



Elaborato di  
**Calcolo Numerico**  
Anno Accademico 2021/2022

*John Smith* - 1234567 - *john.smith@stud.unifi.it*  
*Nice Guy* - 9876543 - *nice.guy@stud.unifi.it*

June 25, 2022

# Contents

<b>1</b>	<b>Capitolo 1</b>	<b>4</b>
1.1	Esercizio 1 . . . . .	4
1.2	Esercizio 2 . . . . .	4
1.3	Esercizio 3 . . . . .	4
<b>2</b>	<b>Capitolo 2</b>	<b>5</b>
2.1	Esercizio 4 . . . . .	5
2.2	Esercizio 5 . . . . .	6
2.3	Esercizio 6 . . . . .	8
2.4	Esercizio 7 . . . . .	9
<b>3</b>	<b>Capitolo 3</b>	<b>13</b>
3.1	Esercizio 8 . . . . .	13
3.2	Esercizio 9 . . . . .	14
3.3	Esercizio 10 . . . . .	15
3.4	Esercizio 11 . . . . .	16
3.5	Esercizio 12 . . . . .	17
3.6	Esercizio 13 . . . . .	18
3.7	Esercizio 14 . . . . .	18
<b>4</b>	<b>Capitolo 4</b>	<b>20</b>
4.1	Esercizio 15 . . . . .	20
4.2	Esercizio 16 . . . . .	21
4.3	Esercizio 17 . . . . .	21
4.4	Esercizio 18 . . . . .	22
4.5	Esercizio 19 . . . . .	23
4.6	Esercizio 20 . . . . .	23
<b>5</b>	<b>Capitolo 5</b>	<b>25</b>
5.1	Esercizio 21 . . . . .	25
5.2	Esercizio 22 . . . . .	25
5.3	Esercizio 23 . . . . .	27
5.4	Esercizio 24 . . . . .	27
5.5	Esercizio 25 . . . . .	28
<b>6</b>	<b>Codici ausiliari</b>	<b>32</b>
6.1	Esercizio 6 . . . . .	32
6.2	Esercizio 7 . . . . .	32
6.3	Esercizio 15 . . . . .	33
6.4	Esercizio 16 . . . . .	33
6.5	Esercizio 18 . . . . .	34
6.6	Esercizio 19 . . . . .	34
6.7	Esercizio 20 . . . . .	34
6.8	Esercizio 21 . . . . .	35
6.9	Esercizio 22 . . . . .	35
6.10	Esercizio 23 . . . . .	35
6.11	Esercizio 24 . . . . .	35
6.12	Esercizio 25 . . . . .	36

## List of Figures

1	iterazioni richieste . . . . .	9
2	iterazioni richieste . . . . .	12
3	risultati interpolazione . . . . .	20
4	risultati interpolazione hermite . . . . .	21

## List of Tables

1	valori approssimati con i metodi di Newton, secanti e Steffensen . . . . .	9
2	valori approssimati da newton, newton modificato e aitken(dati raccolti in es7.m) . . . . .	11
3	valori approssimati . . . . .	19
4	pesi della formula di Newton-Cotes fino al settimo grado . . . . .	26
5	risultati di es24.m . . . . .	28
6	risultati di es25.m . . . . .	31

# 1 Capitolo 1

## 1.1 Esercizio 1

Verificare che,

$$\frac{-f(x+2h) + 8f(x+h) - 8f(x-h) + f(x-2h) + O(h^4)}{12h} = f'(x) + O(h^4)$$

**Soluzione:**

Per verificare la equivalenza ci servirà applicare sviluppo di Taylor per tutti i funzioni. La funzione di Taylor definita così:

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

Calcoliamo le sviluppi di Taylor per tutti funzioni che sono presenti in formula da verificare. A variabile  $x_0$  noi assumiamo il valore  $x$ , quindi  $x_0 = x$ . Allora:

$$f(x+2h) = f(x) + f'(x)(x+2h-x) + O(x+2h-x) = f(x) + 2hf'(x) + O(h)$$

$$f(x+h) = f(x) + f'(x)(x+h-x) + O(x+h-x) = f(x) + hf'(x) + O(h)$$

$$f(x-h) = f(x) + f'(x)(x-h-x) + O(x-h-x) = f(x) - hf'(x) + O(h)$$

$$f(x-2h) = f(x) + f'(x)(x-2h-x) + O(x-2h-x) = f(x) - 2hf'(x) + O(h)$$

Se mettere tutto insieme otteniamo:

$$\begin{aligned} & \frac{-(f(x) + 2hf'(x)) + 8(f(x) + hf'(x)) - 8(f(x) - hf'(x)) + (f(x) - 2hf'(x)) + O(h^4)}{12h} = \\ & = \frac{-\cancel{f(x)} - 2hf'(x) + 8\cancel{f(x)} + 8hf'(x) - 8\cancel{f(x)} + 8hf'(x) + \cancel{f(x)} - 2hf'(x) + O(h^4)}{12h} = \\ & = \frac{12hf'(x) + O(h^4)}{12h} = f'(x) + O(h^4) \end{aligned}$$

## 1.2 Esercizio 2

Calcolare, motivandone i passaggi, la precisione di macchina della doppia precisione dello standard IEEE. Confrontare questa quantità con quanto ritornato dalla variabile `eps` di Matlab, commentando a riguardo.

**Soluzione:**

## 1.3 Esercizio 3

Eseguire il seguente script Matlab:

```
1 format long e
2 (1+(1e-14-1))*1e14
```

Spiegare i risultati ottenuti.

**Soluzione:**

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:

## 2 Capitolo 2

### 2.1 Esercizio 4

Scrivere una function Matlab, *radice(x)* che, avendo in ingresso un numero  $x$  non negativo, ne calcoli la radice quadrata utilizzando solo operazioni algebriche elementari, con la massima precisione possibile. Confrontare con la function *sqrt* di Matlab per 20 valori di  $s$ , equispaziati logaritmicamente nell'intervallo  $[1e-10, 1e10]$ , evidenziando che si è ottenuta la massima precisione possibile.

**Soluzione:**

```
1 function answer = radice(x)
2     % answer = radice(x)
3     % This method is called Babilons method
4     % It gets some value in input and apply the Babilons method
5     % until I needed precision, but we need maximum precision possible,
6     % so I apply this method until I get two equal values
7     % one before another and this will mean this method arrived at a point,
8     % in which value not changes, so it is a maximum precision.
9     % INPUT:  x      — Is a value of which we want to get square root
10    % OUTPUT: answer — This is the square root of the input
11    format long e;
12
13    if x < 0
14        error('Wrong value in input! Should be not less then zero!');
15    end
16
17    answer = x / 2;
18    guess = x;
19
20    while not(answer == guess)
21        guess = answer;
22        answer = (guess + (x / guess)) / 2;
23    end
24
25 end
26
27 for i = 1:0.5:10
28     s = 1e-10 + 1e10 * log10(i);
29     custom = radice(s);
30     native = sqrt(s);
31
32     if abs(custom - native) >= eps(abs(native))
33
34         if abs(custom - native) > eps(abs(native))
35             diff = abs(custom - native);
36             fprintf([ ...
37                 'result is not equal for %.' int2str(abs(int16(log10(eps(s)))) + 1)
38                 ...
39                 'd, with values custom=.' int2str(abs(int16(log10(eps(custom)))) +
40                 20) ...
41                 'd and native=.' int2str(abs(int16(log10(eps(native)))) + 20) ...
42                 'd, with diff=.' int2str(abs(int16(log10(eps(diff)))) + 20) 'd\n'
43                 ...
44                 ], s, custom, native, diff);
45         elseif abs(custom - native) == eps(abs(native))
46             fprintf([ ...
47                 'Difference between this numbers are equal to exponent ' ...
48                 'for this numbers, and equals to %.30d\n'...
49                 ], eps(abs(native)));
```

```

47         end
48
49     end
50
51     fprintf([ ...
52         'custom=%. ' int2str(abs(int16(log10(eps(custom)))) + 1) ...
53         'd\nnative=%. ' int2str(abs(int16(log10(eps(native)))) + 1) ...
54         'd\n\n'
55         ], custom, native);
56
57 end

```

## 2.2 Esercizio 5

Scrivere function Matlab distinte che implementino efficientemente i seguenti metodi per la ricerca degli zeri di una funzione  $f(x)$ :

- il metodo di Newton;
- il metodo delle secanti;
- il metodo di Steffensen:

$$x_{n+1} = x_n - \frac{f(x_n)^2}{f(x_n + f(x_n)) - f(x_n)}, \quad n=0,1,\dots \quad (1)$$

Per tutti i metodi, utilizzare come criterio di arresto

$$|x_{n+1} - x_n| \leq tol * (1 + |x_n|)$$

essendo  $tol$  una opportuna tolleranza specificata in ingresso. Curare particolarmente la robustezza del codice.

**Soluzione:**

- Metodo di Newton

```

1 function [x, i] = newton(f, f1, x0, tol, maxit)
2     % [x, i] = newton(f, f1, x0, tol, maxit)
3     % Newton's method, used to calculate a root of the equation f(x)=0
4     % Input:  f      — function that we can derive
5             f1     — derived function of the 'f'
6             x0     — initial approximation
7             tol    — tolerance
8             maxit  — maximum number of iterations (default = 100)
9     % Output: x      — the approximation of the root function
10            i       — number of iterations
11    % CHECK THIS: bisezione, corde, secanti, aitken, newtonmod
12
13    format long e;
14
15    if nargin < 4
16        error('Number of arguments must be at least 4!');
17    elseif nargin == 4
18        maxit = 100;
19    end
20
21    if tol < eps
22        error('Tolerance cannot be checked!');
23    end
24

```

```

25     x = x0;
26
27     for i = 1:maxit
28         fx = feval(f, x);
29         flx = feval(f1, x);
30         x = x - fx / flx;
31
32         if abs(x - x0) <= tolx * (1 + abs(x0))
33             break;
34         end
35
36         x0 = x;
37     end
38
39     if abs(x - x0) > tolx * (1 + abs(x0))
40         error('The method does not converge!');
41     end
42
43 end

```

- Metodo delle secanti

```

1 function [x, i] = secanti(f, x0, x1, tolx, maxit)
2     % [x, i] = secanti(f, x0, x1, tolx, maxit)
3     % Method of the secant, used to calculate a root of the equation f(x)=0
4     % Input:  f      — function that we can derive
5     %         x0     — initial approximation
6     %         x1     — second initial approximation
7     %         tolx   — tolerance
8     %         maxit  — maximum number of iterations (default = 100)
9     % Output: x      — the approximation of the root function
10    %         i       — number of iterations
11    % CHECK THIS: bisezione, corde, secanti, aitken, newtonmod
12
13    format long e;
14
15    if nargin < 4
16        error('Number of arguments must be at least 4!');
17    elseif nargin == 4
18        maxit = 100;
19    end
20
21    i = 0;
22    f0 = feval(f, x0);
23
24    for i = 1:maxit
25        f1 = feval(f, x1);
26        df1 = (f1 - f0) / (x1 - x0);
27        x = x1 - (f1 / df1);
28
29        if abs(x1 - x0) <= tolx * (1 + abs(x0))
30            break;
31        end
32
33        x0 = x1;
34        x1 = x;
35        f0 = f1;
36
37    end

```

```

38
39     if abs(x - x0) > tolx * (1 + abs(x0))
40         error('The method does not converge!');
41     end
42
43 end

```

- Metodo di Steffensen

```

1  function [x, i] = steffensen(f, x0, tolx, maxit)
2      % [x, i] = steffensen(f, x0, tolx, maxit)
3      % Steffensen's method, used to calculate a root of the equation f(x)=0
4      % Input:  f      — a fixed point iteration function
5              x0      — initial guess to the fixed point
6              tolx     — tolerance
7              maxit    — maximum number of iterations (default = 100)
8      % Output: x      — the approximation of the root function
9              i        — number of iterations
10     % CHECK THIS: bisezione, corde, secanti, aitken, newtonmod
11
12     format long e;
13
14     if nargin < 3
15         error('Number of arguments must be at least 3!');
16     elseif nargin == 3
17         maxit = 100;
18     end
19
20     for i = 1:maxit
21         % get ready to do a large, but finite, number of iterations.
22         % This is so that if the method fails to converge, we won't
23         % be stuck in an infinite loop.
24         fx = feval(f, x0); % calculate the next two guesses for the fixed point.
25         f_fx_x = feval(f, fx + x0);
26         x = x0 - (fx^2 / (f_fx_x - fx));
27         % use Aitken's delta squared method to
28         % find a better approximation to x0.
29         if abs(x - x0) <= tolx * (1 + abs(x0))
30             break; % if we are, stop the iterations, we have our answer.
31         end
32
33         x0 = x;
34     end
35
36     if abs(x - x0) > tolx * (1 + abs(x0))
37         error(['Failed to converge in ' maxit ' iterations!']);
38     end
39
40 end

```

## 2.3 Esercizio 6

Utilizzare le *function* del precedente esercizio per determinare una approssimazione della radice della funzione

$$f(x) = x - \cos\left(\frac{\pi}{2}x\right),$$

per  $tol = 10^{-3}, 10^{-6}, 10^{-9}, 10^{-12}$ , partendo da  $x_0 = 1$  (e  $x_1 = 0.99$  per il metodo delle secanti). Tabulare i risultati, in modo da confrontare le iterazioni richieste da ciascun metodo. Commentare il relativo costo



computazionale, in termini di valutazioni funzionali richieste.

**Soluzione:**

Eseguito lo script es6.msi ottengono i risultati contenuti nella tabella 1 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= $10^{-12}$
newton	5.94611646360541e-01	5.94611644056836e-01	5.94611644056836e-01	5.94611644056836e-01
secanti	5.94611646954077e-01	5.94611644056836e-01	5.94611644056836e-01	5.94611644056836e-01
steffensen	5.94611681141925e-01	5.94611644056837e-01	5.94611644056836e-01	5.94611644056836e-01

Table 1: valori approssimati con i metodi di Newton, secanti e Steffensen

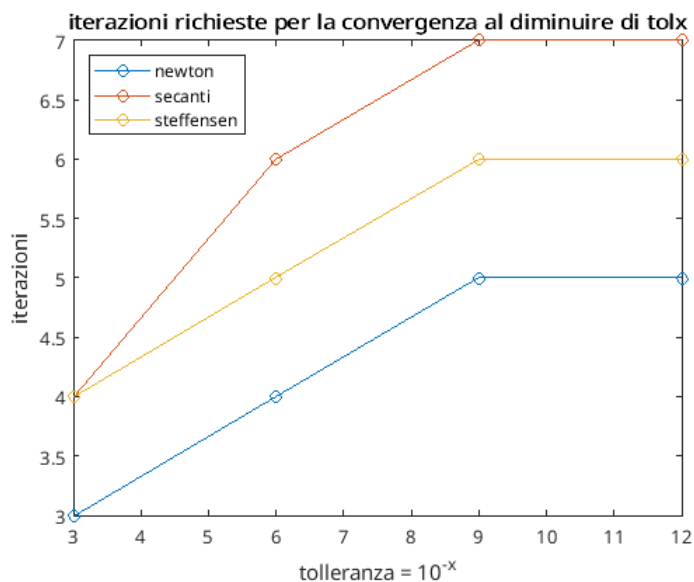


Figure 1: iterazioni richieste

## 2.4 Esercizio 7

Utilizzare le *function* del precedente esercizio per determinare una approssimazione della radice della funzione

$$f(x) = \left[ x - \cos\left(\frac{\pi}{2}x\right) \right]^3,$$

per  $tol = 10^{-3}, 10^{-6}, 10^{-9}, 10^{-12}$ , partendo da  $x_0 = 1$  (e  $x_1 = 0.99$  per il metodo delle secanti). Tabulare i risultati, in modo da confrontare le iterazioni richieste da ciascun metodo. Commentare i risultati ottenuti.

**Soluzione:**

Le nuove funzioni utilizzate in questo esercizio sono:

- Metodo di Newton modificato

```

1 function [x, i] = newtonmod( f, f1, x0, m, tol, maxit )
2 %NEWTONMOLT
3 %[x,i]=Newtonmolt(f,f1,x0,m,tol,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 % tol x  —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 tol x < 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)<=tol x *(1+abs(x0))
34             break;
35         else
36             x0 = x;
37         end
38     end
39
40 end

```

- Metodo delle accelerazioni di Aitken

```

1 function [x, i] = aitken( f, f1, x0, tol x, maxit )
2 %aitken
3 %[x,i]=aitken(f,f1, x0, tol x, maxit)
4 % Pre: f derivabile
5 % Applica il metodo di accelerazione di aitken per il calcolo della
6 % radice (di molteplicita' incognita) dell'equazione f(x)=0
7 % f      —funzione
8 % f1     —derivata di f
9 % x0     —approssimazione iniziale
10 % tol x  —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, corde, newtonmod
14 format long e
15 if nargin<4
16     error('numero argomenti insufficienti');
17 elseif nargin==4
18     maxit = 100;
19 end
20 if tol x < eps
21     error('tolleranza non idonea');

```

```

22 end
23 fx = feval(f,x0);
24 flx = feval(f1,x0);
25 x= x0-fx/flx;
26 for i = 1:maxit
27     x0 = x;
28     fx = feval( f, x0 );
29     flx = feval( f1, x0 );
30     x1 = x0 - fx/flx;
31     fx = feval( f, x1 );
32     flx = feval( f1, x1 );
33     x = x1 -fx/flx;
34     x = (x*x0-x1^2)/(x-2*x1+x0);
35     if abs(x-x0)<=tolx
36         break;
37     end
38 end
39 if abs(x-x0) > tolx*(1+abs(x0))
40     error('metodo non converge');
41 end
42 end

```

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

Tolleranza	Newton	Newton modificato	Aitken
$10^{-3}$	$1.99400296195610 \cdot 10^{-3}$	$1.32348898008484 \cdot 10^{-23}$	$3.72603946110722 \cdot 10^{-24}$
$10^{-6}$	$1.34922220938115 \cdot 10^{-6}$	$1.32348898008484 \cdot 10^{-23}$	$3.72603946110722 \cdot 10^{-24}$
$10^{-9}$	$1.36940553054800 \cdot 10^{-9}$	0	$2.93579661656743 \cdot 10^{-39}$
$10^{-12}$	$1.38989077859525 \cdot 10^{-12}$	0	$2.93579661656743 \cdot 10^{-39}$

Table 2: valori approssimati da newton, newton modificato e aitken(dati raccolti in es7.m)

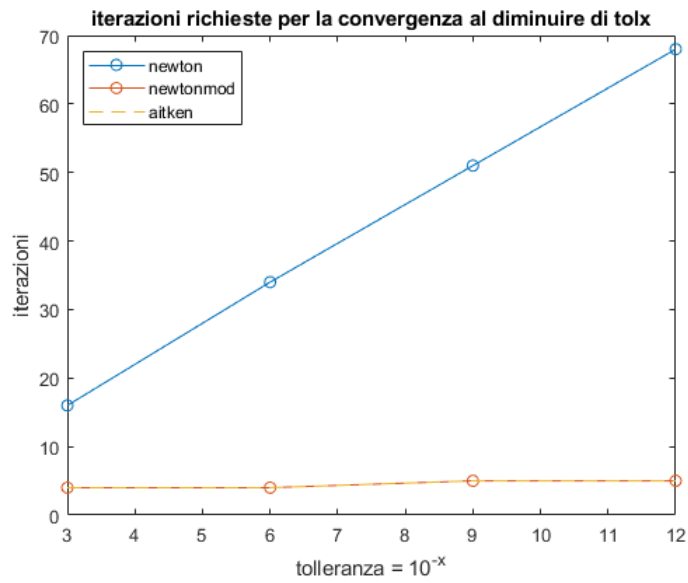


Figure 2: iterazioni richieste

Il metodo di newton classico perde la convergenza quadratica, essendo la radice cercata di molteplicità multipla. Il metodo di newton modificato e il metodo di aitken convergono molto più rapidamente e newton modificato riesce anche a trovare la radice esatta.

## 3 Capitolo 3

### 3.1 Esercizio 8

Scrivere una function Matlab,

```
function x = mialu(A, b)
```

che, data in ingresso una matrice  $A$  ed un vettore  $b$ , calcoli la soluzione del sistema lineare  $Ax = b$  con il metodo di fattorizzazione  $LU$  con pivoting parziale. Curare particolarmente la scrittura e l'efficienza della function, e validarla su due esempi non banali, generati casualmente, di cui sia nota la soluzione.

**Soluzione:**

```
function x = mialu(A, b)
% x = mialu(A, b)
% Method of the secant, used to calculate a root of the equation f(x)=0
% Input:  A    — a matrix representation of the equations with unknown variables
%         b    — result of unknown variables
% Output: x    — array to what equals unknown variables

% Calculate L and U matrixes
n = length(A);
L = zeros(n);
U = zeros(n);
% P = eye(n);

for k = 1:n
    % find the entry in the left column with the largest abs value (pivot)
    [~, r] = max(abs(A(k:end, k)));
    r = n - (n - k + 1) + r;

    A([k r], :) = A([r k], :);
    % P([k r], :) = P([r k], :);
    L([k r], :) = L([r k], :);
    b([k r], :) = b([r k], :);

    % from the pivot down divide by the pivot
    L(k:n, k) = A(k:n, k) / A(k, k);

    U(k, 1:n) = A(k, 1:n);
    A(k + 1:n, 1:n) = A(k + 1:n, 1:n) - L(k + 1:n, k) * A(k, 1:n);

end

U(:, end) = A(:, end);

x = zeros(n, 1);
y = zeros(n, 1);

% calcolo delle soluzioni di Ly = b
for i = 1:1:n
    alpha = 0;

    for k = 1:1:i
        alpha = alpha + L(i, k) * y(k);
    end

    y(i) = b(i) - alpha;
end
```

```

48 % calcolo delle soluzioni di  $Ux = y$ 
49 for i = n:-1:1
50     alpha = 0;
51
52     for k = i + 1:1:n
53         alpha = alpha + U(i, k) * x(k);
54     end
55
56     x(i) = (y(i) - alpha) / U(i, i);
57 end
58
59 end

```

### 3.2 Esercizio 9

Scrivere una function Matlab,

```

1 function x = mialdl(A, b)

```

che, dati in ingresso una matrice  $sdp$   $A$  ed un vettore  $b$ , calcoli la soluzione del corrispondente sistema lineare utilizzando la fattorizzazione  $LDL^T$ . Curare particolarmente la scrittura e l'efficienza della function, e validarla su due esempi non banali, generati casualmente, di cui sia nota la soluzione.

**Soluzione:**

```

1 function x = mialdl(A, b)
2     % x = mialdl(A, b)
3     % Method of the secant, used to calculate a root of the equation f(x)=0
4     % Input:  A      — a matrix representation of the equations with unknown variables
5     %          It is assumed that A is symmetric and postive definite.
6     %          b      — result of unknown variables
7     % Output: x      — array to what equals unknown variables
8
9     % Figure out the size of A.
10    n = size(A, 1);
11    % The main loop. See Golub and Van Loan for details.
12    L = zeros(n, n);
13
14    for j = 1:n,
15
16        if (j > 1),
17            v(1:j - 1) = L(j, 1:j - 1) .* d(1:j - 1);
18            v(j) = A(j, j) - L(j, 1:j - 1) * v(1:j - 1)';
19            d(j) = v(j);
20
21            if (j < n),
22                L(j + 1:n, j) = (A(j + 1:n, j) - L(j + 1:n, 1:j - 1) * v(1:j - 1)') / v(j);
23            end;
24
25            else
26                v(1) = A(1, 1);
27                d(1) = v(1);
28                L(2:n, 1) = A(2:n, 1) / v(1);
29            end;
30
31        end;
32
33    % Put d into a matrix.
34    D = diag(d);

```

```

35 % Put ones on the diagonal of L.
36 L = L + eye(n);
37
38 % n = size(A,1);
39 % L=zeros(n,n);
40 % for j=1:n,
41 %     if (j>1),
42 %         v(1:j-1)=L(j,1:j-1).*d(1:j-1);
43 %         v(j)=A(j,j)-L(j,1:j-1)*v(1:j-1)';
44 %         d(j)=v(j);
45 %         if(j<n),
46 %             L(j+1:n,j)=(A(j+1:n,j)-L(j+1:n,1:j-1)*v(1:j-1)')/v(j);
47 %         end;
48 %     else
49 %         v(1)=A(1,1);
50 %         d(1)=v(1);
51 %         L(2:n,1)=A(2:n,1)/v(1);
52 %     end;
53 % end;
54 % D=diag(d);
55 % L=L+eye(n);
56
57 U = D * L';
58 x = zeros(n, 1);
59 y = zeros(n, 1);
60
61 % calcolo delle soluzioni di Ly = b
62 for i = 1:1:n
63     alpha = 0;
64
65     for k = 1:1:i
66         alpha = alpha + L(i, k) * y(k);
67     end
68
69     y(i) = b(i) - alpha;
70 end
71
72 % calcolo delle soluzioni di Ux = y
73 for i = n:-1:1
74     alpha = 0;
75
76     for k = i + 1:1:n
77         alpha = alpha + U(i, k) * x(k);
78     end
79
80     x(i) = (y(i) - alpha) / U(i, i);
81 end
82
83 end

```

### 3.3 Esercizio 10

Data la function Matlab

```

1 function [A, b] = linsis(n, k, simme)
2 %
3 % [A,b] = linsis(n,k,simme) Crea una matrice A nxn ed un termine noto b,
4 % in modo che la soluzione del sistema lineare

```

```

5 % A*x=b sia x = [1,2,...,n]'.
6 % k `e un parametro ausiliario.
7 % simme, se specificato, crea una matrice
8 % simmetrica e definita positiva.
9 %
10 sigma = 10^(-2 * (1 - k)) / n;
11 rng(0);
12 [q1, r1] = qr(rand(n));
13
14 if nargin == 3
15     q2 = q1';
16 else
17     [q2, r1] = qr(rand(n));
18 end
19
20 A = q1 * diag([sigma 2 / n:1 / n:1]) * q2;
21 x = [1:n]';
22 b = A * x;
23 return

```

che crea sistemi lineari casuali con soluzione nota, risolvere, utilizzando la *function miaqr*, i sistemi lineari generati da  $[A, b] = \text{linsis}(10, 1)$  e  $[A, b] = \text{linsis}(10, 10)$ . Commentare l'accuratezza dei risultati ottenuti, dandone spiegazione esaustiva.

**Soluzione:**

### 3.4 Esercizio 11

Scrivere una function Matlab,

```

1 function [x,nr] = miaqr(A, b)

```

che, data in ingresso la matrice  $A \ m \times n$ , con  $m \geq n = \text{rank}(A)$ , ed un vettore  $b$  di lunghezza  $m$ , calcoli la soluzione del sistema lineare  $Ax = b$  nel senso dei minimi quadrati e, inoltre, la norma,  $nr$ , del corrispondente vettore residuo. Curare particolarmente la scrittura e l'efficienza della *function*. Validare la *function miaqr* su due esempi non banali, generati casualmente, confrontando la soluzione ottenuta con quella calcolata con l'operatore Matlab.

**Soluzione:**

```

1 function [x, nr] = miaqr(A, b)
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
13 if n > m
14     error('Dimensioni errate');
15 end
16
17 QR = A;
18
19 for i = 1:n
20     alfa = norm(QR(i:m, i));
21
22     if alfa == 0

```



```

23         error('la matrice non ha rango massimo');
24     end
25
26     if QR(i, i) >= 0
27         alfa = -alfa;
28     end
29
30     v1 = QR(i, i) - alfa;
31     QR(i, i) = alfa;
32     QR(i + 1:m, i) = QR(i + 1:m, i) / v1;
33     beta = -v1 / alfa;
34     v = [1; QR(i + 1:m, i)];
35     QR(i:m, i + 1:n) = QR(i:m, i + 1:n) - (beta * v) * (v' * QR(i:m, i + 1:n));
36 end
37
38 [m, n] = size(QR);
39 k = length(b);
40
41 if k ~= m
42     error('Dati inconsistenti');
43 end
44
45 x = b(:);
46
47 for i = 1:n
48     v = [1; QR(i + 1:m, i)];
49     beta = 2 / (v' * v);
50     x(i:m) = x(i:m) - beta * (v' * x(i:m)) * v;
51 end
52
53 x = x(1:n);
54
55 for j = n:-1:1
56
57     if QR(j, j) == 0
58         error('Matrice singolare');
59     end
60
61     x(j) = x(j) / QR(j, j);
62     x(1:j - 1) = x(1:j - 1) - QR(1:j - 1, j) * x(j);
63 end
64
65 end
66
67 %{
68
69 A = round (10 * rand (3))
70 b = round (10 * rand (3, 1))
71
72 %}

```

### 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
14 [m,n] = size(QR);
15 k = length(b);
16 if k ~= m
17     error('Dati inconsistenti');
18 end
19 x=b(:);
20 for i = 1:n
21     v=[1; QR(i+1:m,i)];
22     beta = 2/(v'* v);
23     x(i:m) = x(i:m) - beta*(v'*x(i:m))*v;
24 end
25 x=x(1:n);
26 for j = n:-1:1
27     if QR(j,j)==0
28         error('Matrice singolare');
29     end
30     x(j) = x(j) / QR(j,j);
31     x(1:j-1) = x(1:j-1) - QR(1:j-1,j)*x(j);
32 end
33 return
34 end

```

### 3.6 Esercizio 13

```

1 %Codice esercizio 13
2
3 A= [1, 2, 3; 1 2 4; 3 4 5; 3 4 6; 5 6 7];
4 b=[14 17 26 29 38];
5 QR=myqr(A);
6 ris=qrsolve(QR,b);
7 disp(ris);

```

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

### 3.7 Esercizio 14

L'espressione  $A \backslash b$  risolve in matlab, il sistema di equazioni lineari nella forma matriciale  $A \cdot x = b$  per  $x$ . L'espressione  $(A^T \cdot A) \backslash (A^T \cdot b)$ , è matematicamente la stessa operazione dell'espressione precedente, solamente che si moltiplica le due componenti per la trasposta di  $A$ . 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:  $1.5428e+09$ . Nella prima espressione, questo malcondizionamento non influisce sul risultato. Invece nella seconda espressione eseguendo la prima parentesi tonda, il condizionamento è pari a:  $4.4897e+18$ . Questo fa sì che eseguendo la divisione tra una matrice mal condizionata e il vettore, il risultato presenta degli errori.

$A \backslash b$	$(A'^*A) \backslash (A'^*b)$
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

## 4 Capitolo 4

### 4.1 Esercizio 15

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

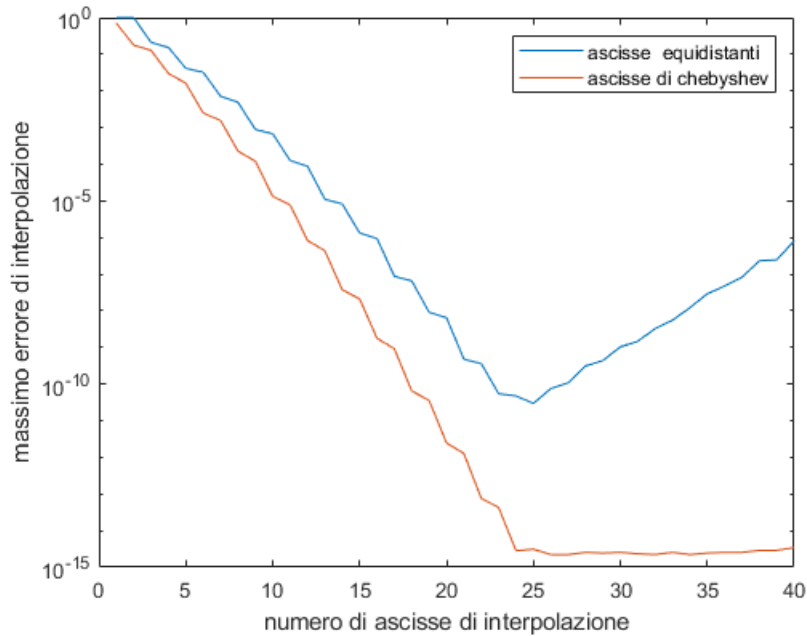


Figure 3: risultati interpolazione

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:

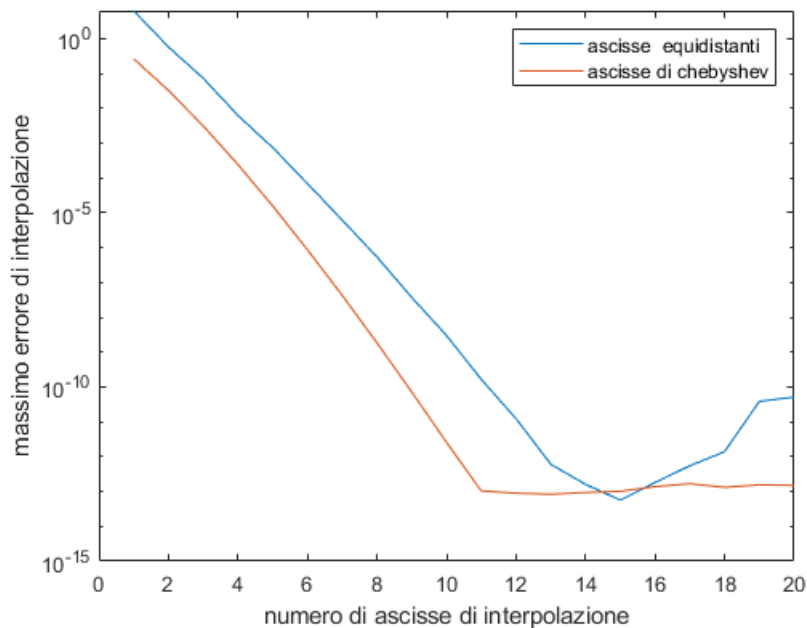


Figure 4: risultati interpolazione hermite

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

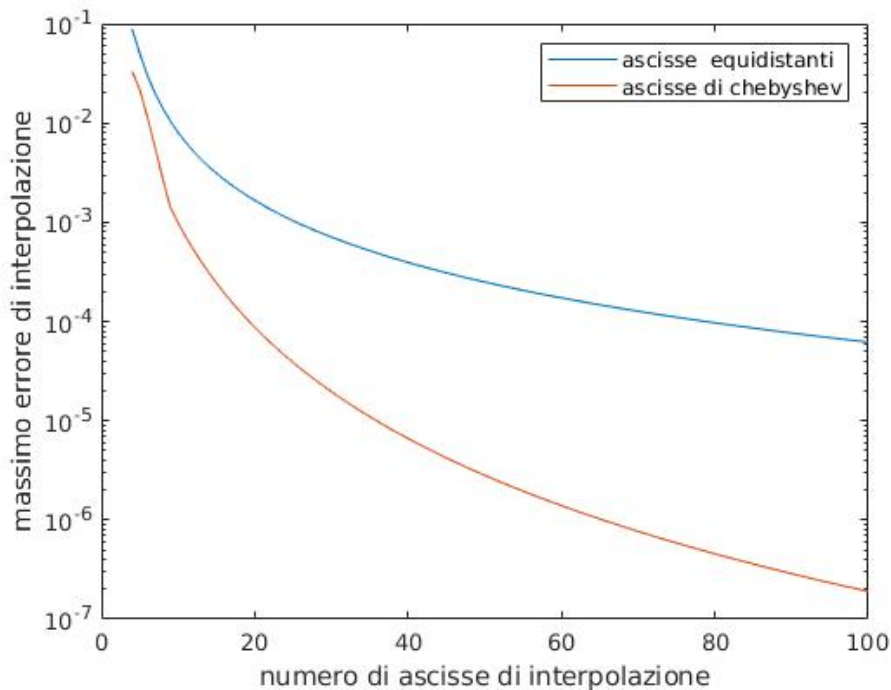
```
1 function output=splinenat(xi,fi,xq)
2 %
3 % output=splinenat(x,y,xq)
4 %funzione che calcola la spline cubica naturale.
5 %Input:
6 % xi=vettore delle ascisse su cui calcolare la spline
7 % fi=vettore dei valori di f(x), con x ascissa
8 % xq= insieme delle ascisse di cui si vuole sapere il valore della spline
9 %Output:
10 % output=vettore delle approssimazioni sulle ascisse xq
11 %
12
13 m = length(xi);
14 l=length(xq);
15
16 if m~=length(fi)
17     error(dati errati);
18 end
19 for i = 1:m-1
20     if any( find(xi(i+1:m)==xi(i)) ),
21         error(ascisse non distinte),
22     end
23 end
24 xi = xi(:);
```

```

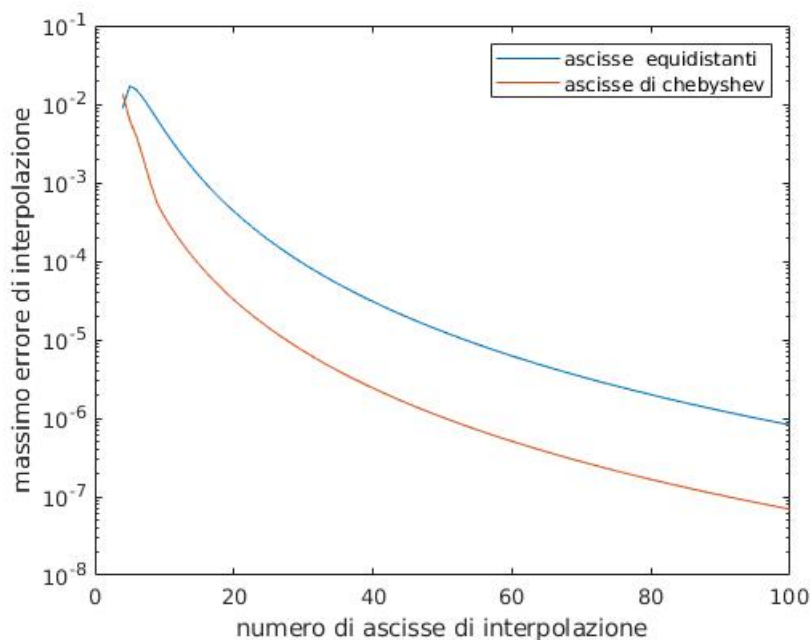
25 fi = fi(:);
26 [xi,ind] = sort(xi);
27 fi = fi(ind);
28 % ordino le ascisse in modo crescente
29 hi = diff(xi);
30 n= m-1;
31 df = diff(fi)./hi;
32 hh = hi(1:n-1)+hi(2:n);
33 rhs = 6*diff(df)./hh;
34 phi = hi(1:n-1)./hh;
35 csi = hi(2:n)./hh;
36 % = 1-phi;
37 d= 2*ones(n-1,1);
38 phi = phi(2:n-1);
39 csi = csi(1:n-2);
40 mi = trisolve( phi, d, csi, rhs );
41 mi = [0;mi;0];
42 r=fi(1:n)-((hi(1:n).^2)/6).*mi(1:n);
43 q=df(1:n)-((hi(1:n)/6).*(mi(2:m)-mi(1:n)));
44 output=zeros(l,1);
45 for i=1:l
46     indp=find(xq(i)>=xi(1:n),1,'last');
47     indg=indp+1;
48     output(i)=((((xq(i)-xi(indp)).^3).*mi(indg))+((xi(indg)-xq(i)).^3).*mi(indp))/(6*hi(
        indp))+q(indp).*(xq(i)-xi(indp))+r(indp);
49 end
50 return
51 end

```

#### 4.4 Esercizio 18



## 4.5 Esercizio 19

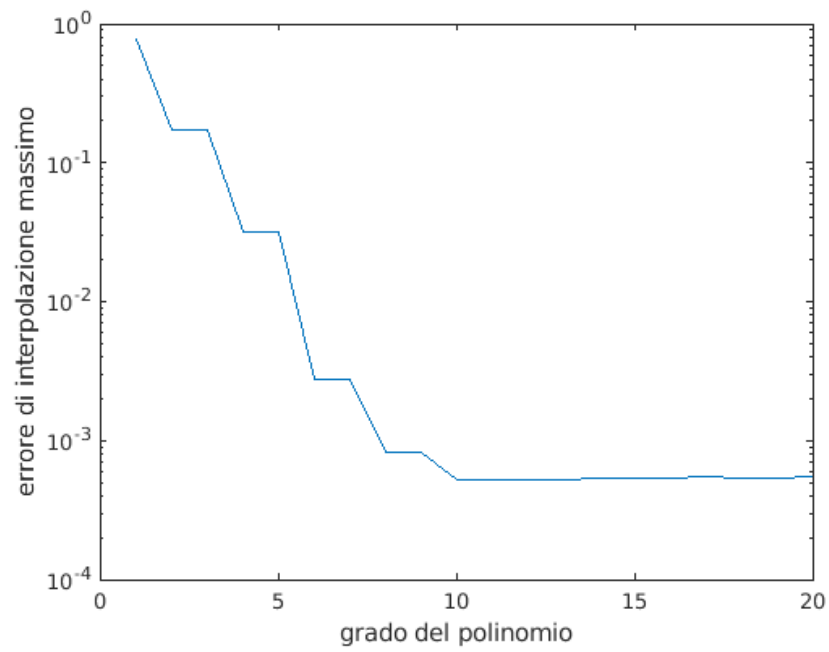


Si nota dai due grafici che la funzione spline riesce a mantenere su entrambe le tipologie di scelta delle ascisse, un basso errore di interpolazione, mentre la funzione `splinenat`, arriva ad avere nelle ascisse di chebyshev, un andamento simile a quello corrispettivo alla funzione spline, essendo comunque meno preciso. L'uso invece delle ascisse equidistanti nella funzione `splinenat`, fa sì che si abbia un andamento sostanzialmente diverso, rispetto a quello delle corrispettive nella funzione spline, e anche rispetto alle ascisse di chebyshev calcolate dalla stessa funzione.

## 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 %     n-      grado
7 %     c-      pesi calcolati
8 %
9 if n<=0
10     error('grado della formula non positivo');
11 end
12 nvalues = floor(n/2 + 1);
13 c=zeros(1,nvalues);
14 for j = 1:nvalues
15     temp = (0:n);
16     temp(j)=[];
17     f = @(x)(prod(x-temp) /prod(j-1-temp));
18     c(j) = integral(f, 0, n, 'ArrayValued', true);
19 end
20 c = [c flip(c)]; %sfrutto la simmetria dei pesi
21 if mod(n,2)==0
22     %elimino la copia del valore centrale prodotta da flip(c) e che risulta di troppo per
23     %n pari
24     c(n/2+1) = [];
25 end
26 return
27 end
```

Eseguendo lo script es21.m si ottiene:

### 5.2 Esercizio 22

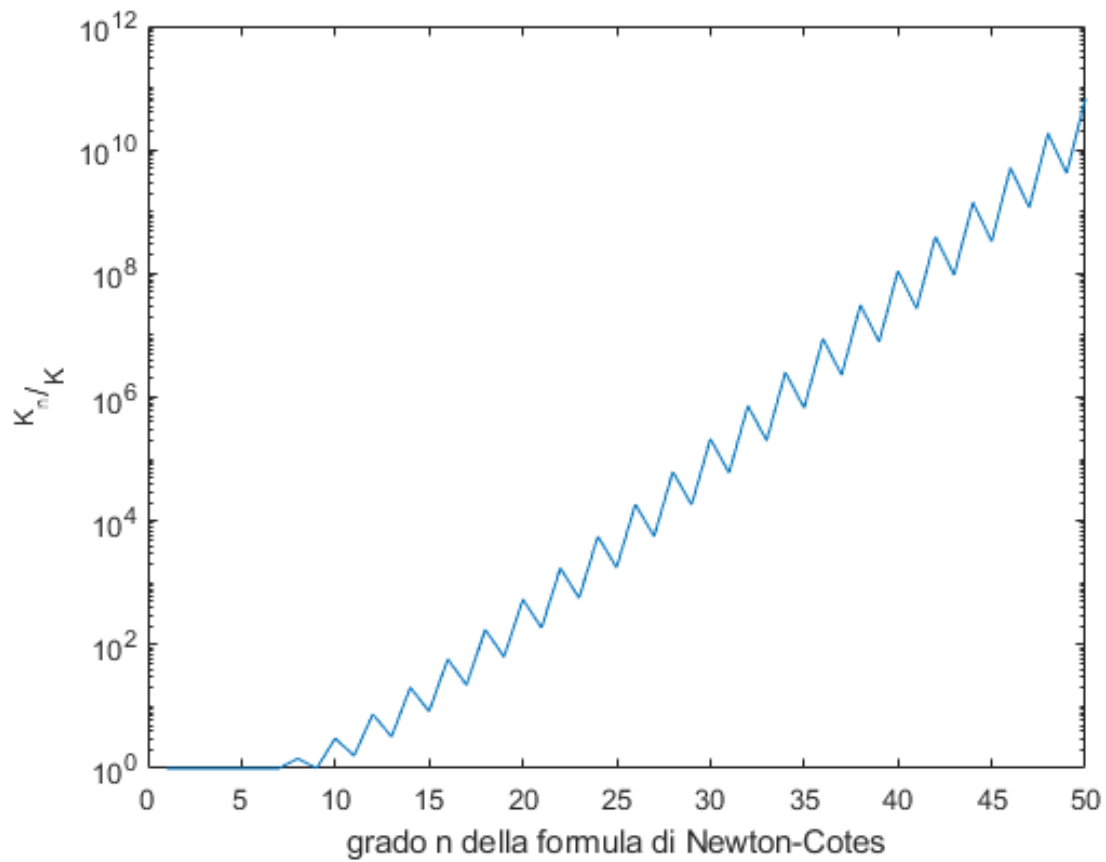
Sappiamo che  $k = (b - a)$  e  $k_n = (b - a) \frac{1}{n} \sum_{i=0}^n |c_{in}|$ . Il rapporto sarà dunque dato da:

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

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

$n \setminus c_{in}$	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	$\frac{95}{288}$	$\frac{125}{96}$	$\frac{125}{144}$	$\frac{125}{144}$	$\frac{125}{96}$	$\frac{95}{288}$		
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



### 5.3 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 %     f—      funzione
8 %     a,b—    intervallo dei punti
9 %     n—      grado della formula di newton cotes
10 %     y—      approssimazione integrale definito di f(x)
11
12 if a > b || n < 0
13     error('dati inconsistenti');
14 end
15 xi = linspace(a, b, n+1);
16 fi = feval(f, xi);
17 h = (b-a) / n;
18 c = ncweights(n);
19 y = h*sum(fi.*c);
20 return
21 end
```

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

grado della formula	valore integrale	errore
1	$4,28 \cdot 10^{-1}$	$2,53 \cdot 10^{-1}$
2	$2,13 \cdot 10^{-1}$	$3,8 \cdot 10^{-2}$
3	$1,96 \cdot 10^{-1}$	$2,1 \cdot 10^{-2}$
4	$1,80 \cdot 10^{-1}$	$5 \cdot 10^{-3}$
5	$1,79 \cdot 10^{-1}$	$4 \cdot 10^{-3}$
6	$1,76 \cdot 10^{-1}$	$1 \cdot 10^{-3}$
7	$1,76 \cdot 10^{-1}$	$1 \cdot 10^{-3}$
8	$1,75 \cdot 10^{-1}$	0
9	$1,75 \cdot 10^{-1}$	0

### 5.4 Esercizio 24

```
1 function I = trapecomp(f, a, b, n)
2 %
3 %
4 % I = trapecomp(f, a, b)
5 %
6 % Approssimazione dell'integrale definito di f(x) con estremi a e b,
7 % mediante la formula composta dei trapezi su n+1 ascisse equidistanti
8 %     f—      funzione
9 %     a,b—    estremi dell'intervallo
10 %     I—      approssimazione integrale definito di f(x)
11 %
12 if a==b
13     I=0;
14 elseif n < 1 || n~=fix(n)
15     error('numero di ascisse non valido');
16 else
17     h=(b-a)/n;
```

```

18     x=linspace(a, b, n+1);
19     f = feval(f, x);
20     I = h*(f(1)/2 + sum(f(2:n)) + f(n+1)/2);
21 end
22 return
23 end

```

```

1 function I = simpcomp(f, a, b, n)
2     %myFun — Description
3     %
4     % I = simpcomp(f, a, b)
5     %
6     % Approssimazione dell'integrale definito di f(x) con estremi a e b,
7     % mediante la formula composta di Simpson su n+1 ascisse equidistanti(n pari)
8     % f—      funzione
9     % a,b—     estremi dell'intervallo
10    % I—       approssimazione integrale definito di f(x)
11    %
12    %
13
14    if a==b
15        I=0;
16    elseif n < 2 || n/2 ~= fix(n/2)
17        error('numero di ascisse non valido');
18    else
19        h=(b-a)/n;
20        x=linspace(a, b, n+1);
21        f = feval(f, x);
22        I = (h/3) * (f(1) + f(n+1) + 4*sum(f(2:2:n)) + 2*sum(f(3:2:n-1)));
23    end
24    return
25    end

```

Approssimando  $\int_{-1}^{1.1} \tan(x)dx$  con le due formule si ottiene:

intervalli\formula	trapezi composta	simpson composta
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

Table 5: risultati di es24.m

La formula composta di simpson converge più rapidamente ed è più precisa rispetto alla formula dei trapezi

## 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 % Approssimazione dell'integrale definito di f(x) con estremi a e b,
7 % mediante la formula adattiva dei trapezi
8 %     f—      funzione
9 %     a,b—    estremi intervallo
10 %     tol—    tolleranza
11 %     fa,fb—  valore della funzione valutata negli punti a,b
12 %
13 %     points— punti valutati
14 %     I2—     valore dell'integrale di f(x)
15 %
16 global points
17 delta = 0.5;
18 if nargin<=4
19     fa = feval(f, a );
20     fb = feval(f, b );
21     if nargin==2
22         points = [a fa; b fb];
23     else
24         points = [];
25     end
26 end
27 h = b-a;
28 x1 = (a+b)/2;
29 f1 = feval(f, x1);
30 if ~isempty(points)
31     points = [points; [x1 f1]];
32 end
33 I1 = .5*h*(fa+fb);
34 I2 = .5*(I1+h*f1);
35 e = abs(I2-I1)/3;
36 if e>tol || abs(b-a) > delta
37     I2 = adaptrap(f, a, x1, tol/2, fa, f1) + adaptrap(f, x1, b, tol/2, f1, fb);
38 end
39 return
40 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 disimpson
5 %     f—      funzione
6 %     a,b—    estremi intervallo
7 %     tol—    tolleranza
8 %     fa,fb—  valore della funzione valutata negli punti a,b
9 %     f1—     valore della funzione valutata nel punto x1, intermedio ad a,b
10 %
11 %     points— punti valutati
12 %     I2—     valore dell'integrale di f(x)
13 %
14 global points
15 delta = 0.5;
16 x1 = (a+b)/2;
17 if nargin<=4
18     fa = feval(f, a );
19     fb = feval(f, b );
20     f1 = feval(f, x1);

```

```

21     if nargin==2
22         points = [a fa;x1 f1; b fb];
23     else
24         points = [];
25     end
26 end
27 h = (b-a)/6;
28 x2 = (a+x1)/2;
29 x3 = (x1+b)/2;
30 f2 = feval(f, x2);
31 f3 = feval(f, x3);
32 if ~isempty(points)
33     points = [points; [x2 f2; x3 f3]];
34 end
35 I1 = h*(fa+fb+4*f1);
36 I2 = .5*h*(fa+4*f2+2*f1+4*f3+fb);
37 e = abs(I2-I1)/15;
38 if e>tol || abs(b-a) > delta
39     I2 = adapsim(f, a, x1, tol/2, fa, f2, f1) + adapsim(f, x1, b, tol/2, f1, f3, fb);
40 end
41 return
42 end

```

Approssimando  $\int_{-1}^1 \frac{1}{1+10^2 x^2} dx$  con le due formule si ottiene:

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 6: risultati di es25.m

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 6, 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 syms x;
2 f = @(x)(x - cos((pi / 2) * x));
3 f1 = matlabFunction(diff(f, x));
4
5 x0 = 1;
6 x1 = 9.9e-1;
7 x = zeros(3, 4);
8 y = zeros(3, 4);
9
10 for i = 3:3:12
11
12     [x(1, i / 3), y(1, i / 3)] = es5_newton(f, f1, x0, 10^(-i));
13     [x(2, i / 3), y(2, i / 3)] = es5_secanti(f, x0, x1, 10^(-i));
14     [x(3, i / 3), y(3, i / 3)] = es5_steffensen(f, x0, 10^(-i));
15 end
16
17 row_names = {'newton', 'secanti', 'steffensen'};
18 colnames = {'10^-3', '10^-6', '10^-9', '10^-12'};
19 values = array2table(x, 'RowNames', row_names, 'VariableNames', colnames);
20 disp(values)
21 figure
22 plot([3, 6, 9, 12], y, 'o-')
23 title('iterazioni richieste per la convergenza al diminuire di tol x')
24 xlabel('tolleranza = 10^{-x}')
25 ylabel('iterazioni')
26 legend({'newton', 'secanti', 'steffensen'}, '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(1,1:end)', '-o');
21 hold on;
```



```

22 plot([3, 6, 9, 12], y(2,1:end)', '-o');
23 plot([3, 6, 9, 12], y(3,1:end)', '—')
24 title('iterazioni richieste per la convergenza al diminuire di tol x')
25 xlabel('tolleranza = 10^{-x}')
26 ylabel('iterazioni')
27 legend({'newton', 'newtonmod', 'aitken'}, 'Location', 'northwest')

```

### 6.3 Esercizio 15

Listing 3: es15.m

```

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

```

### 6.4 Esercizio 16

Listing 4: es16.m

```

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

```

## 6.5 Esercizio 18

Listing 5: es18.m

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

## 6.6 Esercizio 19

Listing 6: es19.m

```
1 %Codice esercizio 19
2 f = @(x)(cos((pi*(x.^2))/2));
3 x = linspace(-1, 1, 100001);
4 linerrors = zeros(1, 40);
5 chebyerrors = zeros(1, 40);
6 for n = 4:100
7     xlin = linspace(-1, 1, n+1);
8     xcheby = chebyshev(-1,1,n+1);
9     ylin = spline(xlin,f(xlin),x);
10    ycheby = spline(xcheby,f(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');
```

## 6.7 Esercizio 20

Listing 7: es20.m

```
1 %Codice esercizio 20
2
```

```

3 f = @(x)(cos((pi*x.^2)/2));
4 fp = @(x)(f(x) + 10^(-3)*rand(size(x)));
5 xi = -1 + 2*(0:10^4)/10^4;
6 fi = f(xi);
7 fpi = fp(xi);
8 errors=zeros(1, 20);
9 for m = 1:20
10     y = minimiquadrati(xi, fpi, m);
11     errors(m) = norm(abs(y-fi), inf);
12 end
13 semilogy(errors);
14 xlabel('grado del polinomio');
15 ylabel('errore di interpolazione massimo');

```

## 6.8 Esercizio 21

Listing 8: es21.m

```

1 %Codice esercizio 21
2
3 for i = 1:7
4     weights= rats(ncweights(i))
5 end

```

## 6.9 Esercizio 22

Listing 9: es22.m

```

1 %Codice esercizio 22
2
3 rapp = zeros(1, 50);
4 for i = 1:50
5     rapp(i) = sum(abs(ncweights(i)))/i;
6 end
7 semilogy(rapp);
8 xlabel('grado n della formula di Newton-Cotes');
9 ylabel('^{K_n}/{_K}');

```

## 6.10 Esercizio 23

Listing 10: es23.m

```

1 %Codice esercizio 23
2
3 value = log(cos(1)/cos(1.1));
4 x = zeros(1,9);
5 errors=zeros(1, 9);
6 for i = 1:9
7     x(i) = newtoncotes(@tan, -1,1.1, i);
8     errors(i) = abs(value-x(i));
9 end

```

## 6.11 Esercizio 24

Listing 11: es24.m

```

1 %Codice esercizio 24
2
3 a = -1;
4 b = 1.1;
5 n = 10;
6 itrap = zeros(1, n);
7 isimp = zeros(1, n);
8 for i = 1:n
9     itrap(i) = trapecomp(@tan, a, b, i*2);
10    isimp(i) = simpcomp(@tan, a, b, i*2);
11 end
12 integrali = [itrap; isimp];
13 row_names = {'trapezi composta', 'simpson composta'};
14 colnames = {'2', '4', '6', '8', '10', '12', '14', '16', '18', '20'};
15 values = array2table(integrali, 'RowNames', row_names, 'VariableNames', colnames);
16 disp(values);

```

## 6.12 Esercizio 25

Listing 12: es25.m

```

1 %Codice esercizio 25
2
3 format long e
4 f = @(x)(1/(1+100*x.^2));
5 a = -1;
6 b = 1;
7 itrap = zeros(1, 5);
8 trap_points = zeros(1, 5);
9 isimp = zeros(1, 5);
10 simp_points = zeros(1, 5);
11 for i = 1:5
12     [itrap(i), points] = adaptrap(f, a, b, 10^(-i-1));
13     trap_points(i) = length(points);
14     [isimp(i), points] = adapsim(f, a, b, 10^(-i-1));
15     simp_points(i) = length(points);
16 end
17 integrali = [itrap; isimp];
18 npoints = [trap_points; simp_points];
19 row_names = {'trapezi adattiva', 'simpson adattiva'};
20 colnames = {'10^-2', '10^-3', '10^-4', '10^-5', '10^-6'};
21 values = array2table(integrali, 'RowNames', row_names, 'VariableNames', colnames);
22 npoints = array2table(npoints, 'RowNames', row_names, 'VariableNames', colnames);
23 disp(values);
24 format
25 disp(npoints);

```