Elaborato di

Calcolo Numerico

Anno Accademico 2016/2017

Giovanni *Bindi* - 5530804 - *giovanni.bindi@stud.unifi.it*
Gabriele *Gemmi* - 5602433 - *gabriele.gemmi@stud.unifi.it*

Contents

1 Capitolo 1

1.1 Esercizio 1.1

Per definizione di metodo iterativo convergente si ha che

$$\lim_{k \to +\infty} x_k = x^*$$

Supponendo la funzione $\Phi(x_n)$ uniformemente continua vale

$$\lim_{k \to +\infty} \Phi(x_k) = x^* = \Phi(\lim_{k \to +\infty} x_k) = x^*$$

Per definizione é $\Phi(x_n) = \ x_{k+1}$ e quindi

$$\lim_{k \to +\infty} \Phi(x_k) = \lim_{k \to +\infty} x_{k+1} = x^*$$

Da cui otteniamo che x^* e' un punto fisso per la funzione $\Phi(x_n)$, ovvero che $x^* = \Phi(x^*)$.

1.2 Esercizio 1.2

Dal momento che le variabili intere di 2 byte in Fortran vengono gestite in Modulo e Segno, la variabile `n`, inizializzata con

```
integer*2 n
```

varia tra $-2^{15} \leq n \leq 2^{15} - 1$ e quindi tra $-32768 \leq n \leq 32767$.
Andando quindi ad eseguire la somma $(32767 + 1)_{10} = (0111111111111111 + 1)_{2,MS} =$
$= (1111111111111111)_{2,MS} = (-327628)_{10}$

1.3 Esercizio 1.3

Per definizione si ha che la precisione di macchina u per arrotondamento e' data da $u = \frac{1}{2}b^{1-m}$.
Se $b = 8, m = 5$ si ha $u = \frac{1}{2} \cdot 8^{-4} = 1,2207031 \cdot 10^{-4}$

1.4 Esercizio 1.4

Il seguente codice

```
format long e;

x_0 = 0;

v = zeros(1);
p = zeros(1);

l = linspace(1,12,12);

for i=1:numel(l)
    v(i) = power(10,-i);
end

for j=1:numel(v)
    p(j) = psi(x_0,v(j));
end

plot(l,p);

function p = psi(x,j)
    %    This function takes in input
    %    two real values x,j and returns
```

```
    %    the difference quotient
    %    psi(x) = ( e^(x+j) − e^x )/j
    p = (exp(x+j) − exp(x))/j;
end
```

restituisce questo output:

i	x_i
1	1.05170918075648
2	1.00501670841680
3	1.00050016670838
4	1.00005000166714
5	1.00000500000697
6	1.00000049996218
7	1.00000004943368
8	0.999999993922529
9	1.00000008274037
10	1.00000008274037
11	1.00000008274037
12	1.00008890058234

Plottando il contenuto della tabella su di un piano XY abbiamo che
Si nota dal grafo 1 come al crescere di i, quindi al diminuire di h, la precisione con cui viene approssimato f'(0) aumenta.

1.5 Esercizio 1.5

Sia $f(x)$ una funzione sufficentemente regolare e $h > 0$ una quantità "piccola"

$$\frac{f(x_0+h)-f(x_0+h)}{2h} = f'(x_0) + O(h^2)$$
$$\frac{f(x_0+h)-2f(x_0)-f(x_0+h)}{h^2} = f''(x_0) + O(h^2)$$

Sviluppiamo la funzione $f(x)$ mediante il polinomio di Taylor al secondo ordine

$$f(x) = f(x_0) + (x-x_0)f'(x_0) + \frac{(x-x_0)^2}{2}f''(x_0) + O((x-x_0)^3)$$

Sostituiamo

$$x = (x_0+h) \text{ e } x = (x_0-h)$$
$$f((x_0+h) = f(x_0) + hf'(x_0) + \frac{h^2}{2}f''(x_0) + O(h^3)$$
$$f((x_0-h) = f(x_0) - hf'(x_0) + \frac{h^2}{2}f''(x_0) + O(h^3)$$

Risostituendo nel rapporto incrementale dell'ipotesi otteniamo:

$$\frac{f(x_0)+hf'(x_0)+\frac{h^2}{2}f''(x_0)+O(h^3)-f(x_0)-hf'(x_0)+\frac{h^2}{2}f''(x_0)+O(h^3)}{2h} = \frac{2hf'(x_0)+O(h^3)}{2h} = f'(x_0)+O(h^2)$$

Per la seconda uguaglianza dell'ipotesi basterà applicare lo stesso procedimento con uno sviluppo di taylor al 3° ordine.

1.6 Esercizio 1.6

Il seguente codice `MATLAB`

```
format long

x = [2,1.5];
y = [];
```

```
rad = sqrt(2);

for i = 2:10
   x(i+1) =  ((x(i)*x(i-1) +2)/(x(i) + x(i-1)));
end

for i=1:10
    y(i) = x(i) - rad;
end


i = [1:10];
semilogy(i,y(i));
```

produce come output grafico 2:
Con $i = 6$ l'errore di convergenza $\epsilon \approx 10^{-10}$. Per valori piú grandi viene approssimato a 0 dal calcolatore. Possiamo quindi prendere $n > 6$ per ottenere un errore di convergenza $\epsilon < 10^{-12}$

1.7 Esercizio 1.7

Abbiamo che $u = \frac{1}{2}b^{1-m} \approx 4.66 \cdot 10^{-10}$, supponendo che la base sia $b = 2$ otteniamo

$$m = 1 - log_2(4.66 \cdot 10^{-10}) \approx 31.99$$

Quindi necessitiamo di 32 cifre per la mantissa.

1.8 Esercizio 1.8

Supponendo $b = 10$ si ha, per entrambi i casi

- troncamento : $m = 1 - log_{10}u \approx -log_{10}u$
- arrotondamento : $m = 1 - log_1 02u = 1 - log_{10}2 - log_1 0u \approx -log_{10}u$

1.9 Esercizio 1.9

```
x=0; delta = 1/10;
while x ~= 1, x = x+delta, end
```

Il valore di delta è uguale a $\frac{1}{10}$. La rappresentazione binaria di questo numero però non è esatta. Si tratta di una rappresentazione periodica e quindi, in decimale, sarà circa 0.0999.
prendendo quindi $delta \approx 0.9$ vedremo che per $i = 10$ $x \approx 0.999$.Mentre, per $i = 11$ $x \approx 1,0989$.
Per questo motivo la condizione x=1 non si avvererà mai e il programma non terminerà.

1.10 Esercizio 1.10

I problemi del calcolo della radice e dell'elevamento a potenza sono sempre ben condizionati (k=1/2 e k=2). Il problema critico è l'overflowper la somma. Per evitare questa criticità proponiamo questa soluzione: Troviamo quindi il massimo dei due addendi:

$$m = max\{|x|, |y|\}$$
$$\sqrt{x^2 + y^2}$$

Dividiamo entrambi i membri per il massimo

$$\sqrt{m^2 * (\frac{x}{m}^2 + \frac{y}{m}^2)}$$

Portiamo fuori il massimo dalla radice

$$m * \sqrt{\frac{x}{m}^2 + \frac{y}{m}^2}$$

In questo modo eviteremo la problematica dell'overflow per la somma e il problema resterà sempre ben condizionato.

1.11 Esercizio 1.11

L'espressione aritmetica dei due algoritmi è:

- 1) (x+y)+z = x+y+z
- 2) x+(y+z) = x+y+z

Ne consegue che sono equivalenti In aritmetica finita, invece, avremo:

- 1) $(x \oplus y) \oplus z = fl(fl(fl(x) + fl(y)) + fl(z)) =$
  $= ((x(1+\varepsilon_x) + y(1+\varepsilon_y))(1+\varepsilon_A) + z(1+\varepsilon_z))(1+\varepsilon_B) =$

  $$\varepsilon_R = \frac{(x(1+\varepsilon_x)(1+\varepsilon_A)(1+\varepsilon_B) + y(1+\varepsilon_y)(1+\varepsilon_A)(1+\varepsilon_B) + z(1+\varepsilon_z))(1+\varepsilon_B) - x - y - z}{z+y+z}$$

  prendo $\varepsilon_M = max\{\varepsilon_x, \varepsilon_y, \varepsilon_z, \varepsilon_A, \varepsilon_B\}$

  $$\varepsilon_R \leq \frac{x(1+\varepsilon_M)^3 + y(1+\varepsilon_M)^3 + z(1+\varepsilon_M)^2 - x - y - z}{z+y+z} =$$

  $$= \frac{x(3\varepsilon_M + 3\varepsilon_M^2 + \varepsilon_M^3) + y(3\varepsilon_M + 3\varepsilon_M^2 + \varepsilon_M^3) + z(2\varepsilon_M + \varepsilon_M^2)}{z+y+z}$$

  Sapendo che $\varepsilon_M \leq 1$ posso affermare che $\varepsilon_M \geq \varepsilon_M^2 \geq \varepsilon_M^3$ ed effettuare un'altra minorazione

  $$\varepsilon_R \leq \frac{7x\varepsilon_M + 7y\varepsilon_M + 3z\varepsilon_M}{x+y+z} = \varepsilon_M(3 + 4\frac{x+y}{x+y+z})$$

- 2) $x \oplus (y \oplus z) = fl(fl(x) + fl(fl(y) + fl(z)))$
  Il procedimento sarà analogo a quello visto prima con lo scambio degli addendi al nominatore della frazione

  $$\varepsilon_R \leq \frac{7z\varepsilon_M + 7y\varepsilon_M + 3x\varepsilon_M}{x+y+z} = \varepsilon_M(3 + 4\frac{z+y}{x+y+z})$$

1.12 Esercizio 1.12

Mostriamo che $k = \frac{1}{2}$ è il numero di condizionamento di: $f(x) = \sqrt{x}$

$$f'(x) = \frac{1}{2\sqrt{x}}$$

$$k \equiv |f'(x) \cdot \frac{x}{y}| = |\frac{1}{2\sqrt{x}} \cdot \frac{x}{\sqrt{x}}| = |\frac{1}{2} \cdot \frac{x}{x}| = |\frac{1}{2}|$$

1.13 Esercizio 1.13

Il seguente codice plotta la funzione3 Il motivo per cui $f(x)$ in $x = \frac{4}{3}$ é pari ad un valore finito risiede nel fatto che $\frac{4}{3}$ é un numero razionale periodico. Il calcolatore memorizzerà $fl(\frac{4}{3})$ con un approssimazione determinata dalla precisione macchina in uso. Calcolando infatti l'espressione mostrata a riga 12 del listato abbiamo questo output:
2.220446049250313e-16
Questo valore analiticamente dovrebbe essere 0, ma dato che la memorizzazione di $\frac{4}{3}$ non é precisa, il valore della funzione nel punto sarà un numero finito.

2 Capitolo 2

Ecco le function da noi utilizzate nel corso del capitolo :

- Medoto di bisezione:

```
function [sol,nit]=bisezione(f,a,b,tolx,maxit);
    nit=maxit;
    j=0;
    if(subs(f,a)*subs(f,b)>0)
        disp('Il metodo di bisezione non puo essere utilizzato in questo caso!')
    else
        while(j<=maxit)
            sol=(a+b)/2;
            ff=subs(f,sol);
            if(abs(ff)<=tolx)
                nit=j;
                break;
        end
        fa=subs(f,a);
        fm=subs(f,sol);
        if(fa*fm<=0)
            b=sol;
        else
            a=sol;
        end
        j=j+1;
    end
end
```

- Metodo di Newton modificato :

```
function [y, i] = NewtonMod(f, df, m, x0, imax, tol, output)
    format long;
    i = 0;
    x = x0;
    vai=1;
    while((i < imax) && vai)
        i = i+1;
        fx = feval(f, x0);
        dfx = feval(df, x0);
        if(dfx ~= 0)
            x = x0 - m * fx / dfx;
        else
            break;
        end
        if(abs(x-x0)<tol)
            vai = 0;
        end
        x0 = x;
    end
    if(output)
        if(vai)
            disp('Impossibile calcolare la tolleranza richiesta nel numero di iter')
                ;
        else
           disp(i);
        end
    end
```

```
    y = x;
end
```

- Metodo di accelerazione di Aitken :

```
function y = Aitken( f, df, x0, imax, tol )
    format long;
    i = 0;
    vai=1;

    while((i < imax) && vai)
        x1 = NewtonMod(f, df, 1, x0, 1, tol, 0);
        x2 = NewtonMod(f, df, 1, x1, 1, tol, 0);
        i = i+1;
        if((x0 - 2*x1 +x2) == 0)
            disp('Errore, impossibile dividere per 0');
            vai = 0;
            break;
        end
        x = (x2*x0 - x1^2)/(x0 - 2*x1 +x2);
        if(abs(x-x0)<tol)
            vai = 0;
        end
        x0 = x;
    end
    if(vai)
        disp('Impossibile calcolare la tolleranza richiesta nel numero di iter');
        disp(i);
    end
    y = x;
end
```

- SecNSqrt.m

```
function y = SecNSqrt(n, alpha, x0, imax, tol)
    format long e
    x1 = (x0 + alpha/x0)/2;
    x = (f(x1,n, alpha) * x0 - f(x0,n, alpha)*x1 ) / (f(x1, n, alpha) - f(x0, n,
        alpha));
    i = 1;
    while(i < imax) && (abs(x-x0)>tol)
        x0=x1;
        x1=x;
        i = i+1;
        x = (f(x1,n, alpha) * x0 - f(x0,n, alpha)*x1 ) / (f(x1, n, alpha) - f(x0, n,
             alpha));
    end
    y = x;
end


function y = f(x, n, alpha)
    y = (x ^ n) - alpha;
end
```

2.1 Esercizio 2.1

```
x_0 = 3;
alpha = 3;
z = NewtonSqrt(alpha, x_0, 200, 0.0001)
```

La tabella delle approssimazioni è la seguente:

i	x_i
1	1.750000000000000
2	1.732142857142857
3	1.732050810014727

2.2 Esercizio 2.2

```
x_0 = 3;
alpha = 3;
z3 = NewtonNSqrt(3, alpha, x_0, 200, 0.0001);
z4 = NewtonNSqrt(4, alpha, x_0, 200, 0.0001);
z5 = NewtonNSqrt(5, alpha, x_0, 200, 0.0001);
```

2.3 Esercizio 2.3

Versione MATLAB

```
%Definire una procedura iterativa basata sul metodo di Newton per
%determinare sqrt(alpha)

x_0 = 3;
alpha = x_0;
n= 2;

z = SecNSqrt(n, alpha, x_0, 200, 0.0001);
```

Verisione Python

```
import numpy as np
import math

def secanti(n,alpha,x0,imax,tolx):
        print x0 =  +str(x0)
        x = ((x0+alpha/x0)/2.0)
        print x1 =  +str(x)
        i = 1
        while ( (i<imax) & (math.fabs(x-x0)>tolx)):
                i = i+1
                x1 = (f(x,n,alpha)*x0 - x*f(x0,n,alpha))/(f(x,n,alpha) - f(x0,n,alpha))
                x0 = x
                x=x1;
                print x+str(i)+ = +str(x)

def f(x,n,alpha):
        return pow(x,n) - alpha

secanti(2.0,2.0,2.0,7,0.000001)
```

2.4 Esercizio 2.4

```
mf = @(x) (x-pi)^10;
dmf = @(x) 10*(x-pi)^9;
```

```
y1 = NewtonMod(mf, dmf, 1, 3, 50, 0.5, 0);
y2 = NewtonMod(mf, dmf, 10, 3, 50, 0.5, 0);
y3 = Aitken(mf, dmf, 3, 50, 0.5);


mf2 = @(x) ((x-pi)^10)*(exp(1)^(2*x));
dmf2 = @(x) (5+x-pi)*(x-pi)^9*2*exp(1)^(2*x);

y21 = NewtonMod(mf2, dmf2, 1, 3, 50, 0.5, 0);
y22 = NewtonMod(mf2, dmf2, 10, 3, 50, 0.5, 0);
y23 = Aitken(mf2, dmf2, 3, 50, 0.5);
```

Questo codice esegue i metodi di Newton, Newton modificato e Aitken per le funzioni date. Questi sono gli output dei tre metodi numerici:

Metodo	$f_1(x)$	$f_2(x)$
NewtonMod (m=1)	3.014159265358980	3.014571920744193
NewtonMod (m=10)	3.141592653589793	3.145719207441934
Aitken	3.141592653589755	3.137995613065127

2.5 Esercizio 2.5

Il metodo di bisezione é applicabile se la funzione f é

1. continua nell'intervallo $[a, b]$
2. tale che $f(a)f(b) < 0$

Dal momento che lo zero delle funzioni $f_1(x) = (x - \pi)^{10}$ e $f_2(x) = e^{2x}(x - \pi)^{10}$ risulta essere in $x = \pi$, ultilizzare come punto iniziale $x_0 = 5 > \pi$ non porterebbe chiaramente alla determinazione dello zero ancor prima di valutare la regolaritá della funzione. Analizzando poi le due f si nota subito che

$$\nexists x | f_1(x) < 0, \forall x \in \mathbb{R},$$

$$\nexists x | f_2(x) < 0, \forall x \in \mathbb{R}.$$

Dal momento che queste sono funzioni esponeziali con minimo $x_{min} = \pi$, risulta evidente come il suddetto metodo non sia applicabile.

2.6 Esercizio 2.6

```
format long

myf = @(x) (1-x-(1+cos(10*x)/2)*sin(x));
df = @(x) (5*sin(x)*sin(10*x)-cos(x)*(cos(10*x)/2 + 1)-1);

x0 = 0;
x1 = 1;
tol = logspace(-1,-10,10);
disp(tol);
imax = 50;
y = zeros(4,10);
for i=1:10
    [temp, y(1,i)] = NewtonMod(myf,df,1,x0,imax,tol(i), 0);
    [temp, y(2,i)] = secanti(myf,x0,x1,tol(i),imax);
    [temp, y(3,i)] = corde(myf,feval(df,x0),x0,tol(i),imax);
end
```

I dati generati dall'esecuzione del codice sono esposti nella seguente tabella

tol_x	Newton	Secanti	Tangenti
10^{-1}	4	13	2
10^{-2}	5	15	8
10^{-3}	5	15	15
10^{-4}	5	16	22
10^{-5}	6	16	28
10^{-6}	6	16	35
10^{-7}	6	17	42
10^{-8}	6	17	48
10^{-9}	6	17	50
10^{-10}	7	17	50

2.7 Esercizio 2.7

```
format long

myf = @(x) (1-x-(1+cos(10*x)/2)*sin(x));
df = @(x) (5*sin(x)*sin(10*x)-cos(x)*(cos(10*x)/2 + 1)-1);

x0 = 0;
x1 = 1;
tol = logspace(-1,-10,10);
disp(tol);
imax = 50;
y = zeros(4,10);
for i=1:10
    [temp, y(1,i)] = NewtonMod(myf,df,1,x0,imax,tol(i), 0);
    [temp, y(2,i)] = secanti(myf,x0,x1,tol(i),imax);
    [temp, y(3,i)] = corde(myf,feval(df,x0),x0,tol(i),imax);
    [temp, y(4,i)] = bisezione(myf,x0,x1,tol(i),imax);
end

plot(y')
```

I dati generati dall'esecuzione del codice sono esposti nella seguente tabella

n	tol_x	y
0	10^{-1}	0.500000000000000
7	10^{-2}	0.488281250000000
10	10^{-3}	0.488769531250000
13	10^{-4}	0.488952636718750
16	10^{-5}	0.488945007324219
20	10^{-6}	0.488943576812744
21	10^{-7}	0.488943815231323
26	10^{-8}	0.488943792879581
30	10^{-9}	0.488943794276565
32	10^{-10}	0.488943794392981

Per la comparazione vedi figura 4

2.8 Esercizio 2.8

```
myfunct = @(x) (x-pi)^(10*x);
dfdx =  @(x) 10*((x-pi)^(10*x))*(x/(x-pi)+log(x-pi));
x0 = 0;

y = NewtonMod(myfunct,dfdx, 1, x0, 5, 0.01, 1);
```

Essendo

$$f'(x) = 10(x-\pi)^{10x}(\frac{x}{x-\pi} + \ln(x-\pi))$$

E dato che quest'ultima contiene $ln(x - \pi)$, il cui dominio é $\{\forall x \in \mathbb{R} | x > \pi\}$, il metodo non potrà convergere per valori di $x_0 \leq \pi$

3 Capitolo 3

3.1 Esercizio 3.1

Una matrice $L \in M_{n\times n}$ si dice triangolare inferiore se $l_{i,j} = 0, \forall i < j$ con $i, j = 1...n$ ed $l_{i,j} \in L$.
Date due matrici triangolari inferiori $L, K \in M_{n\times n}$ la loro somma sará di nuovo una matrice traingolare inferiore $N \in M_{n\times n}$:

$$\forall i < j,\ \ l_{i,j} + k_{i,j} = 0 + 0 = 0.$$

Una matrice $U \in M_{n\times n}$ si dice triangolare superiore se $u_{i,j} = 0, \forall i > j$ con $i, j = 1...n$ ed $u_{i,j} \in U$. Date due matrici triangolari superiori $U, W \in M_{n\times n}$ il loro prodotto $Z \in M_{n\times n}$:

$$z_{i,j} = \sum_{k=1}^{n} u_{i,k} w_{k,j}, \qquad\qquad \forall i, j \in [1, .., n].$$

sará di nuovo una matrice triangolare superiore, dal momento che

$$z_{i,j} = \sum_{k=1}^{n} u_{i,k} w_{k,j} = \underbrace{\sum_{k=1}^{i-1} u_{i,k} w_{k,j}}_{=0 \text{ per k<i}} + \sum_{k=i}^{n} u_{i,k} w_{k,j} = \sum_{k=i}^{n} u_{i,k} w_{k,j}.$$

Dal momento che in quest'ultimo termine abbiamo $k \geq i$ se $i > j$ allora $k > j$, da cui la definizione di matrice triangolare superiore.

3.2 Esercizio 3.2

Se $U, W \in M_{n\times n}$ sono matrici triangolari superiori a diagonale unitaria si ha che gli elementi diagonali della matrice $Z = UW$ saranno calcolati come

$$z_{i,i} = \sum_{k=i}^{i} u_{i,i} w_{i,i} = 1 \cdots 1 = 1.$$

Quindi, riallacciandosi alla dimostrazione dell'esercizio **3.1** si ha che la matrice Z é triangolare superiore a diagonale unitaria.

3.3 Esercizio 3.3

Sia $A \in M_{n\times n}$ una matrice triangolare superiore non singolare.
Se A é invertibile allora puó essere scritta come $A = D(I_n + U)$ dove D é una matrice diagonale per cui vale $diag(D) = diag(A)$, I_n é la matrice identitá ed U é una matrice triangolare strettamente superiore ($diag(U) = 0$).
Per le proprietá delle matrici triangolari strettamente superiori si ha che $U^n = 0$ (ovvero che la matrice U é nilpotente), mentre per le proprietá delle matrici diagonali si ha che D^{-1} é ancora una matrice diagonale.
Volendo provare che $A^{-1} = (I_n + U)^{-1} D^{-1}$ espandiamo in serie $(I_n + U)^{-1}$:

$$(I_n + U)^{-1} = I_n - U + U^2 - ... + (-1)^{n-1} U^{n-1}.(1)$$

Essendo questa una serie di somme e prodotti di matrici triangolari superiori la matrice inversa A^{-1} sará in generale una matrice triangolare superiore. Nel caso la matrice A sia a diagonale unitaria anche la sua inversa avrá diagonale unitaria, dal momento che $D = I_n$, la cui inversa é ancora I_n.

3.4 Esercizio 3.4

L'eliminazione nella prima colonna richiede n somme ed n prodotti per $n - 1$ righe, quindi in totale $(n + n)(n - 1) = 2n(n - 1)$ `flops`. L'eliminazione della seconda richiede $n - 1$ somme ed $n - 1$ prodotti per $n - 2$ righe, quindi in totale $[(n - 1) + (n - 1)](n - 2) = 2(n - 1)(n - 2)$ `flops`.
Procedendo per induzione si ha che il numero totale di operazioni é

$$\sum_{i=0}^{n} 2(n - i)(n - i + 1). \qquad\qquad (1)$$

Operando la sostituzione $y \doteq n - i + 1$ si ha che la (1) diviene :

$$2\sum_{j=0}^{n-1} j(j-1) = 2\sum_{j=0}^{n-1} j^2 + j = 2(\frac{1}{3}n^3 - \frac{1}{3}n)$$

Asintoticamente quindi proprio $\frac{2}{3}n^3$ `flops`.

3.5 Esercizio 3.5

```
function [L,U,P]=LUP(A)
[m,n]=size(A);
if m~=n
    error('matrice non quadrata');
end
L=eye(n);
P=eye(n);
U=A;
for k=1:n
    [pivot, m]=max(abs(U(k:n,k)));
    if pivot==0
        error('Errore: Matrice singolare');
    end
    m=m+k-1;
        if m~=k
        % scambio le righe m e k
        U([k,m], :) = U([m, k], :);
        P([k,m], :) = P([m, k], :);
        if k >= 2
            L([k,m],1:k-1) = L([m,k],1:k-1);
        end
    end
    L(k+1:n,k)=U(k+1:n,k)/U(k,k);
    U(k+1:n,:)=U(k+1:n,:)-L(k+1:n,k)*U(k,:);
end
```

3.6 Esercizio 3.6

```
function [b] = solveLinearLUP(L, U, P, b)
    b = TriangolareInf(L,P*b);
    b = TriangolareSup(U,b);
end
```

3.7 Esercizio 3.7

Sia $\mathbf{v} \in \mathbb{R}^n | \mathbf{v} \neq \mathbf{0}$ e $A \in M_{n\times n}$. A si dice sdp (simmetrica definita positiva) se é simmetrica (A=A^T) e se $\mathbf{v}^T A\mathbf{v} > 0$. Equivalentemente una matrice si dice sdp se é simmetrica ed i suoi autovalori sono > 0. Inoltre una matrice $B \in M_{n\times n}$ si dice nonsingolare se $\det B \neq 0$.
Dal momento che il polinomio caratteristico é invariante per similitudine le matrici quadrate A ed A^T hanno gli stessi autovalori. Inoltre le matrici A^TA ed AA^T sono simmetriche dal momento che:

$$(AA^T)^T = (A^T)^TA^T = AA^T,$$

$$(A^TA)^T = A^T(A^T)^T = A^TA.$$

Vale poi $\det(A^TA) = \det(AA^T) = \det A \det A^T = (\det A)^2$.
Dimostriamo quindi che AA^T e A^TA sono definite positive:

$$\mathbf{v}^TAA^T\mathbf{v} = (A^T\mathbf{v})^T(A^T\mathbf{v}) > 0,$$

$$\mathbf{v}^TA^TA\mathbf{v} = (A\mathbf{v})^T(A\mathbf{v}) > 0.$$

3.8 Esercizio 3.8

Se $A \in M_{m\times n}$ con $m \geq n = rank(A)$ allora diremo che A ha rango massimo.
Questo comporta che la matrice sia invertibile, ovvero che il suo determinante $\det(A) \neq 0$. La matrice é quindi nonsingolare e di conseguenza simmetrica definita positiva, dalla dimostrazione dell'esercizio **3.7**.

3.9 Esercizio 3.9

Si ha ovviamente che

$$A = \frac{1}{2}(A + A^T) + \frac{1}{2}(A - A^T) = \frac{1}{2}A + \frac{1}{2}A^T + \frac{1}{2}A - \frac{1}{2}A^T$$

Definendo $A_s \doteq \frac{1}{2}(A + A^T)$ si mostra come $A_s = A_s^T$ infatti

$$\frac{1}{2}(A + A^T) = [\frac{1}{2}(A + A^T)]^T = \frac{1}{2}(A + A^T)^T = \frac{1}{2}(A^T + (A^T)^T) = \frac{1}{2}(A + A^T)^T.$$

Da cui A_s é detta parte **simmetrica** di A.
Definendo $A_a \doteq \frac{1}{2}(A - A^T)$ si mostra come $A_a = -A_a^T$ infatti

$$\frac{1}{2}(A - A^T) = -\frac{1}{2}(A - A^T)^T = -\frac{1}{2}(A^T - (A^T)^T) = -\frac{1}{2}(A^T - A) = \frac{1}{2}(A - A^T).$$

Da cui A_a é detta parte **antisimmetrica** di A.
Dato poi un generico vettore $\mathbf{x} \in \mathbf{R}^n$ si ha che $\mathbf{x}^T A\mathbf{x} = \mathbf{x}^T A_s\mathbf{x}$ infatti

$$\mathbf{x}^T A\mathbf{x} = \mathbf{x}^T(A_s + A_a)\mathbf{x} = \mathbf{x}^T A_s\mathbf{x} + \mathbf{x}^T A_a\mathbf{x} = \frac{1}{2}(\mathbf{x}^T A\mathbf{x} + \mathbf{x}^T A^T\mathbf{x}) + \frac{1}{2}(\mathbf{x}^T A\mathbf{x} - \mathbf{x}^T A^T\mathbf{x}).$$

Analizzando l'ultimo termine $\frac{1}{2}(\mathbf{x}^T A\mathbf{x} - \mathbf{x}^T A^T\mathbf{x})$ si nota come

$$\frac{1}{2}(\mathbf{x}^T A\mathbf{x} - \mathbf{x}^T A^T\mathbf{x}) = \frac{1}{2}(\mathbf{x}^T A\mathbf{x} - (A\mathbf{x})^T\mathbf{x}) = 0$$

dal momento che, definendo $A\mathbf{x} = \mathbf{y}$ si ha $\mathbf{x}^T\mathbf{y} = \mathbf{y}^T\mathbf{x}$. Da cui la tesi. *symbol*

3.10 Esercizio 3.10

L'algoritmo esegue $i - 1$ somme di due prodotti, una sottrazione ed una divisione per un costo totale di $2(i - 1) + 2$ `flops`. Essendo la matrice triangolare, per ogni colonna eseguirá il calcolo $n - i$ volte.
Quindi

$$\sum_{i=1}^{n} 2i(n-i) = 2(n\sum_{i=1}^{n} i - \sum_{i=1}^{n} i^2) = 2n\frac{n(n+1)}{2} - 2\frac{n(n+1)(2n+1)}{6} = n^3 + n^2 - \frac{2n^3 + 3n^2 + n}{3} = \frac{n^3}{3} - \frac{n}{3} \approx \frac{n^3}{3}.$$

3.11 Esercizio 3.11

```
function [L,D] = fattorizzaLDLt(A)
    [m,n]=size(A);
    if m~=n
        error('la matrice non quadrata')
    end
    if A(1,1)<=0
        error('la matrice non sdp!')
    end
    A(2:n,1)=A(2:n,1)/A(1,1);
    for j=2:n
        v = (A(j,1:(j-1))').*diag(A(1:(j-1),1:(j-1)));
        A(j,j) = A(j,j)-A(j,1:(j-1))*v;
```

```
        if A(j,j)<=0
            error('la matrice non sdp!');
        end
        A((j+1):n,j)=(A((j+1):n,j)−A((j+1):n,1:(j−1))*v)/A(j,j);
    end
    if nargout==1
        L=A;
    else
    for j=1:n
      for i=1:n
           if i==j
                D(i,j) = A(i,j);
                L(i,j) = 1;
         end
           if i>j
                D(i,j) = 0;
                L(i,j) = A(i,j);
           end
            if i<j
                D(i,j) = 0;
                L(i,j) = 0;
           end
       end
    end
    end
end
```

3.12 Esercizio 3.12

```
function [b] = solveLinearLDL(L,D, b)
    b = TriangolareInf(L,b');
    b = diagonale(diag(D),b');
    b = TriangolareSup(L',b');
end
```

3.13 Esercizio 3.13

Il seguente codice

```
A1 = [1,1,1,1;1,2,2,2;1,2,3,3;1,2,3,4]
[L1,D1] = fattorizzaLDLt(A1)
A2 = [1,1,1,1;1,2,2,2;1,2,3,3;1,2,3,2]
[L2,D2] = fattorizzaLDLt(A2)
```

Restituisce il seguente output:

```
A1 =


         1   1   1   1
         1   2   2   2
         1   2   3   3
         1   2   3   4

L1 =


         1   0   0   0
         1   1   0   0
         1   1   1   0
         1   1   1   1
```

```


D1 =

         1  0  0  0
         0  1  0  0
         0  0  1  0
         0  0  0  1

A2 =

         1  1  1  1
         1  2  2  2
         1  2  3  3
         1  2  3  2


Error using es13.>fattorizzaLDLt (line 20) la matrice non sdp!

Error in es13. (line 4) [L2,D2] = fattorizzaLDLt(A2)
```

La prima matrice é fattorizzabile LDL^T e quindi sdp, mentre la seconda no.

3.14 Esercizio 3.14

*Esempio per esercizio **3.5** e **3.6***:

Sia A la matrice da noi scelta per la risoluzione dell'esercizio.

$$A = \begin{pmatrix} 0 & -3 & 8 \\ -1 & 8 & 7 \\ 1 & 3 & 0 \end{pmatrix}$$

Sia b il vettore dei termini noti.

$$b = (3.1416, 1.1618, 2.7183)^T$$

Utilizzando i metodi numerici allegati e risolvendo il sistema lineare

$$Ax = b$$

otteniamo il vettore delle incognite

$$x = (2.469228440366973, 0.083023853211009, 0.423833944954129)^T$$

Quindi il vettore residuo

$$r = Ax - b = 1.0 * 10^{-15}(0.888178419700125, 0, 0)$$

*Esempio per esercizio **3.11** e **3.12***:

Sia A la matrice da noi scelta per la risoluzione dell'esercizio.

$$A = \begin{pmatrix} 14 & 5 & 2 \\ 5 & 8 & 1 \\ 2 & 1 & 4 \end{pmatrix}$$

Sia b il vettore dei termini noti.

$$b = (3.1416, 1.1618, 2.7183)^T$$

Utilizzando i metodi numerici allegati e risolvendo il sistema lineare

$$Ax = b$$

otteniamo il vettore delle incognite

$$x = (0.144645652173913, -0.021765217391304, 0.612693478260870)^T$$

Quindi il vettore residuo

$$r = Ax - b = 1.0 * 10^{-15}(0, -0.444089209850063, 0)$$

```
% 3.5 3.6
A=[0,-3,8;-1,8,7;1,3,0];
[L,U,P] = LUP(A);
b = [3.1416,1.1618,2.7183]';
[x] = solveLinearLUP(L,U,P,b);
r=A*x -b


% 3.11  3.12
A=[14,5,2;5,8,1;2,1,4]
b = [3.1416,1.1618,2.7183]';
[L,D] = fattorizzaLDLt(A)
[x] = solveLinearLUP(L,D*L',eye(3),b)
r=A*x -b
```

3.15 Esercizio 3.15

```
v = [1,1,1,1,1,1,1,1,1];
A = (diag(v*(-100),-1)+eye(10));

norm(A,1)
norm(A,inf)

cond(A,1)
```

Analiticamente abbiamo che $\|A\|_\infty = \|A\|_1 = 101$, come é possibile verificare attraverso l'istruzione `norm` di Matlab. Per quanto riguarda il calcolo del numero di condizionamento $k_\infty(A)$ abbiamo che, andando a calcolare l'inversa, gli elementi della matrice crescono di 2 ordini di grandezza per ogni riga, arrivando ad avere valori prossimi a 10^{18}. Questo implica che $\|A^{-1}\|_\infty > 10^{20}$ per cui possiamo affermare che il problema é malcondizionato. La verifica con l'istruzione `cond` di Matlab restituisce un warning in cui avverte che $RCOND = 9.801980e-21$, confermando quanto appena detto.

3.16 Esercizio 3.16

```
v = [1,1,1,1,1,1,1,1,1];
A = (diag(v*(-100),-1)+eye(10));

b = [1 -99*ones(1,9)]';
c = 0.1*[1 -99*ones(1,9)]';
x = ones(10,1);
y = 0.1*x;

r =A*x -b;
r = A*y -c;

xx(1)=b(1);
for i=2:10
    xx(i)=b(i)+100*xx(i-1);
end
xx=xx(:)
yy(1)=c(1);
```

```

for i=2:10
    yy(i)=c(i)+100*yy(i-1)
end
yy=yy(:)
```

3.17 Esercizio 3.17

```
function A = QRdecomp(A)
    [m,n] = size(A);
    for i=1:n
        alpha = norm(A(i:m, i));
        if alpha==0
            error('La matrice A ha rango non massimo')
        end
        if A(i,i)>=0
            alpha = -alpha;
        end
        v = A(i,i) - alpha;
        A(i,i) = alpha;
        A(i+1:m,i) = A(i+1:m,i)/v;
        beta = -v/alpha;
        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
end
```

3.18 Esercizio 3.18

```
function [x] = solveQR( A, b )
    [m,n] = size(A);
    Qt = eye(m);
    for i=1:n
        Qt= [eye(i-1) zeros(i-1,m-i+1); zeros(i-1, m-i+1)' (eye(m-i+1)-(2/norm([1; A(i+1:m, i)], 2)^2)*([1; A(i+1:m, i)]*[1 A(i+1:m, i)'])))]*Qt;
    end
    x = TriangolareSup(triu(A(1:n, :)), Qt(1:n, :)*b);
end
```

3.19 Esercizio 3.19

```
A = [3,2,1;1,2,3;1,2,1;2,1,2]

b = [6;6;4;4]

x = solveQR(A,b)

r = A*x-b

disp('Norma di r : '), norm(r,2)^2
```

3.20 Esercizio 3.20

Risolviamo il sistema di equazioni non lineari applicando il metodo di Newton. La funzione data è

$$F(x_1, x_2) = \begin{cases} x_2 - cos(x_1) \\ x_1 x_2 - 1/2 \end{cases}$$

Vogliamo trovare $F(x_1, x_2) = 0$ partendo da $x_1(0) = 1$, $x_2(0) = 1$

Troviamo quindi il Jacobiano della funzione: $J = \begin{pmatrix} sin(x_1) & 1 \\ x_1 & x_2 \end{pmatrix}$

Applicando il metodo di Newton si va a risolvere: $\begin{cases} J_F(\underline{x}^{(k)})\underline{d}^{(k)} = -F(\underline{x}^{(k)}) \\ \underline{x}^{(k+1)} = \underline{x}^{(k)} + \underline{d}^{(k)} \end{cases}$

Troviamo quindi: $x_1 = 0.6100$ e $x_2 = 0.8196$

Il codice matlab per il calcolo del minimo è:

```
format short

x(1)=1;
x(2)=1;
imax=1000;
tolx=0.0001;

F= @(x) [x(2) - cos(x(1)); x(1)*x(2)-1/2];
J = @(x) [sin(x(1)),1 ; x(2), x(1)];

[x] = NewtonNL(F, J, x, imax, tolx , 1);


disp ('Radice di F : '), disp (x)
disp ('F(x): '), disp ([x(2) - cos(x(1)), x(1)*x(2)-1/2]);
```

Il codice matlab per la risoluzione di sistemi di equazioni non lineari mediante Newton è:

```
function [x] = NewtonNL(F,J, x, imax, tolx, out)
    i=0;
    xold = x+1;
    while (i< imax )&&( norm (x-xold )> tolx )
        i=i+1;
        xold =x;
        [L,U,P] = LUP(feval(J,x));
        x=x+solveLinearLUP(L,U, P, -feval(F,x));
        if out
            disp(norm(x-xold));
            disp(x);
        end
    end
end
```

i	x_1, x_2	norma dell'incremento
1	0.7458, 0.7542	0.5001
2	0.5531, 0.8653	0.3145
3	0.6042, 0.8241	0.0929
4	0.6100, 0.8197	0.0102
5	0.6100, 0.8196	0.00013

3.21 Esercizio 3.21

Un punto stazionario $(\hat{x_1}, \hat{x_2})$ è tale per cui $J(\hat{x_1}, \hat{x_2}) = 0$. Si ottiene quindi il sistema non lineare:

$$F(\underline{x}) = \underline{0} \text{ con } F = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{pmatrix} = \begin{pmatrix} 4x_1^3 + 2x_1 + x_2 \\ x_1 + 2x_2 - 2 \end{pmatrix}.$$

Troviamo il Jacobiano della funzione: $J = \begin{pmatrix} 12x_1^2 + 2 & 1 \\ 1 & 2 \end{pmatrix}$

Troviamo quindi

$$\min f(x_1, x_2) \approx -0.2573 \text{ in } (0.4433, -1.2217).$$

Il codice matlab per il calcolo del minimo è:

```
format short

x(1)=1;
x(2)=1;
imax=1000;
tolx=0.1;

F= @(x) [4*x(1)^3+2* x(1)+ x(2); x(1)+2*x(2)-2];
J = @(x) [12*x(1)^2+2, 1; 1, 2];

[x] = NewtonNL(F, J, x, imax, tolx, 0);

disp ('Minimo : '), disp (x)
disp ('F(x): '), disp (x(1)^4+ x(1)*( x(1)+ x(2))+(1-x(2))^2)
```

4 Capitolo 4

4.1 Esercizio 4.1

```
function [pval] = poliNewton(xi, fi, xval)
  dd = diffDiv(xi, fi);
  pval = HornerGeneralizzato(xi,dd,xval);
end
```

4.2 Esercizio 4.2

Il seguente listato genera due grafici per le funzioni date:

```
Rungef = @(x) 1./(1.+x.^2);
a=-5; b=5;
res_err = zeros(4,10);
[errors, plots, l] = evaluate_poli(Rungef, a,b, 20, 10, 0, 100);
plots = cat(2,plots', Rungef(l)');
res_err(1,:) = max(abs(errors'))';
plot(l,plots');
lgd=legend('2','4','6','8','10','\infty');
title(lgd, 'Ascisse');
plot(l,errors')
lgd=legend('2','4','6','8','10');
title(lgd, 'Ascisse');
[errors, plots, l] = evaluate_poli(Rungef, a,b, 20, 10, 1, 100);
res_err(2,:) = max(abs(errors'))';
plots = cat(2,plots', Rungef(l)');
plot(l,plots');
lgd=legend('2','4','6','8','10','12','14','16','18','20','\infty');
title(lgd, 'Ascisse');
plot(l,errors')
lgd=legend('2','4','6','8','10','12','14','16','18','20');
title(lgd, 'Ascisse');

Sinf = inline('x.*sin(x)');
a=0; b=pi;
max_n = 20;
[errors, plots, l] = evaluate_poli(Sinf, a,b, max_n, 10, 0, 100);
res_err(3,:) = max(abs(errors'))';

plots = cat(2,plots', Sinf(l)');
lgd=legend('2','4','6','8','10','12','14','16','18','20','\infty');
title(lgd, 'Ascisse');
plot(l,plots');
plot(l,errors')
lgd=legend('2','4','6','8','10','12','14','16','18','20');
title(lgd, 'Ascisse');
[errors, plots, l] = evaluate_poli(Sinf, a,b, max_n, 10, 1, 100);
res_err(4,:) = max(abs(errors'))';

plots = cat(2,plots', Sinf(l)');
plot(l,plots');
lgd=legend('2','4','6','8','10','12','14','16','18','20','\infty');
title(lgd, 'Ascisse');
plot(l,errors')
lgd=legend('2','4','6','8','10','12','14','16','18','20');
title(lgd, 'Ascisse');
```

```
function [errors, plots, l] = evaluate_poli(funct, a, b, maxn, n_steps, cheb_asc, plot_steps)
    errors = zeros(n_steps,plot_steps);
    plots = zeros(n_steps,plot_steps);
    l = linspace(a,b,plot_steps);
    steps = linspace(2, maxn, n_steps);
    for i=1:n_steps
        if cheb_asc == 0
            ascisse = ascisseEquispaziate(a, b, steps(i));
        elseif cheb_asc == 1
            ascisse = chebyshev(a, b, steps(i));
        end

        fInt = poliNewton(ascisse, funct(ascisse), l);
        errors(i,:) = funct(l)-fInt;
        plots(i,:) = fInt;
    end
end
```

```
function [ptx] = ascisseEquispaziate(a, b, n)
  h = (b-a)/n;
  ptx = zeros(n+1, 1);
  for i=1:n+1
    ptx(i) = a +(i-1)*h;
  end
end
```

```
function [xi] = chebyshev(a,b,n)
  xi = zeros(n+1, 1);
  for i=0:n
    xi(n+1-i) = (a+b)/2 + cos(pi*(2*i+1)/(2*(n+1)))*(b-a)/2;
  end
end
```

$$\left(\begin{array}{cccc}
RungeEq & RungeCheb & SinEq & SinCheb \\
\hline
0.646 & 0.4371 & 0.6381 & 0.4371 \\
0.4383 & 0.02286 & 0.04127 & 0.02286 \\
0.6164 & 0.0004779 & 0.001343 & 0.0004779 \\
1.045 & 5.332 \cdot 10^{-6} & 2.575 \cdot 10^{-5} & 5.332 \cdot 10^{-6} \\
1.915 & 3.688 \cdot 10^{-8} & 3.238 \cdot 10^{-7} & 3.688 \cdot 10^{-8} \\
3.612 & 1.734 \cdot 10^{-10} & 2.843 \cdot 10^{-9} & 1.734 \cdot 10^{-10} \\
7.189 & 5.892 \cdot 10^{-13} & 1.873 \cdot 10^{-11} & 5.892 \cdot 10^{-13} \\
14.01 & 3.417 \cdot 10^{-15} & 1.261 \cdot 10^{-13} & 3.417 \cdot 10^{-15} \\
27.51 & 1.776 \cdot 10^{-15} & 8.933 \cdot 10^{-14} & 1.776 \cdot 10^{-15} \\
58.41 & 2.327 \cdot 10^{-15} & 1.768 \cdot 10^{-13} & 2.327 \cdot 10^{-15}
\end{array}\right)$$

Possiamo vedere la differenza tra 5 e 6, nella prima all'aumentare delle ascisse la funzione interpolata degenera, mentre nella seconda già con n=5 abbiamo una buona interpolazione. Nel caso della seconda funzione possiamo vedere che la differenza tra 7 e 8 non è molto rilevante. Infatti già con 4 ascisse abbiamo un interpolazione quasi perfetta. Nelle figure 9 10 11 12 e' possibile vedere l'andamento dell'errore per i vari metodi di interpolazione.

4.3 Esercizio 4.3

```
function [ m ] = momenti(phi, xi, dd)
    n = length(xi) + 1;
    u = zeros(1, n - 1);
    l = zeros(1, n - 2);
```

```
    dd = 6 * dd;
    u(1) = 2;
    for i = 2 : n - 1
        l(i) = phi(i) / u(i - 1);
        u(i) = 2 - l(i) * xi(i - 1);
    end
    y = zeros(1, n - 1);
    y(1) = dd(1);
    for i = 2 : n - 1
        y(i) = dd(i) - l(i) * y(i - 1);
    end
    m = zeros(1, n - 1);
    m(n - 1) = y(n - 1) / u(n - 1);
    for i = n - 2 : - 1 : 1
        m(i) = (y(i) - xi(i) * m(i + 1)) / u(i);
    end
    m = [0, m, 0];
end
```

4.4 Esercizio 4.4

```
function [ xx ] = valuta(p, s, xx)
    n=length(p) - 1;
    k=1;
    j=1;
    for i = 1 : n
        inInt = 1;
        while j <= length(xx) && inInt
            if xx(j) >= p(i) && xx(j) <= p(i + 1)
                j = j + 1;
            else
                inInt = 0;
            end
        end
        xx(k : j - 1) = subs(s(i), xx(k : j - 1));
        k = j;
    end
end
```

4.5 Esercizio 4.5

Il seguente listato valuta la spline naturale e quella not-a-knot per le funzioni date:

```
Rungef = @(x) 1./(1.+x.^2);
a=-5; b=5;
max_n = 20;
n_steps = 5;
[plots, l] = evaluate_spline(Rungef,a,b, max_n, n_steps, 0, 1000);
hold on;
grid on;
plot(l, plots);
hold off
[plots2, l] = evaluate_spline(Rungef,a,b, max_n, n_steps, 1, 1000);
hold on;
grid on;
plot(l, plots2);
hold off
```

```

error = plots' - plots2';
boxplot(error(:,1:5), 4:4:max_n);

Sinf = @(x) x.*sin(x);
a=0; b=pi;
[plots, l] = evaluate_spline(Sinf,a,b, max_n, n_steps, 0, 1000);
hold on;
grid on;
plot(l, plots);
hold off

[plots2, l] = evaluate_spline(Sinf,a,b, max_n, n_steps, 1, 1000);
hold on;
grid on;
plot(l, plots2);
hold off
error = plots' - plots2';

boxplot(error(:,1:5), 4:4:max_n);
```

Nelle figure 13 e 14 possiamo vedere come l'errore nel tra una spline naturale e una spline not a knot sia visibilmente inferiore nel caso della funzione $xsin(x)$.

Nei grafici 15 16 invece vediamo l'approssimazione delle spline naturali e not a knot per le medesime funzioni, comparate con il grafico della funzione originale (in nero).

4.6 Esercizio 4.6

4.7 Esercizio 4.7

4.8 Esercizio 4.8

```
function [y] = sovradet(x,y, m)
  x=x';
  if length(unique (x)) < m+1
    error('non ci sono m ascisse distinte');
  end
  V(:,m+1) = ones(length(x),1);
  for j = m:-1:1
    V(:,j) = x.*V(:,j+1);
  end
  y = V\y';
  y=y';
end
```

4.9 Esercizio 4.9

```
f1 = @(x,e,l) 5*x+ 2 +e*l;
f2 = @(x,e,l) 3*x^2 + 2*x +1 + e*l;
l=rand(1);
e= 0.1;
s= linspace(-1,1,10);
y1 = zeros(10,1);
y2 = zeros(10,1);

for i=1:10
    l=rand(1);
    y1(i) = f1(s(i),e,l);
```

```
    y2(i) = f2(s(i),e,l);

end

res1 = sovradet(s,y1',1);
res2 = sovradet(s,y2',2);
```

4.10 Esercizio 4.10

```
f1 = @(x,e,l) 5*x+ 2 +e*l;
l=rand(1);
e= 0.1;
s= linspace(-1,1,10);
y1 = zeros(10,1);
y = zeros(2,10);
for i=1:10
    l=rand(1);
    y1(i) = f1(s(i),e,l);
end

fit = sovradet(s, y1',1);
y(1,:)=polyval(fit, s);

invfit = sovradet(y1',s,1);
y(2,:)=polyval(invfit',y1);
plot(y1',y);
```

vedi 19 per il grafico

5 Capitoli 5/6

5.1 Esercizio 5.1

```
function [In] = trapeziComp(f,a,b,n)
    h = (b-a)/n;
    In = 0;
    for i=1:n-1
        In = In + f(a+i*h);
    end
    In = (h/2)*(2*In + f(a) + f(b));
end
```

```
function [In] = simpsonComp(f,a,b,n)
    h = (b-a)/n;
    In = f(a)-f(b);
    for i=1:n/2
        In = In + 4*f(a+(2*i-1)*h)+2*f((a+2*i*h));
    end
    In = In*(h/3);
end
```

5.2 Esercizio 5.2

```
format long
F = @(x) x*exp(1)^-x*cos(2*x);
y = (3*(exp(1)^(-2*pi) -1) -10*pi*exp(1)^(-2*pi))/25;
nmax = 8;
err = zeros(nmax,2);
rap = zeros(nmax-1,2);
for i=1:8
    err(i,1) = abs(y - trapeziComp(F,0,2*pi,2^i));
    err(i,2) = abs(y - simpsonComp(F,0,2*pi,2^i));
    if i>1
        rap(i-1,:) = err(i,:)./err(i-1,:);
    end
end

semilogy([1:8],err);
plot([2:nmax],rap);
```

Al posto di riportare i dati in una tabella abbiamo ritenuto piú opportuno mostrare l'andamento dell'errore mediante l'uso di grafici 20 21. L'andamento del rapporto tra gli errori è scorrelato per i primi 2^5 sottointervalli, ma dopo si stabilizza con un rapporto costante.

5.3 Esercizio 5.3

```
f = @(x) x*exp(-x)*cos(2*x);

[In,px] = simpsonAda(f,0,2*pi,10^-5, 5);

disp(In);
disp(px);

[In,px] = trapeziAda(f,0,2*pi,10^-5,3);

disp(In);
```

```
disp(px);
```

```
function [In] = simpsonComp(f,a,b,n)
    h = (b-a)/n;
    In = f(a)-f(b);
    for i=1:n/2
        In = In + 4*f(a+(2*i-1)*h)+2*f((a+2*i*h));
    end
    In = In*(h/3);
end
```

```
function [In,pt] = simpsonAda(f, a, b, tol)
  pt=5
  h = (b-a)/6;
  m = (a+b)/2;
  m1 = (a+m)/2;
  m2 = (m+b)/2;
  In1 = h*(feval(f, a) + 4*feval(f, m) + feval(f, b));
  In = In1/2 + h*(2*feval(f, m1) + 2*feval(f, m2) - feval(f, m));
  err = abs(In-In1)/15;
  if err>tol
    [intSx, ptSx] = simpsonAdattativaRicorsiva(f, a, m, tol/2, 1);
    [intDx, ptDx] = simpsonAdattativaRicorsiva(f, m, b, tol/2, 1);
    In = intSx+intDx;
    pt = pt+ptSx+ptDx;
  end
end
```

5.4 Esercizio 5.4

```
function [xn, i, err] = jacobi(A, b, x0, tol, nmax)
  D = diag(diag(A));
  J = -inv(D)*(A-D);
  q = D\b;
  xn = J*x0 + q;
  i = 1;
  err(i) = norm(xn-x0)/norm(xn);
  while (i<=nmax && err(i)>tol)
    x0 = xn;
    xn = J*x0+q;
    i = i+1;
    err(i) = norm(xn-x0)/norm(xn);
  end
  if i>nmax
    disp('Jacobi non converge nel numero di iter fissato');
  end
end
```

```
function [xn, i, err] = gaussSeidel(A, b, x0, tol, nmax)
  D=diag(diag(A));
  L=tril(A)-D;
  U=triu(A)-D;
  DI=inv(D+L);
  GS=-DI*U;
  b1=(D+L)\b;
  xn=GS*x0+b1;
```

```
  i=1;
  err(i)=norm(xn-x0,inf)/norm(xn);

  while(err(i)>tol && i<=nmax)
    x0=xn;
    xn=GS*x0+b1;
    i=i+1;
    err(i)=norm(xn-x0,inf)/norm(xn);
  end
  if i>nmax
    error('Gauss-Seidel non converge nel numero di iterazioni fissato');
  end
  i=i-1;
end
```

5.5 Esercizio 5.5

```
A = [-4,2,1;1,6,2;1,-2,5];
b = [1,2,3]';
x0 = [0,0,0]';

[z,j,jerr] = jacobi(A,b,x0,1.e-3,25)
[y,i,gerr] = gaussSeidel(A,b,x0,25,1.e-3)
```

Il sistema $A = \begin{pmatrix} -4 & 2 & 1 \\ 1 & 6 & 2 \\ 1 & -2 & 5 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$, partendo dal vettore iniziale $x_0 = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$ restituisce il vettore $\begin{pmatrix} -0.02682 \\ 0.1201 \\ 0.6534 \end{pmatrix}$ con il metodo di `Jacobi` ed il vettore $\begin{pmatrix} -0.02688 \\ 0.12 \\ 0.6532 \end{pmatrix}$ con quello di `Gauss-Seidel`.

Metodo	Iterazioni
Jacobi	12
Gauss-Seidel	8

5.6 Esercizio 5.6

```
H = [0,0,0,0,0;1,0,1,0,0;1,1,0,0,0;0,1,0,0,0;0,1,0,0,0];
p=0.85;


[n,m] = size(H);
if(n~=m), error('Matrice non quadrata'); end
s = sum(H);
S=zeros(n,n);
for i=1:n
    if s(i)~=0
        S(:,i)=H(:,i)/s(i);
    else
        S(:,i)=(1/n);
    end
end
A= eye(n) - p*S;
b = ((1-p)/n).*ones(n,1);
tols= logspace(-1,-10,10);
iters = zeros(10,3);
```

```

for i=1:10
    v=zeros(n,4);
    [v(:,1),iters(i,1)]=PotenzePR(S,p,tols(i));
    [v(:,2),iters(i,2)]=jacobi(A,b,ones(n,1), tols(i), 10000);
    [v(:,3),iters(i,3)]=gaussSeidel(A,b,ones(n,1), tols(i), 10000);
end
plot(iters)
```

Nel grafico 22 é mostrato l'andamento dei vari metodi numerici per il calcolo dell'autovettore. Si nota come al crescere della tolleranza i metodi di Jacobi e delle Potenze non divergano sostanzialmente, mentre il metodo di Gauss-Seidel mostra una maggior efficienza anche per valori di tolleranza ridotti.

6 Figure

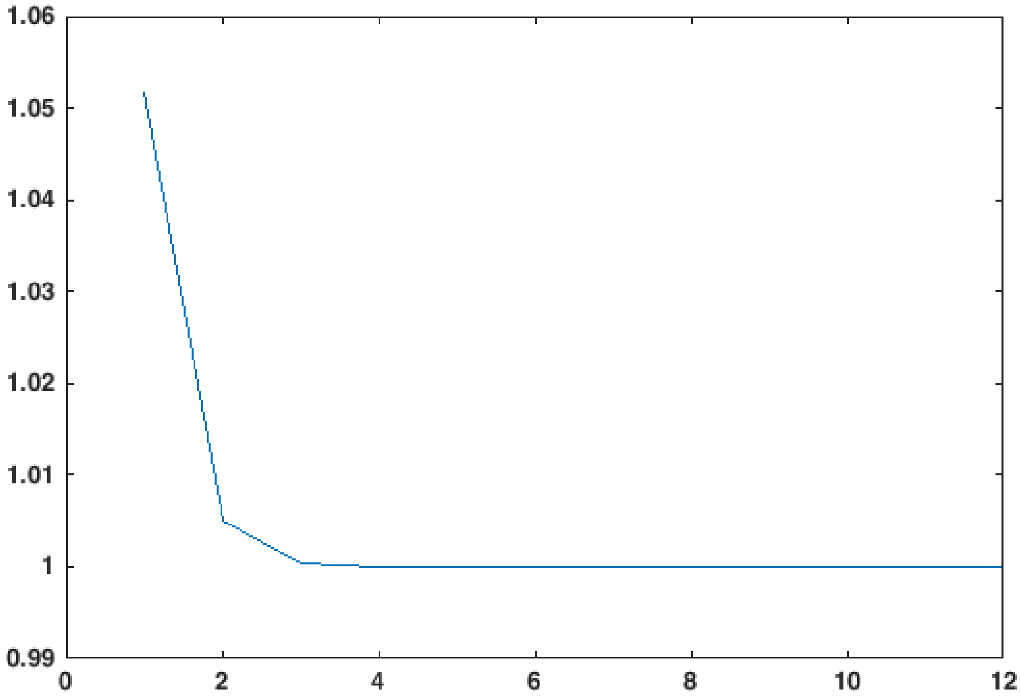

Figure 1: Esercizio 1.4

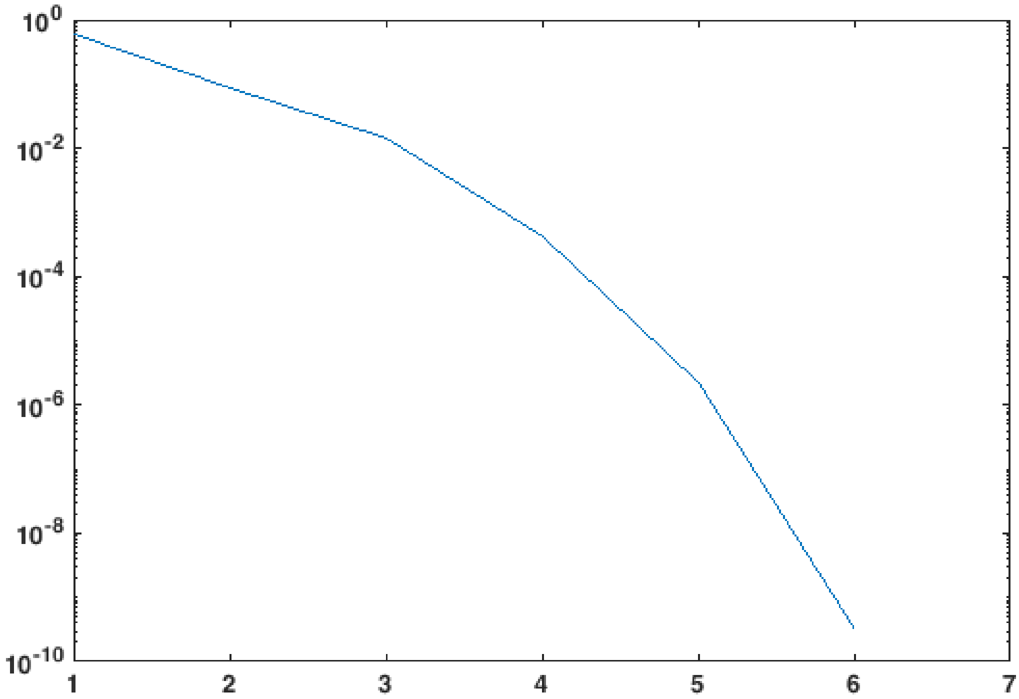

Figure 2: Esercizio 1.6

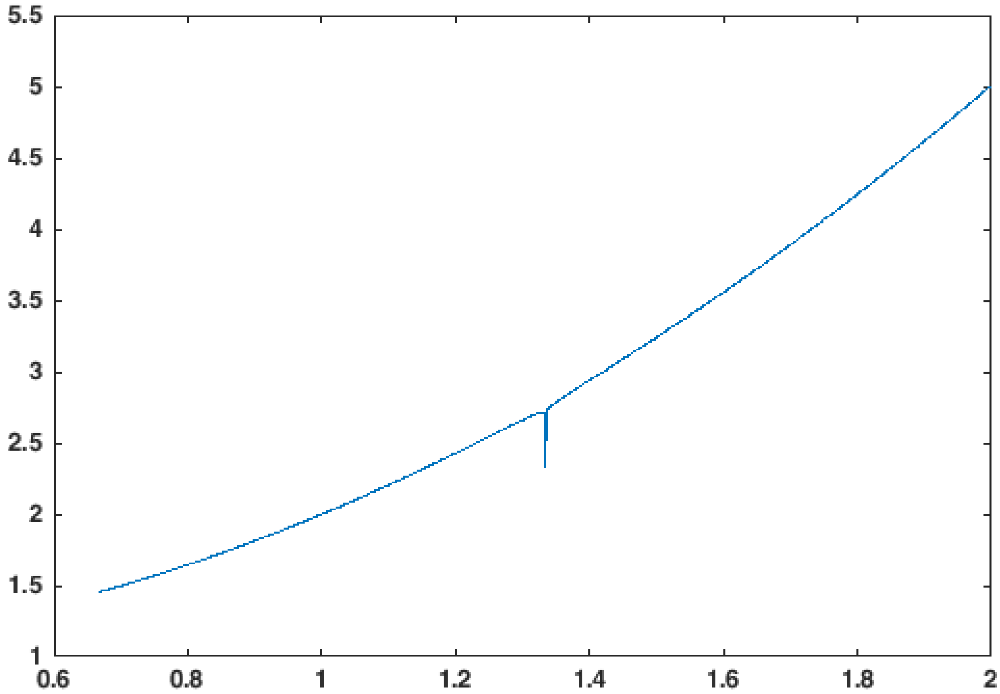

Figure 3: Esercizio 1.13

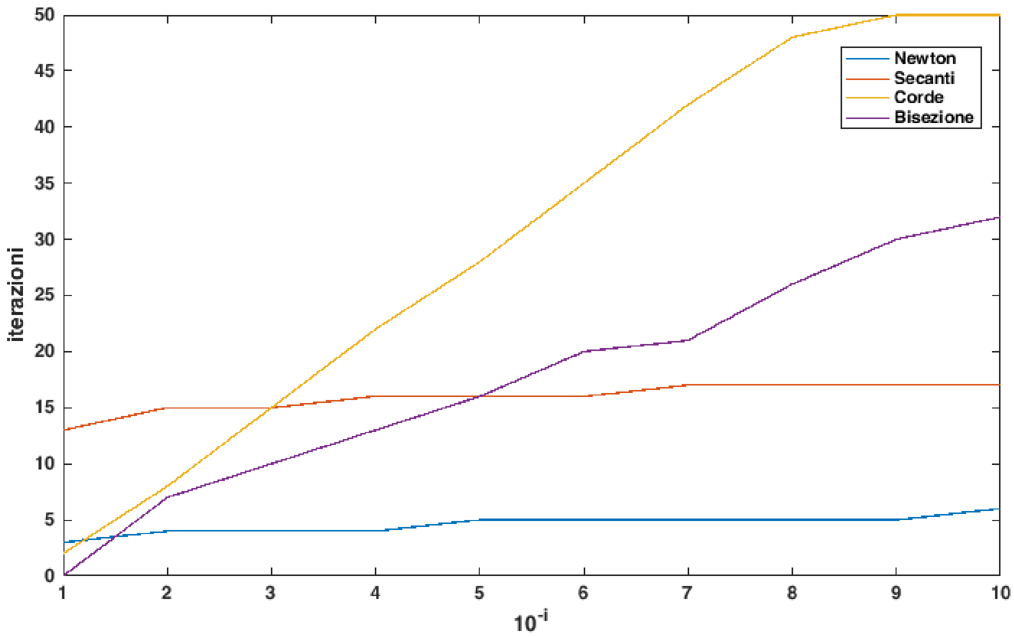

Figure 4: Comparazione del numero di iterazioni necessarie per i vari metodi

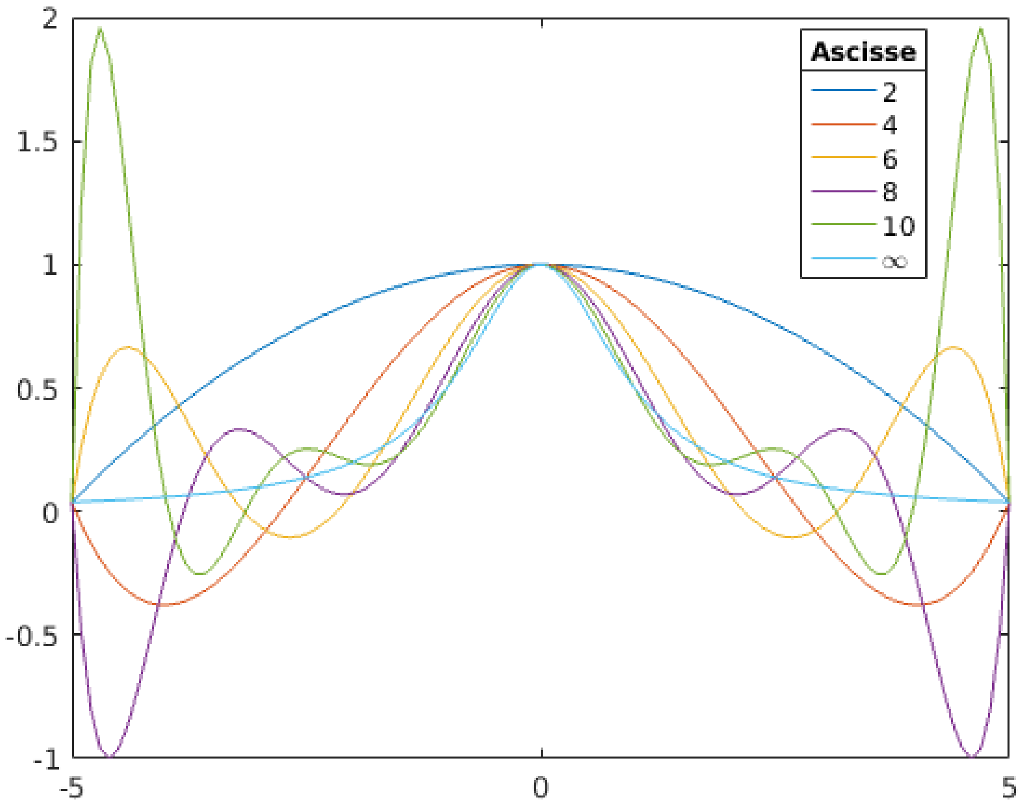

Figure 5: Ascisse Equidistanti per $f(x) = \frac{1}{1+x^2}$

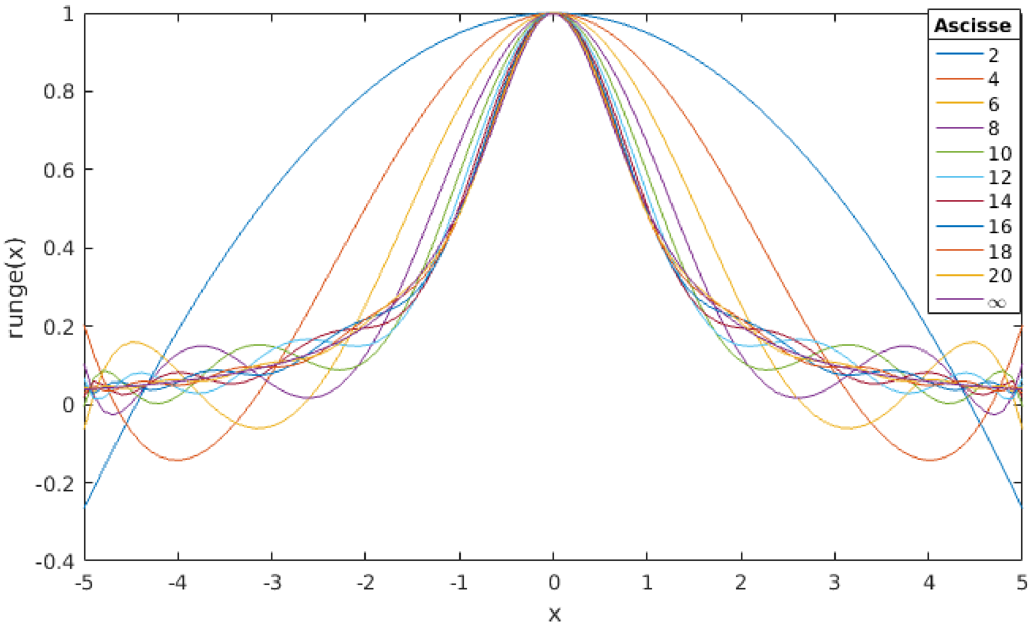

Figure 6: Ascisse di Chebyshev per $f(x) = \frac{1}{1+x^2}$

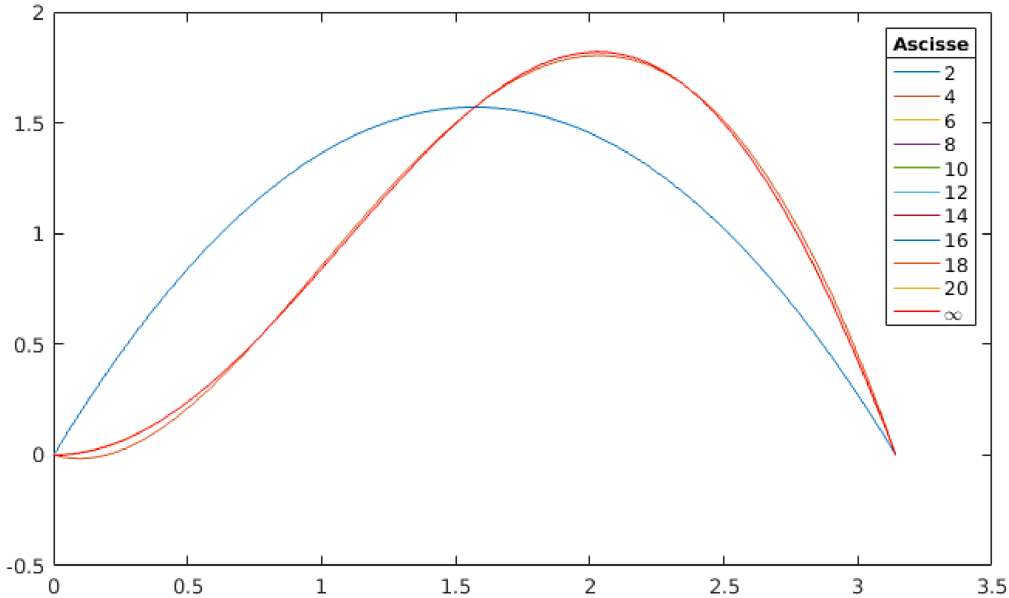

Figure 7: Ascisse Equidistanti per $g(x) = sin(x) * x$

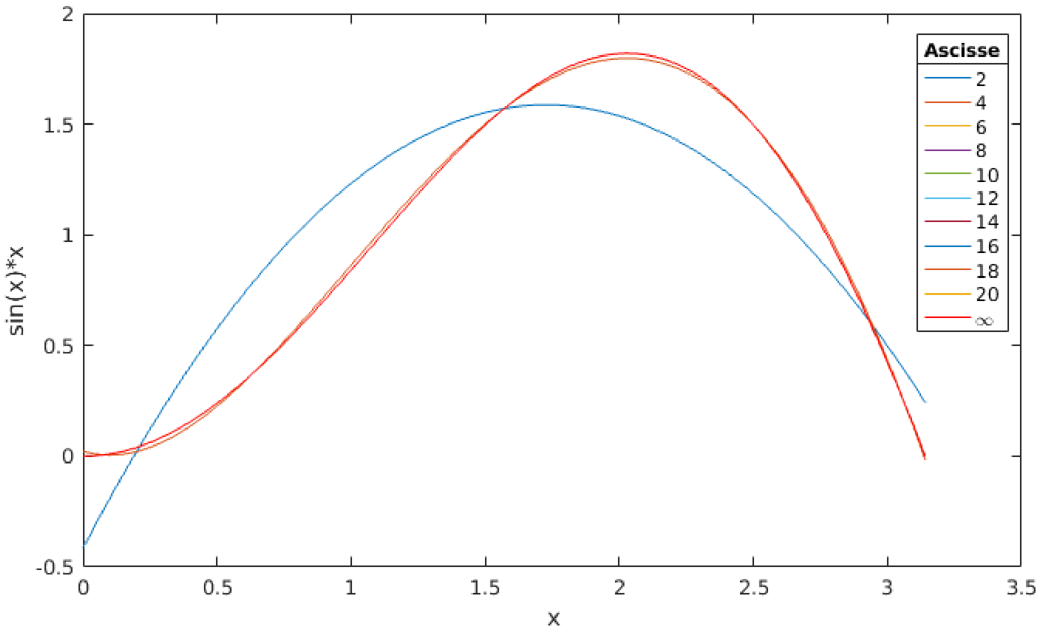

Figure 8: Ascisse di Chebyshev per $g(x) = sin(x) * x$

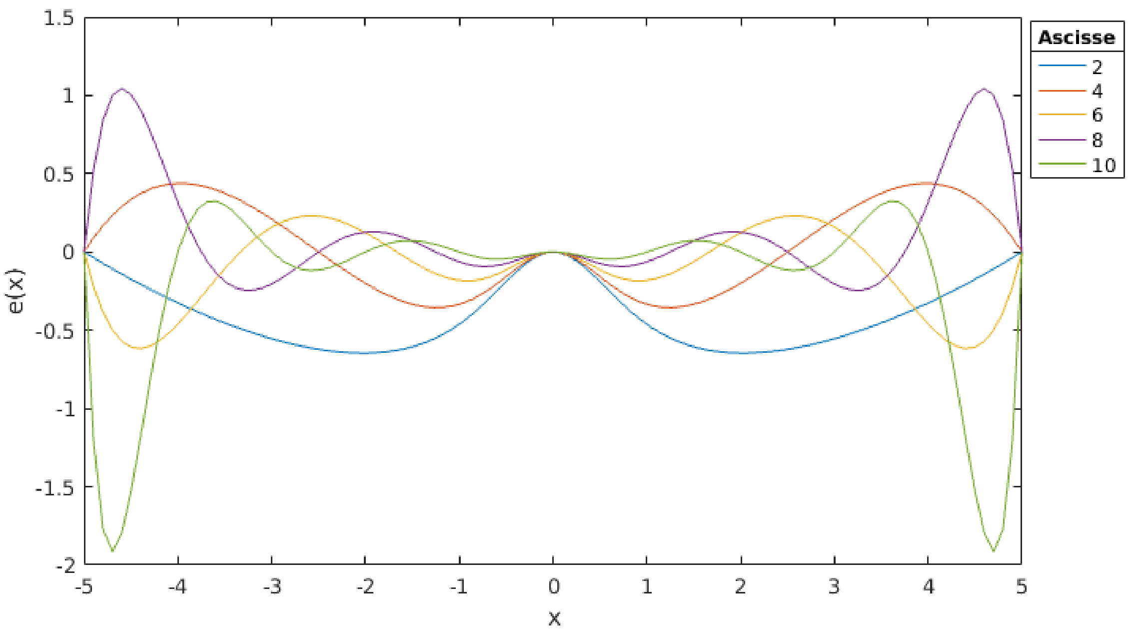

Figure 9: Errore per Ascisse Equidistanti per $f(x) = \frac{1}{1+x^2}$

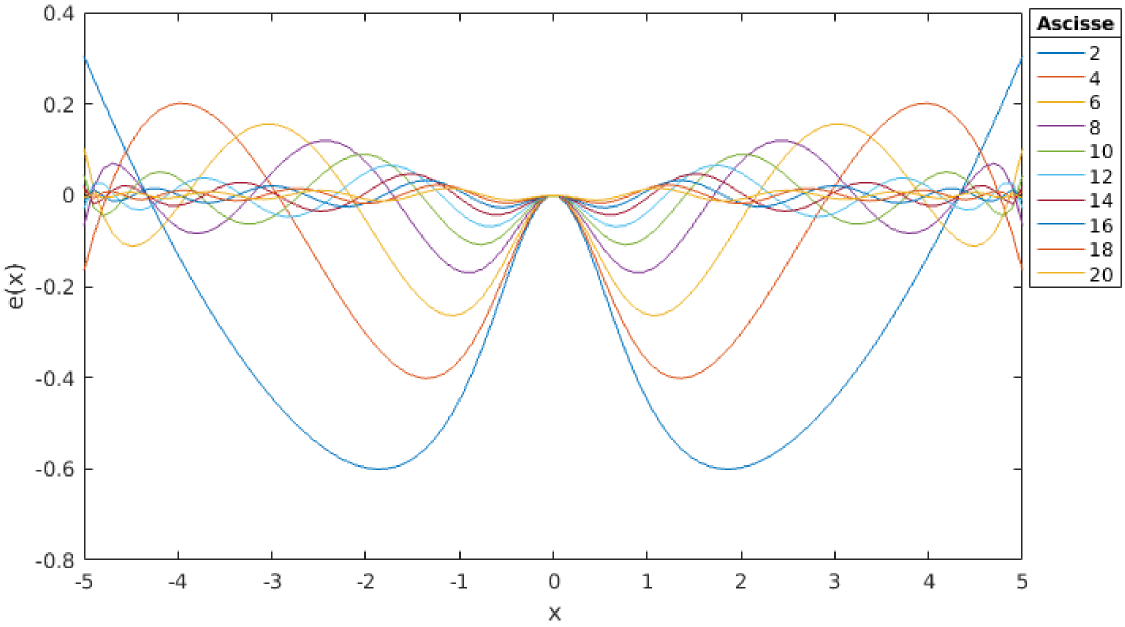

Figure 10: Errore per Ascisse di Chebyshev per $f(x) = \frac{1}{1+x^2}$

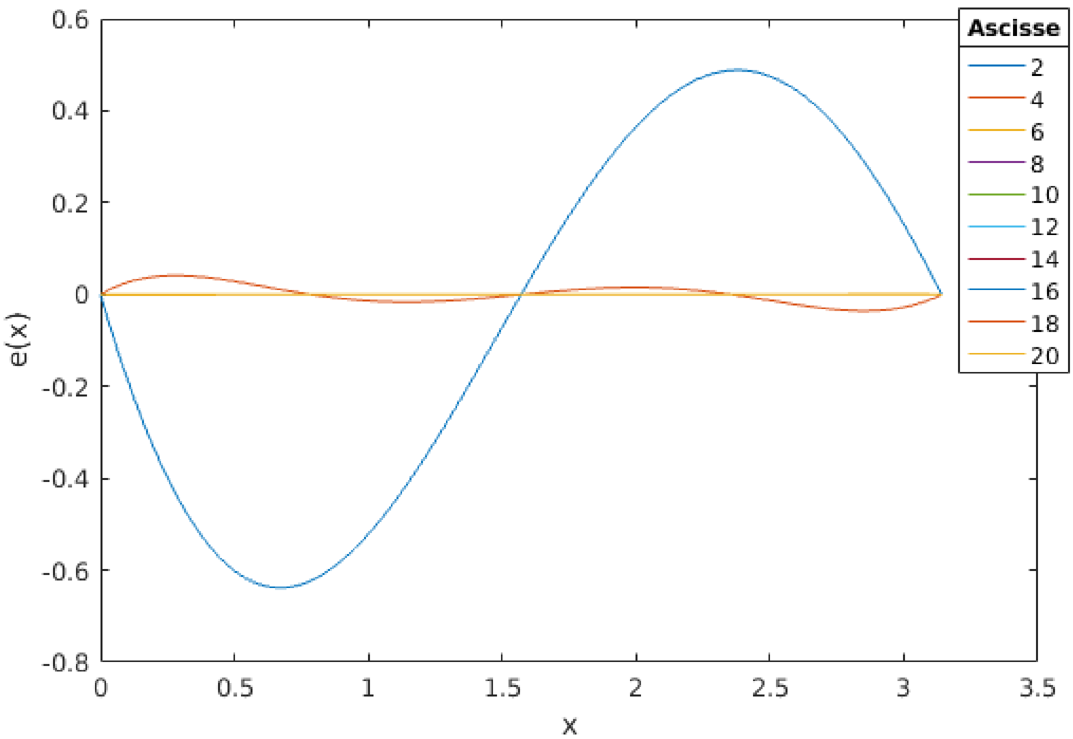

Figure 11: Errore per Ascisse Equidistanti per $g(x) = sin(x) * x$

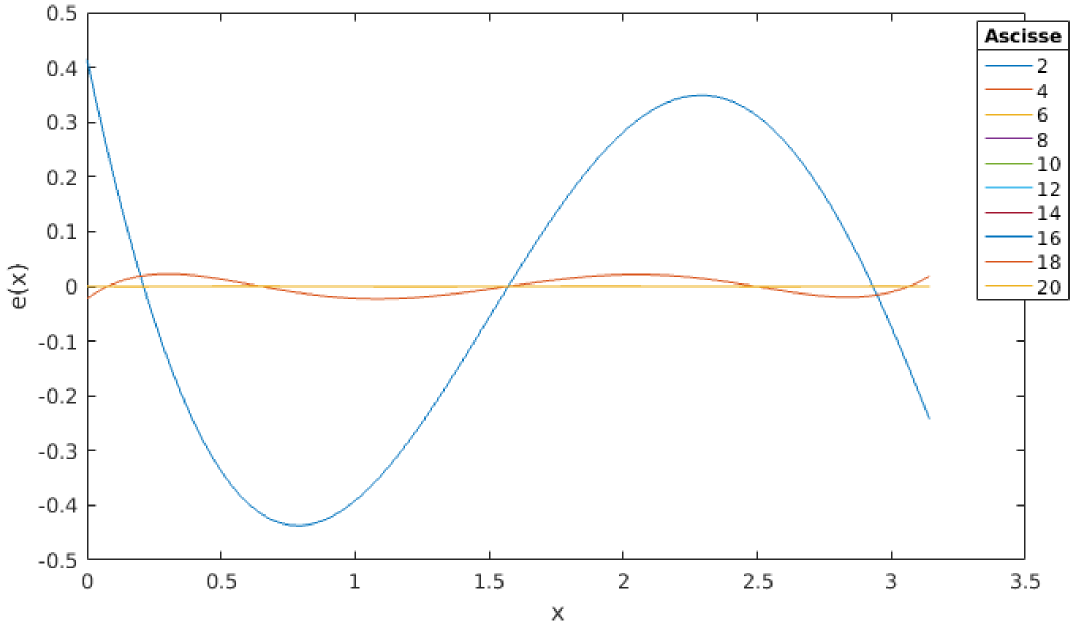

Figure 12: Errore per Ascisse di Chebyshev per $g(x) = sin(x) * x$

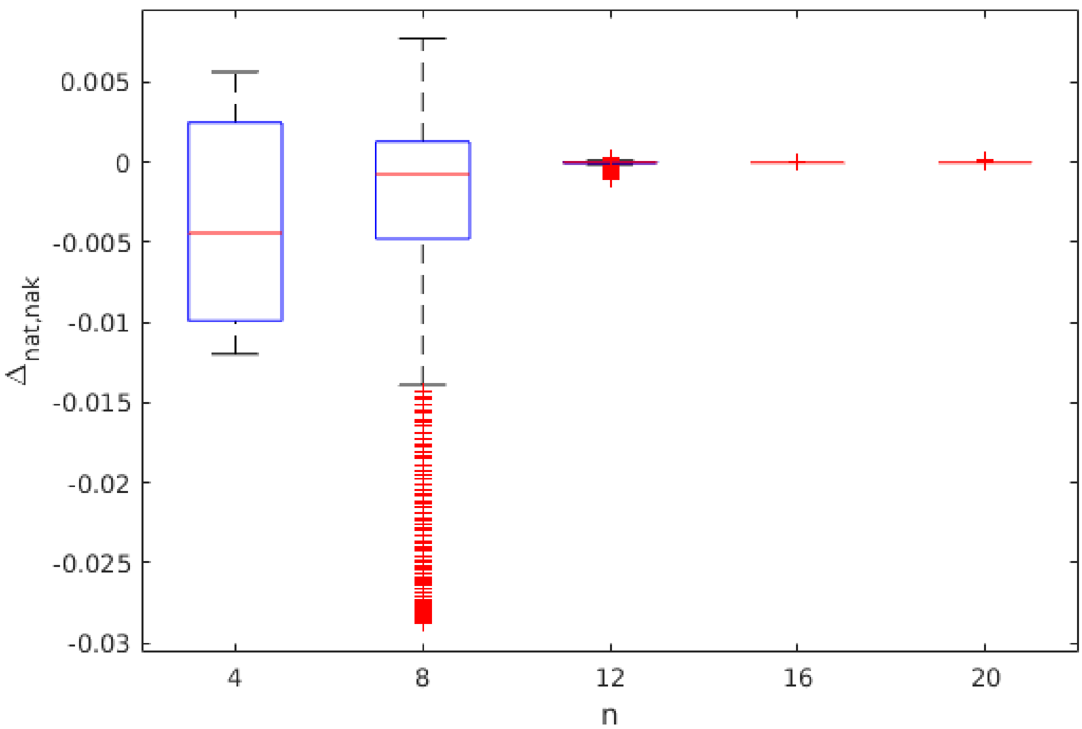

Figure 13: $\Delta_{nat,nak}$ per la funzione di Runge

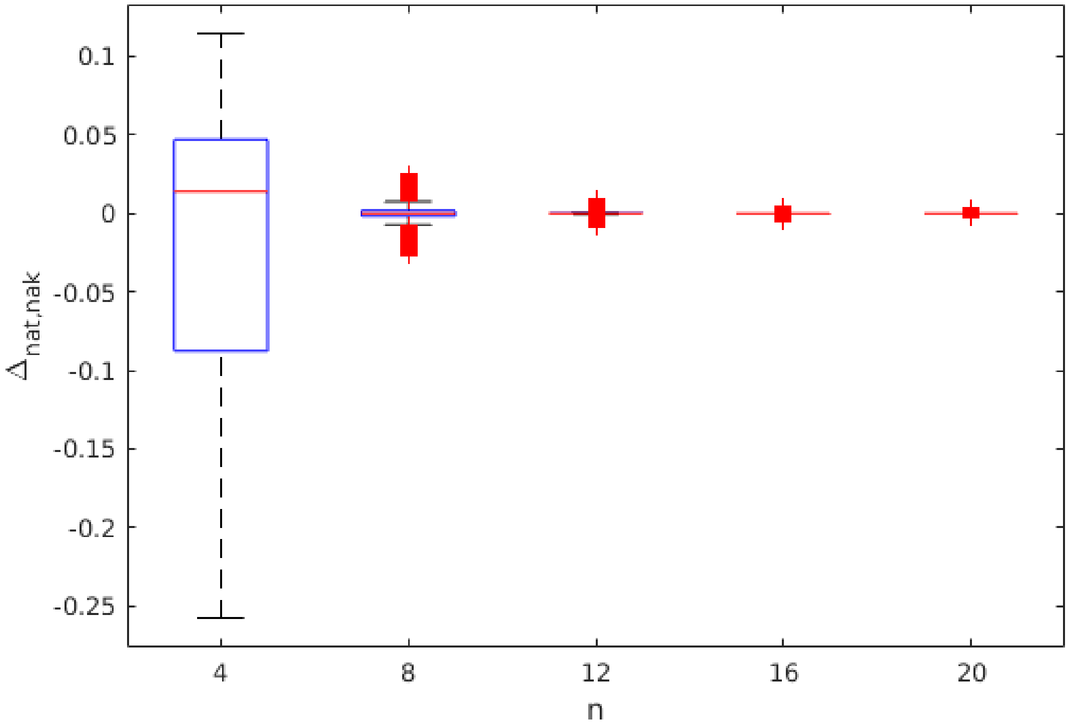

Figure 14: $\Delta_{nat,nak}$ per la funzione $xsin(x)$

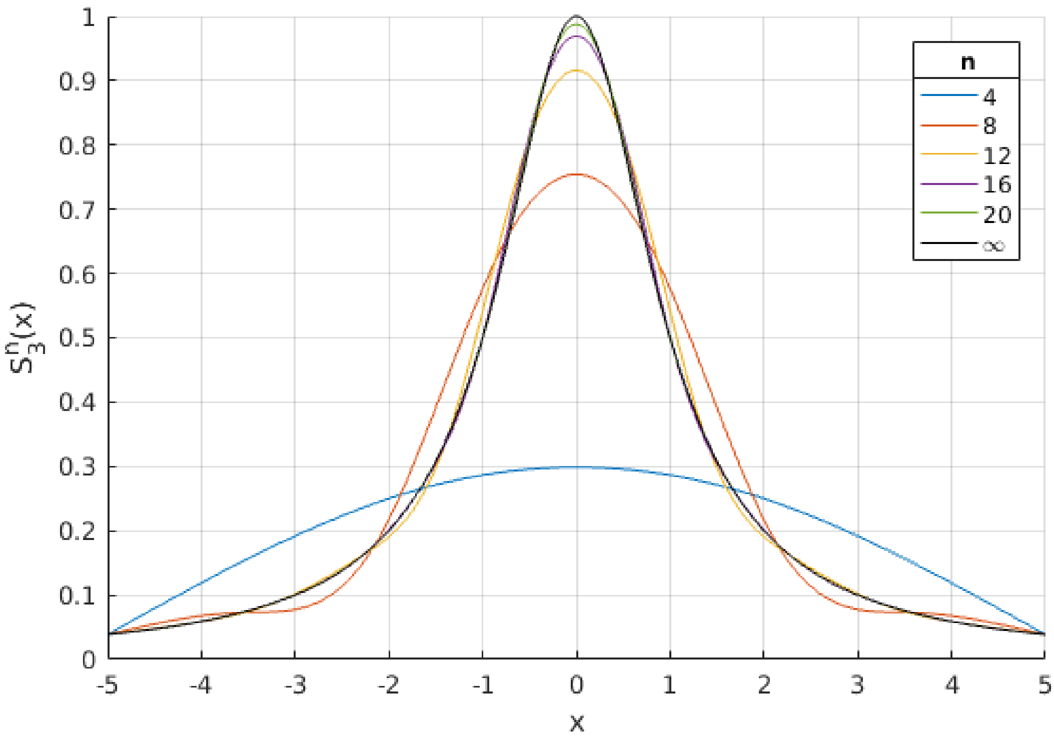

Figure 15: $\frac{1}{1+x^2}$ interpolata con spline naturale

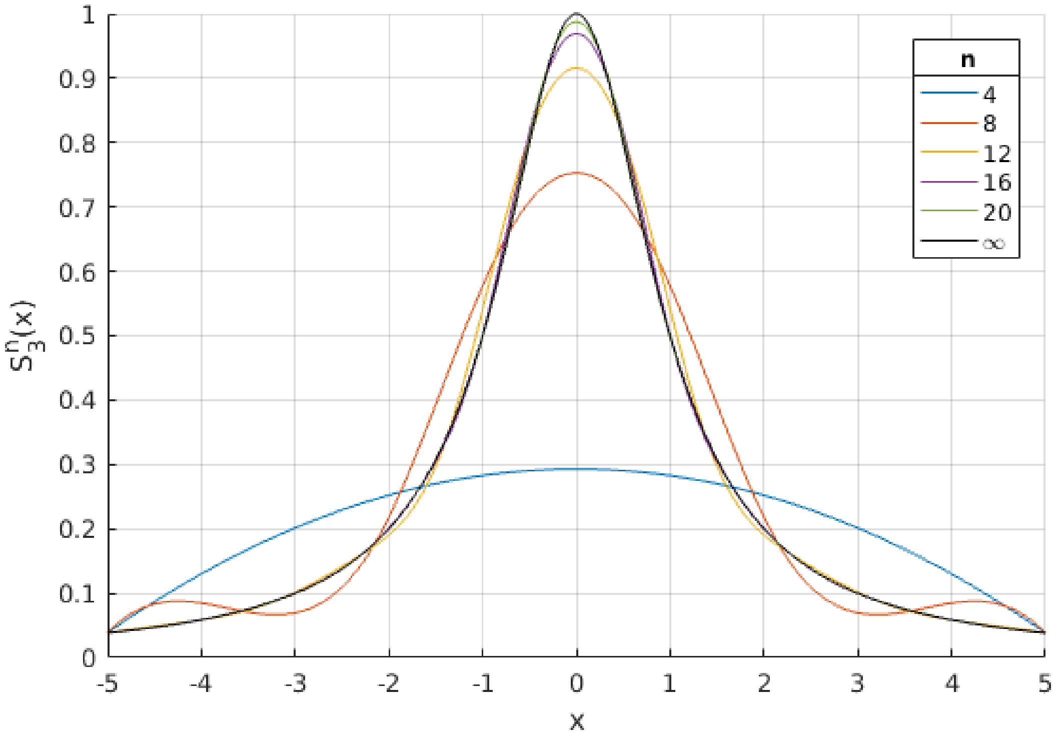

Figure 16: $\frac{1}{1+x^2}$ interpolata con spline not-a-knot

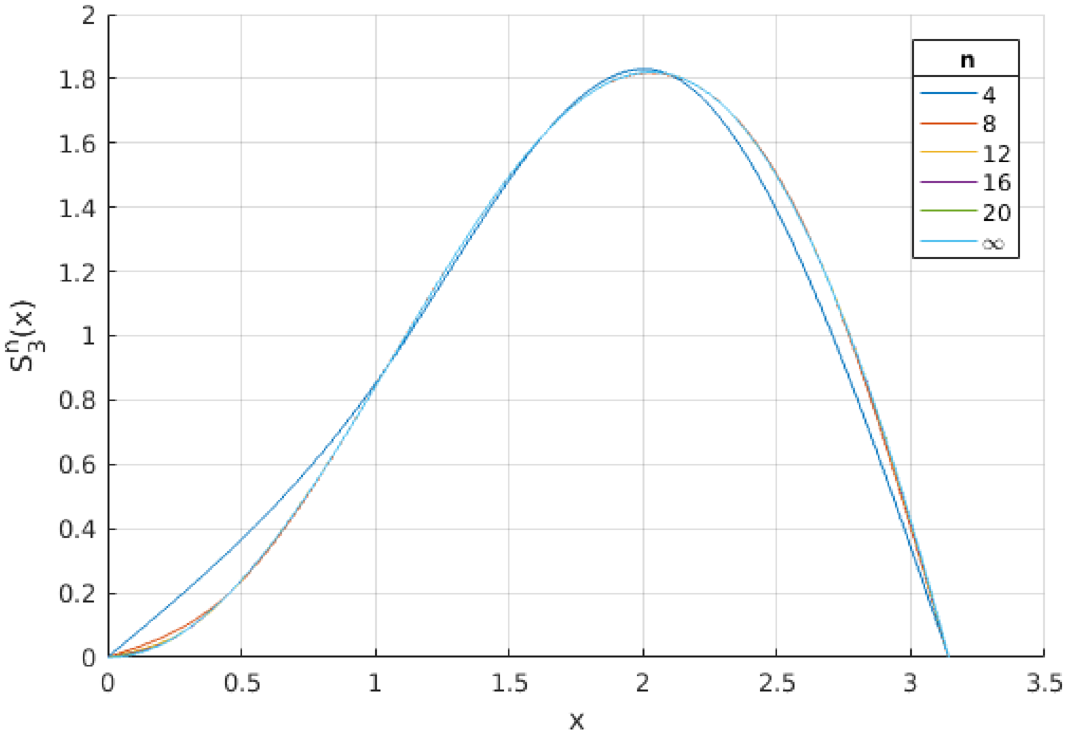

Figure 17: $xsin(x)$ interpolata con spline naturale

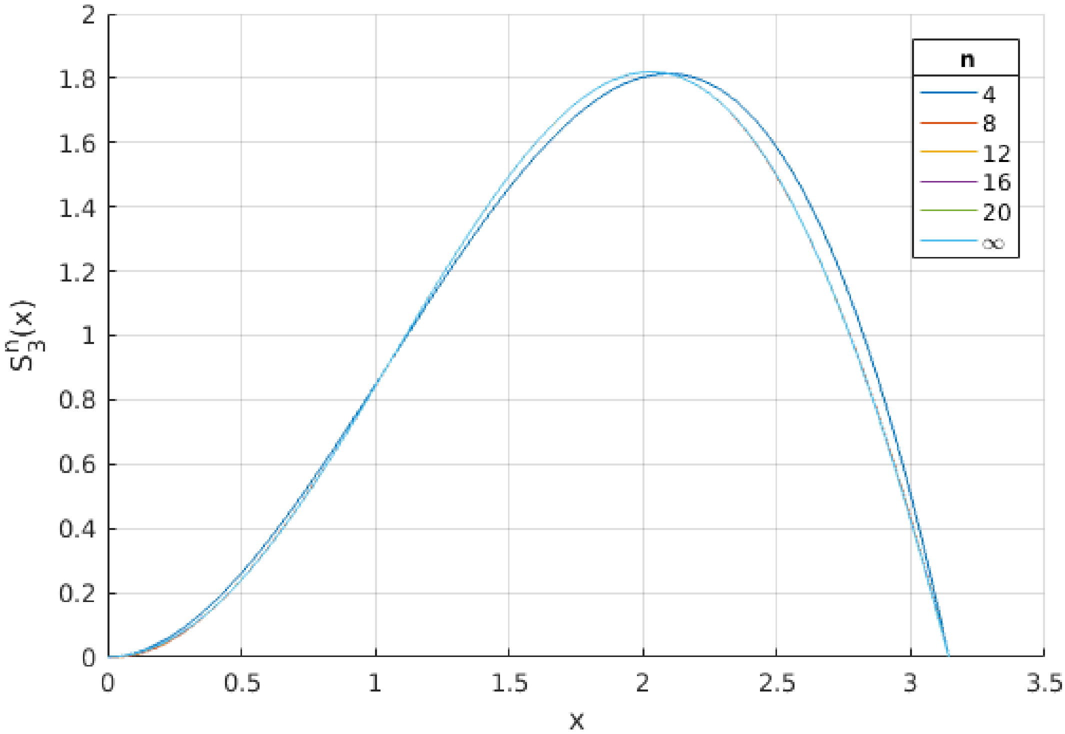

Figure 18: $xsin(x)$ interpolata con spline not-a-knot

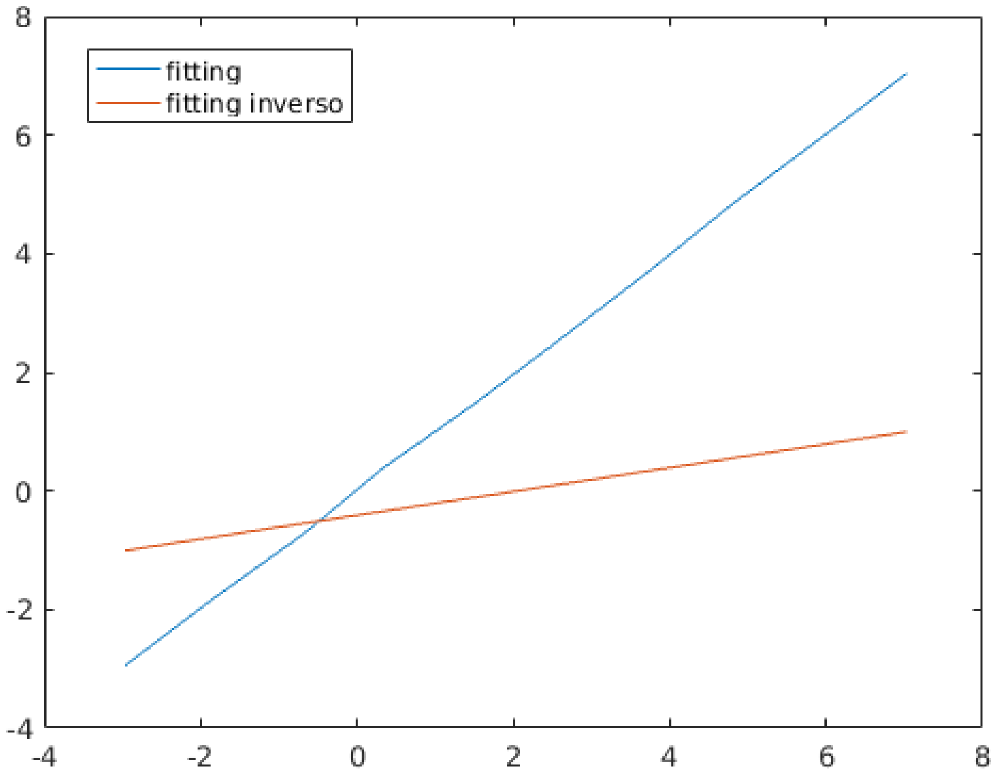

Figure 19: Confronto dei due grafici dell'esercizio 4.8

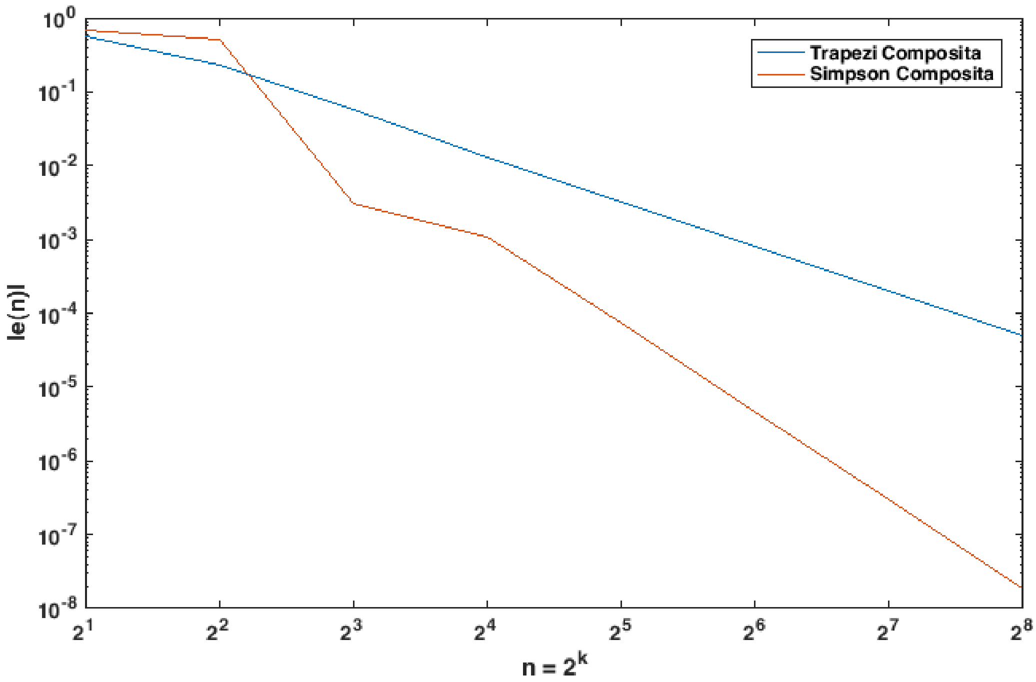

Figure 20: Andamento dell'errore per i metodi di Quadratura

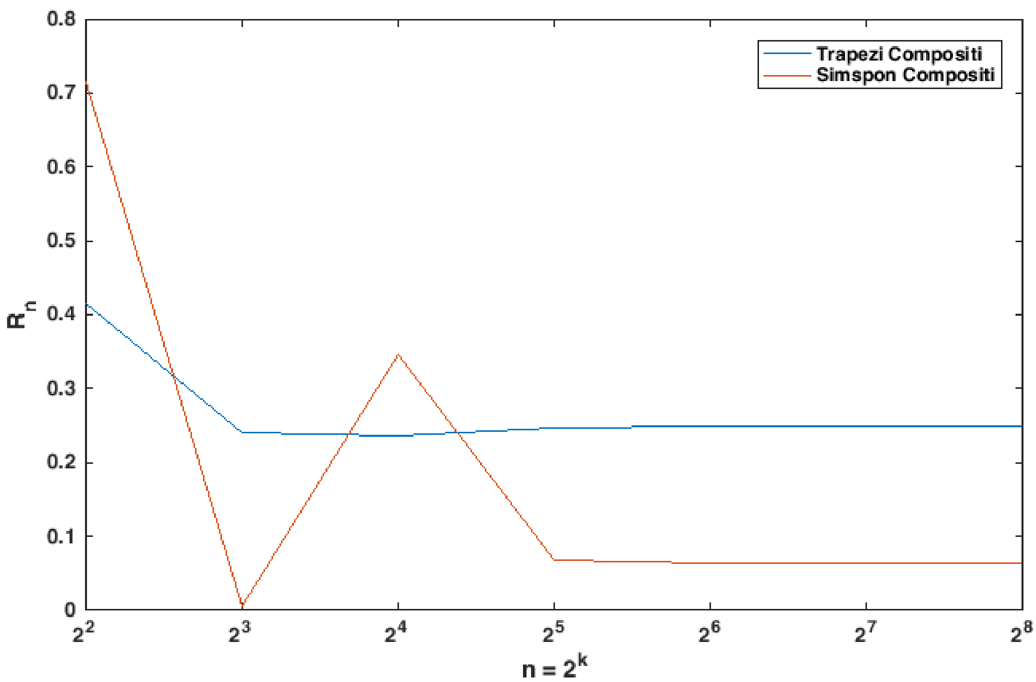

Figure 21: Andamento del Rapporto tra gli errori per i metodi di Quadratura

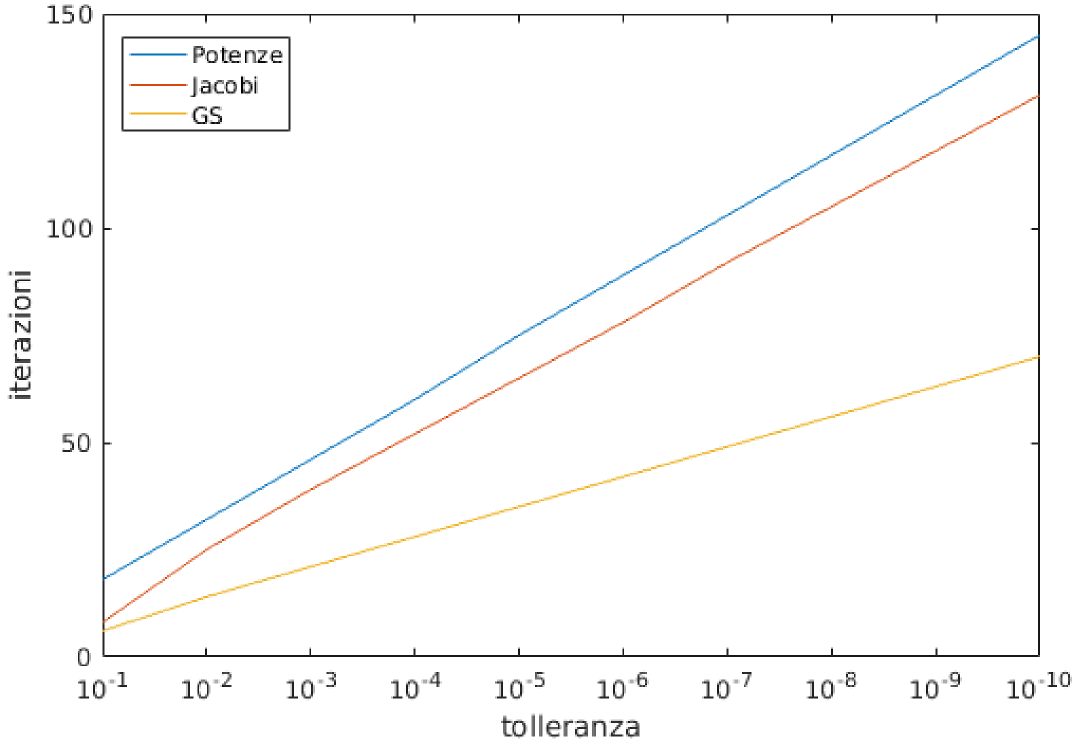

Figure 22: Comparazione dei vari metodi per il calcolo dell'autovettore dominante del PageRank