UNIVERSITÀ
DEGLI STUDI
FIRENZE

Anno accademico 2022/2023

Elaborato Calcolo Numerico

Autore: Giulio Morandini
Matricola: 7030209

Esercizio 1:

Verificare che:

$$-\frac{1}{4}f(x-h)+\frac{5}{6}f(x)+\frac{3}{2}f(x+h)-\frac{1}{2}f(x+2h)+\frac{1}{12}f(x+3h)=hf'(x)+O(h^5).$$

Soluzione:

Per farlo è necessario procurarsi gli sviluppi di taylor al quarto ordine delle funzioni presenti, centrati in x:

$$f(x+2h)=f(x)+2hf'(x)+2h^2f''(x)+\frac{4}{3}h^3f'''(x)+\frac{2}{3}h^4f^{\mathrm{iv}}(x)+O(h^5),$$

$$f(x+3h)=f(x)+3hf'(x)+\frac{9}{2}h^2f''(x)+\frac{9}{2}h^3f'''(x)+\frac{27}{8}h^4f^{\mathrm{iv}}(x)+O(h^5),$$

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

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

Sostituendo al primo membro dell'equazione gli sviluppi ricavati qui sopra otteniamo:

$$-\frac{1}{4}f(x)+\frac{h}{4}f'(x)-\frac{1}{8}h^2f''(x)+\frac{1}{24}h^3f'''(x)-\frac{1}{96}h^4f''''(x)+O(h^5)$$

$$\frac{3}{2}f(x)+\frac{3}{2}hf'(x)+\frac{3}{4}h^2f''(x)+\frac{1}{4}h^3f'''(x)+\frac{1}{16}h^4f''''(x)+O(h^5)$$

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

$$\frac{1}{12}f(x)+\frac{1}{4}hf'(x)+\frac{3}{8}h^2f''(x)+\frac{3}{8}h^3f'''(x)+\frac{9}{32}h^4f''''(x)+O(h^5)$$

Considerando anche $-\frac{5}{6}f(x)$. si ottiene sommando membro a membro:

$$-\tfrac{5}{6}f(x)-\tfrac{1}{4}f(x)+\tfrac{3}{2}f(x)-\tfrac{1}{2}f(x)+\tfrac{1}{12}f(x)=0$$

$$\frac{1}{4}hf'(x)+\frac{3}{2}hf'(x)-hf'(x)+\frac{1}{4}hf'(x)=hf'(x)$$

$$-\frac{1}{8}h^2f''(x)+\frac{3}{4}h^2f''(x)-h^2f''(x)+\frac{3}{8}h^2f''(x)=0$$

$$\frac{1}{24}h^3f'''(x)+\frac{1}{4}h^3f'''(x)-\frac{2}{3}h^3f'''(x)+\frac{3}{8}h^3f'''(x)=0$$

$$-\frac{1}{96}h^4f''''(x)+\frac{1}{16}h^4f''''(x)-\frac{1}{3}h^4f''''(x)+\frac{9}{32}h^4f''''(x)=0$$

Cioè:

$$hf'(x)+O(h^5)$$

Che è equivalente al termine sinistro dell'esercizio.

Esercizio 2:

Matlab utilizza la doppia precisione IEEE. Stabilire, pertanto, il nesso tra la variabile eps e la precisione di macchina di questa aritmetica.

Soluzione:

Sapendo che la precisione di macchina u (nel caso si usi il l'arrotondamento) nello standard IEEE si ottiene come:

$$u = \frac{1}{2} b^{1-m}$$

Dato che lo standard utilizza la rappresentazione per arrotondamento. Data la base binaria ($b=2$) e m = 53 cifre per la mantissa, si ha:

$$\frac{1}{2} 2^{1-53} = 2^{-1} 2^{1-53}$$

$$2^{-53} \approx 1{,}110223e-16$$

La funzione **eps** di MatLab, che contiene la precisione di macchina, viene invece calcolata per troncamento, quindi con la seguente formula:

$$u = b^{1-m}$$

Sostituendo:

$$2^{1-53} = 2^{-52}$$

$$2^{-52} \approx 2.220446e-16$$

La precisione di macchina è definita come il massimo errore relativo dovuto alla rappresentazione in aritmetica finita di un numero reale. Si può quindi vedere come la precisione di macchina calcolata per arrotondamento sia più precisa rispetto al calcolo per troncamento.

Esercizio 3:

Spiegare 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:

$$k = \frac{|x_1| + |x_2|}{|x_1 + x_2|}$$

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 · 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 · 10 − 6 .

che è un risultato diverso rispetto a d.

Esercizio 4:

Scrivere una function Matlab, radice(x) che, avendo in ingresso un numero x non negativo x^(1/6) calcoli utilizzando solo operazioni algebriche elementari, con la massima precisione possibile. Confrontare con il risultato fornito da x^(1/6) per 20 valori di x, equispaziati logaritmicamente nell'intervallo [1e-10,1e10], tabulando i risultati in modo che si possa vericare che si è ottenuta la massima precisione possibile.

Soluzione:

La function è stata ottenuta sfruttando il metodo di Newton per cercare le radici della funzione f(x) = x^n – x0, dove n=6, x0 è il numero di cui vogliamo conoscere la radice, infatti la la radice di tale equazione non sarà altro che radice sesta stessa. Sapendo ciò è bastato calcolare manualmente la derivata di tale funzione che è f0(x) = 6x per potermi ricavare direttamente il passo iterativo che sarà:

$$x_{i+1} = x_i - \frac{x_i^6 - x_0}{6\,x_i}$$

La massima precisione possibile è stata ottenuta utilizzando come tolleranza la precisione di macchina contenuta nella variabile eps di Matlab, infatti sarebbe inutile usare un valore di tolleranza inferiore in quanto sotto tale soglia non avremo alcuna garanzia che la tolleranza venga rispettata.

Codice:

```
function rad=radice(x)
%rad=radice(x)
%Input: x, numero positivo di cui si vuole conoscere la radice sesta
%Output: rad, radice sesta del numero in input
if (x==0)
rad=0;
return;
end
if x<0, error("x non puo' essere negativo"), end
x0=x;
for i=0:10000
rad=x-((x^6 -x0)/(6*x^5));%passo iterativo
if(abs(rad-x)<eps*(1+abs(x))) %controllo di aver raggiunto la massima precisione possibile
break;
end
x=rad;
end
return;
```

Tabella dei valori equispaziati logaritmicamente:

Si riportano i valori restituiti da radice(x) confrontandoli con quelli calcolati da x^(1/6) per valori di x ottenuti con logspace(-10,10,20):

x	radice(x)	x^(1/6)
1.000000e-10	2.154435e-02	2.154435e-02
1.128838e-09	3.226799e-02	3.226799e-02
1.274275e-08	4.832930e-02	4.832930e-02
1.438450e-07	7.238509e-02	7.238509e-02
1.623777e-06	1.084146e-01	1.084146e-01
1.832981e-05	1.623777e-01	1.623777e-01
2.069138e-04	2.432008e-01	2.432008e-01
2.335721e-03	3.642533e-01	3.642533e-01
2.069138e-02	5.455595e-01	5.455595e-01
2.976351e-01	8.171103e-01	8.171103e-01
3.359818e+00	1.223825e+00	1.223825e+00
3.792690e+01	1.832981e+00	1.832981e+00
4.281332e+02	2.745342e+00	2.745342e+00
4.832930e+03	4.111829e+00	4.111829e+00
5.455595e+04	6.158482e+00	6.158482e+00
6.158482e+05	9.223851e+00	9.223851e+00
6.951928e+06	1.381500e+01	1.381500e+01
7.847600e+07	2.069138e+01	2.069138e+01
8.858668e+08	3.099046e+01	3.099046e+01
10000000000	4.641589e+01	4.641589e+01

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). Per tutti i metodi, utilizzare come criterio di arresto C (espresso sul testo degli esercizi) essendo tol una opportuna tolleranza specicata in ingresso. Curare particolarmente la robustezza del codice.

Metodo di Newton:

```
function [x , it,count] = newton (f , f1 , x0 , maxIt , tol )
% Determina uno zero della funzione f applicando il metodo di Newton .
% Input :
% f - Funzione di cui voglio trovare la radice,  f1 - derivata prima della funzione f, x0 - approssimazione iniziale della radice
% maxIt - numero massimo di iterazioni [DEFAULT 100], tol - tolleranza [DEFAULT 10^-3]
% Output :  x - approssimazione della radice  it - numero di iterazioni fatte  count - numero di valutazioni funzionali fatte
if maxIt <0
    maxIt=100;
end
if tol<0
    tol=1e-3;
end
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 ;
    if abs (x - x0 ) <= tol *(1+ abs ( x )) break
    end
end
it = i ;
if ( abs (x - x0 ) > tol *(1+ abs ( x )))
    disp (" il metodo non converge ") end
end
```

Metodo delle secanti:

```
function [x,it,count] = secanti(f,x0,x1,maxIt,tolx)
% [x,it] = 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 - 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 fatte
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 del precedente esercizio per determinare una approssimazione della radice della funzione *f(x)=x-cos(x)* per tol = 10^-3; 10^-6; 10^-9; 10^-12, partendo da x0 = 0 (e x1 = 0.1 per il metodo delle secanti). Tabulare i risultati, in modo da confrontare il costo computazionale di ciascun metodo.

Soluzione:

tol	Newton	Iter.	Val.funz.	Secanti	Iter.	Val.funz.
10^{-3}	7.390851333852840e-01	4	8	7.390849597899151e-01	4	5
10^{-6}	7.390851332151607e-01	5	10	7.390851332185636e-01	5	6
10^{-9}	7.390851332151607e-01	5	10	7.390851332151607e-01	6	7
10^{-12}	7.390851332151607e-01	6	12	7.390851332151607e-01	6	8

Dai risultati ottenuti si conclude che il metodo delle secanti è quello dei due che esegue il minor numero di valutazioni funzionali tra i due. Per quanto riguarda le iterazioni, il loro numero resta stabile e simile per entrambi gli algoritmi.

Esercizio 7:

Utilizzare le function del precedente Esercizio 5 per determinare una approssimazione della radice della funzione *f(x)=[x-cos(x)]^5* per tol = 10^-3; 10^-6; 10^-9; 10^-12, partendo da x0 = 0 (e x1 = 0:1 per il metodo delle secanti). Confrontare con i risultati ottenuti utilizzando il metodo di Newton modicato. Tabulare i risultati, in modo da confrontare il costo computazionale e l'accuratezza di ciascun metodo. Commentare i risultati ottenuti.

Soluzione:

tol	Newton	Iter.	Secanti	Iter.	Newton modificato	Iter.
10^{-3}	7.326406977511085e-01	20	7.490453083854652e-01	22	7.39085133385279-01	3
10^{-6}	7.390787623210328e-01	51	7.390945994146900e-01	67	7.39085133215282e-01	4
10^{-9}	7.390851269057438e-01	82	7.390851421873479e-01	112	7.39085133215282e-01	4
10^{-12}	7.390851332089121e-01	113	7.390851332250874e-01	156	7.39085133215282e-01	5

Tutti i metodi convergono alla soluzione per tutte le tolleranze. In questo caso il numero di iterazioni non si mantiene stbile ma aumenta al variare della tolleranza.
Confrontando i risultati con il metodo di Newton modificato notiamo una drastica diminuzione del numero di iterazioni mentre i valori non si discostano da quelli ottenuti col metodo di newton.
Si può notare immediatamente come i metodi di Newton e Secanti siano estremamente più lenti del metodo di newton modificato, questo perché la funzione data ha molteplicità maggiore di uno, questo rende la convergenza del metodo di Newton lineare; se però utilizziamo l'accelerazione di Aitken per il metodo di Newton possiamo ripristinare la convergenza quadratica. Si può notare anche come la radice raggiunta dal metodo di newton modificato sia più precisa già dalle prime iterazioni.

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'ecienza della function, e validarla su due esempi non banali, generati casualmente, di cui sia nota la soluzione.

Codice:

```
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 è 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 è 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 non è non 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
```

La complessità di tale script è di circa (2/3)n^3 per la funzione di scomposizione LU con pivoting parziale e di circa n^2 per risolvere il sistema lineare precedentemente scomposto.

Validazione:

```
>> A = randi([-6,6],6)

A =

     4    -3     6     4     2     3
     5     1     0     6     3    -6
    -5     6     4     2     3    -3
     5     6    -5    -6    -1    -6
     2    -4    -1     5     2    -5
    -5     6     5     6    -4     4

>> x = randi([-6,6],6,1)

x =

     3
    -2
     6
    -6
    -1
    -2

>> b = A*x

b =

    22
   -14
   -12
    22
   -14
   -37

>> mialu(A,b)

ans =

    3.0000
   -2.0000
    6.0000
   -6.0000
   -1.0000
   -2.0000
```

```
>> A = randi([-15,15],5)

A =

     8     5     5    -5     0
     9     6   -10     3     6
   -10     8   -12    -9    12
     0    -7     0     8    14
    -2     6    14    -8     1

>> x = randi([-15,15],5,1)

x =

   -11
   -11
    -8
    11
    -8

>> b=A*x

b =

  -238
  -100
   -77
    53
  -252

>> mialu(A,b)

ans =

   -11.0000
   -11.0000
    -8.0000
    11.0000
    -8.0000
```

In questi casi la soluzione è corretta (verificabile controllando che Ax - b = 0).

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 due esempi non banali, generati casualmente, di cui sia nota la soluzione.

Codice:

```
function  x = mialdl(A,b)
%Calcola la soluzione del sistema lineare Ax=b utilizzando la fattorizzazione LDLT
% Input:: A = matrice sdp da fattorizzare, b = vettore termini noti
% Output: x = soluzione del sistema
[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
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
```

La complessità di tale script è di circa (1/3)n^3 per la funzione di scomposizione LDLt e sempre di circa n^2

Validazione:

```
>> A = randi([-8,8], 4)

A =

    -1    -5     4     3
    -2     0     4    -6
     5    -1    -4    -6
     5     2     3     0

>> d = randi([5,30], 4,1)

d =

    29
    13
    20
    10

>> A = tril(A,-1) + triu (A',1) + diag (d)

A =

    29    -2     5     5
    -2    13    -1     2
     5    -1    20     3
     5     2     3    10

>> x = randi ([-8,8], 4, 1)

x =

     4
    -4
     0
     3

>> b = A*x

b =

   139
   -54
    33
    42

>> mialdl(A,b)

ans =

    4.0000
   -4.0000
    0.0000
    3.0000
```

```
>> A=randi([-8,8],5)

A =

     5    -7    -6    -6     3
     7    -4     8    -1    -8
    -6     1     8     7     6
     7     8     0     5     7
     2     8     5     8     3

>> d=randi([6,21],5,1)

d =

    18
    17
    12
    16
     8

>> A=tril(A,-1)+triu(A',1)+diag(d)

A =

    18     7    -6     7     2
     7    17     1     8     8
    -6     1    12     0     5
     7     8     0    16     8
     2     8     5     8     8

>> x=randi([-10,21],5,1)

x =

    12
    -9
    -2
    -9
    -7

>> b=A*x

b =

    88
  -199
  -140
  -188
  -186

>> mialdl(A,b)

ans =

   12.0000
   -9.0000
   -2.0000
   -9.0000
   -7.0000
```

Esercizio 10:

Scrivere una function Matlab function [x,nr] = miaqr(A,b) che, data in ingresso la matrice A m x n, 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'ecienza della function. Validare la function miaqr su due esempi non banali, generati casualmente, confrontando la soluzione ottenuta con quella calcolata con l'operatore Matlab \.

Codice:

```
function [x,nr] = miaqr(A,b)
%
% [x,nr] = miaqr(A,b)
%
% La funzione calcola la fattorizzazione QR del sistema lineare
% sovraddimensionato e restituisce, oltre alla fattorizzazione, la norma
% euclideiana del vettore residuo
%
% Input:
% A = matrice da fattorizzare
% b = vettore dei termini noti
%
% Output:
% x = soluzione del sistema Ax=b
% nr = norma del vettore residuo
%
[m,n] = size(A);
if(n>m)
error('Errore: il sistema in input non è 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
```

Validazione:

```
>> A = randi([-15,15], 8 , 5)

A =

    -2   -12     0    11     4
     5     0     6    -8    -1
     6    14    12    10    -5
     8    -5    14    -8    10
    -7     3     1    13     3
     6    -9   -11    -5     2
     5     8   -11    -9    13
   -10    -8    -8    -8    -7

>> b = randi([-15,15], 8, 1)

b =

     8
     8
    -4
     2
   -13
   -14
     1
     9

>> [x,nr] = miaqr(A,b)

x =

   -0.6004
   -0.1458
    0.4611
   -0.5437
   -0.0071


nr =

   19.5503

>> x = A\b

x =

   -0.6004
   -0.1458
    0.4611
   -0.5437
   -0.0071
```

```
>> A = randi([-20,18],10,6)

A =

    12    -1    -1    17    -7    -9
     7    -3    17     1   -13     9
    -8     5    -7   -15   -11     9
    17     7     2   -15     4    -6
   -19     9   -12   -10    -2     2
    -3   -10     9    12    -7   -18
    -6     6   -11   -11    12   -18
     9     5    -1    11     2     0
    11   -14     7   -11     1    10
   -13   -16    14    16    15    16

>> b=randi([-20,18],10,1)

b =

   -15
     2
    -2
   -20
    -7
   -14
    10
    -8
     0
   -14

>> [x,nr]=miaqr(A,b)

x =

   -0.0906
   -0.1659
   -0.3541
   -0.1969
   -0.0383
    0.1093


nr =

   32.1823

>> x=A\b

x =

   -0.0906
   -0.1659
   -0.3541
   -0.1969
   -0.0383
    0.1093
```

Esercizio 11:

Data la function Matlab function [A1,A2,b1,b2] = linsis1(n,simme), che crea sistemi lineari casuali di dimensione n con soluzione nota, risolvere, utilizzando la function mialu, i sistemi lineari generati da [A1,A2,b1,b2]=linsis(5). Commentare l'accuratezza dei risultati ottenuti, dandone spiegazione esaustiva.

Soluzione:

La function mialu calcola la soluzione per il sistema lineare A1x = b1 dato da linsis(5) con un'ottima precisione. La soluzione calcolata per il sistema lineare A2x = b2 con linsis(5) si discosta molto da quella aspettata.
Il risultato ottenuto con la funzione linsis nel caso della matrice A2 è errato, mentre quello ottenuto per la matrice A1 è corretto. Questo risultato è dovuto al differente condizionamento delle 2 matrici per come è stata definita la funzione. A riprova di questo fatto con la funzione cond(A1), che misura il condizionamento della matrice, si hanno i valori 2.5 per A1 mentre 9.99*10^9 per A2
Si riportano i risultati ottenuti dalle 2 chiamate:

```
>> mialu(A1,b1)

ans =

   1.000000000000000
   1.000000000000000
   1.000000000000000
   1.000000000000000
   1.000000000000000
```

```
>> mialu(A2,b2)

ans =

   0.999999647657477
   1.000000446226050
   1.000000098875194
   1.000000207059384
   1.000000011600807
```

Esercizio 12:

Risolvere, utilizzando la function mialdlt, i sistemi lineari generati da [A1,A2,b1,b2]=linsis(5,1). Commentare l'accuratezza dei risultati ottenuti, dandone spiegazione esaustiva.

Soluzione:

```
>> mialdl(A1,b1)

ans =

   1.000000000000000
   0.999999999999999
   1.000000000000000
   1.000000000000000
   1.000000000000000

>> mialdl(A2,b2)

ans =

   1.000000052293492
   1.000000058138758
   1.000000008150719
   1.000000058625536
   1.000000040588329
```

La function restituisce una soluzione corretta per il sistema lineare A1x = b1 generato con linsis(5,1). Anche in questo caso la soluzione calcolata per il sistema lineare A2x = b2 con linsis(5,5) si discosta molto da quella aspettata.
Il motivo è che la seconda matrice ottenuta con linsis(5,5) è mal condizionata. In una prova, il numero di condizionamento della seconda matrice generata, ottenuto eseguendo cond(A2) è pari a 9.999997360722641e+09, che appunto è molto alto. Per contro, la prima matrice ottenuta tramite linsis(5,5) risulta ben condizionata, avendo un numero di condizionamento pari a 2.500000000000000e+00.

Esercizio 13:

Utilizzare la function miaqr per risolvere, nel senso dei minimi quadrati, i sistemi lineari sovradeterminati A x = b, (D*A)x = (D*b), (D1*A)x = (D1*b),deniti dai seguenti dati: (espressi sul testo). Calcolare le corrispondenti soluzioni e residui, e commentare i risultati ottenuti.

Soluzione:

- **Ax = b:**

```
x =

   3.00000000000001
                  5.8
  -2.50000000000001


nr =

   1.26491106406736
```

I valori della soluzione sembrano essere buoni anche se per una minima alterazione dovuta alla rappresentazione dei numeri reali non di macchina.

- **(D*A)x = (D*b):**

```
x =

  -0.602569986232244
     4.7016980266177
    1.75837540156039


nr =

    3.73515111234241
```

Il risultato è diverso se confrontato con i risultati ottenuti con l'operatore \. Inoltre in questo caso la norma euclidea è diversa da quella precedente.

- **(D1*A)x = (D1*b):**

```
x =

   3.00000000000002
   5.80000000000001
  -2.50000000000002


nr =

   3.97383530631843
```

In questo caso i valori si discostano minimamente dai risultati del primo sistema lineare, invece la norma euclidea è poco più che triplicata.

Esercizio 14:

Scrivere una function Matlab [x,nit] = newton(fun,jacobian,x0,tol,maxit) che implementi efficientemente il metodo di Newton per risolvere sistemi di equazioni non lineari. Curare particolarmente il criterio di arresto, che deve essere analogo a quello usato nel caso scalare. La seconda variabile, se specicata, ritorna il numero di iterazioni eseguite. Prevedere opportuni valori di default per gli ultimi due parametri di ingresso.

Codice:

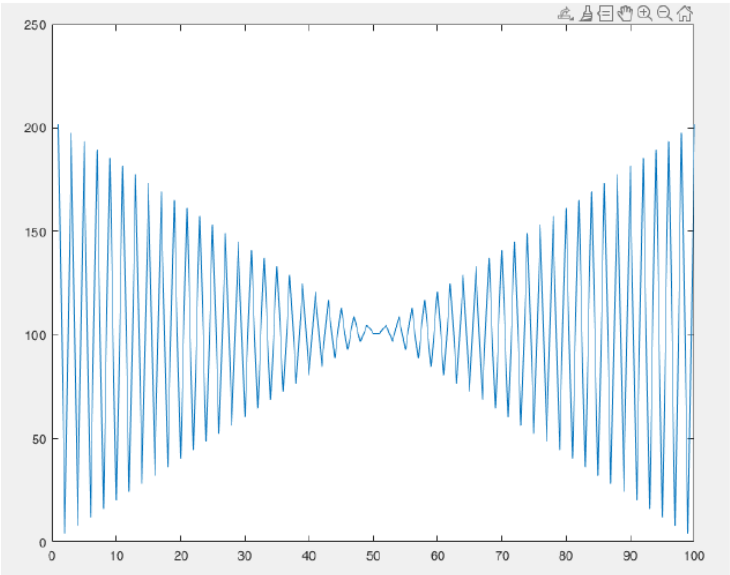
```
function [x,nit] = newtonBis(fun,jacobian,x0,tol,maxit)
% Questo metodo risolve sistemi non lineare di equazioni attraverso l'uso del metodo di Newton. Può restituire inoltre il numero di iterazioni eseguite
% Input: fun = vettore delle funzioni, jacobian = matrice jacobiana di fun,  x0 = vettore delle approssimazioni iniziali, tol = tolleranza specificata
% maxit = quante iterazioni il metodo potrà fare al massimo, se non specificata maxit = 1000
% Output: x = vettore delle soluzioni, nit = numero di iterazioni svolte
if(nargin == 3)
tol = eps;
end
if(nargin == 4)
maxit = 1000;
end
if (tol<0)
error('Errore: La tolleranza in input non può essere minore di 0.');
end
if (maxit<=0)
error('Errore: Il massimo numero di iterazioni non può essere minore di 0.');
end
x = x0;
for nit=1:maxit
x0 = x;
fx0 = feval(fun,x0);
fj = feval(jacobian,x0);
x = x0-(fj\fx0);
if (norm(x-x0)<=tol*(1+norm(x0)))
disp('Tolleranza desiderata raggiunta.');
break
end
end
if not(norm(x-x0)<=tol*(1+norm(x0)))
disp('Il metodo non è convergente.');
end
end
```

Esercizio 15

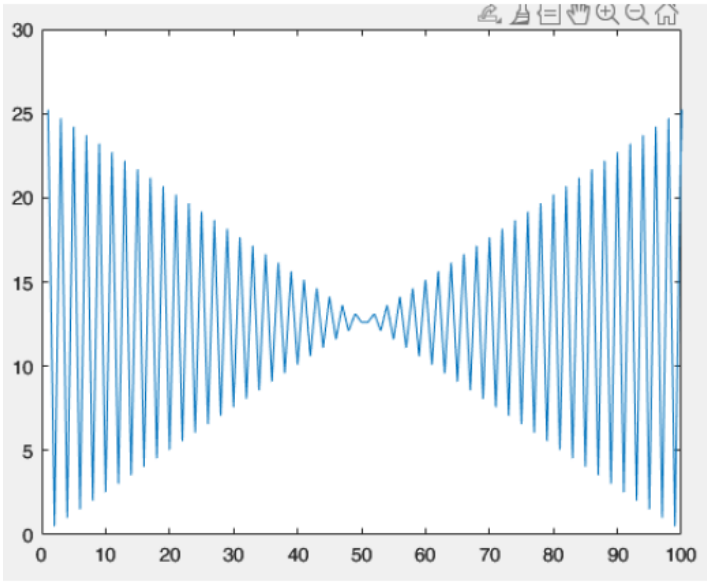
Usare la function del precedente esercizio per risolvere, a partire dal vettore iniziale nullo, il sistema nonlineare derivante dalla determinazione del punto stazionario della funzione descritto nel testo, utilizzando tolleranze tol = 1e-3, 1e-8, 1e-13. Gracare la soluzione e tabulare in modo conveniente i risultati ottenuti.

Soluzione:

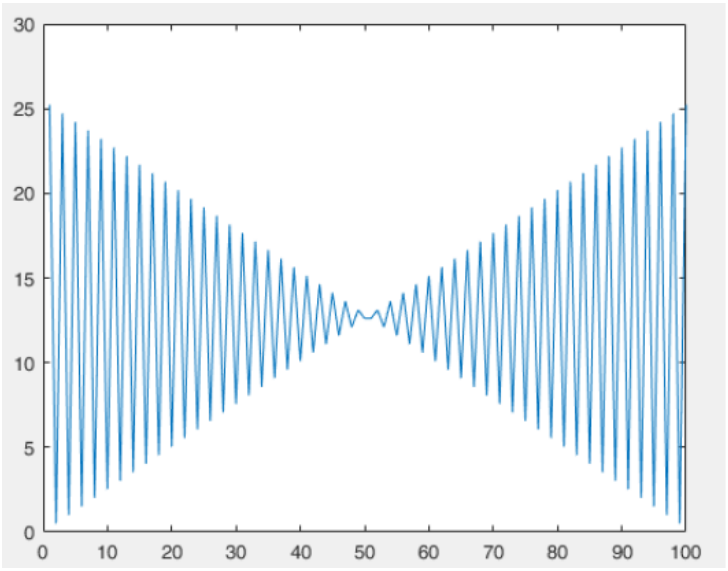
Tol = 1e-03 :



Tol = 1e-08 :



Tol = 1e-13 :



Tol: 1e-3

Indice	Valore
1	201.612479893568
2	4.03224959787136
3	197.580230295697
4	8.06449919574276
5	193.547980697825
6	12.0967487936144
7	189.515731099954
8	16.1289983914861
9	185.483481502082
10	20.161247989358
11	181.45123190421
12	24.1934975872299
13	177.418982306338
14	28.2257471851017
15	173.386732708466
16	32.2579967829736
17	169.354483110595
18	36.2902463808453
19	165.322233512723
20	40.3224959787171
21	161.289983914851
22	44.3547455765887
23	157.25773431698
24	48.3869951744603
25	153.225484719108
26	52.4192447723319
27	149.193235121236
28	56.4514943702037
29	145.160985523364
30	60.4837439680755
31	141.128735925493
32	64.5159935659472
33	137.096486327621
34	68.548243163819
35	133.064236729749
36	72.5804927616906
37	129.031987131878
38	76.6127423595621
39	124.999737534006
40	80.6449919574337
41	120.967487936135
42	84.6772415553052
43	116.935238338263
44	88.7094911531767
45	112.902988740392
46	92.7417407510481
47	108.87073914252
48	96.7739903489194
49	104.838489544649
50	100.806239946791

1e-8

Indice	Valore
1	25.2244874301087
2	0.504489748602265
3	24.7199976815064
4	1.00897949720452
5	24.2155079329042
6	1.51346924580675
7	23.7110181843019
8	2.01795899440899
9	23.2065284356997
10	2.52244874301122
11	22.7020386870975
12	3.02693849161344
13	22.1975489384953
14	3.53142824021566
15	21.6930591898931
16	4.03591798881787
17	21.1885694412908
18	4.54040773742008
19	20.6840796926886
20	5.04489748602228
21	20.1795899440864
22	5.5493872346245
23	19.6751001954842
24	6.05387698322671
25	19.170610446882
26	6.55836673182891
27	18.6661206982798
28	7.06285648043109
29	18.1616309496776
30	7.56734622903327
31	17.6571412010755
32	8.07183597763545
33	17.1526514524733
34	8.57632572623763
35	16.6481617038711
36	9.0808154748398
37	16.1436719552689
38	9.58530522344197
39	15.6391822066668
40	10.0897949720441
41	15.1346924580646
42	10.5942847206463
43	14.6302027094624
44	11.0987744692485
45	14.1257129608603
46	11.6032642178506
47	13.6212232122581
48	12.1077539664528
49	13.1167334636559
50	12.612243715055

1e-13

Indice	Valore
1	25.2244799412148
2	0.504489598824269
3	24.7199903423905
4	1.00897919764855
5	24.2155007435663
6	1.51346879647284
7	23.711011144742
8	2.01795839529715
9	23.2065215459177
10	2.52244799412145
11	22.7020319470934
12	3.02693759294573
13	22.1975423482691
14	3.53142719177003
15	21.6930527494448
16	4.03591679059432
17	21.1885631506205
18	4.54040638941861
19	20.6840735517962
20	5.04489598824292
21	20.1795839529719
22	5.54938558706722
23	19.6750943541476
24	6.05387518589152
25	19.1706047553233
26	6.55836478471582
27	18.666115156499
28	7.06285438354013
29	18.1616255576747
30	7.56734398236444
31	17.6571359588504
32	8.07183358118875
33	17.152646360026
34	8.57632318001305
35	16.6481567612017
36	9.08081277883736
37	16.1436671623774
38	9.58530237766168
39	15.6391775635531
40	10.089791976486
41	15.1346879647288
42	10.5942815753103
43	14.6301983659045
44	11.0987711741346
45	14.1257087670802
46	11.6032607729589
47	13.6212191682559
48	12.1077503717832
49	13.1167295694316
50	12.6122399706075

Valutazioni Funzionali	1e-3	1e-8	1e-13
	225	772	1128

Esercizio 16:

Costruire una function, lagrange.m, avente la stessa sintassi della function spline di Matlab, che implementi, in modo vettoriale, la forma di Lagrange del polinomio interpolante una funzione.
N.B.: il risultato dovra avere le stesse dimensioni del dato di ingresso; questo vale anche per gli esercizi a seguire.

Codice:

```
function YQ=lagrange(X,Y,XQ)
%Input: X: Vettore colonna contenete le ascisse d'interpolazione che devono essere distinte l'una dall'altra,
%Y: Vettore colonna contenete i valori della funzione nelle ascisse d'interpolazione
%XQ: Vettore colonna contenente le ascisse in 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 d'interpolazione e di valori della funzione non uguale!"),
end
if (length(X) ~= length(unique(X))), error("Ascisse d'interpolazione non distinte!"),
end %uso unique per retituire un vettore senza ripetizioni di X
if(isempty(XQ)), error("Il vettore contenente le ascisse in cui interpolare la funzione è vuoto!"),
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
```

Per questa function è stato sufficiente calcolare i polinomi di base di Lagrange:

$$Lin(x) = \prod_{j=0, j\neq i}^{n} \frac{x-x_j}{x_i-x_j}$$

per poi ricavarmi direttamente i valori (che ho salvato nel vettore YQ) tramite la sommatoria:

$$p(x) = \sum_{i=0}^{n} fi * Lin(x)$$

(dove n è il numero di ascisse d'interpolazione meno uno) dove i valori fi sono contenuti nel vettore Y e le ascisse d'interpolazione nel vettore X.

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.

Codice:

```
function YQ=newtonTer(X,Y,XQ)
%implementa in modo vettoriale la forma di Newton del polinomio interpolante una funzione.
%Input: X: Vettore colonna contenete le ascisse d'interpolazione che devono essere distinte,
%Y: Vettore colonna contenete i valori della funzione nelle ascisse d'interpolazione
%XQ: Vettore contente le ascisse in 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 d'interpolazione e di valori della funzione non uguale!"),
end
if (length(X) ~= length(unique(X))), error("Ascisse d'interpolazione non distinte!"),
end %uso la function unique che restuisce un vettore contenente i valori senza ripetizioni di X
if(isempty(XQ)), error("Il vettore contenente le ascisse in cui interpolare la funzione è vuoto!"),
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
```

Per ricavarmi questa function è stato necessario in primo luogo ricavarmi il vettore delle differenze divise grazie alla function divdif riportata in fondo che sfrutta la propietà 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 è unico.

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.

Codice:

```
function yy = hermite(xi,fi, f1i, xx)
%Input
%xi: Vettore colonna contenete le ascisse d'interpolazione distinte,
%fi: Vettore colonna contenete i valori della funzione nelle ascisse d'interpolazion
%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 Hermitte in forma di Newton
%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.
if(isempty(xx)), error("Il vettore contenente le ascisse in cui interpolare la funzione è vuoto!"),
end
if(length(xi)~=length(fi)), error("Numero di ascisse d'interpolazione e di valori della funzione non uguale!"),
end
if (length(xi) ~= length(unique(xi))), error("Ascisse d'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
```

Questa function è molto simile a quella precedente se non per l'algoritmo delle differenze divise che, in questo caso include anche le derivate prime della funzione che intendiamo interpolare, oltre al fatto che, costruendo un polinomio di grado 2n+1 dove n+1 è il numero di ascisse d'interpolazione, deve essere adattato l'algoritmo di horner di conseguenza.

Esercizio 19:

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

Codice:

```
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));
```

In questo caso la function consiste di un solo passaggio fondamentale che non è altro che il calcolo ripetuto di:

$$x_{n-i} = \frac{a+b}{2} + \frac{b-a}{2} cos\left(\frac{(2i+1)\pi}{2(n+1)}\right)$$

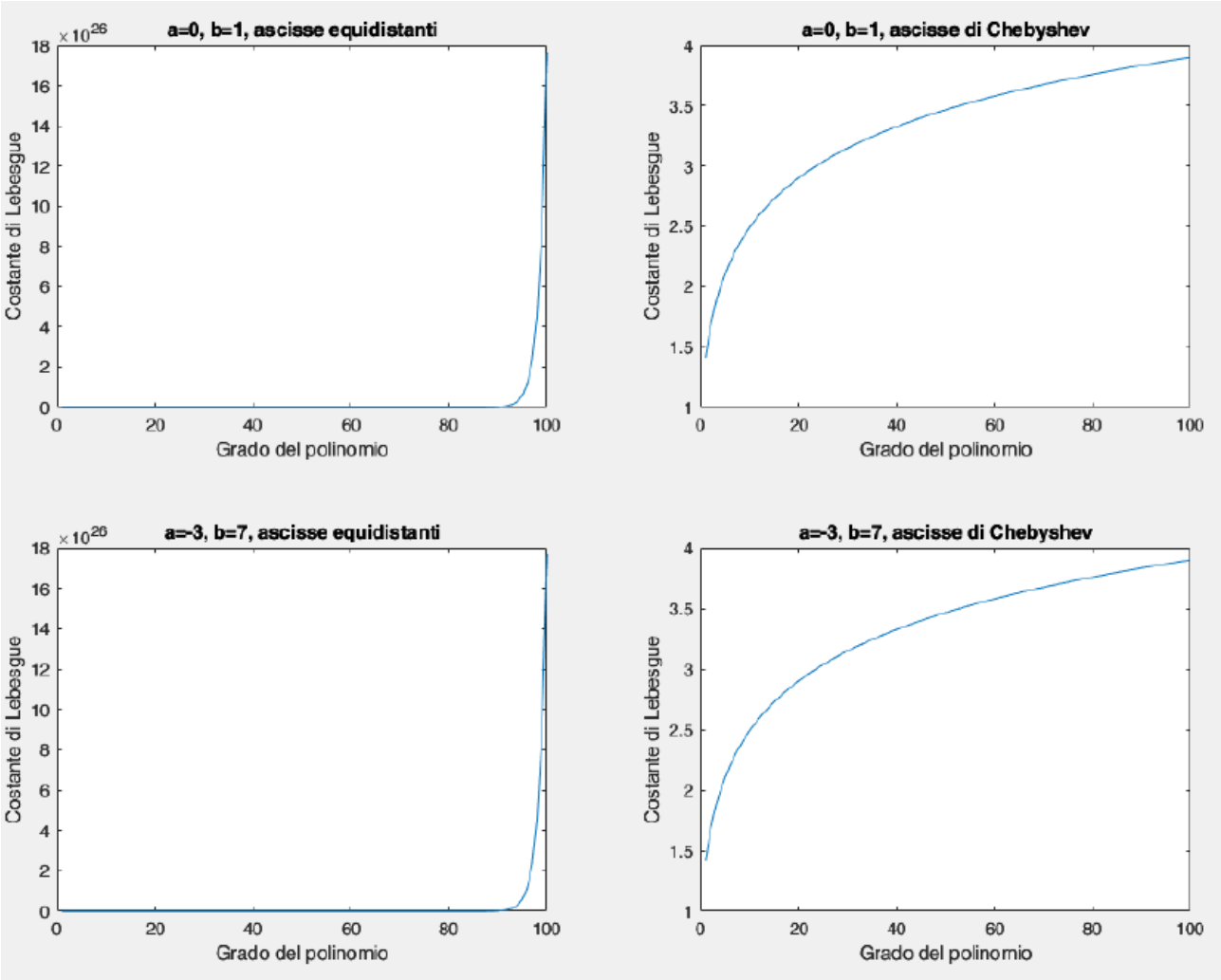
per i che va da 0 ad n grado del polinomio di cui vogliamo ricavarci le ascisse nell'intervallo [a; b].

Esercizio 20:

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 specicato nel vettore nn, utilizzando ascisse equidistanti, se type=0, o di Chebyshev, se type=1 (utilizzare 10001 punti equispaziati nell'intervallo [a,b] per ottenere ciascuna componente di ll). Gracare i risultati ottenuti, per nn=1:100, utilizzando [a,b]=[0,1] e [a,b]=[-3,7]. Giusticare i risultati ottenuti.

Codice:

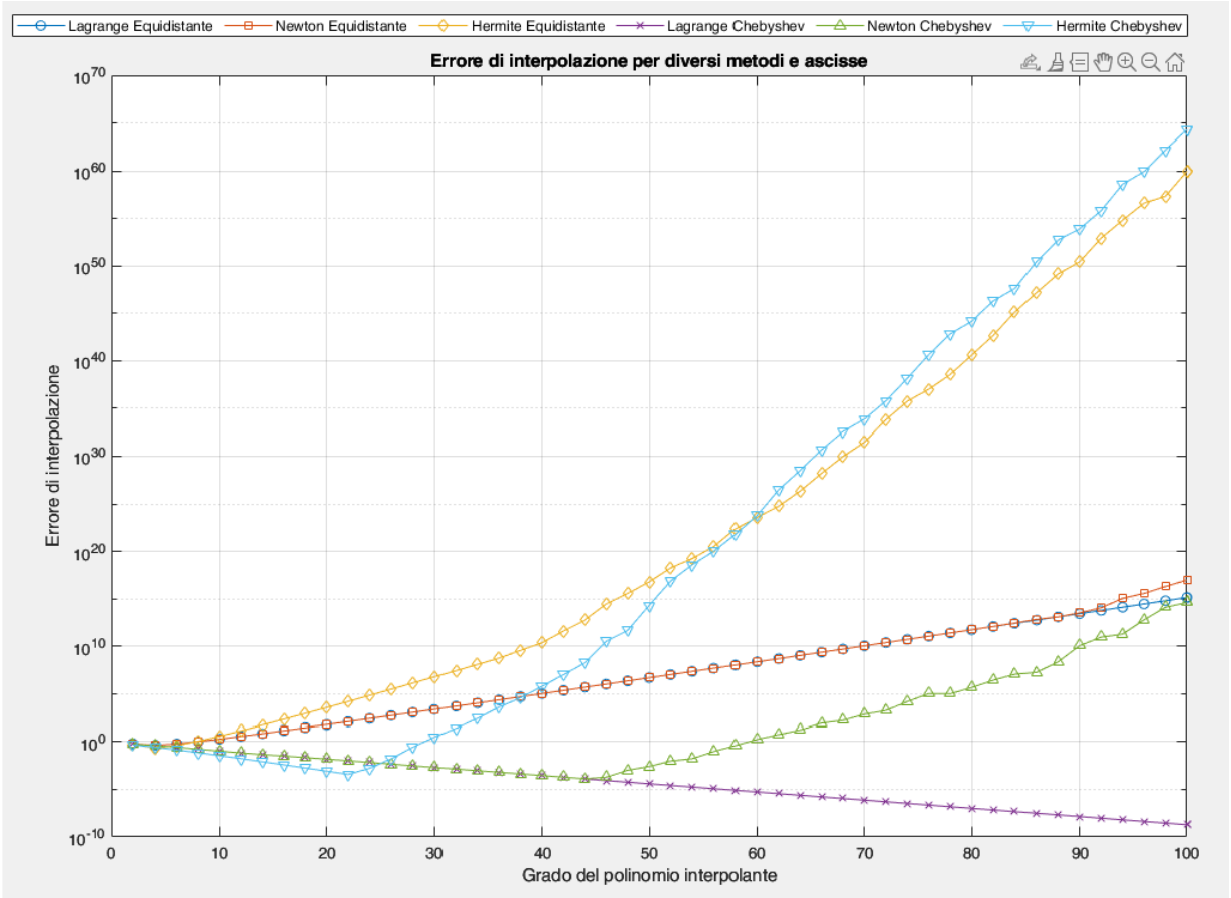
```
function ll = lebesgue(a, b, nn, type)
% Input: a,b = inizio e fine intervallo, nn = vettore con specificato il grado dei polinomi, type = specifica che tipo di ascisse di interpolazione usare
% se 0 utilizza le ascisse equispaziate nell'intervallo [a,b] altrimenti se 1 utilizza le ascisse di chebyshev
% 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 = chebyshev(nn(j), a, b);
end
lin = lebesgue_function(x,xq);
ll(j) = max(abs(lin));
end
end
function lin = lebesgue_function(x,xq)
n = length(x);
if length(unique(x)) ~= n
error('Errore: Alcune ascisse sono uguali.');
end
lin = zeros(size(xq));
for j = 1:n
L = ones(size(xq));
for i = 1:n
if i ~= j
L = L .* (xq - x(i)) / (x(j) - x(i));
end
end
lin = lin + abs(L);
end
end
```

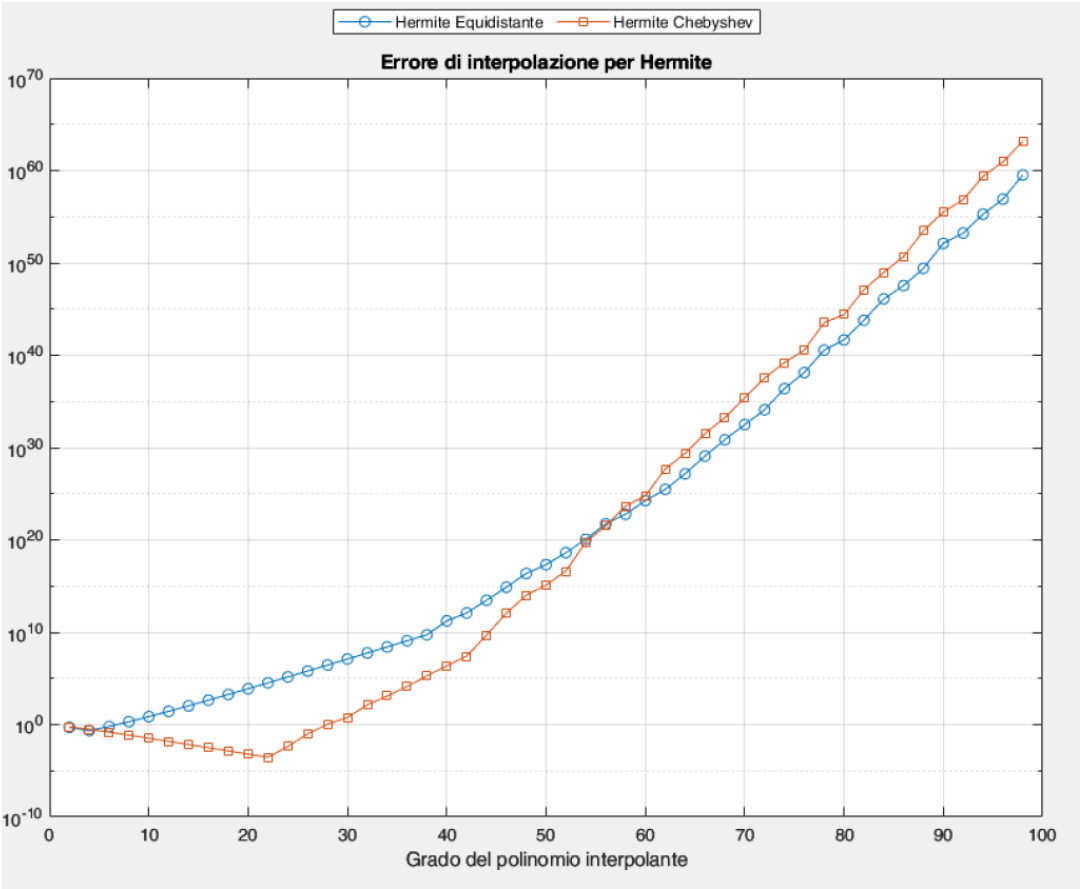


Come ci aspettavamo, la costante di Lebesgue è indipendente dall'intervallo [a,b] scelto e con la scelta delle ascisse di Chebyshev cresce in maniera quasi ottimale, cioè $\approx \log n$.

Esercizio 21:

Utilizzando le function dei precedenti esercizi, gracare (in semilogy) l'andamento errore di interpolazione (utilizzare 10001 punti equispaziati nell'intervallo per ottenerne la stima) per la funzione di Runge, utilizzando sia le ascisse equidistanti che di Chebyshev, per i polinomi interpolanti di grado nn=2:2:100. Gracare l'errore di interpolazione anche per i polinomi interpolanti di Hermite di grado nn=3:2:99, sia utilizzando ascisse equidistanti che ascisse di Chebyshev, nell'intervallo considerato.





Esercizio 22:

Costruire una function, myspline.m, avente sintassi yy = myspline(xi, fi, xx, type) dove type=0 calcola la spline cubica interpolante naturale i punti (xi,fi), e type~=0 calcola quella not-a-knot (default).

Codice:

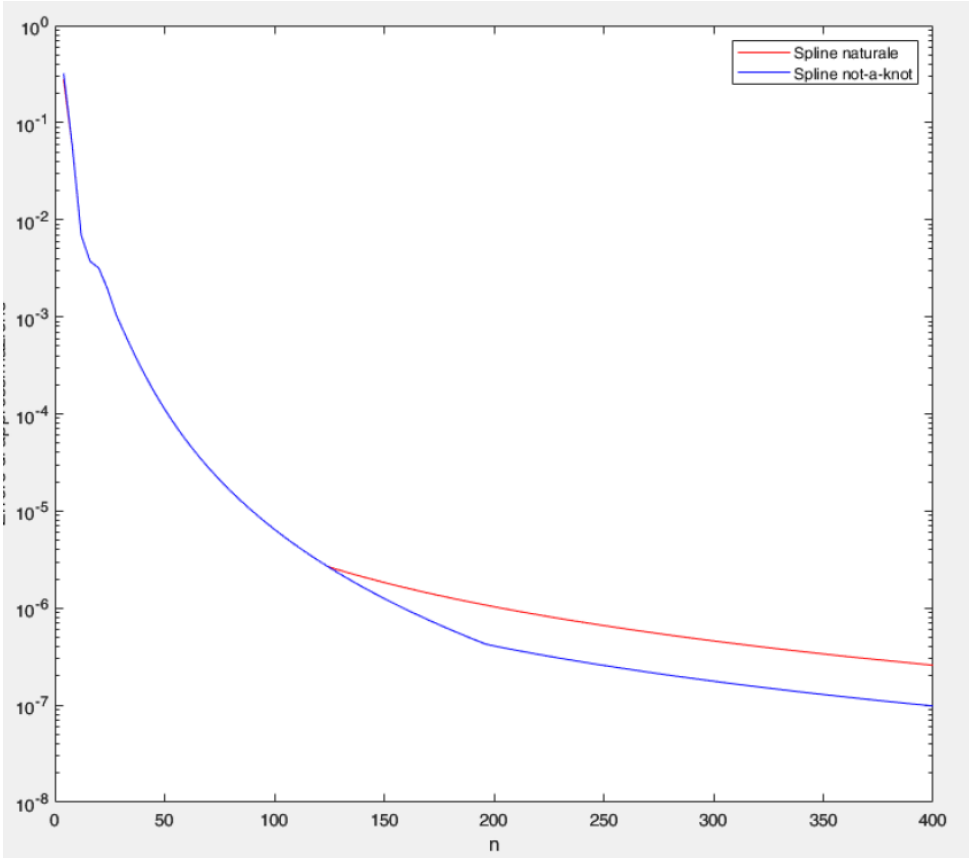
```
function yy = myspline ( xi , fi , xx , type)
% Calcola la spline cubica interpolante naturale o not a knot a seconda del valore "tipo"
% Input: xi = vettore delle ascisse di interpolazione, fi = vettore dei valori della funzione nelle ascissedi interpolazione
% xq = punti in cui si vuole calcolare il polinomio interpolante, tipo = tipo di spline, 0 per naturale oppure 1 per not a knot
% Output: yy = valori che la spline assume nei punti xx
if nargin < 4 error
("Errore: numero degli argomenti errato") ;
elseif length ( xi ) ~= length ( fi ) error
("Errore: i due vettori devono avere la stessa dimensione") ;
elseif length ( xi ) ~= length ( unique ( xi ) ) error
("Errore: le ascisse devono essere distinte tra loro" ) ;
elseif size ( xi , 2) > 1 || size ( fi , 2) > 1 error
("Errore: vettore colonna non valido" )
elseif isempty ( xx )
error ("Errore: partizione assegnata vuota") ;
end
n = length ( xi ) - 1;
epsilon = zeros (n -1 ,1) ;
q = zeros (n -1 ,1) ;
for i = 2 : n
hi = ( xi ( i ) - xi (i -1) ) ;
hi1 = ( xi ( i +1) - xi ( i ) ) ;
q (i -1) = hi / ( hi + hi1 ) ;
epsilon (i -1) = hi1 / ( hi + hi1 ) ;
end
l = size ( xi ) ;
diff_div = fi ;
l = l -1;
for j = 1 : 2
for i = l +1 : -1 : j +1
diff_div ( i ) = ( diff_div ( i ) - diff_div (i -1) ) /( xi ( i ) - xi (i - j ) ) ;
end
end
diff_div = diff_div (3: l +1) ; % Calcolo le differenze divise
a = 2* ones (n -1 ,1) ;
if type == 0
m = tridia (a , q , epsilon , diff_div * 6) ;
m = [0; m ; 0];
else
diff_div = diff_div * 6;
a(1) = 2 - q(1);
epsilon(1) = epsilon(1) - q(1);
diff_div_1 = diff_div(1);
diff_div(1) = (1 - q(1)) * diff_div(1);
q(1) = 0;
a(n-1) = 2 - epsilon(n-1);
q(n-1) = q(n-1) - epsilon(n-1);
diff_div_n = diff_div(n-1);
diff_div(n-1) = (1 - epsilon(n-1)) * diff_div(n-1);
epsilon(n-1) = 0;
m = tridia(a, epsilon, q, diff_div);
m0 = diff_div_1 - m(1) - m(2);
mn = diff_div_n - m(n-1) - m(n-2);
m = [m0; m; mn];
end
yy = zeros ( length ( xx ) , 1) ;
for j = 1 : length ( xx )
for i = 2 : length ( xi )
if (( xx ( j ) >= xi (i -1) && xx ( j ) <= xi ( i ) ) || xx ( j ) < xi (1) )
hi = xi ( i ) - xi (i -1) ;
ri = fi (i -1) - hi ^2/6* m (i -1) ;
qi = ( fi ( i ) - fi (i -1) ) / hi - hi /6*( m ( i ) -m (i -1) ) ;
yy ( j ) =(( xx ( j ) - xi (i -1) ) ^3* m ( i ) +( xi ( i ) - xx ( j ) ) ^3* m (i -1) ) /(6* hi ) + qi*( xx ( j ) - xi (i -1) ) + ri ;
break
end
end
end
return ;
end
```

```
%Il codice utilizza la function di supporto tridia(a,b,c,g):
function x = tridia (a , b , c , g )
% Risolve il sistema lineare fattorizzabile LU
% Input :
% a = Vettore diagonale principale
% b = Vettore sovradiagonale
% c = Vettore sottodiagonale
% g = Vettore dei termini noti
% Output: x = Vettore soluzione
n = length ( g ) ;
x = g ;
for i = 1 : n -1
b ( i ) = b ( i ) / a ( i ) ;
a ( i +1) = a ( i +1) - b ( i ) * c ( i ) ;
x ( i +1) = x ( i +1) - b ( i ) * x ( i ) ;
end
x ( n ) = x ( n ) / a ( n ) ;
for i = n -1 : -1 : 1
x ( i ) = ( x ( i ) - c ( i ) * x ( i +1) ) / a ( i ) ;
end
return ;
end
```

Esercizio 23:

Graficare, utilizzando il formato semilogy, l'errore di approssimazione utilizzando le spline interpolanti naturale e not-a-knot per approssimare la funzione di Runge sull'intervallo [-5,5], utilizzando una partizione delta espressa nel testo con ascisse equidistanti e n=4:4:400. [Utilizzare 10001 punti equispaziati nell'intervallo [-5; 5] per ottenere la stima dell'errore.

Soluzione:



Esercizio 24:

E' noto che un fenomeno fisico evolve come y = xn, con n incognito. Il file data.mat, reperibile a: https://drive.google.com/file/d/14u2Pjnl_BZN1GWEFCZd0tqKCObGGI-M5/view?usp=share_link contiene 1000 coppie di dati (xi, yi), in cui la seconda componente è affetta da un errore con distribuzione Gaussiana a media nulla e varianza "piccola". Utilizzando un opportuno polinomio di approssimazione ai minimi quadrati, stimare il grado n. Argomentare il procedimento seguito, graficando la norma del residuo rispetto a valori crescenti di n. E' richiesto il codice Matlab dell'algoritmo che avrete implementato (potete utilizzare, se lo ritenete opportuno, la function polyfit di Matlab).

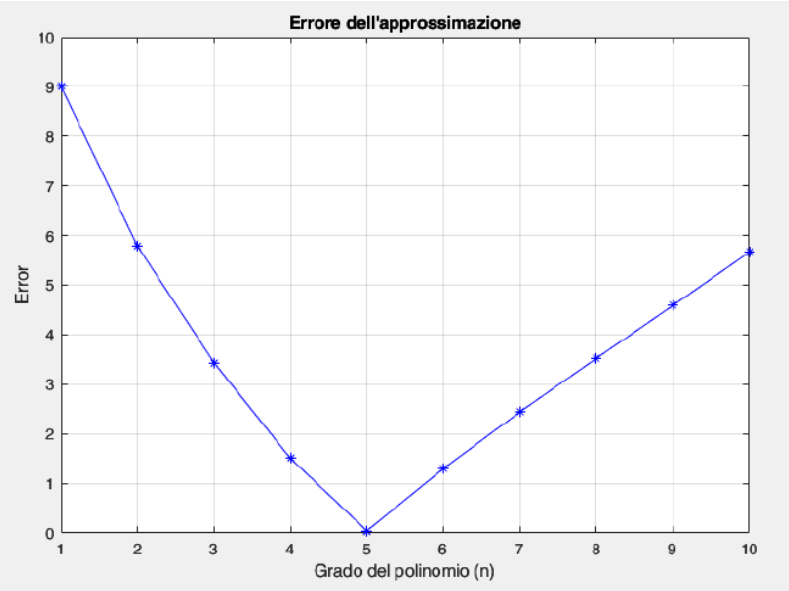
Soluzione:

Codice implementato:

```
data = load('data.mat').data;
x = data(:,1);
y = data(:,2);
% Inizializza vettore errore
error = zeros(1,15);
for m = 1:15 %Calcola i coefficienti del polinomio di approssimazione ai minimi quadrati che meglio approssima i dati ordine decrescente di grado.
p = polyfit(x, y, m); % Calcola il vettore di valori del polinomio approssimante sui punti x dei dati.
y_fit = horner(p, x); % Calcola l'errore di approssimazione tra i dati originali e quelli approssimati
error(m) = norm(x.^m - y_fit);
end

% semilogy
figure;
semilogy(1:15, error, 'b*-');
xlabel('Grado del polinomio (n)');
ylabel('Error');
title("Errore dell'approssimazione");
grid on;
function y = horner(p, x)
y = p(1);
for i = 2:length(p)
y = y .* x + p(i);
end
end
```

Questo script carica un set di dati da un file .mat e tenta di approssimare i dati con polinomi di grado variabile, da 1 a 15. Per ogni grado del polinomio, calcola i coefficienti del polinomio di approssimazione ai minimi quadrati tramite polyfit, e poi valuta il polinomio sui punti x utilizzando l'algoritmo di Horner. Successivamente, calcola l'errore di approssimazione tra i dati originali e quelli approssimati, utilizzando la norma euclidea.



Possiamo osservare come sia inutile incrementare oltre 5 il grado del polinomio di approssimazione ai minimi quadrati in quanto l'errore non diminuisce, anzi aumenta, all'aumentare del grado del polinomio.

Esercizio 25:

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

Codice:

```
function coef=calcolaCoeff(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=calcolaoefficienti(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
```

Possiamo notare come, essendo i pesi di quadratura simmetrici, è stato sufficiente calcolare meno della metà dei pesi richiesti considerando anche che la somma di tutti i pesi di quadratura è uguale al grado n della formula di Newton-Cotes di cui stiamo calcolando i pesi di quadratura. I pesi della quadratura ricavati con la precedente function al variare del grado n sono riportati nei vettori sottostanti:

$$n = 1 \quad \begin{bmatrix} \frac{1}{2} \\ \frac{1}{2} \end{bmatrix}$$

$$n = 2 \quad \begin{bmatrix} \frac{1}{3} \\ \frac{4}{3} \\ \frac{1}{3} \end{bmatrix}$$

$$n = 3 \quad \begin{bmatrix} \frac{3}{8} \\ \frac{9}{8} \\ \frac{9}{8} \\ \frac{3}{8} \end{bmatrix}$$

$$n = 4 \quad \begin{bmatrix} \frac{14}{45} \\ \frac{64}{45} \\ \frac{8}{15} \\ \frac{64}{45} \\ \frac{14}{45} \end{bmatrix}$$

$$n = 5 \quad \begin{bmatrix} \frac{95}{288} \\ \frac{125}{96} \\ \frac{125}{144} \\ \frac{125}{144} \\ \frac{125}{96} \\ \frac{95}{288} \end{bmatrix}$$

$$n = 6 \quad \begin{bmatrix} \frac{41}{140} \\ \frac{54}{35} \\ \frac{27}{140} \\ \frac{68}{35} \\ \frac{27}{140} \\ \frac{54}{35} \\ \frac{41}{140} \end{bmatrix}$$

$$n = 7 \quad \begin{bmatrix} \frac{1073}{3527} \\ \frac{810}{559} \\ \frac{343}{640} \\ \frac{649}{536} \\ \frac{649}{536} \\ \frac{343}{640} \\ \frac{810}{559} \\ \frac{1073}{3527} \end{bmatrix}$$

$$n = 9 \quad \begin{bmatrix} \frac{130}{453} \\ \frac{1374}{869} \\ \frac{243}{2240} \\ \frac{5287}{2721} \\ \frac{704}{1213} \\ \frac{704}{1213} \\ \frac{5287}{2721} \\ \frac{243}{2240} \\ \frac{1374}{869} \\ \frac{130}{453} \end{bmatrix}$$

Esercizio 26:

Scrivere una function Matlab, [If,err] = composita(fun, a, b, k, n) che implementi la formula composita di NewtonCotes di grado k su n+1 ascisse equidistanti, con n multiplo pari di k, in cui:
• fun e la funzione integranda (che accetta input vettoriali);
• [a,b] e l'intervallo di integrazione;
• k, n come su descritti;
• If e l'approssimazione dell'integrale ottenuta;
• err e la stima dell'errore di quadratura.ù

Codice:

```
function [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 dell' intervallo di integrazione errati ");
end
if(mod(n, k) ~= 0 || mod(n/2, 2)~= 0)
error("n non e' un multiplo pari di k");
end
tol = 1e-3;
mu = 1 + mod (k ,2);
c = NewtonWeights (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
```

Esercizio 27:

Utilizzare la function composita per ottenere l'approssimazione dell'integrale descritto sul testo con le formule composite di Newton-Cotes di grado k=1,2,3,4,5,6. Per tutte, utilizzare n=12.

Soluzione:

k	Approssimazione	Errore relativo
1	-0.176260	0.000512
2	-0.177572	0.000319
3	-0.177744	0.000215
6	-0.177454	0.000042

I risultati ottenuti mostrano che l'approssimazione può essere migliorata andando ad aumentare il grado k della function composita. In ogni caso, come si può vedere, per k=6 l'errore relativo diminuisce. Questo ci può suggerire la presenza di un limite oltre il quale andando ad aumentare ulteriormente il grado k non si ottengono miglioramenti significativi dell'approssimaizone.

Esercizio 28:

Implementare la formula composita adattativa di Simpson.

Codice:

```
function [I2,nfeval] = simpsonAdatt( f, a, b, tol, fa, fm,fb )
%Input
%f: Function handler contenente la funzione di cui vogliamo calcolare l'approssimazione dell'integrale definito,
%a,b: Estremi dell'intervallo d'integrazione,
%tol: Tolleranza richiesta,
%(fa,fm,fb): parametri di lavoro da non specificare quando si vuole invocare la funzione
%Output: If: approssimazione dell'integrale ottenuta, nfeval: numero di valutazioni funzionali effettuate,
%Calcola un'approssimazione dell'integrale definito della funzione f con estremi a e b utilizzando la formula adattiva di Newton-Cotes
%di grado 2 con tollerenza tol

if(tol<=0), error('Tolleranza non valida!'),end
m = ( a + b )/2; %calcolo il punto medio
nfeval=0;
if nargin<=4
fa = f(a);
fb = f(b);
fm = f(m);
nfeval=3;
end
nfeval=nfeval+2; %aggiorno il numero di valutazioni funzionali richieste
ma=(a+m)/2; %calcolo il punto medio dell'intervallo [a,m]
mb=(b+m)/2; %calcolo il punto medio dell'intervallo [m,b]
fma=f(ma);
fmb=f(mb);
h = b - a;
I1= ( h/6 )*( fa +4*fm+ fb); %Applico la formula di Simpson su fa,fm ed fb
I2 = I1/2 + (h/6)*(2*fma + 2*fmb-fm); %Applico la formula composita di Simpson su fa,fma,fm,fmb e fb
e = abs( I2 - I1 )/15; %mi ricavo una stima dell'errore
if e>tol %se la tolleranza non è rispettata riapplico la function sui due sottointervalli
[lI,nfevall]=adattivasimp(f,a, m, tol/2, fa, fma, fm );
[rI,nfevalr]=adattivasimp(f,m, b, tol/2, fm, fmb, fb );
I2=lI+rI;
nfeval=nfeval+nfevalr+nfevall; %sommo le valutazioni funzionali richieste nei due sottointervalli
end
return
end
```

Esercizio 29:

Implementare la formula composita adattativa di Newton-Cotes di grado k=4.

Codice:

```
function [I, nfeval] = NewtonCotesAdatt(fun, a, b, tol, fa, f2, fm, f3, fb)
% Input: fun = funzione integranda, a = estremo inferiore dell'intervallo di integrazione, b = estremo superiore dell'intervallo di integrazione
% tol = tolleranza richiesta, [fa,f2,f4,f5,fm] = valori calcolati negli estremi di integrazione e nei punti, intermedi. Sono parametri opzionali che
%diminuiscono il numero di valutazioni necessarie
% Output: I: approssimazione dell'integrale, nfeval = numero di valutazioni funzionali effettuate
if a > b
error("Errore: gli estremi dell'intervallo non vanno bene");
end
if tol < 0
error("Errore: la tolleranza specificata è minore o uguale a 0");
end
xm = (a+b)/2;
x2 = (a+xm)/2;
x3 = (xm+b)/2;
nfeval = 0;
if nargin == 4
fa = feval(fun, a);
fb = feval(fun, b);
fm = feval(fun, xm);
f2 = feval(fun, x2);
f3 = feval(fun, x3);
nfeval = nfeval + 5;
end
h = (b-a)/90;
x4 = (a+x2)/2;
x5 = (x2 + xm)/ 2;
x6 = (xm + x3)/ 2;
x7 = (x3 + b) / 2 ;
f4 = feval(fun, x4);
```

```
f5 = feval(fun, x5);
f6 = feval(fun, x6);
f7 = feval(fun, x7);
nfeval = nfeval + 4;
Itemp = h * (7*fa + 32*f2 + 12*fm + 32*f3 + 7*fb);
I = h/2 * (7*fa + 32*f4 + 12*f2 + 32*f5 + 14*fm + 32*f6 + 12*f3 + 32*f7 + 7*fb);
err = abs(I - Itemp) / 63;
if err > tol
[I1, nfeval1] = NewtonCotesAdatt(fun, a, xm, tol / 2, fa, f4, f2, f5, fm);
[I2, nfeval2] = NewtonCotesAdatt(fun, xm, b, tol / 2, fm, f6, f3, f7, fb);
I = I1 + I2;
nfeval = nfeval + nfeval1 + nfeval2;
end
end
```

Esercizio 30:

Confrontare le formule adattative degli ultimi due esercizi, tabulando il numero di valutazioni funzionali effettuate, rispetto alla tolleranza tol = 1e-2, 1e-3, ..., 1e-9, per ottenere l'approssimazione dell'integrale descritto nel testo. Costruire un'altra tabella, in cui si tabula l'errore vero (essendo l'integrale noto, in questo caso) rispetto a tol.

Tolleranza	Errore	Errore vero (NC)	Errore vero (Simpson)
1.000000e-02		2.305180e-06	2.935280e-04
1.000000e-03		1.065361e-06	4.572239e-04
1.000000e-04		9.852604e-10	2.635476e-05
1.000000e-05		3.104168e-06	2.344814e-06
1.000000e-06		3.697328e-08	3.007626e-07
1.000000e-07		2.281509e-08	3.656476e-08
1.000000e-08		2.133584e-09	3.211098e-09
1.000000e-09		1.496632e-10	3.098395e-10

Tolleranza	Valutazioni funzionali (NC)	Valore calcolato (NC)	Valutazioni funzionali (Simpson)	Valore calcolato (Simpson)
1.000000e-02	153	-8.691346e-01	201	-8.694258e-01
1.000000e-03	193	-8.691333e-01	333	-8.695895e-01
1.000000e-04	257	-8.691323e-01	605	-8.691586e-01
1.000000e-05	369	-8.691354e-01	1061	-8.691346e-01
1.000000e-06	553	-8.691323e-01	1869	-8.691326e-01
1.000000e-07	793	-8.691323e-01	3277	-8.691323e-01
1.000000e-08	1177	-8.691323e-01	5921	-8.691323e-01
1.000000e-09	1753	-8.691323e-01	10589	-8.691323e-01