



Università degli Studi di Napoli - Federico II

Facoltà di
Scienze Matematiche, Fisiche e Naturali

Corso di Laurea in Scienze Informatiche

Calcolo Numerico A.A. 2004 / 2005

Prof.ssa Eleonora Messina

METODO DI ELIMINAZIONE DI GAUSS CON PIVOTING PARZIALE

Realizzato da

Giovanni Di Cecca & Virginia Bellino

50 / 887

408 / 466



<http://www.dicecca.net>

© 2005

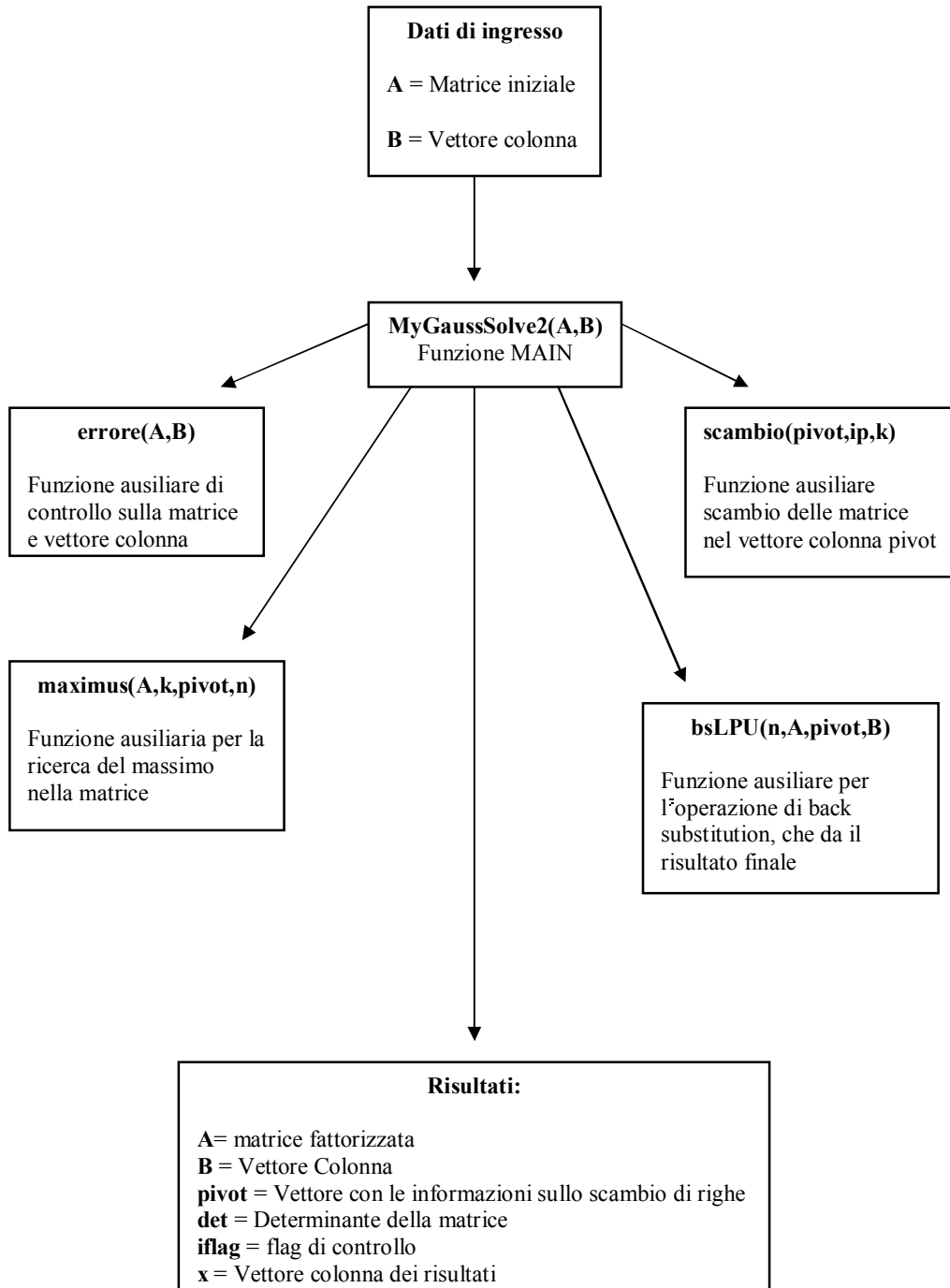
Giovanni Di Cecca & Virginia Bellino – <http://www.dicecca.net>

Indice

- **Grafico**
- **Sezione 1 – Documentazione esterna**
- **Sezione 2 – Codice MATLAB**
- **Sezione 3 - Testing**

Progetto Metodo di Gauss con Pivoting Parziale

Schema grafico



SEZIONE 1

DOCUMENTAZIONE

DOCUMENTAZIONE
ESTERNA

- ◆ **Scopo:** la funzione esegue la risoluzione di un sistema di equazioni lineari del tipo $\mathbf{Ax} = \mathbf{b}$ utilizzando il metodo di eliminazione di Gauss con pivoting parziale.

- ◆ **Specifiche:**

function [A, B, pivot, det, iflag, x] = myGaussSolve2 (A, B)

- ◆ **Descrizione:**

Dopo aver eseguito un controllo sulla dimensione dei dati di input, l'algoritmo esegue la fattorizzazione LU della matrice dei coefficienti, trasformando la matrice iniziale in una matrice a gradini equivalente, da cui vengono ricavate le soluzioni del sistema, applicando il metodo della back substitution (o sostituzione all'indietro).

- ◆ **Riferimenti bibliografici:**

James F. Epperson
INTRODUZIONE ALL'ANALISI NUMERICA
McGraw-Hill

- ◆ **Lista dei parametri:**

Parametri di input:

A: matrice dei coefficienti del sistema lineare. E' di dimensione n*n.
B: vettore dei termini noti. E' di dimensione n.

Parametri di output:

A: matrice dei coefficienti modificata.
B: vettore dei termini noti modificati.
pivot: vettore che registra gli scambi effettuati durante il pivoting.
det: determinante della matrice.
iflag: indicatore di errori.
x: vettore contenente le soluzioni finali del sistema.

- ◆ **Indicatori di errore:**

iflag: indica se durante la procedura si verificano degli errori, bloccando in tal caso l'esecuzione.. Può assumere i seguenti valori:

0 : significa che la procedura è stata eseguita senza problemi.
1 : la matrice dei coefficienti non è quadrata

2 : il vettore dei termini noti non ha la stessa dimensione della matrice dei coefficienti
 3 : il numero di colonne di termini noti inseriti in input è > 1.
 -1: matrice singolare.

♦ Funzioni ausiliarie:

function iflag = errore (A, B)

la routine esegue un controllo sulla dimensione della matrice e del vettore dei termini noti inseriti in input. Se ci sono errori, l'indicatore iflag assume valore 1 e l'esecuzione termina.

function [max, ip] = maximus (A, k, pivot, n)

la routine cerca l'elemento massimo in valore assoluto sulla colonna k e riga da k+1 ad n, salvandone il valore e l'indice di riga.

function pivot = scambio (pivot, ip, k)

la routine esegue lo scambio degli indici nel vettore pivot.

function [x] = bsLPU (n, A, pivot, B)

la routine esegue la back-substitution sul sistema a gradini ottenuto applicando il metodo di gauss, consentendo di ricavare le soluzioni finali del sistema $Ax = b$.

♦ Complessità computazionale:

la complessità totale della funzione è $T(n) = \frac{2}{3} n^3$

- ♦ Accuratezza fornita: poiché il metodo di gauss appartiene alla categoria dei metodi risolutivi diretti, la soluzione ottenuta risulta esatta a meno di un errore di round off legato alla precisione della macchina.

♦ Esempio d'uso:

Esempio 1

EDU>> a=[1 2 -1

1 3 1

2 4 -1]

a =

1 2 -1

1 3 1

2 4 -1

EDU>> b=[1

5

6]

b =

1
5
6

EDU>> [A,B,pivot,det,iflag,x]=myGaussSolve2(a,b)

A =

0.5000 0 -0.5000
0.5000 1.0000 1.5000
2.0000 4.0000 -1.0000

B =

-2
2
6

pivot =

3
2
1

det =

-2

iflag =

0

x =

13
-4
4

@@@

Esempio 2

```
EDU>> a=[1,1,0;2,1,3;1,2,-3]
```

a =

```
1 1 0
2 1 3
1 2 -3
```

```
EDU>> b=[4;3;5]
```

b =

```
4
3
5
```

```
EDU>> [A,B,pivot,det,iflag,x]=myGaussSolve2(a,b)
```

Warning: One or more output arguments not assigned during call to 'mygaussolve2'.

A =

```
0.5000 0.3333 0
2.0000 1.0000 3.0000
0.5000 1.5000 -4.5000
```

B =

```
1.3333
3.0000
3.5000
```

pivot =

```
2
3
1
```

det =

```
0
```

iflag =

```
-1
```

SEZIONE 2

CODICE MATLAB


```
% Progetto Metodo di Gauss con Pivoting Parziale
%
%           Programma elaborato da
%
%   Giovanni DI CECCA & Virginia BELLINO
%           50 / 887           408 / 466
%
%           http://www.dicecca.net

% Funzione Main

% SCOPO: la funzione esegue la risoluzione di un sistema
% di equazioni lineari del tipo  $Ax = b$  utilizzando
% il metodo di eliminazione di Gauss con pivoting parziale

% Parametri di input:
% A: matrice dei coefficienti del sistema lineare. E' di dimensione  $n \times n$ .
% B: vettore dei termini noti. E' di dimensione  $n$ .

% Parametri di output:
% A: matrice dei coefficienti modificata.
% B: vettore dei termini noti modificati.
% pivot: vettore che registra gli scambi effettuati durante il pivoting.
% det: determinante della matrice.
% iflag: indicatore di errori.
% x: vettore contenente le soluzioni finali del sistema.

function [A,B,pivot,det,iflag,x]=myGaussSolve2(A,B)

% Inizializza il flag di errore
% Controllo sulla dimensione della matrice A e del vettore B.
% Se non ci sono errori, si può continuare

iflag = errore (A,B);

% Dimensione della matrice
s = size(A);

% Considera il valore delle righe
n=s(1);

% Crea il vettore colonna pivot
for i=1:n

    pivot(i,1)=[i];

end
```

```
% Inizializza il determinante
det = 1;

% k indica il passo dell'algoritmo
for k=1:n-1

    % Cerca il massimo mediante la funzione apposita salvandone valore e i
    ndice
    % di riga
    [max,ip]=maximus(A,k,pivot,n);

    if (max == 0) % NOTAl: significa che sulla diagonale c'è uno zero...
        det = 0; %...per cui il determiniante di una matrice triangolare,
                % pari al prodotto degli elementi della diagonale, è
                % necessariamente uguale a zero

        iflag = -1; % segnala il verificarsi di una anomalia durante
                % l'applicazione dell'algoritmo:il sistema non
                % è compatibile

        break % Blocca l'esecuzione del programma
    end % Fine dell'if

    % Scambia effettivamente gli indici nel vettore pivot
    % solo se k è diverso da ip
    if (ip ~= k)

        % Esegui la funzione di scambio
        pivot=scambio(pivot,ip,k);

        det = -det; % ad ogni scambio di riga il determinante cambia di segno
    end

    % Inizio della decomposizione LU della matrice dei coefficienti
    for i=k+1:n

        % Calcola e memorizza il moltiplicatore nella posizione (i,k)
        A(pivot(i),k) = A(pivot(i),k)/A(pivot(k),k);

        % Modifica gli elementi della riga i-esima in base al valore del moltiplicatore
        % memorizzato in A(i,k)
        for j=k+1:n
```

```
A(pivot(i),j)=A(pivot(i),j)-A(pivot(i),k)*A(pivot(k),j);

end % Fine del ciclo j

% Modifica il valore dei termini noti
B(pivot(i))=B(pivot(i))-A(pivot(i),k)*B(pivot(k));

end % fine del ciclo i

% Calcola il determinante moltiplicando gli elementi della diagonale
% principale della matrice modificata
det = det * A(pivot(k),k);

end; % del ciclo di k

if (A(pivot(n),n) == 0) % Vedi NOTA1

    det = 0;
    iflag = -1;
    break
end

% Risoluzione del sistema con back Substitution
[x]=bsLPU(n,A,pivot,B);
```

```
% Progetto Metodo di Gauss con Pivoting Parziale
%
%           Programma elaborato da
%
%   Giovanni DI CECCA & Virginia BELLINO
%           50 / 887           408 / 466
%
%           http://www.dicecca.net

% Routine dei controlli sulla matrice A e vettore colonna B

% SCOPO: La funzione fa il controllo sulla matrice A e vettore colonna B
%
% Parametri di Input: Matrice A, Vettore colonna B
%
% Parametri di Output: iflag = Flag di controllo sugli errori
%
%                       0 = OK
%                       1 = Matrice non quadrata
%                       2 = Il vettore colonna dei termini noti non è
uguale alla matrice
%
%                       3 = B è un vettore colonna non una matrice

function iflag=errore(A,B)

% Associa alla variabile matrix la lunghezza e larghezza della matrice A
matrix=size(A);

% Controllo sulla quadratura della matrice
if matrix(1)==matrix(2)

    iflag = 0; % Flag di controllo 0 = OK

else

    iflag = 1; % Matrice non quadrata
    break
end

% calcola la dimensione del vettore colonna
vectorcln=size(B);

% Controllo sul vettore colonna b, il numero delle righe di b deve essere
% uguale a quello della matrice A
if matrix(1)==vectorcln(1)

    iflag = 0; % Flag di controllo 0 = Ok
else
```

```
        iflag = 2; % Il vettore colonna dei termini noti non ha dimensione uguale alla matrice
        break
    end

    % Controllo sul vettore colonna b, il numero delle colonne deve essere 1
    if vectorcln(2)==1
        iflag = 0; % Flag di controllo 0 = Ok
    else

        iflag = 3; % B è un vettore colonna non una matrice
        break
    end
```

```
% Progetto Metodo di Gauss con Pivoting Parziale
%
%           Programma elaborato da
%
%   Giovanni DI CECCA & Virginia BELLINO
%           50 / 887           408 / 466
%
%           http://www.dicecca.net

% funzione maximus

% SCOPO: la funzione ha lo scopo di calcolare il massimo della colonna k
% elezionata, salvandone valore e indice
%
% Parametri di Input: A = matrice
%                   k = Passo dell'algoritmo
%                   pivot = Vettore colonna che contiene gli scambi della
% a matrice
%                   n = Dimensione della matrice
%
% Parametri di Output: max = massimo trovato
%                   ip = Indice del massimo trovato

% Funzione che trova l'elemento massimo in val. assoluto sulla colonna k
% riga da k+1 ad n
function [max,ip]=maximus(A,k,pivot,n)

% Sulla colonna selezionata si ricerca il massimo in valore assoluto, salvandone
% valore e indice
max = abs(A(pivot(k),k));

ip = k;    % ip sta per indice del massimo

for p=k+1:n

    if (abs(A(pivot(p),k))>max)
        % memorizza temporaneamente il valore del massimo
        % della colonna k al di sotto della diagonale principale e del suo
        % indice

        max = abs(A(pivot(p),k));

        ip = p;
    end
end

end
```

```
% Progetto Metodo di Gauss con Pivoting Parziale
%
%           Programma elaborato da
%
%   Giovanni DI CECCA & Virginia BELLINO
%       50 / 887           408 / 466
%
%       http://www.dicecca.net

% Funzione scambio

% SCOPO: la funzione ha lo scopo di scambiare i valori degli indici nel vettore colonna pivot
%
% Parametri di Input: pivot = Vettore colonna con gli indici della matrice
%                   ip = indice del miglior pivot
%                   k = passo dell'algoritmo
%
% Parametri di Output: pivot = Vettore colonna che contiene gli scambi della matrice

% Funzione che esegue lo scambio degli indici, delle righe
% della matrice A, contenuti nel vettore p
function pivot=scambio(pivot,ip,k)

temp=pivot(k);

pivot(k)=pivot(ip);

pivot(ip)=temp;
```

```
% Progetto Metodo di Gauss con Pivoting Parziale
%
%         Programma elaborato da
%
%         Giovanni DI CECCA & Virginia BELLINO
%         50 / 887           408 / 466
%
%         http://www.dicecca.net

% Funzione di Back Substitution

% SCOPO: la funzione ha lo scopo di calcolare i valori delle incognite mediante sostituzione all'indietro
%
% Parametri di Input: n = righe della matrice
%                   A = matrice triangolarizzata
%                   pivot = Vettore colonna che contiene gli scambi eseguiti sulla matrice
%                   B = Vettore colonna dei termini noti modificati
%
% Parametri di Output: x = Vettore colonna con i risultati

function [x]=bsLPU(n,A,pivot,B)

% Calcolo dell'ultima riga della matrice
x(n,1)=B(pivot(n),1)/A(pivot(n),n);

% Ciclo della funzione di sostituzione all'indietro partendo dalla penultima riga fino alla prima
for i=n-1:-1:1

    % Inizializzazione della variabile somma
    somma = 0;

    % Calcolo delle incognite per sostituzione
    for k=(i+1):n
        somma = somma + A(pivot(i),k) * x(k,1);
    end

    % Calcolo del valore delle incognite
    x(i,1)=(B(pivot(i),1)-somma)/A(pivot(i),i);

end
```

```
% Progetto Metodo di Gauss con Pivoting Parziale
%
%           Programma elaborato da
%
%   Giovanni DI CECCA & Virginia BELLINO
%           50 / 887           408 / 466
%
%           http://www.dicecca.net

% Funzione di Test

% Lo scopo di questa funzione è quello di andare ad eseguire
% tutti i test previsti nell'esercizio, in modo automatico
% ed in successione.
%
% Per evitare possibili errori nell'uso delle matrici,
% ad ogni test viene eseguito un reset della memoria,
% sia video (il clc) sia fisica (il clear)

clc % pulisci schermo

clear % Pulisci memoria

disp(' Progetto Metodo di Gauss con Pivoting Parziale')
disp(' ')
disp('           Programma elaborato da')
disp(' ')
disp('   Giovanni DI CECCA & Virginia BELLINO')
disp('           50 / 887           408 / 466')
disp(' ')
disp('           http://www.dicecca.net')
disp(' ')
disp(' ')

disp('test - 1')

a=[2 0 -2 0
    9 10 -4 -1
    3 4 -2 1
    1 -2 -2 3]

b=[0
   -5
   -3
    1]

[A,B,pivot,det,iflag,x]=myGaussSolve2(a,b)
```

```
disp('Premere un tasto per continuare')  
pause
```

```
%-----
```

```
clc % pulisci schermo
```

```
clear % Pulisci memoria
```

```
disp('test - 2')
```

```
a=[3 10 9  
    8 4 -10  
   -18 8 48]
```

```
b=[1  
    0  
   -2]
```

```
[A,B,pivot,det,iflag,x]=myGaussSolve2(a,b)
```

```
disp('Premere un tasto per continuare')  
pause
```

```
%-----
```

```
clc % pulisci schermo
```

```
clear % Pulisci memoria
```

```
disp('test - 3')
```

```
a=[10 1 2 3  
    1 2 0 0  
    2 0 6 1  
    3 0 1 5]
```

```
b=[17  
    -6  
    12  
    31]
```

```
[A,B,pivot,det,iflag,x]=myGaussSolve2(a,b)
```

```
disp('Premere un tasto per continuare')  
pause
```

```
%-----
```

```
clc % pulisci schermo
```

```
clear % Pulisci memoria
```

```
disp('test - 4')
```

```
a=[7 -2 3  
    1 11 3  
    3 12 15]
```

```
b=[8  
    35  
    42]
```

```
[A,B,pivot,det,iflag,x]=myGaussSolve2(a,b)
```

```
disp('Premere un tasto per continuare')  
pause
```

```
%-----
```

```
clc % pulisci schermo
```

```
clear % Pulisci memoria
```

```
disp('test - 5.1')
```

```
q=5;
```

```
% Ciclo di creazione di A  
for t=1: q  
    for z=1: q  
        a(t,z)=(t)^(z-1);  
    end  
end
```

```
% Ciclo di creazione di B  
for t=1: q  
    for z=1: q  
        b(t,1)=a(t,q)/2;  
    end  
end
```

```
[A,B,pivot,det,iflag,x]=myGaussSolve2(a,b)
```

```
disp('Premere un tasto per continuare')  
pause
```

%-----

```
clc % pulisci schermo
```

```
clear % Pulisci memoria
```

```
disp('test - 5.2')
```

```
q=10;
```

```
% Ciclo di creazione di A
```

```
for t=1: q  
    for z=1: q  
        a(t,z)=(t)^(z-1);  
    end  
end
```

```
% Ciclo di creazione di B
```

```
for t=1: q  
    for z=1: q  
        b(t,1)=a(t,q)/2;  
    end  
end
```

```
[A,B,pivot,det,iflag,x]=myGaussSolve2(a,b)
```

```
disp('Premere un tasto per continuare')  
pause
```

%-----

```
clc % pulisci schermo
```

```
clear % Pulisci memoria
```

```
disp('test - 5.3')
```

```
q=20;
```

```
% Ciclo di creazione di A
```

```
for t=1: q  
    for z=1: q  
        a(t,z)=(t)^(z-1);  
    end  
end
```

```
% Ciclo di creazione di B
```

```
for t=1: q
```

```
    for z=1: q
        b(t,1)=a(t,q)/2;
    end
end

[A,B,pivot,det,iflag,x]=myGaussSolve2(a,b)

disp('Premere un tasto per continuare')
pause

%-----

clc % pulisci schermo

clear % Pulisci memoria

disp('test - 5.4')

q=50;

% Ciclo di creazione di A
for t=1: q
    for z=1: q
        a(t,z)=(t)^(z-1);
    end
end

% Ciclo di creazione di B
for t=1: q
    for z=1: q
        b(t,1)=a(t,q)/2;
    end
end

[A,B,pivot,det,iflag,x]=myGaussSolve2(a,b)

disp('Premere un tasto per continuare')
pause

%-----

clc % pulisci schermo

clear % Pulisci memoria

disp('test - 6')

a=[10E-15 3
   2 3]
```

```
b=[3  
    5]
```

```
[A,B,pivot,det,iflag,x]=myGaussSolve2(a,b)
```

```
disp('Premere un tasto per continuare')  
pause
```

```
%-----
```

```
clc % pulisci schermo
```

```
clear % Pulisci memoria
```

```
disp('test - 7')
```

```
a=[1 1  
    1 1.001]
```

```
b=[1  
    0]
```

```
[A,B,pivot,det,iflag,x]=myGaussSolve2(a,b)
```

```
disp('Premere un tasto per continuare')  
pause
```

```
%-----
```

```
clc
```

```
clear
```

```
disp('test - 7.1')
```

```
a=[1 (1+4/1000)  
    1 1.001]
```

```
b=[1  
    0]
```

```
[A,B,pivot,det,iflag,x]=myGaussSolve2(a,b)
```

SEZIONE 3

TESTING

Progetto Metodo di Gauss con Pivoting Parziale

Programma elaborato da

Giovanni DI CECCA & Virginia BELLINO
50 / 887 408 / 466

<http://www.dicecca.net>

test - 1

a =

2	0	-2	0
9	10	-4	-1
3	4	-2	1
1	-2	-2	3

b =

0
-5
-3
1

A =

0.2222	0.7143	0	-2.0000
9.0000	10.0000	-4.0000	-1.0000
0.3333	-0.2143	-1.0000	2.0000
0.1111	-3.1111	-1.5556	3.1111

B =

0
-5.0000
-1.0000
1.5556

pivot =

2
4
3
1

det =

28.0000

iflag =

0

x =

```
    1.0000  
   -1.0000  
    1.0000  
         0
```

Premere un tasto per continuare

test - 2

a =

3	10	9
8	4	-10
-18	8	48

b =

1
0
-2

Warning: One or more output arguments not assigned during call to 'mygaussssolve2'.

> In C:\Documents and Settings\Giovanni\Desktop\Calcolo progetto - 1\progetto - MATLAB\test.m at line 69

A =

-0.1667	11.3333	17.0000
-0.4444	0.6667	0
-18.0000	8.0000	48.0000

B =

0.6667
-1.3333
-2.0000

pivot =

3
1
2

det =

0

iflag =

-1

Premere un tasto per continuare

test - 3

a =

10	1	2	3
1	2	0	0
2	0	6	1
3	0	1	5

b =

17
-6
12
31

A =

10.0000	1.0000	2.0000	3.0000
0.1000	1.9000	-0.2000	-0.3000
0.2000	-0.1053	5.5789	0.3684
0.3000	-0.1579	0.0660	4.0283

B =

17.0000
-7.7000
7.7895
24.1698

pivot =

1
2
3
4

det =

106.0000

iflag =

0

x =

0
-3.0000
1.0000
6.0000

Premere un tasto per continuare

test - 4

a =

7	-2	3
1	11	3
3	12	15

b =

8
35
42

A =

7.0000	-2.0000	3.0000
0.1429	0.8778	-9.4667
0.4286	12.8571	13.7143

B =

8.0000
0
38.5714

pivot =

1
3
2

det =

-90

iflag =

0

x =

2.0000
3.0000
0

Premere un tasto per continuare

test - 5.1

A =

1.0000	1.0000	1.0000	1.0000	1.0000
1.0000	0.2500	0.7500	-1.0000	-6.0000
1.0000	0.5000	-4.0000	-36.0000	-232.0000
1.0000	0.7500	0.7500	-3.0000	-39.0000
1.0000	4.0000	24.0000	124.0000	624.0000

B =

0.5000
-3.0000
-116.0000
-19.5000
312.0000

pivot =

1
5
3
4
2

det =

-48

iflag =

0

x =

0
0
0
0
0.5000

Premere un tasto per continuare

test 5.2

pivot =

1
10
5
8
2
9
3
6
4
7

det =

-4.2475e+017

iflag =

0

x =

0
0
0
0
0
0
0
0
0
0
0
0.5000

Premere un tasto per continuare

test 5.3

pivot =

1
20
10
16
4
18
7
2
13
19
3
14
6
17
9
5

12
15
8
11

det =

-4.0411e+125

iflag =

0

x =

0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0.5000

Premere un tasto per continuare

test 5.4

pivot =

1
50
25
39
9
46
16
4
33
49
2
21
43
12
30
48

0.5000

Premere un tasto per continuare

COMMENTO TEST N° 5

Tutti i sistemi implementati dal test hanno come matrice dei coefficienti la matrice di Vandermonde che è soggetta a malcondizionamento. Per questo motivo, tali sistemi risultano essere malcondizionati, poiché dati e soluzione subiscono perturbazioni che non sono dello stesso ordine. Infatti, al crescere della dimensione della matrice dei coefficienti, la soluzione del sistema non subisce variazioni.

test - 6

a =

0.0000	3.0000
2.0000	3.0000

b =

3
5

A =

0.0000	3.0000
2.0000	3.0000

B =

3.0000
5.0000

pivot =

2
1

det =

-2

iflag =

0

x =

1.0000
1.0000

Premere un tasto per continuare

COMMENTO TEST N° 6

L'utilizzo della tecnica di pivoting rende l'algoritmo meno suscettibile agli errori di arrotondamento legati alla precisione della macchina.

Eseguendo infatti un confronto dei risultati, è possibile osservare che, se usiamo la tecnica di pivoting, le soluzioni ottenute sono; in caso contrario le soluzioni sono , , , ..
Come si può notare, nel primo caso otteniamo due valori precisi, nel secondo caso invece no.

Con pivoting

x =

1.0000

1.0000

senza pivoting

x =

0.9770

1.0000

test - 7

a =

1.0000	1.0000
1.0000	1.0010

b =

1
0

A =

1.0000	1.0000
1.0000	0.0010

B =

1
-1

pivot =

1
2

det =

1

iflag =

0

x =

1.0e+003 *
1.0010
-1.0000

Premere un tasto per continuare

COMMENTO TEST N° 7

Confrontando i due risultati, si evince che la matrice dei coefficienti risulta essere malcondizionata. Infatti, una lieve perturbazione sui dati di input porta alla generazione di una soluzione del sistema molto diversa da quella generata in precedenza.



Università degli Studi di Napoli - Federico II

Facoltà di
Scienze Matematiche, Fisiche e Naturali
Corso di Laurea in Scienze Informatiche

Calcolo Numerico A.A. 2004 / 2005

Prof.ssa Eleonora Messina

**Progetto
Metodo di Simpson**

Realizzato da

Giovanni Di Cecca & Virginia Bellino

50 / 887

408 / 466

Gruppo № 35

2 *Progetto Metodo di Simpson*

Indice

Simpson a schema fisso

- SEZIONE 1: DOCUMENTAZIONE ESTERNA
- SEZIONE 2: CODICE MATLAB
- SEZIONE 3: TESTING

Simpson a schema adattivo

- SEZIONE 1: DOCUMENTAZIONE ESTERNA
- SEZIONE 2: CODICE MATLAB
- SEZIONE 3: TESTING

Osservazioni

Simpson a schema fisso

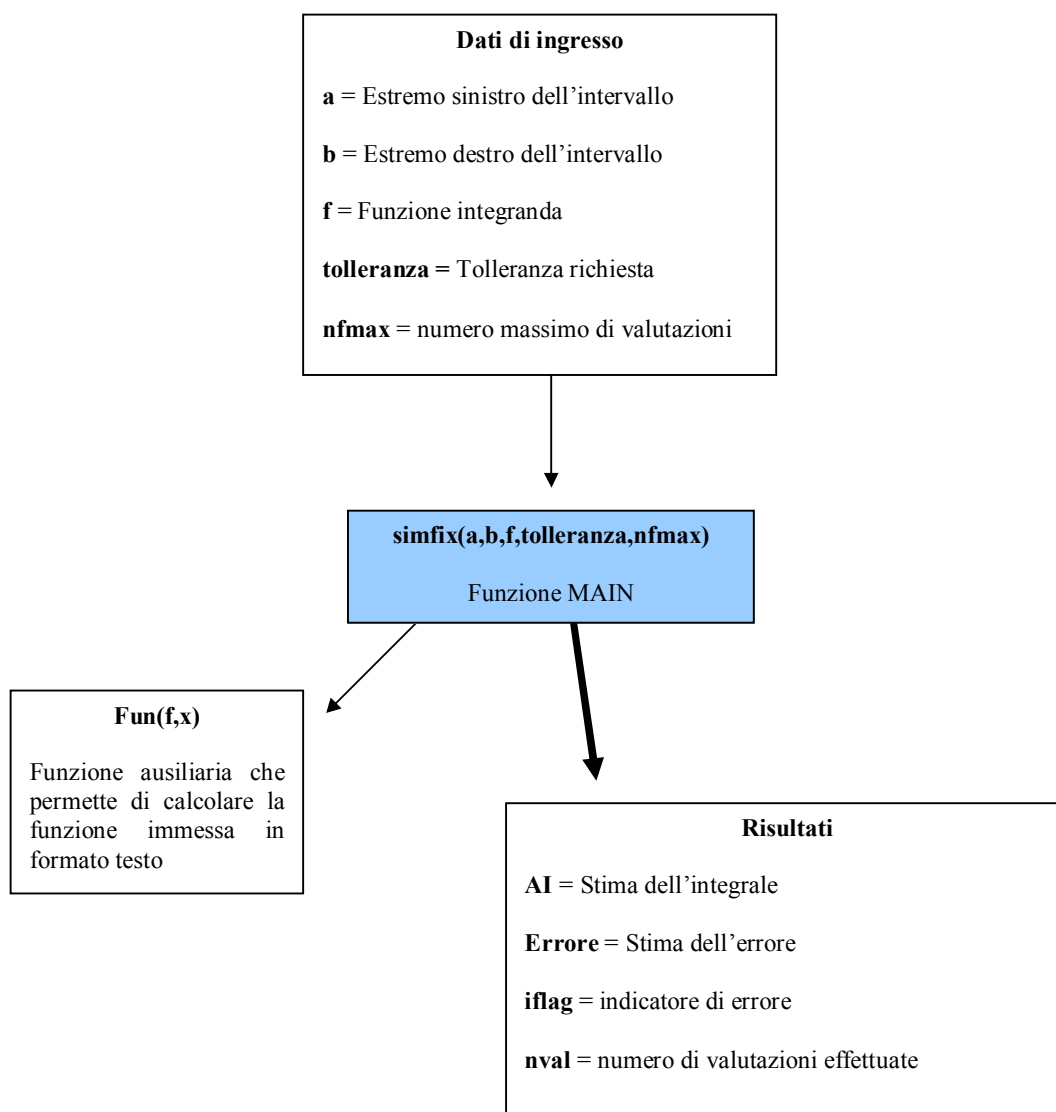
Sezione 1

Documentazione esterna

Progetto Metodo di Simpson a schema fisso

$$\int_a^b f(x) dx$$

Schema grafico



- ♦ **Scopo:** calcolare il valore numerico approssimato di un integrale definito del tipo

$$\int_a^b f(x) dx$$

utilizzando la strategia basata sulla *formula di quadratura composta di Simpson a schema fisso*.

- ♦ **Specifiche:**

[AI,Errore,iflag,nval]=simfix(a,b,f,tolleranza,nfmax)

- ♦ **Descrizione:**

l'algoritmo esegue i seguenti passi:

1. Inizializzazione delle variabili.
2. Valutazione della funzione negli estremi a et b, e salvataggio di tali valori che verranno poi riutilizzati in seguito.
(*nota: tutte le valutazioni di funzione vengono eseguite richiamando il modulo ausiliario Fun*)
3. Suddivisione dell'intervallo di integrazione (a,b) in due sottointervalli e calcolo della prima stima dell'integrale usando la formula di Simpson semplice su 3 punti.
4. Inizio del ciclo while che da luogo al procedimento di Simpson a schema fisso. Tale ciclo termina o quando si raggiunge la tolleranza richiesta, oppure quando si raggiunge il numero massimo di valutazioni di funzione consentito.
5. Quando il ciclo termina, vi è un controllo per stabilire quale delle due suddette condizioni si è verificata, in modo da restituire in output il corrispettivo valore dell'indicatore iflag.
6. Infine, vi sono le istruzioni per eseguire il plot della funzione.

♦ **Riferimenti bibliografici:**

- James F. Epperson
INTRODUZIONE ALL'ANALISI NUMERICA
McGraw-Hill
- Prof. Eleonora Messina
Appunti del corso di calcolo numerico
A. A. 2004/2005

♦ **Lista dei parametri:**

Parametri di input:

a = Estremo sinistro dell'intervallo di integrazione

b = Estremo destro dell'intervallo di integrazione

f = Funzione integranda scritta nel seguente modo:

$$f = '(100./(x.^7)).*\sin(10./(x.^7))'$$

tolleranza = Tolleranza richiesta

nfmax = Massimo numero consentito di valutazioni della funzione

Parametri di output:

AI = Approssimazione dell'integrale

Errore = Stima dell'errore

iflag = Indicatore di errore:

0 Se la condizione di uscita si è verificata per il raggiungimento della tolleranza richiesta;

1 Se la condizione di uscita si è verificata per il raggiungimento del numero massimo di valutazioni;

nval = Numero di valutazioni della funzione effettuato

♦ **Indicatori di errore:**

iflag: indica se durante la procedura si verificano degli errori, bloccando in tal caso l'esecuzione. Può assumere i seguenti valori:

0 Se la condizione di uscita si è verificata per il raggiungimento della tolleranza richiesta;

1 Se la condizione di uscita si è verificata per il raggiungimento del numero massimo di valutazioni;

♦ **Funzioni ausiliarie:**

funzione Fun:

questa routine calcola i valori della funzione data in input secondo Matlab.

La specifica è:

function y=Fun(f, x)

♦ **Complessità computazionale:**

Complessità di tempo: dipende dalla complessità della funzione e dal numero di valutazioni

Complessità di spazio:

```
EDU>>
EDU>> whos
Name      Size      Bytes  Class
AI         1x1         8  double array
Errore     1x1         8  double array
a          1x1         8  double array
b          1x1         8  double array
fun        1x30        60  char array
iflag      1x1         8  double array (logical)
nfmmax     1x1         8  double array
nval       1x1         8  double array
tolleranza 1x1         8  double array

Grand total is 38 elements using 124 bytes
```

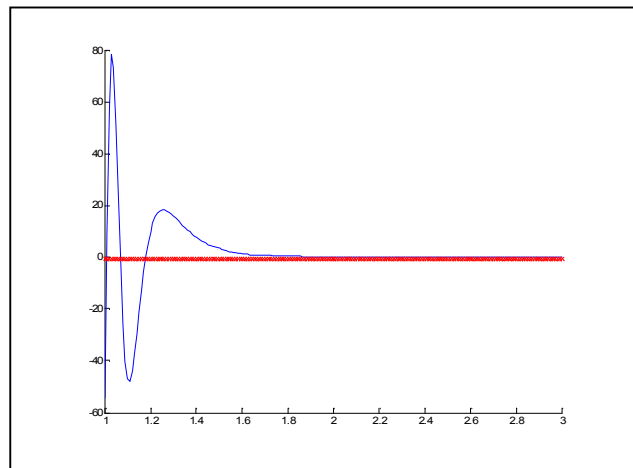
♦ **Accuratezza fornita:**

l'accuratezza del risultato dipende dalla tolleranza richiesta, che viene fornita in input.

12 Progetto Metodo di Simpson

◆ Esempio d'uso:

```
fun =  
(100./(x.^7)).*sin(10./(x.^7))  
  
a =  
    1  
  
b =  
    3  
  
nfmmax =  
    1000  
  
tolleranza =  
    0.0010000000000000  
  
soluzioni della simfix  
AI =  
    3.34836501062069  
  
Errore =  
    8.794999028849541e-004  
  
iflag =  
    0  
  
nval =  
    257
```



◆ **Esempio d'uso:**

Primo TEST

fun =

$(100./(x.^7)).*\sin(10./(x.^7))$

a =

1

b =

3

nfxmax =

100

tolleranza =

0.001000000000000

soluzioni della simfix

AI =

3.36155750916396

Errore =

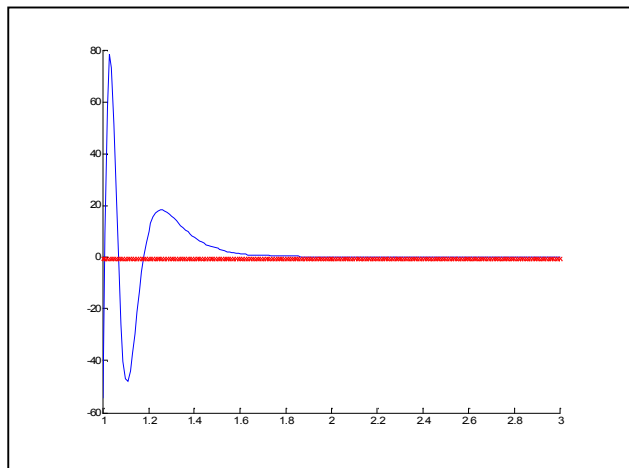
0.01202705155935

iflag =

1

nval =

129



- ◆ **Raccomandazioni d'uso:** I file sono stati implementati su **MATLAB Student Version 6.0**. Durante la fase di test su MATLAB 5.2 si sono verificati degli inconvenienti su alcuni plot. **Si raccomanda l'uso di MATLAB 6.0 o superiore.** Inoltre digitando **help simfix** si ha un rapido help sull'uso del file.

Sezione 2

Codice MATLAB


```
%          Progetto Metodo di Simpson
%          a schema fisso
%
%          Programma elaborato da
%
%          Giovanni DI CECCA & Virginia BELLINO
%          50 / 887          408 / 466
%
%          http://www.dicecca.net
%
%
% Chiamata della funzione
%
% [AI,Errore,iflag,nval]=simfix(a,b,f,tolleranza,nfmax)
%
%
% Informazioni sul programma
%
% Scopo: Calcola l'integrale definito di f(x) utilizzando la formula
%        di Simpson a schema fisso. La funzione prosegue nel processo
%        iterativo fino a quando:
%        1) Il margine di errore raggiunge la tolleranza richiesta;
%        2) Si sono effettuate nfmax valutazioni;
%
% Parametri:
%
% Input:  a = Estremo sinistro dell'intervallo di integrazione
%         b = Estremo destro dell'intervallo di integrazione
%         f = Funzione integranda scritta nel seguente modo:
%
%         f='(100./(x.^7)).*sin(10./(x.^7))'
%
%         tolleranza = Tolleranza richiesta
%         nfmax = Massimo numero consentito di valutazioni della funzione
%
% Output: AI = Approssimazione dell'integrale
%         Errore = Stima dell'errore
%         iflag = Indicatore di errore:
%                0 Se la condizione di uscita si è verificata per il
%                  raggiungimento della tolleranza richiesta;
%                1 Se la condizione di uscita si è verificata per il
%                  raggiungimento del numero massimo di valutazioni;
%
%         nval = Numero di valutazioni della funzione effettuato
%
function [AI,Errore,iflag,nval]=simfix(a,b,f,tolleranza,nfmax)
% Inizializzazione delle variabili
```

```
Iprecedente=0; % Stima dell'integrale al passo precedente

AI=0; % Inizializzazione dell'approssimazione dell'integrale

% Inizializzazione delle variabili usate per la valutazione
% della funzione negli estremi

fa=0;
fb=0;

% Valutazione della funzione negli estremi a et b
fa=Fun(f,a);
fb=Fun(f,b);

Errore=tolleranza;

% Numero di valutazioni della funzione effettuate per calcolare la
% prima approssimazione dell'integrale
nval=3;

iflag=0; % Inizializza l'indicatore di errore (tipo logical)

sommaprecedente=0; % Somma delle valutazioni di funzione all'iterazione k
sommacorrente=0; % Somma delle valutazioni di funzione all'iterazione k+1

h=(b-a)/2;

% Somma delle valutazioni di funzione al passo k+1
% Inizialmente, tale somma contiene il valore della funzione valutata nel
punto medio
% dell'intervallo di integrazione (a,b)
sommacorrente=Fun(f,a+h);

% Calcolo di Simpson semplice su tre punti
AI=(h/3)*(fa + 4*sommacorrente + fb);

k=1; % Inizializza il contatore k

% Inizio procedimento di Simpson a schema fisso
% Il calcolo viene effettuato con un ciclo while.

while (Errore>=tolleranza) & (nval<=nfmax)

    k=k+1; % Passa alla iterazione successiva

    Iprecedente=AI; % Memorizza il valore dell'ultima iterazione
```

```
% Calcola l'ampiezza degli intervalli
h=(b-a)/(2^k);

% Calcola il numero dei nodi in cui andare a valutare la funzione
m=2^(k-1);

% Aggiornamento della somma delle valutazioni di funzione al passo k
sommaprecedente = sommaprecedente + sommacorrente;

% Aggiornamento della somma delle valutazioni di funzione al passo k+1
% con l'aiuto della funzione ausiliaria Fun
sommacorrente=0;

for i=1:1:m
    sommacorrente = sommacorrente + Fun(f,a+((2*i)-1)*h);
end

% Calcolo dell'integrale definito con la formula di Simpson composta
AI=(h/3)*(fa + 2*sommaprecedente + 4*sommacorrente + fb);

% Calcola l'errore o resto utilizzando la stima di Richardson
Errore=abs(AI-Iprecedente)/15;

% Aggiornamento del numero di valutazioni effettuate
nval=nval+m;
end

% Controllo sulla possibilità di terminazione del ciclo per il raggiungime
nto
% di una delle due condizioni previste, e cioè:
% a- il numero di valutazioni della funzione effettuate supera il massimo
consentito
% b- l'errore raggiunge la tolleranza richiesta
iflag=(nval>=nfmax & Errore>=tolleranza);

% Plot della funzione

hold on; % Consenti la possibilità di sovrascrivere il grafico

zoom on; % Abilita la funzione di zoom

% Inserisci i valori degli intervallini nel plot a video
x=linspace(a,b,abs(a-b)/0.01);

eval(sprintf('y_arr=%s;',f));

plot(x,y_arr,'b');
```

```
plot(x,-0.5,'rx');
```

```
hold off; % Disabilita Hold on
```

```
%          Progetto Metodo di Simpson
%          a schema fisso
%
%          Programma elaborato da
%
%          Giovanni DI CECCA & Virginia BELLINO
%          50 / 887          408 / 466
%
%          http://www.dicecca.net
%
% Funzione ausiliaria di simfix
%
% Questa routine calcola i valori della funzione
% data in input secondo Matlab

function y=Fun( f, x )

% crea la funzione
Funzione = sprintf( 'y=%s;', f);

% calcola i valori
eval( Funzione );
```


Sezione 3

Testing

Primo TEST

fun =

$(100./(x.^7)).*\sin(10./(x.^7))$

a =

1

b =

3

nfmax =

1000

tolleranza =

0.001000000000000

soluzioni della simfix

AI =

3.34836501062069

Errore =

8.794999028849541e-004

iflag =

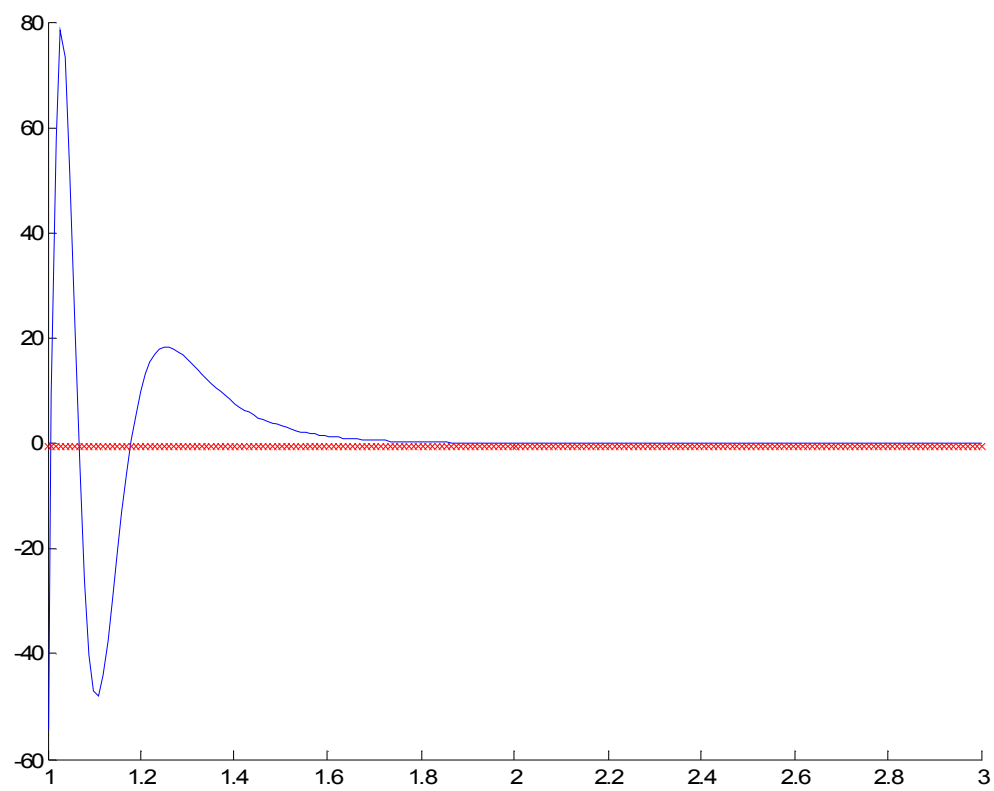
0

nval =

257

Premi un tasto per continuare

26 Progetto Metodo di Simpson



Secondo TEST

fun =

$(100./(x.^7)).*\sin(10./(x.^7))$

a =

1

b =

3

nfmax =

10000

tolleranza =

1.0000000000000000e-006

soluzioni della simfix

AI =

3.34754709533644

Errore =

1.941855582702582e-007

iflag =

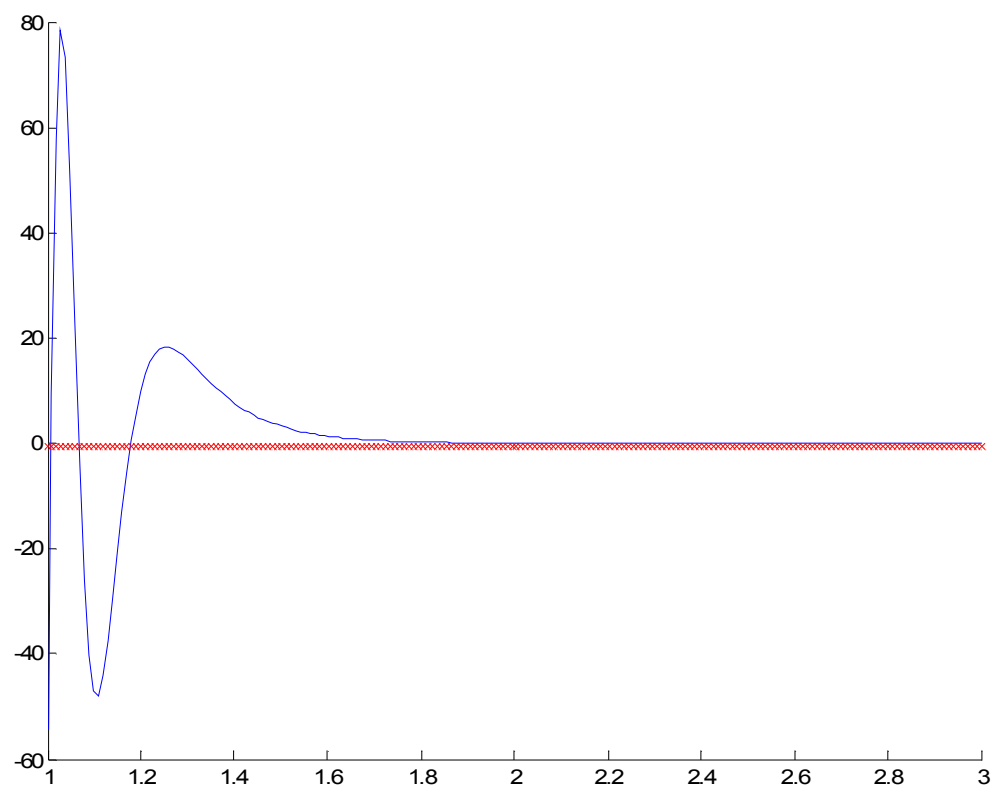
0

nval =

2049

Premi un tasto per continuare

28 Progetto Metodo di Simpson



Terzo TEST

fun =

$(1./((x.^2)+1./(2.^16)))$

a =

-1

b =

1

nfmax =

10000

tolleranza =

0.1000000000000000

soluzioni della simfix

AI =

8.022458597195574e+002

Errore =

0.06625213542247

iflag =

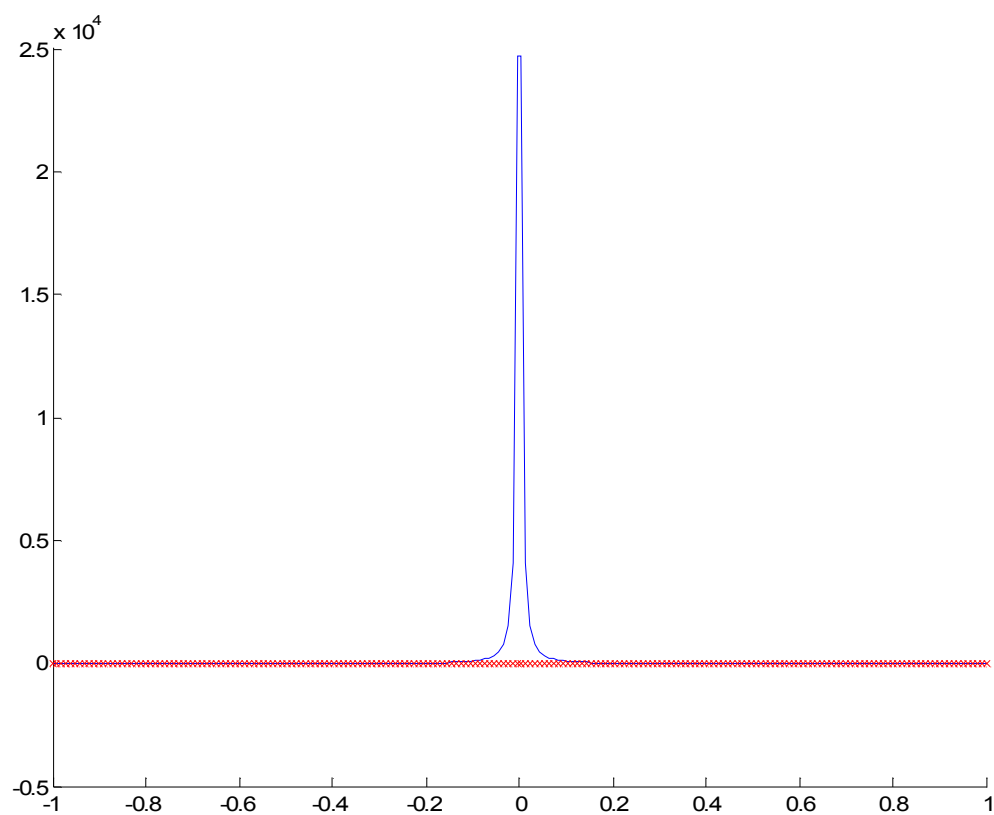
0

nval =

2049

Premi un tasto per continuare

30 Progetto Metodo di Simpson



Quarto TEST

fun =

$(1./((x.^2)+1./(2.^16)))$

a =

-1

b =

1

nfmax =

10000

tolleranza =

0.001000000000000

soluzioni della simfix

AI =

8.022477294848998e+002

Errore =

1.246510228232485e-004

iflag =

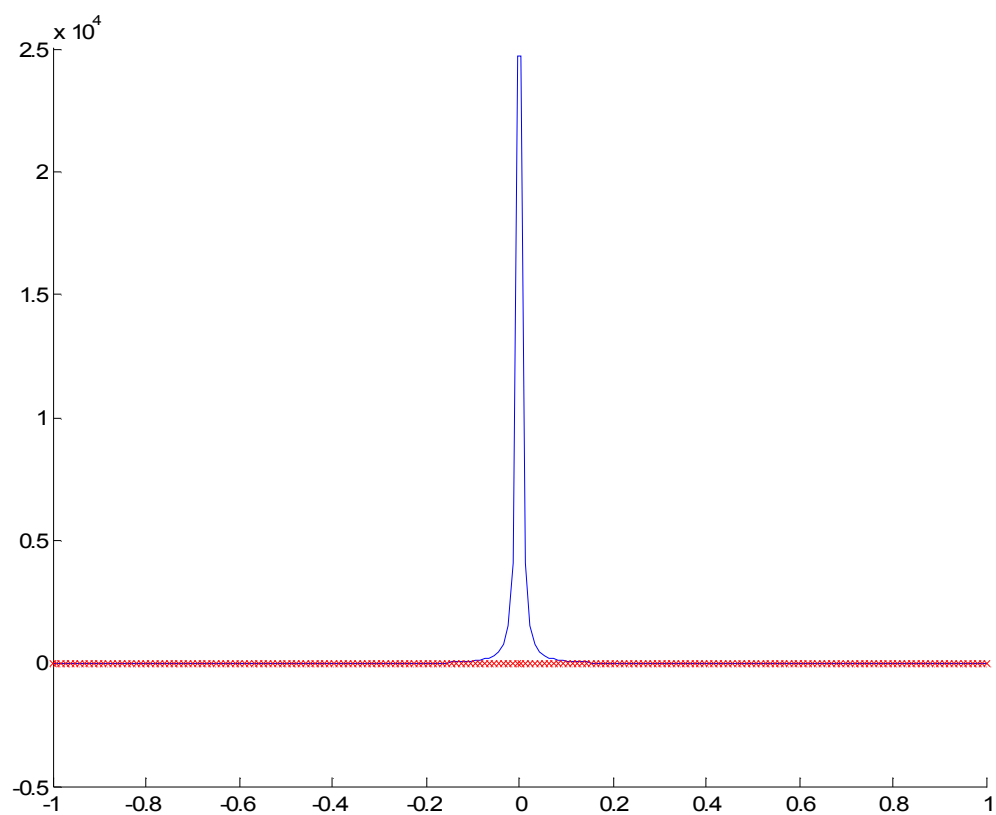
0

nval =

4097

Premi un tasto per continuare

32 Progetto Metodo di Simpson



Quinto TEST

fun =

$\sin(x) + \cos(x)$

a =

-1.57079632679490

b =

1.57079632679490

nfmax =

100

tolleranza =

0.0010000000000000

soluzioni della simfix

AI =

2.00026916994839

Errore =

2.860390024022053e-004

iflag =

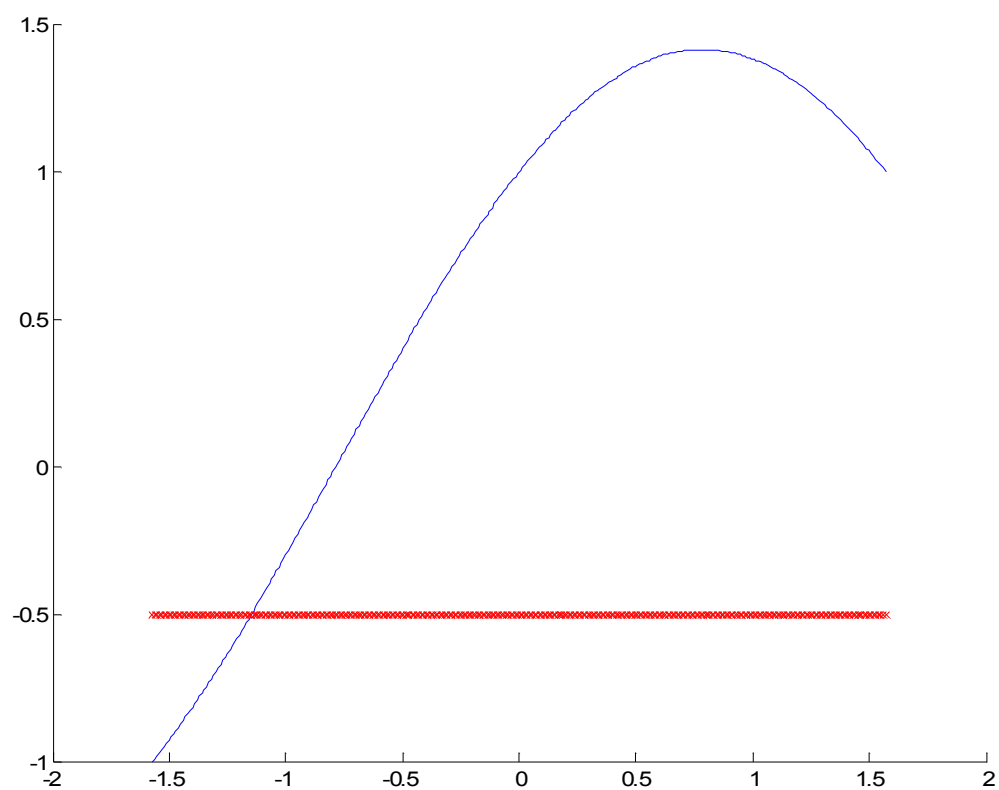
0

nval =

9

Premi un tasto per continuare

34 Progetto Metodo di Simpson



Sesto TEST

fun =

$\sin(x) + \cos(x)$

a =

-1.57079632679490

b =

1.57079632679490

nfmax =

100

tolleranza =

1.0000000000000000e-006

soluzioni della simfix

AI =

2.00000006453000

Errore =

6.458929409092207e-008

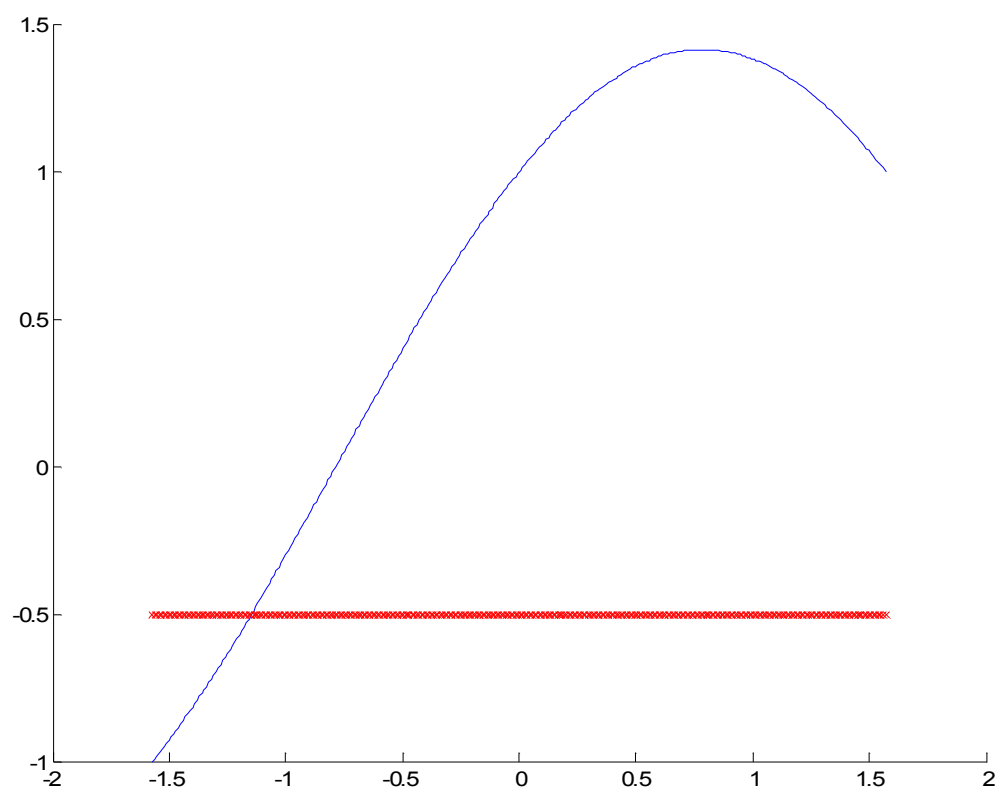
iflag =

0

nval =

65

Fine del test

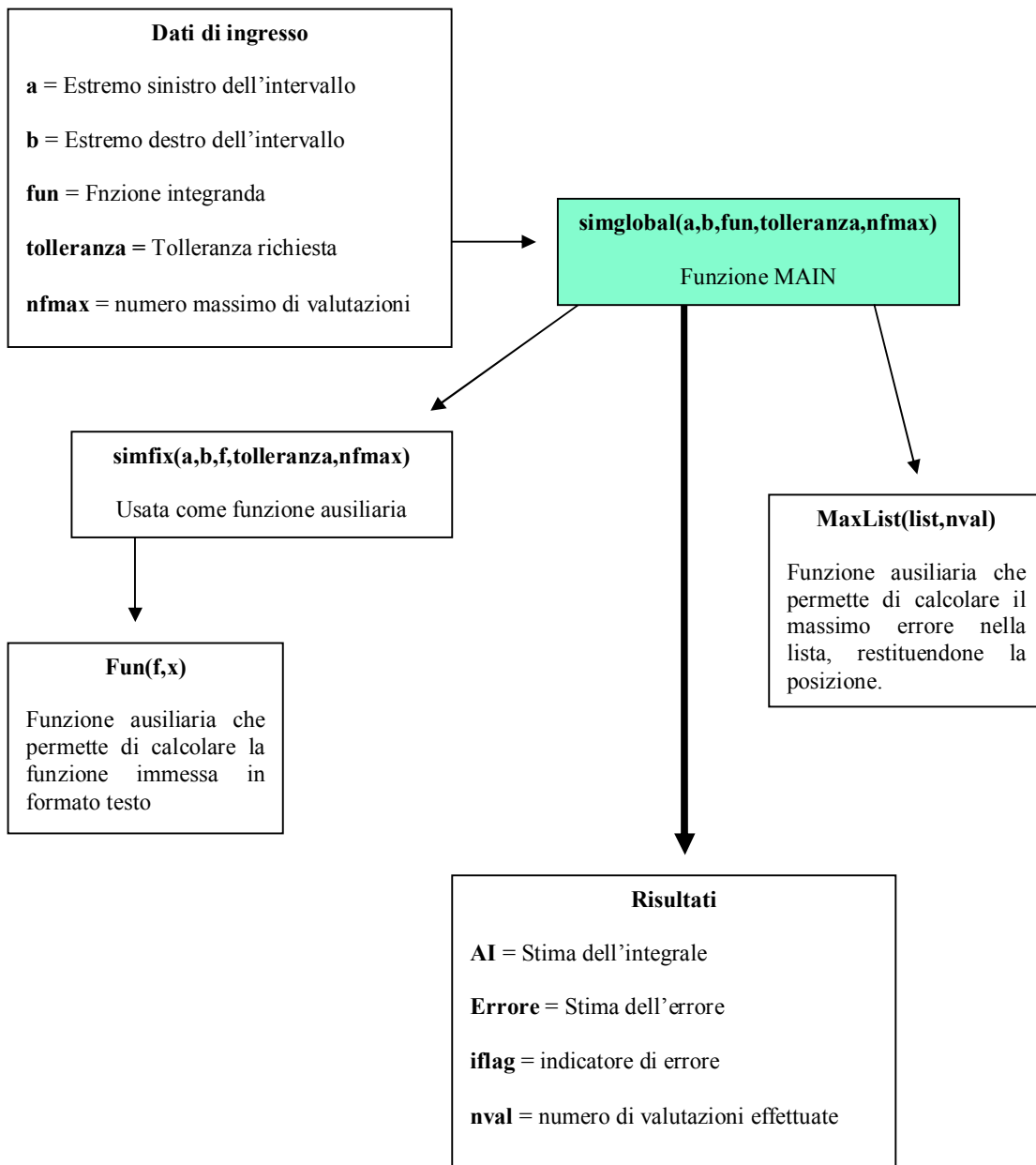


Simpson a schema adattivo con controllo globale dell'errore

Progetto Metodo di Simpson a schema globale

$$\int_a^b f(x) dx$$

Schema grafico



Sezione 1

Documentazione esterna

- ♦ **Scopo:** calcolare il valore numerico approssimato di un integrale definito del tipo

$$\int_a^b f(x) dx$$

utilizzando la strategia basata sulla *formula di quadratura composta di Simpson a schema adattivo con controllo globale dell'errore*.

- ♦ **Specifiche:**

function [AI,Errore,iflag,nval]=simglobal(a,b,fun,tolleranza,nfmax)

- ♦ **Descrizione:**

dopo aver inizializzato le variabili e calcolato la prima stima dell'integrale richiamando la funzione simfix (eseguita su 3 punti), l'algoritmo crea una lista in cui verranno memorizzati tutti i dati relativi ai sottointervalli.

Successivamente, ha inizio il ciclo while che esegue il procedimento di Simpson a schema adattivo. Tale ciclo esegue le seguenti operazioni:

- Ricerca nella lista dell'intervallo con errore massimo utilizzando il modulo ausiliario MaxList.
- L'intervallo selezionato viene suddiviso in 2 parti per calcolare le nuove approssimazioni di integrale ed errore.
- Rimozione dalla lista della vecchia approssimazione relativa all'intervallo selezionato.
- Calcolo delle nuove approssimazioni sul sottointervallo di sinistra e di destra.
- Aggiornamento della lista con i nuovi dati.
- Aggiornamento delle stime complessive di integrale ed errore

Infine, vi è un controllo sulle condizioni di terminazione del ciclo, cui seguono le istruzioni per il plot della funzione.

♦ Riferimenti bibliografici:

- James F. Epperson
INTRODUZIONE ALL'ANALISI NUMERICA
McGraw-Hill
- Prof. Eleonora Messina
Appunti del corso di calcolo numerico
A.A 2004/2005

♦ Lista dei parametri:

Parametri di input:

a = Estremo sinistro dell'intervallo di integrazione

b = Estremo destro dell'intervallo di integrazione

fun = Funzione integranda scritta nel seguente modo:

$$f = (100./(x.^7)).*\sin(10./(x.^7))'$$

tolleranza = Tolleranza richiesta

nfmax = Massimo numero consentito di valutazioni della funzione

Parametri di output:

AI = Approssimazione dell'integrale

Errore = Stima dell'errore

iflag = Indicatore di errore:

0 Se la condizione di uscita si è verificata per il raggiungimento della tolleranza richiesta;

1 Se la condizione di uscita si è verificata per il raggiungimento del numero massimo di valutazioni;

nval = Numero di valutazioni della funzione effettuato

♦ **Indicatori di errore:**

iflag: indica se durante la procedura si verificano degli errori, bloccando in tal caso l'esecuzione. Può assumere i seguenti valori:

0 Se la condizione di uscita si è verificata per il raggiungimento della tolleranza richiesta;

1 Se la condizione di uscita si è verificata per il raggiungimento del numero massimo di valutazioni;

♦ **Funzioni ausiliarie:**

funzione MaxList:

funzione che esegue la ricerca nella lista dell'intervallo con errore massimo. Una volta trovato, ne restituisce i dati al programma chiamante.

La specifica è:

function [E,approx,xl,xh,k]=MaxList(list,c)

♦ **Complessità computazionale:**

Complessità di tempo: dipende dalla complessità della funzione e dal numero di valutazioni

Complessità di spazio:

```
EDU>>
EDU>> whos
Name      Size      Bytes  Class
AI         1x1         8  double array
Errore     1x1         8  double array
a          1x1         8  double array
b          1x1         8  double array
fun        1x30        60  char array
iflag      1x1         8  double array (logical)
nfmax      1x1         8  double array
nval       1x1         8  double array
tolleranza 1x1         8  double array

Grand total is 38 elements using 124 bytes
```

♦ **Accuratezza fornita:**

l'accuratezza del risultato dipende dalla tolleranza richiesta, che viene fornita in input.

♦ Esempio d'uso:

```

fun =
(100./(x.^7)).*sin(10./(x.^7))

a =
    1

b =
    3

nfmmax =
    1000

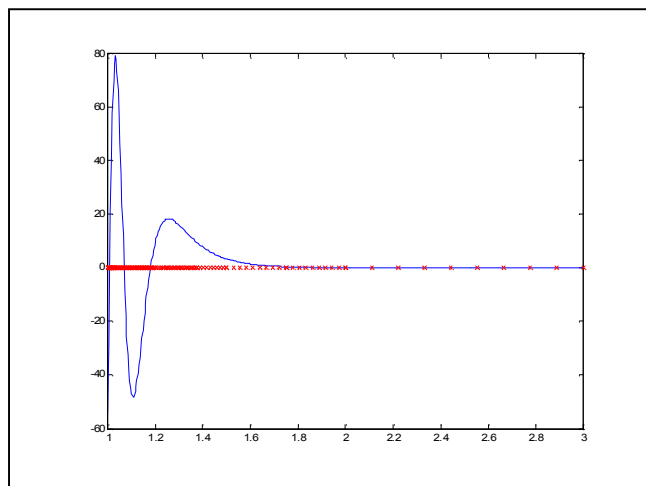
tolleranza =
    0.0010000000000000

soluzioni della simglobal
AI =
    3.34792914607643

Errore =
    7.037416393202394e-004

iflag =
    0

nval =
    14
    
```



◆ **Esempio d'uso:**

```

fun =
(100./(x.^7)).*sin(10./(x.^7))

a =
    1

b =
    3

nfmmax =
    8

tolleranza =
    0.0010000000000000

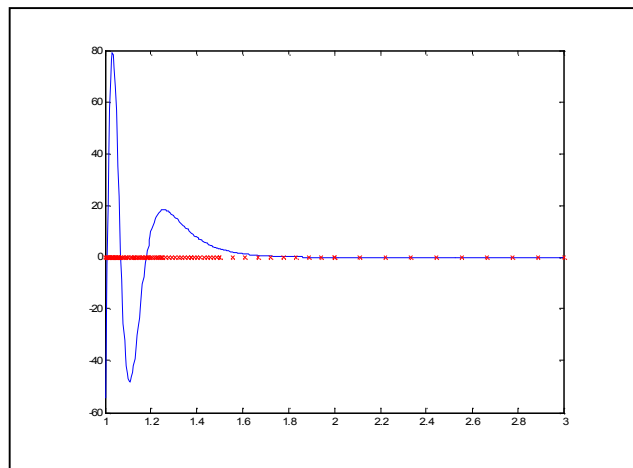
soluzioni della singlobal
AI =
    3.34925629756928

Errore =
    0.00599895147469

iflag =
    1

nval =
    9

```



- ◆ **Raccomandazioni d'uso:** I file sono stati implementati su **MATLAB Student Version 6.0**. Durante la fase di test su MATLAB 5.2 si sono verificati degli anomalie su alcuni plot. **Si raccomanda l'uso di MATLAB 6.0 o superiore.** Inoltre digitando **help singlobal** si ha un rapido help sull'uso del file.

Sezione 2

Codice MATLAB


```
%
    Progetto Metodo di Simpson
    a schema globale
%
    Programma elaborato da
%
    Giovanni DI CECCA & Virginia BELLINO
    50 / 887          408 / 466
%
    http://www.dicecca.net
%
% Chiamata della funzione
%
% [AI,Errore,iflag,nval]=simglobal(a,b,fun,tolleranza,nfmax)
%
%
% Informazioni sul programma
%
%
% Scopo: Calcola l'integrale definito di f(x) utilizzando la formula
%        di Simpson a schema adattivo con controllo globale dell'errore.
%        La funzione prosegue nel processo iterativo fino a quando:
%
%        1) Il margine di errore raggiunge la tolleranza richiesta;
%        2) Si sono effettuate nfmax valutazioni;
%
% Parametri:
%
% Input : a = Estremo sinistro dell'intervallo di integrazione
%         b = Estremo destro dell'intervallo di integrazione
%         fun = Funzione integranda scritta nel seguente modo:
%
%         fun='(100./(x.^7)).*sin(10./(x.^7))'
%
%         tolleranza = Tolleranza richiesta
%         nfmax = Massimo numero consentito di valutazioni della funzione
%
% Output: AI = Stima dell'integrale
%         Errore = Stima dell'errore
%         iflag = Indicatore di errore:
%                0 Se la condizione di uscita si è verificata per il
%                  raggiungimento della tolleranza richiesta;
%                1 Se la condizione di uscita si è verificata per il
%                  raggiungimento del numero massimo di valutazioni;
%
%         nval = Numero di valutazioni di funzione effettuato
```

```
function [AI,Errore,iflag,nval]=simglobal(a,b,fun,tolleranza,nfmax)
```

```
%Inizializzo le variabili
nval=1;

% Chiama la funzione simfix per eseguire la prima stima
% dell'integrale (P.S.: z et g non vengono considerati)
[Iprecedente,Errore,z,g]=simfix(a,b,fun,0,3);

% Crea una lista in cui vengono inseriti i dati relativi agli intervalli
list(nval)=struct('x1',a,'xh',b,'approx',Iprecedente,'est',Errore);

AI=Iprecedente;

% Inizio procedimento Simpson adattivo
while (Errore>=tolleranza & nval<=nfmax)

    % Rileva nella lista l'intervallo con il massimo
    % errore mediante una funzione apposita
    [E,Iprecedente,xa,xb,k]=MaxList(list,nval);

    % Suddivide l'intervallo selezionato in due sottointervalli
    h=(xb-xa)/2;

    % Calcola il punto medio
    xmid=xa+h;

    % Rimuove dalla lista la vecchia approssimazione
    % relativa all'intervallo selezionato
    list(k)=[];

    % Calcola la nuova approssimazione sul sottointervallo di sinistra
    % ed aggiorna la lista
    [I1,E1,z,g]=simfix(xa,xmid,fun,0,3);
    list(nval)=struct('x1',xa,'xh',xmid,'approx',I1,'est',E1);

    % Calcola la nuova approssimazione sul sottointervallo di destra
    % ed aggiorna la lista
    nval=nval+1;
    [I2,E2,z,g]=simfix(xmid,xb,fun,0,3);
    list(nval)=struct('x1',xmid,'xh',xb,'approx',I2,'est',E2);

    % Ricava la nuova stima dell'integrale e dell'errore.
    AI=AI-Iprecedente+I1+I2;
    Errore=Errore-E+E1+E2;
end

% Controllo sulla possibilità di terminazione del ciclo per il raggiungime
nto
% di una delle due condizioni previste, e cioè:
```

```
% a- il numero di valutazioni della funzione effettuate supera il massimo
consentito
% b- l'errore raggiunge la tolleranza richiesta
iflag=(nval>=nfmax & Errore>=tolleranza);

% Disegna la funzione f(x)
x=linspace(a,b,500);
eval(sprintf('y_arr=%s;',fun));
plot(x,y_arr);

zoom on
hold on

% Calcola i valori dei nodi
arr_nodi=[];
for (i=1:nval)
    x=linspace(getfield(list(i),'xl'),getfield(list(i),'xh'),10);
    arr_nodi = [arr_nodi x];
end

% Disegna i nodi
plot(arr_nodi,0,'rx');

hold off
```

```
%          Progetto Metodo di Simpson
%          a schema globale
%
%          Programma elaborato da
%
%          Giovanni DI CECCA & Virginia BELLINO
%          50 / 887          408 / 466
%
%          http://www.dicecca.net

% Funzione ausiliaria di simglobal

function [E,approx,xl,xh,k]=MaxList(list,c)

% Salva i valori alla testa della struttura
E=getfield(list(1),'est');

approx=getfield(list(1),'approx');

xl=getfield(list(1),'xl');

xh=getfield(list(1),'xh');

k=1;

% Ricerca dei dati relativi all'intervallo con errore massimo
for (j=2:1:c)

    % Confronta la testa della lista con gli
    % altri valori
    if (E<getfield(list(j),'est'))

        E=getfield(list(j),'est');

        approx=getfield(list(j),'approx');

        xl=getfield(list(j),'xl');

        xh=getfield(list(j),'xh');

        % Indica la posizione nella list adell'intervallo
        % con il massimo errore
        k=j;

    end % End if

end % end for
```

Sezione 3

Testing

Primo TEST

fun =

$(100./(x.^7)).*\sin(10./(x.^7))$

a =

1

b =

3

nfmax =

1000

tolleranza =

0.001000000000000

soluzioni della simglobal

AI =

3.34792914607643

Errore =

7.037416393202394e-004

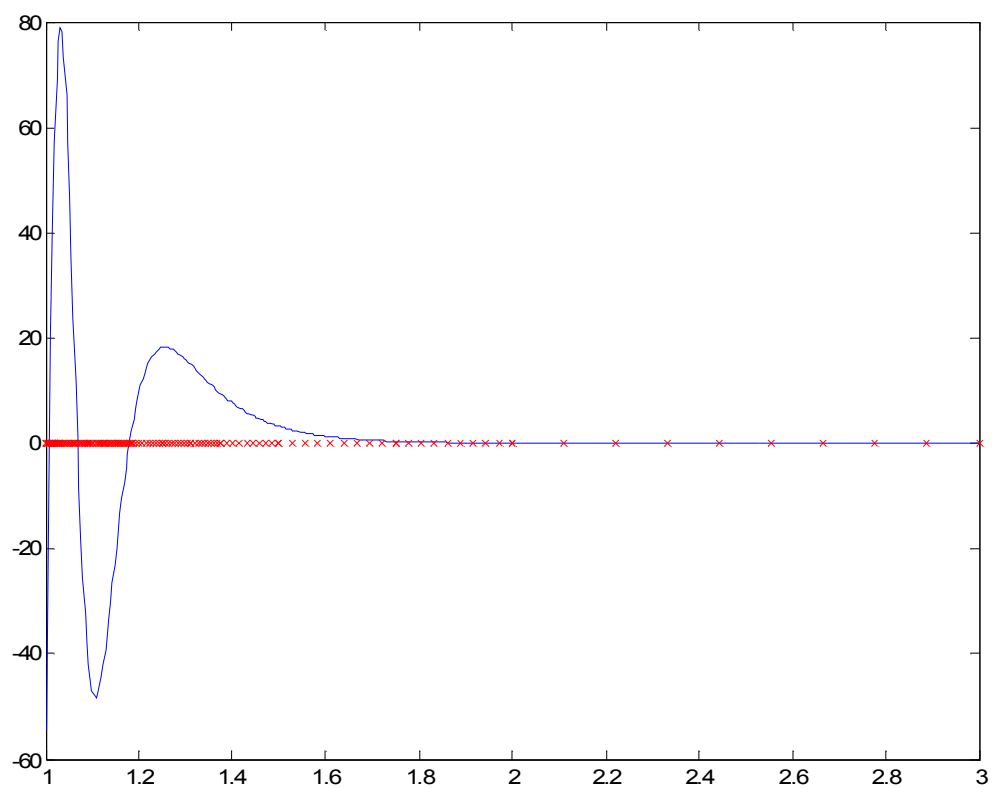
iflag =

0

nval =

14

Premi un tasto per continuare



Secondo TEST

fun =

$(100./(x.^7)).*\sin(10./(x.^7))$

a =

1

b =

3

nfmax =

1000

tolleranza =

$1.0000000000000000e-006$

soluzioni della simglobal

AI =

3.34754720021388

Errore =

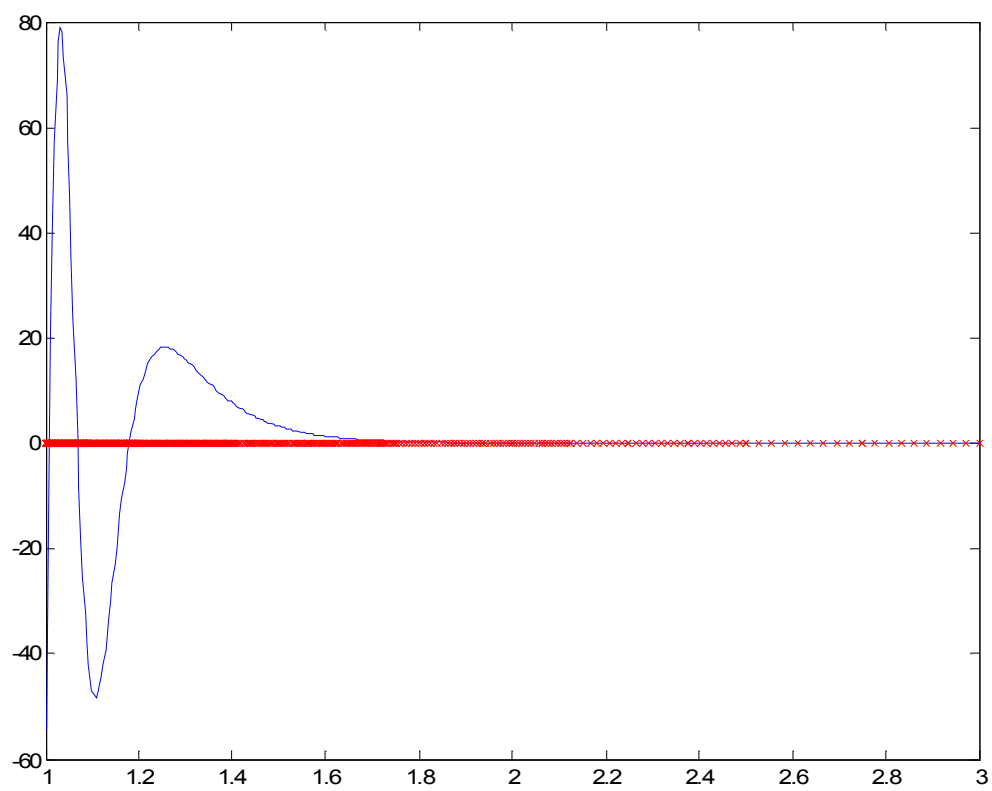
$9.519758794661029e-007$

iflag =

0

nval =

72



Terzo TEST

fun =

$(1./((x.^2)+1./(2.^16)))$

a =

-1

b =

1

nfmax =

1000

tolleranza =

0.1000000000000000

soluzioni della simglobal

AI =

8.023725162233195e+002

Errore =

0.09883973133026

iflag =

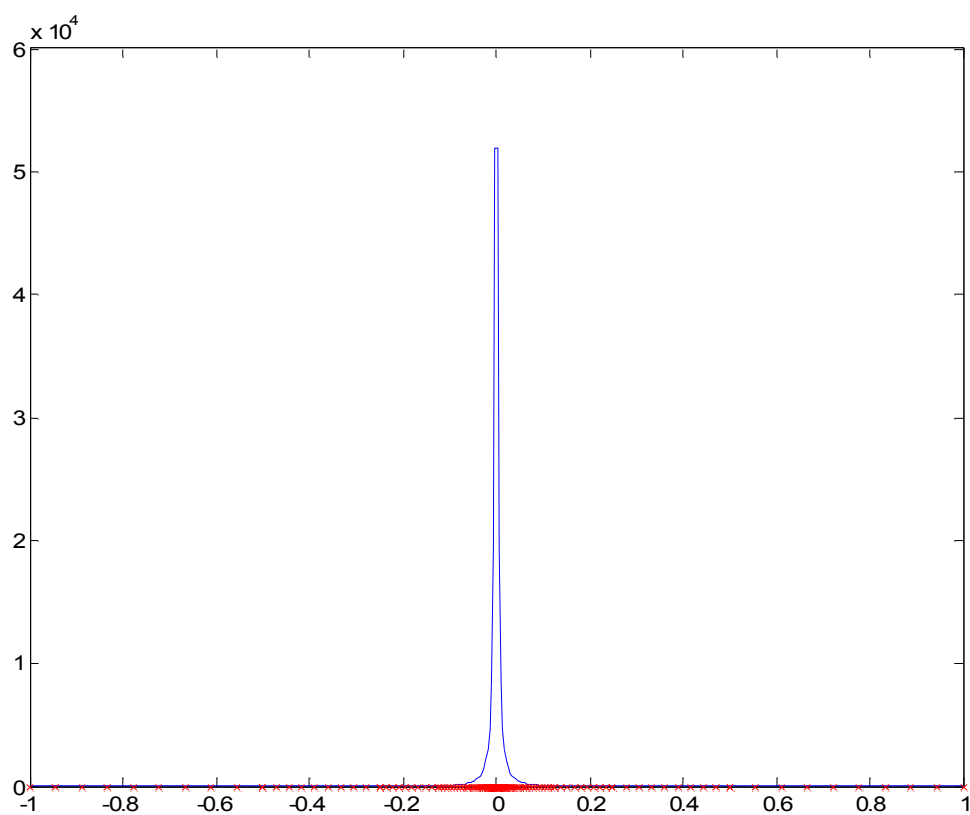
0

nval =

20

Premi un tasto per continuare

60 Progetto Metodo di Simpson



Quarto TEST

fun =

$(1./((x.^2)+1./(2.^16)))$

a =

-1

b =

1

nfmax =

1000

tolleranza =

0.001000000000000

soluzioni della simglobal

AI =

8.022483189444289e+002

Errore =

9.754756183261852e-004

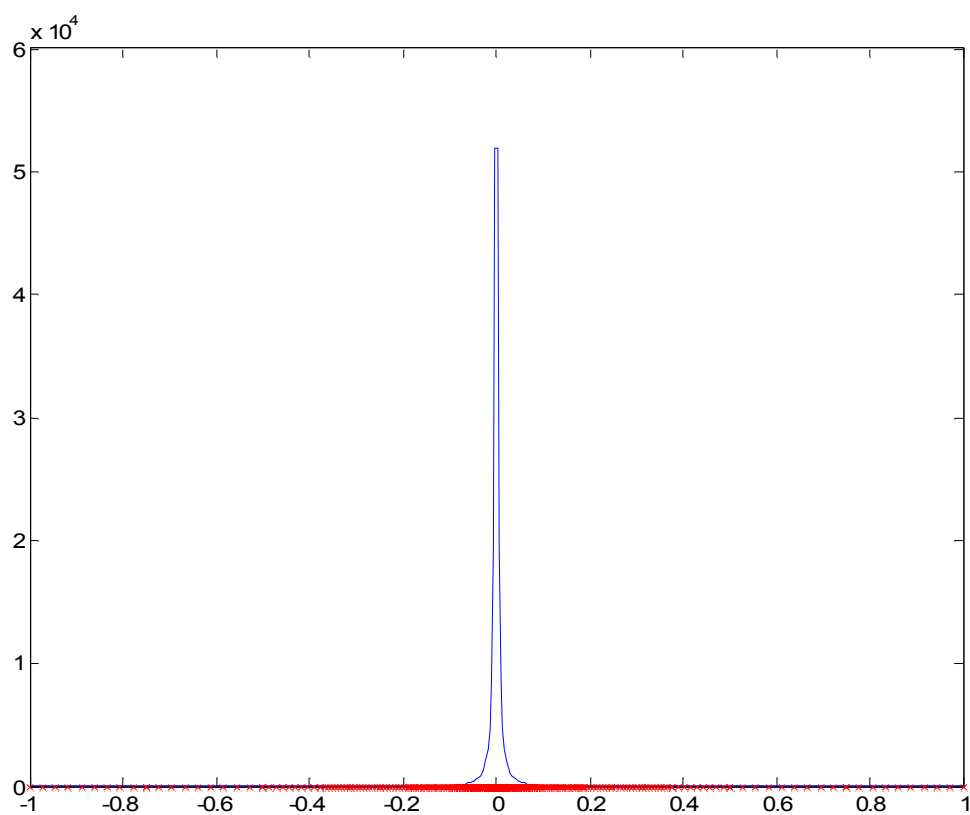
iflag =

0

nval =

65

Premi un tasto per continuare



Quinto TEST

fun =

$\sin(x) + \cos(x)$

a =

-1.57079632679490

b =

1.57079632679490

nfmax =

1000

tolleranza =

0.0010000000000000

soluzioni della simglobal

AI =

2.00026916994839

Errore =

2.860390024022120e-004

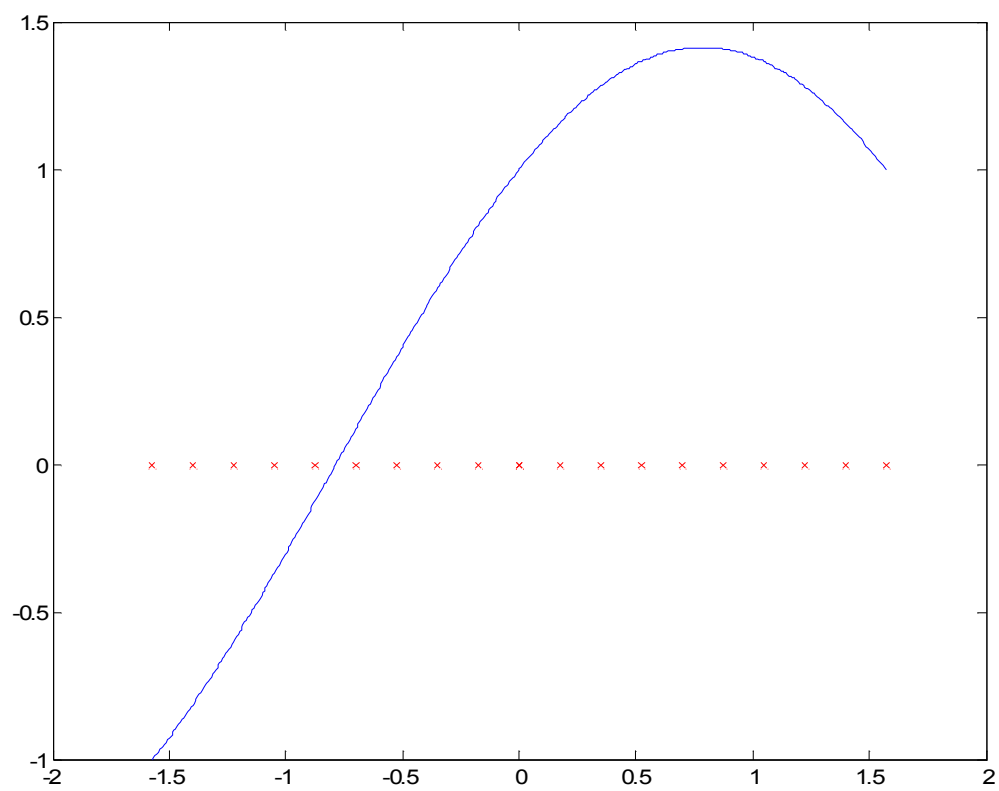
iflag =

0

nval =

2

Premi un tasto per continuare



Sesto TEST

fun =

$\sin(x) + \cos(x)$

a =

-1.57079632679490

b =

1.57079632679490

nfmax =

1000

tolleranza =

1.0000000000000000e-006

soluzioni della singlobal

AI =

2.00000077120336

Errore =

7.739978070791364e-007

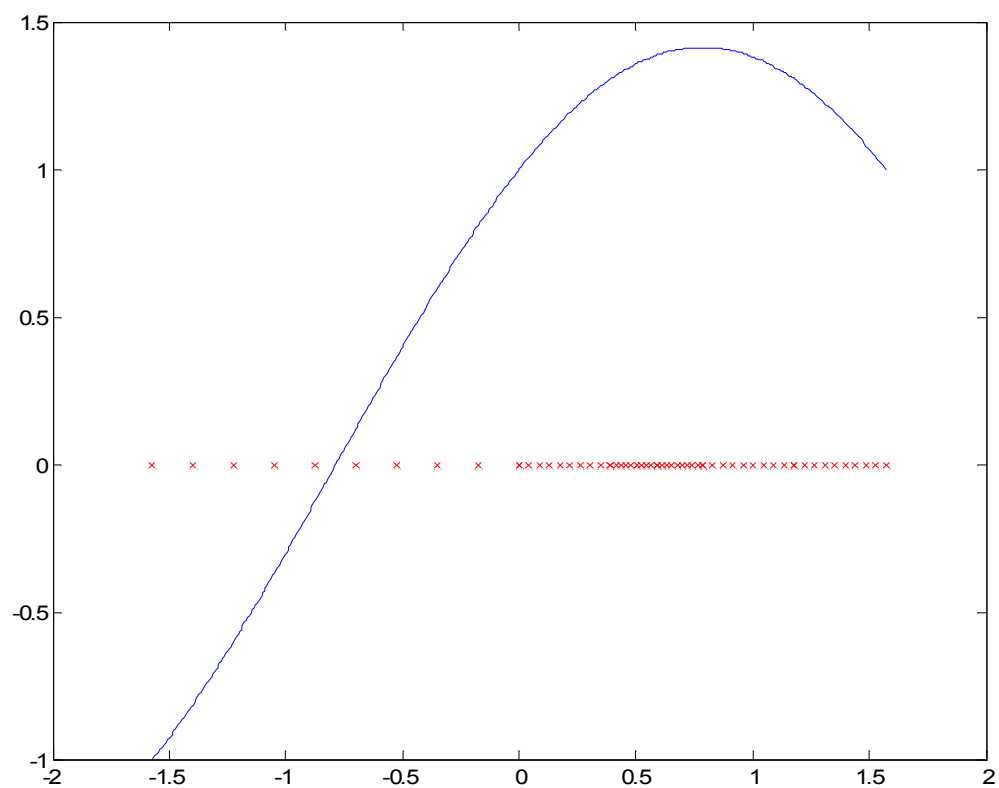
iflag =

0

nval =

6

Fine del test



Osservazioni

Simpson fisso

Poiché l'idea di fondo dell'algoritmo di Simpson a schema fisso è quella di campionare la funzione in punti equidistanti (e ciò si ottiene dimezzando l'ampiezza degli intervalli ad ogni iterazione), osservando i test si può notare che esso risulta efficiente solo nei casi in cui la funzione presenta un andamento abbastanza regolare in tutto l'intervallo di integrazione.

In caso contrario infatti, ovvero quando la funzione presenta brusche oscillazioni in alcuni sottointervalli, per arrivare ad ottenere una precisione maggiore del risultato, vengono effettuate molte valutazioni della funzione assolutamente non necessarie, dovute al fatto che, andando ad infittire il numero dei nodi nelle cosiddette “zone difficili”, si aumenta inutilmente anche il numero dei nodi nelle zone ove l'andamento è regolare.

Inoltre, è opportuno osservare anche che, diminuendo la tolleranza, e quindi aumentando la precisione richiesta, cresce il numero di nodi in cui valutare la funzione per ottenere il risultato.

Simpson adattivo

La strategia adattiva, basata sull'idea di scegliere i nodi adeguandosi all'andamento della funzione, consente di aumentare l'efficienza dell'algoritmo quando ci si trova davanti ad una funzione che presenta un andamento non regolare caratterizzato da bruschi cambiamenti lungo l'intervallo di integrazione.

In questi casi infatti, il numero dei nodi viene infittito nelle “zone difficili” caratterizzate da bruschi cambiamenti di andamento ed alto tasso di errore, mentre nei sottointervalli più regolari, ove l'errore è già basso, il numero dei nodi si dirada.

In tal modo è possibile raggiungere la precisione richiesta con un numero minore di iterazioni.

@@@@@

Quando invece ci si trova davanti a funzioni con un andamento oscillante ma abbastanza morbido su tutto l'intervallo di integrazione, i due algoritmi presentano pressappoco la medesima efficienza.

Ma osserviamo ora più in dettaglio i risultati di ciascun test.

PRIMA FUNZIONE

In questo caso, la funzione presenta inizialmente una brusca oscillazione, per poi stabilizzarsi ed assumere un andamento regolare.

Usando lo schema fisso, il numero delle valutazioni effettuate risulta considerevole a causa dei motivi esposti in precedenza (si va da 257 per una tolleranza di 10^{-3} fino a salire a 2049 per una tolleranza di 10^{-6}), mentre scegliendo lo schema adattivo tali cifre scendono sensibilmente (14 per una tolleranza di 10^{-3} e 72 per una tolleranza di 10^{-6}).

In casi di questo tipo dunque, è migliore l'algoritmo adattivo.

SECONDA FUNZIONE

La seconda funzione ha un andamento in gran parte lineare, ma è caratterizzata da un picco altissimo in prossimità dello zero.

Ciò implica che, usando lo schema fisso il numero di valutazioni effettuate per raggiungere il risultato è enorme con entrambe le tolleranze date in input.

Invece, la scelta dello schema adattivo si rivela migliore poiché il risultato viene raggiunto con sole 20 valutazioni per una tolleranza di 10^{-1} (contro le 2049 dello schema fisso) e 65 valutazioni per una tolleranza di 10^{-3} (contro le 4097 dello schema fisso).

Anche questo esempio mostra dunque la migliore efficienza dello schema adattivo in presenza di funzioni non regolari.

TERZA FUNZIONE

La terza funzione ha invece un andamento abbastanza regolare, e ciò implica che sia l'algoritmo a schema fisso che quello a schema adattivo producono risultati non molto dissimili tra loro.

Ciò è evidente soprattutto nel primo caso, ove, con tolleranza 10^{-3} lo schema fisso esegue 9 valutazioni, mentre quello adattivo appena 2.

Aumentando la tolleranza a 10^{-6} il divario diventa più evidente con 65 valutazioni per lo schema fisso e appena 6 per quello adattivo.



Università degli Studi di Napoli - Federico II

Facoltà di
Scienze Matematiche, Fisiche e Naturali

Corso di Laurea in Scienze Informatiche

Calcolo Numerico A.A. 2004 / 2005

Prof.ssa Eleonora Messina

Esercitazione Fitting

Realizzato da

Giovanni Di Cecca & Virginia Bellino

50 / 887

408 / 466



<http://www.dicecca.net>

2 *Esercitazione Fitting*

Indice

Testo dell'esercitazione

Codice MATLAB

Esercizio 1

- Funzione di Runge interpolato con polinomio di grado 10
 - Plot dei dati
- Funzione di Runge interpolato con polinomio di grado 20
 - Plot dei dati
- Profilo superiore di un aereo su punti definiti
 - Plot dei dati

Esercizio 2

- Funzione di Runge interpolato con i nodi di Chebychev in $n = 10$
 - Plot dei dati
- Funzione di Runge interpolato con i nodi di Chebychev in $n = 20$
 - Plot dei dati
- Profilo superiore di un aereo su punti definiti interpolato con un Spline cubica
 - Plot dei dati

Osservazioni

Testo dell'esercitazione

CORSO DI LAUREA IN INFORMATICA

a.a. 2004/2005

Calcolo Numerico

Esercitazione in Matlab

1. Usando la routine di Matlab “polyfit” costruire il polinomio di Lagrange di grado n ; commentare le prestazioni nei seguenti casi:

i. Sia $f(x)=1/(1+25x^2)$, dove $x \in [-1,1]$. Usando le routine di Matlab “polyval” e “plot”, plottare $f(x)$ ed i polinomi interpolanti di grado 10 e 20 su nodi equispaziati in $[-1,1]$.

ii. E' data una tabella di 20 punti che individuano il profilo superiore di un aereo

$x = 1 \ 2 \ 4 \ 11 \ 25 \ 45 \ 47 \ 51 \ 58 \ 63 \ 74 \ 105 \ 125 \ 145 \ 170 \ 180 \ 190 \ 204 \ 215 \ 228$

$y = 2 \ 3 \ 4 \ 6 \ 9 \ 12 \ 13 \ 15 \ 20 \ 23 \ 25 \ 24 \ 23 \ 21 \ 19 \ 20 \ 24 \ 33 \ 39 \ 40$

plottare i dati ed il polinomio interpolante relativo ai dati.

2. Sviluppare i seguenti temi confrontando le prestazioni ottenute con quelle dei punti corrispondenti dell'esercizio 1.

i. Ripetere l'esercizio 1.i. utilizzando come nodi i punti di Chebychev in $[-1,1]$.

$$x_k = \cos\left(\frac{(2k+1)\pi}{2(n+1)}\right), \quad 0 \leq k \leq n$$

dove $n=10$ ed $n=20$ rispettivamente. Commentare.

ii. Relativamente alla tabella dell'esercizio 1.ii., plottare i dati e la spline cubica (usare la routine routine “spline” di Matlab) associata ai dati. Commentare.

Codice MATLAB

```
% Esercitazione Fitting
%
% Programma elaborato da
%
% Giovanni DI CECCA & Virginia BELLINO
% 50 / 887 408 / 466
%
% http://www.dicecca.net
%
% Punto 1.i

clear % Pulisci memoria

clc % Pulisci schermo

% -----
% Costruzione della Funzione di Runge
% -----

% Polinomio di grado 10
pol=10

% Inserisci i nodi di interpolazione
x=linspace(1,-1,pol+1);

% Funzione di Runge
fun=inline(' (1+25*x.^2).^(-1) ');

% Vettore contenente i valori della funzione in x
y=feval(fun,x);

% -----
% Costruzione della Polinomio interpolante
% della Funzione di Runge
% -----

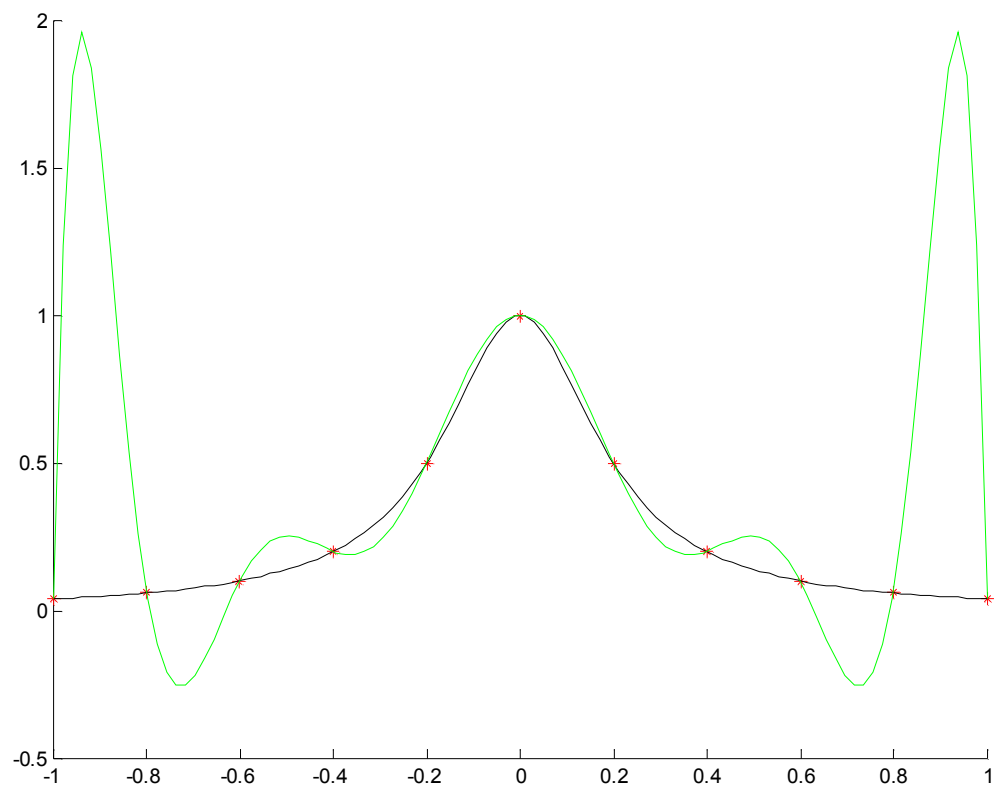
% Punti equispaziati in -1, 1
x1=linspace(-1,1);

% Vettore contenente i valori della funzione nei punti x1
y1=feval(fun,x1);

% Vettore dei coefficienti del polinomio interpolatore
p=polyfit(x,y,pol);

% Valuta il polinomio interpolante nei punti x1
f=polyval(p,x1);
```

```
% -----  
% Plot dei risultati  
% -----  
  
hold on  
  
zoom on  
  
% Plot dei punti della funzione di Runge  
plot(x,y,'r*')  
  
% Plot della funzione di Runge  
plot(x1,y1,'k')  
  
% Plot dei punti polinomio interpolante nei  
% punti equispaziati x1 della funzione di Runge  
plot(x1,f,'g')  
  
hold off
```



```
%          Esercitazione Fitting
%
%          Programma elaborato da
%
%          Giovanni DI CECCA & Virginia BELLINO
%          50 / 887          408 / 466
%
%          http://www.dicecca.net
%
% Punto 1.i

clear % Pulisci memoria

clc % Pulisci schermo

% -----
% Costruzione della Funzione di Runge
% -----

% Polinomio di grado 10
pol=20

% Inserisci i nodi di interpolazione
x=linspace(1,-1,pol+1);

% Funzione di Runge
fun=inline(' (1+25*x.^2).^(-1) ');

% Vettore contenente i valori della funzione in x
y=feval(fun,x);

% -----
% Costruzione della Polinomio interpolante
% della Funzione di Runge
% -----

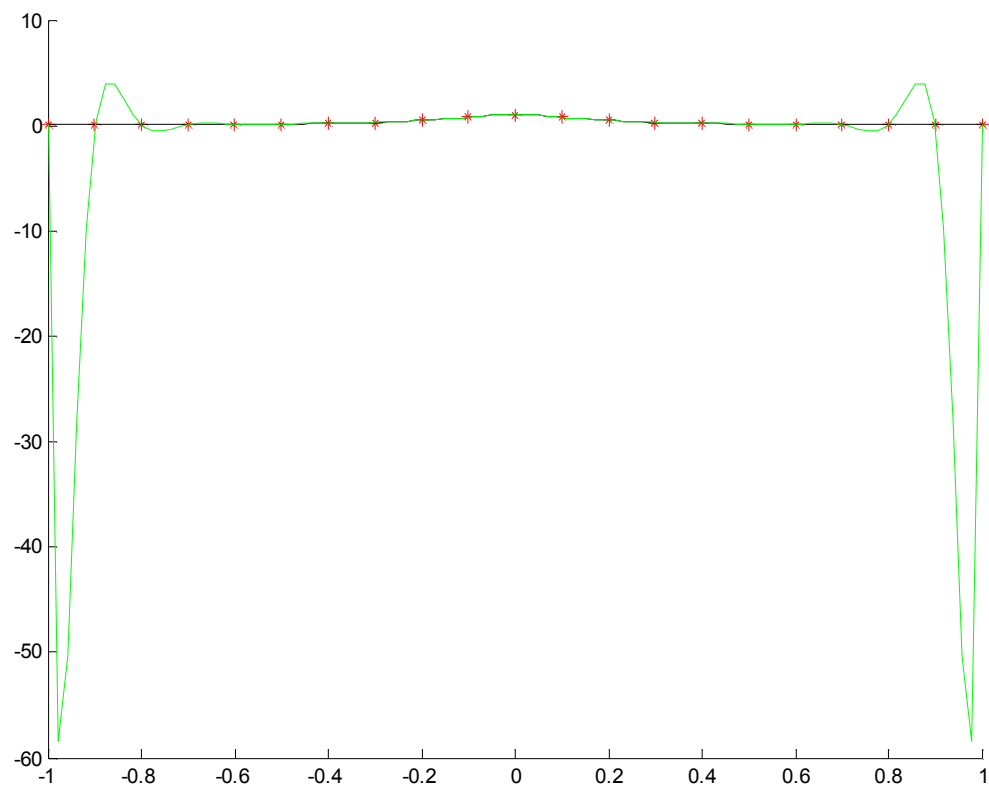
% Punti equispaziati in -1, 1
x1=linspace(-1,1);

% Vettore contenente i valori della funzione nei punti x1
y1=feval(fun,x1);

% Vettore dei coefficienti del polinomio interpolatore
p=polyfit(x,y,pol);

% Valuta il polinomio interpolante nei punti x1
f=polyval(p,x1);
```

```
% -----  
% Plot dei risultati  
% -----  
  
hold on  
  
zoom on  
  
% Plot dei punti della funzione di Runge  
plot(x,y,'r*')  
  
% Plot della funzione di Runge  
plot(x1,y1,'k')  
  
% Plot dei punti polinomio interpolante nei  
% punti equispaziati x1 della funzione di Runge  
plot(x1,f,'g')  
  
hold off
```




```
% Esercitazione Fitting
%
% Programma elaborato da
%
% Giovanni DI CECCA & Virginia BELLINO
% 50 / 887 408 / 466
%
% http://www.dicecca.net
%
% Punto 1.ii

clear % Pulisci memoria

clc % Pulisci schermo

% -----
% Dati ingresso
% -----

x=[1,2,4,11,25,45,47,51,58,63,74,105,125,145,170,180,190,204,215,228]

y=[2,3,4,6,9,12,13,15,20,23,25,24,23,21,19,20,24,33,39,40]

% -----
% Costruzione del polinomio interpolante
% -----

% Valutazione dei punti con un polinomio di grado 19, in quanto
% si stanno utilizzando 20 punti x^0 compreso
p=polyfit(x,y,19)

% Punti equispaziati calcolati in base agli estremi della x
ls=linspace(1,228)

% Valutazione del polinomio interpolante
f=polyval(p,ls)

% -----
% Plot dei risultati
% -----

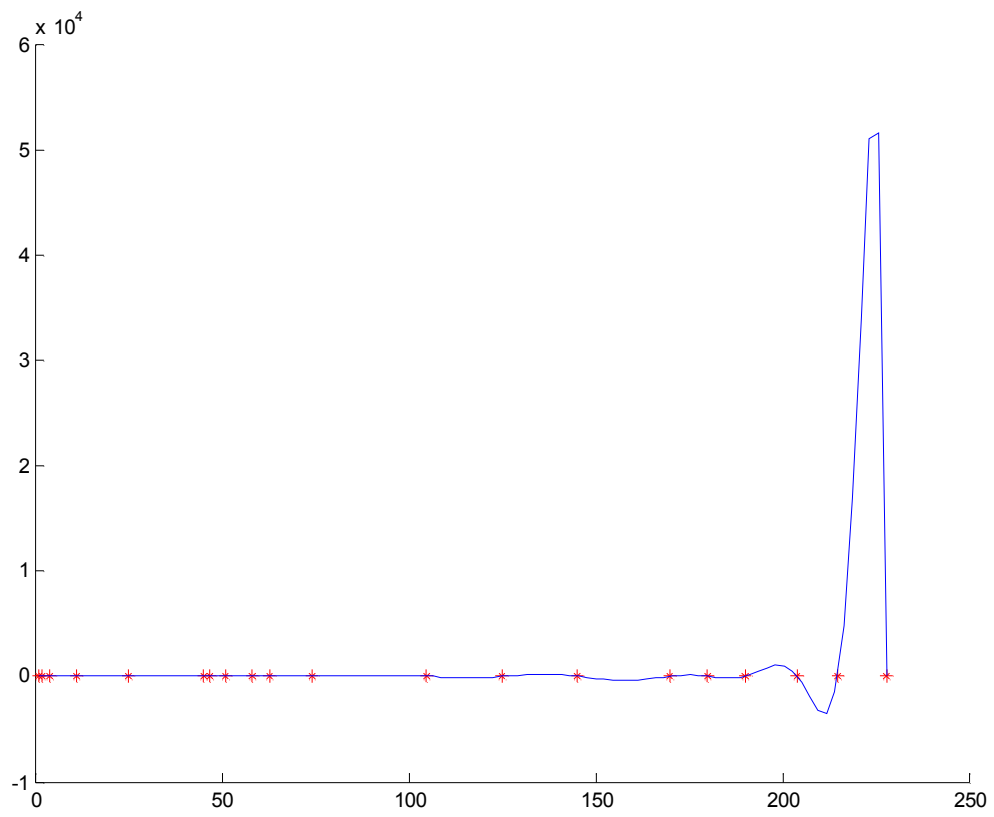
hold on

% Plot dei punti x et y
plot(x,y,'r*')

% Plot del polinomio interpolato
```

```
plot(ls,f,'b')
```

```
hold off
```



```
%          Esercitazione Fitting
%
%          Programma elaborato da
%
%          Giovanni DI CECCA & Virginia BELLINO
%          50 / 887          408 / 466
%
%          http://www.dicecca.net
%
% Punto 2.i

clear % Pulisci memoria

clc % Pulisci schermo

pol=10 % Polinomio di grado 10

% -----
% Costruzione della Funzione di Runge
% Costruzione dei nodi di Chebychev
% -----

% Inserisci i nodi interpolazione
x=linspace(-1,1);

% Nodi di Chebychev su cui costruiamo il polinomio
cheb=cos((pi*(2*[0:pol]))/(2*pol));

% Funzione di Runge
fun=inline(' (1+25*x.^2).^(-1) ');

% Vettore contenente le valutazioni della Funzione di Runge
% nei nodi x
y=feval(fun,x);

% Vettore contenente le valutazioni della funzione di Runge
% sui nodi di Chebychev
y_cheb=feval(fun,cheb);

% -----
% Costruzione del polinomio interpolante
% -----

% Vettore dei coefficienti del polinomio interpolatore
% dei nodi di Chebychev
```

```
p=polyfit(cheb,y_cheb,pol);

% Valutazione dei coefficienti del polinomio interpolatore
% nei punti x di interpolazione
f=polyval(p,x);

% -----
% Plot dei risultati
% -----

hold on

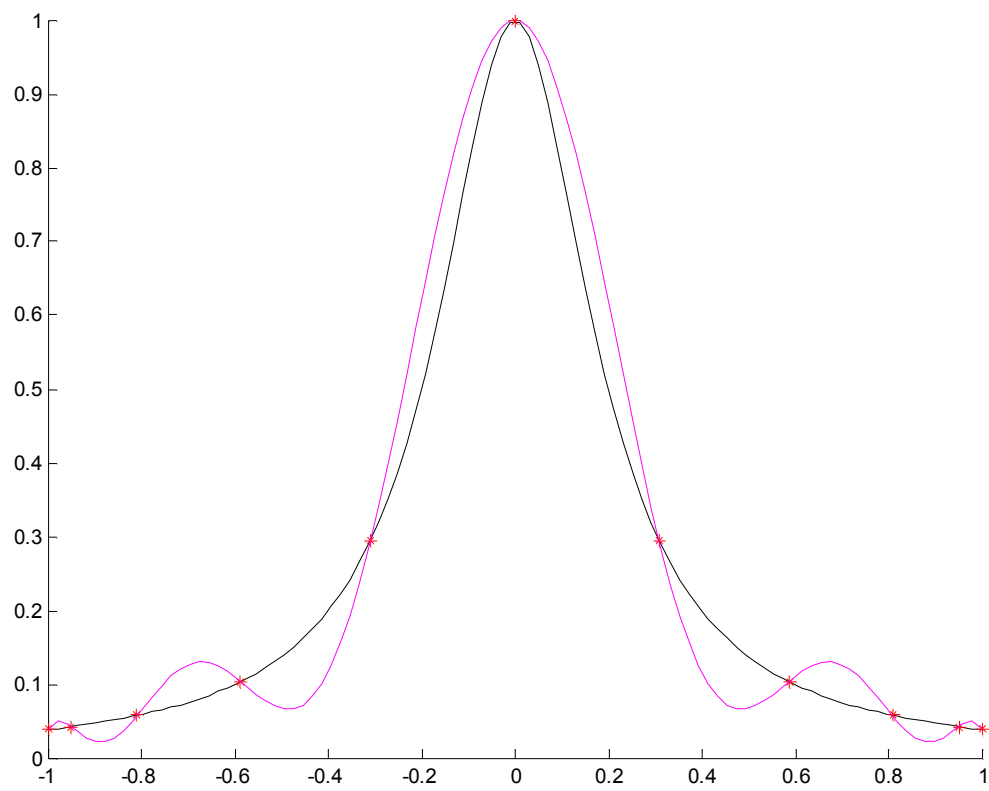
zoom on

% Plot dei punti della Funzione di Runge
plot (x,y,'k')

% Plot dei punti di interpolazione nei
% nodi di Chebychev
plot (cheb,y_cheb,'r*')

% Plot del polinomio interpolante nei nodi di
% interpolazione x et f
plot (x,f,'m')

hold off
```



```
%          Esercitazione Fitting
%
%          Programma elaborato da
%
%          Giovanni DI CECCA & Virginia BELLINO
%          50 / 887          408 / 466
%
%          http://www.dicecca.net
%
% Punto 2.i

clear % Pulisci memoria

clc % Pulisci schermo

pol=20 % Polinomio di grado 20

% -----
% Costruzione della Funzione di Runge
% Costruzione dei nodi di Chebychev
% -----

% Inserisci i nodi interpolazione
x=linspace(-1,1);

% Nodi di Chebychev su cui costruiamo il polinomio
cheb=cos((pi*(2*[0:pol]))/(2*pol));

% Funzione di Runge
fun=inline(' (1+25*x.^2).^(-1) ');

% Vettore contenente le valutazioni della Funzione di Runge
% nei nodi x
y=feval(fun,x);

% Vettore contenente le valutazioni della funzione di Runge
% sui nodi di Chebychev
y_cheb=feval(fun,cheb);

% -----
% Costruzione del polinomio interpolante
% -----

% Vettore dei coefficienti del polinomio interpolatore
% dei nodi di Chebychev
```

```
p=polyfit(cheb,y_cheb,pol);

% Valutazione dei coefficienti del polinomio interpolatore
% nei punti x di interpolazione
f=polyval(p,x);

% -----
% Plot dei risultati
% -----

hold on

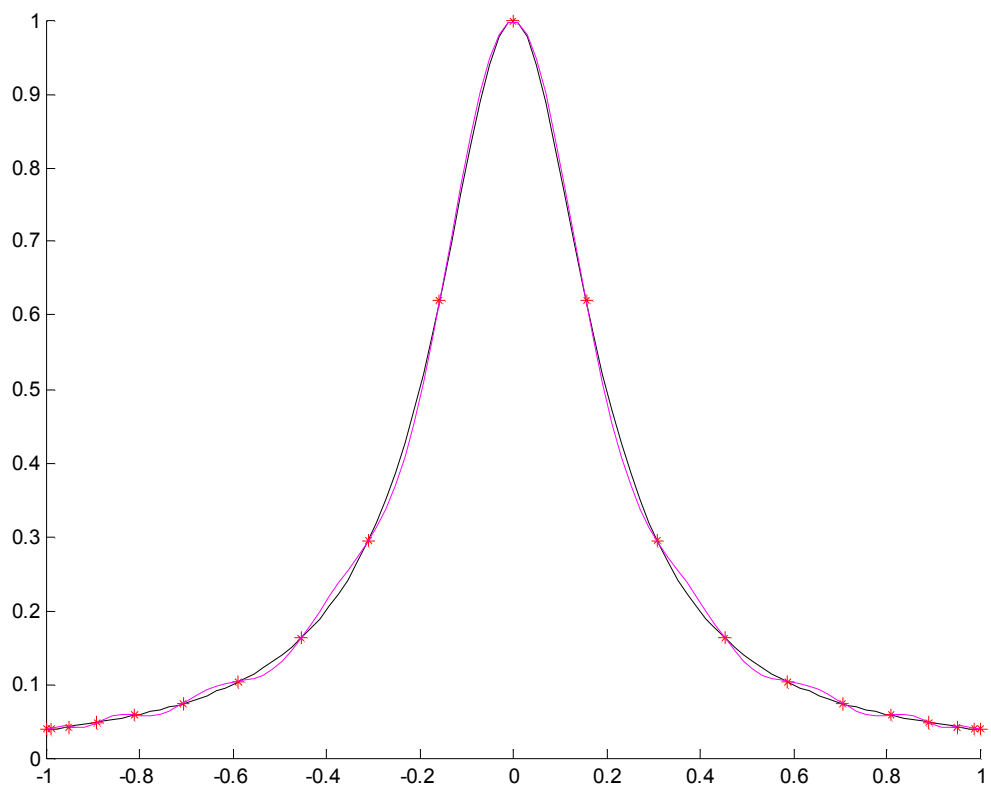
zoom on

% Plot dei punti della Funzione di Runge
plot (x,y,'k')

% Plot dei punti di interpolazione nei
% nodi di Chebychev
plot (cheb,y_cheb,'r*')

% Plot del polinomio interpolante nei nodi di
% interpolazione x et f
plot (x,f,'m')

hold off
```




```
%          Esercitazione Fitting
%
%          Programma elaborato da
%
%          Giovanni DI CECCA & Virginia BELLINO
%          50 / 887          408 / 466
%
%          http://www.dicecca.net
%
% Punto 2.ii
%
% spline

% Pulisci memoria
clear

% Pulisci video
clc

% -----
% Dati ingresso
% -----

x=[1,2,4,11,25,45,47,51,58,63,74,105,125,145,170,180,190,204,215,228]

y=[2,3,4,6,9,12,13,15,20,23,25,24,23,21,19,20,24,33,39,40]

% -----
% Costruzione della Spline cubica
% -----

% Punti equispaziati calcolati in base agli estremi della x
ls=linspace(1,228)

% Funzione di spline per disegnare la figura
sf=spline(x,y,ls);

% -----
% Plot dei risultati
% -----

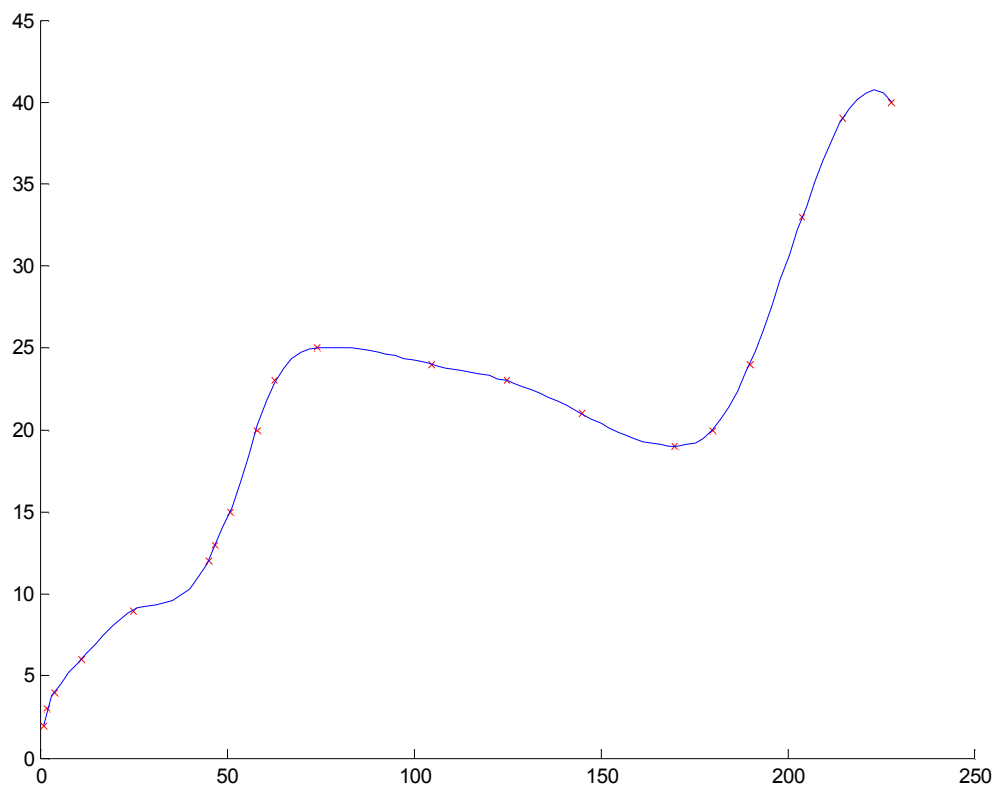
hold on

% Plot dei punti
plot(x,y,'rx')

% Plot della spline
```

```
plot(ls,sf,'b')
```

```
hold off
```





Osservazioni

The word "Osservazioni" is rendered in a large, elegant serif typeface. Each letter is filled with a different color from a rainbow spectrum, starting with magenta for the 'O', transitioning through red, orange, yellow, green, and blue, and ending with purple for the final 'i'. The letters are slightly shadowed, giving them a three-dimensional appearance as if they are floating above a surface.

- **Funzione di Runge**

Sia $f(x)$ una funzione definita in un generico intervallo $[a,b]$ e sia $p(x)$ un polinomio interpolante costruito su $n + 1$ nodi equispaziati appartenenti al medesimo intervallo.

In generale, si potrebbe pensare che, se f è sufficientemente regolare in $[a,b]$, al crescere del grado n il polinomio $p(x)$ approssimi sempre meglio la funzione $f(x)$.

In realtà, ciò non sempre è vero, anzi, può accadere che la successione dei polinomi interpolanti non converga affatto ad $f(x)$, e questo è proprio quanto accade per la funzione di Runge esaminata nella esercitazione.

Infatti, si può osservare che, aumentando il grado del polinomio interpolante, la successione descritta da $p(x)$ non converge affatto come si può notare soprattutto agli estremi, ove sono presenti dei picchi. Questo accade poiché l'errore commesso usando il polinomio di interpolazione dipende in parte anche dalla scelta del tipo di nodi.

Nel caso specifico, si dimostra che la scelta di nodi equispaziati non è ottimale.

La situazione migliora notevolmente se scegliamo come nodi di interpolazione le **ascisse di Chebyshev**. Si tratta infatti di nodi che minimizzano l'errore, e ciò appare evidente osservando i plot relativi ai files *cheby10* e *cheby20*.

- **Profilo superiore di un aereo**

In questo caso, partendo da un numero moderatamente grosso di nodi, è stato approssimato

l'andamento del profilo superiore di un aereo, dapprima utilizzando un polinomio interpolante (*file aereo.m*), poi ricorrendo ad una spline cubica interpolante (*file splinefly.m*).

Osservando i grafici, si può notare che l'interpolazione polinomiale non è la scelta più ottimale, poiché genera una approssimazione poco attendibile del fenomeno dovuta al fatto che il numero di nodi preso in considerazione è moderatamente grosso.

Utilizzando invece una spline cubica interpolante la situazione migliora notevolmente, poiché questa funzione, racchiudendo in sé le caratteristiche migliori ereditate dalla interpolazione a tratti unite ad una maggiore regolarità, consente di mitigare le oscillazioni, generando così una approssimazione del fenomeno più aderente alla realtà.