Elaborato di Calcolo Numerico 2023-2024

Autore: Jonathan Colombo Matricola: 7011579 Email: jonathan.colombo@edu.unifi.it
Autore: Matteo Pascuzzo Matricola: 7072913 Email: matteo.pascuzzo@edu.unifi.it

00/06/2024

Esercizio 1

Dimostrare che:

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

Soluzione Per verificare l'equazione basta scrivere gli sviluppi di Taylor centrati in x delle funzioni presenti:

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

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

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

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

Riscrivo il numeratore sostituendo le funzioni con gli sviluppi appena ottenuti:

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

Proseguo lo sviluppo del numeratore:

$$
\begin{aligned}
&25f(x) - 48f(x) - 48h'f(x) - \frac{48h^2f''(x)}{2} - \frac{48h^3f'''(x)}{6} - \frac{48h^4f''''(x)}{24} + O(h^5) \\
&+ 36f(x) - 72hf'(x) + 72h^2f''(x) - 48h^3f'''(x) + 24h^4f''''(x) + O(h^5) \\
&- 16f(x) + 48hf'(x) - 72h^2f''(x) + 72h^3f'''(x) - 54h^4f''''(x) + O(h^5) \\
&+ 3f(x) - 12hf'(x) + 24h^2f''(x) - 32h^3f'''(x) + 32h^4f''''(x) + O(h^5) \\
&= 12hf'(x) + O(h^5)
\end{aligned}
$$

Quindi l'equazione iniziale diventa:

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

come volevamo dimostrare.

Esercizio 2

La funzione:

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

ha un asintoto in x = 4/3, in cui tende a -∞. Graficarla in Matlab, utilizzando x = linspace(1, 5/3, 100001) (in modo che il floating di 4/3 sia contenuto in x) e vedere dove si ottiene il minimo. Commentare i risultati ottenuti.

Soluzione

Di seguito è riportato il codice Matlab e il grafico ottenuto dall'esecuzione del programma:

```
clc, clearvars, close all
x = linspace(1,5/3, 100001); %genero centomilauno valori da 1 a 5/3
y = 1 + x.^2 + (log(abs(3*(1 - x) + 1))/80);
plot(x,y); %grafica i punti (x,y)
grid on;
xlabel('ascissa x');
ylabel('ordinata f(x)');
title('Grafico della funzione f(x)');

[min_value, min_index] = min(y); %restituisce il minimo valore e l'indice
min_x = x(min_index);
disp(['Il minimo della funzione si verifica in x = ', num2str(min_x), '
    con valore f(x) = ', num2str(min_value)]);

% Calcolo dei limiti della funzione mentre x si avvicina a 4/3
x_right = 4/3 + 0.001;  % x si avvicina a 4/3 da destra
x_left = 4/3 - 0.001;   % x si avvicina a 4/3 da sinistra

lim_right = 1 + x_right^2 + log(abs(3*(1 - x_right) + 1))/80;
lim_left = 1 + x_left^2 + log(abs(3*(1 - x_left) + 1))/80;

```

```
disp(['Limite della funzione mentre x si avvicina a 4/3 da destra: ',
    num2str(lim_right)]);
disp(['Limite della funzione mentre x si avvicina a 4/3 da sinistra: ',
    num2str(lim_left)]);
```

Codice Matlab esercizio 2

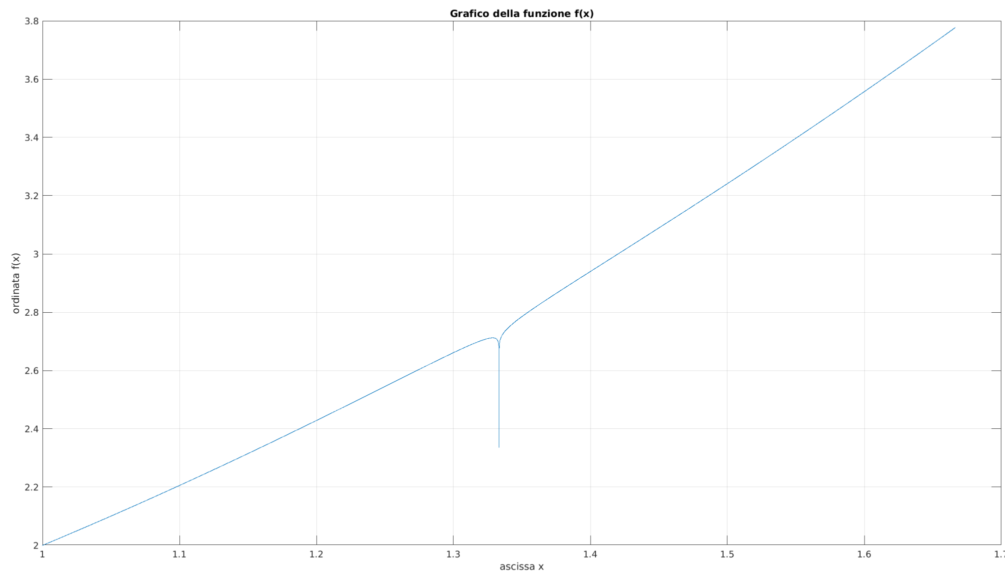


Grafico della funzione

Minimo della Funzione: Il minimo della funzione si verifica in x=1 con valore f(x)=2. Questo è coerente con la definizione della funzione, poiché per x=1, f(x)=2. Quindi, il minimo locale e assoluto si trova proprio in x=1. Anche se teoricamente ci si aspetterebbe un comportamento asintotico in x=4/3, i risultati numerici dei limiti mostrano valori finiti che si stabilizzano attorno a 2.7.

Esercizio 3

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

Soluzione

La cancellazione numerica consiste nella perdita di cifre significative nel risultato derivante dalla somma di addendi quasi opposti. Questo rispecchia il malcondizionamento di questa operazione. Infatti, se x e y sono i due numeri da sommare, il numero di condizionamento è dato da:

$$k = \frac{|x1| + |x2|}{|x1 + x2|}$$

Esempio: Abbiamo i seguenti due numeri reali p1 e p2: p1 = 0.12345789, p2 = 0.12345666 la cui differenza d vale:

$$d = p1 - p2 = 0.00000123 = 1.23 \cdot 10^{-6}$$

Supponiamo che l'architettura del calcolatore ci permetta di memorizzare solo le prime 6 cifre dopo la virgola, quindi i due numeri per essere rappresentati all'interno del calcolatore, verrebbero troncati appena dopo la sesta cifra: t1 = 0.123457 e t2 = 0.123456 Effettuando la sottrazione e riscrivendo il risultato nella notazione floating point si ha:

$$dt = t1 - t2 = 0.000001 = 1 \cdot 10^{-6}$$

che è un risultato diverso rispetto a d.

Esercizio 4

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

Soluzione: di seguito è riportato il codice Matlab.

```matlab
function [x, it, count] = bisezione( a, b, f, tolx )
%
% [x, it, count] = bisezione( a, b, f, tolx )
% Metodo di bisezione per calcolare una radice di f(x), interna ad [a,b], con tolleranza tolx.
%
if a >= b
    error('Estremi intervallo errati');
end
if tolx<=0
    error('Tolleranza non appropriata');
end
count = 0;
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+2;
    if abs(fx)<=tolx*f1x
        break
    elseif fa*fx<0
        b = x; fb = fx;
    else
        a = x; fa = fx;
    end
end
```

```
it =  i;
return
```

Codice Matlab metodo di bisezione

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).

Soluzione:

Di seguito è riportato il codice Matlab del metodo di Newton.

```
function [x, it, count] = newton( f, f1, x0, tolx, maxit )
%
% [x, it, count] = newton( f, f1, x0, tolx, maxit)
% Metodo di Newton per determinare una approssimazione
% della radice di f(x)=0 con tolleranza (mista) tolx, a
% partire da x0, entro maxit iterationi (default = 100).
% f1 implementa f%(x) mentre in uscita flag vale -1 se
% tolleranza non soddisfatta entro maxit iterate o
% la derivata si annulla, altrimenti ritorna il numero
% di iterazioni richieste.

if maxit <0
    maxit=100;
end
if tolx<0
    error('Tolleranza non idonea');
end
it = 0;
count=0;
x = x0 ;
for i =1: maxit
    x0 = x ;
    fx = feval (f , x0 );
    f1x = feval ( f1 , x0 );
    count=count+2;
    if f1x==0
        break
    end
    x = x0 - (fx / f1x) ;
    %x = x0 - m *(fx / f1x) ; riga da scommentare per il metodo di newton modificato,
                            % dell'esercizio 7 con m = 5
    if abs (x - x0 ) <= tolx *(1+ abs ( x ))
        break
    end
    it = it + 1;
end
if ( abs (x - x0 ) > tolx *(1+ abs ( x )))
```

```
    disp (' Il metodo non converge ')
end
end
```

Codice Matlab metodo di Newton

Di seguito è riportato il codice Matlab del metodo delle secanti.

```
function [x,it,count] = secanti(f,x0,x1,maxIt,tolx)
% [x, it, count] = secanti(f,x0,x1,maxIt,tolx)
% Calcola uno zero della funzione f usando il metodo delle secanti.
% Input:
%   f - funzione di cui voglio determinare la radice,
%   x0 - prima approssimazione iniziale della radice x1
%   x1 - seconda approssimazione iniziale della radice,
%   maxIt - numero massimo d'iterazioni [DEFAULT 100],
%   tolx - tolleranza [DEFAULT 10^-3]
%
% Output:
%   x - approssimazione della radice,
%   it - numero di iterazioni eseguite,
%count - numero di valutazioni funzionali eseguite

if maxIt <0
    maxIt=100; end
if tolx<0
    tolx=1e-3; end
count=0;
f0=feval(f,x0);
f1=feval(f,x1);
count=count+2;
if f1==0
    x=x1; return; end
x=(x0*f1-x1*f0)/(f1-f0);
for i=1:maxIt
    if abs(x-x1)<= tolx*(1+abs(x1))
        break
    end
    x0=x1;
    f0=f1;
    x1=x;
    f1= feval(f,x);
    count=count+1;
    if f1==0
        break
    end
    x=(x0*f1-x1*f0)/(f1-f0);
end
it=i;
if abs(x-x1)>tolx*(1+abs(x1))
    disp('Il metodo non converge');
end
```

```
end
```

Codice Matlab metodo delle secanti

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 x0 = 1 (e x1 = 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.

Soluzione:

Table 1: Risultati dei metodi di Secanti, Bisezione e Newton

Metodo	Tolleranza	Radice	Iterazioni	Numero di Valutazioni Funzionali
Secanti	10^{-3}	1.1522e-06	6	7
Secanti	10^{-6}	2.0949e-16	8	9
Secanti	10^{-9}	2.0949e-16	8	9
Secanti	10^{-12}	-1.2557e-17	9	10
Bisezione	10^{-3}	0.00097656	9	38
Bisezione	10^{-6}	9.5367e-07	19	38
Bisezione	10^{-9}	9.3132e-10	29	58
Bisezione	10^{-12}	9.0949e-13	39	78
Newton	10^{-3}	2.8423e-09	5	10
Newton	10^{-6}	3.5748e-17	6	12
Newton	10^{-9}	3.5748e-17	6	12
Newton	10^{-12}	3.5748e-17	6	12

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.

Soluzione:

La molteplicità per il metodo di Newton è stata calcolata osservando in quale ordine di derivazione la funzione non si annulla per x = 0. Calcolo delle derivate:

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

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

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

$$f'''(x) = e^x - sin(x) - cos(x)$$

$$f^{(4)}(x) = e^x - cos(x) + sin(x)$$

$$f^{(5)}(x) = e^x + sin(x) + cos(x)$$

Table 2: Risultati dei metodi di Secanti, Bisezione, Newton e Newton modificato

Metodo	Tolleranza	Radice	Iterazioni	Numero di Valutazioni Funzionali
Secanti	10^{-3}	0.005576	33	34
Secanti	10^{-6}	-0.0010403	61	62
Secanti	10^{-9}	-0.001075	89	90
Secanti	10^{-12}	-0.0010751	100	101
Bisezione	10^{-3}	0.0375	3	6
Bisezione	10^{-6}	0.003125	5	10
Bisezione	10^{-9}	0.0011163	31	62
Bisezione	10^{-12}	0.0011163	32	64
Newton	10^{-3}	0.0039218	25	50
Newton	10^{-6}	-8.0477e-05	42	84
Newton	10^{-9}	-8.0477e-05	42	84
Newton	10^{-12}	-8.0477e-05	42	84
Newton modificato	10^{-3}	2.6016e-05	2	4
Newton modificato	10^{-6}	2.6016e-05	2	4
Newton modificato	10^{-9}	2.6016e-05	2	4
Newton modificato	10^{-12}	2.6016e-05	2	4

Dalla tabella possiamo effettuare diverse considerazioni:

- non tutti i metodi convergono alla soluzione per ogni livello di tolleranza;
- il numero di iterazioni non si mantiene stabile ma aumenta al variare della tolleranza;
- confrontando i risultati dei vari metodi con quelli ottenuti utilizzando il metodo di Newton modificato, si osserva una drastica diminuzione del numero di iterazioni e del numero di valutazioni funzionali. Inoltre, il metodo di Newton modificato fornisce il risultato più preciso indipendentemente dal grado di tolleranza impostato.

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 validarla su un congruo numero di esempi significativi, che evidenzino tutti i suoi possibili output.

Soluzione: di seguito è riportato il codice Matlab.

```
function x = mialu(A,b)
% x = mialu(A,b)
% Data in ingresso una matrice A ed un vettore b, calcola la soluzione del sistema lineare Ax=b con il metodo di fattorizzazione
% LU con pivoting parziale
% Input: A = matrice dei coefficienti, b = vettore dei termini noti
% Output: x = soluzione del sistema lineare
[m,n] = size(A);
if m ~= n
    error('Errore: matrice in input non quadrata');
end
if n~=length(b)
    error(' Errore: la dimensione del vettore b non coincide con la dimensione della matrice A ');
end
if size(b,2)>1
    error(' Errore: il vettore b non ha la struttura di un vettore colonna ');
end
p = (1:n).';
for i=1:n
    [mi,ki]=max(abs(A(i:n,i)));
    if mi==0
        error(' Errore: matrice singolare! ');
    end
    ki = ki+i-1;
    if ki>i
        A([i,ki],:) = A([ki,i],:);
        p([i,ki]) = p([ki,i]);
    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(p);
for i=1:n
    x(i+1:n) = x(i+1:n)-A(i+1:n,i)*x(i);
end
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
```

Codice Matlab metodo mialu

Primo esempio: Il codice restituisce correttamente un messaggio di errore nel caso in cui venga data in input una matrice singolare A. Data una matrice dei coefficienti singolare

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \\ 3 & 6 & 9 \end{bmatrix} \in \mathbb{R}^{3\times 3}$$

lo script restituisce: "Errore: matrice singolare!".

Secondo esempio: Il codice restituisce correttamente un messaggio di errore nel caso in cui venga data in input una matrice non quadrata. Data una matrice dei coefficienti

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \in \mathbb{R}^{2\times 3}$$

lo script restituisce: "Errore: matrice in input non quadrata".

Terzo esempio: Il codice restituisce correttamente la soluzione del sistema lineare. Data una matrice dei coefficienti

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 5 & 4 \\ 6 & 7 & 8 \end{bmatrix} \in \mathbb{R}^{3\times 3}$$

e un vettore

$$\mathbf{b} = \begin{pmatrix} 10 \\ 20 \\ 30 \end{pmatrix} \in \mathbb{R}^{3}$$

lo script restituisce le soluzioni: -0.5000, 3.0000 e 1.5000 .

Quarto esempio: Il codice restituisce correttamente un messaggio di errore nel caso in cui la lunghezza di b non sia compatibile con il numero delle righe/colonne della matrice A. Data una matrice

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 5 & 4 \\ 6 & 7 & 8 \end{bmatrix} \in \mathbb{R}^{3\times 3}$$

e un vettore

$$\mathbf{b} = \begin{pmatrix} 10 \\ 20 \\ 30 \\ 40 \end{pmatrix} \in \mathbb{R}^{4}$$

lo script restituisce: "Errore: la dimensione del vettore b non coincide con la dimensione della matrice A".

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.

Soluzione: di seguito è riportato il codice Matlab

```
function x = mialdl(A,b)
% x = mialdl(A,b)
% Calcola la soluzione del sistema lineare Ax = b con il metodo di
    fattorizzazione LDLt
% Input:
% A = matrice simmetrica definita positiva da fattorizzare
% b = vettore dei termini noti
% Output:
```

```matlab
% x = soluzione del sistema

[m,n] = size(A);
if m~=n
    error("Errore: La matrice deve essere quadrata");
end
if n~=length(b)
    error(' Errore: la dimensione del vettore b non coincide con la dimensione della matrice A ');
end
% Verifica della simmetria
if ~isequal(A, A')
    error('Matrice A non simmetrica.');
end
if A(1,1)<=0
    error('Errore: La matrice deve essere sdp');
end
A(2:n,1)=A(2:n,1)/A(1,1); %fattorizzazione LDLT
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("Errore: La matrice deve essere sdp");
    end
    A(i+1:n,i) = (A(i+1:n,i)-A(i+1:n,1:i-1)*v)/A(i,i);
end
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
```

Codice Matlab metodo mialdl

Primo esempio: Il codice restituisce correttamente un messaggio di errore nel caso in cui venga data in input una matrice A non quadrata. Data una matrice dei coefficienti non quadrata

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \in \mathbb{R}^{2\times 3}$$

lo script restituisce: "Errore: La matrice deve essere quadrata".

Secondo esempio: Il codice restituisce correttamente un messaggio di errore nel caso in cui la lunghezza di b non sia compatibile con il numero delle righe/colonne della matrice A. Data una matrice

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 5 & 4 \\ 6 & 7 & 8 \end{bmatrix} \in \mathbb{R}^{3\times 3}$$

e un vettore

$$\mathbf{b} = \begin{pmatrix} 10 \\ 20 \\ 30 \\ 40 \end{pmatrix} \in \mathbb{R}^4$$

lo script restituisce: "Errore: la dimensione del vettore b non coincide con la dimensione della matrice A".

Terzo esempio: Il codice restituisce correttamente un messaggio di errore nel caso in cui la matrice A sia simmetrica ma non definita positiva. Per questo esempio si può utilizzare uno script per osservare il comportamento corretto della funzione mialdl.

```
% Dimensioni della matrice A
n = 3;

% Generazione di una matrice simmetrica ma non definita positiva
A = randn(n,n);
A = 0.5 * (A + A'); % Garantisce la simmetria

disp('Matrice A:');
disp(A);

% Lunghezza desiderata del vettore dei termini noti b
lunghezza_b = 3;

% Generazione del vettore dei termini noti b
b = rand(lunghezza_b, 1);

disp('vettore dei termini noti');
disp(b);

% Chiamata alla funzione mialdl per calcolare la soluzione del sistema lineare Ax = b
x = mialdl(A, b);
```

Codice Matlab terzo esempio

Quarto esempio: Il codice restituisce correttamente un messaggio di errore nel caso in cui la matrice A definita positiva ma non simmetrica. Data una matrice

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \in \mathbb{R}^{3\times 3}$$

e un vettore

$$\mathbf{b} = \begin{pmatrix} 10 \\ 20 \\ 30 \end{pmatrix} \in \mathbb{R}^3$$

lo script restituisce: "Errore: matrice A non simmetrica".

Quinto esempio: Il codice restituisce correttamente le soluzioni del sistema lineare. Data una matrice dei coefficienti simmetrica definita positiva

$$A = \begin{bmatrix} 6 & 2 & 1 \\ 2 & 5 & 2 \\ 1 & 2 & 4 \end{bmatrix} \in \mathbb{R}^{3\times 3}$$

e un vettore dei termini noti

$$\mathbf{b} = \begin{pmatrix} 0.9294 \\ 0.7757 \\ 0.4868 \end{pmatrix} \in \mathbb{R}^3$$

lo script restituisce le soluzioni: 0.1172, 0.0891 e 0.0478.

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.

Soluzione: di seguito è riportato il codice Matlab

```matlab
function [x,nr] = miaqr(A,b)
% [x,nr] = miaqr(A,b)
% La funzione miaqr(A,b) calcola la fattorizzazione QR del sistema lineare
% Ax = b sovradimensionato restituendo,oltre alla fattorizzazione,la norma
% euclidea del vettore residuo.
% Input:
% A = matrice da fattorizzare
% b = vettore dei termini noti
% Output:
% x = soluzione del sistema
% nr = norma euclidea del vettore residuo

[m,n] = size(A);
if(n>m)
    error('Errore: sistema in input non sovradeterminato');
end
k = length(b);
if k~=m
    error('Errore: La dimensione della matrice e del vettore non coincidono'); end
for i = 1:n
    a = norm(A(i:m,i),2);
    if a==0
        error('Errore: La matrice non ha rango massimo');
    end
    if A(i,i)>=0
        a = -a;
    end
    v1 = A(i,i)-a;
    A(i,i) = a;
    A(i+1:m,i) = A(i+1:m,i)/v1;
    beta = -v1/a ;
    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));
end
for i=1:n
```

```
    v = [1;A(i+1:m,i)];
    beta = 2/(v'*v);
    b(i:k) = b(i:k)-(beta*(v'*b(i:k)))*v;
end
for j=n:-1:1
    b(j) = b(j)/A(j,j);
    b(1:j-1) = b(1:j-1)-A(1:j-1,j)*b(j);
end
x = b(1:n);
nr = norm(b(n+1:m));
end
```

Codice Matlab metodo miaqr

Primo esempio: Il codice restituisce correttamente un messaggio di errore nel caso in cui la matrice dei coefficienti A abbia un numero di colonne maggiore di quello delle righe. Data una matrice dei coefficienti A

$$A = \begin{bmatrix} 6 & 2 & 1 \\ 2 & 5 & 2 \end{bmatrix} \in \mathbb{R}^{2\times 3}$$

e un vettore

$$\mathbf{b} = \begin{pmatrix} 4 \\ 1 \end{pmatrix} \in \mathbb{R}^2$$

lo script restituisce: "Errore: sistema in input non sovradeterminato".

Secondo esempio: Il codice restituisce correttamente un messaggio di errore nel caso in cui il vettore dei termini b abbia un numero diverso di righe rispetto ad A. Data una matrice dei coefficienti

$$A = \begin{bmatrix} 6 & 2 \\ 1 & 1 \\ 5 & 4 \end{bmatrix} \in \mathbb{R}^{3\times 2}$$

e un vettore

$$\mathbf{b} = \begin{pmatrix} 1 \\ 4 \\ 5 \\ 8 \end{pmatrix} \in \mathbb{R}^2$$

lo script restituisce: "Errore: La dimensione della matrice e del vettore non coincidono".

Terzo esempio: Il codice restituisce correttamente un messaggio di errore nel caso in cui la matrice A non abbia rango massimo. Data una matrice dei coefficienti

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 4 & 6 & 8 \\ 3 & 6 & 9 & 12 \\ 3 & 6 & 9 & 12 \end{bmatrix} \in \mathbb{R}^{4\times 4}$$

e un vettore

$$\mathbf{b} = \begin{pmatrix} 1 \\ 4 \\ 5 \\ 8 \end{pmatrix} \in \mathbb{R}^4$$

lo script restituisce: "Errore: La matrice non ha rango massimo".

Quarto esempio: Il codice restituisce le soluzioni corrette e norma del vettore residuo uguale a 0 data una matrice una matrice quadrata e un vettore dei termini noti in input. Data una matrice dei coefficienti

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \in \mathbb{R}^{3\times 3}$$

e un vettore

$$\mathbf{b} = \begin{pmatrix} 10 \\ 11 \\ 12 \end{pmatrix} \in \mathbb{R}^{3}$$

lo script restituisce correttamente le soluzioni: -3.3333. -2.3333 e 6.0000. Norma euclidea del vettore residuo: 0.

Esercizio 11

Risolvere i sistemi lineari, di dimensione n, $A_n x_n = b_n, \;\; n = 1, \ldots, 15$ di cui

$$\mathbf{A}_n = \begin{bmatrix} 1 & 1 & \ldots & \ldots & \ldots & 1 \\ 10 & \ddots & \ddots & \ldots & \ldots & 1 \\ 10^2 & \ddots & \ddots & \ddots & & \vdots \\ \vdots & \vdots & \ldots & \ldots & \ldots & 1 \\ 10^{n-1} & 1 & \ldots & \ldots & 10 & 1 \end{bmatrix} \in \mathbb{R}^{n\times n}$$

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

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

Soluzione: di seguito è riportato il codice Matlab

```
% Generazione del sistema lineare An
n = 15; % dimensione della matrice
A = ones(n); % inizializza una matrice di uno

% Genera la matrice desiderata
for i = 1:n
    for j = 1:i
        A(i, j) = 10^(i-j); % assegna il valore 10^(i-j) alla posizione (i
            , j)
    end
end

```

```
disp(A); % visualizza la matrice generata

%Generazione del vettore dei termini noti
b = zeros(1, n);
for i = 1:n
    b(i) = n - i + ((10^(i) - 1 )/9);
end

disp(b); % visualizza il vettore generato
b = b.'; % vettore b trasposto
% Richiamo della funzione mialu
x = mialu(A,b);

% Stampa del risultato
disp(x);

condizionamento_2 = cond(A);  % Calcolo del numero di condizionamento usando la norma 2
disp(['Numero di condizionamento (norma 2): ', num2str(condizionamento_2)]);
```

Codice Matlab esercizio 11

Di seguito è riportata la tabella contenente i risultati del sistema lineare.

Risultati del sistema lineare
1.00000000000000
1.00000000000000
1.00000000000000
0.999999999999895
0.999999999999708
1.00000000000027
1.00000000020709
0.999999998551276
1.00000000622338
0.999999819370156
1.00000059950348
0.999996185302734
0.999949137369792
1.00179036458333
0.998263888888889

Risultati esercizio 11

I risultati ottenuti tramite la funzione `mialu` sembrano essere accurati nonostante il numero di condizionamento della matrice sia notevolmente elevato (Numero di condizionamento: 2.45×10^{14}). Un numero di condizionamento elevato indica il sistema è malcondizionato, il che potrebbe portare a una variazione nei risultati calcolati. Tuttavia, analizzando i valori del vettore errore, possiamo notare che essi sono estremamente piccoli. Questo indica che la soluzione approssimata è in realtà molto vicina alla soluzione esatta. I valori del vettore errore sono i seguenti: 3.55×10^{-15}, 3.55×10^{-15}, 2.84×10^{-14}, 4.55×10^{-13}, 1.82×10^{-12}, 0, 4.66×10^{-10}, 0, 0, 2.38×10^{-7}, -1.91×10^{-6}, -1.53×10^{-5}, 0, -0.002, 0.

Esercizio 12

Fattorizzare, utilizzando la function mialdlt, le matrici sdp

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

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

Soluzione: Di seguito è riportato il grafico della esecuzione.

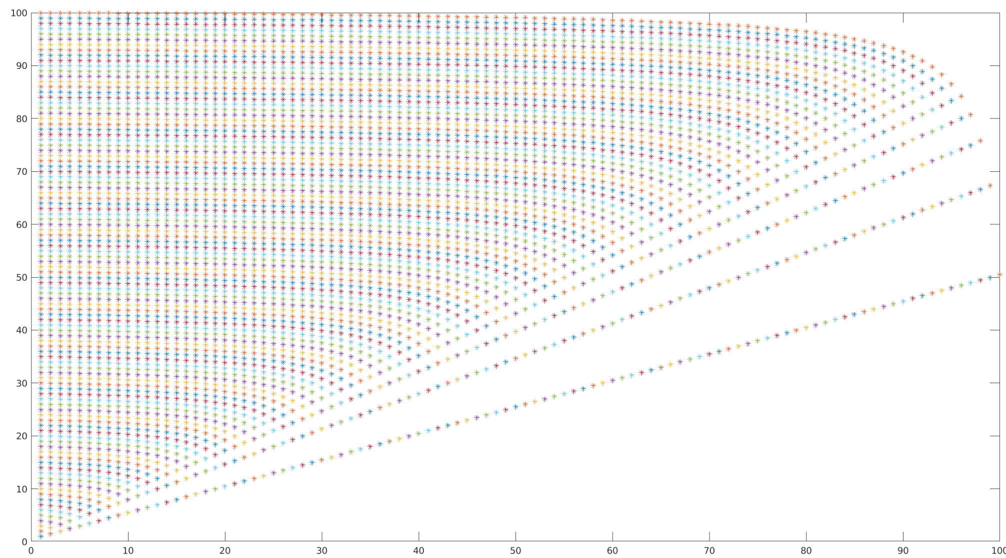


Grafico della funzione

Di seguito è riportato il codice Matlab per la function mialdlt

```
function A = mialdlt(A)

% mialdl(A,b) effettua la fattorizzazione della matrice A con il metodo
% di fattorizzazione LDLt
% Input:
% A = matrice simmetrica definita positiva da fattorizzare
% Output:
% A = matrice fattorizata

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

```
end
A(2:n,1)=A(2:n,1)/A(1,1); %fattorizzazione LDLT
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('Errore: La matrice deve essere sdp');
    end
    A(i+1:n,i) = (A(i+1:n,i)-A(i+1:n,1:i-1)*v)/A(i,i);
end
return
```

Codice Matlab mialdlt

Di seguito è riportato il codice Matlab dell'esercizio 12.

```
% Generazione del sistema lineare An
n = 100; % dimensione della matrice
listA = cell(n,1);% genero una lista di 100 elementi eterogenei

for i = 1:n
    A = -1 * ones(i);
    for j= 1:i
        A(j,j) = i; %faccio l'assegnazione del valore n-esimo sull' elemento diagonale
    end
listA{i} = A;
end

listResults = cell(n,1);
listDiag = cell(n,1);

for i = 1:n
    listResults{i} = mialdlt(listA{i});
    listDiag{i} = diag(listResults{i});
end

x = 1;
y = listDiag{1};
plot(x,y, "*");
hold on; %crea una sola finestra graficando tutte le curve
for i=2:n
    x = 1:i;
    y = listDiag{i};
    plot(x,y, "*");
end

hold off;% fa si che il grafico sia completato in un unica finestra
```

Codice Matlab esercizio 12

Esercizio 13

Utilizzare la function miaqr per risolvere, nel senso dei minimi quadrati, il sistema lineare sovradeterminato $Ax = b$, in cui

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

dove viene minimizzata la seguente norma pesata del residuo $r = (r1, \ldots, r5)^T$:

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

con w1 = w2 = 0.5, w3 = 0.75, w4 = w5 = 0.25. Dettagliare l'intero procedimento, calcolando, in uscita, anche p_w.

Soluzione: di seguito è dettagliato l'intero procedimento della risoluzione dell'esercizio 13.

Calcolo dei pesi

La matrice dei pesi W è una matrice diagonale con le radici quadrate dei pesi sui termini diagonali:

$$W = \begin{pmatrix} \sqrt{0.5} & 0 & 0 & 0 & 0 \\ 0 & \sqrt{0.5} & 0 & 0 & 0 \\ 0 & 0 & \sqrt{0.75} & 0 & 0 \\ 0 & 0 & 0 & \sqrt{0.25} & 0 \\ 0 & 0 & 0 & 0 & \sqrt{0.25} \end{pmatrix}$$

Modifica del sistema

Moltiplichiamo la matrice A e il vettore b per la matrice W:

$$\tilde{A} = WA, \quad \tilde{b} = Wb$$

Fattorizzazione QR e Risoluzione del sistema

Definiamo il sistema ed eseguiamo la risoluzione utilizzando la funzione `miaqr`. Di seguito è riportato il codice dell'esercizio 13.

```
%Utilizzare la function miaqr per risolvere, nel senso dei minimi quadrati, il sistema
%lineare sovradeterminato
% Definizione della matrice A e del vettore dei termini noti b
A = [7,2,1; 8,7,8; 7,0,7; 4,3,3; 7,0,10];
b = [1,2,3,4,5];
disp('matrice dei coefficienti');
disp(A);
disp('vettore dei termini noti');
disp(b);

```

```matlab
b = [1; 2; 3; 4; 5];

% Definizione dei pesi omega
omega = [0.5; 0.5; 0.75; 0.25; 0.25];
omega = omega ./ 2.25; % normalizzo i pesi in modo che la loro somma sia 1.

B = eye(5) .* sqrt(omega);
A = B * A;
b = B * b;

[x, pw] = miaqr(A, b);

disp('La soluzione x: ');
disp(x);
fprintf('La norma pesata del residuo: %.10f\n', pw);
```

Codice Matlab esercizio 13

Risultati

La soluzione del sistema è:

$$x = \begin{pmatrix} 0.1531 \\ -0.1660 \\ 0.3185 \end{pmatrix}$$

La norma pesata del residuo è: 1.0626524785.

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 non lineari. 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 di ingresso (rispettivamente, la tolleranza per il criterio di arresto, ed il massimo numero di iterazioni). La function fun deve avere sintassi: `[f,jacobian]=fun(x)`, se il sistema da risolvere è $f(x) = 0$.

Soluzione: di seguito è riportato il codice Matlab.

```matlab
function [x, nit] = newton(fun, x0, tol, maxit)
%
%   [x, nit] = newton(fun, x0, tol, maxit);
%
%   Metodo di Newton per la risoluzione di sistemi di equazioni non lineari
%
%   Input:
%       fun - identificatore di una function che restituisce la coppia
%             [f, jacobian] dove f gradiente di una funzione f(x) di
%             cui vogliamo approssimare una radice, mentre jacobian matrice Hessiana di f(x);
```

```
%       x0 - vettore valori iniziali;
%       tol - tolleranza (default = 1e-6);
%       maxit - numero massimo di iterazioni (default = 1000).
%
%   Output:
%       x - soluzione del sistema;
%       nit - numero di iterazioni eseguite.
%

    % Imposta i valori predefiniti per tol e maxit se non specificati
    if nargin < 2
        error('Numero di argomenti in ingresso errato');
    elseif nargin == 2
        tol = 1e-6;
        maxit = 1000;
    elseif nargin == 3
        maxit = 1000;
    elseif maxit <= 0 || tol <= 0
        error('Dati in ingresso errati');
    end

    % Forza x0 a essere un vettore colonna
    x0 = x0(:);

    % Inizializza il numero di iterazioni
    nit = maxit;

    % Loop iterativo del metodo di Newton
    for i = 1:maxit
        % Calcola il gradiente e la matrice Hessiana
        [f, jacobian] = fun(x0);

        % Risolvi il sistema lineare jacobian * delta = -f
        delta = mialum(jacobian, -f);

        % Aggiorna x
        x = x0 + delta;

        % Verifica la condizione di convergenza
        if norm(delta ./ (1 + abs(x0)), inf) <= tol
            nit = i;
            break
        end

        % Aggiorna x0 per la prossima iterazione
        x0 = x;
    end
end

% Funzione ausiliaria per la risoluzione di sistemi lineari (LU senza
   pivoting)
```

```matlab
function x = mialum(A, b)
    % Fattorizzazione LU senza pivoting
    [L, U] = lu_no_pivot(A);

    % Risolvi il sistema Ly = b
    y = forward_substitution(L, b);

    % Risolvi il sistema Ux = y
    x = backward_substitution(U, y);
end

% Fattorizzazione LU senza pivoting
function [L, U] = lu_no_pivot(A)
    [n, ~] = size(A);
    L = eye(n);
    U = A;
    for i = 1:n-1
        if U(i, i) == 0
            error('Matrice singolare');
        end
        for j = i+1:n
            L(j, i) = U(j, i) / U(i, i);
            U(j, i:n) = U(j, i:n) - L(j, i) * U(i, i:n);
        end
    end
end

% Risoluzione del sistema triangolare inferiore Ly = b
function y = forward_substitution(L, b)
    n = length(b);
    y = zeros(n, 1);
    for i = 1:n
        y(i) = (b(i) - L(i, 1:i-1) * y(1:i-1)) / L(i, i);
    end
end

% Risoluzione del sistema triangolare superiore Ux = y
function x = backward_substitution(U, y)
    n = length(y);
    x = zeros(n, 1);
    for i = n:-1:1
        x(i) = (y(i) - U(i, i+1:n) * x(i+1:n)) / U(i, i);
    end
end
```

Codice Matlab esercizio 14

Esercizio 15

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

$$f(x) = \frac{1}{2}x^T Q x + e^T[\cos(\alpha x) + \beta \exp(-x)]$$

$$e = (1, ..., 1)^T \in R^{50},$$

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

utilizzando tolleranze tol = 10^{-3}, 10^{-8}, 10^{-13} (le function cos e exp sono da intendersi in modo vettoriale). Graficare la soluzione e tabulare in modo conveniente i risultati ottenuti.

Soluzione: di seguito è riportato il codice Matlab

```
format long;
n = 50;
x0 = zeros(n, 1);
tol = [1e-3, 1e-8, 1e-13];
colors = {'b', 'r', 'm'};
L = zeros(n, length(tol));

% Ciclo per le diverse tolleranze
for i = 1:length(tol)
    x = newton(@fun, x0, tol(i));
    figure;
    plot(1:n, x, 'Color', colors{i});
    title(['Tolleranza ', num2str(tol(i))]);
    xlabel('Indice radice');
    ylabel('Valore di x');
    L(:,i) = x;
end

disp('Soluzioni:');
disp(L);
```

Codice Matlab esercizio 15

Di seguito sono riportati i grafici con le specifiche tolleranze.

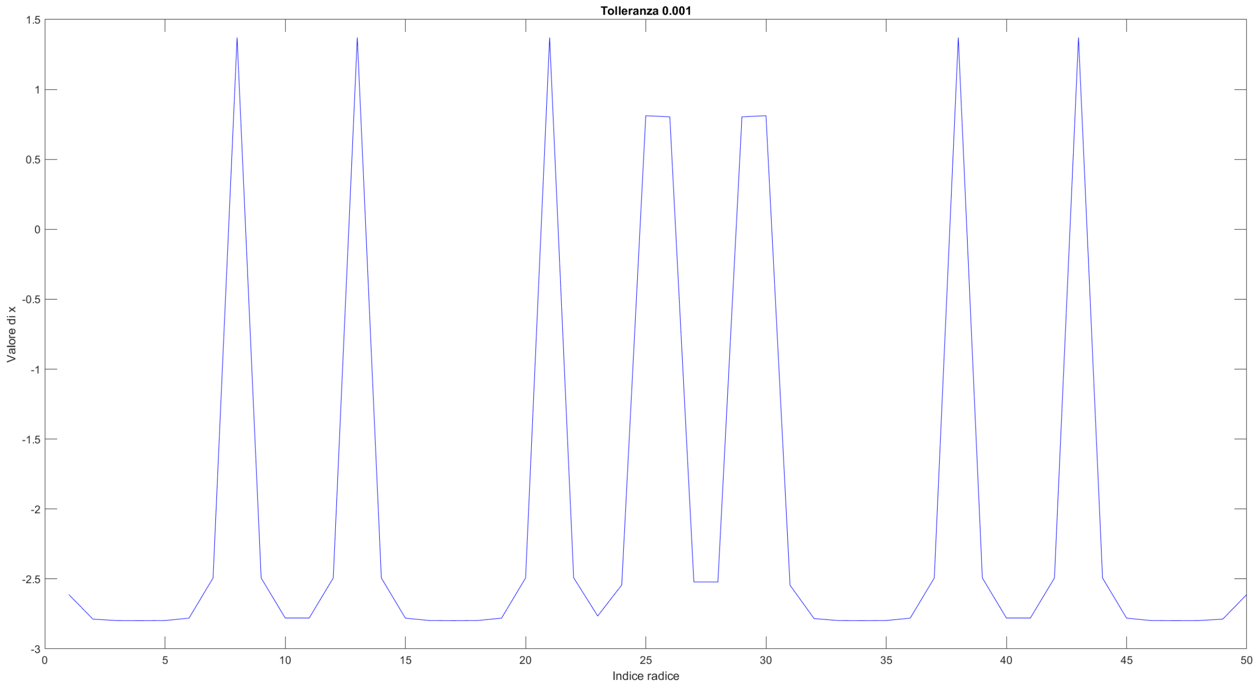


Grafico con tolleranza 10^−3

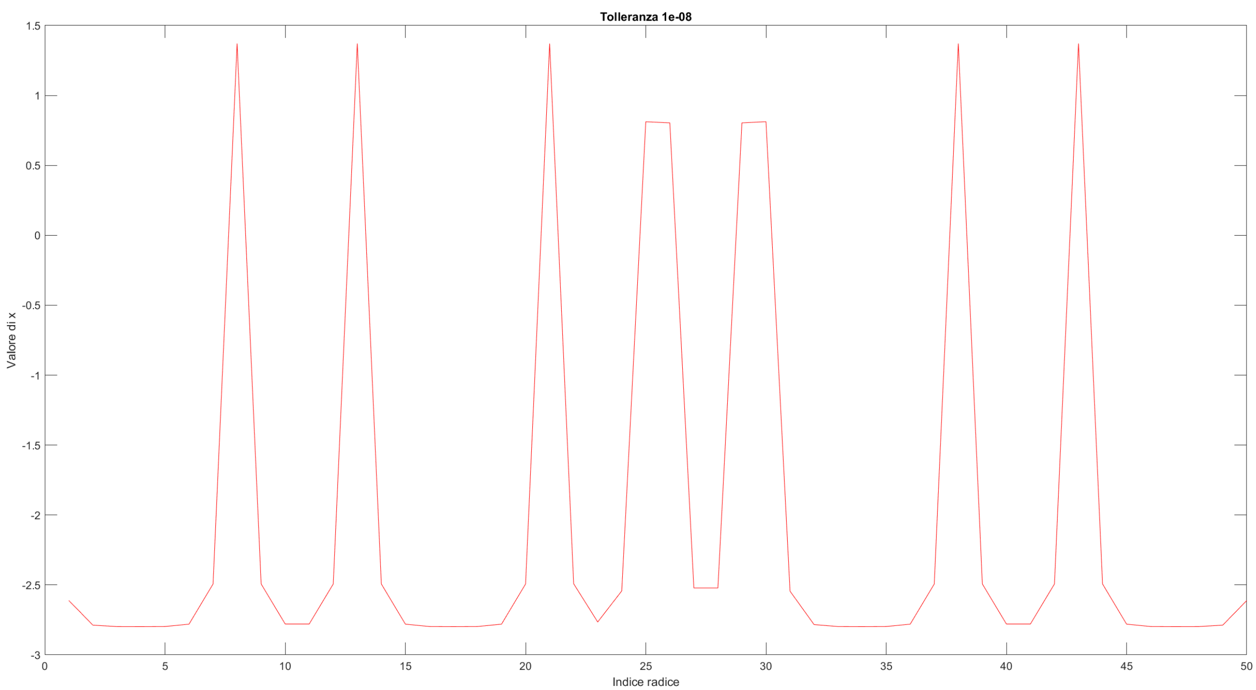


Grafico con tolleranza 10^−8

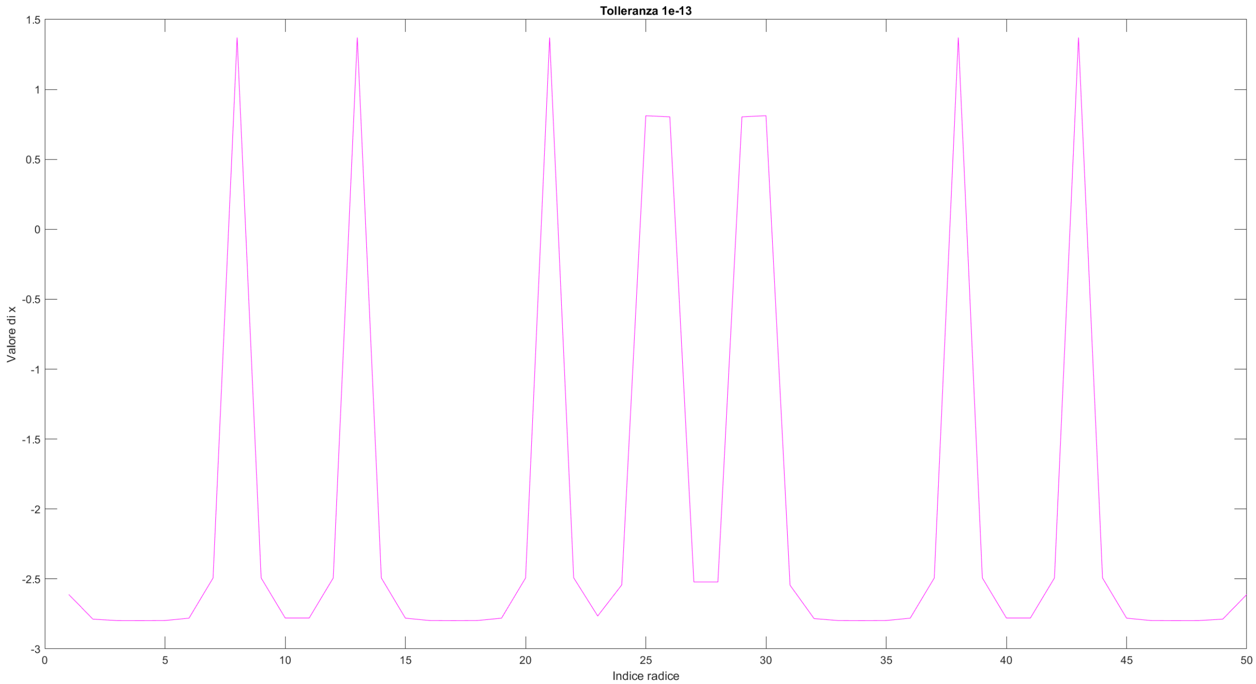


Grafico con tolleranza $10^{-}13$

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.

Soluzione: di seguito è riportato il codice Matlab.

```
function YQ = lagrange(X,Y,XQ)
% Input: X: Vettore colonna contenente le ascisse d'interpolazione che
% devono essere distinte.
% Y: Vettore colonna contenente i valori della funzione nelle ascisse
% d'interpolazione.
% XQ: Vettore colonna contenente le ascisse di cui vogliamo approssimare la funzione
%
% Output:
% YQ: Valori approssimati della funzione con il polinomio interpolante in
% forma di Lagrange.
% Calcola i valori approssimati della funzione(di cui conosciamo i valori Y che assume nelle ascisse X)
% calcolati attraverso il polinomio interpolante in forma di Lagrange nelle ascisse XQ.
if(length(X)~=length(Y)), error('Numero di ascisse di interpolazione e di valori della funzione sono diversi!'),
end
if (length(X) ~= length(unique(X))), error('Ascisse di interpolazione non distinte!'),
end % unique per restituire un vettore con x distinte
```

```
if(isempty(XQ)), error('Il vettore contenente le ascisse in cui interpolare la funzione non contiene nessun elemento!'),
end
if(size(X,2)>1||size(Y,2)>1||size(XQ,2)>1),error('Inserire vettori colonna!'),
end
n=size(X,1);
L=ones(size(XQ,1),n);
for i=1:n
    for j=1:n
        if (i~=j)
            L(:,i)=L(:,i).*((XQ-X(j))/(X(i)-X(j))); %calcolo i polinomi di base di lagrange Lin(x)
        end
    end
end
YQ=zeros(size(XQ));
for i=1:n
    YQ=YQ+Y(i).*L(:,i); %calcolo la sommatoria dei prodotti fi*Lin(x) (con i=0,...,n))
end
end
```

Codice Matlab esercizio 16

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.

Soluzione: Per ricavare questa function è stato necessario in primo luogo ricavarmi il vettore delle differenze divise grazie alla function divdif riportata in fondo che sfrutta la proprietà delle differenze divise:

$$f[x_0, x_1, ..., x_{r-1}, x_r] = \frac{f[x_1, ..., x_{r-1}, x_r] - f[x_0, x_1, ..., x_{r-1}]}{x_r - x_0}$$

Infine è stato sufficiente sfruttare l'algoritmo di Horner per calcolare i valori che il polinomio assume nelle ascisse XQ e salvarli in YQ. Notare, inoltre, come tale function calcoli lo stesso valore della precedente function, in quanto, pur ricavando i valori del polinomio interpolante in forme differenti, il polinomio di grado n interpolante una funzione in un insieme di n+1 ascisse distinte è unico.

Di seguito è riportato il codice Matlab:

```
function YQ = newton(X,Y,XQ)
% function YQ=newton(X,Y,XQ)
% Implementa in modo vettoriale la forma di Newton del polinomio interpolante una funzione.
% Input:
% X: Vettore colonna contenente le ascisse d'interpolazione che
% devono essere distinte.
% Y: Vettore colonna contenente i valori della funzione nelle ascisse d'interpolazione
% XQ: Vettore contenente le ascisse di cui vogliamo approssimare la
```

```
% funzione.
% Output: YQ: Valori approssimati della funzione con il polinomio interpolante in forma di Newton
% Calcola i valori approssimati della funzione(di cui conosciamo i valori Y che assume nelle ascisse X)
% calcolati attraverso il polinomio interpolante in forma di Newton nelle ascisse XQ.

if(length(X)~=length(Y)), error('Numero di ascisse di interpolazione e di valori della funzione non uguale!'),
end
if (length(X) ~= length(unique(X))), error('Ascisse di interpolazione non distinte!'),
end %uso la function unique che restuisce un vettore contenente i valori di x distinte
if(isempty(XQ)), error('Il vettore contenente le ascisse in cui interpolare la funzione non ha elementi!'),
end
if(size(X,2)>1||size(Y,2)>1),error("Inserire vettori colonna!"),
end
df=divdif(X,Y);
n=length(df)-1;
YQ=df(n+1)*ones(size(XQ));
for i=n:-1:1 %algoritmo di horner
    YQ=YQ.*(XQ-X(i))+df(i);
end
return
end
function df=divdif(x,f)
%function per il calcolo delle differenze divise per il polinomio interpolante in forma di newton
n=size(x);
if(n~=length(f)), error('Dati errati!'), end
df=f;
n=n-1;
for i=1:n
    for j=n+1:-1:i+1
        df(j)=(df(j)-df(j-1))/(x(j)-x(j-i));
    end
end
return;
end
```

Codice Matlab esercizio 17

Esercizio 18

Costruire una function, **hermite.m**, avente sintassi **yy = hermite(xi, fi, f1i, xx)** che implementi, in modo vettoriale, il polinomio interpolante di Hermite.

Soluzione:

```
function yy = hermite(xi,fi, f1i, xx)

%La function hermite(xi,fi, f1i, xx) calcola i valori approssimati della
%funzione(di cui conosciamo sia i valori Y che assume nelle ascisse X ed
%i valori Y1 la cui derivata prima assume nelle stesse ascisse) calcolati
%attraverso il polinomio interpolante in forma di Lagrange
%nelle ascisse XQ.
%
% Input:
%xi = Vettore colonna contenente le ascisse d'interpolazione distinte
%fi = Vettore colonna contenente i valori della funzione nelle ascisse
%     d'interpolazione.
%f1i = Vettore colonna contenente i valori che la derivata prima della
%      funzione assume nelle ascisse d'interpolazione.
%xx = Vettore contenente le ascisse in cui vogliamo approssimare la
%     funzione.
%
% Output: yy = Valori approssimati della funzione con il polinomio
%              interpolante di Hermite in forma di Newton.

if(isempty(xx)), error('Il vettore contenente le ascisse in cui interpolare la funzione non ha elementi!'),
end
if(length(xi)~=length(fi)), error('Numero di ascisse di interpolazione e di valori della funzione non uguale!'),
end
if (length(xi) ~= length(unique(xi))), error('Ascisse di interpolazione non distinte!'),
end %uso la function unique che restuisce un vettore contenente i valori senza ripetizioni di X
if(length(f1i)~=length(fi)), error('Lunghezza dei dati in ingresso non compatibile!'),
end
if(size(xi,2)>1||size(fi,2)>1||size(f1i,2)>1),error('Inserire vettori colonna!'),
end
n=(length(xi));
fi(1:2:2*n-1)=fi;
fi(2:2:2*n)=f1i;
df=diffdivher(xi,fi');
n=length(df)-1;
yy=df(n+1)*ones(size(xx));  %algoritmo di Horner per il calcolo dei valori
                            %di un polinomio
for i=n:-1:1
    yy=yy.*(xx-xi(ceil(i/2)))+df(i);
end
return
end

function f=diffdivher(x,f)

```

```
%function per calcolare le differenze divise
%per il polinomio interpolante di hermite

n=(length(f)/2)-1;
for i=2*n+1:-2:3
    f(i)=(f(i)-f(i-2))/(x(ceil(i/2))-x(ceil((i-1)/2)));
end
for j= 2:2*n+1
    for i=(2*n+2):-1:j+1
        f(i)=(f(i)-f(i-1))/(x(ceil(i/2))-x(ceil((i-j)/2)));
    end
end
end
```

Codice Matlab esercizio 18

Questa function è simile a quella dell'esercizio precedente tranne per l'algoritmo che calcola le differenze divise, il quale, in questo caso, include anche le derivate prime della funzione interpolanda. Inoltre, poichè stiamo costruendo un polinomio di grado 2n + 1, dove n + 1 è il numero delle ascisse di interpolazione, anche l'algoritmo di Horner deve essere adattato di conseguenza.

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.

Soluzione: di seguito è riportato il codice Matlab.

```
function p1 = horner_modificato(x, a, xq)
%   p1 = horner_derivata_prima(x, a, xq);
%   Calcola la derivata prima di un polinomio p(x), con coefficienti 'a'
%   rappresentato nella base di Newton.
%   Input:
%       x - vettore delle ascisse per i polinomi di base di Newton;
%       a - coefficienti del polinomio;
%       xq - vettore delle ascisse in cui valutare la derivata prima del
%           polinomio.
%   Output:
%       p1 - derivata prima del polinomio valutata nelle ascisse xq.
if nargin < 3
    error('Numero di parametri insufficienti');
```

```
end
n = length(x);
if n ~= length(a)
    error('Parametri in ingresso errati');
end
p = a(n);
p1 = 0;
for i = n-1:-1:1
    p1 = p + (xq - x(i)) .* p1;
    p = a(i) + (xq - x(i)) .* p;
end
return
```

Codice Matlab esercizio 19

Esercizio 20

Utilizzando le function degli esercizi 18 e 19, calcolare il polinomio interpolante di Hermite la funzione

$$f(x) = exp(x/2 + exp(-x))$$

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.

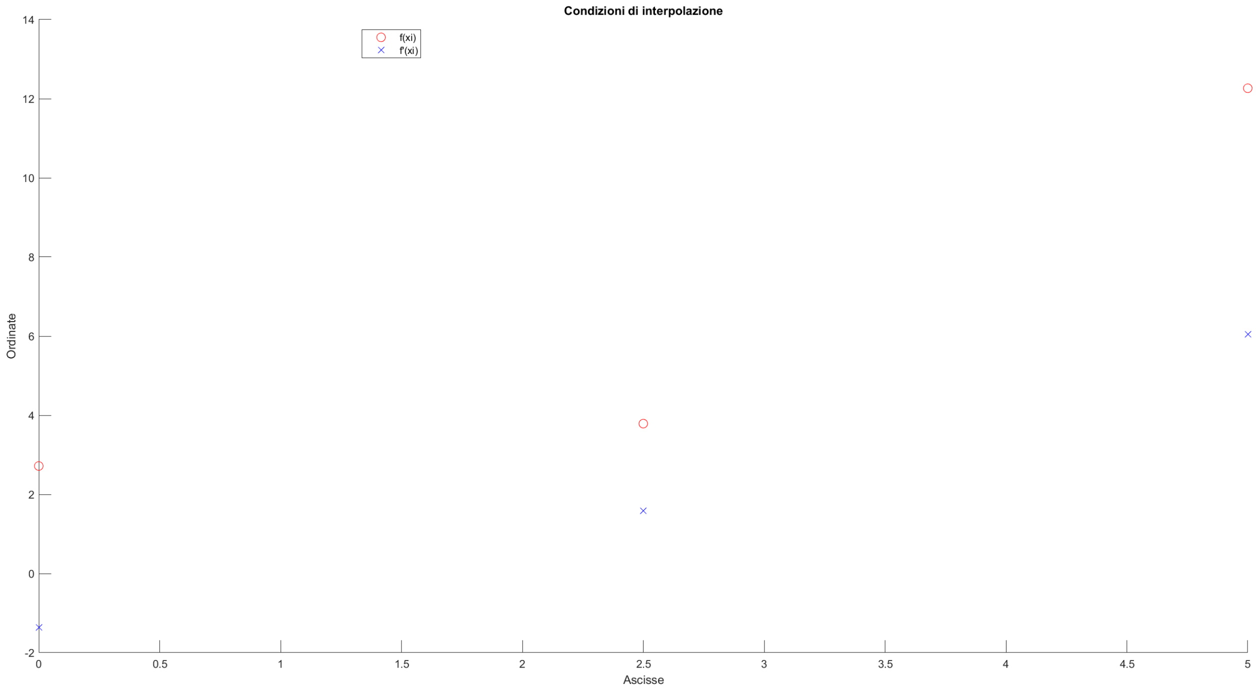
Soluzione: di seguito è riportato il codice Matlab dell'esercizio 21.

```
xi = [0; 2.5; 5]; % Ascisse per l'interpolazione
xx = linspace(0, 5, 100); % 100 punti per il grafico

% Definizione delle funzioni e valutazione di queste
f = @(x) exp(x / 2 + exp(-x)); % Funzione interpolanda
f1 = @(x) (1 / 2 - exp(-x)) .* exp(x / 2 + exp(-x)); % Derivata prima della funzione interpolanda
fi = f(xi);
f1i = f1(xi);

% Calcolo del polinomio interpolante di Hermite
yy = hermite(xi, fi, f1i, xx);

% Preparazione per calcolo della derivata del polinomio interpolante
n = length(xi);
y = zeros(1, 2 * n);
x = zeros(1, 2 * n);
for i = 1:n
    y(2 * i - 1:2 * i) = [fi(i); f1i(i)];
    x(2 * i - 1:2 * i) = [xi(i); xi(i)];
end

dd = diffdivher(x, y); % Differenze divise
dy = horner_modificato(x, dd, xx); % Calcolo della derivata prima del polinomio interpolante
```

```

% Grafico
figure;

subplot(2, 1, 1); % Primo subplot per le funzioni
hold on;
plot(xx, f(xx), 'b', 'LineWidth', 1.2); % Funzione interpolanda in blu
plot(xx, yy, 'g', 'LineWidth', 1.2); % Polinomio interpolante in verde
plot(xi, fi, 'ro'); % Punti di interpolazione
hold off;
xlabel('Ascisse');
ylabel('Ordinate');
title('Funzione interpolanda e polinomio interpolante');
legend('Funzione interpolanda', 'Polinomio interpolante', 'Punti di interpolazione', 'Location', 'best');

subplot(2, 1, 2); % Secondo subplot per le derivate
hold on;
plot(xx, f1(xx), 'c', 'LineWidth', 1.2); % Derivata della funzione interpolanda in ciano
plot(xx, dy, 'm', 'LineWidth', 1.2); % Derivata del polinomio interpolante in magenta
plot(xi, f1i, 'ro'); % Punti di interpolazione delle derivate
hold off;
xlabel('Ascisse');
ylabel('Ordinate');
title('Derivata della funzione interpolanda e del polinomio interpolante');
legend('Derivata funzione interpolanda', 'Derivata polinomio interpolante', 'Punti di interpolazione delle derivate', 'Location', 'best');

% Verifica grafica delle condizioni di interpolazione
figure;
hold on;
plot(xi, fi, 'ro', 'MarkerSize', 8, 'DisplayName', 'f(xi)'); % Condizioni p(xi) = f(xi)
plot(xi, f1i, 'bx', 'MarkerSize', 8, 'DisplayName', 'f''(xi)'); % Condizioni p'(xi) = f'(xi)
hold off;
xlabel('Ascisse');
ylabel('Ordinate');
title('Condizioni di interpolazione');
legend('Location', 'best');
```

Codice Matlab esercizio 21

Di seguito sono riportati i vari grafici.



Condizioni d'interpolazione

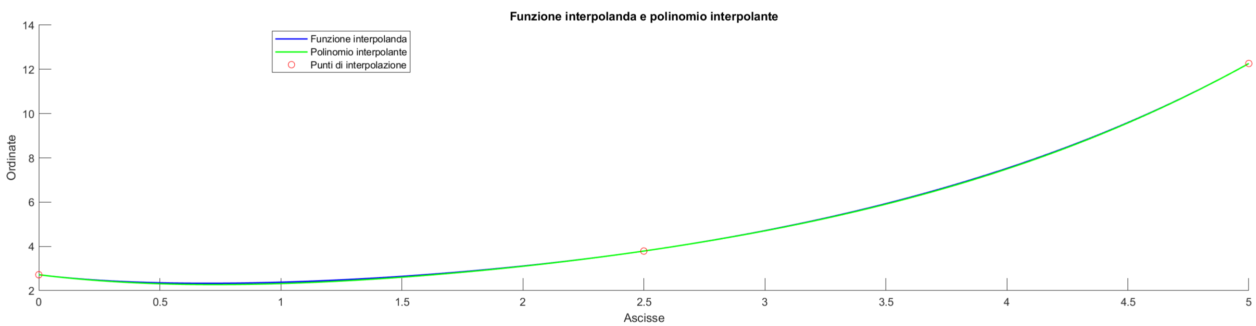


Grafico funzione interpolanda e polinomio interpolante in [0,5]

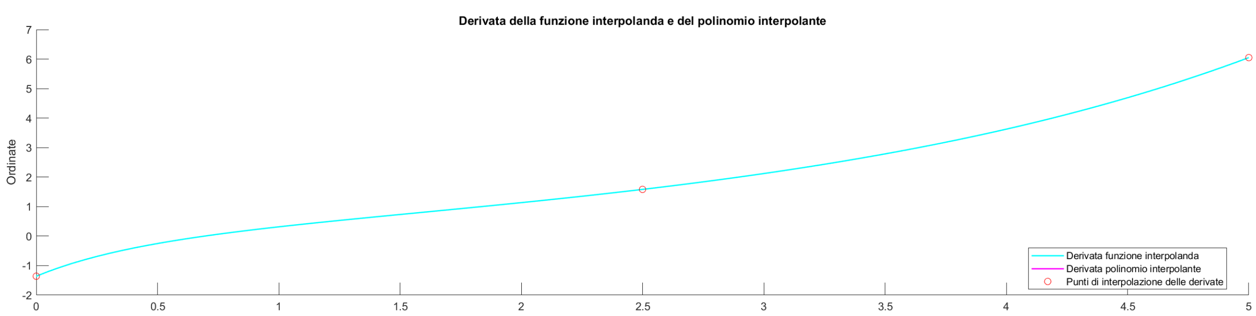


Grafico derivata prima funzione interpolanda, e derivata prima del polinomio interpolante

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.

Soluzione: di seguito è riportato il codice Matlab.

```
function x = cheby(n,a,b)
% function x = cheby(n,a,b)
% Input: n: grado del polinomio interpolante, a,b: estremi dell'intervallo di interpolazione
% Output: x: ascisse di Chebyshev ricavate
% Ricava le ascisse di Chebyshev nell'intervallo [a,b] per un polinomio interpolante di grado n
if(n<0), error('Grado del polinomio interpolante non valido!'),end
if(a>=b), error('Intervallo definito in maniera non corretta!'),end
n=n+1;
x(n:-1:1)=(a+b)/2+((b-a)/2)*cos(((2*(1:n)-1)*pi)/(2*n));
end
```

Codice Matlab esercizio 21

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 ciascu- na 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.

Soluzione: di seguito è riportato il codice Matlab.

```
function ll = lebesgue(a, b, nn, type)
% function ll = lebesgue(a, b, nn, type)
% Input: a,b = inizio e fine intervallo, nn = vettore con specificato il grado dei polinomi,
% type = specifica che tipo di ascisse di interpolazione usare
% se 0 utilizza le ascisse equispaziate nell'intervallo [a,b] altrimenti se
% 1 utilizza le ascisse di Chebyshev.
% Output: ll = vettore delle costanti di Lebesgue per ogni grado specificato in input
if a >= b
    error('Errore: a deve essere minore di b.');
end
if any(mod(nn, 1) ~= 0) || any(nn < 0)
    error('Errore: nn deve contenere solo numeri interi non negativi.');
end
if type ~= 0 && type ~= 1
    error('Errore: type deve essere 0 o 1.');
end
ll = ones(numel(nn), 1);
xq = linspace(a, b, 10001);
```

```
for j = 1:numel(nn)
    if type == 0 % Ascisse equidistanti
        x = linspace(a, b, nn(j)+1);
    else % Ascisse di Chebyshev
        x = cheby(nn(j), a, b);
    end
    lin = lebesgue_function(x,xq);
    ll(j) = max(abs(lin));
end
end
```

Codice Matlab esercizio 22

Di seguito sono riportati i vari grafici.

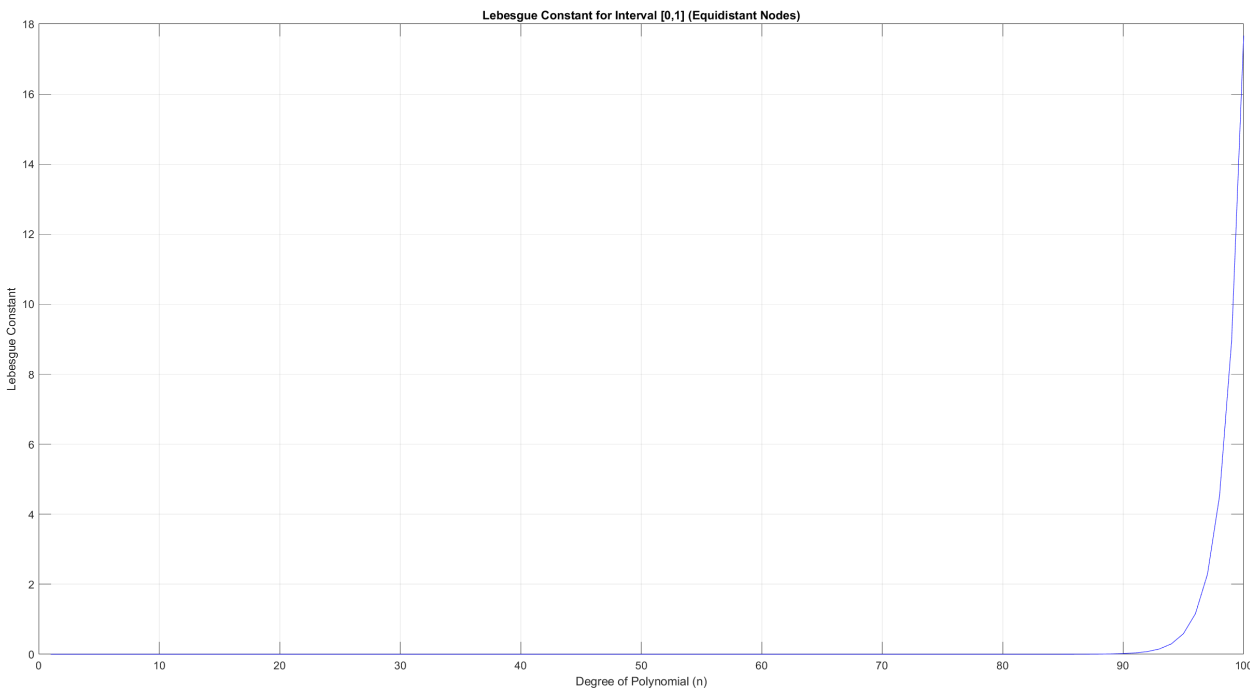


Grafico con ascisse equidistanti per intervallo [0,1]

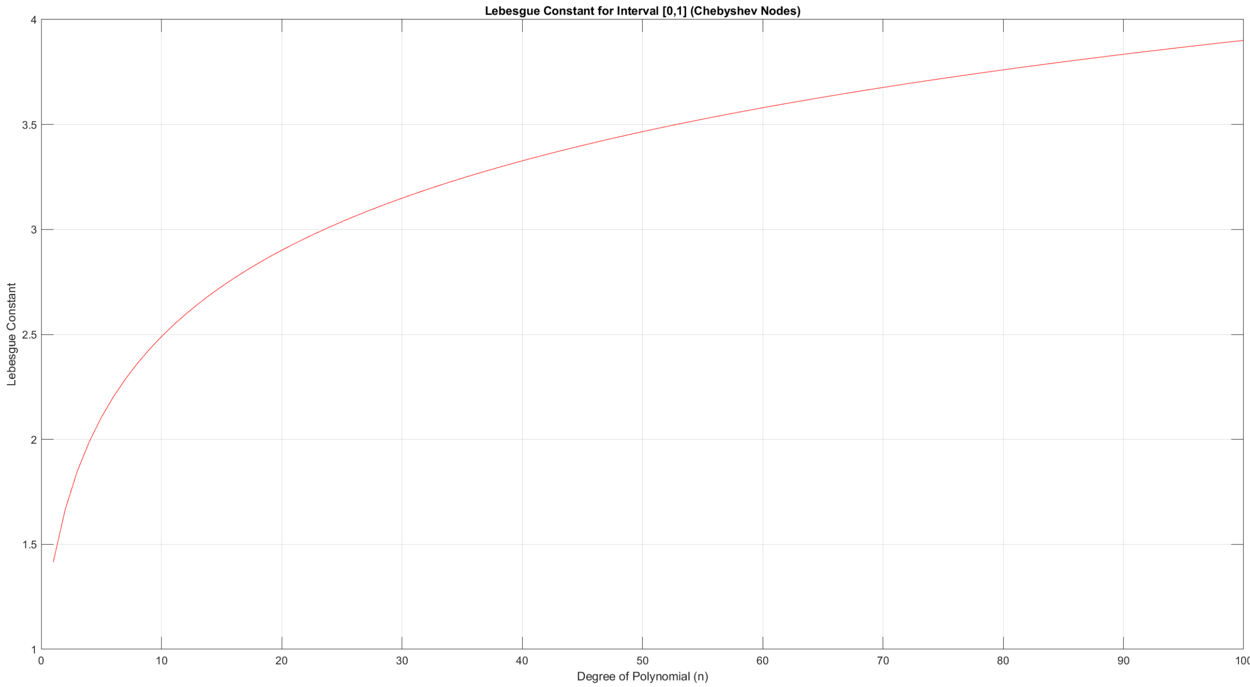


Grafico con ascisse di Chebyshev per intervallo [0,1]

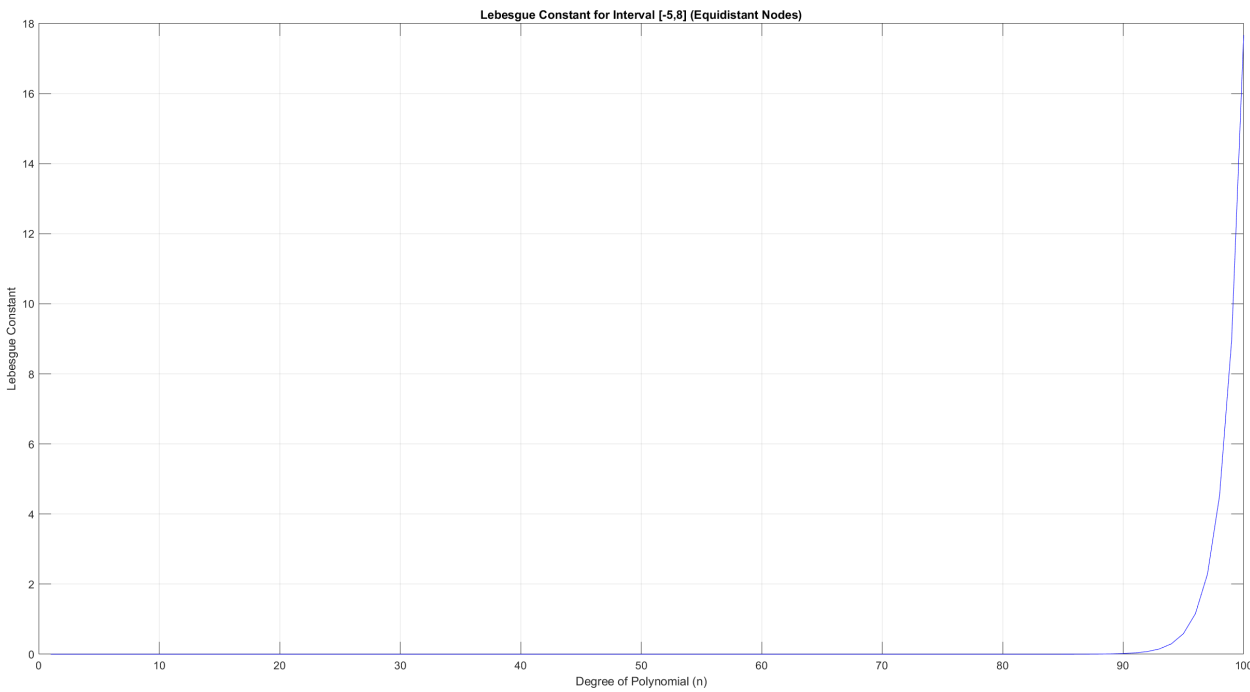


Grafico con ascisse equidistanti per intervallo [-5,8]

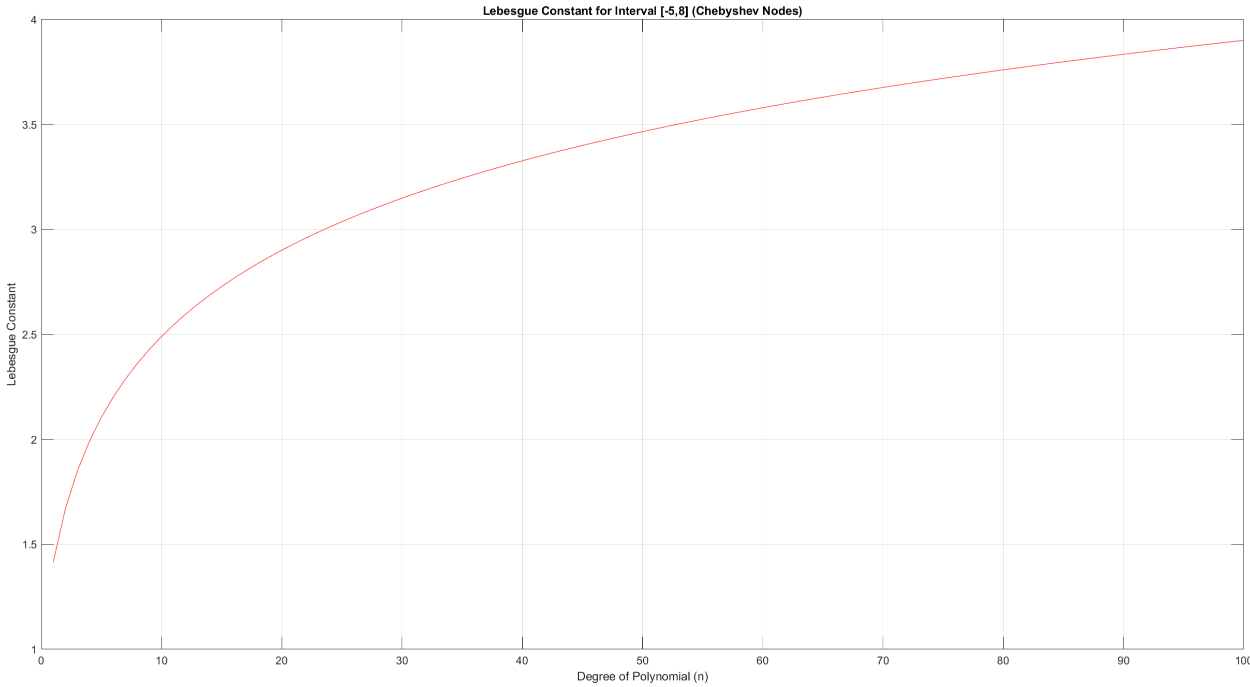


Grafico con ascisse di Chebyshev per intervallo [-5,8]

Come ci aspettavamo, la costante di Lebesgue è indipendente dall'intervallo [a,b] e con la scelta delle ascisse di Chebyshev cresce in maniera quasi ottimale, cioè logaritmica.

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 ottenerne la stima) per la funzione di Runge,

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

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

Soluzione: di seguito è riportato il codice Matlab dell'esercizio 23.

```
% Definizione della funzione di Runge
f = @(x) 1 ./ (1 + x.^2);

% Generare i punti di valutazione equispaziati nell'intervallo [-5, 5]
XQ = linspace(-5, 5, 10001)';

% Valori esatti della funzione di Runge nei punti di valutazione
YQ_exact = f(XQ);

% Preallocazione del vettore per l'errore
error_lagrange = zeros(1, 50);
error_newton = zeros(1, 50);
```

```

% Calcolo dell'errore di interpolazione per i gradi n=2:2:100
for k = 1:50
    n = 2 * k;

    % Ascisse di Chebyshev
    X = cheby(n, -5, 5)';

    % Valori della funzione di Runge nelle ascisse di Chebyshev
    Y = f(X);

    % Interpolazione con polinomio di Lagrange
    YQ_lagrange = lagrange(X, Y, XQ);

    % Interpolazione con polinomio di Newton
    YQ_newton = newton(X, Y, XQ);

    % Calcolo dell'errore massimo in valore assoluto
    error_lagrange(k) = max(abs(YQ_lagrange - YQ_exact));
    error_newton(k) = max(abs(YQ_newton - YQ_exact));
end

% Graficare l'andamento dell'errore in scala semilogaritmica
figure;
semilogy(2:2:100, error_lagrange, 'b-o', 'DisplayName', 'Lagrange');
hold on;
semilogy(2:2:100, error_newton, 'r-+', 'DisplayName', 'Newton');
hold off;
xlabel('Grado del polinomio');
ylabel('Errore massimo di interpolazione');
title('Andamento errore di interpolazione per la funzione di Runge');
legend('Location', 'NorthEast');
grid on;
```

Codice Matlab esercizio 23

Grafico in semilogy

Di seguito è riportato il grafico dell'esercizio 23.

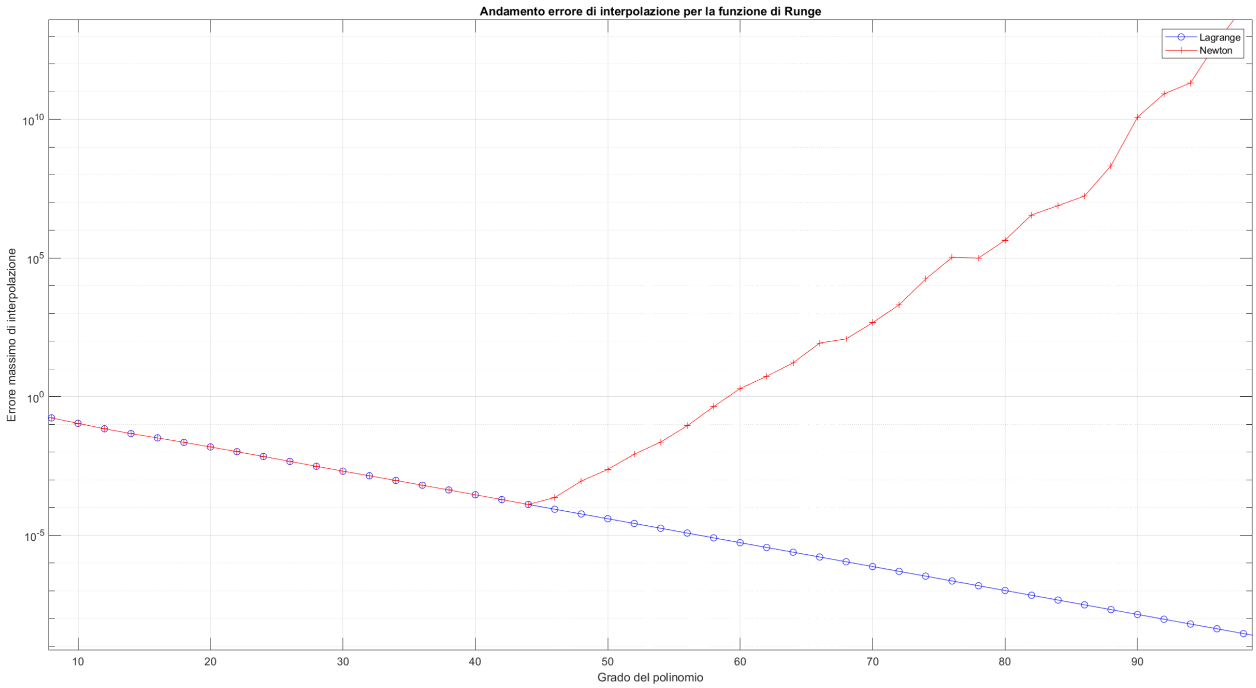


Grafico con ascisse di Chebyshev per intervallo [-5,8]

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).

Soluzione: di seguito sono è riportato il codice Matlab.

```
function yy = spline0(x, y, xq)
% yy = spline0(x, y, xq)
% funzione per il calcolo della spline cubica naturale nei punti di interpolazione x
% Input:
%   x - vettore delle coordinate x
%   y - valori della funzione alle coordinate x
%   xq - vettore dei punti in cui calcolare il polinomio
% Output:
%   yy - valori della spline nei punti xq
if any(size(x) ~= size(y)) || length(x) ~= length(unique(x))
    error('Dimensioni dei dati errate!');
end
n= length(x);
if length(y) ~= n
    error('Dati errati');
end
n=n-1;
h(1:n)= x(2:n+1) - x(1:n);
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));
```

```
alpha(1:n-1) = 2;
df= y;
for j=1:2
    for i=n+1:-1:j+1
        df(i)=(df(i) - df(i-1))/(x(i) - x(i-j));
    end
end
df=df(1, 3:n+1);
df=6*df;
% valutazione della spline
m=tridia(alpha, phi, csi, df);
m=[0, m, 0];
yy = zeros(size(xq));
j = 1;
for i=2:n+1
    ri= y(i-1) - (h(i-1)^2)/6 * (m(i-1));
    qi= (y(i) - y(i-1))/h(i-1) - h(i-1)/6*(m(i) - m(i-1));
    while j<= length(xq) && xq(j) <= x(i)
        yy(j)= ((xq(j) - x(i-1))^3*m(i)+(x(i) -xq(j))^3*m(i-1))/ ...
            (6*h(i-1))+qi*(xq(j)-x(i -1))+ ri;
        j= j+1;
    end
end
return

function x= tridia(alpha, phi, csi, x)
n= length(alpha);
for i= 1:n-1
    phi(i)=phi(i)/alpha(i);
    alpha(i+1)=alpha(i+1)- phi(i)*csi(i);
    x(i+1)=x(i+1)- phi(i)*x(i);
end
x(n)= x(n)/alpha(n);
for i=n-1:-1:1
    x(i)=(x(i) - csi(i)*x(i+1))/alpha(i);
end
return
```

Codice Matlab esercizio 24

Esercizio 25

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

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

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?

Soluzione: di seguito sono è riportato il codice Matlab.

```
f = @(x) 1 ./ (1 + x .^ 2);
a = -10;
b = 10;
xq = linspace(a, b, 10001); % Ascisse sulle quali calcolare il valore
fxq = f(xq);
e_nat = zeros(1, 200); % Matrice degli errori massimi per spline naturale
e_nak = zeros(1, 200); % Matrice degli errori massimi per spline not-a-knot
h = zeros(1, 200); % Distanze h

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

    % Genera la spline naturale usando spline0
    s_ni = spline0(xi, fi, xq);

    % Genera la spline not-a-knot usando spline
    s_naki = spline(xi, fi, xq);

    % Calcola l'errore massimo per entrambe le spline
    e_nat(index) = max(abs(fxq - s_ni));
    e_nak(index) = max(abs(fxq - s_naki));

    % Calcola la distanza h
    h(index) = 20 / n;
    index = index + 1;
end

% Plot degli errori
figure;
loglog(h, e_nat, 'b', h, e_nak, 'r');
xlabel('Distanza h');
ylabel('Errore massimo');
title('Errore massimo di interpolazione con spline naturale e not-a-knot');
legend('Naturale', 'Not-A-Knot');
grid on;
```

Codice Matlab esercizio 25

Di seguito è riportato il grafico.

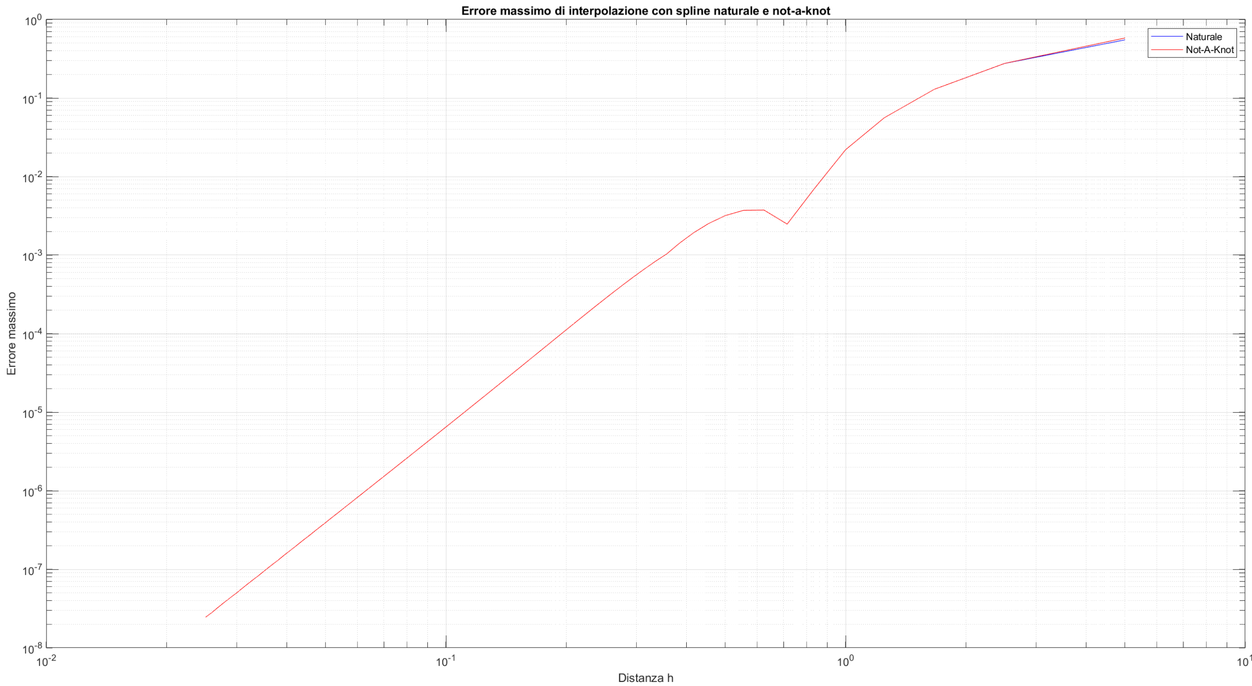


Grafico errore di approssimazioni per spline cubica naturale, not-a-knot per funzione di Runge in $[-10, 10]$

L'errore dei due tipi di spline cubica abbia lo stesso tipo di crescita che risulta essere quasi uguale inizialmente, andando poi a seguire un andamento diverso.

Esercizio 26

Graficare, utilizzando il formato loglog, l'errore di approssimazione utilizzando le *spline* interpolanti naturale e *not-a-knot* per approssimare la funzione di Runge sull'intervallo $[0, 10]$, utilizzando una partizione uniforme

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

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 i risultati ottenuti, rispetto a quelli del precedente esercizio.

Soluzione: di seguito è riportato il codice Matlab.

```
f = @(x) 1 ./ (1 + x.^ 2);
a = 0;
b = 10;
xq = linspace(a, b, 10001); % Ascisse sulle quali calcolare il valore
fxq = f(xq);
e_nat = zeros(1, 200); % Matrice degli errori spline naturale
e_nak = zeros(1, 200); % Matrice degli errori spline not-a-knot
h = zeros(1, 200); % Distanze h
index = 1;
for n = 4:4:800
```

```matlab
    xi = linspace(a, b, n+1); % Ascisse equidistanti
    fi = f(xi); % Valori della funzione sulle ascisse
    % Genera la spline naturale usando spline0
    s_ni = spline0(xi, fi, xq);
    % Genera la spline not-a-knot usando spline
    s_naki = spline(xi, fi, xq);
    % Calcola l'errore massimo per entrambe le spline
    e_nat(index) = max(abs(fxq - s_ni));
    e_nak(index) = max(abs(fxq - s_naki));
    % Calcola la distanza h
    h(index) = 10 / n;
    index = index + 1;
end

% Plot
figure;
loglog(h, e_nat, 'r', h, e_nak, 'b');
xlabel('Distanza h');
ylabel('Errore massimo');
title('Errore massimo di interpolazione con spline naturale e not-a-knot');
legend('Naturale', 'Not-A-Knot');
grid on;
```

Codice Matlab esercizio 26

Di seguito è riportato il grafico.

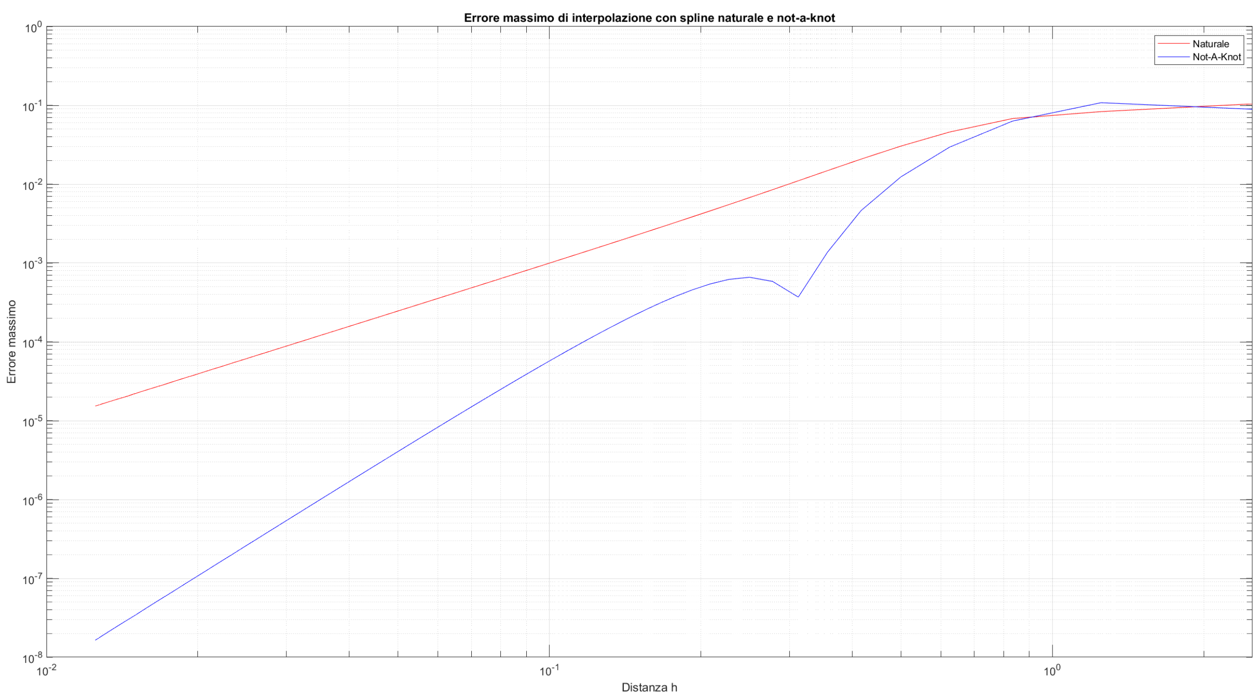


Grafico errore di approssimazioni per spline cubica naturale, not-a-knot per funzione di Runge in $[0, 10]$

Per entrambe le spline, l'errore massimo diminuisce all'aumentare del numero di nodi cioè al diminuire della distanza h. Nelle distanze molto piccole, l'errore massimo per la spline not-a-knot è inizialmente inferiore rispetto alla spline naturale. Per distanze maggiori, l'errore massimo delle due spline tende a convergere, indicando che entrambi i metodi offrono prestazioni simili quando i nodi sono distanti.

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)) * .05;
```

Graficare convenientemente i risultati ottenuti.

Soluzione: di seguito è riportato il codice Matlab.

```
rng(0);
xi = linspace(0,2*pi,101);
yi = sin(xi) + rand(size(xi)) * .05;
M = zeros(101,4);
for i = 1:101
    M(i,:) = xi(i) .^ (0:3);
end
x = miaqr(M,yi(:));
y = polyval(flip(x'), xi);
plot(xi, yi, '.', xi, y);
xlabel('x');
ylabel('y');
grid on;
```

Codice Matlab esercizio 26

Di seguito è riportato il grafico.

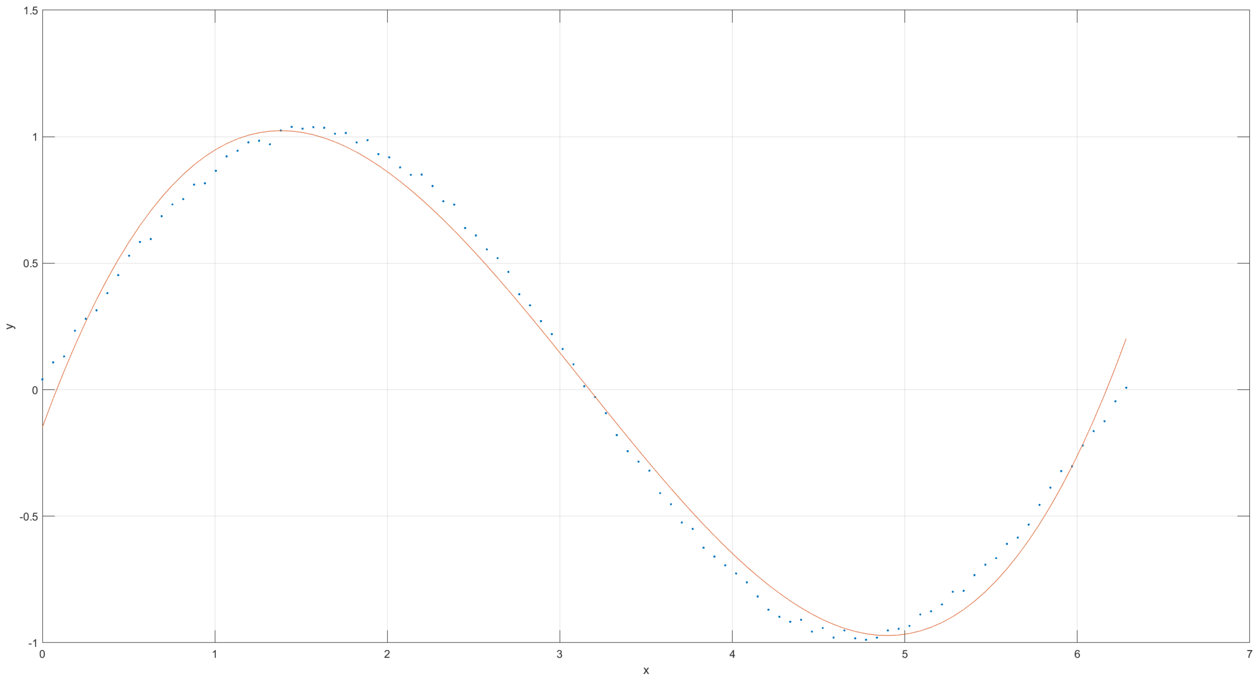


Grafico esercizio 27

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 $1, 2, \dots, 7, 9$ (come numeri razionali).

Soluzione: Di seguito è riportata la tabella contenente i pesi della quadratura della formula di Newton-Cotes di grado n.

Grado	Pesi
1	$\left[\frac{1}{2}, \frac{1}{2}\right]$
2	$\left[\frac{1}{3}, \frac{4}{3}, \frac{1}{3}\right]$
3	$\left[\frac{3}{8}, \frac{9}{8}, \frac{9}{8}, \frac{3}{8}\right]$
4	$\left[\frac{14}{45}, \frac{64}{45}, \frac{8}{15}, \frac{64}{45}, \frac{14}{45}\right]$
5	$\left[\frac{95}{288}, \frac{125}{96}, \frac{125}{144}, \frac{125}{144}, \frac{125}{96}, \frac{95}{288}\right]$
6	$\left[\frac{41}{140}, \frac{54}{35}, \frac{27}{140}, \frac{68}{35}, \frac{27}{140}, \frac{54}{35}, \frac{41}{140}\right]$
7	$\left[\frac{5257}{17280}, \frac{25039}{17280}, \frac{343}{640}, \frac{20923}{17280}, \frac{20923}{17280}, \frac{343}{640}, \frac{25039}{17280}, \frac{5257}{17280}\right]$
9	$\left[\frac{25713}{89600}, \frac{141669}{89600}, \frac{243}{2240}, \frac{10881}{5600}, \frac{26001}{44800}, \frac{26001}{44800}, \frac{10881}{5600}, \frac{243}{2240}, \frac{141669}{89600}, \frac{25713}{89600}\right]$

Pesi delle formule di Newton-Cotes per diversi gradi

Di seguito è riportato il codice Matlab.

```
function coef=calcolaCoefficientiGrado(n)
% coef=calcolaCoefficientiGrado(n)
```

```
% Calcola i pesi della quadratura della formula di quadratura di
% Newton-Cotes di grado n.
% Input
%   n: Grado (maggiore di 0) della formula di Newton-Cotes di cui vogliamo conoscere i pesi
%   della quadratura
% Output
%   coef: pesi della quadratura della formula di grado desiderato
if(n<=0), error('Valore del grado della formula di Newton-Cotes non valido'),end
coef=zeros(n+1,1);
if (mod(n,2) == 0)
    for i=0:n/2-1
        coef(i+1)=calcolaCoefficienti(i,n);
    end
    coef(n/2+1)=n-sum(coef)*2;
    coef((n/2)+1:n+1)=coef((n/2)+1:-1:1);
else
    for i=0:round(n/2,0)-2
        coef(i+1)=calcolaCoefficienti(i,n);
    end
    coef(round(n/2,0))=(n-sum(coef)*2)/2;
    coef(round(n/2,0)+1:n+1)=coef(round(n/2,0):-1:1);
end
return
end

function cin=calcolaCoefficienti(i,n)
%Calcola il peso della quadratura della formula di Newton-Cotes numero i di grado n
d=i-[0:i-1 i+1:n];
den=prod(d);
a=poly([0:i-1 i+1:n]);
a=[a./((n+1):-1:1) 0];
num=polyval(a,n);
cin=num/den;
end
```

Codice Matalb esercizio 28

Esercizio 29

Scrivere una function Matlab, `[If,err] = composita( 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.

Soluzione: di seguito è riportato il codice Matlab.

```
function [If , err] = composita (fun, a, b, k, n)
% [If , err] = composita (fun, a, b, k, n)
% Input:
%   fun: funzione integranda, a,b: estremi sinistro e destro dell' intervallo di integrazione
%   k: grado della formula di quadratura composita di Newton-Cotes
%   n: numero di sottointervalli in cui suddividere l'intervallo di integrazione
% Output:
%   If: approssimazione dell'integrale ottenuta, err: stima dell'errore di quadratura.
if k<1
    error("Grado k errato");
end
if a > b
    error ('Estremi intervallo di integrazione errati');
end
if(mod(n, k) ~= 0 || mod(n/2, 2)~= 0)
    error("n non multiplo pari di k");
end
tol = 1e-3;
mu = 1 + mod (k ,2);
c = calcolaCoefficientiGrado(k);
x = linspace (a ,b , n +1);
fx = feval ( fun , x );
h = (b - a )/ n ;
If1 = h * sum ( fx (1: k +1).* c (1: k +1));
err = tol + eps ;
while tol < err
    n = n *2; % raddoppio i punti
    x = linspace (a ,b , n +1);
    fx (1:2: n +1) = fx (1:1: n /2+1);
    fx (2:2: n ) = feval ( fun , x (2:2: n ));
    h = (b - a )/ n ;
    If = 0;
    for i = 1: k +1
        If = If + h * sum ( fx ( i : k : n ))* c ( i );
    end
    If = If + h * fx ( n +1)* c ( k +1);
    err = abs ( If - If1 )/(2^( k + mu ) -1);
    If1 = If ;
end
return
end
```

Codice Matlab esercizio 29

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.

Soluzione: di seguito è riportato il codice Matlab.

```matlab
% Definizione della funzione integranda
fun = @(x) exp(3 * x);

% Estremi dell'intervallo di integrazione
a = 0;
b = 1;

% Valore esatto dell'integrale
I_exact = (1/3) * (exp(3) - 1);

% Valori di k e n
k_values = [1, 2, 3, 6];
n = 12;

% Preallocazione delle variabili per memorizzare i risultati
If_values = zeros(size(k_values));
err_estimated = zeros(size(k_values));
err_actual = zeros(size(k_values));

% Ciclo sui valori di k
for i = 1:length(k_values)
    k = k_values(i);
    [If, err] = composita(fun, a, b, k, n);
    If_values(i) = If;
    err_estimated(i) = err;
    err_actual(i) = abs(I_exact - If);
end

% Tabulazione dei risultati
fprintf('k\tI_approx\t\tErr_estimated\t\tErr_actual\n');
for i = 1:length(k_values)
    fprintf('%d\t%.10f\t\t%.10f\t\t%.10f\n', k_values(i), If_values(i),
        err_estimated(i), err_actual(i));
end
```

Codice Matlab esercizio 30

Descrizione dei Risultati

- **k:** Rappresenta il grado della formula di quadratura di Newton-Cotes utilizzata per l'approssimazione dell'integrale.

k	Approssimazione Integrale	Errore stimato	Errore reale
1	6.3639164195	0.0008872455	0.0020707785
2	6.3618461801	0.0000011534	0.0000005390
3	6.3618468534	0.0000005849	0.0000012123
6	6.3618456411	0.0000000000	0.0000000000

Risultati dell'approssimazione dell'integrale con diversi gradi k e $n = 12$

- **Approssimazione Integrale:** È l'approssimazione dell'integrale ottenuta utilizzando la formula di quadratura composita di Newton-Cotes.
- **Errore stimato:** Rappresenta la stima dell'errore di quadratura.
- **Errore reale:** Indica l'errore reale rispetto al valore esatto dell'integrale.

Analisi dei Risultati

- Per $k = 1$, l'approssimazione dell'integrale è leggermente superiore al valore esatto, con un errore stimato e reale relativamente piccolo.
- Aumentando k, l'approssimazione dell'integrale diventa sempre più vicina al valore esatto, e l'errore stimato diminuisce in modo significativo.
- Per $k = 6$, l'approssimazione dell'integrale coincide praticamente con il valore esatto, e sia l'errore stimato che quello reale sono nulli, indicando una precisione molto elevata dell'approssimazione.

In generale, l'approssimazione dell'integrale diventa più accurata aumentando il grado della formula di quadratura di Newton-Cotes utilizzata.