



Elaborato di  
**Calcolo Numerico**  
Anno Accademico 2019/2020

Niccolò *Piazzesi* - 6335623 - [niccolo.piazzesi@stud.unifi.it](mailto:niccolo.piazzesi@stud.unifi.it)  
Pietro *Bernabei* - 6291312 - [pietro.bernabei@stud.unifi.it](mailto:pietro.bernabei@stud.unifi.it)

# 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 . . . . .	5
2.3	Esercizio 6 . . . . .	8
2.4	Esercizio 7 . . . . .	9
<b>3</b>	<b>Capitolo 3</b>	<b>11</b>
3.1	Esercizio 8 . . . . .	11
3.2	Esercizio 9 . . . . .	11
3.3	Esercizio 10 . . . . .	12
3.4	Esercizio 11 . . . . .	12
3.5	Esercizio 12 . . . . .	13
3.6	Esercizio 13 . . . . .	13
3.7	Esercizio 14 . . . . .	13
<b>4</b>	<b>Capitolo 4</b>	<b>14</b>
<b>5</b>	<b>Capitolo 5</b>	<b>15</b>
5.1	Esercizio 21 . . . . .	15
5.2	Esercizio 22 . . . . .	15
5.3	Esercizio 23 . . . . .	16

## List of Figures

1	iterazioni richieste . . . . .	8
---	--------------------------------	---

## List of Tables

1	valori approssimati . . . . .	8
2	valori approssimati . . . . .	12

# 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  $u = 1.1102e - 16 = \frac{\epsilon}{2}$ , dove  $\epsilon$  è la precisione di macchina.  $\epsilon$  è il più piccolo valore di macchina per il quale  $a + \epsilon \neq a$  per un qualsiasi numero  $a$ . Quando  $u$  assume valore  $\frac{\epsilon}{2}$  il controllo interno  $1 + u == 1$ , che corrisponde alla condizione di uscita, risulta vero, perchè  $u$  è minore di  $\epsilon$ .

## 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  $a = 1e20$ . Sottrarre un numero da se stesso ha sempre risultato esatto 0.
- nel secondo caso la sottrazione avviene tra i termini  $a + b = 1e20 + 100$  e  $a = 100$ . Poichè  $1e20$  è molto più grande di 100,  $a+b$  è "quasi uguale" ad  $a$ . La sottrazione amplifica gli errori di approssimazione causati dalla rappresentazione in aritmetica finita dei numeri coinvolti. A causa di questi errori il calcolatore approssima la differenza con 0.

## 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, aitken, 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.m si 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

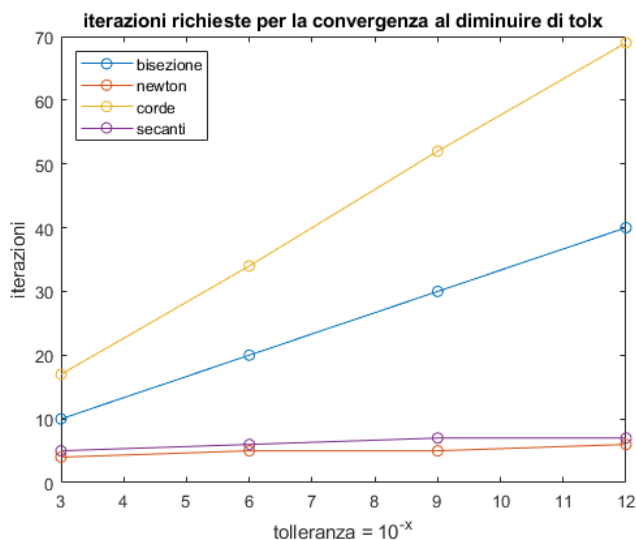


Figure 1: iterazioni richieste



## 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 function [x, i] = aitken( f, f1, x0, tolX, maxit )
2 %aitken
3 %[x,i]=aitken(f,f1, x0, tolX, 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 % 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, 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<eps
21         error('tolleranza non idonea');
22     end
23     x = x0;
24     for i = 1:maxit
25         x0 = x;
26         fx = feval( f, x0 );
27         flx = feval( f1, x0 );
28         x1 = x0 - fx/flx;
29         fx = feval( f, x1 );
30         flx = feval( f1, x1 );
31         x = x1 - fx/flx;
32         x = (x*x0-x1^2)/(x-2*x1+x0);
33         if abs(x-x0)<=tol*(1+abs(x0))
34             break;
35         end
36     end
37     if abs(x-x0) > tol*(1+abs(x0))
38         disp('metodo non converge');
39     end
40 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  $\tan(0) = 0$ .

## 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

Si nota da questa tabella, che sigma e la norma euclidea, tra la differenza di x e xref, sono direttamente proporzionali

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

Table 2: valori approssimati

### 3.4 Esercizio 11

```

1 function QR = myqr(A)
2 %QR = myqr(A)
3 % calcola la fattorizzazione QR di Householder della matrice A
4
5 [m,n] = size(A);
6 if n > m
7     error('Dimensioni errate');
8 end
9 QR = A;
10 for i = 1:n
11     alfa = norm(QR(i:m,i));
12     if alfa == 0
13         error('la matrice non ha rango massimo');
14     end
15     if QR(i,i) >= 0
16         alfa = -alfa;
17     end
18     v1 = QR(i,i) - alfa;
19     QR(i,i) = alfa;

```

```

20     QR(i+1:m,i) = QR(i+1:m,i)/v1;
21     beta = -v1/alfa;
22     v = [1; QR(i+1:m,i)];
23     QR(i:m,i+1:n) = QR(i:m,i+1:n) - (beta * v) * (v' * QR(i:m,i+1:n));
24 end
25 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 %
7 [m,n] = size(QR);
8 k = length(b);
9 if k ~= m
10     error('Dati inconsistenti');
11 end
12 x=b(:);
13 for i = 1:n
14     v=[1; QR(i+1:m,i)];
15     beta = 2/(v'* v);
16     x(i:m) = x(i:m) - (beta*(v'*x(i:m))*v);
17 end
18 x=x(1:n);
19 for j = n:-1:1
20     if QR(j,j)==0
21         error('Matrice singolare');
22     end
23     x(j) = x(j) / QR(j,j);
24     x(1:j-1) = x(1:j-1) - QR(1:j-1,j)*x(j);
25 end
26 return
27 end

```

### 3.6 Esercizio 13

Comandi: A= [1, 2, 3; 1 2 4; 3 4 5; 3 4 6; 5 6 7];  
b=[14 17 26 29 38];  
QR=myqr(A);  
ris=qrsolve(QR,b);  
ris =  
1.0000 2.0000 3.0000

### 3.7 Esercizio 14

l'espressione Arisolve in matlab, il sistema di equazioni lineari nella forma matriciale  $A^*x=b$  per X  
l'espressione  $(A'*A)$  ( $A'*b$ ) impiega lo stesso operatore quindi risolve il sistema delle equazioni lineari  
delle due parentesi. Le due espressioni sono equivalenti dal punto di vista matematico, ma in Matlab non  
danno lo stesso risultato, siccome nella seconda espressioni un overflow nel calcolo della moltiplicazione  
all'interno delle due parentesi che porta a porre una serie di valori a 0, questo fa si che non si ottiene lo  
stesso risultato

## 4 Capitolo 4

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

## 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
19 c = [c flip(c)];
20 if mod(n,2)==0
21     c(n/2+1) = [];
22 end
23 return
24 end

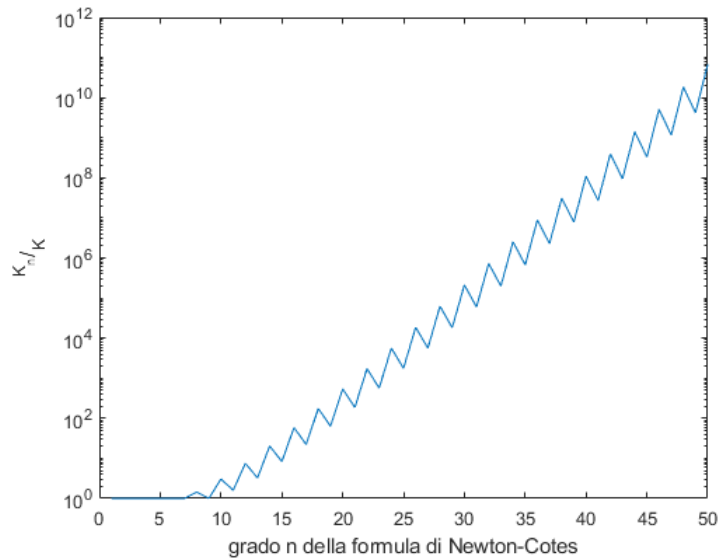
```

### 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 il rapporto per  $n = 1, \dots, 50$  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,
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

```

```

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

```

RISULTATI:



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