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

1.1 Esercizio 1

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

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

inoltre per definizione di Φ si calcola il limite:

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

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

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

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 $-32768 \leq$ `numero` ≤ 32767 ($-2^{15} \leq$ `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:

$$(32767)_{10} + (1)_{10} = (0111111111111111)_{2,MS} + (0000000000000001)_{2,MS} =$$

$$= (1000000000000000)_{2,MS} = (-32768)_{10}$$

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

$$(-32768)_{10} - (1)_{10} = (1000000000000000)_{2,MS} - (0000000000000001)_{2,MS} =$$

$$= (0111111111111111)_{2,MS} = (32767)_{10}$$

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:

```
format long e;

h=zeros(12,1);
f=zeros(12,1);

for i=1:12
    h(i)= power(10,-i);
end

for j=1:12
    f(j)=lim(0,v(j));
end

f

function p=lim(x,y)
    p=(exp(x+y) - exp(x))/y;
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:

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

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

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

nel secondo caso:

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

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

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

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=ϵ:

```
format longEng
format compact

conv=sqrt(2);
x=[2,1.5];
r=[x(1)-conv,x(2)-conv];

for i= 2:7
    x(i+1) = (x(i)*x(i-1)+2)/(x(i)+x(i-1))
end

for i=3:8
    r(i)=x(i)-conv;
end

x
r
```

restituisce i valori:

n	x_n	ϵ
0	2.00000000000000e+000	585.786437626905e-003
1	1.50000000000000e+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	0.00000000000000e+000
7	1.41421356237310e+000	0.00000000000000e+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:

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

- seguendo gli stessi procedimenti del punto precedente possiamo scrivere:
$$\varepsilon_2 = \frac{(x(1+\varepsilon_x) + (y(1+\varepsilon_y) + z(1+\varepsilon_z))(1+\varepsilon_a))(1+\varepsilon_b) - (x+y+z)}{x+y+z} =$$
$$= ... \leq \frac{2\cdot x\cdot\varepsilon_m + 3\cdot y\cdot\varepsilon_m + 3\cdot z\cdot\varepsilon_m}{x+y+z} \leq \frac{3\cdot\varepsilon_m\cdot(x+y+z)}{x+y+z} = 3\cdot\varepsilon_m$$

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 $f'(x) = \frac{1}{2\cdot\sqrt{x}}$, 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 $\left|3(1-\frac{3}{4})+1\right|$ che teoricamente è zero, invece si ottiene $2.220446049250313e-16$. Si può vedere che il codice MatLab:

```
format long;

x=linspace(2/3,2,1001);
y= [];

for i = 1:1001
    y(i) = log(abs(3*(1-x(i))+1))/80 + x(i)^2 +1;
end

plot(x,y);
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

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

La tabella delle approssimazioni è la seguente:

i	x_i
1	1.750000000000000e+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:

```
x_0 = 3;
alpha = 3;
```

```
disp('n=3');
n3 = NewtonSqrtN(3, alpha, x_0, 100, 0.0001);
disp('n=4');
n4 = NewtonSqrtN(4, alpha, x_0, 100, 0.0001);
disp('n=5');
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:

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

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	$\lvert$newton-$\sqrt{2}\rvert$	$\lvert$secanti-$\sqrt{2}\rvert$
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 $\left|newton - \sqrt{2}\right| \approx \left|secanti - \sqrt{2}\right|$, 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

```
funct = @(x) (x-pi)^10;
dfunct = @(x) 10*(x-pi)^9;
disp('newton modificato funct 1');
NM11 = NewtonMod(funct, dfunct, 1, 3, 50, 1, 0)
NM12 = NewtonMod(funct, dfunct, 1, 3, 50, 0.1, 0)
NM13 = NewtonMod(funct, dfunct, 1, 3, 50, 0.01, 0)
NM14 = NewtonMod(funct, dfunct, 1, 3, 50, 0.001, 0)
NM15 = NewtonMod(funct, dfunct, 1, 3, 50, 0.0001, 0)
NM16 = NewtonMod(funct, dfunct, 1, 3, 50, 0.00001, 0)
NM17 = NewtonMod(funct, dfunct, 1, 3, 50, 0.000001, 0)
disp('aitken funct 1');
A11 = Aitken(funct, dfunct, 3, 50, 1)
A12 = Aitken(funct, dfunct, 3, 50, 0.1)
A13 = Aitken(funct, dfunct, 3, 50, 0.01)
funct2 = @(x) ((x-pi)^10)*(exp(1)^(2*x));
dfunct2 = @(x) (5+x-pi)*(x-pi)^9*2*exp(1)^(2*x);
disp('newton modificato funct 2');
NM21 = NewtonMod(funct2, dfunct2, 1, 3, 50, 1, 0)
NM22 = NewtonMod(funct2, dfunct2, 1, 3, 50, 0.1, 0)
NM23 = NewtonMod(funct2, dfunct2, 1, 3, 50, 0.01, 0)
NM24 = NewtonMod(funct2, dfunct2, 1, 3, 50, 0.001, 0)
NM25 = NewtonMod(funct2, dfunct2, 1, 3, 50, 0.0001, 0)
NM26 = NewtonMod(funct2, dfunct2, 1, 3, 50, 0.00001,
    0)
NM27 = NewtonMod(funct2, dfunct2, 1, 3, 50, 0.000001,
    0)
disp('aitken funct 2');
A21 = Aitken(funct2, dfunct2, 3, 50, 1)
A22 = Aitken(funct2, dfunct2, 3, 50, 0.1)
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

2.7 Esercizio 7

2.8 Esercizio 8

2.9 Funzioni MatLab Usate

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

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

end
```

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

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

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

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

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

2.9.4 Metodo di Newton Modificato

```
1   function [y, i] = NewtonMod(f, df, m, x0, imax, tol,
        output)
2       format long;
3       i = 0;
4       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          else
24             disp(i);
25          end
26      end
27      y = x;
28  end
```

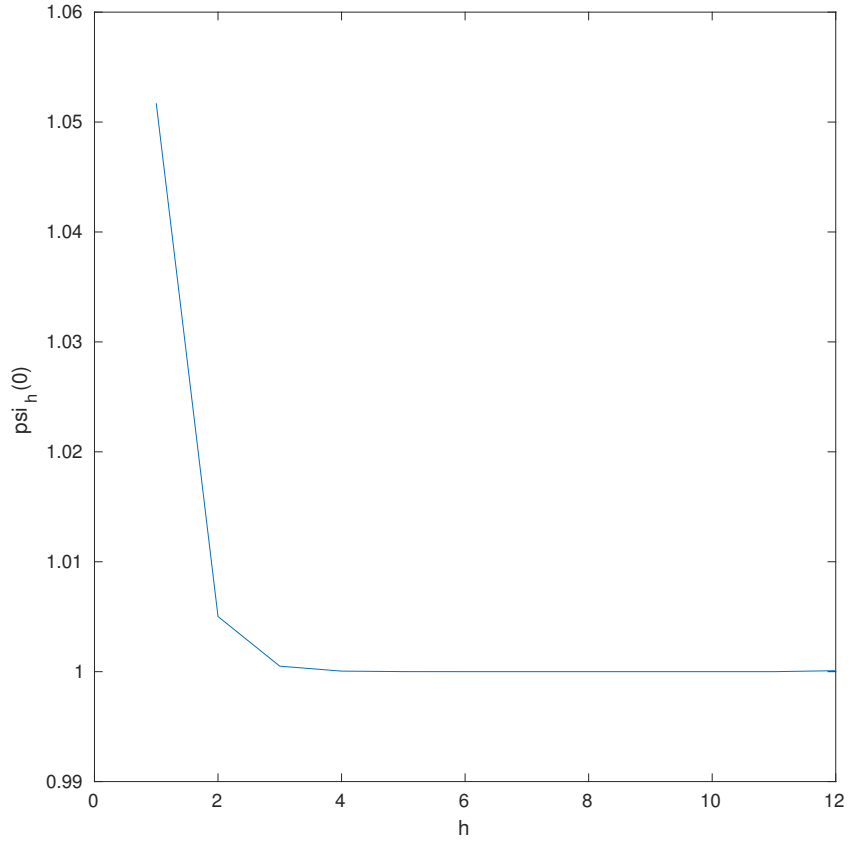
2.9.5 Metodo di accelerazione di Aitken

```
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
```

3 Grafici

3.1 Esercizio 1.4

Figure 1: Esercizio 1.4



3.2 Esercizio 1.13

Figure 2: Esercizio 1.13