Elaborato Calcolo Numerico

Alessio Santoro (7029440) - Bro aggiungiti anche te

A.A. 2022/2023

Nota: Per gli esercizi che prevedono delle *funcion* Matlab, si specifica nella relativa risposta al quesito i file tra gli alleagti a cui essa si riferisce.

1

Si considera lo sviluppo delle funzioni $f(x-h), f(x+h), f(x+2h), f(x+3h)$:

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

Si sostiuiscono le espressioni così trovate nella parte sinistra dell'equaziome iniziale e si ottiene la seguente espressione:

$$\begin{aligned}
&-\frac{1}{4}\left[f(x) - hf'(x) + \frac{h^2}{2}f''(x) - \frac{h^3}{6}f^{(3)}(x) + \frac{h^4}{24}f^{(4)}(x) + O(h^5)\right] + \\
&-\frac{5}{6}\left[f(x)\right] + \\
&+\frac{3}{2}\left[f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f^{(3)}(x) + \frac{h^4}{24}f^{(4)}(x) + O(h^5)\right] + \\
&-\frac{1}{2}\left[f(x) + 2hf'(x) + \frac{4h^2}{2}f''(x) + \frac{8h^3}{6}f^{(3)}(x) + \frac{16h^4}{24}f^{(4)}(x) + O(h^5)\right] + \\
&+\frac{1}{12}\left[f(x) + 3hf'(x) + \frac{9h^2}{2}f''(x) + \frac{27h^3}{6}f^{(3)}(x) + \frac{81h^4}{24}f^{(4)}(x) + O(h^5)\right]
\end{aligned}$$

Si procede a moltiplicare i coefficienti di ogni espressione e poi raccogliere i termini che contengono le derivate dello stesso ordine, una volta raccolti i temrini assumono i seguenti valori che, stando all'equazione iniziale dovranno poi essere

sommati:

$$f(x)\left[-\frac{1}{4}-\frac{5}{6}+\frac{3}{2}-\frac{1}{2}+\frac{1}{12}\right]=0 \tag{1}$$

$$f'(x)\cdot h\left[\frac{1}{4}+\frac{3}{2}-\frac{1}{2}2+\frac{1}{12}3\right]=hf'(x) \tag{2}$$

$$f''(x)\cdot\frac{h^2}{2}\left[-\frac{1}{4}+\frac{3}{2}-\frac{1}{2}4+\frac{1}{12}9\right]=0 \tag{3}$$

$$f^{(3)}(x)\cdot\frac{h^3}{6}\left[-\frac{1}{4}+\frac{3}{2}-\frac{1}{2}8+\frac{1}{12}27\right]=0 \tag{4}$$

$$f^{(4)}(x)\cdot\frac{h^4}{24}\left[-\frac{1}{4}+\frac{3}{2}-\frac{1}{2}16+\frac{1}{12}81\right]=0 \tag{5}$$

Dalle espressioni (1)...(5) e dalle proprietà degli "O-grande" di moltiplicazione per una costante segue l'asserto.

2

La doppia precisione dello standard IEEE 754 è una rappresentazione in base binaria, in forma normalizzata $(1.f)$ che approssima per arrotondamento e occupa 64 bit, di cui 52 dedicati alla frazione (53 alla mantissa).
Si può dunque ottenere il valore della precisione di macchina (u) dalla seguente espressione, dove: $b=2$ rappresenta la base, e $m=53$ la mantissa:

$$u=\frac{1}{2}b^{1-m}=2^{-53}$$

Invece `eps` è definito dalla stessa funzione `help` di Matlab come la distanza tra `1.0` e il maggior valore a doppia precisione successivo disponibile, ovvero 2^{-52}.
Si osserva infatti che, considerato il valore $x=1+u=1+2^{-53}\neq 1$ e sia fl la funzione di floating, allora vale che $fl(x)=1$, poichè $u=2^{-53}<2^{-52}=$`eps`.
Vi è dunque un errore di rappresentazione del valore x (ε_x), determinato dalla seguente espressione:

$$\varepsilon_x=\frac{|x-fl(x)|}{|x|}=\frac{|1+2^{-53}-1|}{|1+2^{-53}|}=\frac{|2^{-53}|}{|1+2^{-53}|}<|2^{-53}|=u$$

3

La cancellazione numerica è quel fenomeno in cui, sommando in aritmetica finita due numeri quasi opposti si verifica la perdita di cifre signficative. Questo è dovuto all'espressione del numero di condizionamento della somma in aritmetica finita (k) che per due valori x e y è dato da:

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

Infatti, se $x\to -y$ allora $k\to\infty$ e la somma tra x e y risulta mal condizionata.

4

Sia $x^* \in \mathbb{R}$ il valore di cui si ricerca la radice sesta.
Per calcolarlo si definisce una funzione $f(x)$ come segue:

$$f(x) = x^6 - x^*$$

La cui derivata è:

$$f'(x) = 6x^5$$

La funzione $f(x)$ si annulla solo nella radice sesta di x^*, quindi avendo un'approsimazione iniziale x_0 si può applicare il metodo di Newton alla funzione $f(x)$ per ricercarne una radice che coinciderà con il valore cercato:

$$x_{i+1} = x_i + \frac{f(x_i)}{f'(x_i)} = x_i - \frac{x_i^6 - x^*}{6x_i^5} = \frac{1}{6}\left[5x_i + \frac{x^*}{x_i}\right]$$

La `function` che implementa il metodo presentato è contenuta nel file `radice.m`:

```
function root = radice(x)
%
%   root = radice(x)
%
%   Questa funzione calcola la radice sesta di un valore non negativo
%   attraverso il metodo iterativo di Newton utilizzando solo operazioni elementari
%
%   Input:
%       x: valore di cui si vuole calcolare la radice sesta
%   Output:
%       root; risultato del calcolo
if(x<0), error("Value x must be not negative");end
if(x==0)
    root = 0;
    return;
end

root = x;
er = 1;

while(er >= eps*(1+abs(x)))
    xi = (5*root+x/root^5)/6;
    er = abs(root - xi);
    root = xi;
end
return;
end
```

I dati sul confronto tra il risultato offerto dalla funzione e il valore `x^(1/6)` sono contentuti nel file `4_table.txt`:

x	radice(x)	x^(1/6)	errore
1e-10	0.021544	0.021544	3.4694e-18
1.1288e-09	0.032268	0.032268	6.9389e-18
1.2743e-08	0.048329	0.048329	6.9389e-18
1.4384e-07	0.072385	0.072385	1.3878e-17
1.6238e-06	0.10841	0.10841	4.1633e-17
1.833e-05	0.16238	0.16238	0
0.00020691	0.2432	0.2432	2.7756e-17
0.0023357	0.36425	0.36425	5.5511e-17
0.026367	0.54556	0.54556	0
0.29764	0.81711	0.81711	1.1102e-16
3.3598	1.2238	1.2238	2.2204e-16
37.927	1.833	1.833	0
428.13	2.7453	2.7453	4.4409e-16
4832.9	4.1118	4.1118	8.8818e-16
54556	6.1585	6.1585	0
6.1585e+05	9.2239	9.2239	1.7764e-15
6.9519e+06	13.815	13.815	1.7764e-15
7.8476e+07	20.691	20.691	3.5527e-15
8.8587e+08	30.99	30.99	3.5527e-15
1e+10	46.416	46.416	7.1054e-15

5

Il seguente testo è cotnenuto nel file `newtonMethod.m` e rappresenta il metodo di Newton:

```
function [x,n] = newtonMethod(f,df, x0, tol)
%
%   x = newtonMethod(f,df,x0,tol, itmax)
%
%   Ricerca la radice di una funzione di cui è nota la derivata a partire
%   da un approssimazione iniziale mediante il metodo di Newton
%
%   Input:
%       f: funzione di cui si ricercano le radici
%       df: derivata della funzione f
%       x0: approssimazione iniziale della radice
%       tol: errore assoluto ammissibile
%   Output:
%       x: approssimazione della radice di f
%       n: numero di iterazioni eseguite

%controllo valori input
if nargin ~= 4, error("Missing arguments"); end
```

```
if tol<0, error("Invalid arguments: tolerance must be non negative"); end
x = x0;
fx = feval(f,x);
dfx =  feval(f,x);

x = x0- fx/dfx;

n = 1;
while abs(x-x0) > tol*( 1 + abs(x0))

    x0 = x;

    fx = feval(f,x0);
    dfx = feval(df, x0);

    if dfx==0
        error("Value of derivative function is 0, invalid first approximation");
    end
    n = n+1;
    x = x0 - fx/dfx; %calcolo effettivo
end
return
end
```

Da qui in poi viene presentato il contenuto del file `secantsMethod.m` che rappresenta il metodo delle secanti:

```
function [x,i] = secantsMethod(f, x0, x1, tol)
%
%   x = secantsMethod(f,df,x0,tol, itmax)
%
%   Ricerca la radice di una funzione di cui è nota la derivata a partire
%   da un approssimazione iniziale mediante il metodo delle secanti
%
%   Input:
%       f: funzione di cui si ricercano gli 0
%       x0: prima approssimazione iniziale della radice
%       x1: seconda approssimazione iniziale della radice
%       tol: errore assoluto ammissibile
%   Output:
%       x: approssimazione della radice di f
%       i: numero di iterazioni eseguite

%controllo valori input
if nargin ~= 4, error("Missing arguments"); end
if tol<0, error("Invalid arguments: tolerance must be non negative"); end

fx0 = feval(f,x0);
fx1= feval(f,x1);
```

```
x = x1-(fx1*(x1-x0))/(fx1-fx0);

i = 1;
while abs(x-x1) > tol*( 1 + abs(x1))
    i = i+1;

    x0 = x1;
    x1 = x;
    fx0 = fx1;
    fx1 = feval(f,x1);
    if fx0 == fx1, error("Invalid initial approximations: "+ ...
            "function assume same value in different points");end
    x = x1-(fx1*(x1-x0)/(fx1-fx0));
end
return
end
```

6

Nel file `6_result.txt` è contenuta la tabella dei risultati delle funzioni precedentemente mostrate:

```
Tolleranza    Ris. Newton    Iterazoni Newton    Ris. secanti    Iterazioni secanti
    ----------    -----------    ----------------    ------------    ------------------

      0.001         0.73909              8              0.7391              4
      1e-06         0.73909              9             0.73909              6
      1e-09         0.73909             10             0.73909              7
      1e-12         0.73909             10             0.73909              7
```

Per entrambi i metodi, la parte più costosa computazionalmente è la valutazione funzionale, dato che tutte le altre operazioni che vengono svolte sono operazioni elementari.
Il metodo di Newton esegue due valutazioni in ogni iterazione.
Sia n il numero di iterazioni, il costo computazionale del metodo di Newton è dato da $2(n+1)$.
Il metodo delle secanti esegue due valutazioni iniziali e poi una per ogni iterazione, quindi il suo costo computazionale per n iterazioni è dato da $n+2$.

Tolleranza	Iterazioni Newton	Costo Newton	Iterazioni secanti	Costo secanti
10^{-3}	8	16	4	6
10^{-6}	9	18	6	8
10^{-9}	10	20	6	8
10^{-12}	10	20	7	9

7

La seguente tabella fornisce i risultati dell'utilizzo delle funzioni precedenti per calcolare la radice della funzione $f(x) = [x - cos(x)]^5$:

tolleranza	Newton ris.	Newton iter.	Secant ris.	Secant iter.
10e-3	0.74512	18	0.73015	26
10e-6	0.73909	49	0.73908	70
10e-9	0.73909	80	0.73909	115
10e-12	0.73909	111	0.73909	159

Dopo aver sviluppato la *function* `modifiedNewtonMethod.m` si sono riscontrati i seguenti risultati:

tolleranza	risultato Newton modificato	numero di iterazioni
1e-3	0.73909	22
1e-6	0.73909	23
1e-9	0.73909	24
1e-12	0.73909	24

Come atteso, i metodi di Newton e delle secanti sono più lenti a causa del metodo di Newton modificato, a causa della natura multipla della radice.
Infatti il metodo di Newton e quello delle secanti hanno convergenza quadratica nel caso di radici a molteplicità 1, ma solo lineare nel caso di radici multiple.
La modifica che abbiamo fatto, ovvero $x_{i+1} = x_i - m \cdot \frac{f(x_i)}{f'(x_i)}$, nonostante richieda che la molteplcità m della radice sia nota, ripristina la convergenza quadratica del metodo di Newton.
I rislutati sono contentuti nel file `table_7.txt` e si mostra di seguito il codice della funztion del metodo di Newton modificato:

```
function [x,n] = modifiedNewtonMethod(f,df, m, x0, tol)
%
%   x = newtonMethod(f,df,x0,tol, itmax)
%
%   Ricerca la radice di una funzione di cui è nota la derivata a partire
%   da un approssimazione iniziale mediante il metodo di Newton
%
%   Input:
%       f: funzione di cui si ricercano le radici
%       df: derivata della funzione f
%       m: molteplicità (nota) della radice
%       x0: approssimazione iniziale della radice
%       tol: errore assoluto ammissibile
%   Output:
%       x: approssimazione della radice di f
%       n: numero di iterazioni eseguite

%controllo valori input
if nargin ~= 5, error("Missing arguments"); end
if tol<0, error("Invalid arguments: tolerance must be non negative"); end
```

```
x = x0;
fx = feval(f,x);
dfx =  feval(f,x);

x = x0- m*fx/dfx;

n = 1;
while abs(x-x0) > tol*( 1 + abs(x0))

    x0 = x;

    fx = feval(f,x0);
    dfx = feval(df, x0);

    if dfx==0
        error("Value of derivative function is 0, invalid first approximation");
    end
    n = n+1;
    x = x0 - m*fx/dfx; %calcolo effettivo
end
return
end
```

8

Il codice della *function* è contenuto nel file `mialu.m`:

```
function x = mialu(A,b)
%
% x = mialu(A,b)
%
% presa in input una matrice ed un vettore calcola la soluzione del
% corrispondente sistema lineare utilizzando il metodo di fattorizzazione
% LU con pivoting
%
% Input:
% A = matrice dei coefficienti
% b = vettore dei termini noti
%
% Output:
% x = soluzione del sistema lineare
%
[m,n] = size(A);
if m ~= n
    error("La matrice non è quadrata");
end
if n ~= length(b)
    error("la lunghezza del vettore dei termini noti " + ...
        "non è coerente con quella della matrice");
```

```
end
p = (1:n).';
for i = 1:n
    [mi, ki] = max(abs(A(i:n,i)));
    if mi == 0
        error("la matrice è 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
return;
end
```

Un esempio di utilizzo è contenuto nel file di testo `ex_8_mialu.txt`:

9

Il codice della *function* è contenuto nel file `mialdl.m`:

```
function x = mialdl(A,b)
%
% x = mialdl(A,b)
%
% presa in input una matrice ed un vettore calcola la soluzione del
% corrispondente sistema lineare utilizzando il metodo di fattorizzazione
% LDL
%
% Input:
% A = matrice dei coefficienti
% b = vettore dei termini noti
%
% Output:
% x = soluzione del sistema lineare
%
[m,n] = size(A);
```

```
if m ~= n
    error("la matrice non è quadrata");
end
if n ~= length(b)
    error("la lunghezza del vettore dei termini noti " + ...
        "non è coerente con quella della matrice");
end
if A(1,1) <= 0
    error("la matrice non è sdp");
end
% la matrice non è memorizzata in forma compressa! (cit. libro)
A(2:n,1) = A(2:n,1)/A(1,1);
for i = 2:n
    v = (A(i,1:i-1).') .* diag(A(1:i-1,1:i-1));
    A(i,i) = A(i,i) - A(i,1:i-1)*v;
    if A(i,i) <= 0
        error("la matrice non è 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 = 2:n
    x(i:n) = x(i:n) - A(i:n,i-1) * x(i-1);
end
x = x ./ diag(A);
for i = n-1:-1:1
    x(1:i) = x(1:i) - A(i+1,1:i) .'* x(i+1);
end
end
```

Un esempio di utilizzo è contenuto nel file di testo `9_mialdl.txt`:

```
% si genera una matice quadrata casuale
A = randi([-8,8],4)

A =

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

% si generano i valori di una diagonale
d = randi([5,30],4,1)

d =

    21
    20
    10
    12
```

```
% si costruisce una matrice adeguata per la fattorizzazione LDL
A = tril(A,-1)+triu(A',1)+diag(d)

A =

    21    -1    -1    -3
    -1    20     5     5
    -1     5    10     1
    -3     5     1    12

% Si genera la soluzione, da confrontare dopo
x = randi([-8,8],4,1)

x =

     0
    -5
     6
    -5

% si calcolano i termini noti
b = A*x

b =

    14
   -95
    30
   -79

% si usa la funzione per calcolare la soluzione
mialdl(A,b)

ans =

    0.0000
   -5.0000
    6.0000
   -5.0000

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

A =

    -7     7    -4     0
    -4     4    -1    -5
     5     0     8     0
    -8     1     1     2
```

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

d =

    22
    15
    14
    30

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

A =

    22    -4     5    -8
    -4    15     0     1
     5     0    14     1
    -8     1     1    30

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

x =

    -8
     7
     7
     5

b = A*x

b =

  -209
   142
    63
   228

mialdl(A,b)

ans =

   -8.0000
    7.0000
    7.0000
    5.0000
```

10

La funzione è nel file `functions/miaqr.m`, mostrato di seguito insieme ad un esempio in cui viene applicato:

```
function [x,nr] = miaqr(A,b)
%
%   [x, nr] = miaqr(A,b)
%
%   Calcola la soluzione del sistema lineare sovradimensioanto Ax = b
%   nel senso dei minimi quadrati e restituisce la norma del
%   corrispondente vettore residuo
%
%   Input:
%       A:  matrice dei coefficienti del sistema
%       b:  vettore dei termini noti
%   Output:
%       x:  soluzione nel senso dei minimi quadrati
%       nr: norma del vettore resiudo
[m,n] = size(A);
if(n>=m), errror("Il sistema non è sovradimensionato"); end
if(m~=length(b)), error("Le dimensioni della matrice e del vettore " + ...
        "non sono compatibili");end
for i=1:n
    alfa = norm( A(i:m,i));
    if alfa==0,error("La matrice A non ha rango massimo");end
    if(A(i,i)>=0), alfa = -alfa; end
    v = A(i,i) - alfa;
    A(i,i) = alfa;
    A(i+1:m,i) = A(i+1:m,i)/v;
    beta = -v/alfa;
    A(i:m,i+1:n) = A(i:m,i+1:n)-(beta*[1;A(i+1:m,i)])*...
        ([1;A(i+1:m,i)]'*A(i:m,i+1:n));
end
for i=1:n
    v = [1;A(i+1:m,i)];
    beta = 2/(v'*v);
    b(i:end) = b(i:end)-(beta*(v'*b(i:end)))*v;
end
for i=n:-1:1
    b(i) = b(i)/A(i,i);
    b(1:i-1) = b(1:i-1)-A(1:i-1,i)*b(i);
end
x = b(1:n);
nr = norm(b(n+1:m));
return ;
end
```

```
>> A = randi([-20,20],7,4)

A =
```

```
    -6     6   -13    -1
    13    -2    -5    -3
   -20     2     5    -2
   -19    -8    11    -8
   -14    10   -17     0
     6   -13    18     0
    10     8    11    13

>> b = randi([-20,20],7,1)

b =

    12
     6
    -5
    13
     1
    -6
    18

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

x =

    0.5023
    2.5325
    1.1667
   -2.1687


nr =

   22.8572

>> A\b

ans =

    0.5023
    2.5325
    1.1667
   -2.1687

>> A = randi([-20,20],7,4)

A =

    15   -11    -8   -16
     2    14    17   -10
```

```
     5   -13    -3    -4
     4   -11   -13     4
   -12   -14    17   -10
    -8   -11    20     4
    -1    -3    -3     9

>> b = randi([-20,20],7,1)

b =

   -11
   -16
    -8
    -7
    -3
     0
   -17

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

x =

   -1.2544
    0.2774
   -0.6423
   -0.2978


nr =

   22.2472

>> A\b

ans =

   -1.2544
    0.2774
   -0.6423
   -0.2978

>>
```

11

Di seguito un esempio di applicazione di `mialu` per risolvere i sistemi generati da `linsis`:

```
[A1,A2,b1,b2]=linsis(5)

A1 =

    0.0659   -0.4423    0.2073   -0.5127    0.3531
    0.7016   -0.2493   -0.1158    0.1664   -0.0385
   -0.1391   -0.4272    0.4168    0.2575   -0.0030
    0.2598   -0.4140   -0.0020   -0.2632   -0.6674
    0.0654   -0.2921   -0.6037   -0.1323    0.5153


A2 =

   -0.2172   -0.0838    0.2868   -0.3463    0.3624
    0.3869    0.1493   -0.0275    0.3514   -0.0281
   -0.1833   -0.3713    0.4292    0.2834   -0.0016
   -0.0576   -0.0121    0.0871   -0.0767   -0.6569
   -0.1544   -0.0139   -0.5420   -0.0032    0.5225


b1 =

   -0.3287
    0.4645
    0.1050
   -1.0869
   -0.4475


b2 =

    0.0019
    0.8320
    0.1565
   -0.7163
   -0.1910

mialu(A1,b1)

ans =

    1.0000
    1.0000
    1.0000
    1.0000
```

```
    1.0000

mialu(A2,b2)

ans =

    1.0000
    1.0000
    1.0000
    1.0000
    1.0000
```

Il risltato sembra essere corretto, ma se si sottrae le soluzioni ad un vettore composto di soli 1, si può osservare l'errore nella risoluzione.
Nel sistema $A_1x = b_1$ l'errore è nell'ordine di 10^{-15} mentre nel secondo sistema $A_2x = b_2$ l'ordine di errore è di 10^{-6}.
L'errore molto maggiore nel secondo sistema è dovuto al mal condizionamento della matrice dei coefficienti.

```
mialu(A1,b1)-[1 1 1 1 1]'

ans =

   1.0e-15 *

         0
   -0.1110
         0
         0
    0.2220

mialu(A2,b2)-[1 1 1 1 1]'

ans =

   1.0e-06 *

    0.3523
   -0.4462
   -0.0989
   -0.2071
   -0.0116

>> cond(A1)

ans =

    2.5000

>> cond(A2)
```

```
ans =

   1.0000e+10

>> cond(b1)

ans =

     1

>> cond(b2)

ans =

     1

>>
```

12

Similemente a quanto si è ottenuto per l'esercizio precedente, si può osservare come i risultati ottenuti dalla funzione `mialdl` siano accurati con un oridne di grandezza dell'errore di 10^{-15} per il primo sistema e 10^{-6} per il secondo. Anche in questo caso la differenza è dovuta alla differenza del condizionamento delle due matrici A_1 e A_2.

```
[A1,A2,b1,b2] = linsis(5,1)

A1 =

    0.7625    0.0003    0.1094    0.0730    0.1782
    0.0003    0.7442    0.0084    0.1772    0.1085
    0.1094    0.0084    0.6419    0.0332   -0.1291
    0.0730    0.1772    0.0332    0.8375   -0.1198
    0.1782    0.1085   -0.1291   -0.1198    0.8139


A2 =

    0.5197   -0.2696    0.0716   -0.1992   -0.0102
   -0.2696    0.4441   -0.0337   -0.1254   -0.1010
    0.0716   -0.0337    0.6360   -0.0092   -0.1585
   -0.1992   -0.1254   -0.0092    0.5324   -0.3310
   -0.0102   -0.1010   -0.1585   -0.3310    0.6677


b1 =

    1.1234
    1.0385
```

```
    0.6638
    1.0011
    0.8517


b2 =

    0.1124
   -0.0855
    0.5063
   -0.1323
    0.0670

>> x1 = mialdl(A1,b1)

x1 =

    1.0000
    1.0000
    1.0000
    1.0000
    1.0000

>> x2 = mialdl(A2,b2)

x2 =

    1.0000
    1.0000
    1.0000
    1.0000
    1.0000

>> x1 - [1 1 1 1 1]'

ans =

   1.0e-15 *

         0
   -0.4441
   -0.1110
   -0.1110
         0

>> x2 - [1 1 1 1 1]'

ans =

   1.0e-06 *
```

```
    0.3138
    0.3488
    0.0489
    0.3518
    0.2435

>>
```

13

Di seguito si mostra come sono stati assegnati i valori richiesti, le soluzioni trovate con la funzione `miaqr` sono confrontate con il risultato dell'operatore \.

```
>> A = [ 1 3 2; 3 5 4; 5 7 6; 3 6 4; 1 4 2 ];
>> b = [ 15 28 41 33 22 ]';
>> D = diag(1:5);
>> D1 = diag(pi*[1 1 1 1 1])

D1 =

    3.1416         0         0         0         0
         0    3.1416         0         0         0
         0         0    3.1416         0         0
         0         0         0    3.1416         0
         0         0         0         0    3.1416

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

x =

    3.0000
    5.8000
   -2.5000


nr =

    1.2649

>> A\b

ans =

    3.0000
    5.8000
   -2.5000

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

```
x =

   -0.6026
    4.7017
    1.7584


nr =

    3.7352

>> (D*A)\(D*b)

ans =

   -0.6026
    4.7017
    1.7584

>> [x,nr] = miaqr(D1*A,D1*b)

x =

    3.0000
    5.8000
   -2.5000


nr =

    3.9738

>> (D1*A)\(D1*b)

ans =

    3.0000
    5.8000
   -2.5000
```

Si osserva come le soluzioni siano coerenti, ma la norma del vettore residuo aumenta, negli ultimi due sistemi è quasi il triplo che nel primo.

14

La funzione è contenuta nel file `newton.m` nella cartella `14`.

```
function [x,nit] = newton(fun,jacobian,x0,tol,maxit)
%
```

```
%   [x,nit] = newton(fun,jacobian,x0,tol,maxit)
%
%   Utilizza il metodo di Newton per risolvere sistemi di equazioni
%   nonlineari
%
%   Input:
%       fun: vettore delle funzioni del sistema
%       jacobian: matrice jacobiana di fun
%       x0: approssimazione iniziale
%       tol: tolleranza aspettata
%       maxit: numero massimo di iterazioni ammesso
%   Output:
%       x: approssimazione della funzione
%       nit: numero delle iterazioni del metodo
if(nargin<4), tol = eps;    end
if(nargin<5), maxit = 1e3;  end
if(tol<=0),   error("La tolleranza deve essere positiva");          end
if(maxit<=0), error("Il numero di iterazioni deve essere positivo");end
x = x0;
nit = 0;
while(nit<maxit&&norm(x-x0)<=tol*(1+norm(x0)))
    x0 = x;
    fx0 = feval(fun,x0);
    jx0 = feval(jacobian,x0);
    x = x0 - fx0/jx0;
end
if(nit == maxit)
    disp("Il numero di iterazioni specificato non ha permesso " + ...
        "di raggiungere     la tolleranza desiderata");
end
return ;
end
```

15 •

16

```
function l = lagrange(x,y,xq)
% l = lagrange(x,y,xq)
%
%   Implementa in modo vettoriale la forma di lagrange del polinomio
%   interpolante di una funzione
%
%   Input:
%       x:  ascisse di interpolazione
%       y:  valori della funzione sulle ascisse di interpolazione
%       xq: punti in cui calcolare il polinomio
%   Output:
%       l:  polinomio di lagrange calcolato
```

```
n = length(x);
if n ~= length(y)
    error("il numero di punti sulle ascisse x non è coerente con" + ...
        " il numero di quelli sulle ordinate");
end
if n ~= length(unique(x))
    error("ad una stessa ascissa non possono corrispondere più punti")
end
l = zeros(size(xq));
for k = 1:n
    Lkn = ones(size(xq));
    for j = 1:n
        if k ~= j
            Lkn = Lkn .* ((xq - x(j))/(x(k) - x(j)));
        end
    end
    l = l + y(k)*Lkn;
end
return;
end
```

17

```
function l = newton(x,y,xq)
%   Implementa in modo vettoriale, la forma di Newton del polinomio
%   interpolante una funzione
%
%   Input:
%       x:  vettore contenente le ascisse di interpolazione
%       y:  valori assunti dalla funzione sulle ascisse di
%           interpolazione
%       xq: punti su cui si vuole calcolare la funzione
%   Output:
%       l:  approsimazione dei valori della funzione secondo
%           il polinomio interpolante

n = length(x);
if n ~= length(y)
    error("il numero di punti sulle ascisse x non è coerente con il numero" + ...
        " di quelli sulle ordinate");
end
if n ~= length(unique(x))
    error("ad una stessa ascissa non possono corrispondere più punti")
end
f = y; %differenze divise
for k = 1:n-1
    for r = n:-1:k+1
        f(r) = (f(r) - f(r-1)) ./ (x(r) - x(r-k));
```

```
    end
end
l = ones(size(xq)) * f(n);
for k = n-1:-1:1
    l = l .* (xq - x(k)) + f(k);
end
end
```

18

```
function yy = hermite( xi, fi, f1i, xx )
%   Implementa in modo vettoriale il polinomio interpolante di Hermite
%
%   Input:
%       xi:     vettore delle ascisse di interpolazione
%       fi:     valori assunti dalla funzione sulle ascisse
%               di interpolazione
%       fii:    valori assunti dalla derivata della funzione sulle ascisse
%               di interpolazione
%       xx:     vettore di ascisse su cui si vuole calcolare il polinomio
%   Output:
%       yy:     valori assunti dal polinomio sui punti specificati

n = length(xi);
if n ~= length(fi)
    error("il numero di punti sulle ascisse xi " + ...
        "non è coerente con il numero di quelli sulle ordinate fi");
end
if n ~= length(f1i)
    error("il numero di punti sulle ascisse xi non è coerente" + ...
        " con il numero di quelli sulle ordinate fi"); % ---
end
if n ~= length(unique(xi))
    error("le ascisse di interpolazione non sono tutte distinte");
end
x = repelem(xi,2);
%differenze divise
f(1:2:2*n-1)=f1;
f(2:2:2*n) = f1i;
% algortimo 4.2 libro
n = length(f)/2-1;
for i = (2*n-1):-2:3
    f(i)= (f(i)-f(i-2))/(x(i)-x(i-1));
end
for j = 2:2*n-1
    for i = (2*n):-1:j+1
        f(i) = (f(i)-f(i-1))/(x(i)- x(i-j));
    end
end
```

```
%algoritmo di horner
n = length(f)-1;
yy = f(n+1)*ones(size(xx));
for i = n:-1:1
    yy = yy .* (xx-x(i))+f(i);
end
end
```

19

```
function x = chebyshev(n,a,b)
%
%   Genera n+1 coordinate di Chebyschev nell'intervallo [a,b]
%
%   Input:
%       n:  numero di coordinate da generare (n+1)
%       a:  estremo inferiore dell'intervallo
%       b:  estremo superiore dell'intervallpo
%   Output:
%       x:  vettore contenente le coordinate
if n <= 0
    error("il grado del polinomio deve essere maggiore di zero");
end
if a >= b
    error("l'estremo inferiore dell' intervallo non può essere " + ...
        "minore o coincidente con quello maggiore");
end
x = cos( (2*(n:-1:0)+1)*pi ./ (2*(n+1)) );
x = x * (b-a)/2 + (a+b)/2;
end
```