



Elaborato di
Calcolo Numerico
Anno Accademico 2019/2020

Niccolò *Piazzesi* - 6335623 - niccolo.piazzesi@stud.unifi.it
Pietro *Bernabei* - 6291312 - pietro.bernabei@stud.unifi.it

Contents

| | | |
|----------|-------------------------|-----------|
| 1 | Capitolo 1 | 4 |
| 1.1 | Esercizio 1 | 4 |
| 1.2 | Esercizio 2 | 4 |
| 1.3 | Esercizio 3 | 4 |
| 2 | Capitolo 2 | 5 |
| 2.1 | Esercizio 4 | 5 |
| 2.2 | Esercizio 5 | 5 |
| 2.3 | Esercizio 6 | 8 |
| 2.4 | Esercizio 7 | 9 |
| 3 | Capitolo 3 | 12 |
| 3.1 | Esercizio 8 | 12 |
| 3.2 | Esercizio 9 | 12 |
| 3.3 | Esercizio 10 | 13 |
| 3.4 | Esercizio 11 | 13 |
| 3.5 | Esercizio 12 | 14 |
| 3.6 | Esercizio 13 | 14 |
| 3.7 | Esercizio 14 | 15 |
| 4 | Capitolo 4 | 16 |
| 4.1 | Esercizio 15 | 16 |
| 4.2 | Esercizio 16 | 17 |
| 4.3 | Esercizio 17 | 17 |
| 4.4 | Esercizio 18 | 18 |
| 4.5 | Esercizio 19 | 18 |
| 4.6 | Esercizio 20 | 18 |
| 5 | Capitolo 5 | 20 |
| 5.1 | Esercizio 21 | 20 |
| 5.2 | Esercizio 22 | 21 |
| 5.3 | Esercizio 23 | 22 |
| 5.4 | Esercizio 24 | 22 |
| 5.5 | Esercizio 25 | 22 |
| 6 | Codici ausiliari | 25 |
| 6.1 | Esercizio 6 | 25 |
| 6.2 | Esercizio 7 | 25 |
| 6.3 | Esercizio 15 | 26 |
| 6.4 | Esercizio 16 | 26 |
| 6.5 | Esercizio 18 | 26 |
| 6.6 | Esercizio 19 | 27 |
| 6.7 | Esercizio 20 | 27 |
| 6.8 | Esercizio 21 | 28 |
| 6.9 | Esercizio 22 | 28 |
| 6.10 | Esercizio 23 | 28 |
| 6.11 | Esercizio 24 | 28 |
| 6.12 | Esercizio 25 | 29 |

List of Figures

| | | |
|---|--|----|
| 1 | iterazioni richieste | 8 |
| 2 | risultati interpolazione | 16 |
| 3 | risultati interpolazione hermite | 17 |

List of Tables

| | | |
|---|--|----|
| 1 | valori approssimati | 8 |
| 2 | valori approssimati | 13 |
| 3 | valori approssimati | 15 |
| 4 | pesi della formula di Newton-Cotes fino al settimo grado | 20 |
| 5 | risultati | 24 |

1 Capitolo 1

1.1 Esercizio 1

Sia $f(x)$ una funzione sufficientemente regolare e sia $h > 0$ una quantità abbastanza "piccola". Possiamo sviluppare i termini $f(x-h)$ e $f(x+h)$ mediante il polinomio di Taylor:

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

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

Sostituiamo i termini nell'espressione iniziale:

$$\begin{aligned} & \frac{f(x-h) - 2f(x) + f(x+h)}{h^2} = \\ = & \frac{f(x) - hf'(x) + \frac{h^2}{2}f''(x) - \frac{h^3}{6}f'''(x) + O(h^4) - 2f(x) + f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(x) + O(h^4)}{h^2} = \\ = & \frac{h^2f''(x) + O(h^4)}{h^2} = f''(x) + O(h^2) \end{aligned}$$

1.2 Esercizio 2

Eseguendo lo script si ottiene $u = 1.1102 \cdot 10^{-16} = \frac{\epsilon}{2}$, dove ϵ è la precisione di macchina. Il controllo interno ci dice che si esce dal ciclo solamente quando u diventa talmente piccolo che la somma $1+u$ viene percepita dal calcolatore come uguale a 1. Questo avviene se $u < \epsilon$, e la prima iterazione in cui il controllo risulta vero è proprio quando $u = \frac{\epsilon}{2}$. Il codice può quindi essere utilizzato per calcolare la precisione di macchina di un calcolatore, moltiplicando per 2 il valore di u restituito.

1.3 Esercizio 3

Quando si esegue $a - a + b$ il risultato è 100 mentre quando si esegue $a + b - a$ si ottiene 0. La differenza dei risultati è dovuta al fenomeno della cancellazione numerica:

- nel primo caso la sottrazione avviene sullo stesso numero $a = 10^{20}$. Sottrarre un numero da se stesso ha sempre risultato esatto 0 e quindi il risultato finale è corretto
- nel secondo caso la sottrazione avviene tra i termini $a+b = 10^{20} + 100$ e $a = 10^{20}$. $a+b$ ha le prime 18 cifre in comune con a e, a causa degli errori di approssimazione, le ultime tre cifre vengono cancellate dalla sottrazione, dando 0 come risultato finale.

2 Capitolo 2

2.1 Esercizio 4

```
1 function x1=radn(x, n)
2 %
3 % x1=radn(n,x)
4 % funzione Matlab che implementa il metodo di newton per il calcolo della
5 % radice n-esima di un numero positivo x
6 %
7 format long e
8 imax=1000;
9 tol=eps;
10 if x<=0
11     error('valore in ingresso errato');
12 end
13 x1=x/2;
14 for i=1:imax
15     x0=x1;
16     fx=x0^n-x;
17     fx1=(n)*x0^(n-1);
18     x1=x0-fx/fx1;
19     if abs(x1-x0)<=tol
20         break
21     end
22 end
23 if abs(x1-x0)>tol
24     error('metodo non converge')
25 end
26 end
```

2.2 Esercizio 5

- Metodo di bisezione

```
1 function [x,i] = bisezione(f,a,b,tol)
2 %bisez
3 %[x,i]=bisezione(f, a, b, tol, maxit)
4 %Pre: f continua in [a,b]
5 % Applica il metodo di bisezione per il calcolo della
6 % radice dell'equazione f(x)=0
7 % f      -funzione
8 % a, b    - estremi dell'intervallo
9 %
10 % tol     -tolleranza
11 % restituisce in x l'approssimazione della radice e in i il numero di iterazioni
12 % VEDI ANCHE: newton, corde, secanti, aiten, newtonmod
13     format long e
14     fa = feval(f,a);
15     fb = feval(f,b);
16     if(fa * fb > 0 )
17         error('gli estremi hanno lo stesso segno');
18     end
19     x0=a;
20     imax = ceil(log2(b-a) - log2(tol));
21     for i = 1:imax
22         x = (a+b)/2;
23         fx = feval(f,x);
```

```

24         if abs(x-x0) <= tol*(1+abs(x0))
25             break
26         end
27         x0=x;
28         if fa*fx<0
29             b = x;
30             fb = fx;
31         else
32             a = x;
33             fa = fx;
34         end
35     end
36
37 end

```

- Metodo di Newton

```

1 function [x,i] = newton( f, f1, x0, tol, maxit )
2 %newton
3 %[x,i]=newton(f,f1, x0, tol, maxit)
4 %Pre: f derivabile
5 % Applica il metodo di newton per il calcolo della
6 % radice dell'equazione f(x)=0
7 % f      --funzione
8 % f1     --derivata di f
9 % x0     --approssimazione iniziale
10 % tol    --tolleranza
11 % maxit  --numero massimo di iterazioni(default=100)
12 % restituisce in x l'approssimazione della radice e in i il numero di iterazioni
13 % VEDI ANCHE: bisezione, corde, secanti, aitken, newtonmod
14
15     format long e
16     if nargin<4
17         error('numero argomenti insufficienti');
18     elseif nargin==4
19         maxit = 100;
20     end
21     if tol<eps
22         error('tolleranza non idonea');
23     end
24     x = x0;
25     for i = 1:maxit
26         fx = feval( f, x );
27         f1x = feval( f1, x );
28         x = x - fx/f1x;
29         if abs(x-x0)<=tol*(1+abs(x0))
30             break;
31         else
32             x0 = x;
33         end
34     end
35     if abs(x-x0) > tol*(1+abs(x0))
36         error('metodo non converge');
37     end
38 end

```

- Metodo delle secanti

```

1 function [x, i]=secanti(f,x0,x1,tolx,maxit)
2 %secanti
3 %[x,i]=secanti(f, x0, x1, tolx, maxit)
4 %
5 % Applica il metodo delle secanti per il calcolo della
6 % radice dell'equazione f(x)=0
7 % f      -funzione
8 % x0     -approssimazione iniziale
9 % x1     -seconda approssimazione iniziale
10 % tolx   -tolleranza
11 % maxit  -numero massimo di iterazioni(default=100)
12 % restituisce in x l'approssimazione della radice e in i il numero di iterazioni
13 % VEDI ANCHE: bisezione, newton, corde, aitken, newtonmod
14
15 format long e
16 if nargin<4
17     error('numero argomenti insufficienti');
18 elseif nargin==4
19     maxit = 100;
20 end
21 i=0;
22 f0=feval(f,x0);
23 for i=1:maxit
24     f1=feval(f,x1);
25     df1=(f1-f0)/(x1-x0);
26     x=x1-(f1/df1);
27     if abs(x1-x0)<=tolx*(1+abs(x0))
28         break;
29     end
30     x0=x1;
31     x1=x;
32     f0=f1;
33
34 end
35 if abs(x-x0) > tolx*(1+abs(x0))
36     error('metodo non converge');
37 end
38 end

```

- Metodo delle corde

```

1 function [x,i] = corde( f, f1, x0, tolx, maxit )
2 %corde
3 %[x,i]=corde(f,f1, x0, tolx, maxit)
4 %Pre: f derivabile
5 % Applica il metodo delle corde per il calcolo della
6 % radice dell'equazione f(x)=0
7 % f      -funzione
8 % f1     -derivata di f
9 % x0     -approssimazione iniziale
10 % tolx   -tolleranza
11 % maxit  -numero massimo di iterazioni(default=100)
12 % restituisce in x l'approssimazione della radice e in i il numero di iterazioni
13 % VEDI ANCHE: bisezione, newton, secanti, aitken, newtonmod
14
15 format long e
16 if nargin<4
17     error('numero argomenti insufficienti');
18 elseif nargin==4

```

```

19         maxit = 100;
20     end
21     if tolx<eps
22         error('tolleranza non idonea');
23     end
24     flx = feval(f1, x0);
25     x = x0;
26     for i = 1:maxit
27         fx = feval( f, x );
28         if fx==0
29             break;
30         end
31         x = x - fx/flx;
32         if abs(x-x0)<=tolx*(1+abs(x0))
33             break;
34         else
35             x0 = x;
36         end
37     end
38     if abs(x-x0) > tolx*(1+abs(x0))
39         error('metodo non converge');
40     end
41 end

```

2.3 Esercizio 6

Eseguendo lo script es6.msi ottengono i risultati contenuti nella tabella 2 e nella figura 1. Come si può notare, il metodo di newton e il metodo delle secanti convergono molto più rapidamente del metodo di bisezione e del metodo delle corde.

| Metodo | tolleranza= 10^{-3} | tolleranza= 10^{-6} | tolleranza= 10^{-9} | tolleranza= 10^{-12} |
|-----------|-----------------------|-----------------------|-----------------------|------------------------|
| bisezione | 0.739257812500000 | 0.739085197448730 | 0.739085133187473 | 0.739085133215667 |
| newton | 0.739085133385284 | 0.739085133215161 | 0.739085133215161 | 0.739085133215161 |
| corde | 0.739567202212256 | 0.739084549575213 | 0.739085132739254 | 0.739085133215737 |
| secanti | 0.739085133215001 | 0.739085133215161 | 0.739085133215161 | 0.739085133215161 |

Table 1: valori approssimati

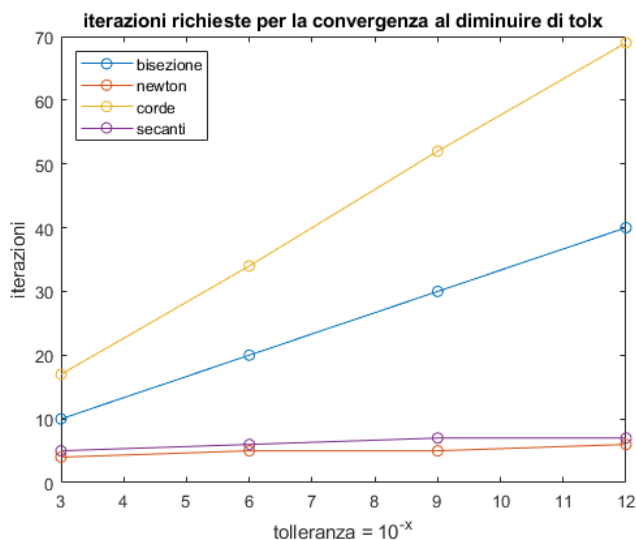


Figure 1: iterazioni richieste

2.4 Esercizio 7

Le nuove funzioni utilizzate in questo esercizio sono:

- Metodo di Newton modificato

```
1 function [x, i] = newtonmod( f, f1, x0, m, tolX, maxit )
2 %NEWTONMOLT
3 %[x,i]=Newtonmolt(f,f1,x0,m,tolX,maxit)
4 % Pre: f derivabile
5 % Applica il metodo di Newton per il calcolo della
6 % radice (di molteplicita' nota r) dell'equazione f(x)=0
7 % f      -funzione
8 % f1     -derivata di f
9 % x0     -approssimazione iniziale
10 % m      -molteplicita' della radice
11 % tolX   -tolleranza
12 % maxit  -numero massimo di iterazioni(default=100)
13 % restituisce in x l'approssimazione della radice e in i il numero di iterazioni
14 % VEDI ANCHE: bisezione, newton, secanti, corde, aitken
15
16     format long e
17     if nargin<5
18         error('numero argomenti insufficienti');
19     elseif nargin==5
20         maxit = 100;
21     end
22     if tolX<eps
23         error('tolleranza non idonea');
24     end
25     x = x0;
26     for i = 1:maxit
27         fx = feval( f, x );
28         f1x = feval( f1, x );
29         if fx==0
30             break;
31         end
32         x = x - m*(fx/f1x);
33         if abs(x-x0)<=tolX*(1+abs(x0))
34             break;
35         else
36             x0 = x;
37         end
38     end
39
40 end
```

- Metodo delle accelerazioni di Aitken

```
1 <<<<<< HEAD
2 function [x,i]=aitken(f,f1,x0,tolX, itmax)
3     %Metodo di accelerazione di Aitken
4     %
5     % [x,i]=aitken(f,f1,x0,itmax,tolX,rtolX)
6     %
7     %Questo metodo prende in input:
8     % f: la funzione di cui si vuol trovare uno zero
9     % f1: la derivata della funzione
10    % x0: valore di x di partenza
11    % imax: numero massimo di iterazioni consentite
```

```

12 % tolx: tolleranza assoluta sul valore dello zero
13 %Questo metodo restituisce:
14 % x: zero della funzione
15 % i: numero di iterazioni fatte
16 i=0;
17 fx = feval(f,x0);
18 if fx==0
19     x=x0;
20     return
21 end
22 flx = feval(f1,x0);
23 if flx==0
24     error('La derivata prima ha assunto valore zero, impossibile
        continuare!')
25 end
26 x= x0-fx/flx;
27 go = 1;
28 while (i<itmax) && go
29     i = i+1;
30     x0 = x;
31     fx = feval(f,x0);
32     flx = feval(f1,x0);
33     if flx==0
34         %In questo caso non possiamo andare avanti, rimane solo da
            controllare
35         %se per caso abbiamo trovato una soluzione esatta o almeno nella
            tolleranza %richiesta
36         if fx == 0
37             %Abbiamo trovato una soluzione esatta
38             return
39         elseif (abs(x-x0)<=(tolx*(1 + abs(x0))))
40             %Abbiamo trovato una soluzione nella tolleranza richiesta
41             return
42         end
43         %Evitiamo una divisione per zero.
44         error('La derivata prima ha assunto valore zero, impossibile
            continuare!')
45     end
46     x1 = x0-fx/flx;
47     fx = feval(f,x1); flx = feval(f1,x1);
48     if flx==0
49         if fx == 0
50             return
51         elseif (abs(x-x0)<=(tolx*(1 + abs(x0))))
52             return
53         end
54         error('La derivata prima ha assunto valore zero, impossibile
            continuare!')
55     end
56     x = x1 - fx/flx;
57     t = ((x-2*x1)+x0);
58
59     if t == 0
60         if feval(f,x) == 0
61             return
62         end
63         error('Impossibile determinare la radice nella tolleranza
            desiderata')

```

```

64         end
65         x = (x*x0-x1^2)/t;
66         go = (abs(x-x0)>(tolx*(1 + abs(x0))));
67     end
68     if go, disp('Il metodo non converge.'), end
69 =====
70 function [x, i] = aitken( f, f1, x0, tolx, maxit )
71 %aitken
72 %[x,i]=aitken(f,f1, x0, tolx, maxit)
73 % Pre: f derivabile
74 % Applica il metodo di accelerazione di aitken per il calcolo della
75 % radice (di molteplicita' incognita) dell'equazione f(x)=0
76 % f      -funzione
77 % f1     -derivata di f
78 % x0     -approssimazione iniziale
79 % tolx   -tolleranza
80 % maxit  -numero massimo di iterazioni(default=100)
81 % restituisce in x l'approssimazione della radice e in i il numero di iterazioni
82 % VEDI ANCHE: bisezione, newton, secanti, corde, newtonmod
83 format long e
84 if nargin<4
85     error('numero argomenti insufficienti');
86 elseif nargin==4
87     maxit = 100;
88 end
89 if tolx<eps
90     error('tolleranza non idonea');
91 end
92 x = x0;
93 for i = 1:maxit
94     fx = feval( f, x0 );
95     f1x = feval( f1, x0 );
96     if f1x==0
97         break;
98     end
99     x1 = x0 - fx/f1x;
100    fx = feval( f, x1 );
101    f1x = feval( f1, x1 );
102    if f1x==0
103        break;
104    end
105    x = x1 -fx/f1x;
106    x = (x*x0-x1^2)/(x-2*x1+x0);
107    if abs(x-x0)<=tolx*(1+abs(x0))
108        break;
109    else
110        x0 = x;
111    end
112 end
113 if abs(x-x0) > tolx*(1+abs(x0))
114     error('metodo non converge');
115 end
116 end
117 >>>>>> 5cd1ff828560c11c9f921316a0662e417fff06e4

```

La radice nulla della funzione $f(x) = x^2 \tan(x)$ ha molteplicita $m = 3$, in quanto 0 annulla due volte il termine x^2 e $\tan(0) = 0$.

3 Capitolo 3

3.1 Esercizio 8

```
1 function [LU,p]=palu(A)
2 % [LU,p]=palu(A)
3 % funzione che dato in input matrice A restituisce matrice fattorizzata LU
4 % e il relativo vettore p di permutazione di LU con pivoting parziale di A
5 % input:
6 %   A= matrice di cui si vuole calcolare la fattorizzazione lu con pivoting
7 %   parziale
8 % output:
9 %   LU=matrice quadrata di dimensioni n*n, composta dalla matrice
10 %   triangolare superiore U e la matrice triangolare inferiore a diagonale
11 %   unitaria L
12 %   p= vettore di permutazione di dimensione n, generato dalla
13 %   fattorizzazione di A con pivoting parziale
14 %
15
16 [n,m]=size(A);
17 if(n~=m)
18     error(matrice A non quadrata);
19 end
20 LU=A;
21 p=[1:n];
22 for i=1:n-1
23     [mi,ki]=max(abs(LU(i:n,i)));
24     if mi==0
25         error('La matrice e'' non singolare')
26     end
27     ki=ki+i-1;
28     if ki>i
29         p([i ki])=p([ki i]);
30         LU([i ki],:)= LU([ki i],:);
31     end
32     LU(i+1:n,i)=LU(i+1:n,i)/LU(i,i);
33     LU(i+1:n,i+1:n)=LU(i+1:n,i+1:n)-LU(i+1:n,i)*LU(i,i+1:n);
34 end
35 return
36 end
```

3.2 Esercizio 9

```
1 function x=LUsolve(LU,p,b)
2 %
3 % funzione che risolve il sistema lineare LUx=b(p):
4 %input:
5 %   LU=matrice quadrata (n*n) fattorizzata LU, ottenuta attrarso la
6 %   fattorizzazione con pivoting parziale
7 %   p= vettore di permutazione per b, di dimensione n, con valori da (1 a
8 %   n)
9 %   b=vettore dei termini noti
10 %output:
11 %   x=vettore delle incognite calcolate
12 %
13 %
14 [m,n]=size(LU);
```

```

15     if(m~=n || n~=length(b)) error('dati inconsistenti')
16     else if(min(abs(diag(LU)))==0)
17         error(fattorizzazione errata);
18     end
19 end
20 x=b(p);
21 for i=1:n-1
22     x(i+1:n)=x(i+1:n)-(LU(i+1:n,i)*x(i));
23 end
24 x(n)=x(n)/LU(n,n);
25 for i=n-1:-1:1
26     x(1:i)=x(1:i)-(LU(1:i,i+1)*x(i+1));
27     x(i)=x(i)/LU(i,i);
28 end
29 return
30 end

```

3.3 Esercizio 10

| i | Sigma | Norma |
|----|-----------|------------|
| 1 | 10^{-1} | 8.9839e-15 |
| 2 | 10^1 | 1.4865e-14 |
| 3 | 10^3 | 1.3712e-12 |
| 4 | 10^5 | 1.2948e-10 |
| 5 | 10^7 | 5.3084e-09 |
| 6 | 10^9 | 1.0058e-06 |
| 7 | 10^{11} | 8.5643e-05 |
| 8 | 10^{13} | 0.0107 |
| 9 | 10^{15} | 0.9814 |
| 10 | 10^{17} | 4.1004e+03 |

Table 2: valori approssimati

Tabella che composta: i=indice dell'iterazione; Sigma=valore calcolato e usato dalla funzione linsis(), per introdurre un errore nella matrice generata A e nel suo vettore dei termini noti b, che cresce al crescere dell'iterazione. Norma= valore della distanza tra il vettore x, soluzione del sistema lineare $LU \cdot x = b$, il quale è affetto da errore, e il vettore xref, soluzione corretta del sistema. Da questa tabella quindi si può notare come all'incremento della iterazione, e quindi della sigma, l'errore nelle soluzioni cresce, quasi proporzionalmente come sigma, con un fattore di 10^2 ;

3.4 Esercizio 11

```

1 function QR = myqr(A)
2 %   QR = myqr(A)
3 %   calcola la fattorizzazione QR di Householder della matrice A
4 %   Input:
5 %       A= matrice quadrata da fattorizzare
6 %
7 %   Output:
8 %       QR=matrice contenente le informazioni sui fattori Q e R della
9 %       fattorizzazione QR di A
10 %
11 [m,n] = size(A);
12 if n > m
13     error('Dimensioni errate');
14 end
15 QR = A;

```

```

16   for i = 1:n
17       alfa = norm(QR(i:m,i));
18       if alfa == 0
19           error('la matrice non ha rango massimo');
20       end
21       if QR(i,i) >= 0
22           alfa = -alfa;
23       end
24       v1 = QR(i,i) - alfa;
25       QR(i,i) = alfa;
26       QR(i+1:m,i) = QR(i+1:m,i)/v1;
27       beta = -v1/alfa;
28       v = [1; QR(i+1:m,i)];
29       QR(i:m,i+1:n) = QR(i:m,i+1:n) - (beta * v) * (v' * QR(i:m,i+1:n));
30   end
31 end

```

3.5 Esercizio 12

```

1  function x = qrsolve(QR, b)
2  %
3  %
4  %   x = qrSolve(QR, b)
5  %   risolve il sistema QR*x=b nel senso dei minimi quadrati.
6  %   Input:
7  %       QR=matrice contenente le informazioni Q e R della
8  %       fattorizzazione di una matrice quadrata A
9  %       b=termine noto del sistema lineare
10 %   Output:
11 %       x=vettore delle soluzioni del sistema lineare
12 %
13 [m,n] = size(QR);
14 k = length(b);
15 if k ~= m
16     error('Dati inconsistenti');
17 end
18 x=b(:);
19 for i = 1:n
20     v=[1; QR(i+1:m,i)];
21     beta = 2/(v'* v);
22     x(i:m) = x(i:m) - beta*(v'*x(i:m))*v;
23 end
24 x=x(1:n);
25 for j = n:-1:1
26     if QR(j,j)==0
27         error('Matrice singolare');
28     end
29     x(j) = x(j) / QR(j,j);
30     x(1:j-1) = x(1:j-1) - QR(1:j-1,j)*x(j);
31 end
32 return
33 end

```

3.6 Esercizio 13

```

1  A= [1, 2, 3; 1 2 4; 3 4 5; 3 4 6; 5 6 7];

```

```

2 b=[14 17 26 29 38];
3 QR=myqr(A);
4 ris=qr.solve(QR,b);
5 disp(ris);

```

Il risultato finale è $ris = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$

3.7 Esercizio 14

| A\b | (A'*A)\(A'*b) |
|--------|---------------|
| 1.0000 | 3.5759 |
| 2.0000 | -3.4624 |
| 3.0000 | 9.5151 |
| 4.0000 | -1.2974 |
| 5.0000 | 7.9574 |
| 6.0000 | 4.9125 |
| 7.0000 | 7.2378 |
| 8.0000 | 7.9765 |

Table 3: valori approssimati

L'espressione $A \backslash b$ risolve in matlab, il sistema di equazioni lineari nella forma matriciale $A \cdot x = b$ per X .

L'espressione $(A' \cdot A) \backslash (A' \cdot b)$ impiega lo stesso operatore \backslash , quindi risolve il sistema delle equazioni lineari delle due parentesi. La Matrice A viene calcolata usando la funzione `vander()` che genera una matrice di tipo Vandermonde, la quale è mal condizionata. Usando la funzione `cond()` sulla matrice A si ottiene una condizionamento pari a: $1.5428e+09$. Eseguendo poi la moltiplicazione della prima parentesi tonda, il condizionamento è pari a: $4.4897e+18$. Andando così ad eseguire una divisione tra una matrice mal condizionata e un vettore, il risultato presenta degli errori.

4 Capitolo 4

4.1 Esercizio 15

Eseguendo il codice es15.m si ottengono i seguenti risultati:

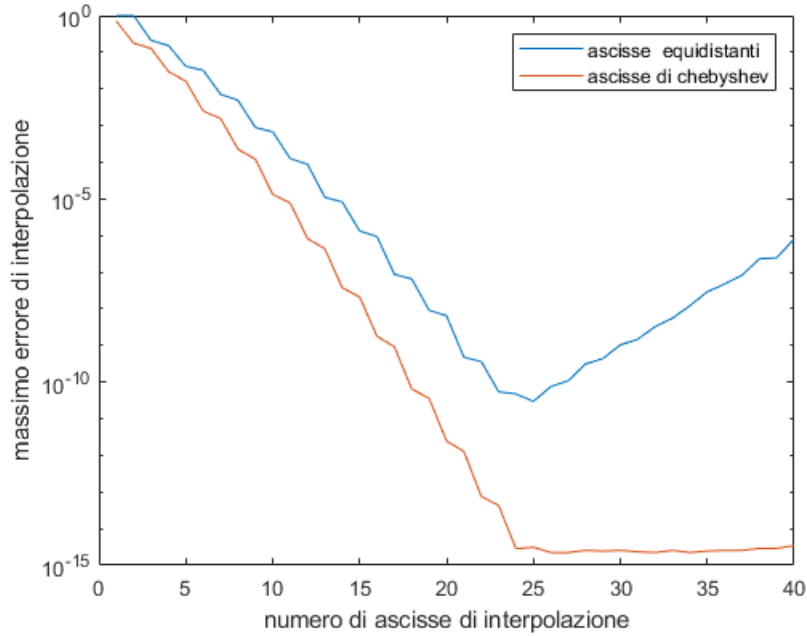


Figure 2: risultati interpolazione

Per le ascisse di chebyshev, si ha una decrescita esponenziale dell'errore massimo per $n \leq 25$ per poi assestarsi a circa $2 \cdot 10^{-15}$ per n successivi. Per quanto riguarda le ascisse equidistanti invece, si può notare come l'errore massimo torni a crescere esponenzialmente per $n > 25$. I risultati confermano il mal condizionamento del problema di interpolazione polinomiale quando vengono usate ascisse d'interpolazione equidistanti,

4.2 Esercizio 16

Eseguendo es16.m si ottiene:

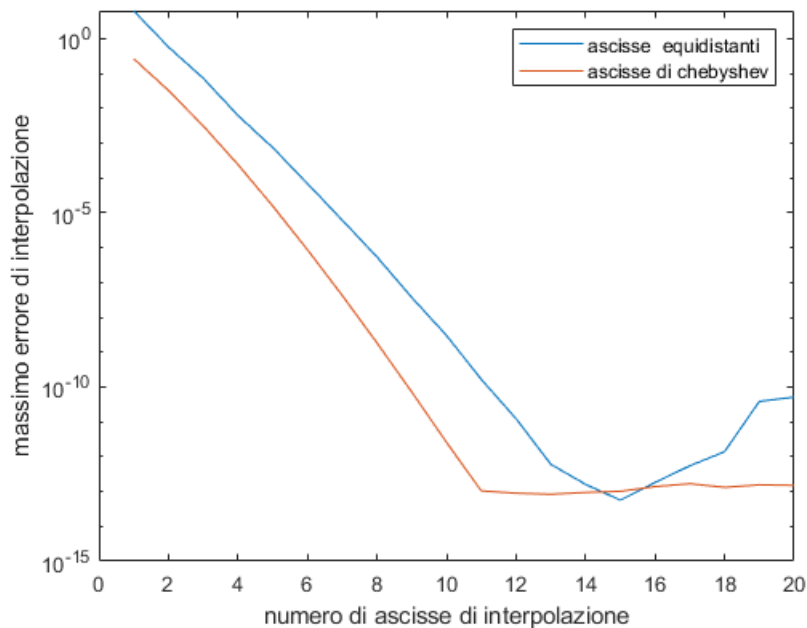


Figure 3: risultati interpolazione hermite

Si può notare come, rispetto all'interpolazione classica, l'errore decresca più rapidamente per $n \leq 15$. Anche in questo caso l'errore commesso usando le ascisse di chebyshev è migliore in confronto al caso delle ascisse equidistanti (eccetto per $n = 15$).

4.3 Esercizio 17

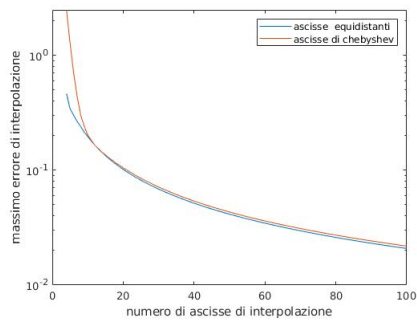
```
1 function output=splinenat(x,y,xq)
2 %
3 % output=splinenat(x,y,xq)
4 %funzione che calcola la spline cubica naturale.
5 %Input:
6 % x=vettore delle ascisse su cui calcolare la spline
7 % y=vettore dei valori di f(x), con x ascissa
8 % xq= insieme delle ascisse di cui si vuole sapere il valore della spline
9 %Output:
10 % output=vettore delle approssimazioni sulle ascisse xq
11 %
12 n=length(x);
13 l=length(xq);
14 if(length(y)~=n), error('dati in input con dimensioni differenti');end
15 [x1,i]=sort(x);
16 y1=y(i);
17 m=spline0(x1,y1);
18 h=diff(x1);
19 df = diff(y1)./h;
20 r=y(1:(n-1))-((h(1:n-1).^2)/6)*m(2:n);
21 q=df(1:n-1)-h(1:n-1)*(m(2:n)-m(1:n-1));
22 output=zeros(l,1);
23 for i=1:l
24     indg=find(xq(i)<=x1(2:n),1)+1;
```

```

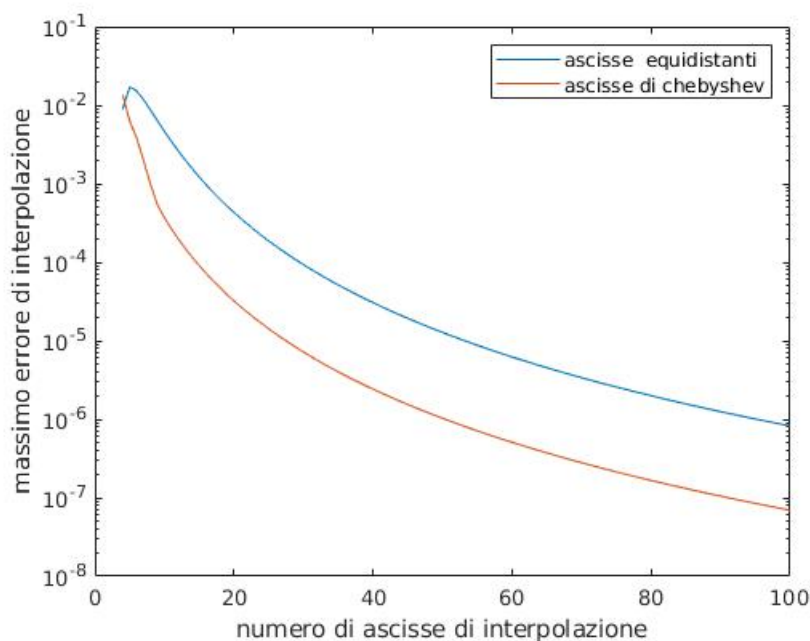
25 indp=find(xq(i)>=x1(1:n-1),1,'last');
26 output(i)=(((xq(i)-x1(indp)).^3).*m(indg)+((x1(indg)-xq(i))^3).*m(indp))/(6*h(indp))+q(
    indp).*(xq(i)-x1(indp))+r(indp);
27 end
28 return
29 end

```

4.4 Esercizio 18



4.5 Esercizio 19



4.6 Esercizio 20

```

1 function y = minimiquadrati(xi, fi, m)
2 %
3 %
4 % y = minimiquadrati(xi, fi, m)
5 % calcola il valore del polinomio di approssimazione ai minimi quadrati di grado m
6 % sulle ascisse xi. fi contiene i valori approssimati di una funzione f valutata su xi
7 if length(unique(xi)) < m+1
8     error('ascisse distinte non sufficienti');
9 end
10 fi = fi(:);
11 V = fliplr(vander(xi));

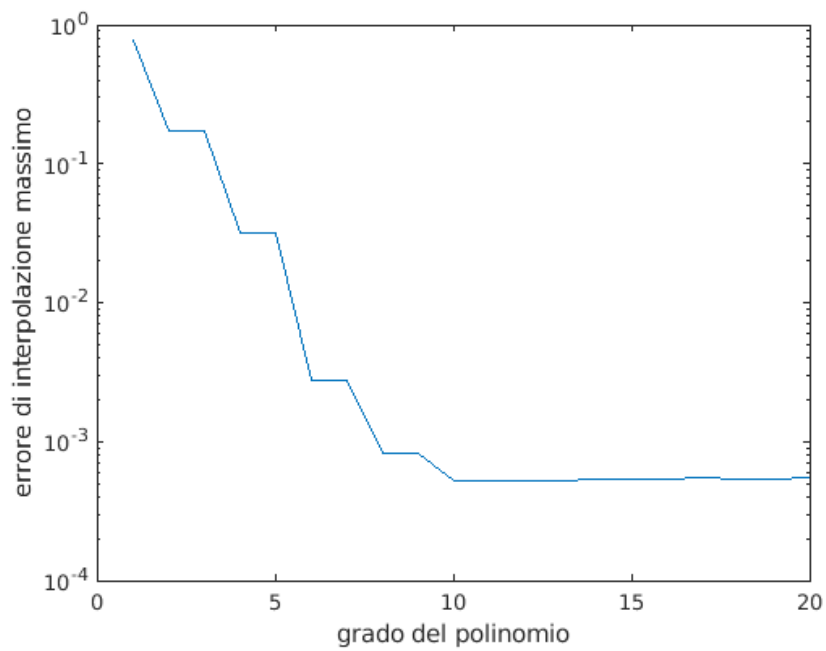
```

```

12 V = V(1:end, 1:m+1);
13 QR = myqr(V);
14 p = qrsolve(QR, fi);
15 y = p(m+1)*ones(size(xi));
16 for i = 0:m-1
17     y = y.*xi+p(m-i);
18 end
19 end

```

Eseguendo es20.m si ottiene:



Si nota una decrescita dell'errore esponenziale fino a $m = 10$, dove si assesta tra 10^{-3} e 10^{-4} .

5 Capitolo 5

5.1 Esercizio 21

```

1 function c = ncweights(n)
2 %
3 %
4 % c = nc-weights(n)
5 % calcola i pesi della formula di newton cotes di grado n;
6 %
7 if n<=0
8     error('grado della formula non positivo');
9 end
10 c=zeros(1,floor(n / 2 + 1));
11 for j = 1:(ceil((n+1)/2))
12     temp = (0:n);
13     vj = temp(j);
14     temp(j)=[];
15     f = @(x)(prod(x-temp) /prod(vj-temp));
16     c(j) = integral(f, 0, n, 'ArrayValued', true);
17 end
18 c = [c flip(c)]; %sfrutto la simmetria dei pesi
19 if mod(n,2)==0
20     %elimino la copia del valore centrale prodotta da flip(c) e che risulta di troppo per
        n pari
21     c(n/2+1) = [];
22 end
23 return
24 end

```

Eseguendo lo script es21.m si ottiene:

| $n \setminus c_{in}$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|----------------------|-------------------|-------------------|-------------------|-------------------|-------------------|-------------------|-------------------|-------------------|
| 1 | $\frac{1}{2}$ | $\frac{1}{2}$ | | | | | | |
| 2 | $\frac{1}{3}$ | $\frac{4}{3}$ | $\frac{1}{3}$ | | | | | |
| 3 | $\frac{3}{8}$ | $\frac{9}{8}$ | $\frac{9}{8}$ | $\frac{3}{8}$ | | | | |
| 4 | $\frac{14}{45}$ | $\frac{64}{45}$ | $\frac{8}{15}$ | $\frac{64}{45}$ | $\frac{14}{45}$ | | | |
| 5 | $\frac{95}{288}$ | $\frac{95}{288}$ | $\frac{95}{288}$ | $\frac{95}{288}$ | $\frac{95}{288}$ | $\frac{95}{288}$ | | |
| 6 | $\frac{41}{140}$ | $\frac{54}{35}$ | $\frac{27}{140}$ | $\frac{68}{35}$ | $\frac{27}{140}$ | $\frac{54}{35}$ | $\frac{41}{140}$ | |
| 7 | $\frac{108}{355}$ | $\frac{810}{559}$ | $\frac{343}{640}$ | $\frac{649}{536}$ | $\frac{649}{536}$ | $\frac{343}{640}$ | $\frac{810}{559}$ | $\frac{108}{355}$ |

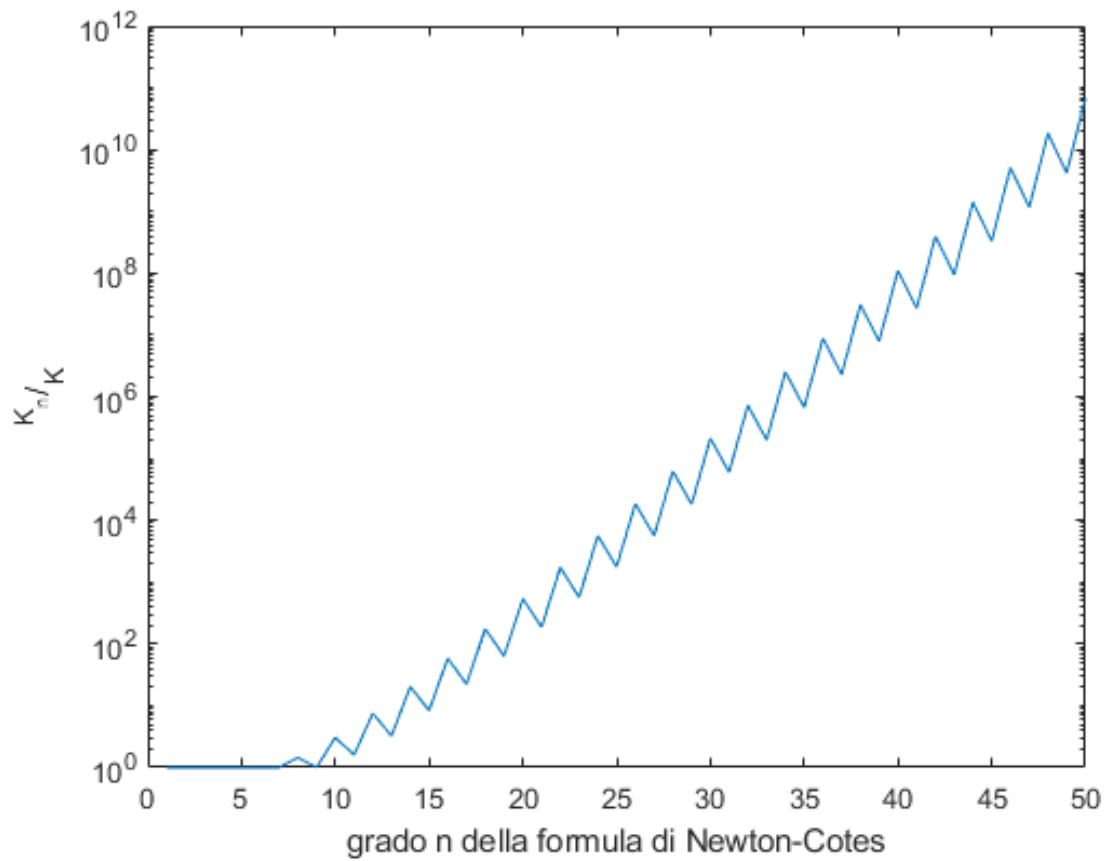
Table 4: pesi della formula di Newton-Cotes fino al settimo grado

5.2 Esercizio 22

Sappiamo che $k = (b - a)$ e $k_n = (b - a) \frac{1}{n} \sum_{i=0}^n |c_{in}|$. Il rapporto sarà dunque dato da:

$$\frac{k_n}{k} = \frac{(b - a) \frac{1}{n} \sum_{i=0}^n |c_{in}|}{b - a} = \frac{1}{n} \sum_{i=0}^n |c_{in}|$$

Calcolando $\frac{k_n}{k}$ per $n = 1, \dots, 50$ (es22.m) si ottiene:



5.3 Esercizio 23

```
1 function y = newtoncotes(f,a, b, n)
2 %
3 % y= newtoncotes(f,a,b, n)
4 % calcola l'approssimazione dell'integrale definito per la funzione f sull'intervallo [a,
   b],
5 % utilizzando la formula di newton cotes di grado n.
6 %
7
8 if a > b || n < 0
9     error('dati inconsistenti');
10 end
11 xi = linspace(a, b, n+1);
12 fi = feval(f, xi);
13 h = (b-a) / n;
14 c = ncweights(n);
15 y = h*sum(fi.*c);
16 return
17 end
```

RISULTATI PER N DA 1 A 9(es23.m):

| grado della formula | valore integrale | errore |
|---------------------|------------------|--------|
| 1 | 0.428 | 0.253 |
| 2 | 0.213 | 0.038 |
| 3 | 0.196 | 0.021 |
| 4 | 0.180 | 0.005 |
| 5 | 0.179 | 0.004 |
| 6 | 0.176 | 0.001 |
| 7 | 0.176 | 0.001 |
| 8 | 0.175 | 0.000 |
| 9 | 0.175 | 0.000 |

5.4 Esercizio 24

| intervalli\formula | trapezi composita | simpson composita |
|--------------------|-------------------|-------------------|
| 2 | 0.266403558406035 | 0.266403558406035 |
| 4 | 0.203432804450016 | 0.182442553131343 |
| 6 | 0.188498346613972 | 0.177333443886033 |
| 8 | 0.182789408875225 | 0.175908277016961 |
| 10 | 0.180034803521960 | 0.175392868382289 |
| 12 | 0.178504015707472 | 0.175172572071972 |
| 14 | 0.177568218195411 | 0.175066546519247 |
| 16 | 0.176955413111201 | 0.175010747856527 |
| 18 | 0.176532709616469 | 0.174979254439942 |
| 20 | 0.176229037552030 | 0.174960448895386 |

5.5 Esercizio 25

```
1 function [I2, points] = adaptrap(f, a, b, tol, fa, fb)
2 %
3 %
4 % Syntax: [I2, points] = adaptrap(f, a, b, tol, fa, fb)
5 %
6 %
7 global points
```

```

8 delta = 0.5;
9 if nargin<=4
10     fa = feval(f, a );
11     fb = feval(f, b );
12     if nargin==2
13         points = [a fa; b fb];
14     else
15         points = [];
16     end
17 end
18 h = b-a;
19 x1 = (a+b)/2;
20 f1 = feval(f, x1);
21 if ~isempty(points)
22     points = [points; [x1 f1]];
23 end
24 I1 = .5*h*(fa+fb);
25 I2 = .5*(I1+h*f1);
26 e = abs(I2-I1)/3;
27 if e>tol || abs(b-a) > delta
28     I2 = adaptrap(f, a, x1, tol/2, fa, f1) + adaptrap(f, x1, b, tol/2, f1, fb);
29 end
30 return
31 end

```

```

1 function [I2, points] = adapsim(f, a, b, tol, fa, f1, fb)
2 % [I2, points] = adapsim(f, a, b, tol, fa, f1, fb)
3 % Approssimazione dell'integrale definito di f(x) con estremi a e b,
4 % mediante la formula adattiva di simpson
5 global points
6 delta = 0.5;
7 x1 = (a+b)/2;
8 if nargin<=4
9     fa = feval(f, a );
10    fb = feval(f, b );
11    f1 = feval(f, x1);
12    if nargin==2
13        points = [a fa;x1 f1; b fb];
14    else
15        points = [];
16    end
17 end
18 h = (b-a)/6;
19 x2 = (a+x1)/2;
20 x3 = (x1+b)/2;
21 f2 = feval(f, x2);
22 f3 = feval(f, x3);
23 if ~isempty(points)
24     points = [points; [x2 f2; x3 f3]];
25 end
26 I1 = h*(fa+fb+4*f1);
27 I2 = .5*h*(fa+4*f2+2*f1+4*f3+fb);
28 e = abs(I2-I1)/15;
29 if e>tol || abs(b-a) > delta
30     I2 = adapsim(f, a, x1, tol/2, fa, f2, f1) + adapsim(f, x1, b, tol/2, f1, f3, fb);
31 end
32 return
33 end

```

| tolleranza\formula | trapezi adattiva | simpson adattiva |
|--------------------|---------------------------------|---------------------------------|
| 10^{-2} | 0.295559711784128, punti = 21 | 0.281297643062670, punti = 17 |
| 10^{-3} | 0.294585368185034 , punti = 93 | 0.281297643062670, punti = 17 |
| 10^{-4} | 0.294274200873635, punti = 277 | 0.294259338419631, punti = 41 |
| 10^{-5} | 0.294230142164878, punti = 793 | 0.294227809768005, punti = 81 |
| 10^{-6} | 0.294226019603178, punti = 2692 | 0.294225764620384 , punti = 145 |

Table 5: risultati

Per ciascuna formula, l'operazione che comporta maggior costo computazionale ad ogni chiamata è la valutazione funzionale dei punti di un sottointervallo. Poichè ogni punto viene valutato una sola volta, possiamo confrontare il costo delle due formule andando a vedere quanti punti aggiuntivi sono stati utilizzati. Osservando i dati riportati nella tabella 5(raccolti in es25.m), è palese come la formula di simpson adattiva convergà più rapidamente rispetto alla formula dei trapezi adattiva.

6 Codici ausiliari

6.1 Esercizio 6

Listing 1: es6.m

```
1 f = @(x)(x-cos(x));
2 f1 = @(x)(1+sin(x));
3
4 x0 = 0;
5 x1 = 1;
6 x=zeros(4,4);
7 y= zeros(4, 4);
8 for i=3:3:12
9
10     [x(1, i/3), y(1, i/3)] = bisezione(f, x0, x1, 10^(-i));
11     [x(2, i/3), y(2, i/3)] = newton(f, f1, x0, 10^(-i));
12     [x(3, i/3), y(3, i/3)] = corde(f, f1, x0, 10^(-i));
13     [x(4,i/3), y(4, i/3)] = secanti(f, x0, x1, 10^(-i), 100);
14 end
15 row_names = {'bisezione', 'newton', 'corde', 'secanti'};
16 colnames = {'10^-3', '10^-6', '10^-9', '10^-12'};
17 values = array2table(x, 'RowNames', row_names, 'VariableNames', colnames);
18 disp(values)
19 figure
20 plot([3, 6, 9, 12], y, 'o-')
21 title('iterazioni richieste per la convergenza al diminuire di tol x')
22 xlabel('tolleranza = 10^{-x}')
23 ylabel('iterazioni')
24 legend({'bisezione', 'newton', 'corde', 'secanti'}, 'Location', 'northwest')
```

6.2 Esercizio 7

Listing 2: es7.m

```
1 f = @(x)(x^2*tan(x));
2 f1 = @(x)(2*x*tan(x) +(x^2)/(cos(x)^2));
3 m = 3;
4 x0 = 1;
5 y= zeros(3, 4);
6 x=-1*ones(3,4);
7 for i=3:3:12
8     [x(1, i/3), y(1, i/3)] = newton(f, f1, x0, 10^(-i));
9     [x(2, i/3), y(2, i/3)] = newtonmod(f, f1, x0, m, 10^(-i));
10    [x(3, i/3), y(3, i/3)] = aitken(f, f1, x0, 10^(-i));
11 end
12 disp(x);
13 disp(y);
14 row_names = {'newton', 'newton modificato', 'aitken'};
15 colnames = {'10^-3', '10^-6', '10^-9', '10^-12'};
16 values = array2table(x, 'RowNames', row_names, 'VariableNames', colnames)
17
18 format
19 iterations = array2table(y, 'RowNames', row_names, 'VariableNames', colnames)
20 plot([3, 6, 9, 12], y, '-')
21 title('iterazioni richieste per la convergenza al diminuire di tol x')
22 xlabel('tolleranza = 10^{-x}')
23 ylabel('iterazioni')
```

```
24 legend({'newton', 'newtonmod', 'aitken'}, 'Location', 'northwest')
```

6.3 Esercizio 15

Listing 3: es15.m

```
1 f = @(x)(cos((pi*x.^2)/2));
2 x = linspace(-1, 1, 100001);
3 linerrors = zeros(1, 40);
4 chebyerrors = zeros(1, 40);
5 for n = 1:40
6     xlin = linspace(-1, 1, n+1);
7     xcheby = chebyshev(-1,1,n+1);
8     ylin = lagrange(xlin,f(xlin),x);
9     ycheby = lagrange(xcheby,f(xcheby),x);
10    linerrors(n) = norm(abs(f(x) - ylin), inf);
11    chebyerrors(n) = norm( abs(f(x) - ycheby), inf);
12 end
13 semilogy(linerrors);
14 hold on;
15 semilogy(chebyerrors);
16 xlabel('numero di ascisse di interpolazione');
17 ylabel('massimo errore di interpolazione');
18 legend({'ascisse equidistanti', 'ascisse di chebyshev'}, 'Location', 'northeast');
```

6.4 Esercizio 16

Listing 4: es16.m

```
1 f = @(x)(cos((pi*x.^2)/2));
2 f1 = @(x)(-pi*x.*sin((pi*x.^2)/2));
3 x = linspace(-1, 1, 100001);
4 linerrors = zeros(1, 20);
5 chebyerrors = zeros(1, 20);
6 for n = 1:20
7     xlin = linspace(-1, 1, n+1);
8     xcheby = chebyshev(-1,1, n+1);
9     ylin = hermite(xlin,f(xlin),f1(xlin),x);
10    ycheby = hermite(xcheby,f(xcheby),f1(xcheby),x);
11    linerrors(n) = norm(abs(f(x) - ylin), inf);
12    chebyerrors(n) = norm( abs(f(x) - ycheby), inf);
13 end
14 semilogy(linerrors);
15 hold on;
16 semilogy(chebyerrors);
17 xlabel('numero di ascisse di interpolazione');
18 ylabel('massimo errore di interpolazione');
19 legend({'ascisse equidistanti', 'ascisse di chebyshev'}, 'Location', 'northeast');
```

6.5 Esercizio 18

Listing 5: es18.m

```
1 f = @(x)(cos((pi*(x.^2))/2));
2 x = linspace(-1, 1, 100001);
3 linerrors = zeros(1, 40);
```

```

4 chebyerrors = zeros(1, 40);
5 for n = 4:100
6     xlin = linspace(-1, 1, n+1);
7     xcheby = chebyshev(-1,1,n+1);
8     xcheby(1)=-1;
9     xcheby(n+1)=1;
10    ylin = splinenat(xlin,f(xlin),x);
11    ycheby = splinenat(xcheby,f(xcheby),x);
12    ylin=ylin';
13    ycheby=ycheby';
14    linerrors(n) = norm(abs(f(x) - ylin), inf);
15    chebyerrors(n) = norm( abs(f(x) - ycheby), inf);
16 end
17 semilogy(linerrors);
18 hold on;
19 semilogy(chebyerrors);
20 xlabel('numero di ascisse di interpolazione');
21 ylabel('massimo errore di interpolazione');
22 legend({'ascisse equidistanti', 'ascisse di chebyshev'}, 'Location', 'northeast');

```

6.6 Esercizio 19

Listing 6: es19.m

```

1 f = @(x)(cos((pi*(x.^2))/2));
2 x = linspace(-1, 1, 100001);
3 linerrors = zeros(1, 40);
4 chebyerrors = zeros(1, 40);
5 for n = 4:100
6     xlin = linspace(-1, 1, n+1);
7     xcheby = chebyshev(-1,1,n+1);
8     ylin = spline(xlin,f(xlin),x);
9     ycheby = spline(xcheby,f(xcheby),x);
10    linerrors(n) = norm(abs(f(x) - ylin), inf);
11    chebyerrors(n) = norm( abs(f(x) - ycheby), inf);
12 end
13 semilogy(linerrors);
14 hold on;
15 semilogy(chebyerrors);
16 xlabel('numero di ascisse di interpolazione');
17 ylabel('massimo errore di interpolazione');
18 legend({'ascisse equidistanti', 'ascisse di chebyshev'}, 'Location', 'northeast');

```

6.7 Esercizio 20

Listing 7: es20.m

```

1 f = @(x)(cos((pi*x.^2)/2));
2 fp = @(x)(f(x) + 10^(-3)*rand(size(x)));
3 xi = -1 + 2*(0:10^4)/10^4;
4 fi = f(xi);
5 fpi = fp(xi);
6 errors=zeros(1, 20);
7 for m = 1:20
8     y = minimiquadrati(xi, fpi, m);
9     errors(m) = norm(abs(y-fi), inf);
10 end

```

```

11 semilogy(errors);
12 xlabel('grado del polinomio');
13 ylabel('errore di interpolazione massimo');

```

6.8 Esercizio 21

Listing 8: es21.m

```

1 for i = 1:7
2     weights= rats(ncweights(i))
3 end

```

6.9 Esercizio 22

Listing 9: es22.m

```

1 rapp = zeros(1, 50);
2 for i = 1:50
3     rapp(i) = sum(abs(ncweights(i)))/i;
4 end
5 semilogy(rapp);
6 xlabel('grado n della formula di Newton-Cotes');
7 ylabel('^{\mathcal{K}_n}/_{\mathcal{K}}');

```

6.10 Esercizio 23

Listing 10: es23.m

```

1 value = log(cos(1)/cos(1.1));
2 x = zeros(1,9);
3 errors=zeros(1, 9);
4 for i = 1:9
5     x(i) = newtoncotes(@tan, -1,1.1, i);
6     errors(i) = abs(value-x(i));
7 end

```

6.11 Esercizio 24

Listing 11: es24.m

```

1 a = -1;
2 b = 1.1;
3 n = 10;
4 itrap = zeros(1, n);
5 isimp = zeros(1, n);
6 for i = 1:n
7     itrap(i) = trapecomp(@tan, a, b, i*2);
8     isimp(i) = simpcomp(@tan, a, b, i*2);
9 end
10 integrali = [itrap; isimp];
11 row_names = {'trapezi composta', 'simpson composta'};
12 colnames = {'2', '4', '6', '8', '10', '12', '14', '16', '18', '20'};
13 values = array2table(integrali, 'RowNames', row_names, 'VariableNames', colnames);
14 disp(values);

```

6.12 Esercizio 25

Listing 12: es25.m

```
1 format long e
2 f = @(x)(1/(1+100*x.^2));
3 a = -1;
4 b = 1;
5 itrap = zeros(1, 5);
6 trap_points = zeros(1, 5);
7 isimp = zeros(1, 5);
8 simp_points = zeros(1, 5);
9 for i = 1:5
10     [itrap(i), points] = adaptrap(f, a, b, 10^(-i-1));
11     trap_points(i) = length(points);
12     [isimp(i), points] = adapsim(f, a, b, 10^(-i-1));
13     simp_points(i) = length(points);
14 end
15 integrali = [itrap; isimp];
16 npoints = [trap_points; simp_points];
17 row_names = {'trapezi adattiva', 'simpson adattiva'};
18 colnames = {'10^-2', '10^-3', '10^-4', '10^-5', '10^-6'};
19 values = array2table(integrali, 'RowNames', row_names, 'VariableNames', colnames);
20 npoints = array2table(npoints, 'RowNames', row_names, 'VariableNames', colnames);
21 disp(values);
22 format
23 disp(npoints);
```