Relazione Elaborato Calcolo Numerico

Autori:
Carmelo Casula 7085019 carmelo.casula@edu.unifi.it
Jorge Fabrizio Leandro Borjas Bringas 7072734 jorge.borjas@edu.unifi.it

November 2024

Esercizio 1

Dimostra che:

$$\frac{25f(x) - 48f(x-h) + 36f(x-2h) - 16f(x-3h) + 3f(x-4h)}{12h} = f'(x) + O(h^4) \tag{1}$$

Svolgimento

Per verificare l'Equazione 1 bisogna scrivere gli sviluppi di Taylor centrati in x delle funzioni

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

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

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

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

Scrivo di nuovo il numeratore dell'equazione 1 sostituendo le funzioni f(x-h),f(x-2h),f(x-3h) e f(x-4h) con gli sviluppi appena ricavati

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

$$+36 * (f(x) - 2hf'(x) + \frac{4h^2 f''(x)}{2} - \frac{8h^3 f'''(x)}{6} + \frac{16h^4 f''''(x)}{24} + O(h^5))$$
$$-16 * (f(x) - 3hf'(x) + \frac{9h^2 f''(x)}{2} - \frac{27h^3 f'''(x)}{6} + \frac{81h^4 f''''(x)}{24} + O(h^5))$$
$$+3 * (f(x) - 4hf'(x) + \frac{16h^2 f''(x)}{2} - \frac{64h^3 f'''(x)}{6} + \frac{256h^3 f'''(x)}{24} + O(h^5))$$

Continuo con lo sviluppo del numeratore

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

Quindi l'equazione 1 diventa, dopo aver sostituito il numeratore con il risultato

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

Esercizio 2

La funzione

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

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.

Svolgimento

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

Risultati

Risultati ottenuti dallo script: Il minimo si verifica in x=1 con il valore f(x)=2, il limite mentre la funzione si avvicina da destra a 4/3 vale 2.7078, mentre da sinistra 2.7025

Grafico

Ecco il grafico ottenuto dall'esecuzione:

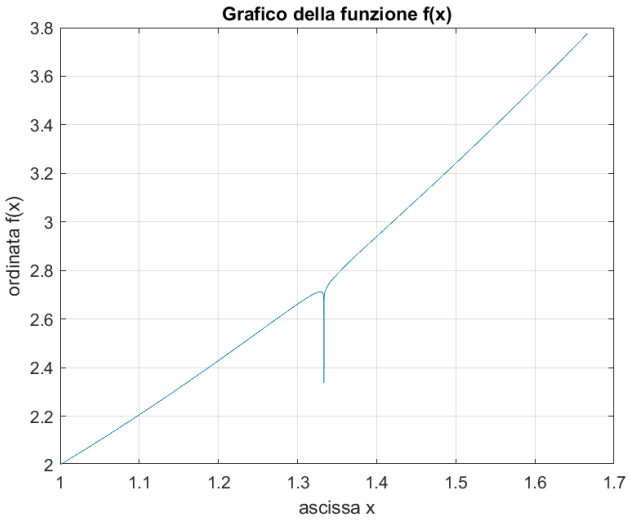


Grafico funzione $f(x) = 1 + x^2 + \frac{log(|3(1-x)+1|)}{80}$

Esercizio 3

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

Svolgimento

La cancellazione numerica si verifica quando il risultato della somma di addendi quasi opposti comporta una perdita di cifre significative. Questo fenomeno riflette il malcondizionamento dell'operazione. Se x e y sono i due numeri da sommare, il numero di condizionamento è dato da:

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

Esempio

Dati p1 e p2 due numeri reali tali che p1=0.123321, p2=0,123209, la cui differenza d vale:

$$d = p1 - p2 = 0,000112 = 1,12 * 10^{-4} \tag{2}$$

Supponendo che l'architettura del calcolatore permette di memorizzare solo le prima 4 cifre dopo la virgola, quindi i due numeri per essere rappresentati all'interno del nostro calcolatore verrebbero troncati appena dopo la quarta cifra quindi t1=0.1233 e t2 0.1232, facendo la sottrazione e riscrivendo il risultato in virgola mobile avremmo:

$$dt = t1 - t2 = 0,0001 = 1 * 10^{-4}$$

Che è un risultato diverso da quello dell'equazione 2

Esercizio 4

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

Svolgimento

```
    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 non corretti');
end
if tolx<=0
    error('Tolleranza non corretta');
end
count = 0;
fa = feval(f,a);
fb = feval(f,b);
if fa*fb>=0
    error('Intervallo di confidenza non corretto')
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
```

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).

Svolgimento

Metodo 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 errata');
end
it = 0;
count=0;
x = x0 ;
m = 5;
for i =1: maxit
    x0 = x ;
    fx = feval (f , x0 );
    f1x = feval ( f1 , x0 );
    count=count+2;
    if f1x==0
        error('Derivata prima uguale a zero! Metodo non convergente');
    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
    it = it + 1;
```

```
    if abs (x - x0 ) <= tolx *(1+ abs ( x ))
        break
    end
end
if ( abs (x - x0 ) > tolx *(1+ abs ( x )))
    disp (' Il metodo non converge ')
end
end
```

Metodo 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
```

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.

Soluzione

Risultati con i metodi di Seccanti, Bisezione e Newton

Risultati dei metodi di Secanti, Bisezione e Newton.

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

Svolgimento

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:

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

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

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

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

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

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

Tabulazione dei risultati

Di seguito è riportata la tabella contenente i risultati dei vari metodi.

Risultati dei metodi di Secanti, Bisezione, Newton e Newton modificato

Metodo	Tolleranza	Radice	Numero 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}	non converge	-	-
Newton	10^{-9}	non converge	-	-
Newton	10^{-12}	non converge	-	-
Newton modificato	10^{-3}	non converge	-	-
Newton modificato	10^{-6}	non converge	-	-
Newton modificato	10^{-9}	non converge	-	-
Newton modificato	10^{-12}	non converge	-	-

Commento risultati

Osservando la tabella si possono considerare alcune cose: 1) Non tutti imetodi convergono alla soluzione per tutti i livelli di tolleranza; 2)Il numero di iterazioni non è stabile, aumenta al variare della tolleranza 3)Confrontando i risultati dei metodi osserviamo che il metodo di bisezione è il migliore in quanto converge più velocemnete rispetto agli altri

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.

Svolgimento

```
    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: la matrice in input non e 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: la matrice e 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
```

Esempio 1

Il codice restituisce un'errore in caso sia data in input una matrice singolare, data una matrice:

$$A = \begin{bmatrix} 2 & 4 & 6 \\ 1 & 3 & 5 \\ 4 & 8 & 12 \end{bmatrix} \in \mathbb{R}^{3\times 3}$$

verrà visualizzato "Errore la matrice e singolare"

Esempio 2

Viene restituito un'errore nel caso si dia in input una matrice non quadrata come questa:

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

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

Esempio 3

Il codice restituisce la soluzione del sistema con la seguente matrice:

$$A = \begin{bmatrix} 2 & 1 & 3 \\ 0 & 1 & 4 \\ 5 & 2 & 1 \end{bmatrix} \in \mathbb{R}^{3\times 3}$$

e un vettore:

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

verranno restituite le soluzioni: $-6.6667, 33.3333$ e-3.3333

Esempio 4

Viene restituito un'errore nel caso la lunghezza di b non è compatibile con il numero di righe/colonne di A, come ad esempio:

$$A = \begin{bmatrix} 2 & 1 & 3 \\ 0 & 1 & 4 \\ 5 & 2 & 1 \end{bmatrix} \in \mathbb{R}^{3\times 3}$$

e un vettore:

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

viene restituito "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 LDLT . 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.

```
function x = mialdl(A, b)
%
%   x = mialdl(A, b);
%
%   Risolve il sistema lineare Ax = b, con A matrice simmetrica e definita
%   positiva, mediante fattorizzazione LDLT di A.
%
%   Input:
%       A - matrice dei coefficienti;
%       b - vettore dei termini noti.
%
%   Output:
%       x - vettore soluzione.
%
    [m,n] = size(A);
    if m ~= n
        error('Matrice non quadrata');
    end
    k = length(b);
    if k ~= n
        error('Vettore dei termini noti errato');
    end

    if A(1,1) <= 0
        error('Matrice non sdp');
    end
    x = b;
    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('Matrice non sdp');
        end
        A(j+1:n,j) = (A(j+1:n,j) - A(j+1:n,1:j-1) * v) / A(j,j);
    end

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

    %risolviamo per D
    for i = 1:n
        x(i) = x(i) / A(i,i);
    end

    %risolviamo per L'
    for i = n:-1:2
        x(1:i-1) = x(1:i-1) - A(i,1:i-1).' * x(i);
    end
```

```
return
```

Esercizio 10

Scrivere una function Matlab, `function [x,nr] = miaqr(A,b)` che, data in ingresso la matrice A $m \times n$, con $m \geq n = rank(A)$, 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.

Svolgimento

```
    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
```

Esempio 1

Viene restituito un messaggio di errore se viene data una matrice non quadrata

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

viene restituito: "Errore: La matrice deve essere quadrata".

Esempio 2

Viene restituito un messaggio di errore se la lunghezza di b è incompatibile con il numero di righe/colonne di A:

$$A = \begin{bmatrix} 1 & 13 & 3 \\ 9 & 5 & 10 \\ 21 & 7 & 14 \end{bmatrix} \in \mathbb{R}^{3\times 3}$$

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

viene restituito: "Errore: la dimensione del vettore b non coincide con la dimensione della matrice A".

Esempio 3

Viene restituito un messaggio di errore se la matrice A è simmetrica ma non definita positiva

Esempio 4

Viene restituito un messaggio di errore se la matrice è definita positiva ma non simmetrica

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

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

viene restitu: "Errore: matrice A non simmetrica"

Esempio 5

Vengono restituite le soluzioni del sistema lineare, con una matrice dei coefficenti simmetrica definita positiva

$$A = \begin{bmatrix} 4 & 1 & 2 \\ 1 & 3 & 0 \\ 2 & 0 & 5 \end{bmatrix} \in \mathbb{R}^{3\times 3}$$

e un vettore dei termini noti

$$\mathbf{b} = \begin{pmatrix} 7 \\ 4 \\ 6 \end{pmatrix} \in \mathbb{R}^3$$

vengono restituite le soluzioni 1.1395, 0.9535 e 0.7442

Esercizio 11

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

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

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

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

Svolgimento

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

% Genera la matrice
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
b = b.'; % vettore b trasposto
% Richiamo della funzione mialu
x = mialu(A,b);

% Stampa del risultato
disp(x);

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

% Calcolo del vettore errore
e = b - A * x;
disp('Vettore errore:');
disp(e);
```

Risultati

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

Valutazione risultati

Sebbene i risultati ottenuti con la funzione Mialu sembrino corretti, il numero di condizionamenti nella matrice è piuttosto elevato (numero di condizionamenti: 2.45×10^{14}). Alcuni Un condizionamento elevato indica che il sistema è malcondizionato e può portare a cambiamenti nei risultati. Tuttavia, se i valori del vettore errore, vedrai che sono molto piccoli, i risultati del vetore errore sono: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

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

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

Svolgimento

Codice matlab function mialdlt

```
    function A = mialdlt(A)
%
%   A = mialdlt(A);
%
%   Calcola la fattorizzazione LDL' della matrice A sovrascrivendo la
%   porzione strettamente triangolare inferiore di A, con la porzione
%   strettamente triangolare inferiore di L, e sovrascrivendo la diagonale
%   di A con il fattore D.
%
%   Input:
%       A - matrice dei coefficienti.
%
%   Output:
%       A - matrice dei coefficienti in input, sovrascritta nella sua
%       porzione strettamente triangolare inferiore con la porzione
%       strettamente triangolare inferiore del fattore L e con la diagonale
%       sovrascritta dal fattore D.
%
    [m,n] = size(A);
    if m ~= n
        error('Errore: la matrice non e quadrata');
    end
    if A(1,1) <= 0
        error('Errore: la matrice non e sdp');
    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('Errore: la matrice non e 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
```

Script d'esecuzione

```
    hold on;

for n = 2:100
    % Costruisci la matrice An
    An = diag(n * ones(n, 1)+1) - ones(n);

    % Fattorizza An utilizzando mialdlt
    Af = mialdlt(An);

    % Plot
    plot(1:n, diag(Af), 'LineWidth', 2);
end

% fattorizzazione e plot per n = 1
A1 = 1;
Af1 = mialdlt(A1);
plot(1, diag(Af1), '.', 'LineWidth', 2);

xlabel('Indice diagonale');
ylabel('Elemento diagonale di D');
title('Elementi diagonali di D rispetto all''indice diagonale');
grid on;
hold off;
```

Grafico esecuzione

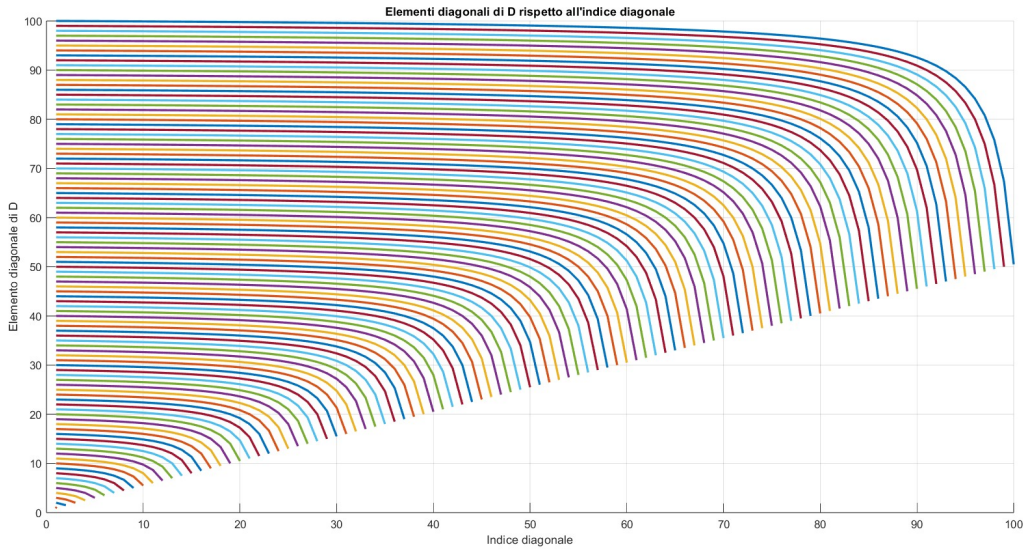


Grafico della funzione

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.

Svolgimento

Procedimento per la risoluzione dell'esercizio 13

Calcolo dei pesi

La matrice dei pesi B è una matrice diagonali con le radici quadrate dei pesi sui termini diagonali

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

Modifica del sistema

Si moltiplica la matrice A e il vettore b per la matrice W

$$\tilde{A} = BA, \quad \tilde{b} = Bb$$

Fattorizzazione e risoluzione del sistema

Definiamo il sistema e la funzione di analisi utilizzando la funzione miaqr. Di seguito è riportato il codice per l'esercizio 13.

```
    format long
% Definizione della matrice A e del vettore b
A = [7 2 1
     8 7 8
     7 0 7
     4 3 3
     7 0 10];
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;

% cosi facendo, si minimizza la norma pesata
[x, pw] = miaqr(A, b);

disp('La soluzione nel senso dei minimi quadrati x e: ');
disp(x);
fprintf('La norma pesata del residuo (p_\x03C9) e: %.15f\n', pw);
```

Conclusioni

La risoluzione 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 $f(x) = 0$.

Svolgimento

```
function [x, nit] = newtonNoPivoting(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 e il gradiente di una funzione f(x) di
%           cui vogliamo approssimare una radice, mentre jacobian
%           e la 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
    )
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
```

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.

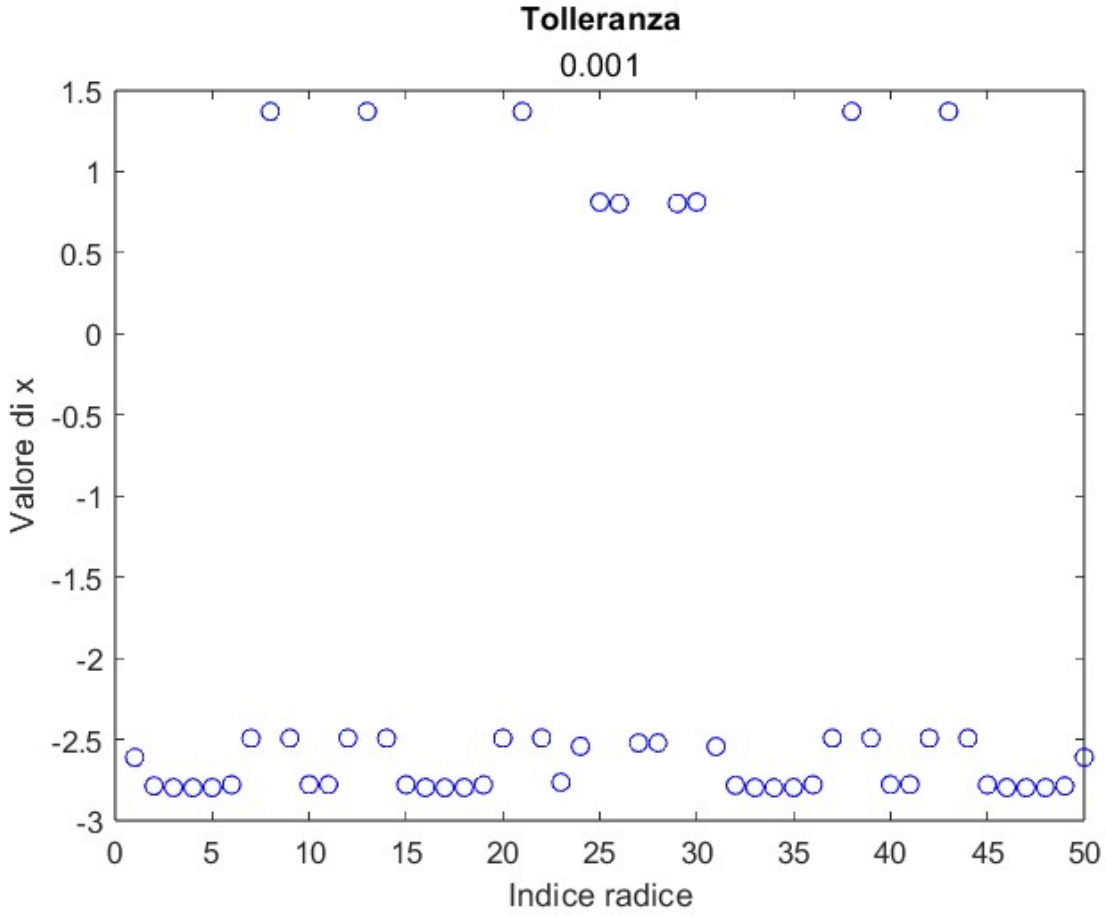
Svoglimento

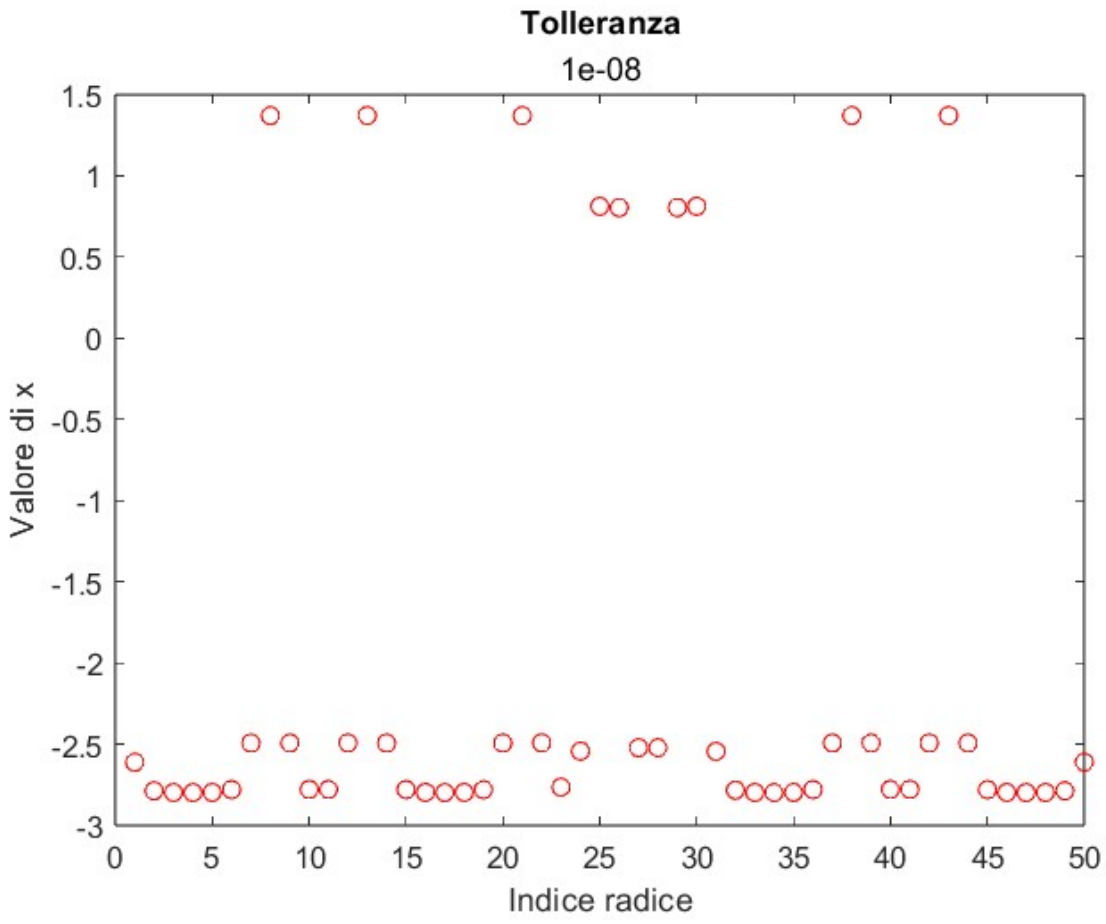
Script matlab

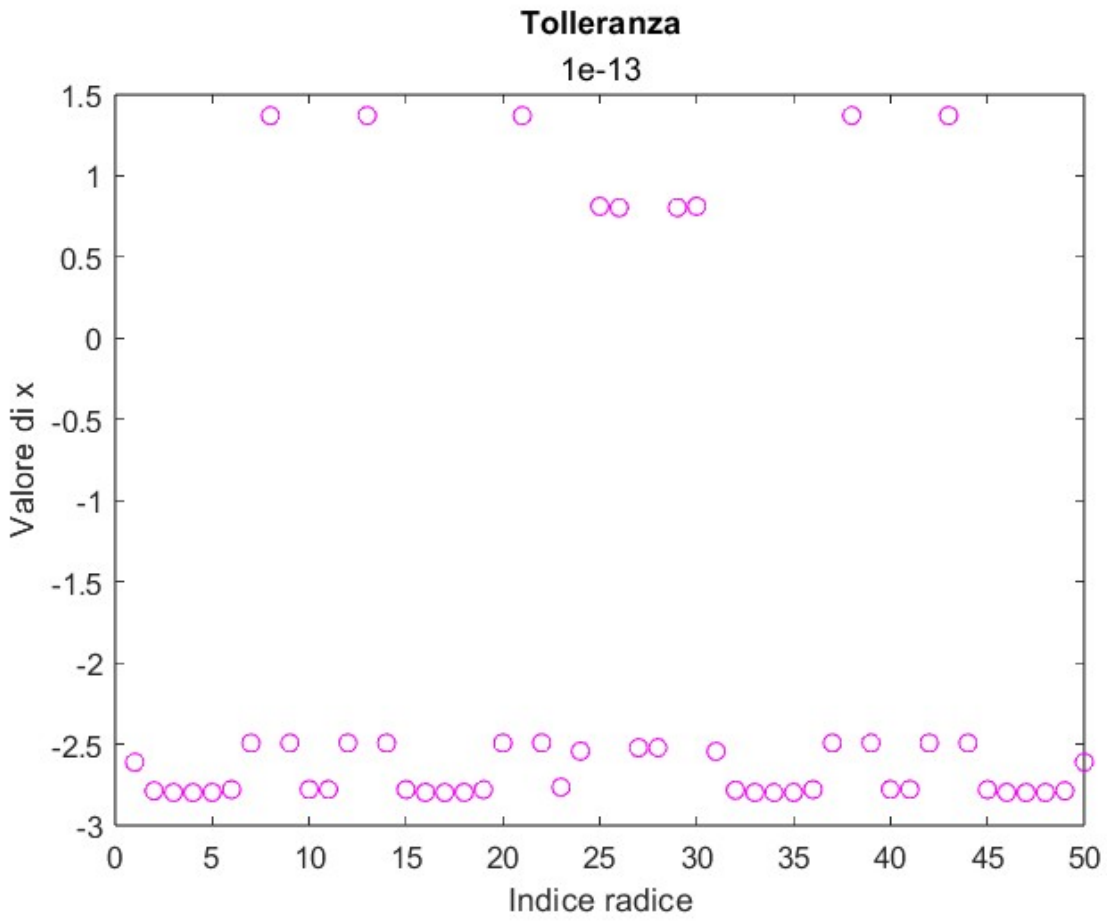
```
   format long
n = 50;
x0 =zeros(n,1);
tol = [1e-3, 1e-8, 1e-13];
color = ['b', 'r', 'm'];
L = zeros(50, 3);
for i = 1:length(tol)
    [x,nit] = newtonNoPivoting(@fun, x0, tol(i));
    figure;
    plot(1:n, x, 'o', Color=color(i));
```

```
    title('Tolleranza',num2str(tol(i)));
    xlabel('Indice radice');
    ylabel('Valore di x');
    L(:,i) = x;
end
disp(L);
```

1 Grafici




Tolleranza
1e-08
Valore di x
Indice radice
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2.5
-3
0
5
10
15
20
25
30
35
40
45
50



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.

Svolgimento

```
    function ll = lagrange(x, y, xx)
%
% function ll = lagrange(x, y, xx)
%
% Calcola il polinomio interpolante di Lagrange nei punti specificati in xx.
%
% INPUT:
% x         - vettore delle ascisse dei punti di interpolazione
% y         - vettore delle ordinate dei punti di interpolazione
% xx        - vettore dei punti in cui calcolare il polinomio interpolante
%
% OUTPUT:
% ll        - valori del polinomio interpolante nei punti xx
```

```
%

% Inizializzare il vettore dei valori del polinomio interpolante
ll = zeros(size(xx));
 % Loop su ciascun punto di interpolazione
for k = 1:length(x)
    % Inizializzare il termine di Lagrange Lk(x)
    Lkn = ones(size(xx));
    % Calcolare il termine di Lagrange Lk(x) per ogni punto xx
    for j = 1:length(x)
        if k ~= j
            Lkn = Lkn .* ((xx - x(j)) / (x(k) - x(j)));
        end
    end
    % Aggiungere il termine di Lagrange pesato al polinomio interpolante
    ll = ll + y(k) * Lkn;
end
end
```

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.

Svolgimento

```
    function YQ = newton_es17(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('Il numero di ascisse di interpolazione e di
    valori della funzione non e uguale!'),
end
if (length(X) ~= length(unique(X))), error('Le ascisse di interpolazione non
    sono 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 non corretti'), 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
```

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.

Svolgimento

```
    function yy = hermite(xi, fi, f1i, xx)
%
%   yy = hermite(xi, fi, f1i, xx);
%
%   Calcola il polinomio interpolante di Hermite definito dalle
%   terne di dati (xi, fi, f1i) nei punti del vettore xx.
%
%   Input:
%       (xi,fi,f1i) - sono i dati del problema, dove fi ed f1i indicano
%           rispettivamente il valore che le funzioni f(x) e f'(x)
%           assumono nel punto xi;
%       xx - vettore delle ascisse, non ancora raddoppiato, in cui
%           valutare il polinomio.
%
%   Output:
```

```
%       yy - polinomio interpolante in forma di Hermite valutato nei punti
%            del vettore delle ascisse xx opportunamente raddoppiato.
%
    if nargin < 4
        error('Parametri in ingresso insufficienti.');
    end
    n = length(xi);
    if n ~= length(fi) || n ~= length(f1i) || length(unique(xi)) ~= n
        error('parametri in ingresso errati');
    end

    x = zeros(1,2*n); % vettore delle ascisse raddoppiato
    y = zeros(1,2*n);
    x(1:2:2*n-1) = xi;
    x(2:2:2*n) = xi;
    y(1:2:2*n-1) = fi;
    y(2:2:2*n) = f1i;
    y = diff_div_hermite(x,y); % calcolo delle differenze divise

    n = n * 2;
    % valutazione del polinomio interpolante mediante algoritmo di
    % Horner generalizzato
    yy = y(n) * ones(1,length(xx));
    for i = n-1:-1:1
        yy = yy .* (xx - x(i)) + y(i);
    end
return
```

Esercizio 19

Si consideri la seguente base di Newton,

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

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.

Svolgimento

```
    function p_prime = horner(x, a, x_nodes)
%
% p_prime = derivata_horner(x, a, x_nodes)
```

```
%
% derivata_horner calcola la derivata di un polinomio in base Newton
% Utilizza l'algoritmo di Horner modificato per calcolare la
% derivata polinomio rappresentato nella base di Newton.
%
% INPUT:
%   x           - Il punto in cui calcolare la derivata del polinomio.
%   a           - Coefficienti del polinomio (array di numeri).
%   x_nodes     - Nodi del polinomio di Newton (array di numeri).
%
% OUTPUT:
%   p_prime     - Derivata del polinomio in x.
%

n = length(a); % Numero dei coefficienti
omega = ones(1, n); % Array per i valori di omega_i(x),
                    % inizializzato a 1
omega_prime = zeros(1, n); % Array per i valori di omega_i'(x),
                           % inizializzato a 0
% Calcola omega_i(x) per ogni i
for i = 2:n
    for j = 1:i-1
        omega(i) = omega(i) * (x - x_nodes(j));
    end
end

% Calcola omega_i'(x) per ogni i
for i = 2:n
    for k = 1:i
        product = 1;
        for j = 1:i
         if j ~= k
                product = product * (x - x_nodes(j));
            end
        end
           omega_prime(i) = omega_prime(i) + product;
    end
end

% Applica il metodo di Horner modificato per calcolare
% la derivata del polinomio
p_prime = 0;
for i = n:-1:1
    p_prime = p_prime * x + a(i) * omega_prime(i);
end
end
```

Esercizio 20

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

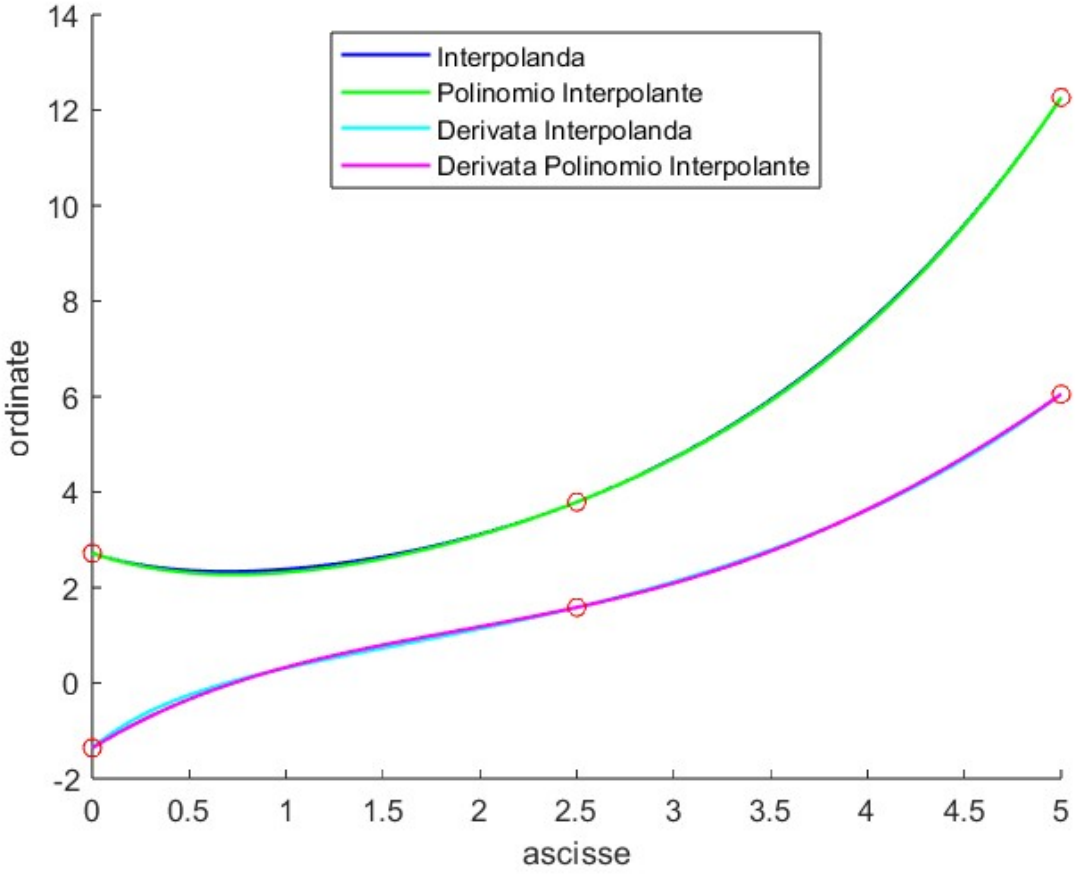
$$f(x) = exp(x/2 + exp(-x))$$

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

Svolgimento

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

%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);

yy = hermite(xi, fi, f1i, xx); % Polinomio interpolante di Hermite

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 = diff_div_hermite(x,y);
y = horner_per_esercizio20(x,dd,xx); % Calcolo della derivata prima
    % del polinomio interpolante nei punti xx

hold on;
plot(xx, f(xx), 'b', 'LineWidth', 1.2);% Plot della funzione interpolanda in
    blu
plot(xx, yy, 'g', 'LineWidth', 1.2); % Plot del polinomio interpolante in
    verde
plot(xx, f1(xx), 'c', 'LineWidth', 1.2); % Plot della derivata della funzione
    interpolanda in rosso
plot(xx, y, 'm', 'LineWidth', 1.2); % Plot della derivata del polinomio
    interpolante in magenta
plot(xi, f(xi), 'ro'); % Condizioni di interpolazione p(xi) = f(xi)
plot(xi, f1(xi),'ro'); % Condizioni di interpolazione p'(xi) = f'(xi)
xlabel('ascisse');
ylabel('ordinate');
hold off;

legend('Interpolanda', 'Polinomio Interpolante', 'Derivata Interpolanda', ...
    'Derivata Polinomio Interpolante', 'Location', 'best');
```

Grafico



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.

Svolgimento

```
    function x = chebyshev(n, a, b)
%
% x = chebyshev(n, a, b)
%
% La funzione calcola le n+1 ascisse di Chebyshev nell'intervallo
% [a, b]
%
% INPUT:
%   n          - Numero di ascisse che vogliamo calcolare
%   [a, b]     - Intervallo in cui vengono calcolate le ascisse di
%                Chebyshev
```

```
%
% OUTPUT:
%     x           - Ascisse di Chebyshev calcolate sull'intervallo [a, b]
%
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));
return
```

Esercizio 22

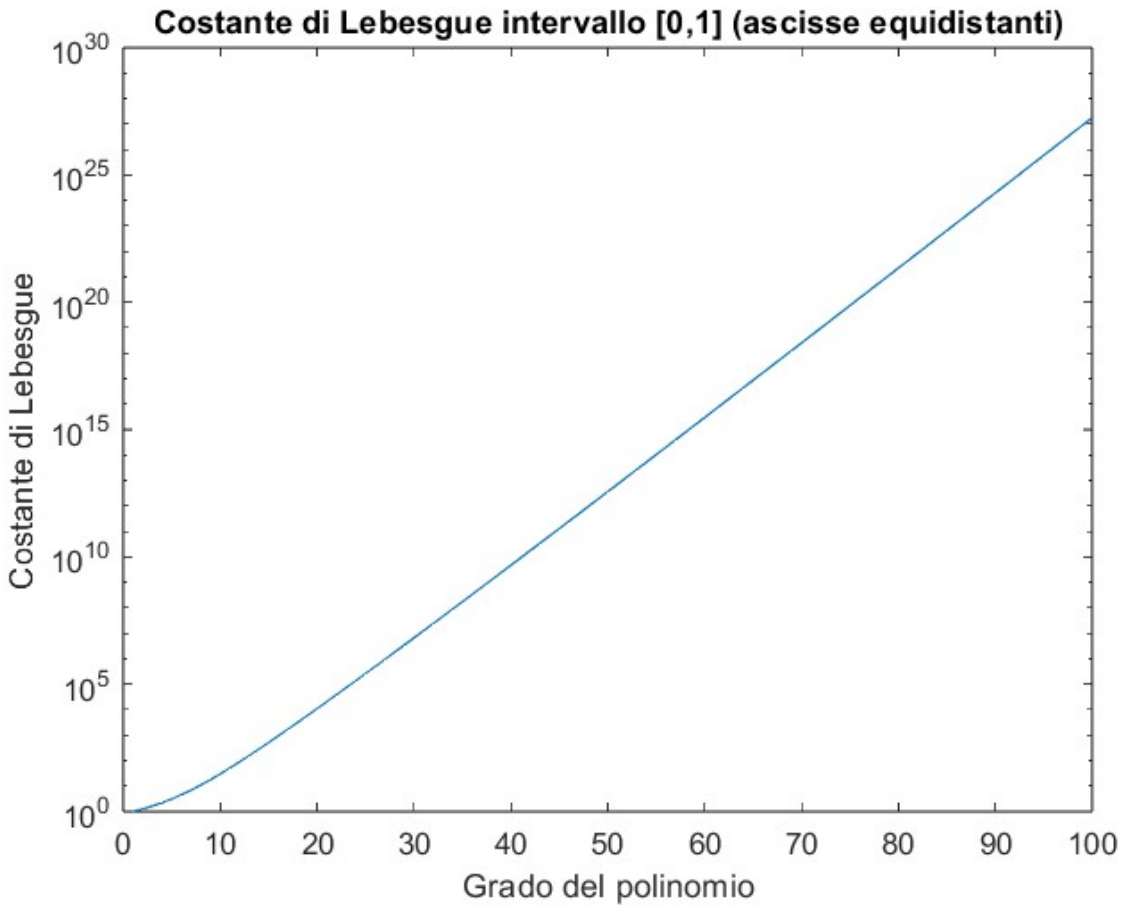
Costruire una function Matlab, con sintassi **ll = lebesgue(a, b, nn, type)**, che approssimi la costante di Lebesgue per l'interpolazione polinomiale sull'intervallo [a,b], per i polinomi di grado specificato nel vettore nn, utilizzando ascisse equidistanti, se type=0, o di Chebyshev, se type=1 (utilizzare 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.

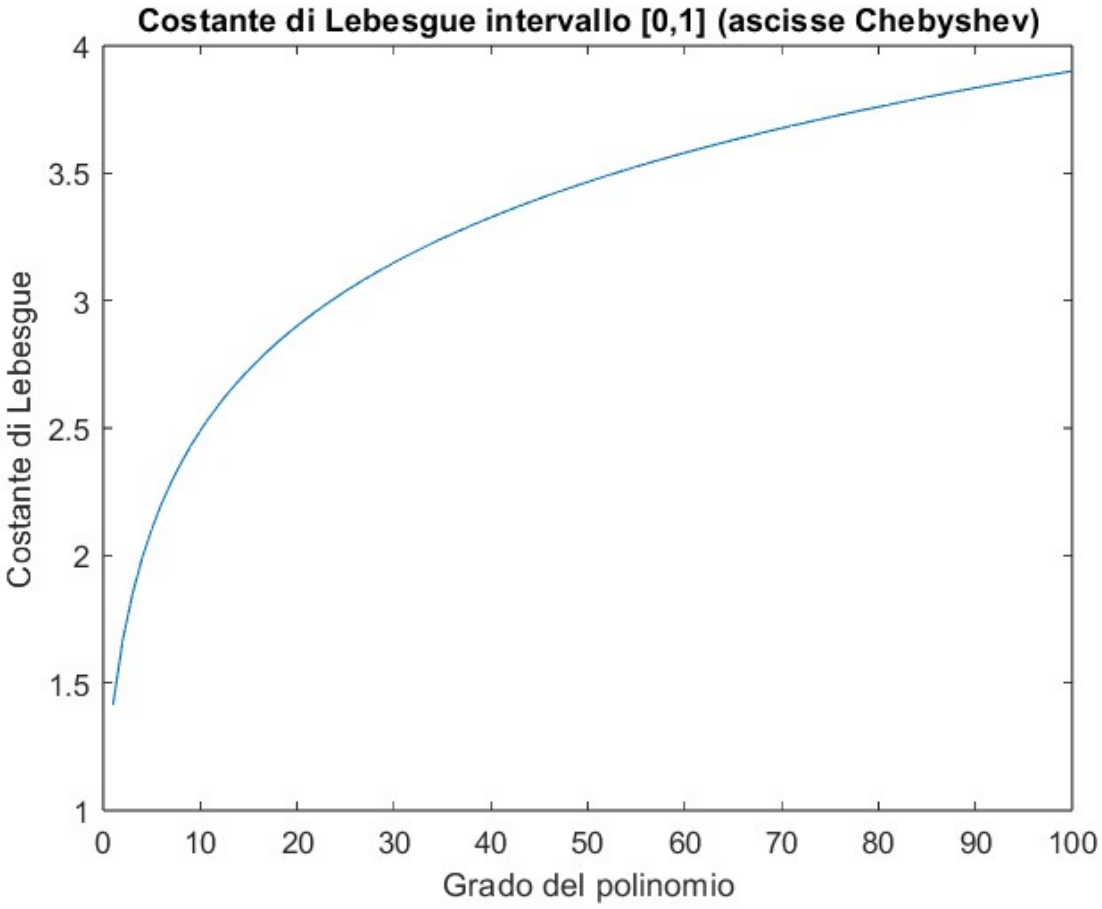
Svolgimento

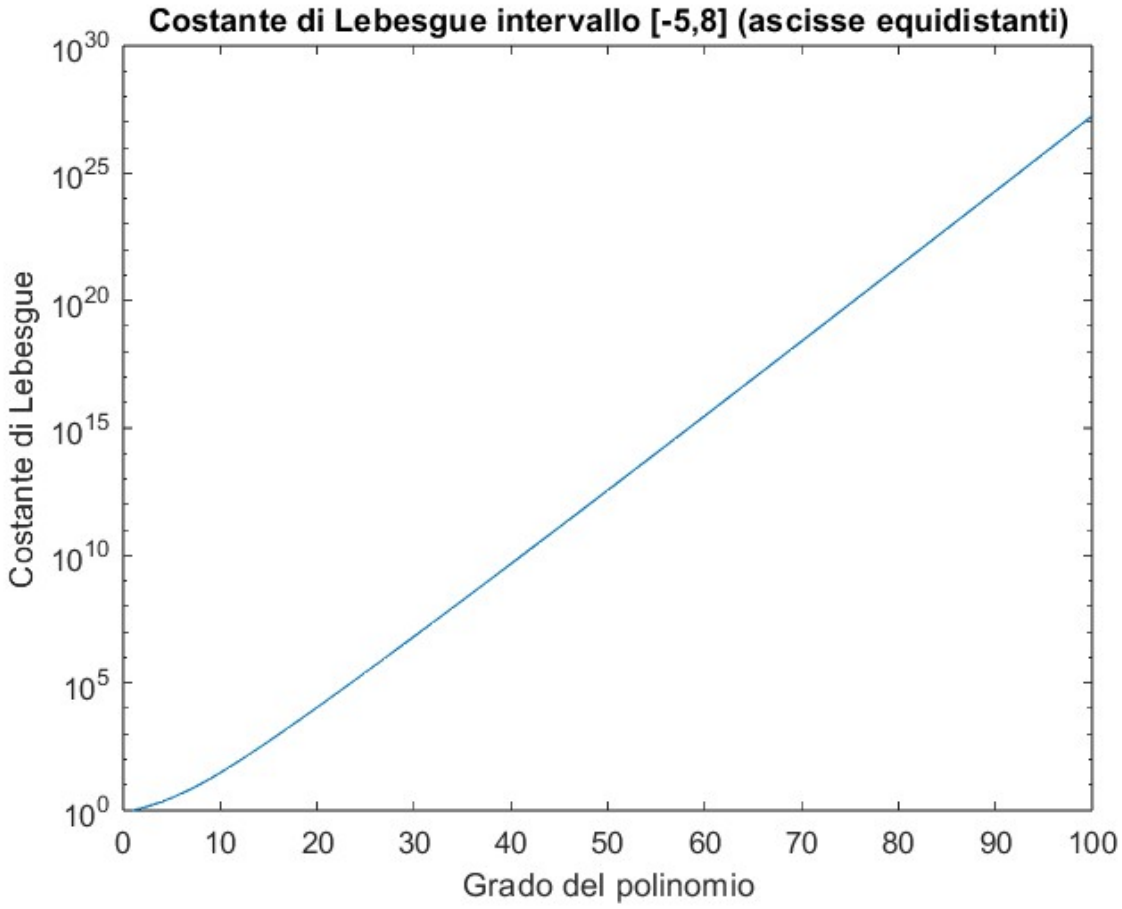
```
    function ll = lebesgue(a, b, nn, type)
% function ll = lebesgue(a, b, nn, type)
% Input: a,b = inizio e fine intervallo, nn = vettore con specificato il grado
    dei polinomi,
% type = specifica che tipo di ascisse di interpolazione usare
% se 0 utilizza le ascisse equispaziate nell'intervallo [a,b] altrimenti se
% 1 utilizza le ascisse di Chebyshev.
% Output: ll = vettore delle costanti di Lebesgue per ogni grado specificato
   in input
if nargin < 4
        error('parametri in ingresso insufficienti');
    elseif type == 0 % Ascisse equidistanti
        x = linspace(a, b, nn+1);
    elseif type == 1 % Ascisse di Chebyshev
        x =  chebyshev_es22(a, b, nn);
    else
        error('Tipo non valido. Usare 0 per ascisse equidistanti e 1 per
            ascisse di Chebyshev.');
    end

    n = length(x);
    % Calcolo della costante di Lebesgue
    xq = linspace(a, b, 10001);
    p = zeros(1, 10001);
    for i = 1:n
        v = [1:i-1, i+1:n];
        L = ones(size(xq));
        for j = v
            L = L .* (xq - x(j)) / (x(i) - x(j));
        end
```

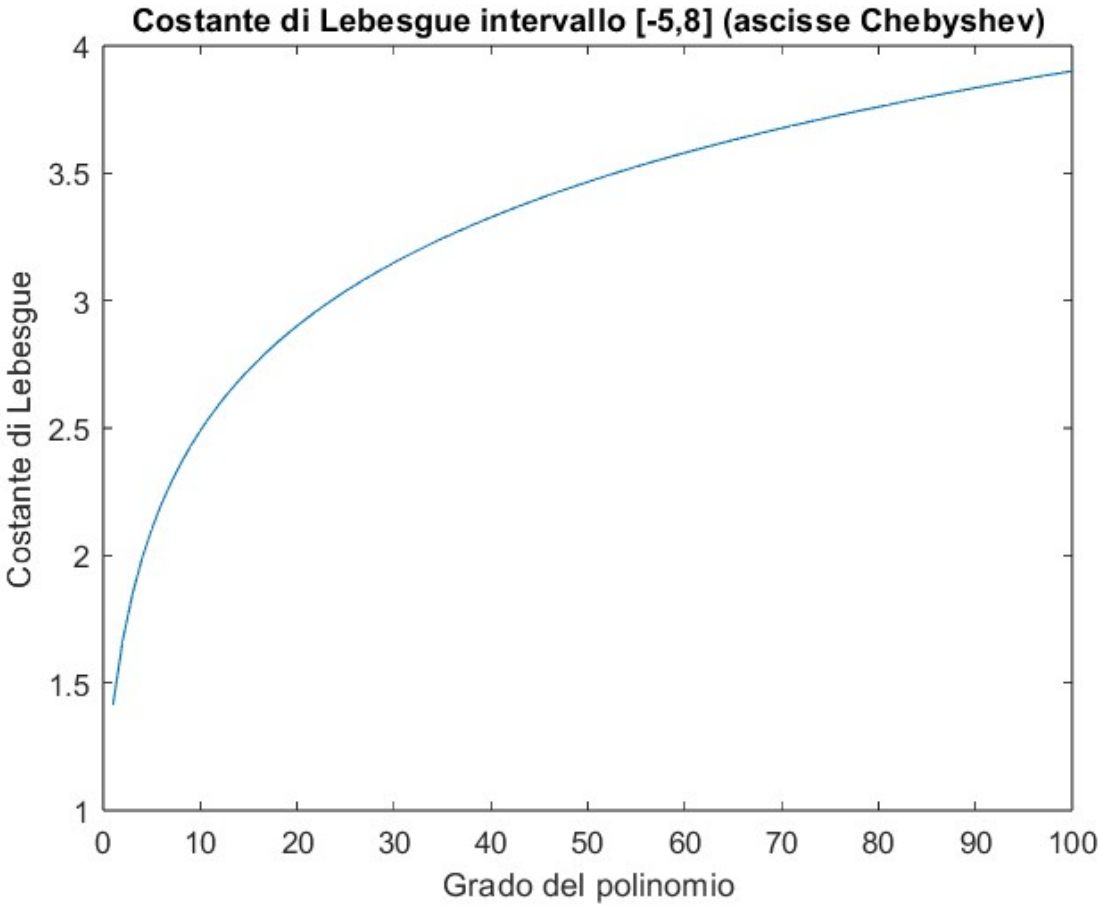
```
        p = p + abs(L);
    end
    ll = max(p);
return
```

Grafici




Costante di Lebesgue intervallo [0,1] (ascisse Chebyshev)
Costante di Lebesgue
Grado del polinomio
4
3.5
3
2.5
2
1.5
1
0
10
20
30
40
50
60
70
80
90
100


Costante di Lebesgue intervallo [-5,8] (ascisse equidistanti)
Costante di Lebesgue
Grado del polinomio
10^30
10^25
10^20
10^15
10^10
10^5
10^0
0
10
20
30
40
50
60
70
80
90
100



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,

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

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

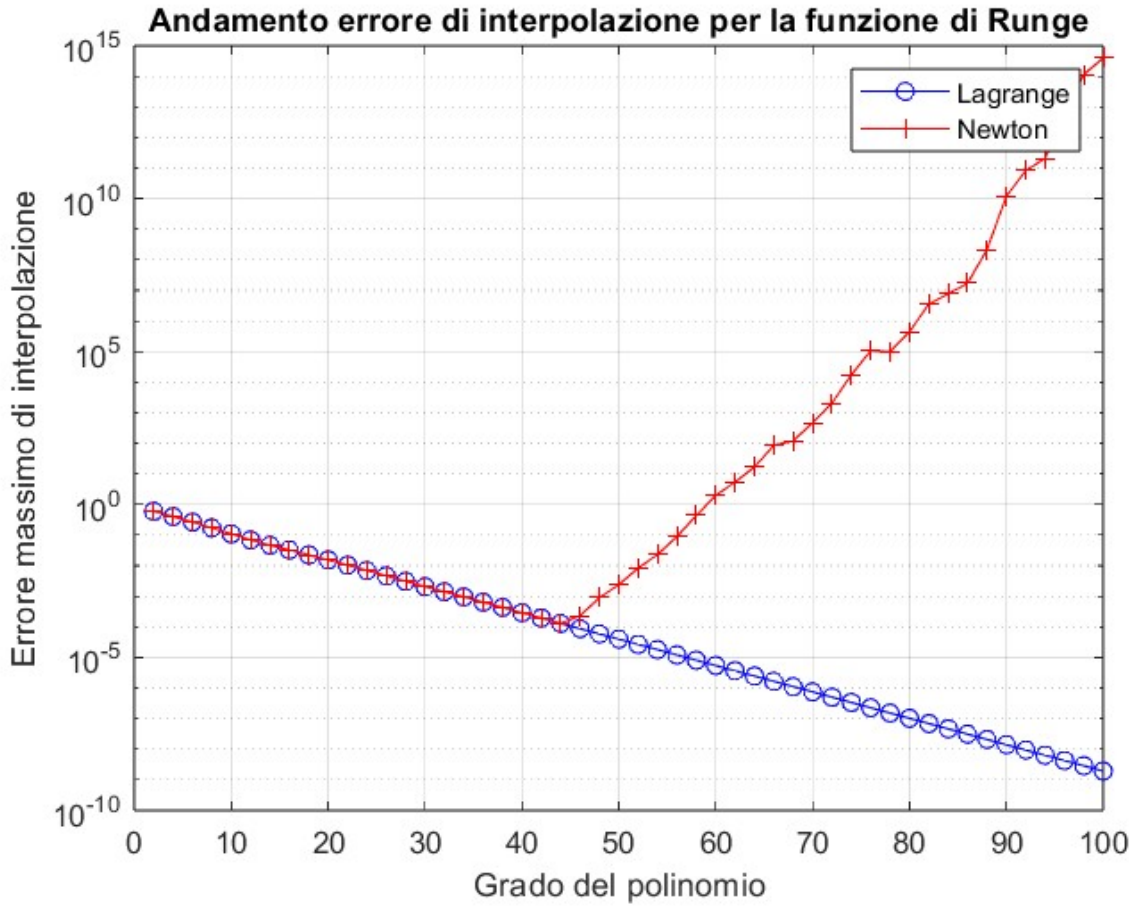
```
    % Definizione della funzione di Runge
f = @(x) 1 ./ (1 + x.^2);

% 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;


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

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

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
```

Grafico



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)**

Svolgimento

```
    function s = spline0(x, y, xq)
%
%   s = spline0(x, y, xq);
%
%   Calcola la spline cubica naturale interpolante una funzione.
%
%   Input:
%       x - ascisse di interpolazione;
%       y - valori della funzione valutata nelle ascisse di interpolazione;
%       xq - vettore dei punti in cui calcolare la spline.
%
```

```
%   Output:
%       s - valore della spline cubica naturale calcolata 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; % cosi gli indici delle ascisse vanno da 1 a 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); % vettore contenente 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));
    s = 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));
            s(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
```

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

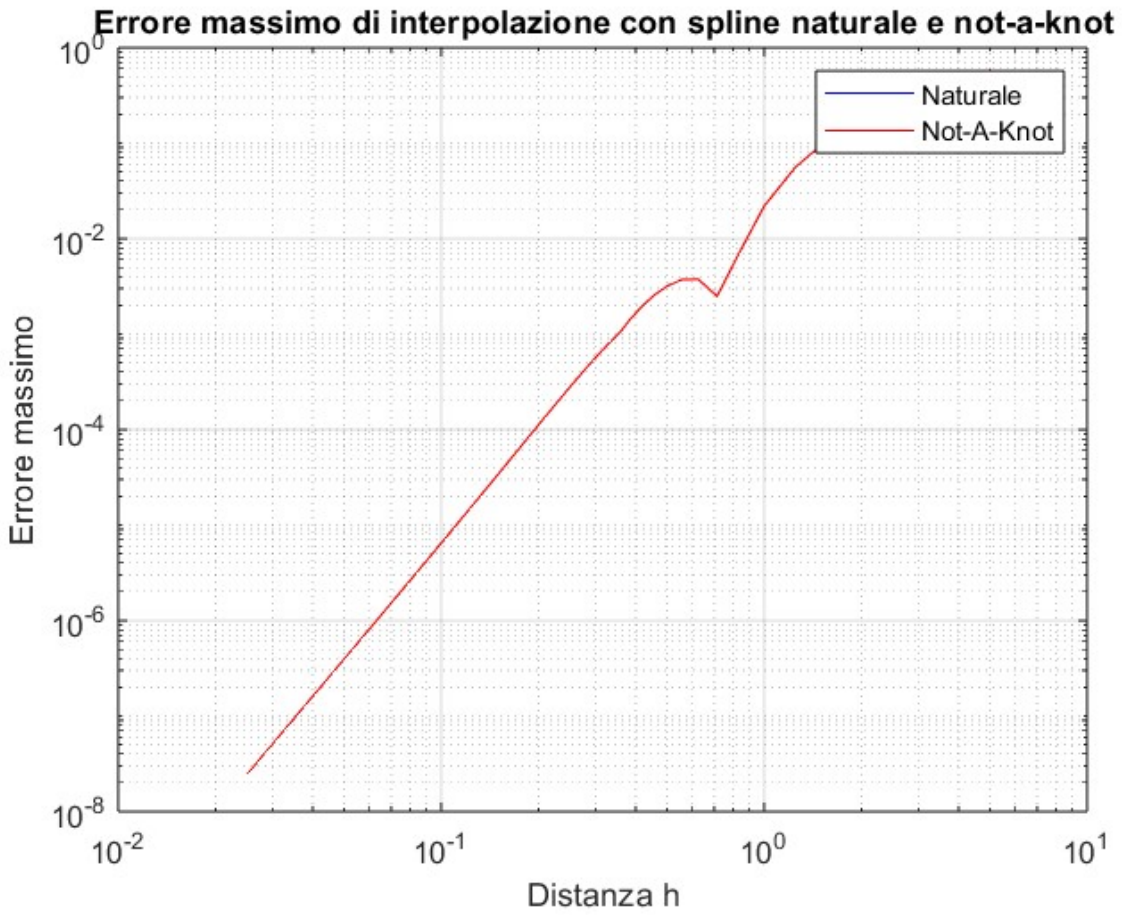
$$\Delta = \{x_i = -10 + i\frac{20}{n}, i = 0, \ldots, n\}, \quad n = 4 : 4 : 800,$$

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

Svolgimento

```
    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;
```

Grafico



Osserviamo che la curva di errore dei due tipi di spline cubica sembra abbia lo stesso tipo di crescita all'inizio, per poi avere un'andamento di crescita diverso

Esercizio 26

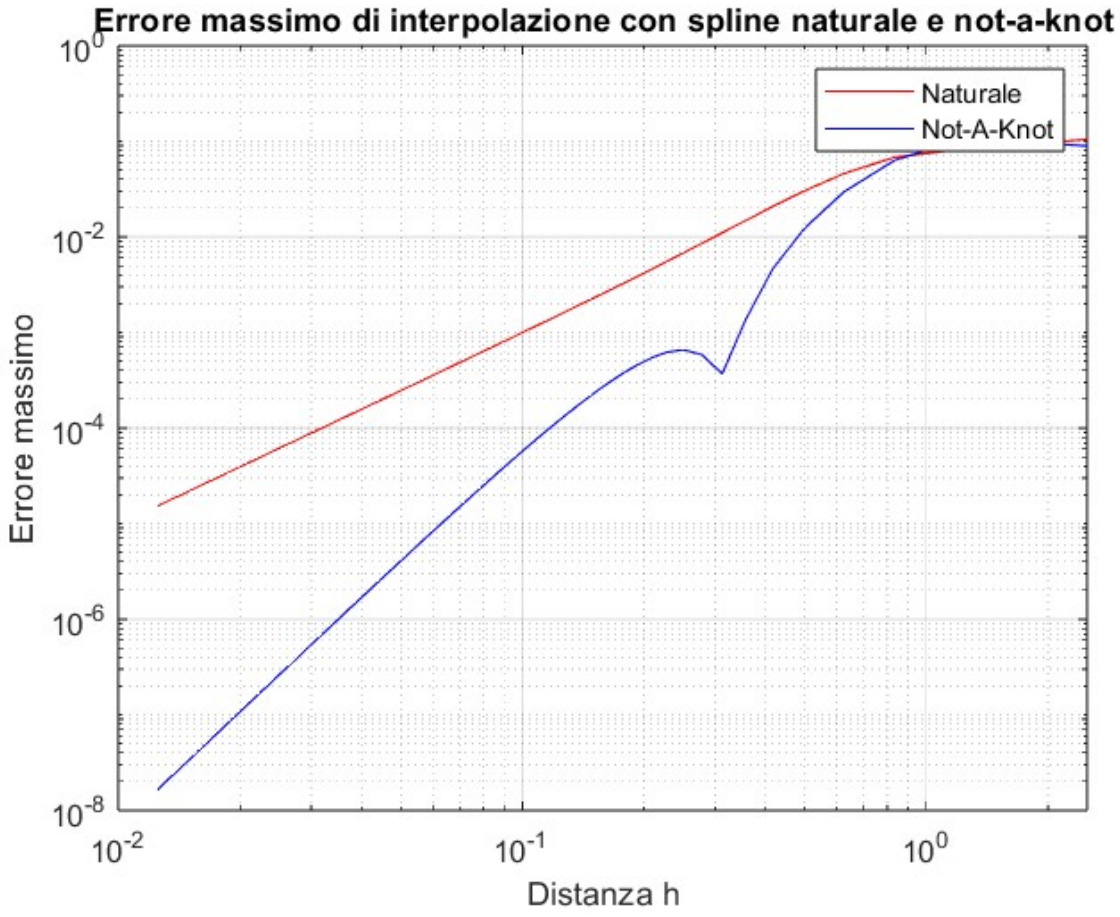
Graficare, utilizzando il formato loglog, l'errore di approssimazione utilizzando le *spline* interpolanti naturale e *not-a-knot* per approssimare la funzione di Runge sull'intervallo $[0, 10]$, utilizzando una partizione uniforme

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

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.

Svolgimento

```
    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
    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;
```



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;
```

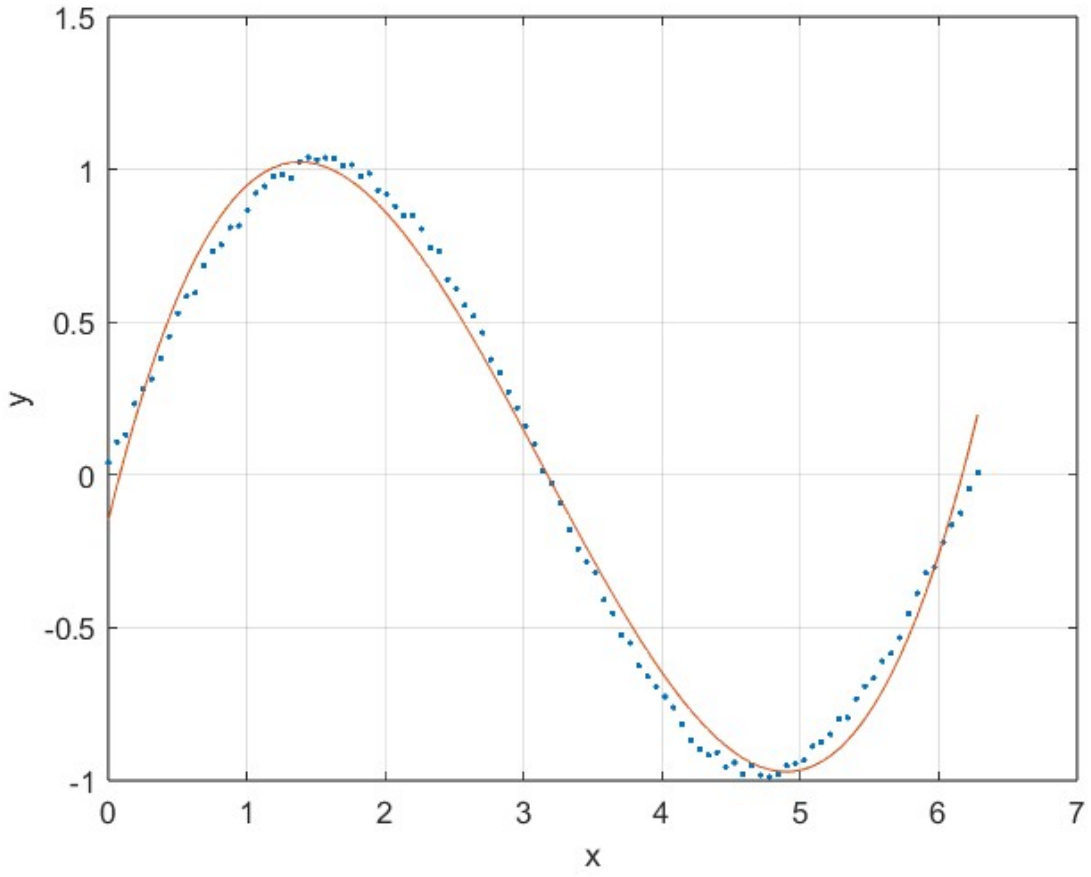
Graficare convenientemente i risultati ottenuti.

Svolgimento

```
    rng(0);
xi = linspace(0,2*pi,101);
yi = sin(xi) + rand(size(xi)) * .05;
M = zeros(101,4); %matrice 101 righe(date dalle 101 ascisse) e 4 colonne (
    grado 3)
for i = 1:101
    M(i,:) = xi(i) .^ (0:3);
end
```

```
x = miaqr(M, yi(:)); %coefficienti polinomio
y = polyval(flip(x'), xi);
plot(xi, yi, '.', xi, y);
xlabel('x');
ylabel('y');
grid on;
```

Grafico



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

Esecuzione

```
  function kg = calcolaCoefficienti(n)
%
```

```
% w = newtoncotes_weights(n)
%
% Calcola i pesi della formula di quadratura di Newton-Cotes di grado n
% INPUT:
%   n     - grado della formula di Newton-Cotes
% OUTPUT:
%   w     - vettore dei pesi dei coefficienti
%

%controllo
if ~isnumeric(n) || n <= 0 || mod(n, 1) ~= 0
   error("Il grado n deve essere un numero intero positivo");
end

%inizializzazioni
x = 0:n;
kg = zeros(1, n+1);

%calcolo dei pesi usando polinomio Lagrange
for i = 1:n+1
   L = 1;
   for j = 1:n+1
       if j ~= i
           L = conv(L, [1,-x(j)]) / (x(i)-x(j));
       end
   end
   kg(i) = polyval(polyint(L), n);
end
end
```

Risultati

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

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.

```
   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

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
```

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.

Svolgimento

```
    % 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\tIntegrale approssimato\t\tErrore stimato\t\tErrore reale\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
```

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 $n = 12$