascisse,fi,dfi,x);
19
20 % Grafico di f(x)
21 figure;
22 plot(x, y, 'g', "LineWidth", 2);
23 hold on;
24 plot(x, yy, 'r-.', "LineWidth", 2);
25 set(gca, 'XMinorGrid', 'on');
26 set(gca, 'YMinorGrid', 'on');
27 %evidenzio i punti di interesse
28 scatter(ascisse, fi, 100, 'r', 'filled');
29 for i = 1:length(ascisse)
30     text(ascisse(i), fi(i), sprintf('%0.2f, %0.2f', ascisse(i), fi(i)), ...
31         'VerticalAlignment', 'bottom', 'HorizontalAlignment', 'right');
32 end
33 xlabel('x');
34 ylabel('f(x)');
35 title('Grafico di f(x) con interpolazione Hermite e punti interpolazione');
36 legend('f(x)', 'Interpolazione Hermite', 'Punti interpolazione');
37 grid on;
38 zoom on;
39
40 % Grafico della derivata prima
41 figure;
42 plot(x, dy, "g", "LineWidth", 2);
43 hold on;
44 set(gca, 'XMinorGrid', 'on');
45 set(gca, 'YMinorGrid', 'on');
46 % Calcolo delle differenze divise per Hermite
47 coeff = repelem(fi, 2);
48 for i = 1:length(dfi)
49     coeff(i*2) = dfi(i);
50 end
51 n = length(ascisse) - 1;
52 for j = (2*n+1):-2:3
53     coeff(j) = (coeff(j) - coeff(j-2)) / (ascisse((j+1)/2) - ascisse((j-1)/2));
54 end
55 for j = 2:2*n+1
56     for i = (2*n+2):-1:j+1
57         coeff(i) = (coeff(i) - coeff(i-1)) / (ascisse(round(i/2)) - ascisse(round((i-j)/2)));
58     end
59 end
60
61 ascHerm = repelem(ascisse, 2);
62
63 y1herm = horner(ascHerm, coeff, 0);
64 y2herm = horner(ascHerm, coeff, 2.5);
65 y3herm = horner(ascHerm, coeff, 5);
66 yherm = [y1herm, y2herm, y3herm];
67
68 fherm = hermite(ascisse, yherm, d2fi, x);
69
70 plot(x, fherm, "r-.", "LineWidth", 2);
71
72 % evidenzio i punti di interesse
73 scatter(ascisse, dfi, 100, 'r', 'filled');
74 for i = 1:length(ascisse)
75     text(ascisse(i), dfi(i), sprintf('%0.2f, %0.2f', ascisse(i), dfi(i)), ...

```

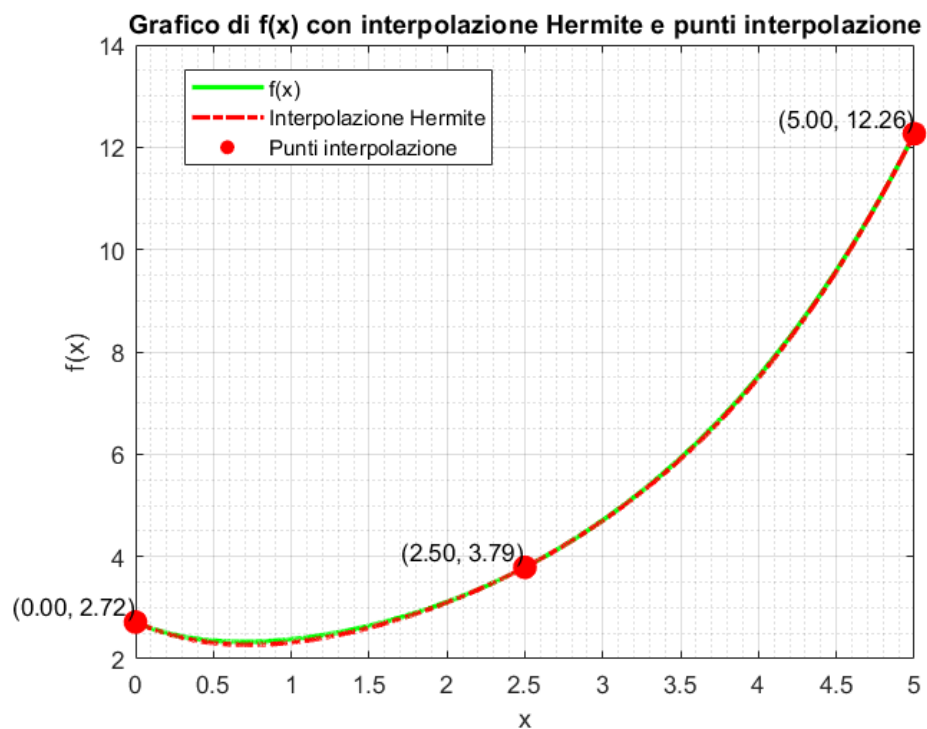
```

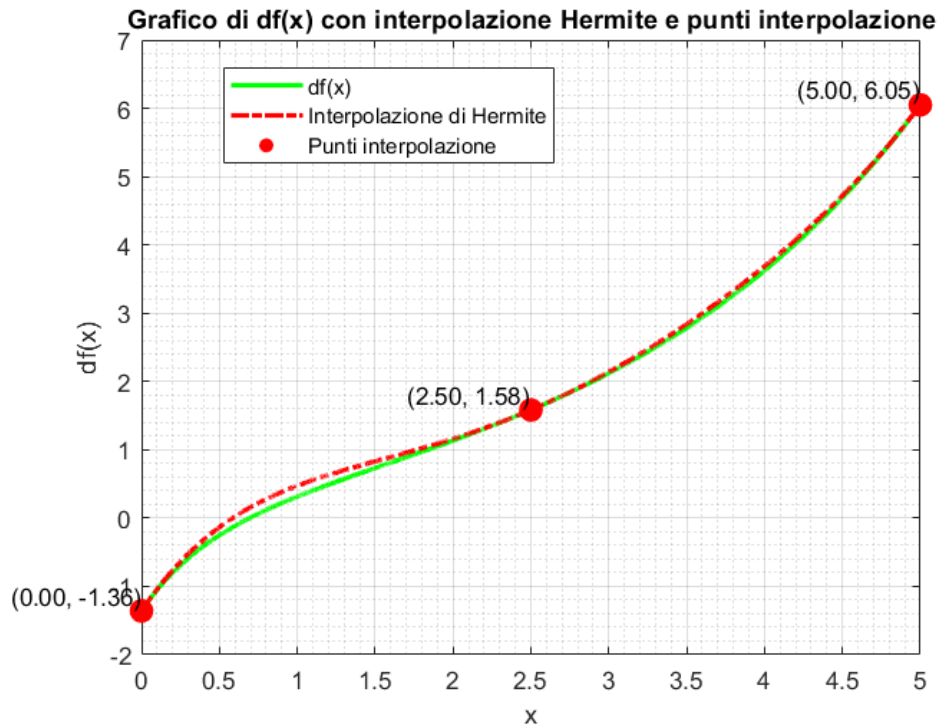
76         'VerticalAlignment', 'bottom', 'HorizontalAlignment', 'right');
77 end
78
79 xlabel('x');
80 ylabel('df(x)');
81 title('Grafico di df(x) con interpolazione Hermite e punti interpolazione');
82 legend('df(x)', 'Interpolazione di Hermite', 'Punti interpolazione');
83 grid on;
84 zoom on;

```

Listing 22: Codice per graficare  $f(x)$  e  $df(x)$

Di seguito sono riportati i grafici di, rispettivamente  $f(x)$  e  $df(x)$ :





## 21 Esercizio 21

Costruire una function Matlab che, specificato in ingresso il grado  $n$  del polinomio interpolante, e gli estremi dell'intervallo  $[a, b]$ , calcoli le corrispondenti ascisse di Chebyshev.

### 21.1 Soluzione

```

1 function x = chebyshev(n,a,b)
2 % x = chebyshev(n,a,b)
3 % funzione per il calcolo delle ascisse di Chebyshev
4 %
5 % input: n- grado del polinomio
6 %       [a,b]- estremi dell'intervallo
7 % output: x- vettore contenente le ascisse di chebyshev
8
9 if(a>=b)
10 error('valori di intervallo errati');
11 end;
12 if(n<0)
13 error('grado del polinomio errato');
14 end;
15 x = (a+b)/2+((b-a)/2)*cos(pi*(2*(n:-1:0)+1)./(2*(n+1)));
16 end

```

Listing 23: Codice della funzione per il calcolo delle ascisse di Chebyshev

## 22 Esercizio 22

Costruire una function Matlab, con sintassi `ll = lebesgue( a, b, nn, type )`, che approssimi la costante di Lebesgue per l'interpolazione polinomiale sull'intervallo  $[a,b]$ , per i polinomi di grado specificato nel vettore `nn`, utilizzando ascisse equidistanti, se `type=0`, o di Chebyshev, se `type=1` (utilizzare 10001 punti equispaziati nell'intervallo  $[a,b]$  per ottenere ciascuna componente di `ll`). Graficare opportunamente i risultati ottenuti, per `nn=1:100`, utilizzando  $[a,b]=[0,1]$  e  $[a,b]=[-5,8]$ . Commentare i risultati ottenuti.

### 22.1 Soluzione

```
1 function ll = lebesgue( a, b, nn, type )
2 %
3 % ll = lebesgue( a, b, nn, type)
4 %Funzione che approssima la costante di Lebesgue
5 %
6 % Input:a,b- estremi dell'intervallo, rispettivamente inferiore e superiore
7 %       nn- grado del polinomio
8 %       type- o 0 o 1, per il tipo di ascisse di interpolazione da usare
9 %
10 % Output:ll- approssimazione della costante di Lebesgue
11
12 if nn<= 0
13 error('grado non valido');
14 end
15 if a>= b
16 error('Estremi errati');
17 end
18 if type~=0 && type~=1
19 error('Valore di type non corretto');
20 end
21 fi=linspace(a,b,10001);
22 ll= zeros(1, length(nn));
23 for i=1:length(nn)
24 if type==0
25     x = linspace (a,b, nn(i));
26 else
27     x=chebyshev(nn(i),a,b);
28 end
29 lan= zeros(1,length(fi));
30 %calcolo del polinomio di lagrange
31 for k=1:length(x)
32     lin = ones(size(fi));
33
34     for j=1:length(x)
35         if k~=j
36             lin = lin .*((fi-x(j))/(x(k)-x(j)));
37
38         end
39     end
40 lan = lan + abs(lin);
41 end
42 %calcolo della costante di lebesgue
43 ll(i) = max(lan);
44 end
45 end
```

Listing 24: Codice della funzione che approssima la costante di Lebesgue nell'intervallo  $(a,b)$

il seguente codice è stato usato per graficare i grafici richiesti

```

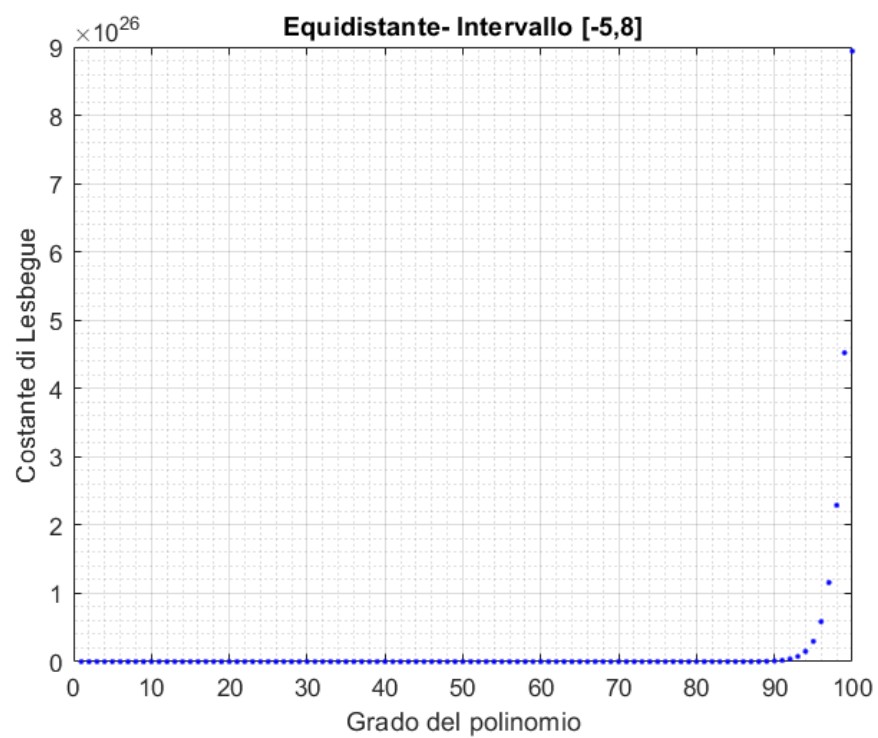
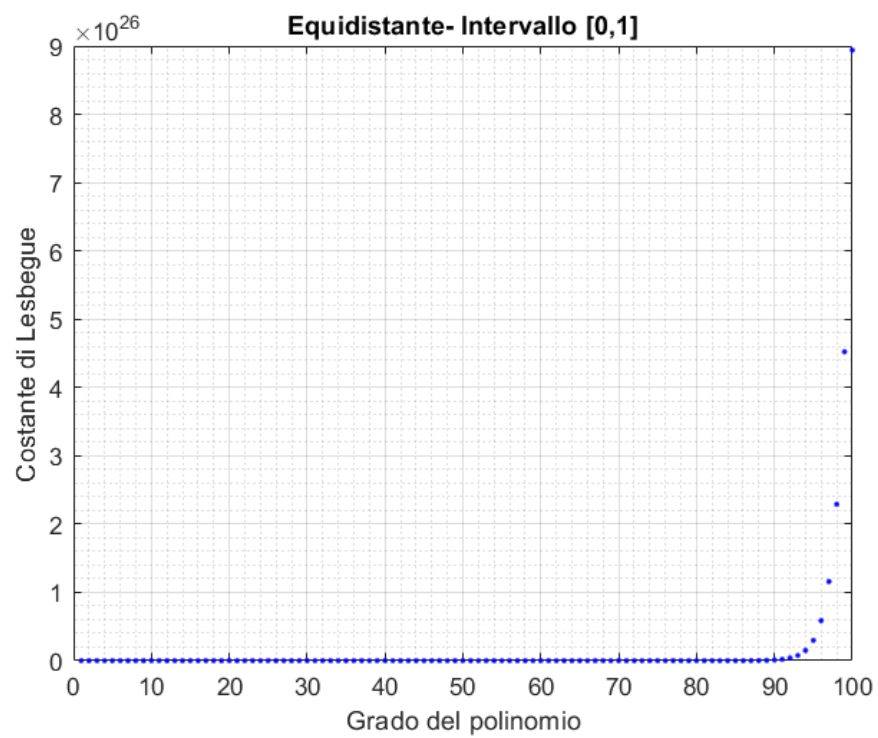
1
2 nn=1:100;
3 %costante di Lebesgue per ascisse di Chebyshev
4 cheb1 = lebesgue(0, 1, nn, 1);
5 cheb2 = lebesgue(-5, 8, nn, 1);
6 %costante di Lebesgue per ascisse equidistanti
7 leb1 = lebesgue(0, 1, nn, 0);
8 leb2 = lebesgue(-5, 8, nn, 0);
9
10 % Grafico Equidistante aintervallo [0,1]
11 figure;
12 plot(nn, leb1, 'b');
13 title('Equidistante- Intervallo [0,1]');
14 xlabel('Grado del polinomio');
15 ylabel('Costante di Lesbegue');
16 grid on;
17 set(gca, 'XMinorGrid', 'on');
18 set(gca, 'YMinorGrid', 'on');
19
20 % Grafico Equidistante intervallo [-5,8]
21 figure;
22 plot(nn, leb2, 'b');
23 title('Equidistante- Intervallo [-5,8]');
24 xlabel('Grado del polinomio');
25 ylabel('Costante di Lesbegue');
26 grid on;
27 set(gca, 'XMinorGrid', 'on');
28 set(gca, 'YMinorGrid', 'on');
29
30 % Grafico Chebyshev intervallo [0,1]
31 figure;
32 plot(nn, cheb1, 'r');
33 title('Chebyshev- Intervallo [0,1]');
34 xlabel('Grado del polinomio');
35 ylabel('Costante di Lesbegue');
36 grid on;
37 set(gca, 'XMinorGrid', 'on');
38 set(gca, 'YMinorGrid', 'on');
39
40 % Grafico Chebyshev intervallo [-5,8]
41 figure;
42 plot(nn, cheb2, 'r');
43 title('Chebyshev- Intervallo [-5,8]');
44 xlabel('Grado del polinomio');
45 ylabel('Costante di Lesbegue');
46 grid on;
47 set(gca, 'XMinorGrid', 'on');
48 set(gca, 'YMinorGrid', 'on');

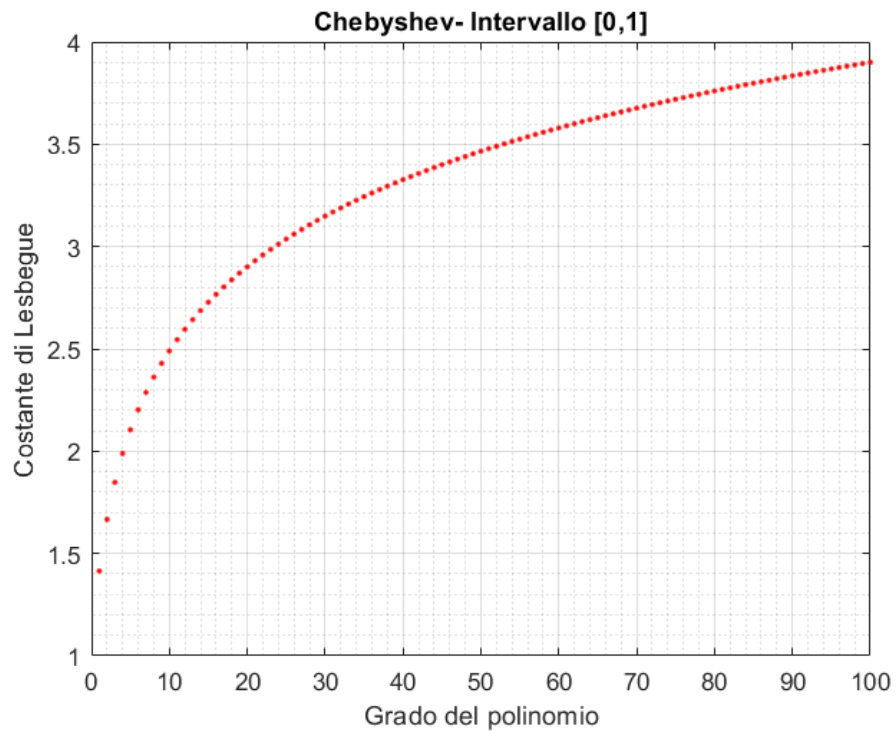
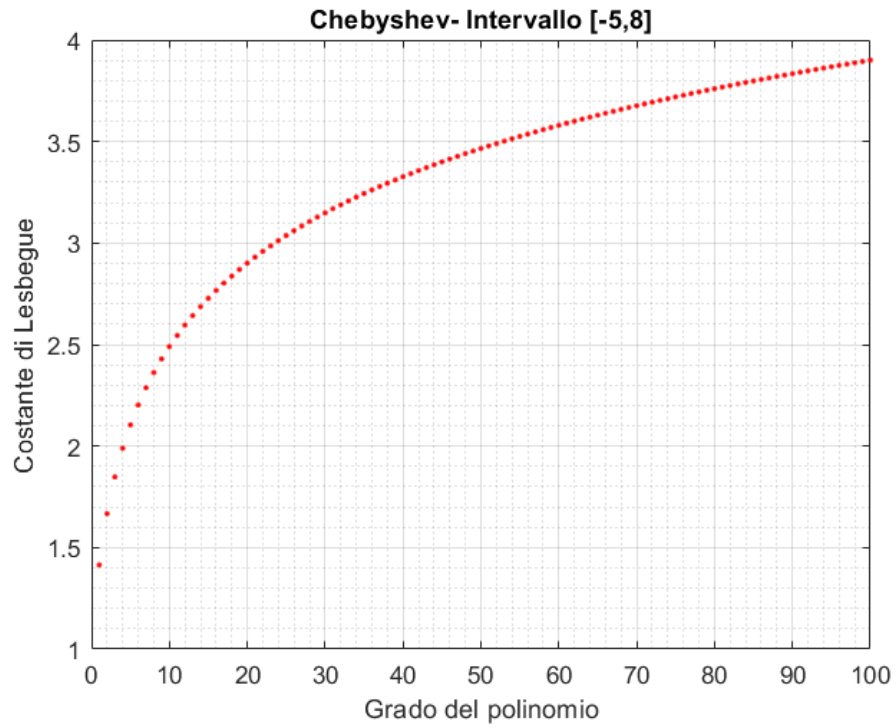
```

Listing 25: Codice per graficare i grafici richiesti

Di seguito verranno mostrati i grafici risultanti







Osservando i grafici si nota chiaramente come la costante di Lesbeque cresca esponenzialmente nel caso di ascisse equidistanti mentre se utilizziamo le ascisse di Chebyshev cresce in maniera ottimale.

## 23 Esercizio 23

Utilizzando le function degli esercizi 16 e 17, graficare (in semilogy) l'andamento errore di interpolazione (utilizzare 10001 punti equispaziati nell'intervallo per ottenere la stima) per la funzione di Runge,

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

utilizzando le ascisse di Chebyshev, per i polinomi interpolanti di grado  $n = 2:2:100$ . Commentare i risultati ottenuti.

### 23.1 Soluzione

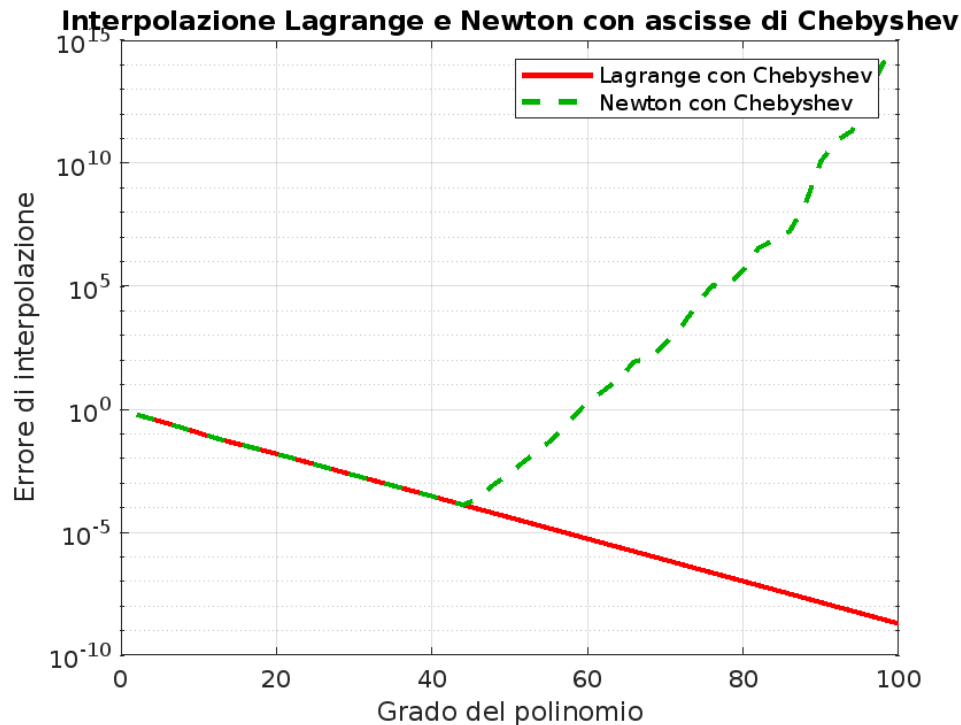
```
1 % Funzione di Runge
2 funzione_runge = @(x) 1./(1+x.^2);
3
4 % Intervallo di interesse
5 inizio_intervallo = -5;
6 fine_intervallo = 5;
7
8 punti_equispaziati = 10001;
9 xq = linspace(inizio_intervallo, fine_intervallo, punti_equispaziati);
10 gradi_polinomio = 2:2:100;
11
12 % Memorizzare gli errori
13 errori = struct('Lagrange', zeros(size(gradi_polinomio)), 'Newton', zeros(size(
    gradi_polinomio)));
14
15 % Calcolo dei valori esatti della funzione di Runge
16 valori_esatti = funzione_runge(xq);
17
18 % Ciclo per calcolare gli errori
19 for i = 1:length(gradi_polinomio)
20     n = gradi_polinomio(i);
21
22     % Calcolo delle ascisse di Chebyshev
23     ascisse = chebyshev(n, inizio_intervallo, fine_intervallo);
24
25     % Calcolo dei valori approssimati usando l'interpolazione di Lagrange e Newton
        con ascisse di Chebyshev
26     val_lagrange = lagrange(ascisse, funzione_runge(ascisse), xq);
27     val_newton = newton(ascisse, funzione_runge(ascisse), xq);
28
29     % Calcolo dell'errore
30     error_lagrange(i) = max(abs(valori_esatti - val_lagrange));
31     error_newton(i) = max(abs(valori_esatti - val_newton));
32 end
33
34 % Grafico dell'errore in scala logaritmica
35 figure;
36
37 semilogy(gradi_polinomio, error_lagrange, '-r', 'LineWidth', 2);
38 hold on;
39 colore_verde = [0, 0.7, 0];
40 semilogy(gradi_polinomio, error_newton, '--', 'Color', colore_verde, 'LineWidth',
    2);
41
42 xlabel('Grado del polinomio');
43 ylabel('Errore di interpolazione');
```

```

44 legend('Lagrange con Chebyshev', 'Newton con Chebyshev');
45 title('Interpolazione Lagrange e Newton con ascisse di Chebyshev');
46
47 grid on;
48 hold off;

```

Listing 26: Codice per graficare l'andamento dell'errore di interpolazione



Il grafico dell'errore di interpolazione mostra come l'errore varia al variare del grado del polinomio per le forme di Lagrange e Newton, utilizzando le ascisse di Chebyshev.

Per quanto riguarda l'interpolazione di Lagrange, si osserva una riduzione dell'errore al crescere del grado del polinomio, evidenziando il beneficio delle ascisse di Chebyshev nella riduzione dell'errore di interpolazione. Tuttavia, nel caso dell'interpolazione di Newton, nonostante le ascisse di Chebyshev favoriscano una migliore crescita della costante di Lebesgue, si registra un significativo aumento dell'errore per gradi del polinomio più elevati. Questo fenomeno è attribuibile al problema di Runge, che rende il problema di interpolazione in base di Newton mal condizionato per gradi polinomiali elevati. Pertanto, se da un lato l'utilizzo delle ascisse di Chebyshev migliora l'efficienza dell'interpolazione polinomiale in base di Lagrange, dall'altro, per l'interpolazione in base di Newton, la loro efficacia diminuisce oltre un certo grado del polinomio.

## 24 Esercizio 24

Costruire una function, `spline0.m`, avente la stessa sintassi della function `spline` di Matlab, che calcoli la *spline* cubica interpolante naturale i punti  $(x_i, f_i)$ .

**N.B.:** il risultato dovrà avere le stesse dimensioni del dato in ingresso

### 24.1 Soluzione

```

1 function yy = spline0(x, y, xq)
2 %

```

```

3 % yy = spline0(x, y, xq)
4 %funzione che calcola la spline cubica naturale nei punti di interpolazione
5 %
6 % input: x-vettore delle x
7 %       y- valori che prende la funzione nelle x
8 %       xq- vettore dei punti in cui va calcolato il polinomio
9 % output: yy-valori della spline nei punti xq
10 if nargin < 3
11     error('parametri in ingresso insufficienti');
12 end
13 n = length(x);
14 if n ~= length(y)
15     error('parametri in ingresso errati');
16 end
17 n = n - 1;
18 h = zeros(1, n);
19 for i = 1:n
20     h(i) = x(i+1) - x(i);
21 end
22 dd = y;
23 % calcolo differenze divise
24 for j = 1:2
25     for i = n+1:-1:j+1
26         dd(i) = (dd(i)- dd(i-1)) / (x(i) - x(i - j));
27     end
28     if j == 1
29         dd1f = dd(2:end);
30         % dd1f contiene le differenze divise f[x(k-1),x(k)], k = 2:n
31     end
32 end
33 phi = h(2:n-1) ./ (h(2:n-1) + h(3:n));
34 csi = h(2:n-1) ./ (h(1:n-2) + h(2:n-1));
35 a(1:n-1) = 2;
36 m = zeros(1, n + 1);
37 m(2:n) = trilu(a, phi, csi, 6 * dd(3:end));
38 yy = zeros(1, length(xq));
39 for j = 1:length(xq)
40     if xq(j) >= x(1) && xq(j) <= x(n+1)
41         for k=2:n+1
42             if xq(j) <= x(k)
43                 i = k-1;
44                 break
45             end
46         end
47         ri = y(i) - (h(i) ^ 2) / 6 * m(i);
48         qi = dd1f(i) - h(i) / 6 * (m(i+1) - m(i));
49         yy(j) = ((xq(j) - x(i)) ^ 3 * m(i+1) + (x(i+1) - xq(j)) ^ 3 * m(i)) ...
50             / (6 * h(i)) + qi * (xq(j) - x(i)) + ri;
51     end
52 end
53 return

```

Listing 27: Codice della funzione spline0

```

1 function x = trilu(a, b, c, x)
2 % Risolve un sistema lineare tridiagonale fattorizzabile LU.
3 %
4 % x = trilu(a, b, c, x)
5 %

```

```

6 % Input: a- vettore degli elementi diagonali
7 %       b- vettore degli elementi sottodiagonali
8 %       c- vettore degli elementi sopradiagonali
9 %       x- vettore dei termini noti
10 %
11 % Output: x- soluzione
12 %
13 n = length(a);
14 for i = 1:n-1
15     b(i) = b(i)/a(i);
16     a(i+1) = a(i+1) - b(i)*c(i);
17     x(i+1) = x(i+1) - b(i)*x(i);
18 end
19 x(n) = x(n)/a(n);
20 for i = n-1:-1:1
21     x(i) = (x(i) - c(i)*x(i+1))/a(i);
22 end
23 return
24 end

```

Listing 28: Codice della funzione per risolvere un sistema tridiangolare fattorizzabile LU

## 25 Esercizio 25

Graficare, utilizzando il formato loglog, l'errore di approssimazione utilizzando le *spline* interpolanti naturale *not-a-knot* per approssimare la funzione di Runge sull'intervallo  $[-10, 10]$ , utilizzando una partizione uniforme

$$\Delta = \left\{ x_i = -10 + i \frac{20}{n}, \quad i = 0, \dots, n \right\}, \quad n = 4:4:800,$$

rispetto alla distanza  $h = 20/n$  tra le ascisse. Utilizzare 10001 punti equispaziati nell'intervallo  $[-10, 10]$  per ottenere la stima dell'errore. Che tipo di decrescita si osserva?

### 25.1 Soluzione

Di seguito è presente il codice usato per graficare le spline:

```

1 f = @(x) 1 ./ (1 + x.^ 2);
2 a = -10;
3 b = 10;
4 x = linspace(a, b, 10001);
5 fx = f(x);
6 errKnot = zeros(1, 200);
7 errNaturali = zeros(1, 200);
8 h = zeros(1, 200);
9
10 index = 1;
11 for n = 4:4:800
12     xi = linspace(a, b, n+1); % Ascisse equidistanti
13     fi = f(xi);
14
15     % calcolo le spline e i loro rispettivi errori massimi
16     splineKnot = spline(xi, fi, x);
17     splineNat = spline0(xi, fi, x);
18
19     errKnot(index) = max(abs(fx - splineKnot));
20     errNaturali(index) = max(abs(fx - splineNat));
21     h(index) = 20 / n;

```

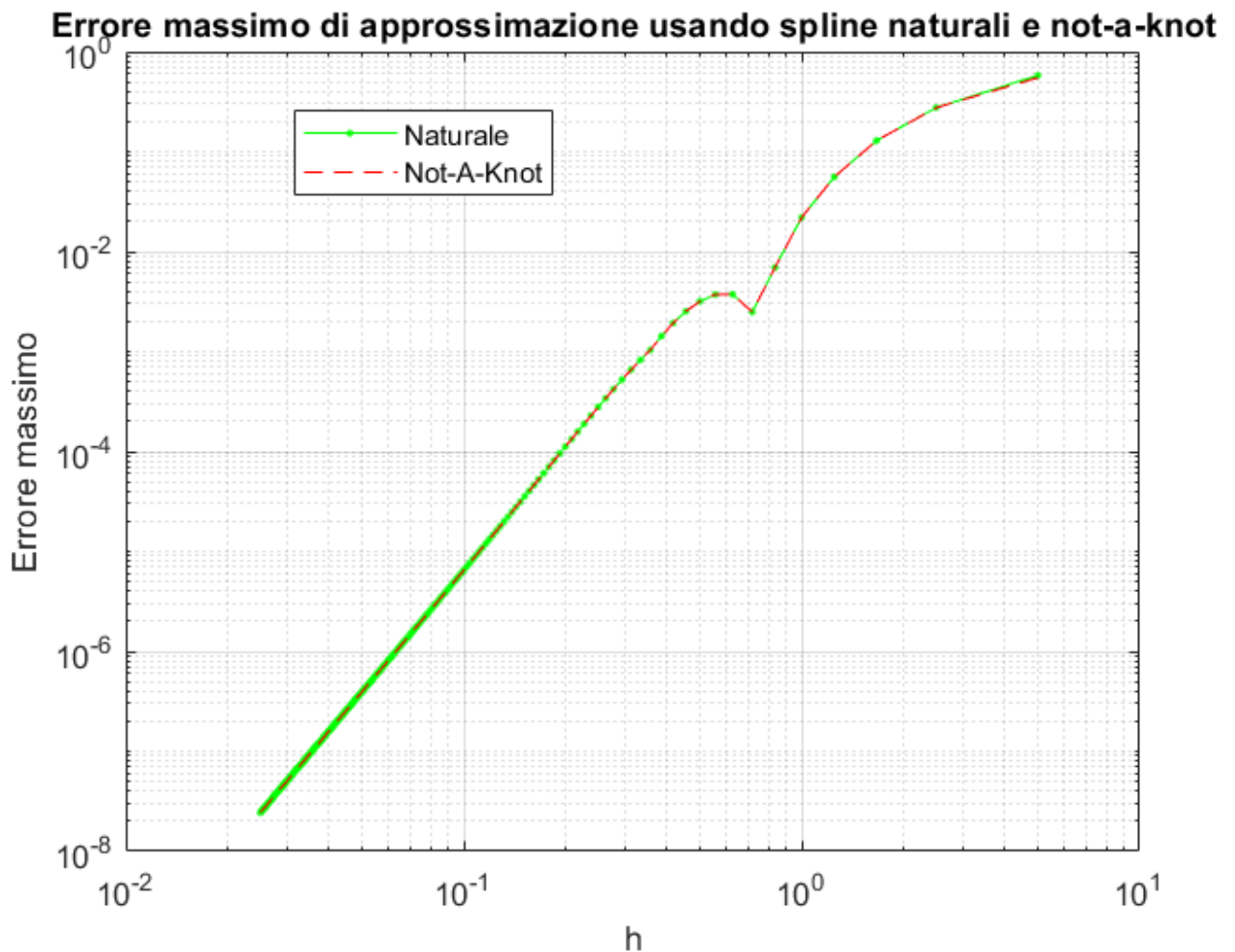
```

22     index = index + 1;
23 end
24
25 %Grafico gli errori
26 figure;
27 loglog(h, errKnot, 'g.-',h, errNaturali, 'r--' );
28 xlabel('h');
29 ylabel('Errore massimo');
30 title('Errore massimo di approssimazione usando spline naturali e not-a-knot');
31 legend('Naturale', 'Not-A-Knot');
32 grid on;

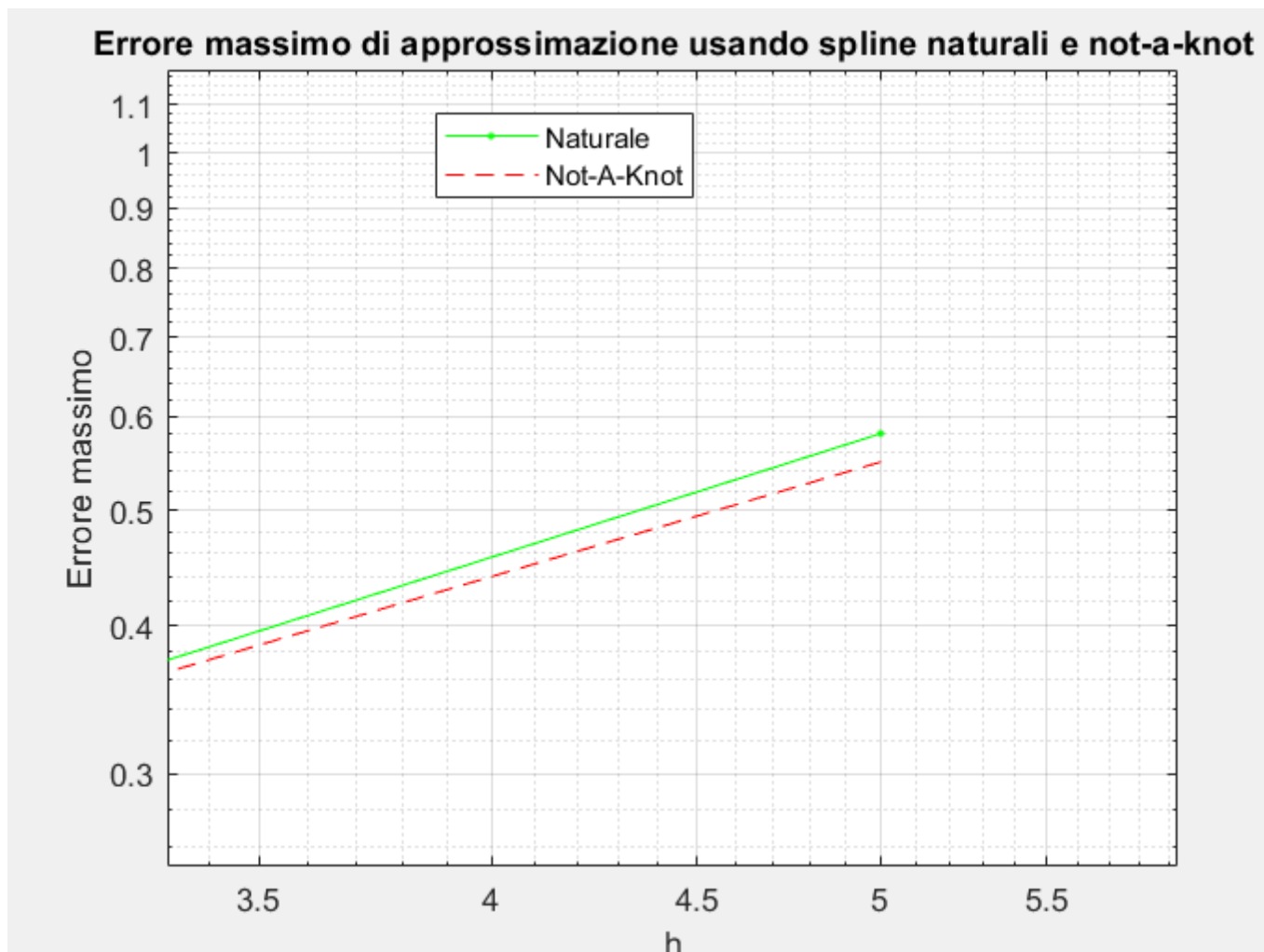
```

Listing 29: Codice per graficare le spline nell'intervallo  $(-10,10)$

Il grafico risultante è il seguente:



Dal grafico si nota che nonostante le due spline abbiano errori leggermente diversi alla fine i due grafici si incontrano e terminano in modo quasi identico, quindi possiamo dire che all'aumentare del numero di intervalli gli errori si equivalgono.



## 26 Esercizio 26

Graficare, utilizzando il formato loglog, l'errore di approssimazione utilizzando le *spline* interpolanti naturale *not-a-knot* per approssimare la funzione di Runge sull'intervallo  $[-10, 10]$ , utilizzando una partizione uniforme

$$\Delta = \left\{ x_i = i \frac{20}{n}, \quad i = 0, \dots, n \right\}, \quad n = 4:4:800,$$

rispetto alla distanza  $h = 10/n$  tra le ascisse. Utilizzare 10001 punti equispaziati nell'intervallo  $[0, 10]$  per ottenere la stima dell'errore. Che tipo di decrescita si osserva? Confrontare e discutere i risultati ottenuti, rispetto a quelli del precedente esercizio.

### 26.1 Soluzione

Di seguito è riportato il codice usato per graficare le due spline

```

1 f = @(x) 1 ./ (1 + x.^ 2);
2 a = 0;
3 b = 10;
4 x = linspace(a, b, 10001);
5 fx = f(x);
6 errKnot = zeros(1, 200);

```



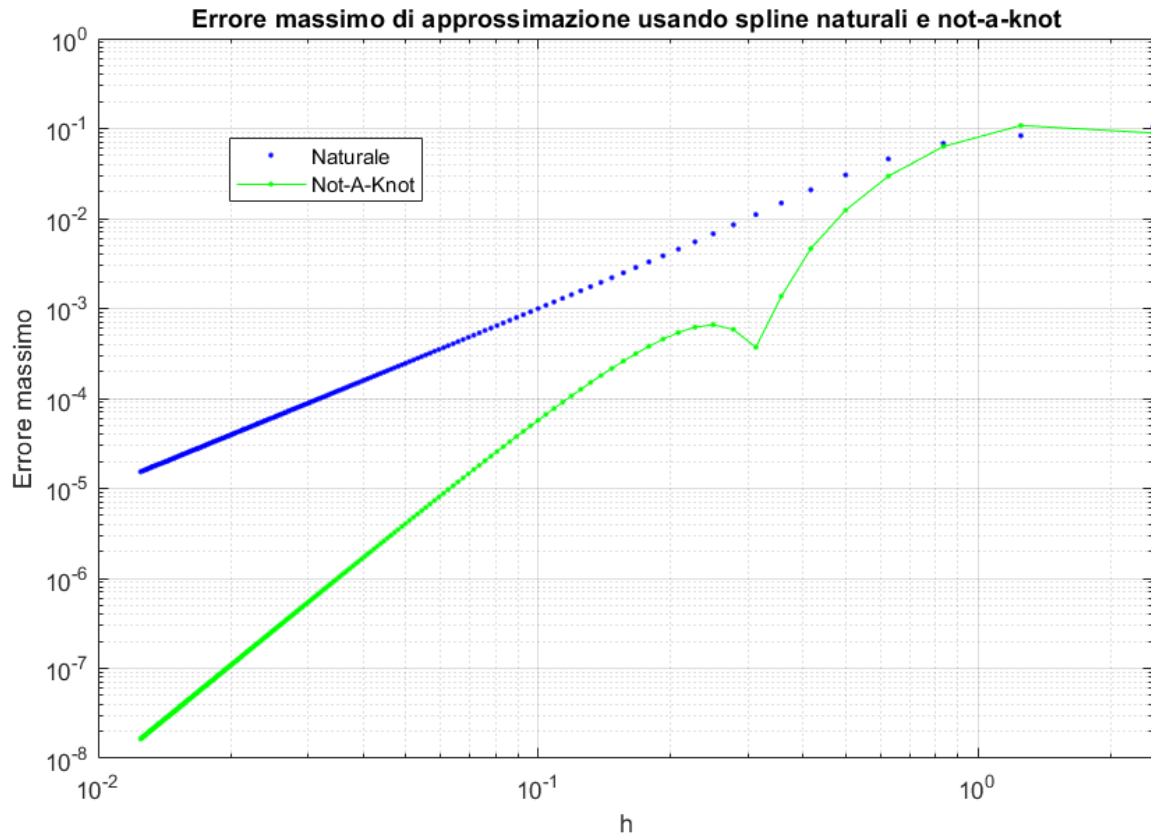
```

7 errNaturali = zeros(1, 200);
8 h = zeros(1, 200);
9
10 index = 1; %index = n/4
11 for n = 4:4:800
12     xi = linspace(a, b, n+1); % Ascisse equidistanti
13     fi = f(xi);
14
15     % calcolo le spline e i loro rispettivi errori massimi
16     splineKnot = spline(xi, fi, x);
17     splineNat = spline0(xi, fi, x);
18
19     errKnot(index) = max(abs(fx - splineKnot));
20     errNaturali(index) = max(abs(fx - splineNat));
21     h(index) = 10 / n;
22     index = index + 1;
23 end
24
25 %Grafico gli errori
26 figure;
27 loglog(h, errNaturali, 'b.', h, errKnot, 'g.-');
28 xlabel('h');
29 ylabel('Errore massimo');
30 title('Errore massimo di approssimazione usando spline naturali e not-a-knot');
31 legend('Naturale', 'Not-A-Knot');
32 grid on;

```

Listing 30: Codice per il grafico delle spline

Il grafico risultante è il seguente:



Come possiamo notare a differenza dell'esercizio precedente l'uso delle spline not-a-knot porta a ridurre l'errore di interpolazione all'aumentare del numero di intervalli, mentre nel precedente esercizio i due metodi avevano andamenti quasi identici in questo caso la spline naturale ha una decrescita dell'errore più regolare della spline not-a-knot.

## 27 Esercizio 27

Calcolare i coefficienti del polinomio di approssimazione ai minimi quadrati di grado 3 per i seguenti dati:

```
>> rng(0);
xi=linspace(0, 2*pi, 101);
yi=sin(xi)+rand(size(xi))*0.5;
```

Graficare convenientemente i risultati ottenuti.

### 27.1 Soluzione

```
1  rng(0);
2  xi = linspace(0, 2*pi, 101);
3  yi = sin(xi) + rand(size(xi)) * 0.05;
4
5  % Calcolo dei coefficienti del polinomio di grado 3
6  grado_polinomio = 3;
7  coefficienti_minimi_quadrati = polyfit(xi, yi, grado_polinomio);
8
9  % Mostra i coefficienti calcolati
10 disp('Coefficienti del polinomio di approssimazione:');
```

```

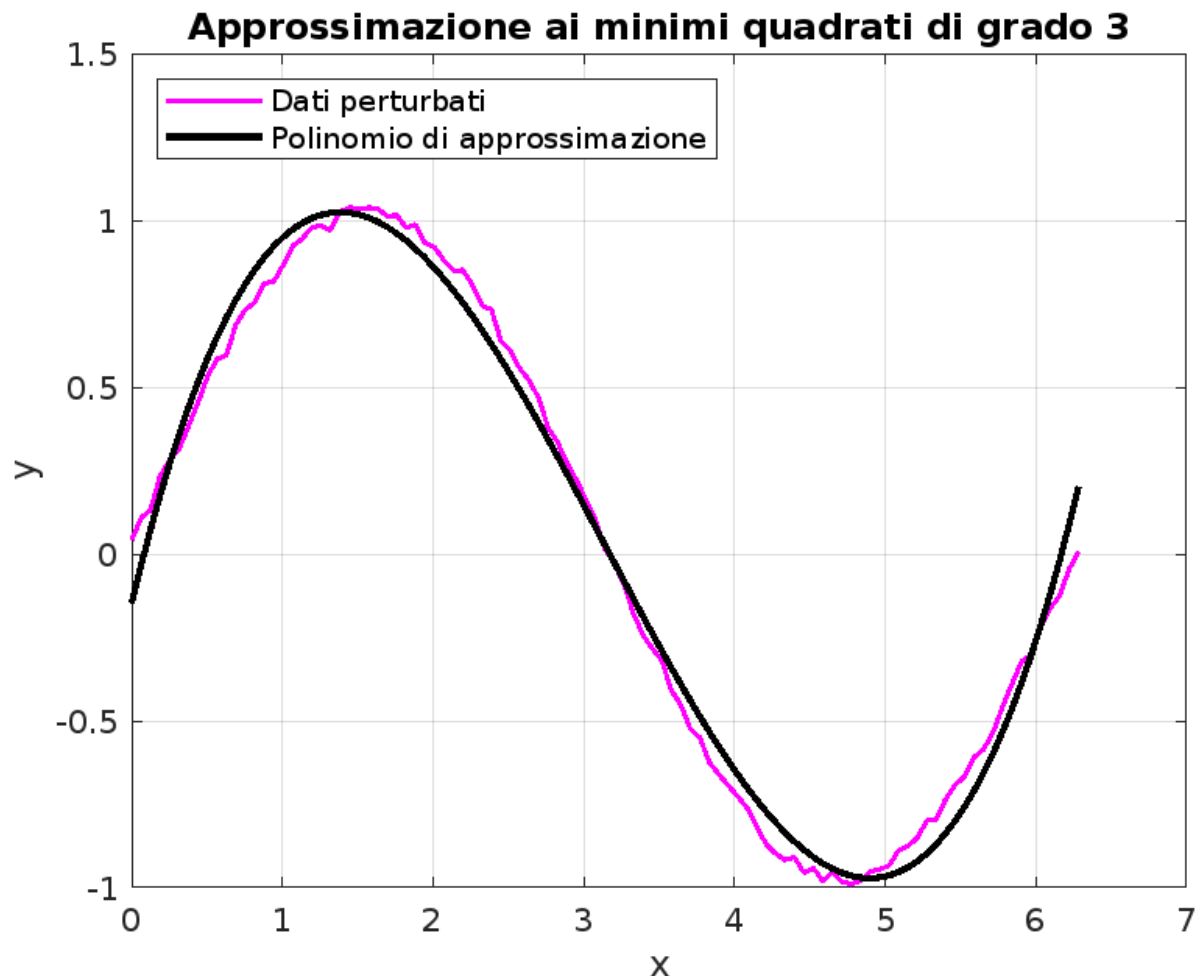
11 disp(coefficienti_minimi_quadrati);
12
13 % Valutazione del polinomio nei punti xi
14 yi_polyval = polyval(coefficienti_minimi_quadrati, xi);
15
16 % Grafico dei dati e del polinomio di approssimazione
17 figure;
18 plot(xi, yi, 'm-', 'LineWidth', 1.5, 'DisplayName', 'Dati perturbati');
19 hold on;
20 plot(xi, yi_polyval, 'k-', 'LineWidth', 2, 'DisplayName', 'Polinomio di
    approssimazione');
21 xlabel('x');
22 ylabel('y');
23 title('Approssimazione ai minimi quadrati di grado 3');
24 legend('show', 'Location', 'northwest');
25 grid on;

```

Listing 31: Codice per ottenere i risultati il grafico

Coefficienti del polinomio di approssimazione			
0.0920	-0.8666	1.8704	-0.1484

Il grafico risultante è il seguente:



## 28 Esercizio 28

Costruire una function Matlab che, dato in input  $n$ , restituisca i pesi della quadratura della formula di Newton-Cotes di grado  $n$ . Tabulare, quindi, i pesi delle formule di grado 1, 2, ..., 7 e 9 (come numeri razionali).

### 28.1 Soluzione

```
1 function p = newtonCotes(n)
2 %
3 %   c = newtonCotes(n);
4 %   Funzione che calcola i valori dei pesi per la formula di Newton-Cotes di grado
   n.
5 %
6 %   Input: n - grado della formula
7 %   Output: p - vettore dei pesi risultante
8 %
9   if nargin < 1
10      error('Errore: parametri in ingresso insufficienti');
11   elseif n < 1
12      error('Errore: Parametri in ingresso sbagliati');
13   end
14
15   k = n+1:-1:1;
16   p = zeros(1,n+1);
17   for i=0:n
18      d = [0:i-1 i+1:n];
19      den = prod(i - d);
20      alpha = poly(d);
21      alpha = [alpha ./ k 0];
22      p(i+1) = polyval(alpha,n) / den;
23   end
24   return
```

Listing 32: Codice della funzione per calcolare i pesi di Newton-Cotes

Di seguito è riportato il codice usato per calcolare i pesi delle formule dei gradi richiesti:

```
1 gradi = [1, 2, 3, 4, 5, 6, 7, 9];
2
3 fprintf('Grado | Pesi della quadratura\n');
4 for i = 1:length(gradi)
5     fprintf('-----');
6 end
7 fprintf('\n');
8 for i = 1:length(gradi)
9     g = gradi(i);
10    if g == 8 || g>9
11        fprintf('Grado %d non accettabile\n', g);
12        continue;
13    end
14
15    pesi = newtonCotes(g);
16    % Convertiamo i pesi in forma frazionaria
17    pesifraz = rats(pesi);
18    fprintf(' %d | ', g);
19    disp(pesifraz);
20 end
```

Listing 33: Codice per mostrare i pesi

i pesi risultanti sono:

Grado	Pesi della quadratura										
1		1/2	1/2								
2		1/3	4/3	1/3							
3		3/8	9/8	9/8	3/8						
4		14/45	64/45	8/15	64/45	14/45					
5		95/288	125/96	125/144	125/144	125/96	95/288				
6		41/140	54/35	27/140	68/35	27/140	54/35	41/140			
7		108/355	810/559	343/640	649/536	649/536	343/640	810/559	108/355		
9		130/453	419/265	23/212	307/158	213/367	213/367	307/158	23/212	419/265	130/453

## 29 Esercizio 29

Scrivere una function in Matlab, `[If,err] = composita( fun, a, b, k, n)` che implementi la formula composta di Newton-Cotes di grado  $k$  su  $n+1$  ascisse equidistanti, con  $n$  multiplo pari di  $k$ , in cui:

- `fun` è la funzione integranda (che accetta input vettoriali);
- `[a,b]` è l'intervallo di integrazione;
- `k`, `n` come su descritti;
- `If` è l'approssimazione dell'integrale ottenuta;
- `err` è la stima dell'errore di quadratura.

Le valutazioni funzionali devono essere fatte tutte insieme in modo vettoriale, senza ridondanze.

### 29.1 Soluzione

```

1 function [If, err] = composita(fun, a, b, k, n)
2 % [If, err] = composita(fun, a, b, k, n)
3 % Implementa la formula composta di Newton-Cotes di grado k su n+1 ascisse
  equidistanti
4 %
5 % Input: fun - funzione integranda
6 %         [a,b] - intervallo di integrazione
7 %         k - grado della formula di Newton-Cotes
8 %         n - numero di suddivisioni dell'intervallo di integrazione (multiplo pari
  di k)
9 %
10 % Output: If - approssimazione dell'integrale
11 %          err - stima dell'errore di quadratura
12 %
13 if nargin < 5
14     error('Numero di parametri insufficienti');
15 end
16 % Controlla che k sia un intero positivo
17 if ~isnumeric(k) || mod(k,1) ~= 0 || k <= 0
18     error('k deve essere un intero positivo');
19 end
20 % Controlla che gli estremi di integrazione siano corretti
21 if a > b
22     error('Gli estremi di integrazione non sono validi');
23 end

```

```

24 % Controlla che n non sia un multiplo pari di k
25 if mod(n,k)~=0
26 error("Errore: il valore k deve essere un multiplo pari di n")
27 end
28 % Controlla che n sia un multiplo pari
29 if mod(n,2)==0
30     mu = 2;
31 else
32     mu = 1;
33 end
34 % Calcola i pesi della formula di Newton-Cotes di grado k
35 pesi = newtonCotes(k);
36 % Calcola l'approssimazione dell'integrale
37 If = 0;
38 I1 = 0;
39 h = (b - a) / n;
40 h1 = (b - a) / (n/2);
41 for i = 0:(n-1)
42     x = linspace(a + i*h, a + (i+1)*h, k+1);
43     y = feval(fun, x);
44     If = If + h/k*sum(y.*pesi);
45     if i<n/2
46         x1 = linspace(a + i*h1, a + (i+1)*h1, k+1);
47         y1 = feval(fun, x1);
48         I1 = I1 + h1/k*sum(y1.*pesi);
49     end
50
51 end
52 % Stima l'errore
53 err = abs((If-I1)/(2^(k+mu)-1));
54 end

```

Listing 34: Formula composite di Newton-Cotes di grado k su ascisse equidistanti, con n multiplo pari di k

## 30 Esercizio 30

Calcolare l'espressione del seguente integrale:

$$I(f) = \int_0^1 e^{3x} dx.$$

Utilizzare la function del precedente esercizio per ottenere un'approssimazione dell'integrale per i valori  $k = 1, 2, 3, 6$ , e  $n = 12$ . Tabulare i risultati ottenuti, confrontando l'errore stimato con quello vero.

### 30.1 Soluzione

di seguito è riportato il codice per calcolare l'approssimazione dell'integrale

```

1 f = @(x) exp(3 * x);
2 F = @(x) exp(3 * x) / 3;
3 a = 0;
4 b = 1;
5 IntEsatto = F(b) - F(a); % Valore esatto dell'integrale
6 format long
7 k = [1, 2, 3, 6];
8 n = 12;
9
10 If = zeros(size(k));

```

```

11 err = zeros(size(k));
12 % Calcolo l'integrale usando il metodo di Newton-Cotes composito per ciascun k
13 for i = 1:length(k)
14     [If(i), err(i)] = composita(f, a, b, k(i), n);
15 end
16
17 disp('Valore esatto:');
18 disp(IntEsatto);
19 err1 = (IntEsatto - If);
20
21 T = table(k', If', err', err1', ...
22           'VariableNames', {'k', 'Integr_Approx', 'Errore_Stimato', '
23                               Errore_Effettivo'});
24 disp(T);

```

Listing 35: funzione per calcolare l'approssimazione dell'integrale

I risultati vengono quindi:

```

>> TesterEs30
Valore esatto:
6.361845641062557

```

<b>k</b>	<b>Integr_Approx</b>	<b>Errore_Stimato</b>	<b>Errore_Effettivo</b>
1	6.39494578983666	0.014127046469173	-0.0331001487740989
2	6.36185425384364	8.56183766337892e-06	-8.61278107855412e-06
3	6.36184946975619	1.84285245248774e-06	-3.82869363768634e-06
6	6.36184564106228	6.13365567487028e-14	2.71782596428238e-13