

Elaborato di
Calcolo Numerico
Anno Accademico 2016/2017

Gabriele Puliti - 5300140 - *gabriele.puliti@stud.unifi.it*
Luca Passaretta - -

August 31, 2017

Capitoli

1	Capitolo 1	5
1.1	Esercizio 1	5
1.2	Esercizio 2	5
1.3	Esercizio 3	6
1.4	Esercizio 4	6
1.5	Esercizio 5	7
1.6	Esercizio 6	8
1.7	Esercizio 7	9
1.8	Esercizio 8	9
1.9	Esercizio 9	10
1.10	Esercizio 10	10
1.11	Esercizio 11	10
1.12	Esercizio 12	11
1.13	Esercizio 13	11
2	Capitolo 2	12
2.1	Esercizio 1	12
2.2	Esercizio 2	12
2.3	Esercizio 3	13
2.4	Esercizio 4	14
2.5	Esercizio 5	15
2.6	Esercizio 6	15
2.7	Esercizio 7	17
2.8	Esercizio 8	18
2.9	Funzioni MatLab Usate	18
2.9.1	Metodo Newton per $\sqrt{\alpha}$	18
2.9.2	Metodo Newton per $\sqrt[n]{\alpha}$	18
2.9.3	Metodo delle secanti per $\sqrt[n]{\alpha}$	19
2.9.4	Metodo di Newton Modificato	19
2.9.5	Metodo di accelerazione di Aitken	20
2.9.6	Metodo delle secanti	21
2.9.7	Metodo della bisezione	22
2.9.8	Metodo delle corde	22
3	Grafici	i
3.1	Esercizio 1.4	i
3.2	Esercizio 1.13	i

1 Capitolo 1

1.1 Esercizio 1

Sapendo che il metodo iterativo è convergente a x^* allora per definizione si ha:

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

inoltre per definizione di Φ si calcola il limite:

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

infine ipotizzando che la funzione Φ sia uniformemente continua, è possibile calcolare il limite:

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

dai due limiti si ha la tesi:

$$\Phi(x^*) = x^*$$

1.2 Esercizio 2

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

```
integer*2 numero
```

varia tra $-32768 \leq \text{numero} \leq 32767$ ($-2^{15} \leq \text{numero} \leq 2^{15} - 1$).

Durante la terza iterazione del primo ciclo for si arriva al valore massimo rappresentabile tramite gli interi a 2 byte; alla quarta iterazione si avrà quindi la somma del **numero** in modulo e segno:

$$\begin{aligned} (32767)_{10} + (1)_{10} &= (0111111111111111)_{2,MS} + (0000000000000001)_{2,MS} = \\ &= (1000000000000000)_{2,MS} = (-32768)_{10} \end{aligned}$$

Nel secondo ciclo for, durante la quinta iterazione, al **numero** viene sottratto 1:

$$\begin{aligned} (-32768)_{10} - (1)_{10} &= (1000000000000000)_{2,MS} - (0000000000000001)_{2,MS} = \\ &= (0111111111111111)_{2,MS} = (32767)_{10} \end{aligned}$$

Da cui si spiega l'output del codice.

1.3 Esercizio 3

Per definizione si ha che la precisione di macchina u , per arrotondamento e' data da:

$$u = \frac{1}{2}b^{1-m}$$

Se $b = 8, m = 5$ si ha:

$$u = \frac{1}{2} \cdot 8^{1-5} = \frac{1}{2} \cdot 8^{-4} = 1,2 \cdot 10^{-4}$$

1.4 Esercizio 4

Il codice seguente:

```
1 format long e;  
2  
3 h=zeros(12,1);  
4 f=zeros(12,1);  
5  
6 for i=1:12  
7     h(i)= power(10,-i);  
8 end  
9  
10 for j=1:12  
11     f(j)=lim(0,v(j));  
12 end  
13  
14 f  
15  
16 function p=lim(x,y)  
17     p=(exp(x+y) - exp(x))/y;  
18 end
```

restituisce questo risultato (assumendo che $f(x) = e^x$ e $x_0 = 0$):

h	$\Psi_h(0)$
10^{-1}	1.051709180756477e+00
10^{-2}	1.005016708416795e+00
10^{-3}	1.000500166708385e+00
10^{-4}	1.000050001667141e+00
10^{-5}	1.000005000006965e+00
10^{-6}	1.000000499962184e+00
10^{-7}	1.000000049433680e+00
10^{-8}	9.999999939225290e-01
10^{-9}	1.000000082740371e+00
10^{-10}	1.000000082740371e+00
10^{-11}	1.000000082740371e+00
10^{-12}	1.000088900582341e+00

si può notare che al diminuire del valore h , la funzione $\Psi_h(0)$ approssima con maggior precisione il valore $f'(0)$, come si può vedere dal plot 1.

1.5 Esercizio 5

Per dimostrare le due uguaglianze è necessario sviluppare in serie di Taylor $f(x)$ fino al secondo ordine:

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

Da cui possiamo sostituire con i valori di $x = x + h$ e $x = x - h$:

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

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

Andando a sostituire questi valori si ottiene, nel primo caso:

$$\begin{aligned} & \frac{f(x_0 + h) - f(x_0 - h)}{2h} = \\ &= \frac{(f(x_0) + hf'(x_0) + \frac{h^2 f''(x_0)}{2} + O(h^2)) - (f(x_0) - hf'(x_0) + \frac{h^2 f''(x_0)}{2} + O(h^2))}{2h} = \\ &= \frac{2hf'(x_0) + O(h^2)}{2h} = f'(x_0) + O(h^2) \end{aligned}$$

nel secondo caso:

$$\begin{aligned} & \frac{f(x_0 + h) - 2f(x_0) + f(x_0 - h))}{h^2} = \\ = & \frac{f(x_0) + hf'(x_0) + \frac{h^2 f''(x_0)}{2} + O(h^2) - 2f(x_0) + f(x_0) - hf'(x_0) + \frac{h^2 f''(x_0)}{2} + O(h^2)}{h^2} = \\ = & \frac{h^2 f''(x_0) + O(h^2)}{h^2} = f''(x_0) + O(h^2) \end{aligned}$$

Abbiamo quindi dimostrato che:

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

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

1.6 Esercizio 6

Il codice MatLab, indicando con $x=x_n$ e $r=\epsilon$:

```

1 format longEng
2 format compact
3
4 conv=sqrt(2);
5 x=[2,1.5];
6 r=[x(1)-conv,x(2)-conv];
7
8 for i= 2:7
9     x(i+1) = (x(i)*x(i-1)+2)/(x(i)+x(i-1))
10 end
11
12 for i=3:8
13     r(i)=x(i)-conv;
14 end
15
16 x
17 r

```


restituisce i valori:

n	x_n	ϵ
0	2.000000000000000e+000	585.786437626905e-003
1	1.500000000000000e+000	85.7864376269049e-003
2	1.42857142857143e+000	14.3578661983335e-003
3	1.41463414634146e+000	420.583968367971e-006
4	1.41421568627451e+000	2.12390141496321e-006
5	1.41421356268887e+000	315.774073555986e-012
6	1.41421356237310e+000	0.000000000000000e+000
7	1.41421356237310e+000	0.000000000000000e+000

I valori indicano che per valori di n superiori a 5 l'errore, indicato con ϵ , è dell'ordine di 10^{-12} .

1.7 Esercizio 7

Sapendo che la rappresentazione del numero è stata fatta usando l'arrotondamento, la precisione di macchina si calcola:

$$u = \frac{b^{1-m}}{2}$$

il cui valore sappiamo essere pari a:

$$u \approx 4.66 \cdot 10^{-10}$$

dato che stiamo cercando il numero di cifre binarie allora si deve avere $b = 2$, è quindi possibile ricavare m :

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

possiamo pertanto affermare che servono 32 cifre dedicate alla mantissa per rappresentare il numero con precisione macchina $4.66 \cdot 10^{-10}$.

1.8 Esercizio 8

Sapendo che la mantissa in decimale è calcolabile tramite la funzione:

- $m = 1 - \log_{10}u$ (troncamento)
- $m = 1 - \log_{10}(2 \cdot u)$ (arrotondamento)

e che la precisione di macchina assuma un valore accettabilmente piccolo in modo tale che il $\log_{10}u \approx 1$, cioè $0 \leq u < 1$, allora è possibile scrivere:

- $m = 1 - \log_{10}u \approx -\log_{10}u$ (troncamento)
- $m = 1 - \log_{10}2u = 1 - \log_{10}2 - \log_{10}u \approx -\log_{10}u$ (arrotondamento)

1.9 Esercizio 9

```
1 x=0;  
2 delta = 1/10;  
3 while x<1,x=x+delta , end
```

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

1.10 Esercizio 10

All'interno della radice può presentarsi un problema di overflow dato che la somma dei due quadrati potrebbe essere molto grande, tanto grande da poter superare il limite massimo rappresentabile dalla macchina:

$$realmax = (1 - b^{-m}) \cdot b^{b^s - \nu}$$

Per risolvere questo problema è necessario prendere il massimo valore tra le due variabili:

$$m = \max\{|x|, |y|\}$$

e moltiplicare e dividere per questo valore:

$$\sqrt{x^2 + y^2} = m \cdot \frac{\sqrt{x^2 + y^2}}{m} = m \cdot \sqrt{\frac{x^2 + y^2}{m^2}} = m \cdot \sqrt{\left(\frac{x}{m}\right)^2 + \left(\frac{y}{m}\right)^2}$$

In questo modo si eviterà il problema di overflow, il problema è ben condizionato dato che potenza e radice sono ben condizionate e grazie alla modifica proposta indicata sopra.

1.11 Esercizio 11

Le due espressioni in aritmetica finita vengono scritte tenendo conto dell'errore di approssimazione sul valore reale:

- $fl(fl(fl(x) + fl(y)) + fl(z)) = ((x(1 + \varepsilon_x) + y(1 + \varepsilon_y))(1 + \varepsilon_a) + z(1 + \varepsilon_z))(1 + \varepsilon_b)$
- $fl(fl(x) + fl(fl(y) + fl(z))) = (x(1 + \varepsilon_x) + (y(1 + \varepsilon_y) + z(1 + \varepsilon_z))(1 + \varepsilon_a))(1 + \varepsilon_b)$

Indichiamo con $\varepsilon_x, \varepsilon_y, \varepsilon_z$ i relativi errori di x, y, z e con $\varepsilon_a, \varepsilon_b$ gli errori delle somme, per calcolare l'errore relativo delle due espressioni consideriamo $\varepsilon_m = \max\{\varepsilon_x, \varepsilon_y, \varepsilon_z, \varepsilon_a, \varepsilon_b\}$, dalla definizione di errore relativo si ha quindi:

•

$$\begin{aligned}\varepsilon_1 &= \frac{((x(1+\varepsilon_x) + y(1+\varepsilon_y))(1+\varepsilon_a) + z(1+\varepsilon_z))(1+\varepsilon_b) - (x+y+z))}{x+y+z} \approx \\ &\approx \frac{x(1+\varepsilon_x + \varepsilon_a + \varepsilon_b) + y(1+\varepsilon_y + \varepsilon_a + \varepsilon_b) + z(1+\varepsilon_z + \varepsilon_b) - x - y - z}{x+y+z} \leq \\ &\leq \frac{3 \cdot x \cdot \varepsilon_m + 3 \cdot y \cdot \varepsilon_m + 2 \cdot z \cdot \varepsilon_m}{x+y+z} \leq \frac{3 \cdot \varepsilon_m \cdot (x+y+z)}{x+y+z} = 3 \cdot \varepsilon_m\end{aligned}$$

- seguendo gli stessi procedimenti del punto precedente possiamo scrivere:

$$\begin{aligned}\varepsilon_2 &= \frac{(x(1+\varepsilon_x) + (y(1+\varepsilon_y) + z(1+\varepsilon_z))(1+\varepsilon_a))(1+\varepsilon_b) - (x+y+z))}{x+y+z} = \\ &= \dots \leq \frac{2 \cdot x \cdot \varepsilon_m + 3 \cdot y \cdot \varepsilon_m + 3 \cdot z \cdot \varepsilon_m}{x+y+z} \leq \frac{3 \cdot \varepsilon_m \cdot (x+y+z)}{x+y+z} = 3 \cdot \varepsilon_m\end{aligned}$$

1.12 Esercizio 12

Sapendo che il numero di condizionamento del problema è dato da:

$$k = \left| f'_x \cdot \frac{x}{f(x)} \right|$$

Dato che la nostra funzione è $f(x) = \sqrt{x}$ allora la derivata è data da $f'(x) = \frac{1}{2\sqrt{x}}$, sostituendo i valori otteniamo, come volevamo:

$$k = \left| \frac{1}{2 \cdot \sqrt{x}} \cdot \frac{x}{\sqrt{x}} \right| = \left| \frac{1}{2} \right| = \frac{1}{2}$$

1.13 Esercizio 13

Il problema è che stiamo rappresentando dei numeri reali in un calcolatore, quindi la loro rappresentazione comporta delle approssimazioni. Nella riga 11 abbiamo calcolato e restituito in output il valore interno al logaritmo $\left| 3\left(1 - \frac{3}{4}\right) + 1 \right|$ che teoricamente è zero, invece si ottiene $2.220446049250313e-16$. Si può vedere che il codice MatLab:

```

1 format long;
2
3 x=linspace(2/3,2,1001);
4 y= [];
5
6 for i = 1:1001
7     y(i) = log(abs(3*(1-x(i))+1))/80 + x(i)^2 +1;
8 end
9
10 plot(x,y);
11 disp (abs(3*(1-4/3)+1))

```

calcola i valori della funzione ottenendo il grafico 2 e si può notare che l'asintoto verticale in $x = \frac{4}{3}$ viene rappresentato come minimo della funzione $f(x)$.

2 Capitolo 2

Le funzioni usate nei codici seguenti sono in fondo al capitolo

2.1 Esercizio 1

```

1 x_0 = 3;
2 alpha = 3;
3 n = NewtonSqrt(alpha, x_0, 100, 0.0001);

```

La tabella delle approssimazioni è la seguente:

i	x_i
1	1.7500000000000000e+00
2	1.732142857142857e+00
3	1.732050810014727e+00

2.2 Esercizio 2

La radice da approssimare in questo caso è $\sqrt[n]{2}$ per gli $n = 3, 4, 5$. Il codice MatLab corrispondente è il seguente:

```

1 x_0 = 3;
2 alpha = 3;

```

```

3 disp('n=3');
4 n3 = NewtonSqrtN(3, alpha, x_0, 100, 0.0001);
5 disp('n=4');
6 n4 = NewtonSqrtN(4, alpha, x_0, 100, 0.0001);
7 disp('n=5');
8 n5 = NewtonSqrtN(5, alpha, x_0, 100, 0.0001);

```

l'output corrispondente è:

i	$n = 3$	$n = 4$	$n = 5$
1	1.631784138709347e+00	1.771797299323380e+00	1.943788863498140e+00
2	1.463411989089094e+00	1.463688102853308e+00	1.597060655491283e+00
3	1.442554125137959e+00	1.336940995805593e+00	1.369877122538772e+00
4	1.442249634601091e+00	1.316557487370408e+00	1.266284124539191e+00
5	1.442249570307411e+00	1.316074279204018e+00	1.246387399421677e+00
6	_____	1.316074012952573e+00	1.245731630753065e+00
7	_____	_____	1.245730939616284e+00

2.3 Esercizio 3

Per confrontare il metodo delle secanti con quello di Newton abbiamo creato il codice MatLab:

```

1 x_0 = 3;
2 alpha = x_0;
3 n= 2;
4
5 s = SecSqrtN(n, alpha, x_0, 100, 0.0001);

```

I risultati ottenuti dall'utilizzo del metodo delle secanti sono:

i	metodo di Newton	metodo delle secanti	$ newton - \sqrt{2} $	$ secanti - \sqrt{2} $
1	1.750000000000000e+00	1.736842105263158e+00	3.357864376269049e-01	3.226285428900628e-01
2	1.732142857142857e+00	1.732142857142857e+00	3.179292947697618e-01	3.179292947697618e-01
3	1.732050810014727e+00	1.732050934706042e+00	3.178372476416318e-01	3.178373723329468e-01
4	_____	1.732050807572255e+00	_____	3.178372451991598e-01

Si nota dalla tabella che $|newton - \sqrt{2}| \approx |secanti - \sqrt{2}|$, cioè che l'ordine di grandezza dell'errore assoluto tra i due metodi è lo stesso. Si può quindi affermare che l'uso del metodo di Newton o del metodo delle secanti è equivalente.

2.4 Esercizio 4

```
1 funct = @(x) (x-pi)^10;
2 dfunct = @(x) 10*(x-pi)^9;
3 disp('newton modificato funct 1');
4 NM11 = NewtonMod(funct, dfunct, 1, 3, 50, 1, 0)
5 NM12 = NewtonMod(funct, dfunct, 1, 3, 50, 0.1, 0)
6 NM13 = NewtonMod(funct, dfunct, 1, 3, 50, 0.01, 0)
7 NM14 = NewtonMod(funct, dfunct, 1, 3, 50, 0.001, 0)
8 NM15 = NewtonMod(funct, dfunct, 1, 3, 50, 0.0001, 0)
9 NM16 = NewtonMod(funct, dfunct, 1, 3, 50, 0.00001, 0)
10 NM17 = NewtonMod(funct, dfunct, 1, 3, 50, 0.000001, 0)
11 disp('aitken funct 1');
12 A11 = Aitken(funct, dfunct, 3, 50, 1)
13 A12 = Aitken(funct, dfunct, 3, 50, 0.1)
14 A13 = Aitken(funct, dfunct, 3, 50, 0.01)
15 funct2 = @(x) ((x-pi)^10)*(exp(1)^(2*x));
16 dfunct2 = @(x) (5+x-pi)*(x-pi)^9*2*exp(1)^(2*x);
17 disp('newton modificato funct 2');
18 NM21 = NewtonMod(funct2, dfunct2, 1, 3, 50, 1, 0)
19 NM22 = NewtonMod(funct2, dfunct2, 1, 3, 50, 0.1, 0)
20 NM23 = NewtonMod(funct2, dfunct2, 1, 3, 50, 0.01, 0)
21 NM24 = NewtonMod(funct2, dfunct2, 1, 3, 50, 0.001, 0)
22 NM25 = NewtonMod(funct2, dfunct2, 1, 3, 50, 0.0001, 0)
23 NM26 = NewtonMod(funct2, dfunct2, 1, 3, 50, 0.00001,
    0)
24 NM27 = NewtonMod(funct2, dfunct2, 1, 3, 50, 0.000001,
    0)
25 disp('aitken funct 2');
26 A21 = Aitken(funct2, dfunct2, 3, 50, 1)
27 A22 = Aitken(funct2, dfunct2, 3, 50, 0.1)
28 A23 = Aitken(funct2, dfunct2, 3, 50, 0.01)
```

Questo codice esegue i metodi di Newton, Newton modificato e Aitken per le funzioni date.

L'output della $f_1(x)$ rispetto alle diverse tolleranze sono:

tolx	Newton modificato	Aitken
1	3.014159265358979	3.141592653589755
0.1	3.014159265358979	3.141592653589789
0.01	3.057983607571556	3.141592653589789
0.001	3.133359078021984	_____
0.0001	3.140781835025782	_____
0.00001	3.140862916882183	_____
0.000001	3.140862916882183	_____

L'output della $f_2(x)$ rispetto alle diverse tolleranze è:

tolx	Newton modificato	Aitken
1	3.014571920744193	3.137995613065127
0.1	3.014571920744193	3.141590324813442
0.01	3.059075504146139	3.141590324813442
0.001	3.132710330277176	_____
0.0001	3.140719503754643	_____
0.00001	3.140885428300940	_____
0.000001	3.140885428300940	_____

2.5 Esercizio 5

Il metodo di bisezione è applicabile in f se è:

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

il metodo di bisezione non è possibile utilizzarlo a causa della seconda condizione dato che $f_1(x) = (x - \pi)^{10}$ e $f_2(x) = e^{2x}(x - \pi)^{10}$ sono sempre positive quindi non è possibile stabilire un intervallo $[a, b]$ tale che $f(a)f(b) < 0, \forall a, b \in \mathbb{R}$

2.6 Esercizio 6

```

1 myf = @(x) (1-x-(1+cos(10*x)/2)*sin(x));
2 df = @(x) (5*sin(x)*sin(10*x)-cos(x)*(cos(10*x)/2 + 1)
3      -1);

```

```

4 x0 = 0;
5 x1 = 1;
6 tol = logspace(-1,-10,10);
7 disp(tol);
8 imax = 50;
9 count = zeros(3,10);
10 for i=1:10
11     [temp, count(1,i)] = NewtonMod(myf,df,1,x0,imax,
12                                   tol(i), 0);
13     [temp, count(2,i)] = secanti(myf,x0,x1,tol(i),imax
14                                   );
15     [temp, count(3,i)] = corde(myf,feval(df,x0),x0,tol
16                                   (i),imax);
17 end
18 disp('nella prima riga ci sono le iterazioni di newton
19     modificato,');
20 disp('nella seconda riga ci sono il numero di
21     iterazioni del metodo delle secanti');
22 disp('nella terza riga ci sono il numero di iterazioni
23     del metodo delle corde');
24 count

```

Il numero di iterazioni effettuate all'interno dei 3 algoritmi, vengono salvate nella variabile count che restituisce i valori sotto forma di tabella:

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

2.7 Esercizio 7

```
1 format long
2
3 myf = @(x) (1-x-(1+cos(10*x)/2)*sin(x));
4 df = @(x) (5*sin(x)*sin(10*x)-cos(x)*(cos(10*x)/2 + 1)
5         -1);
6
7 x0 = 0;
8 x1 = 1;
9 tol = logspace(-1,-10,10);
10 imax = 50;
11 count = zeros(1,10);
12 for i=1:10
13     [temp, count(1,i)] = bisection(myf,x0,x1,tol(i),imax)
14     ;
15     dips('B');
16     i
17     temp
18 end
19 disp('nella riga ci sono il numero di iterazioni del
20     metodo della bisezione');
21 count
```

Il numero di iterazioni del metodo di bisezione risultanti sono:

tol_x	bisezione
10^{-1}	0
10^{-2}	7
10^{-3}	10
10^{-4}	13
10^{-5}	16
10^{-6}	20
10^{-7}	21
10^{-8}	26
10^{-9}	30
10^{-10}	32

2.8 Esercizio 8

```
1 myfunct = @(x) (x-pi)^(10*x);
2 dfdx = @(x) 10*((x-pi)^(10*x))*(x/(x-pi)+log(x-pi));
3 x0 = 0;
4
5 y = NewtonMod(myfunct, dfdx, 1, x0, 5, 0.01, 1);
```

Il codice Matlab soprastante restituirà in output un valore $\notin \mathbb{R}$, più precisamente il valore è pari a $-0.010239076641281 + 0.028100085752476i$ che $\in \mathbb{C}$. Questo significa che il metodo non potrà convergere partendo dal punto iniziale x_0 .

2.9 Funzioni MatLab Usate

2.9.1 Metodo Newton per $\sqrt{\alpha}$

```
1 function y = NetwtonSqrt(alpha, x0, imax, tol)
2     format long e
3     x = (x0 + alpha/x0)/2;
4     i = 1;
5     while(i < imax) && (abs(x-x0)>tol)
6         x0=x;
7         i = i+1;
8         x = (x0 + alpha/x0)/2;
9         disp(x);
10    end
11    y = x;
12
13 end
```

2.9.2 Metodo Newton per $\sqrt[n]{\alpha}$

```
1 function y = NetwtonSqrtN(n, alpha, x0, imax, tol)
2     format long e
3     x = (((n-1)*x0^n + alpha)/ x0^(n-1)) / n;
4     i = 1;
5     while(i < imax) && (abs(x-x0)>tol)
6         x0=x;
```

```

7         i = i+1;
8         x = (((n-1)*x0^n + alpha)/ x0^(n-1)) / n;
9         disp(x);
10    end
11    y = x;
12 end

```

2.9.3 Metodo delle secanti per $\sqrt[n]{\alpha}$

```

1 function y = SecSqrtN(n, alpha, x0, imax, tol)
2     format long e
3     x1 = (x0 + alpha/x0)/2;
4     x = (f(x1,n, alpha) * x0 - f(x0,n, alpha)*x1 ) / (
5         f(x1, n, alpha) - f(x0, n, alpha));
6     i = 1;
7     while(i < imax) && (abs(x-x0)>tol)
8         x0=x1;
9         x1=x;
10        i = i+1;
11        x = (f(x1,n, alpha) * x0 - f(x0,n, alpha)*x1 )
12            / (f(x1, n, alpha) - f(x0, n, alpha));
13        disp(x)
14    end
15    y = x;
16 end
17
18 function y = f(x, n, alpha)
19     y = (x ^ n) - alpha;
20 end

```

2.9.4 Metodo di Newton Modificato

```

1 function [y, i] = NewtonMod(f, df, m, x0, imax, tol,
2     output)
3     format long;
4     i = 0;
5     x = x0;

```

```

5     vai=1;
6     while((i < imax) && vai)
7         i = i+1;
8         fx = feval(f, x0);
9         dfx = feval(df, x0);
10        if(dfx ~= 0)
11            x = x0 - m * fx / dfx;
12        else
13            break;
14        end
15        if(abs(x-x0)<tol)
16            vai = 0;
17        end
18        x0 = x;
19    end
20    if(output)
21        if(vai)
22            disp(' [Errore] Tolleranza non calcolabile '
23                );
24        else
25            disp(i);
26        end
27    end
28    y = x;
end

```

2.9.5 Metodo di accelerazione di Aitken

```

1 function y = Aitken( f, df, x0, imax, tol )
2     format long;
3     i = 0;
4     vai=1;
5
6     while((i < imax) && vai)
7         x1 = NewtonMod(f, df, 1, x0, 1, tol, 0);
8         x2 = NewtonMod(f, df, 1, x1, 1, tol, 0);
9         i = i+1;
10        if((x0 - 2*x1 +x2) == 0)
11            disp(' [Errore] Divisione per 0');

```

```

12         vai = 0;
13         break;
14     end
15     x = (x2*x0 - x1^2)/(x0 - 2*x1 +x2);
16     if(abs(x-x0)<tol)
17         vai = 0;
18     end
19     x0 = x;
20 end
21 if(vai)
22     disp(' [Errore] Tolleranza non calcolabile ');
23     disp(i);
24 end
25 y = x;
26 end

```

2.9.6 Metodo delle secanti

```

1 function [x,nit , tolf]=secanti(f,x0,x1,tolx ,maxit)
2     nit=0;
3     fx0=fval(f,x0);
4     err=abs(x1-x0);
5     while (nit<maxit & err>tolx)
6         fx1=fval(f,x1);
7         dfx1=(fx1-fx0)/(x1-x0);
8         tolf=tolx*abs(dfx1);
9         if abs(fx1)<=tolf
10             break
11         end
12         x2=x1-(fx1/dfx1);
13         err=abs(x2-x1);
14         x0=x1;
15         x1=x2;
16         fx0=fx1;
17         nit=nit+1;
18     end
19     x=x1;
20 end

```

2.9.7 Metodo della bisezione

```
1 function [sol , nit]=bisect ( f , a , b , tol x , maxit ) ;
2     nit=maxit ;
3     j=0;
4     if (subs ( f , a ) *subs ( f , b ) > 0)
5         disp ( ' [Warning] il metodo non puo'' essere
6             usato ' ) ;
7     else
8         while ( j <= maxit )
9             sol = ( a + b ) / 2 ;
10            ff = subs ( f , sol ) ;
11            if ( abs ( ff ) <= tol x )
12                nit = j ;
13                break ;
14            end
15            fa = subs ( f , a ) ;
16            fm = subs ( f , sol ) ;
17            if ( fa * fm <= 0 )
18                b = sol ;
19            else
20                a = sol ;
21            end
22            j = j + 1 ;
23 end
```

2.9.8 Metodo delle corde

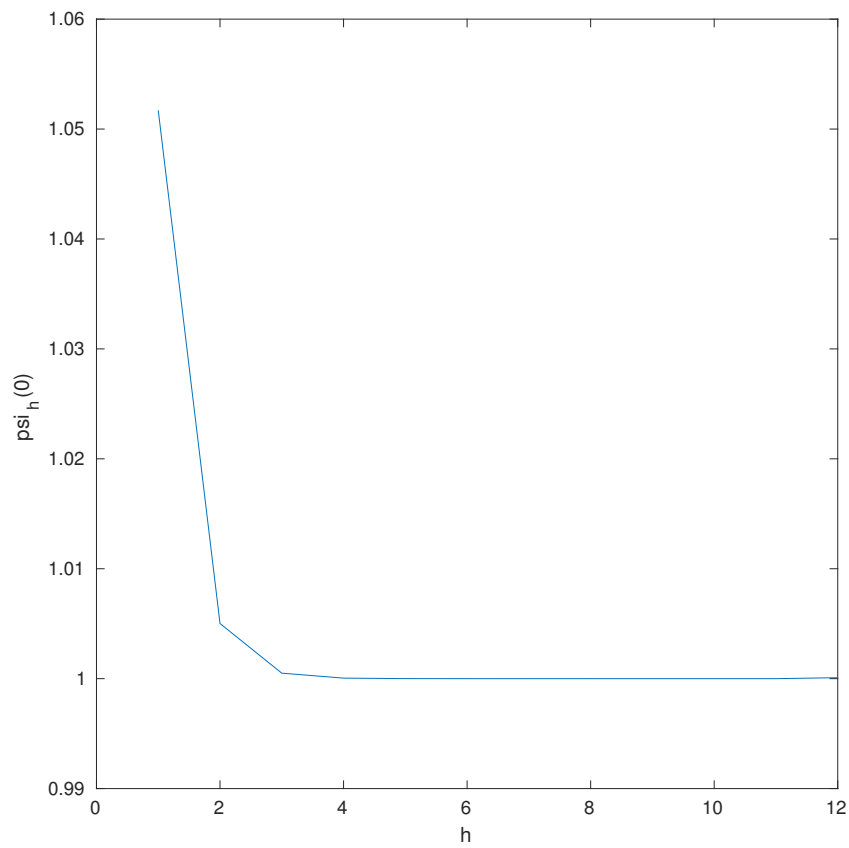
```
1 function [x , nit]=corde ( f , m , x0 , tol x , maxit )
2     nit=0;
3     err=tol x + 1;
4     x=x0;
5     while ( nit < maxit && err > tol x )
6         fx=feval ( f , x ) ;
7         tolf=tol x * abs ( m ) ;
8         if abs ( fx ) <= tolf
9             break
10        end
```

```
11         x1=x-fx/m;  
12         err=abs(x1-x);  
13         x=x1;  
14         nit=nit+1;  
15     end  
16 end
```

3 Grafici

3.1 Esercizio 1.4

Figure 1: Esercizio 1.4



3.2 Esercizio 1.13

Figure 2: Esercizio 1.13

