Elaborato di

Calcolo Numerico

Giovanni *Bindi* - 5530804 - *giovanni.bindi@stud.unifi.it*
Gabriele *Gemmi* - 5602433 - *gabriele.gemmi@stud.unifi.it*
Gabriele *Puliti* - 5300140 - *gabriele.puliti@stud.unifi.it*

March 3, 2017

Contents

1 Capitolo 1

1.1 Esercizio 1.1

Per definizione di metodo iterativo convergente si ha che

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

Supponendo la funzione $\Phi(x_n)$ uniformemente continua vale

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

Per definizione é $\Phi(x_n) = x_{k+1}$ e quindi

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

Da cui otteniamo che x^* e' un punto fisso per la funzione $\Phi(x_n)$, ovvero che $x^* = \Phi(x^*)$.

1.2 Esercizio 1.2

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

```
integer*2 n
```

varia tra $-2^{15} \leq n \leq 2^{15} - 1$ e quindi tra $-32768 \leq n \leq 32767$.
Andando quindi ad eseguire la somma $(32767 + 1)_{10} = (0111111111111111 + 1)_{2,MS} =$
$= (1111111111111111)_{2,MS} = (-327628)_{10}$

1.3 Esercizio 1.3

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 $u = \frac{1}{2} \cdot 8^{-4} = 1,2207031 \cdot 10^{-4}$

1.4 Esercizio 1.4

Il seguente codice in Python

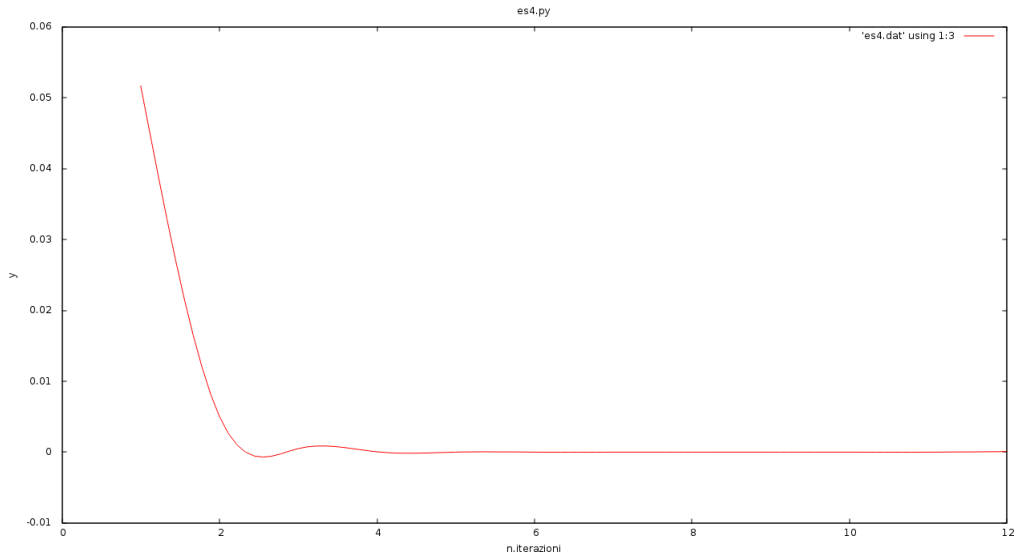
```
import scipy
import numpy, math

x_0 = 0
h = []
for i in range(1,13):
        h.append(math.pow(10,-i))

def psi(x,j):
        return (math.exp(x+j) - math.exp(x))/j

for i in h:
        print str(h.index(i)+1)+   + str(psi(x_0, i)) +
            + str(psi(x_0, i) -1)
```

restituisce questo output:

i	x_i	y_i
1	1.05170918076	0.0517091807565
2	1.00501670842	0.00501670841679
3	1.00050016671	0.000500166708385
4	1.00005000167	5.0001667141e-05
5	1.00000500001	5.00000696491e-06
6	1.00000049996	4.99962183653e-07
7	1.00000004943	4.94336802603e-08
8	0.999999993923	-6.07747097092e-09
9	1.00000008274	8.27403709991e-08
10	1.00000008274	8.27403709991e-08
11	1.00000008274	8.27403709991e-08
12	1.00008890058	8.8900582341e-05

Graficando il contenuto della tabella su di un piano XY abbiamo che

magari rendiamo piu carino il grafico ed aumentiamo la scala per le ultime iterazioni in cui la funzione oscilla tra e-8 e e-5



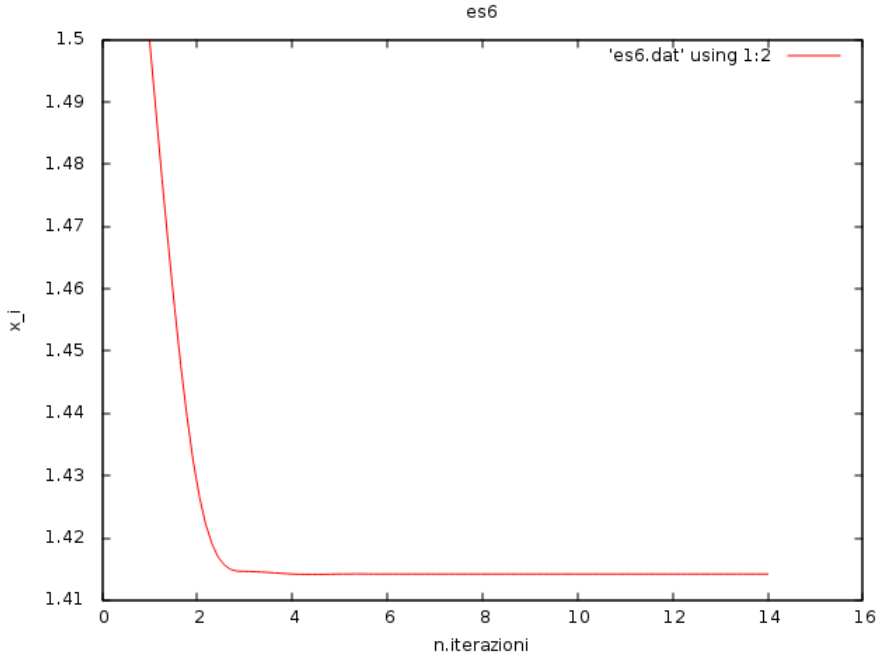
1.5 Esercizio 1.5

1.6 Esercizio 1.6

Il seguente codice `MATLAB`

```
format long
x = [2,1.5];
y = [];
rad = sqrt(2)

for i = 2:15
    x(i+1) =  ((x(i)*x(i-1) +2)/(x(i) + x(i-1)));
end

for i=1:15
    y(i) = x(i) - rad;
end
```

produce la seguente curva (non so se e' lui o il .py, da ricontrollare)



1.7 Esercizio 1.7

Abbiamo che $u = \frac{1}{2}b^{1-m} \approx 4.66 \cdot 10^{-10}$, supponendo che la base sia $b = 2$ otteniamo

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

Quindi necessitiamo di 32 cifre per la mantissa.

1.8 Esercizio 1.8

Supponendo $b = 10$ si ha, per entrambi i casi

- troncamento : $m = 1 - log_{10}u \approx -log_{10}u$
- arrotondamento : $m = 1 - log_1 02u = 1 - log_{10}2 - log_1 0u \approx -log_{10}u$

1.9 Esercizio 1.9

```
x=0; delta = 1/10;
while x ~= 1, x = x+delta, end

% the code is not working because the x will never be exactly 1. So it
% loops forever
```

1.10 Esercizio 1.10

1.11 Esercizio 1.11

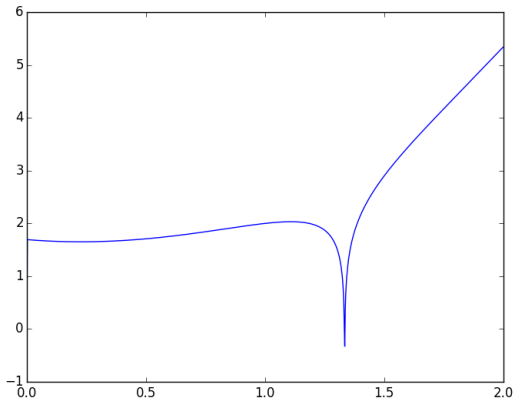
1.12 Esercizio 1.12

1.13 Esercizio 1.13

Il seguente codice é in in Python

```
import scipy as sp
import numpy as np
import math
import matplotlib.pyplot as plt

x = np.arange((2/3),2,0.002)
y = np.empty(1000, dtype=np.float64)

for i in range(1,1000):
        y[i] = (math.log(abs(3*(1-x[i]) + 1))/2 + x[i]*x[i] + 1)

l = 4.0/3
m = (math.log(abs(3*(1-l) + 1))/2 + l*l + 1)

print l
print m

plt.plot(x,y)
plt.savefig('es13.png')
```

e plotta la funzione



chiaro che son tutti grafici da rifare

2 Capitolo 2

2.1 Esercizio 2.1

```
%Definire una procedura iterativa basata sul metodo di Newton per

%determinare sqrt(alpha)

x_0 = 3;
alpha = x_0;

z = NewtonSqrt(3, 3, 200, 0.1)
```

2.2 Esercizio 2.2

```
%Definire una procedura iterativa basata sul metodo di Newton per
%determinare sqrt(alpha)

```

```
x_0 = 3;
alpha = x_0;

z = NewtonNSqrt(3, 3, 3, 200, 0.0001)
```

2.3 Esercizio 2.3

Versione MATLAB

```
%Definire una procedura iterativa basata sul metodo di Newton per
%determinare sqrt(alpha)

x_0 = 3;
alpha = x_0;

z = SecNSqrt(3, 3, 3, 200, 0.0001);
```

Verisione Python

```
import numpy as np
import math

def secanti(n,alpha,x0,imax,tolx):
        print x0 = +str(x0)
        x = ((x0+alpha/x0)/2.0)
        print x1 = +str(x)
        i = 1
        while ( (i<imax) & (math.fabs(x-x0)>tolx)):
                i = i+1
                x1 = (f(x,n,alpha)*x0 - x*f(x0,n,alpha))/(f(x,n,alpha) - f(
                    x0,n,alpha))
                x0 = x
                x=x1;
                print x+str(i)+=+str(x)

def f(x,n,alpha):
        return pow(x,n) - alpha

secanti(2.0,2.0,2.0,7,0.000001)
```

2.4 Esercizio 2.4

Versione MATLAB

```
mf = '(x-pi)^10';
dmf = '10*(x-pi)^9';

y = NetwonMod(mf, dmf, 1, 3, 50, 0.5);
```

Verisione Python