Elaborato di Calcolo Numerico A.A 2023-2024

Autore	Matricola	e-mail
Ede Boanini	[redacted]	[redacted]
Ginevra Lavacchi		

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:

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

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

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

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

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

$$25f(x) - 48\cdot\left(f(x) - hf'(x) + \frac{h^2 f''(x)}{2} - \frac{h^3 f'''(x)}{6} + \frac{h^4 f^{(4)}(x)}{24} + O(h^5)\right)$$

$$+ 36\cdot\left(f(x) - 2hf'(x) + \frac{4h^2 f''(x)}{2} - \frac{8h^3 f'''(x)}{6} + \frac{16h^4 f^{(4)}(x)}{24} + O(h^5)\right)$$

$$- 16\cdot\left(f(x) - 3hf'(x) + \frac{9h^2 f''(x)}{2} - \frac{27h^3 f'''(x)}{6} + \frac{81h^4 f^{(4)}(x)}{24} + O(h^5)\right)$$

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

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:

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

2 Esercizio 2

La funzione

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

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

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

(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.

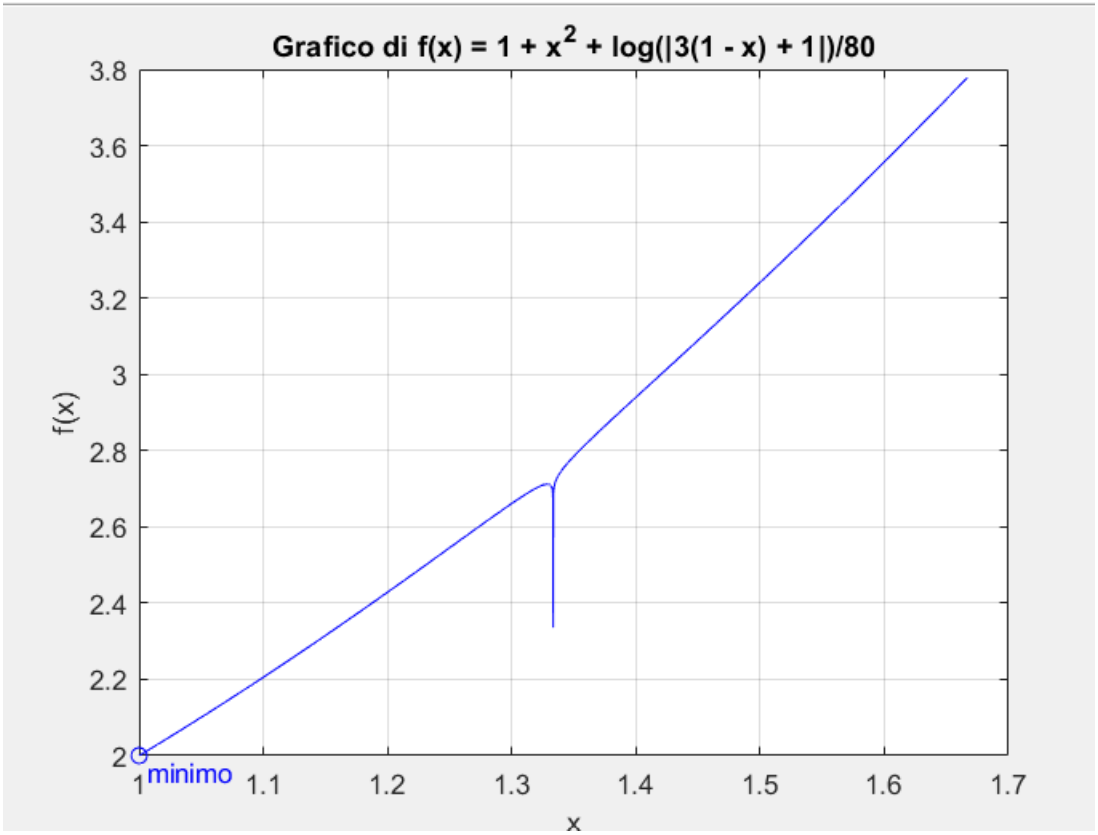
2.1 Soluzione

Il codice Matlab è:

```
% Definisco i limiti dell' intervallo x
x1 = 1; x2 = 4/3 - 0.01; % Intervallo a sinistra dell'asintoto
x3 = 4/3 + 0.01; x4 = 5/3; % Intervallo a destra dell'asintoto
% Uso linspace per generare i valori x
x=linspace(x1,x4,100001);
% Calcolo i corrispondenti valori y
y= 1 + x.^2 + (log(abs(3*(1 - x) + 1)))/80;
%calcolo i valori dei limiti della funzione in 4/3
x_sx=1 + x2.^2 + (log(abs(3*(1 - x2) + 1)))/80
x_dx=1 + x3.^2 + (log(abs(3*(1 - x3) + 1)))/80
% Creo il grafico
figure
plot(x,y,'b')
hold on
title('Grafico di f(x) = 1 + x^2 + log(|3(1 - x) + 1|)/80')
xlabel('x')
ylabel('f(x)')
grid on
% Trovo il minimo della funzione
[min_val, min_idx] = min(y);
min_x = x(min_idx);

plot(min_x,min_val,'bo')
text(min_x, min_val, ' minimo','VerticalAlignment','top', 'HorizontalAlignment','
    left', 'FontSize',10,'Color','b');
fprintf('Il minimo della funzione si ottiene per x = %f, dove f(x) = %f\n', min_x,
    min_val);
fprintf('Limite sinistro:%f',x_sx);
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 $|x + y| << |x| + |y|$ e quindi $K >> 1$; 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

```
function [x,count] = bise( a, b, f, tolx )
%
% x = bise( a, b, f, tolx ) Metodo di bisezione per calcolare
% una radice di f(x), interna ad [a,b],
% con tolleranza tolx. Il metodo conta anche il numero di iterazioni
% (count)
%
count=0;
if a>=b, error('estremi intervallo errati'), end
if tolx<=0; error('tolleranza non appropriata'), end
fa = feval(f,a);
fb = feval(f,b);
if fa*fb>=0, error('intervallo di confidenza non appropriato'), end
imax = ceil( log2(b-a)-log2(tolx) );
if imax<1, x = (a+b)/2; return, end
for i = 1:imax
x = (a+b)/2;
fx = feval( f, x );
f1x = abs(fb-fa)/(b-a);
count=count+1;
if abs(fx)<=tolx*f1x
break
elseif fa*fx<0
b = x; fb = fx;
else
a = x; fa = fx;
end
end
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:

```
function [x,count] = newtonZeri(x0,f,df,tol,itmax)
%
% [x,it] = newtonZeri(x0,f,df,tol,itmax)
% funzione che ricerca gli zeri della funzione f tramite il metodo di Newton con
    approssimazione iniziale x0
%
% input: x0- approssimazione iniziale della radice
%        f- funzione per cui si ricercano gli zeri
%        df- derivata di f
%        tol- tolleranza
%        itmax- massimo numero di iterazioni
%
% output: x- radice
%         count- numero iterazioni
%
if nargin~=5,error("input errato"),end;
if tol<0,error('Errore tolleranza: la tolleranza deve essere maggiore di 0.'),end;
if itmax<=0
error('Errore numero iterazioni: le iterazioni devono essere maggiori di 0.');
end;
count=0;
x = x0;

for i=1:itmax
x0 = x;
fx0 = feval(f,x0);
dfx0 = feval(df,x0);
count=count+1;
if dfx0==0, error('Errore derivata prima uguale a 0'),end;
x = x0-(fx0/dfx0);
if abs(x-x0)<=tol
    break
end
end
if abs(x-x0)>tol
disp("Il metodo non converge");
end
return
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

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

```
%         count- numero iterazioni
%
if nargin~=5,error("input errato"),end;
if tol<=0,error("tolleranza non adeguata"),end;
if itmax<=0,error("num it non adeguato"),end;
count=0;
fx0=feval(f,x0);
fx1=feval(f,x1);

for i=1:itmax
if fx1==fx0
    error("approssimazione errata");
end
x=(fx1*x0-fx0*x1)/(fx1-fx0);
count=count+1;
if abs(x-x1)<=tol
    break
elseif i<itmax
    x0=x1;
    fx0=fx1;
    x1=x;
    fx1=feval(f,x1);
end
end
if abs(x-x1)>tol
error("tolleranza non rispettata");
end

return
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 $X_0 = 1$($e\ x_1 = 0.9$ 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:

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

```
x1=0.9;

for i=1:length(toll)
    [x_b,count_b]=bise(a, b, f, toll(i));
    [x_n, count_n]=newtonZeri(x0,f,f1,toll(i),1000);
    [x_s,count_s]=sec(x0,x1,f,toll(i),1000);

    radici=[x_s;x_b;x_n];
    countazioni=[count_s;count_b;count_n];
    disp("tolleranza = "+toll(i));
    metodo=["secanti";"bisezione";"newton"];

    t=table(metodo,radici,countazioni);
    disp(t);
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 iterazoni effettuate.

Per la tolleranza=10^{-3}

Metodo	*Radici*	*Iterazioni*
secanti	1.1522e-06	6
bisezione	0.00097656	9
newton	2.8423e-09	5

Per la tolleranza=10^{-6}

Metodo	*Radici*	*Iterazioni*
secanti	2.0949e-16	8
bisezione	9.5367e-07	19
newton	3.5748e-17	6

Per la tolleranza=10^{-9}

Metodo	*Radici*	*Iterazioni*
secanti	2.0949e-16	8
bisezione	9.3132e-10	29
newton	3.5748e-17	7

Per la tolleranza=10^{-12}

Metodo	*Radici*	*Iterazioni*
secanti	-1.2557e-17	9
bisezione	9.0949e-13	39
newton	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

```
function [x, it] = newtonModificato(x0, f, f1, m, tolx, maxit)
%
%  [x, it] = newtonModificato(x0, f, f1, m, tolx, maxit)
%  Funzione che implementa il metodo di Newton modificato per determinare
%  una approssimazione della radice
%
%  Input: x0 - approssimazione iniziale della radice
%         f - funzione che implementa f(x)
%         f1 - funzione che implementa f'(x)
%         m - molteplicit  della radice di f(x)
%         tolx - accuratezza richiesta (default = 1e-6)
%         maxit - numero massimo di iterazioni (default = 1000)
%
%  Output: x - approssimazione della radice di f(x);
%          it - num iterazioni
%
    if nargin < 4
        error('Numero di parametri in ingresso errato');
    elseif nargin == 4
        tolx = 1e-6;
        maxit = 1000;
    elseif nargin == 5
        maxit = 1000;
    end
    if tolx < 0 || maxit <= 0
        error('Parametri in ingresso errati');
    end

    x = x0;
    for it = 1:maxit
        fx0  = feval(f, x);
        f1x0 = feval(f1, x);
        if f1x0 == 0
            error('La derivata prima si annulla');
        end
        x = x - m * (fx0 / f1x0);
        % Verifica della condizione di convergenza
        if abs(x - x0) <= tolx * (1 + abs(x0))
            break
        else
        x0 = x;
        end
        if it == maxit
            disp('Il metodo non converge.');
        end
    end
return
```

Listing 6: Codice del metodo newtonModificato

```
% Tolleranze
toll=[1e-3;1e-6;1e-9;1e-12];

% Funzione
f=@(x) exp(x) - cos(x) + sin(x) - x*(x+2);

% Derivata della funzione
```

```
f1=@(x) exp(x) + sin(x) + cos(x) -2*x -2;

% Intervallo confidenza per metodo bisezione
a=-0.1;
b=1;

% Approssimazioni iniziali
x0=1;
x1=0.9;

% Molteplicit  della radice m=5
%Vettori di appoggio
radici=ones(4,1);
iterazioni=ones(4,1);

%Metodo Secanti
disp("Metodo Secanti");
for i=1:length(toll)
    [x_sec,it_sec]=sec(x0,x1,f,toll(i),1000);
    radici(i)=x_sec;
    iterazioni(i)=it_sec;
end
t=table(radici,iterazioni,toll,'VariableNames',{'Radici','Iterazioni','Tolleranza'
    });
disp(t);

%Metodo Bisezione
disp("Metodo Bisezione");
for i=1:length(toll)
    [x_b,it_b]=bise(a,b,f,toll(i));
    radici(i)=x_b;
    iterazioni(i)=it_b;
end
t=table(radici,iterazioni,toll,'VariableNames',{'Radici','Iterazioni','Tolleranza'
    });
disp(t);

%Metodo Newton
disp("Metodo Newton");
for i=1:length(toll)
    try
        [x_n, it_n]=newtonZeri(x0,f,f1,toll(i),1000);
        radici(i)=x_n;
        iterazioni(i)=it_n;
    catch ME
        % Stampa il messaggio di errore
        disp(['Errore: ', ME.message]);
        radici(i) = NaN;
        iterazioni(i) = NaN;
    end
end
t=table(radici,iterazioni,toll,'VariableNames',{'Radici','Iterazioni','Tolleranza'
    });
disp(t);

%Metodo Newton Modificato
disp("Metodo Newton Modificato");
for i=1:length(toll)
    try
```

```
        [x_nm,it_nm]=newtonModificato(x0,f,f1,5,toll(i),1000);
        radici(i)=x_nm;
        iterazioni(i)=it_nm;
    catch ME
        % Stampa il messaggio di errore
        disp(['Errore: ', ME.message]);
        radici(i) = NaN;
        iterazioni(i) = NaN;
    end
end
t=table(radici,iterazioni,toll,'VariableNames',{'Radici','Iterazioni','Tolleranza'
    });
disp(t);
```

Listing 7: Codice per ottenere i risultati

Per la tolleranza=10^{-3}

Metodo	**Radici**	**Iterazioni**
secanti	0.005576	33
bisezione	0.0375	3
newton	0.0039218	25
newton modificato	*NaN*	*NaN*

Per la tolleranza=10^{-6}

Metodo	**Radici**	**Iterazioni**
secanti	-0.0010403	61
bisezione	0.003125	5
newton	*NaN*	*NaN*
newton modificato	*NaN*	*NaN*

Per la tolleranza=10^{-9}

Metodo	**Radici**	**Iterazioni**
secanti	-0.001075	89
bisezione	0.0011163	31
newton	*NaN*	*NaN*
newton modificato	*NaN*	*NaN*

Per la tolleranza=10^{-12}

Metodo	**Radici**	**Iterazioni**
secanti	-0.0010751	123
bisezione	0.0011163	32
newton	*NaN*	*NaN*
newton modificato	*NaN*	*NaN*

- **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 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 risulatati 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 bisenzione 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'efficenza della *function*, e valudarla su un congruo numero di esempi significativi, che evidenzino tutti i suoi possibili *output*.

8.1 Soluzione

```
function x = mialu(A, b)
    % x=mialu(A,b) Metodo di fattorizzazione LU con pivoting parziale
    % Input: -A: matrice nxn
    %        -b: vettore dei termini noti
    % Output: -x: soluzione del sistema Ax=b

    %vari controlli
    [m,n]=size(A);
    if m~=n, error('La matrice deve essere quadrata'), end;
    if length(b)~=n, error('Dimensione del vettore b sbagliata'), end;

    for i=1:n-1
        if A(i,i) == 0
            error('Matrice singolare');
        end
        A(i+1:n,i) = A(i+1:n,i) / A(i,i);
        A(i+1:n,i+1:n) = A(i+1:n,i+1:n) - A(i+1:n,i) * A(i,i+1:n);
    end

    x = b(:);
    % Risoluzione per L
    for i=2:n
        x(i:n) = x(i:n) - A(i:n,i-1) * x(i-1);
    end

    % Risoluzione per U
    for i=n:-1:1
        x(i) = x(i) / A(i,i);
        x(1:i-1) = x(1:i-1) - A(1:i-1,i) * x(i);
    end

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 LDT^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

```
function x = mialdl(A,b)
%
% x = mialdl(A,b)
%
% Dati in ingresso una matrice sdp A ed un vettore b, calcolare la
% soluzione del corrispondente sistema lineare utilizzando la
% fattorizzazione LDL^t
%
% input: - A: matrice dei coefficienti
%        - b: vettore dei termini noti
%
% output: x - soluzione del sistema Ax=b
%

[m,n] = size(A);
if m~=n
error('La matrice A deve essere quadrata. Verifica le dimensioni di A');
end
if m~=length(b)
error('Le dimensioni della matrice A e del vettore b non sono compatibili');
end
if A(1,1)<=0
error('La matrice A deve essere simmetrica e definita positiva');
end

% Fattorizzazione LDL^t
A(2:n,1)=A(2:n,1)/A(1,1);
for i=2:n
v= (A(i,1:i-1).').* diag(A(1:i-1,1:i-1));
A(i,i) = A(i,i) - A(i,1:i-1)*v;
if A(i,i)<=0
    error('La matrice A deve essere simmetrica e definita positiva');
end
A(i+1:n,i) = (A(i+1:n,i)- A(i+1:n,i:i-1)*v)/A(i,i);
end

```

```
%Risoluzione del sistema lineare
x=b;
for i=1:n
x(i+1:n) = x(i+1:n)-(A(i+1:n,i)*x(i));
end
x = x ./ diag(A);
for i=n:-1:2
x(1:i-1) = x(1:i-1)-A(i,1:i-1).'*x(i);
end
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 mxn, con $m \geq n = rank(A)$, 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

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

```
%         b- vettore dei termini noti
% Output: x-soluzione del sistema Ax=b
%          nr- norma del vettore residuo
[m,n]=size(A);
for i=1:n
alfa=norm(A(i:m,i));
if alfa==0
    error('Matrice non a rango massimo');
end
if length(b)~=m, error('Dimensione del vettore b sbagliata'), end;
if A(i,i)>=0
    alfa=-alfa;
end
v1=A(i,i)-alfa;
A(i,i)=alfa;
A(i+1:m,i)=A(i+1:m,i)/v1;
beta=-v1/alfa;
A(i:m,i+1:n)=A(i:m,i+1:n)-(beta*[1;A(i+1:m,i)])*...
    ([1 A(i+1:m,i)']*A(i:m,i+1:n));
b(i:m)=b(i:m)-(beta*[1 A(i+1:m,i)']*b(i:m))*...
    [1;A(i+1:m,i)];
end
%risoluzione sistema Ax=b
x=b(:);
for i=n:-1:1
x(i)=x(i)/A(i,i);
x(1:i-1)=x(1:i-1)-A(1:i-1,i)*x(i);
end
%norma del vettore residuo
nr=norm(x(n+1:m));

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 $b = \begin{bmatrix} 10 \\ 11 \\ 12 \end{bmatrix}$ sappiamo che $x = A^{-1}b$ 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 $b = \begin{bmatrix} 10 \\ 11 \\ 12 \\ 13 \end{bmatrix}$ Il programma riconosce che la dimensione del vettore b 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,

$$A_n x_n = b_n,\ n = 1, \ldots, 15$$

in cui

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

la cui soluzione è il vettore $\mathbf{x}_n = (1, \ldots, 1)^T \in \mathbb{R}^n$, utilzzando 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

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

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

12.1 Soluzione

```

valori_n = 1:100;
max=length(valori_n);
for k = 1:max
    n = valori_n(k);
    A = ones(n);
%eseguo 2 for annidati per poter scorrere tutta la matrice A
    for i = 1:n
        for j = 1:n
            if (i ~= j)
                A(i,j) = -A(i,j);
            else
                A(i,j) = n;
            end
        end
    end
%con questo for modifico la matrice in modo da avere gli elementi diagonali
%con n e gli elementi che non sono diagonali li nego
    A = mialdlt(A);
    diagonal_elements = diag(A);

    if  n==1||mod(n,1)==0
        plot(1:n, diagonal_elements,'.');
        hold on;
    end
end

title('Esercizio 12');
xlabel('Indice dellla diagonale');
ylabel('Valore dei fattori di D');
```

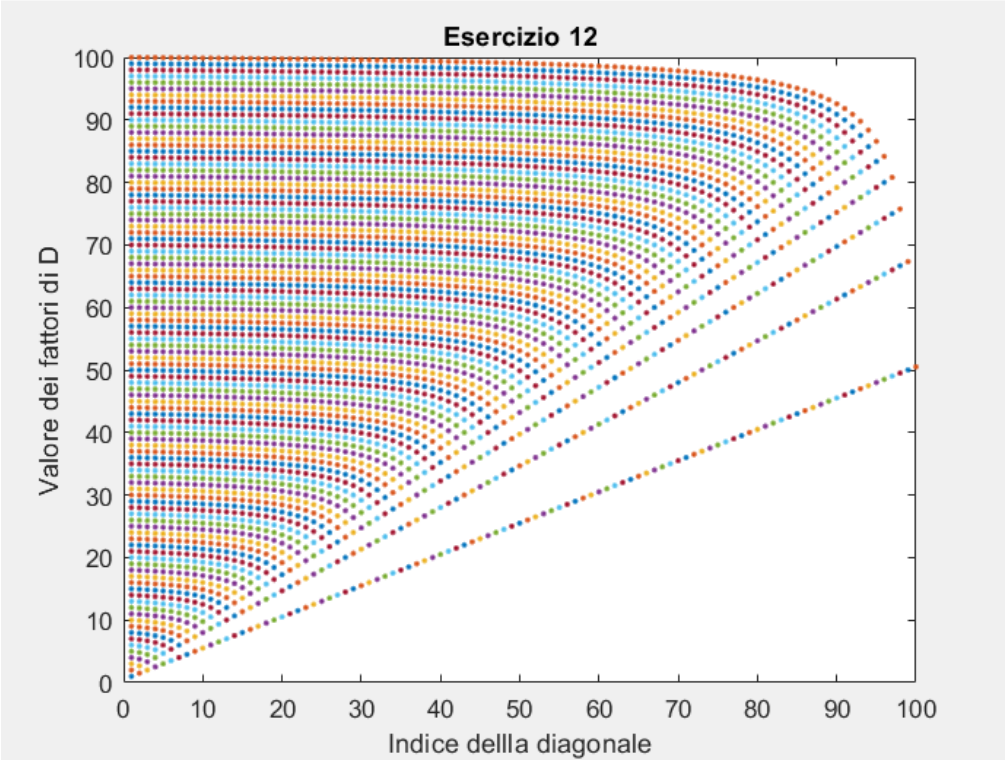
Listing 11: Codice del metodo per graficare gli elementi diagonali di D

```
    function A = mialdlt(A)
%
%   A = mialdlt(A)
%
%   Data in input una matrice A sdp restituisce A contenente gli elementi L
%   e D

[m,n] = size(A);
if A(1,1)<=0
    error('Errore: La matrice deve essere sdp');
end
if m~=n
    error("Errore: La matrice deve essere quadrata");
end

A(2:n,1) = A(2:n,1)/A(1,1);

for j=2:n
    v = (A(j,1:j-1).').*diag(A(1:j-1,1:j-1));
    A(j,j) = A(j,j)-A(j,1:j-1)*v;
    if A(j,j)<=0,error('La matrice deve essere sdp.'),end;
```

```
    A(j+1:n,j) = (A(j+1:n,j)-A(j+1:n,1:j-1)*v)/A(j,j);
end

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 $\mathbf{r}=(r_1, \ldots, r_5)^T$

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

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

```
% Inizializzazione dei dati
A = [7 2 1; 8 7 8; 7 0 7; 4 3 3; 7 0 10];
b = [1; 2; 3; 4; 5];

% Vettore dei pesi
omega = [0.5; 0.5; 0.75; 0.25; 0.25];

% Normalizzazione dei pesi in modo che la somma sia 1
omega = omega ./ 2.25;

% Calcolo della matrice dei pesi B
B = diag(sqrt(omega));

% Risoluzione del sistema utilizzando QR
[x, nr] = miaqr(B*A, B*b);

disp("Soluzione trovata per il sistema sovradeterminato:");
disp(x);

disp("Norma del vettore residuo:");
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,xo, 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: |Xn+1 - Xn| <= tol * (1 + |Xn|)
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
```

```
F = eye(50) * 4;
for i = 1:49
    F(i, i+1) = 1;
    F(i+1, i) = 1;
end
e = ones(50,1);
a = 2;
b = -1.1;
grad = @(x) F * x - a * e .* sin(a * x) - b * e .* exp(-x);
jacob = @(x) F - a^2 * diag(e .* cos(a * x)) + b * diag(e .* exp(-x));

f=grad(x);
jacobian=jacob(x);

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:

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

$$Q = \begin{pmatrix} 4 & 1 & & \\ 1 & \ddots & \ddots & \\ & \ddots & \ddots & 1 \\ & & 1 & 4 \end{pmatrix} \in \mathbb{R}^{50x50}, \qquad \alpha = 2, \quad \beta = -1.1,$$

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 convenient i risultati ottenuti.

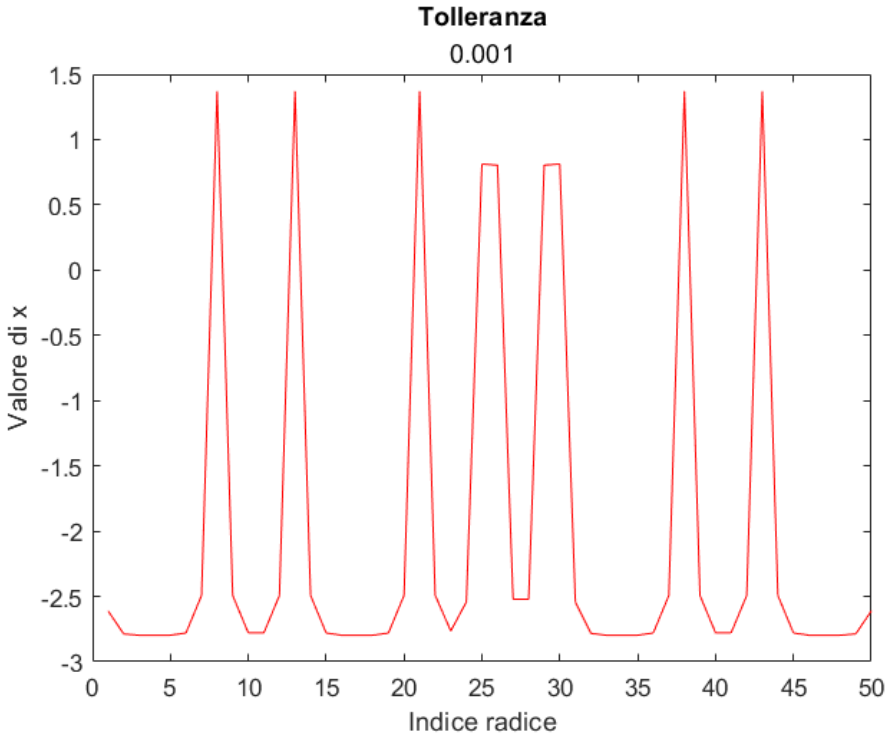
15.1 Soluzione

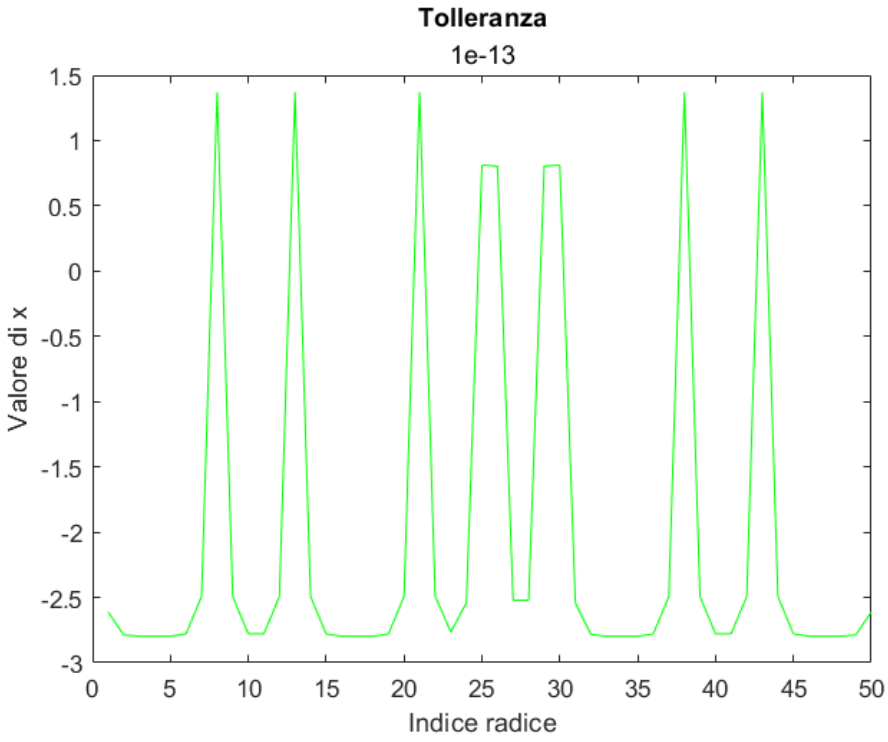
Di seguito è riportato il codice usato per graficare la soluzione

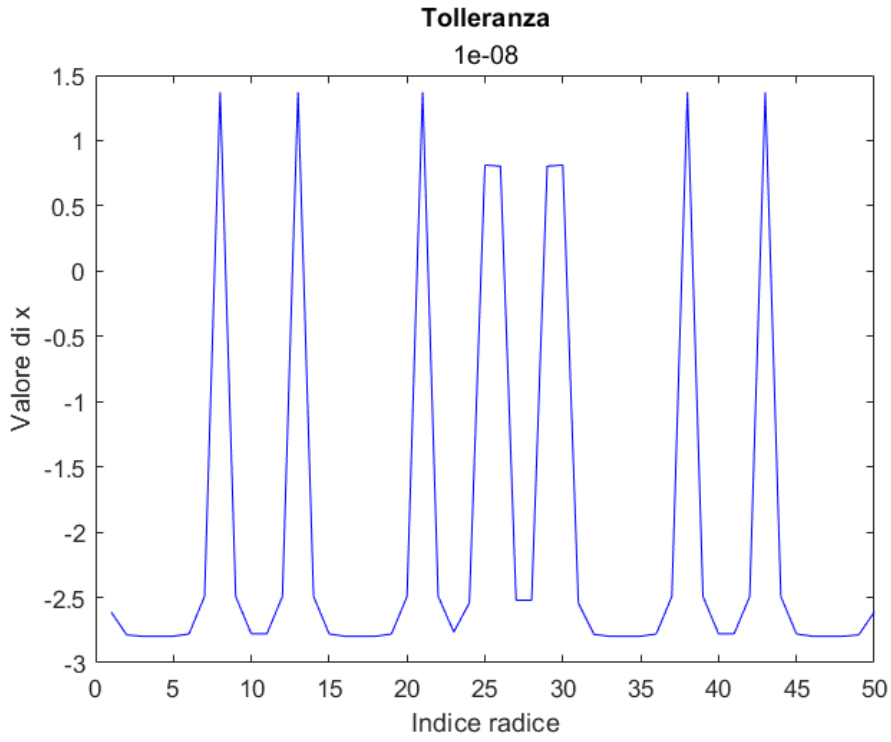
```
format long
n = 50;
x0 =zeros(n,1);
toll = [1e-3, 1e-8, 1e-13];
c = ['r', 'b', 'g'];
L = zeros(50, 3);
for i = 1:length(toll)
    [x,nit] = newton(@fun, x0, toll(i),1000);
    figure;
    plot(1:n, x, '-', Color=c(i));
    title('Tolleranza',num2str(toll(i)));
    xlabel('Indice radice');
    ylabel('Valore di x');
    L(:,i) =x;
end
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

```
function p = lagrange(X,Y,XQ)
% p = lagrange(X,Y,XQ)
%
% function che implementa la forma di lagrange del polinomio
% interpolante una funzione
%
% input:X- vettore delle coordinate x
%       Y- valori della funzione alle coordinate x
%       XQ- vettore dei punti in cui calcolare il polinomio
%
% output:p- polinomio interpolante in forma di Lagrange
%
n= length(X);
if length(Y)~= n
error('Le dimensioni di X e Y non sono corrette');
end
if length(unique(X))~= n
error('Presenza di ascisse uguali');
end
if nargin<3
    error('numero paramentri errato');
end
```

```
p= zeros(size(XQ));
for i= 1:n
    L= ones(size(XQ));
    for j= 1:n
        if j~= i
        L= L .* (XQ - X(j)) / (X(i) - X(j));
        end
    end
    p= p + Y(i) * L;
end
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

```
function pn = newton(X, Y, XQ)
% pn = newton(X, Y, XQ)
%
% function che implementa la forma di Newton del polinomio
% interpolante una funzione
%
% Input: X - vettore delle ascisse
%        Y - vettore dei valori della funzione
%        XQ - vettore dei punti in cui calcolare il polinomio
%
% Output: pn - polinomio interpolante in forma di Newton
%
if length(X) ~= length(Y) || length(X) <= 0
    error('Dati non corretti');
end

% Verifica che i valori in X siano distinti
if length(unique(X)) ~= length(X)
    error('I valori di X devono essere distinti');
end

% Calcola le differenze divise usando la function differenzeDivise
dd = differenzeDivise(X, Y);
n = length(dd) - 1;

pn = dd(n+1) * ones(size(XQ));

% Costruisci il polinomio interpolante di Newton
for i = n:-1:1
    pn = pn.*(XQ- X(i)) + dd(i);
end
return
end
```

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

```
function dd = differenzeDivise(x, f)
% dd = differenzeDivise(x, f)
%
% function che calcola le differenze divise in (x_i, f_i)
%
% Input: x - vettore delle ascisse
%        f - vettore delle ordinate
%
% Output: dd - vettore delle differenze divise
%
% Numero di dati nel vettore x
k = length(x);
if length(f) ~= k
    error('I vettori x e f devono avere la stessa dimensione');
end

k = k - 1;

% Inizializza il vettore delle differenze divise con i valori di f
dd = f;

% Calcola le differenze divise
for j = 1:k
    for i = k+1:-1:j+1
        % Formula per calcolare le differenze divise
        dd(i) = (dd(i) - dd(i-1)) / (x(i) - x(i-j));
    end
end
```

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

18 Esercizio 18

Costruire una function `hermite.m`, avente sintassi

$$\textbf{yy = hermite(xi, fi, f1i, xx)}$$

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

```
function yy = hermite(xi,fi,f1i,xx)
% yy = hermite(xi,fi,f1i,xx)
% funzione che implementa la forma di hermite del polinomio interpolante una
    funzione
%
% input:xi- vettore delle coordinate x
%       fi- valori della funzione alle coordinate x
%       f1i- valori delle derivate della funzione nei punti x
%       xx- vettore dei punti in cui il polinomio interpolante va calcolato
%
% output:yy- polinomio interpolante in forma di Newton

n= length(xi);
if n~= length(fi) ||length(f1i)~= n || n<= 0
```

```
    error("Dimensioni dei dati in input errate");
end
if length(unique(xi))~=n
error("Valori delle ascisse non distinti tra di loro");
end
fi = repelem (fi,2);
for i= 1: length(f1i)
    fi(i*2) = f1i(i);
end
%calcolo delle differenze divise
n=length(xi)-1;
for i=(2*n+1): -2: 3
    fi(i)= (fi(i)- fi(i-2))/(xi((i+1)/2) - xi((i-1)/2));
end
for i= 2: 2*n+1
    for i=(2*n+2): -1: i+1
        fi(i)=(fi(i)- fi(i-1)) / ( xi(round(i/2))-xi(round((i-i)/2)));
    end
end
%calcolo polinomio interpolante
n=length(fi)-1;
yy= fi(n+1)*ones(size(xx));
for i = n : -1:1
    yy= yy .*(xx- xi(round(i/2)))+fi(i) ;
end
return
end
```

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

19 Esercizio 19

Si consideri la seguente base di Newton,

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

con $x_0, \ldots, x_n$ 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:

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

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

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

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.

```
function derivataP = horner(ascisse, coeff, xi)
%
% horner(ascisse, coeff, xi)
% Determina la derivata di un polinomio in forma di Newton in un punto dato
%
% Input:
%   ascisse - Vettore delle ascisse [x0,x1,...,xn]
%   coeff - Vettore dei coefficienti [a0,a1,...,an]
%   xi - punto in cui calcolare la derivata
% Output:
%   derivataP - Derivata del polinomio calcolata in xi
%
n = length(coeff) - 1;
%Imposta i valori iniziali per il polinomio P e la sua derivata derivataP
P = coeff(n+1);
derivataP = 0;

for k = n:-1:1
    derivataP = derivataP * (xi - ascisse(k)) + P;
    P = P * (xi - ascisse(k)) + coeff(k);
end
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

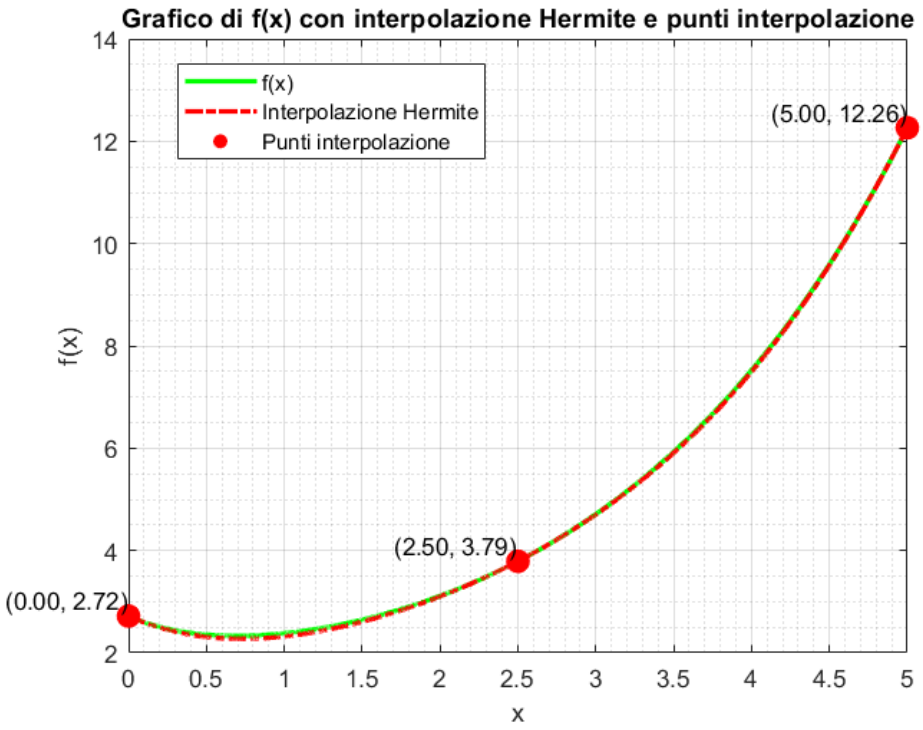
```
ascisse=[0,2.5,5];
a=0;
b=5;
f = @(x) exp(x/2 + exp(-x));
df = @(x) exp(x/2 + exp(-x)) .* (1/2 - exp(-x));
d2f = @(x) exp(x/2 + exp(-x)) .* ((1/2 - exp(-x)).^2 + exp(-x));

%genero i valori di x nell'intervallo [a,b] e calcolo i valori delle
%funzioni
x = linspace(a,b, 10001);
y = f(x);
dy = df(x);
d2fi = d2f(ascisse);

fi=f(ascisse);
dfi=df(ascisse);
%calcolo l'interpolazioni
```

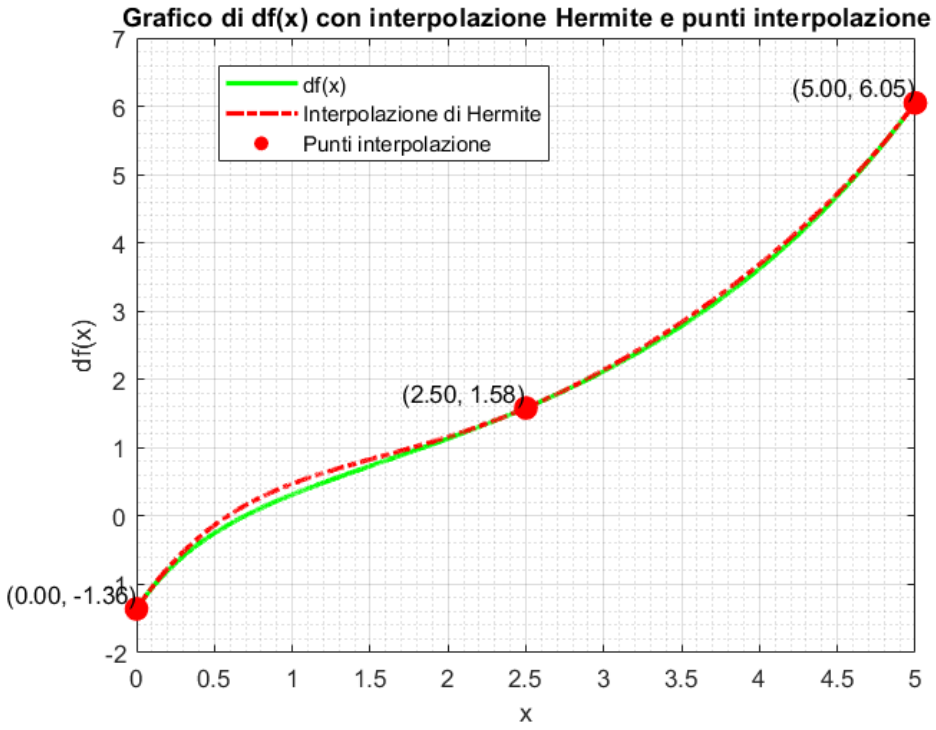
```
yy=hermite(ascisse,fi,dfi,x);

% Grafico di f(x)
figure;
plot(x, y, 'g', "LineWidth", 2);
hold on;
plot(x, yy, 'r-.', "LineWidth", 2);
set(gca, 'XMinorGrid', 'on');
set(gca, 'YMinorGrid', 'on');
%evidenzio i punti di interesse
scatter(ascisse, fi, 100, 'r', 'filled');
for i = 1:length(ascisse)
    text(ascisse(i), fi(i), sprintf('(%0.2f, %0.2f)', ascisse(i), fi(i)), ...
        'VerticalAlignment', 'bottom', 'HorizontalAlignment', 'right');
end
xlabel('x');
ylabel('f(x)');
title('Grafico di f(x) con interpolazione Hermite e punti interpolazione');
legend('f(x)', 'Interpolazione Hermite', 'Punti interpolazione');
grid on;
zoom on;

% Grafico della derivata prima
figure;
plot(x, dy, "g", "LineWidth", 2);
hold on;
set(gca, 'XMinorGrid', 'on');
set(gca, 'YMinorGrid', 'on');
% Calcolo delle differenze divise per Hermite
coeff = repelem(fi, 2);
for i = 1:length(dfi)
    coeff(i*2) = dfi(i);
end
n = length(ascisse) - 1;
for j = (2*n+1):-2:3
    coeff(j) = (coeff(j) - coeff(j-2)) / (ascisse((j+1)/2) - ascisse((j-1)/2));
end
for j = 2:2*n+1
    for i = (2*n+2):-1:j+1
        coeff(i) = (coeff(i) - coeff(i-1)) / (ascisse(round(i/2)) - ascisse(round((
            i-j)/2)));
    end
end

ascHerm = repelem(ascisse, 2);

y1herm = horner(ascHerm, coeff, 0);
y2herm = horner(ascHerm, coeff, 2.5);
y3herm = horner(ascHerm, coeff, 5);
yherm = [y1herm, y2herm, y3herm];

fherm = hermite(ascisse, yherm, d2fi, x);

plot(x, fherm, "r-.", "LineWidth", 2);

% evidenzio i punti di interesse
scatter(ascisse, dfi, 100, 'r', 'filled');
for i = 1:length(ascisse)
    text(ascisse(i), dfi(i), sprintf('(%0.2f, %0.2f)', ascisse(i), dfi(i)), ...
```

```
        'VerticalAlignment', 'bottom', 'HorizontalAlignment', 'right');
end

xlabel('x');
ylabel('df(x)');
title('Grafico di df(x) con interpolazione Hermite e punti interpolazione');
legend('df(x)', 'Interpolazione di Hermite', 'Punti interpolazione');
grid on;
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

```
function x = chebyshev(n,a,b)
% x = chebyshev(n,a,b)
% funzione per il calcolo delle ascisse di Chebyshev
%
% input: n- grado del polinomio
%        [a,b]- estremi dell'intervallo
% output: x- vettore contenente le ascisse di chebyshev

if(a>=b)
error('valori di intervallo errati');
end;
if(n<0)
error('grado del polinomio errato');
end;
x = (a+b)/2+((b-a)/2)*cos(pi*(2*(n:-1:0)+1)./(2*(n+1)));
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

```
function ll = lebesgue( a, b, nn, type )
%
% ll = lebesgue( a, b, nn, type)
%Funzione che approssima la costante di Lebesgue
%
% Input:a,b- estremi dell'intervallo, rispettivamente inferiore e superiore
%       nn- grado del polinomio
%       type- o 0 o 1, per il tipo di ascisse di interpolazione da usare
%
% Output:ll- approssimazione della costante di Lebesgue

if nn<= 0
error('grado non valido');
end
if a>= b
error('Estremi errati');
end
if type~=0 && type~=1
error('Valore di type non corretto');
end
fi=linspace(a,b,10001);
ll= zeros(1, length(nn));
for i=1:length(nn)
if type==0
    x = linspace (a,b, nn(i));
else
    x=chebyshev(nn(i),a,b);
end
lan= zeros(1,length(fi));
%calcolo del polinomio di lagrange
for k=1:length(x)
    lin = ones(size(fi));

    for j=1:length(x)
        if k~=j
            lin = lin .*((fi-x(j))/(x(k)-x(j)));

        end
    end
lan = lan + abs(lin);
end
%calcolo della costante di lebesgue
ll(i) = max(lan);
end
end
```

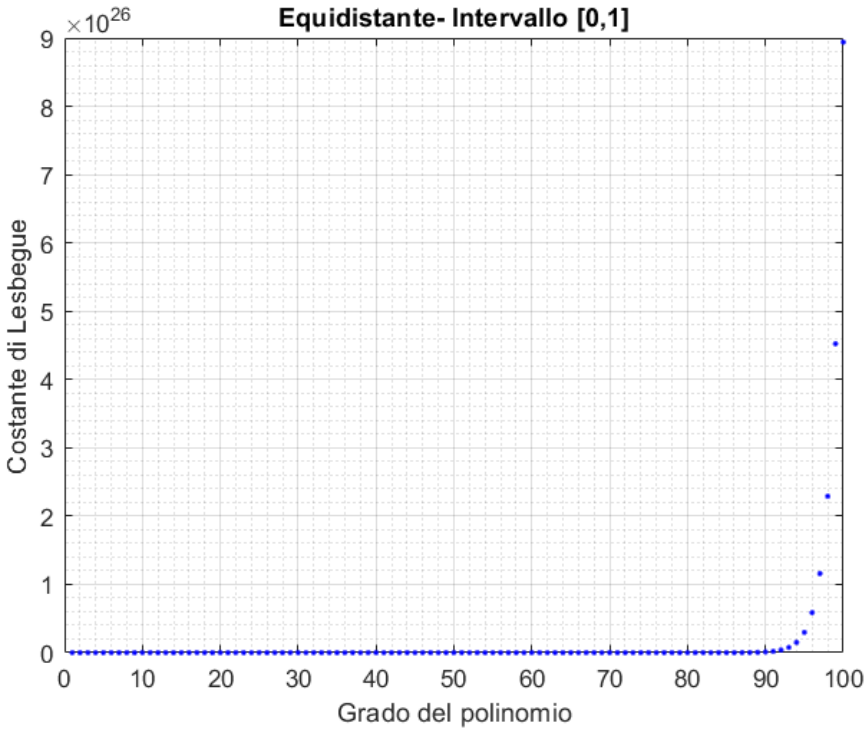
Listing 24: Codice della funzione che approssima la costante di Lesbegue nell'intervallo (a,b)

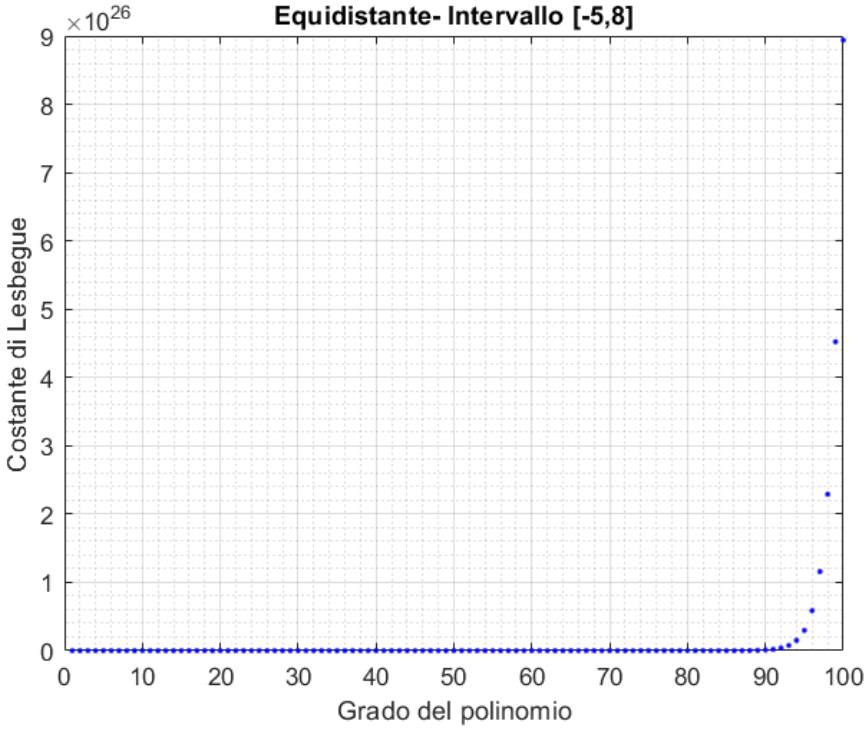
il seguente codice è stato usato per graficare i grafici richiesti

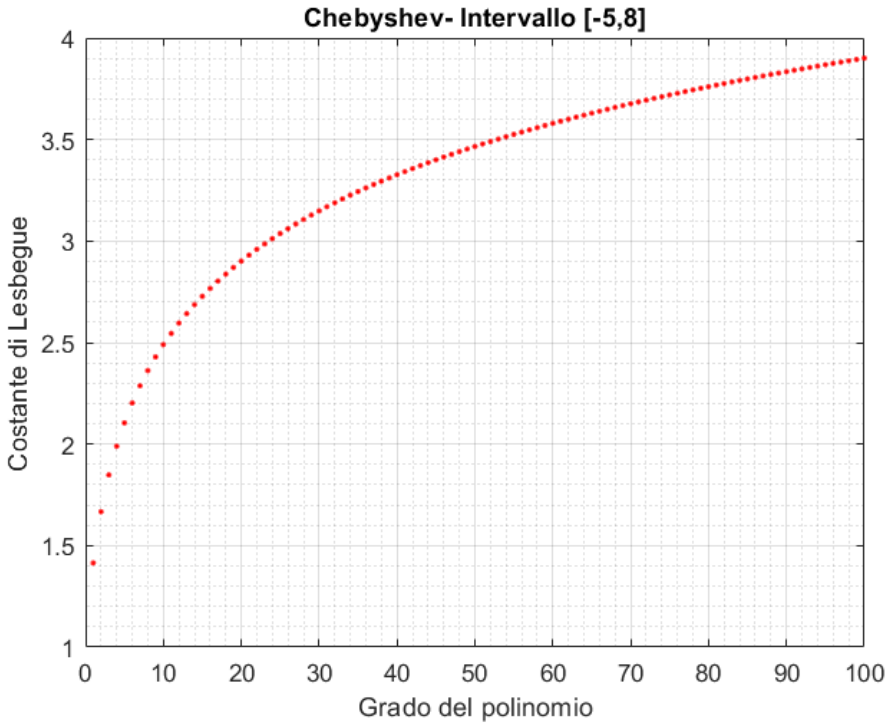
```

nn=1:100;
%costante di Lebesgue per ascisse di Chebyshev
cheb1 = lebesgue(0, 1, nn, 1);
cheb2 = lebesgue(-5, 8, nn, 1);
%costante di Lebesgue per ascisse equidistanti
leb1 = lebesgue(0, 1, nn, 0);
leb2 = lebesgue(-5, 8, nn, 0);

% Grafico Equidistante aintervallo [0,1]
figure;
plot(nn, leb1, 'b');
title('Equidistante- Intervallo [0,1]');
xlabel('Grado del polinomio');
ylabel('Costante di Lesbegue');
grid on;
set(gca, 'XMinorGrid', 'on');
set(gca, 'YMinorGrid', 'on');

% Grafico Equidistante intervallo [-5,8]
figure;
plot(nn, leb2, 'b');
title('Equidistante- Intervallo [-5,8]');
xlabel('Grado del polinomio');
ylabel('Costante di Lesbegue');
grid on;
set(gca, 'XMinorGrid', 'on');
set(gca, 'YMinorGrid', 'on');

% Grafico Chebyshev intervallo [0,1]
figure;
plot(nn, cheb1, 'r');
title('Chebyshev- Intervallo [0,1]');
xlabel('Grado del polinomio');
ylabel('Costante di Lesbegue');
grid on;
set(gca, 'XMinorGrid', 'on');
set(gca, 'YMinorGrid', 'on');

% Grafico Chebyshev intervallo [-5,8]
figure;
plot(nn, cheb2, 'r');
title('Chebyshev- Intervallo [-5,8]');
xlabel('Grado del polinomio');
ylabel('Costante di Lesbegue');
grid on;
set(gca, 'XMinorGrid', 'on');
set(gca, 'YMinorGrid', 'on');
```

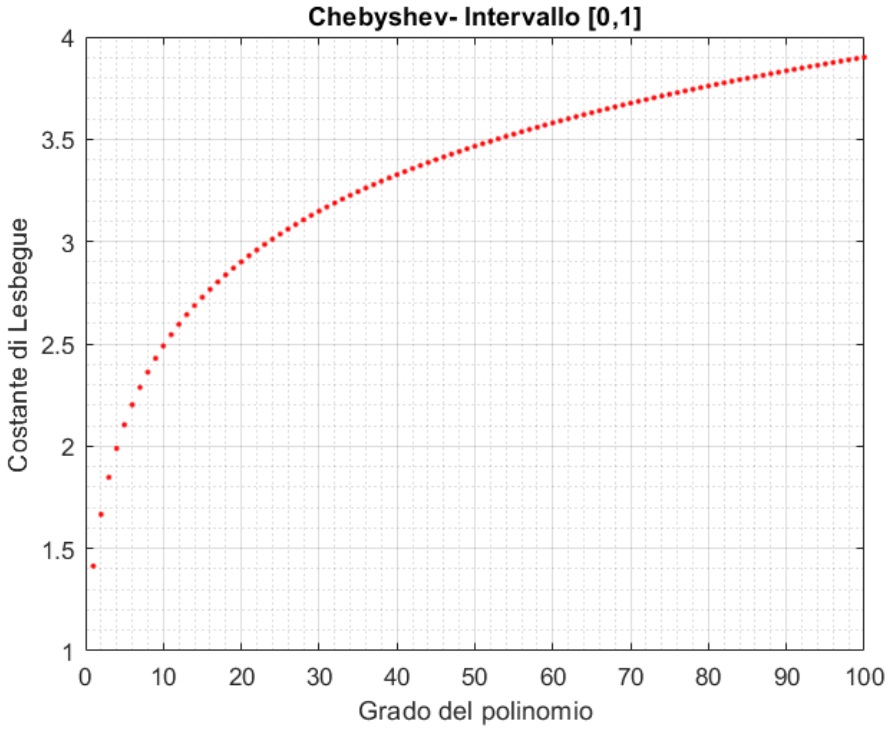
Listing 25: Codice per graficare i grafici richiesti

Di seguito verranno mostrati i grafici risultanti


Equidistante- Intervallo [0,1]
×10^26
Costante di Lesbegue
Grado del polinomio
0
1
2
3
4
5
6
7
8
9
0
10
20
30
40
50
60
70
80
90
100


Equidistante- Intervallo [-5,8]
×10^26
Costante di Lesbegue
Grado del polinomio
0
1
2
3
4
5
6
7
8
9
0
10
20
30
40
50
60
70
80
90
100





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,

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

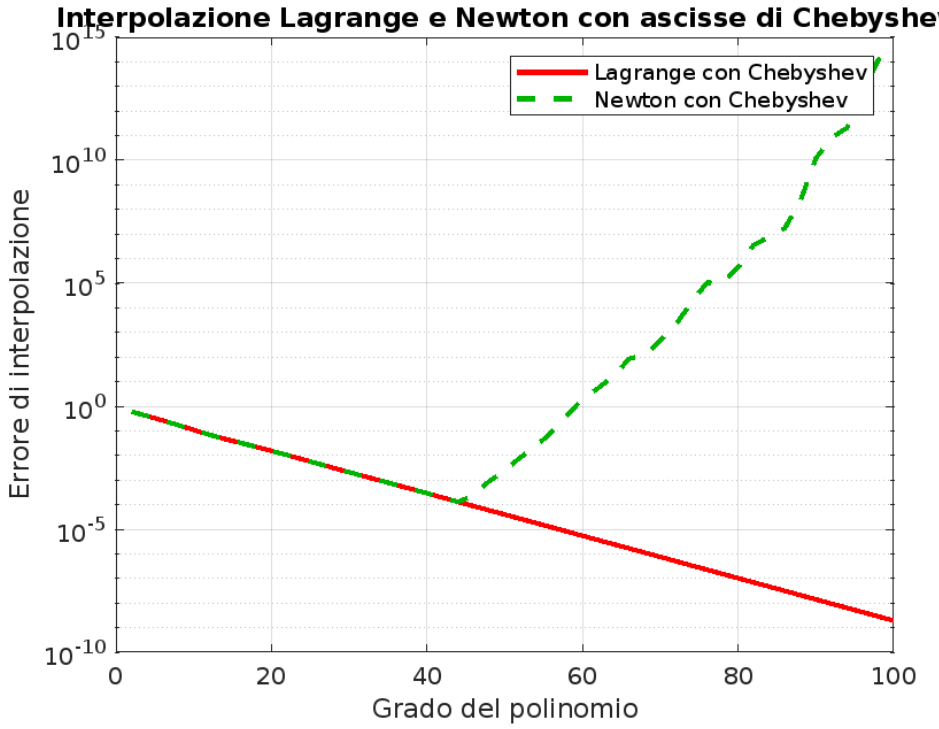
utilizzando le ascisse di Chebyshev, per i polinomi interpolanti di grado `n = 2:2:100`. Commentare i risultati ottenuti.

23.1 Soluzione

```
% Funzione di Runge
funzione_runge = @(x) 1./(1+x.^2);

% Intervallo di interesse
inizio_intervallo = -5;
fine_intervallo = 5;

punti_equispaziati = 10001;
xq = linspace(inizio_intervallo, fine_intervallo, punti_equispaziati);
gradi_polinomio = 2:2:100;

% Memorizzare gli errori
errori = struct('Lagrange', zeros(size(gradi_polinomio)), 'Newton', zeros(size(
    gradi_polinomio)));

% Calcolo dei valori esatti della funzione di Runge
valori_esatti = funzione_runge(xq);

% Ciclo per calcolare gli errori
for i = 1:length(gradi_polinomio)
    n = gradi_polinomio(i);

    % Calcolo delle ascisse di Chebyshev
    ascisse = chebyshev(n, inizio_intervallo, fine_intervallo);

    % Calcolo dei valori approssimati usando l'interpolazione di Lagrange e Newton
        con ascisse di Chebyshev
    val_lagrange = lagrange(ascisse, funzione_runge(ascisse), xq);
    val_newton = newton(ascisse, funzione_runge(ascisse), xq);

    % Calcolo dell'errore
    error_lagrange(i) = max(abs(valori_esatti - val_lagrange));
    error_newton(i) = max(abs(valori_esatti - val_newton));
end

% Grafico dell'errore in scala logaritmica
figure;

semilogy(gradi_polinomio, error_lagrange, '-r', 'LineWidth', 2);
hold on;
colore_verde = [0, 0.7, 0];
semilogy(gradi_polinomio, error_newton, '--', 'Color', colore_verde, 'LineWidth',
    2);

xlabel('Grado del polinomio');
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 `(xi,fi)`.

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

24.1 Soluzione

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

```
% yy = spline0(x, y, xq)
%funzione che calcola la spline cubica naturale nei punti di interpolazione
%
% input: x-vettore delle x
%        y- valori che prende la funzione nelle x
%        xq- vettore dei punti in cui va calcolato il polinomio
% output: yy-valori della spline nei punti xq
if nargin < 3
    error('parametri in ingresso insufficienti');
end
n = length(x);
if n ~= length(y)
    error('parametri in ingresso errati');
end
n = n - 1;
h = zeros(1, n);
for i = 1:n
    h(i) = x(i+1) - x(i);
end
dd = y;
% calcolo differenze divise
for j = 1:2
    for i = n+1:-1:j+1
        dd(i) = (dd(i)- dd(i-1)) / (x(i) - x(i - j));
    end
    if j == 1
        dd1f = dd(2:end);
        % dd1f contiene le differenze divise f[x(k-1),x(k)], k = 2:n
    end
end
phi = h(2:n-1) ./ (h(2:n-1) + h(3:n));
csi = h(2:n-1) ./ (h(1:n-2) + h(2:n-1));
a(1:n-1) = 2;
m = zeros(1, n + 1);
m(2:n) = trilu(a, phi, csi, 6 * dd(3:end));
yy = zeros(1,length(xq));
for j = 1:length(xq)
    if xq(j) >= x(1) && xq(j) <= x(n+1)
        for k=2:n+1
            if xq(j) <= x(k)
                i = k-1;
                break
            end
        end
        ri = y(i) - (h(i) ^ 2) / 6 * m(i);
        qi = dd1f(i) - h(i) / 6 * (m(i+1) - m(i));
        yy(j) = ((xq(j) - x(i)) ^ 3 * m(i+1) + (x(i+1) - xq(j)) ^ 3 * m(i)) ...
            / (6 * h(i)) + qi * (xq(j) - x(i)) + ri;
    end
end
return
```

Listing 27: Codice della funzione spline0

```
function x = trilu(a, b, c, x)
% Risolve un sistema lineare tridiagonale fattorizzabile LU.
%
% x = trilu(a, b, c, x)
%
```

```
% Input: a- vettore degli elementi diagonali
%        b- vettore degli elementi sottodiagonali
%        c- vettore degli elementi sopradiagonali
%        x- vettore dei termini noti
%
% Output: x- soluzione
%
n = length(a);
for i = 1:n-1
b(i) = b(i)/a(i);
a(i+1) = a(i+1) - b(i)*c(i);
x(i+1) = x(i+1) - b(i)*x(i);
end
x(n) = x(n)/a(n);
for i = n-1:-1:1
x(i) = (x(i) - c(i)*x(i+1))/a(i);
end
return
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

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

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

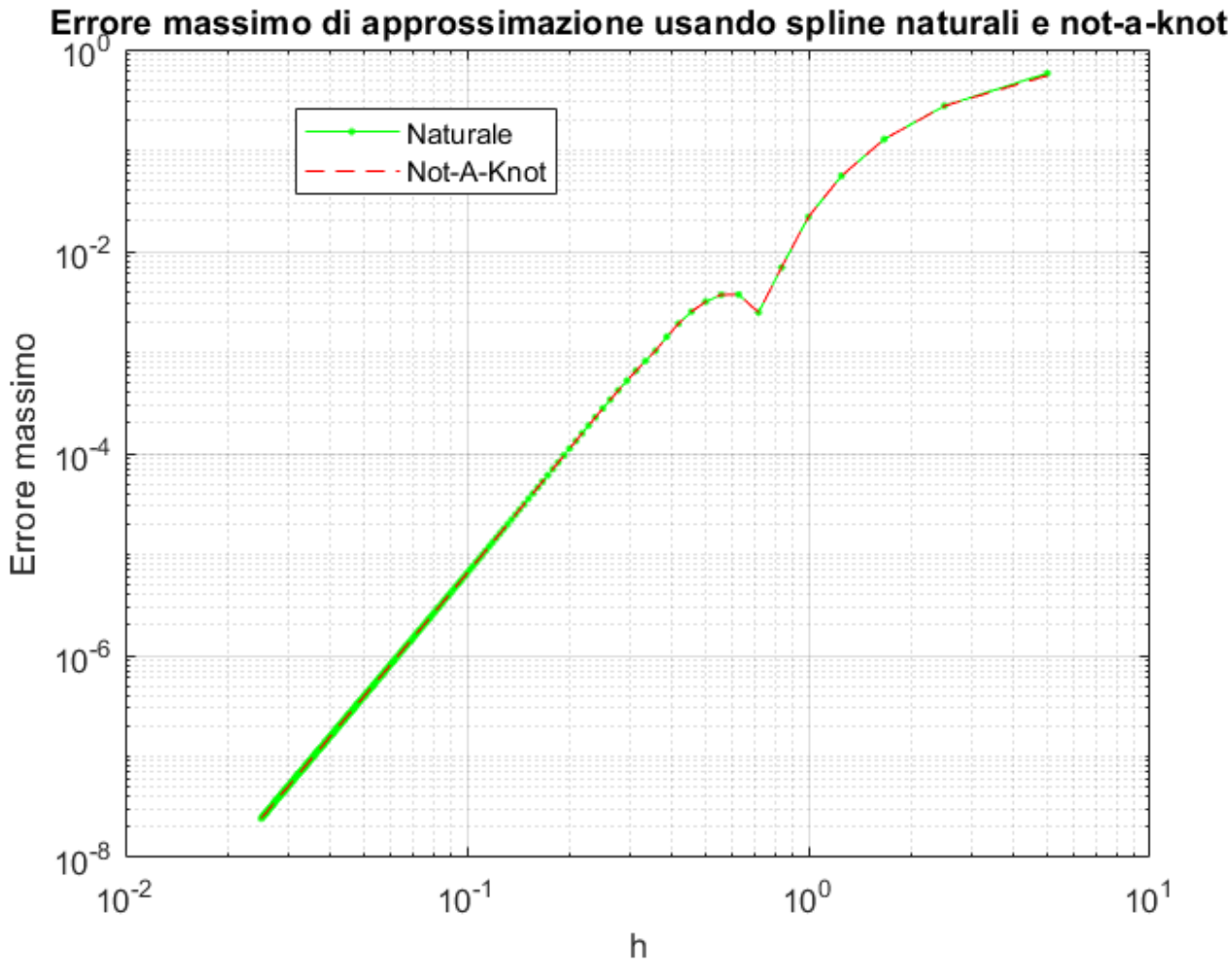
Di seguito è presente il codice usato per graficare le spline:

```
f = @(x) 1 ./ (1 + x.^ 2);
a = -10;
b = 10;
x = linspace(a, b, 10001);
fx = f(x);
errKnot = zeros(1, 200);
errNaturali = zeros(1, 200);
h = zeros(1, 200);

index = 1;
for n = 4:4:800
    xi = linspace(a, b, n+1); % Ascisse equidistanti
    fi = f(xi);

    % calcolo le spline e i loro rispettivi errori massimi
    splineKnot = spline(xi, fi, x);
    splineNat = spline0(xi, fi, x);

    errKnot(index) = max(abs(fx - splineKnot));
    errNaturali(index) = max(abs(fx - splineNat));
    h(index) = 20 / n;
```

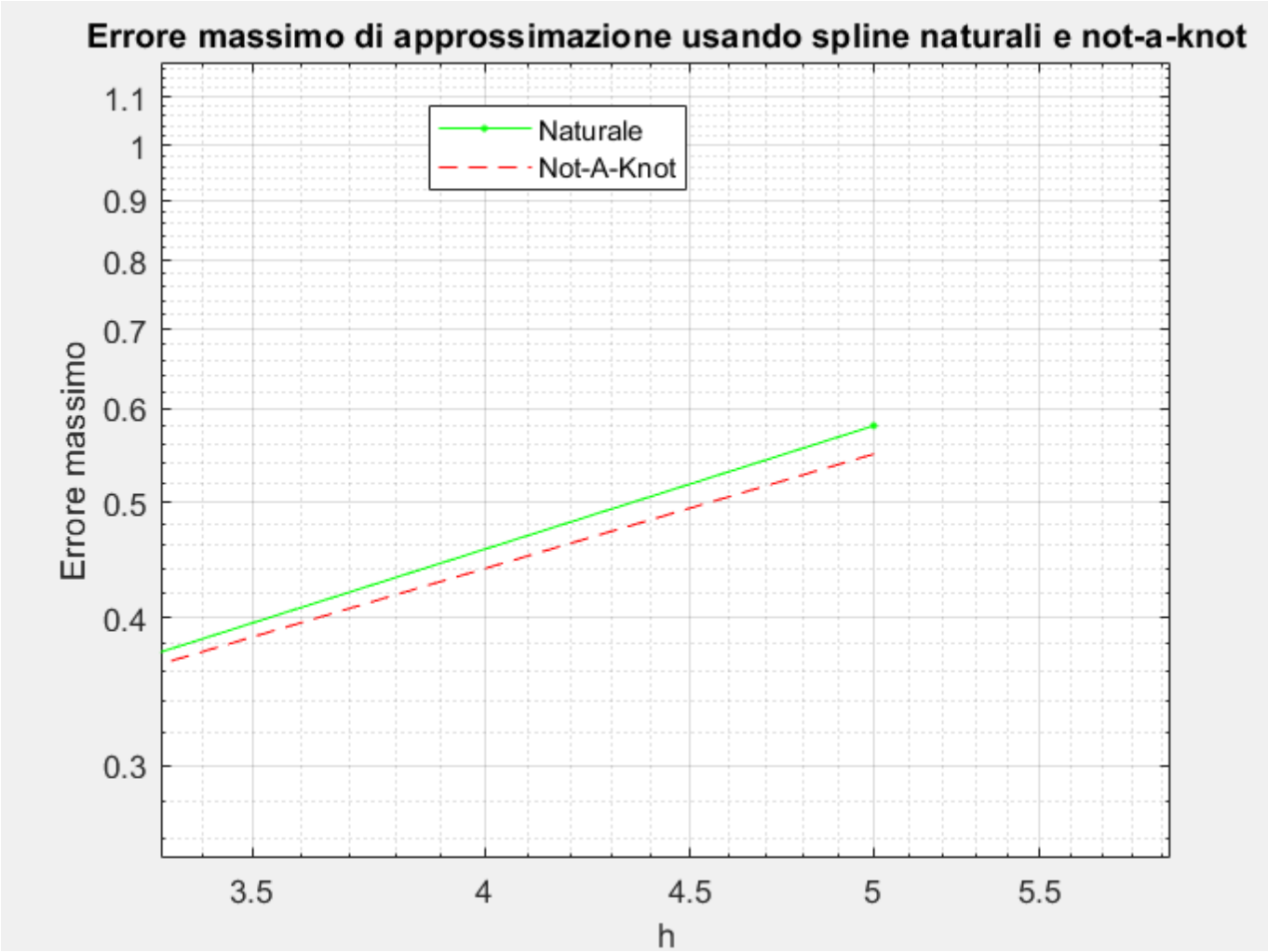
```
    index = index + 1;
end

%Grafico gli errori
figure;
loglog(h, errKnot, 'g.-',h, errNaturali, 'r--' );
xlabel('h');
ylabel('Errore massimo');
title('Errore massimo di approssimazione usando spline naturali e not-a-knot');
legend('Naturale', 'Not-A-Knot');
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 incotrano 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

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

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

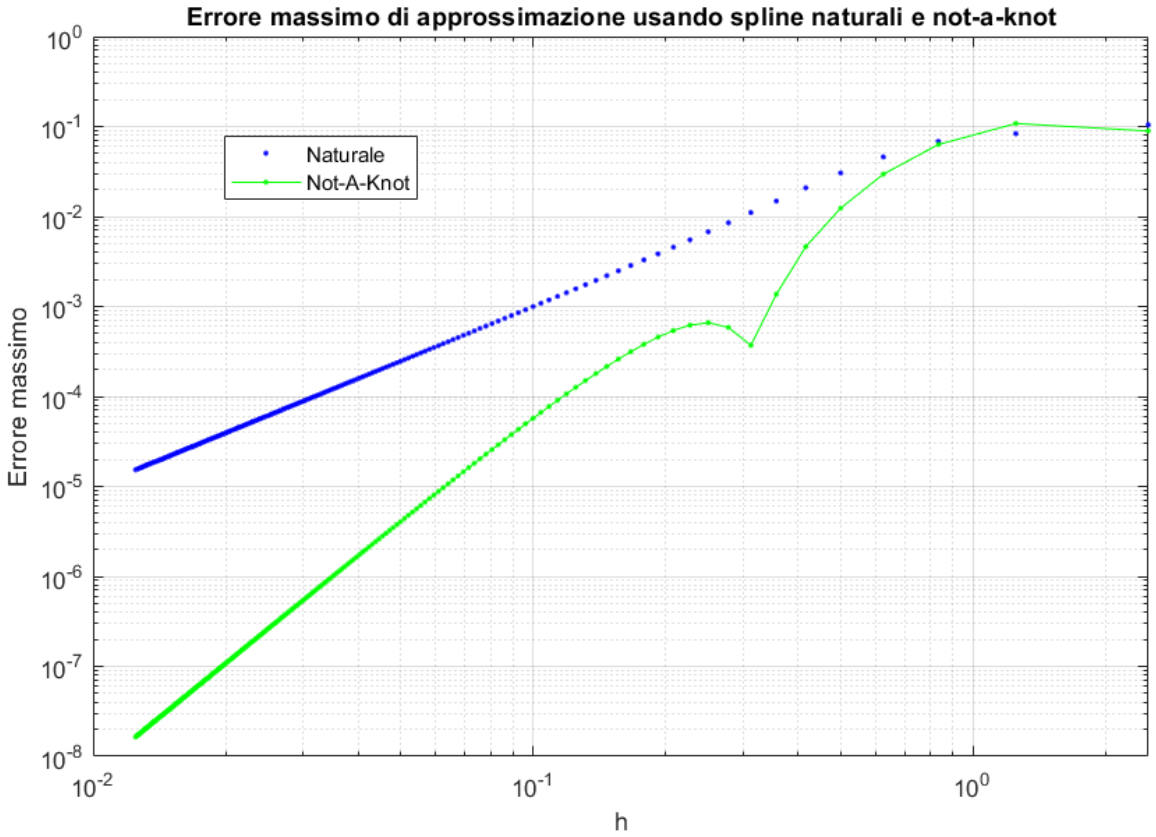
Di seguito è riportato il codice usato per graficare le due spline

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

```
errNaturali = zeros(1, 200);
h = zeros(1, 200);

index = 1; %index = n/4
for n = 4:4:800
    xi = linspace(a, b, n+1); % Ascisse equidistanti
    fi = f(xi);

    % calcolo le spline e i loro rispettivi errori massimi
    splineKnot = spline(xi, fi, x);
    splineNat = spline0(xi, fi, x);

    errKnot(index) = max(abs(fx - splineKnot));
    errNaturali(index) = max(abs(fx - splineNat));
    h(index) = 10 / n;
    index = index + 1;
end

%Grafico gli errori
figure;
loglog(h, errNaturali, 'b.', h, errKnot, 'g.-');
xlabel('h');
ylabel('Errore massimo');
title('Errore massimo di approssimazione usando spline naturali e not-a-knot');
legend('Naturale', 'Not-A-Knot');
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

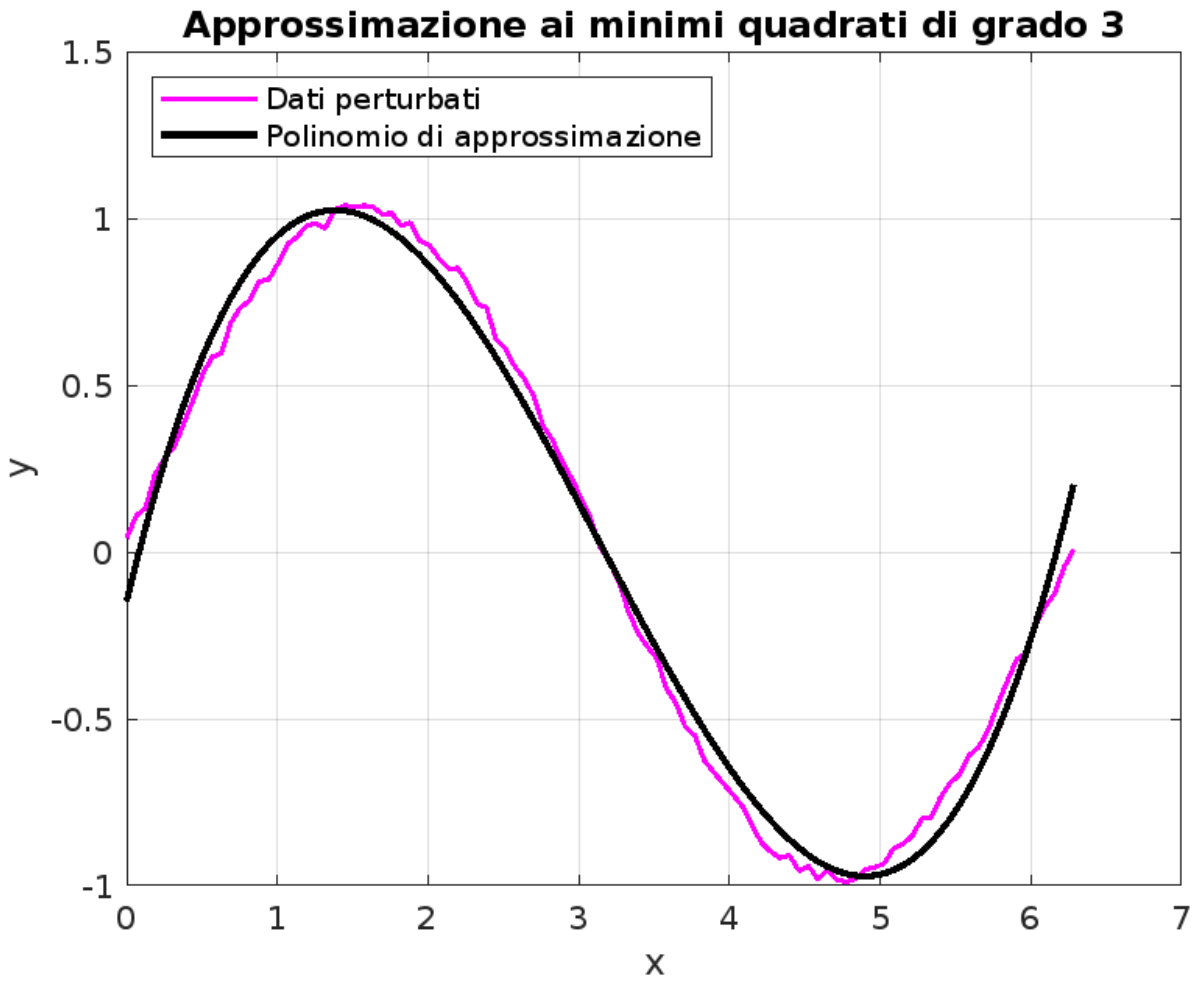
```
rng(0);
xi = linspace(0, 2*pi, 101);
yi = sin(xi) + rand(size(xi)) * 0.05;

% Calcolo dei coefficenti del polinomio di grado 3
grado_polinomio = 3;
coefficenti_minimi_quadrati = polyfit(xi, yi, grado_polinomio);

% Mostra i coefficienti calcolati
disp('Coefficenti del polinomio di approssimazione:');
```

```
disp(coefficenti_minimi_quadrati);

% Valutazione del polinomio nei punti xi
yi_polyval = polyval(coefficenti_minimi_quadrati, xi);

% Grafico dei dati e del polinomio di approssimazione
figure;
plot(xi, yi, 'm-', 'LineWidth', 1.5, 'DisplayName', 'Dati perturbati');
hold on;
plot(xi, yi_polyval, 'k-', 'LineWidth', 2, 'DisplayName', 'Polinomio di
    approssimazione');
xlabel('x');
ylabel('y');
title('Approssimazione ai minimi quadrati di grado 3');
legend('show', 'Location', 'northwest');
grid on;
```

Listing 31: Codice per ottenere i risultati il grafico

Coefficenti 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 $1, 2, \ldots, 7$ e 9 (come numeri razionali).

28.1 Soluzione

```
function p = newtonCotes(n)
%
%   c = newtonCotes(n);
%   Funzione che calcola i valori dei pesi per la formula di Newton-Cotes di grado
    n.
%
%   Input: n - grado della formula
%   Output: p - vettore dei pesi risultante
%
    if nargin < 1
        error('Errore: parametri in ingresso insufficienti');
    elseif n < 1
        error('Errore:Parametri in ingresso sbagliati');
    end

    k = n+1:-1:1;
    p = zeros(1,n+1);
    for i=0:n
        d = [0:i-1 i+1:n];
        den = prod(i - d);
        alpha = poly(d);
        alpha = [alpha ./ k 0];
        p(i+1) = polyval(alpha,n) / den;
    end
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:

```
gradi = [1, 2, 3, 4, 5, 6, 7, 9];

fprintf('Grado  |  Pesi della quadratura\n');
for i = 1:length(gradi)
    fprintf('--------------------');
end
fprintf('\n');
for i = 1:length(gradi)
    g = gradi(i);
    if g == 8 || g>9
        fprintf('Grado %d non accettabile\n', g);
        continue;
    end

    pesi = newtonCotes(g);
    % Convertiamo i pesi in forma frazionaria
    pesifraz = rats(pesi);
    fprintf(' %d     |  ' , g);
    disp(pesifraz);
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] = composite( fun, a, b, k, n)` che implementi la formula composita 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

```
function [If, err] = composita(fun, a, b, k, n)
% [If, err] = composita(fun, a, b, k, n)
% Implementa la formula composita di Newton-Cotes di grado k su n+1 ascisse
    equidistanti
%
% Input: fun - funzione integranda
%        [a,b] - intervallo di integrazione
%        k - grado della formuala di Newton-Cotes
%        n - numero di suddivisioni dell'intervallo di integrazione (multiplo pari
    di k)
%
% Output: If - approssimazione dell'integrale
%         err - stima dell'errore di quadratura
%
if nargin < 5
    error('Numero di parametri insufficienti');
end
% Controlla che k sia un intero positivo
if ~isnumeric(k) || mod(k,1) ~=0 || k<=0
    error('k deve essere un intero positivo');
end
% Controlla che gli estremi di integrazione siano corretti
if a>b
    error('Gli estremi di integrazione non sono validi');
end
```

```
% Controlla che n non sia un multiplo pari di k
if mod(n,k)~=0
error("Errore: il valore k deve essere un multiplo pari di n")
end
% Controlla che n sia un multiplo pari
if mod(n,2)==0
    mu = 2;
else
    mu = 1;
end
% Calcola i pesi della formula di Newton-Cotes di grado k
pesi = newtonCotes(k);
% Calcola l'approssimazione dell'integrale
If = 0;
I1 = 0;
h = (b - a) / n;
h1 = (b - a) / (n/2);
for i = 0:(n-1)
    x = linspace(a + i*h, a + (i+1)*h, k+1);
    y = feval(fun, x);
    If = If + h/k*sum(y.*pesi);
    if i<n/2
        x1 = linspace(a + i*h1, a + (i+1)*h1, k+1);
        y1 = feval(fun, x1);
        I1 = I1 + h1/k*sum(y1.*pesi);
    end

end
% Stima l'errore
err = abs((If-I1)/(2^(k+mu)-1));
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 calcoalre l'approssimazione dell'integrale

```
f = @(x) exp(3 * x);
F = @(x) exp(3 * x) / 3;
a = 0;
b = 1;
IntEsatto = F(b) - F(a);  % Valore esatto dell'integrale
format long
k = [1, 2, 3, 6];
n = 12;

If = zeros(size(k));
```

```
err = zeros(size(k));
% Calcolo l'integrale usando il metodo di Newton-Cotes composito per ciascun k
for i = 1:length(k)
    [If(i), err(i)] = composita(f, a, b, k(i), n);
end

disp('Valore esatto:');
disp(IntEsatto);
err1 = (IntEsatto - If);

T = table(k', If', err', err1', ...
          'VariableNames', {'k', 'Integr_Approx', 'Errore_Stimato', '
             Errore_Effettivo'});
disp(T);
```

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

I risultati vengono quindi:

```
>> TesterEs30
Valore esatto:
   6.361845641062557
```

k	Integr_Approx	Errore_Stimato	Errore_Effettivo
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