Lab 3 Report

Amlan Nayak MS18197

3rd Feb 2021

1 Find root of equation for $f(x) = cos(x) - xe^x$ using Muller's method

1.1 Algorithm and Flowchart

As taught in class lecture, a_0, a_1 and a_2 were calculated from the inputs. The initial input values were also stored. Then the sign of a_1 was used to determine variable x was defined in such a way as to minimize the change in c. With a, b and c being initial inputs, all the quantities were defined as:

$$D = (c-a)(c-b)(b-a)$$

$$a_0 = \frac{1}{D}[(f(c)-f(b))(c-a) - (f(c)-f(a))(c-b)]$$

$$a_1 = \frac{1}{D}[(f(c)-f(b))(c-a)^2 - (f(c)-f(a))(c-b)^2]$$

$$a_2 = f(c)$$

$$x = c - \frac{2a_2}{a_1 \pm \sqrt{a_1^2 - 4a_0a_2}}$$

We then redefined our variables in the computation as $a = b, b = c$ and $c = x$. This procedure loop over and over until the tolerance condition of $|\frac{c_n - c_{n-1}}{c_{n-1}}| > 10^{-7}$ is not satisfied. That gives us an approximate value of the root near out initial guesses.

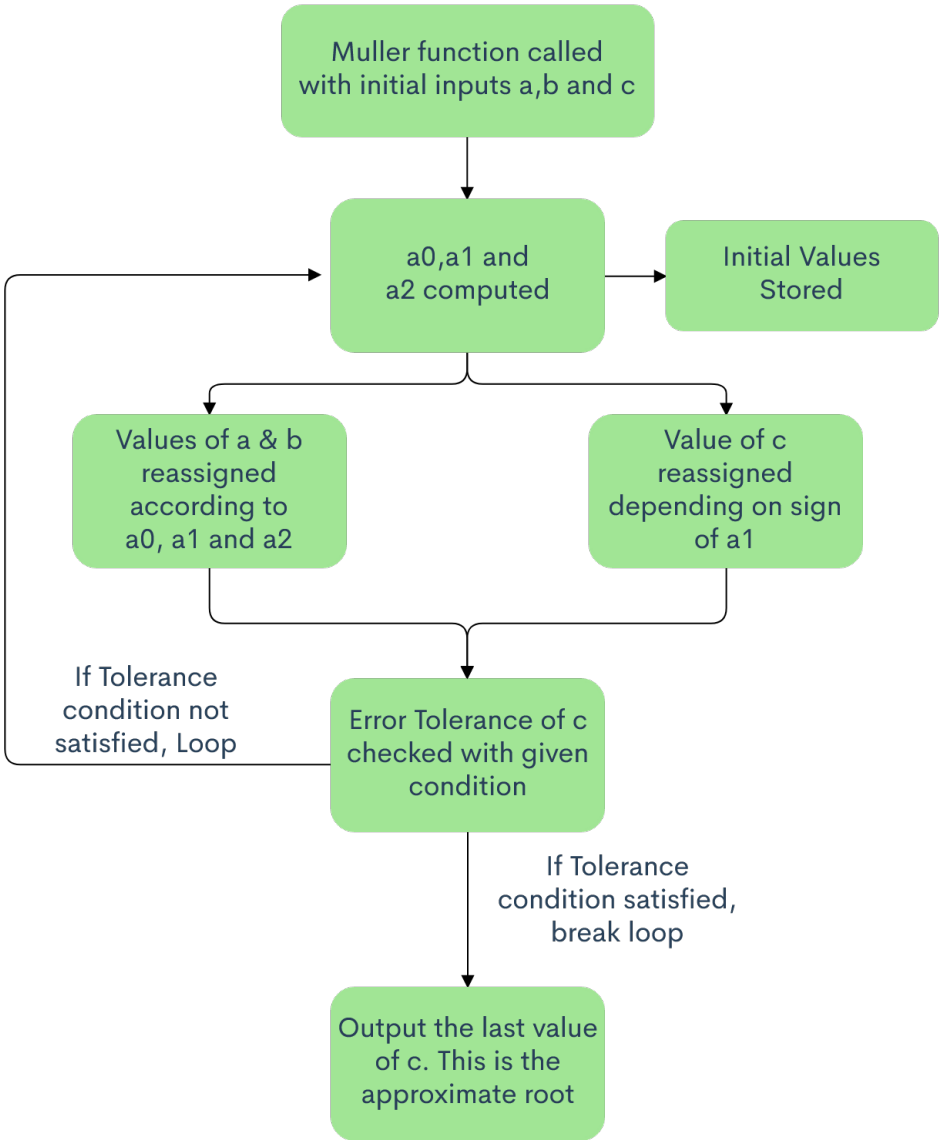

Figure 1: Flowchart of Muller method

1.2 Code and Output

```
import numpy as np
def f(x):
 return np.cos(x)-x*np.exp(x)
def muller(a,b,c):
 c_before=100
 x = 200
 s=a
 t=b
 u=c
 while abs((x - c_before) / c_before ) > 0.0000001:
  c_before=c
  D= (c-b)*(c-a)*(b-a)
  a_1= ((f(c)-f(b))*((c-a)**2) - (f(c)-f(a))*((c-b)**2)) / D
  a_0= ((f(c)-f(b))*(c-a) - (f(c)-f(a))*(c-b)) / D
  a_2= f(c)
  if a_1 >= 0:
   x = c - 2*a_2 / (a_1 + (a_1**2 - 4*a_0*a_2)**0.5)
  else:
   x = c - 2*a_2 / (a_1 - (a_1**2 - 4*a_0*a_2)**0.5)
  a=b
  b=c
  c=x
 print("The root for cos f(x)= cos(x) - x(e**x)is {} when initial guesses are ({},{},{})".format(c,s,t,u))
muller(-1,-2,-3),muller(-4,-6,-8)
```

Figure 2: Muller Code in Python in emacs

```
nayakamlan@DESKTOP-C21G1MH:~$ winhome
nayakamlan@DESKTOP-C21G1MH:/mnt/c/Users/Dell$ cd Desktop/CP1/Lab3
nayakamlan@DESKTOP-C21G1MH:/mnt/c/Users/Dell/Desktop/CP1/Lab3$ python3 MS18197_3_code1.py
The root for cos f(x)= cos(x) - x(e**x)is -1.8639951924025282 when initial guesses are (-1,-2,-3)
The root for cos f(x)= cos(x) - x(e**x)is -7.857022499050332 when initial guesses are (-4,-6,-8)
```

Figure 3: Output

2 Use multi-point iteration method to solve $f(x) = x^3 - 13x - 12$

2.1 Algorithm

Multi-point iteration method is two step method where we start with an initial guess. First multi-point method comprises of the following relations:

$$x^*_{k+1} = x_k - \frac{1}{2}\frac{f(x_k)}{f'(x_k)}$$

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x^*_{k+1})}$$

The value of x^*_{k+1} is computed from x_k in 1st equation and use to determine x_{k+1} in 2nd equation. We then use to similarly determine x_{k+2} from

x_{k+1} using the same two equations. This goes in a loop till the tolerance condition of $|\frac{x_k - x_{k-1}}{x_{k-1}}| > 10^{-8}$ is satisfied. Once it is not, the loop breaks and prints the value of x_k. This is the value of our approximate root.

The second multi-point iteration method is similar. The equations here are:

$$x^*{}_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

$$x_{k+1} = x^*{}_{k+1} - \frac{f(x^*_{k+1})}{f'(x_k)}$$

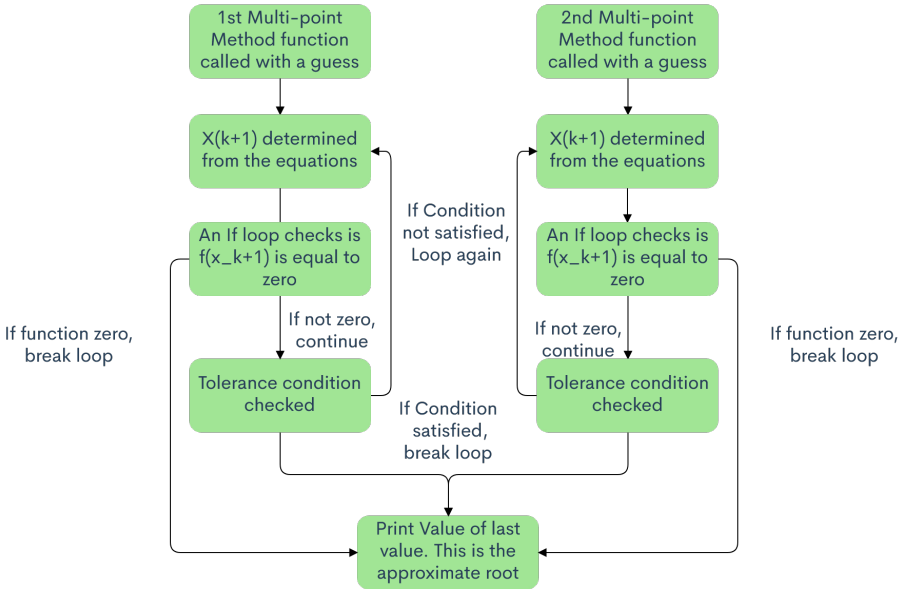

Figure 4: Flowchart of Multi-point Methods

2.2 Code and Output

```
1import numpy as np
2def f(x):
3  return x**3- 13*x - 12
4def f1(x):
5  return 3*(x**2) - 13
6def multipoint1(x):
7  x_ini=x
8  x_2= 200
9  x_before=300
10  while abs((x_2 - x_before)/x_before) > 0.00000001:
11    x_before=x
12    y = x - 0.5*(f(x)/f1(x))
13    x_2 = x - f(x)/f1(y)
14    x = x_2
15    if f(x)==0:
16      break
17    else:
18      continue
19  print("The root closest to {} of f(x) = x**3 - 13x - 12 is {} by first multipoint method".format(x_ini,x_2))
```

Figure 5: 1st multi-point method code in emacs

```
20def multipoint2(x):
21  x_ini=x
22  x_2= 200
23  x_before=300
24  while abs((x_2 - x_before)/x_before) > 0.00000001:
25    x_before=x
26    y = x - f(x)/f1(x)
27    x_2 = y - f(y)/f1(x)
28    x = x_2
29    if f(x)==0:
30      break
31    else:
32      continue
33  print("The root closest to {} of f(x) = x**3 - 13x - 12 is {} by second multipoint method".format(x_ini,x_2))
34multipoint1(5)
35multipoint1(-11)
36multipoint1(1)
37multipoint2(5)
38multipoint2(-11)
39multipoint2(1)
```

Figure 6: 2nd multi-point method code in emacs

```
nayakamlan@DESKTOP-C21G1MH:/mnt/c/Users/Dell/Desktop/CP1/Lab3$ python3 MS18197_3_code2.py
The root closest to 5 of f(x) = x**3 - 13x - 12 is 4.0 by first multipoint method
The root closest to -11 of f(x) = x**3 - 13x - 12 is -3.0 by first multipoint method
The root closest to 1 of f(x) = x**3 - 13x - 12 is -1.0 by first multipoint method
The root closest to 5 of f(x) = x**3 - 13x - 12 is 4.0 by second multipoint method
The root closest to -11 of f(x) = x**3 - 13x - 12 is -3.0 by second multipoint method
The root closest to 1 of f(x) = x**3 - 13x - 12 is -1.0 by second multipoint method
```

Figure 7: Output

3 Find root of equations correct to four decimal place for $f(x) = x^3+x^2-1 = 0$ and $g(x) = x-e^{-x} = 0$ using Iteration method and Aitken's Δ^2 method

3.1 Algorithm

For both methods, one first needs to determine a function $x = \phi(x)$ from the given equation such that the derivative of $\phi(x)$ is always less than 1 in the interval where the root lies. This ensures convergence. For $f(x)$, it was determined that $\phi(x) = 1/\sqrt{1+x}$. For $g(x)$, it was determined that $\phi(x) = e^{-x}$.

In iteration method, we use the following relation:

$$x_{n+1} = \phi(x_n)$$

we successively find the values one after another in a loop. In my code, I have executed the loop for 19 times to ensure an accurate result, after which it prints the value of the root. During the loop, if the value of $f(x_n) = 0$ for some x, then the loop breaks and the value of x_n is printed.

In Aitken's Δ^2 method, we use the same $\phi(x)$ we determined for a function. Given an initial guess x_k, we determine $x_{k+1} = \phi(x_k)$ and $x_{k+2} = \phi(x_{k+1})$. From there, we determine a new variable x_k^* using the relation:

$$x_k^* = x_k - \frac{(x_{k+1} - x_k)^2}{x_{k+2} - 2x_{k+1} + x_k}$$

We then assign x_k^* as the new x_k and continue the same process described above in a loop. The loop breaks when the condition $|x_{k+2} - 2x_{k+1} + x_k| > 10^{-4}$ is no longer satisfied. This happens when the we have converged to the actual root and the second order difference approaches zero. We print out the last value. This is our approximate root.

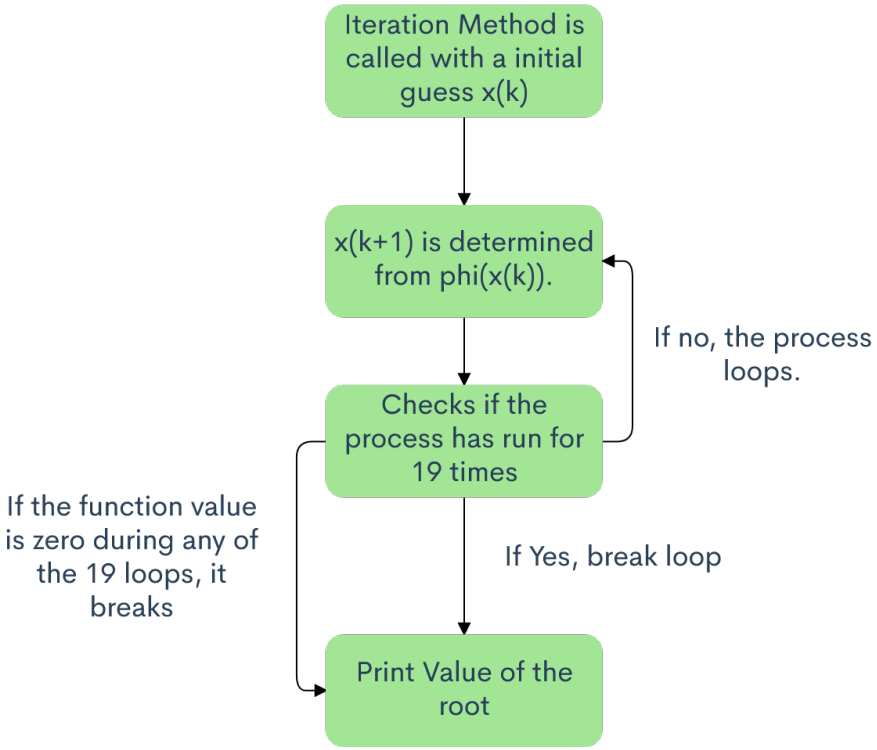

Figure 8: Flowchart of Iteration method code

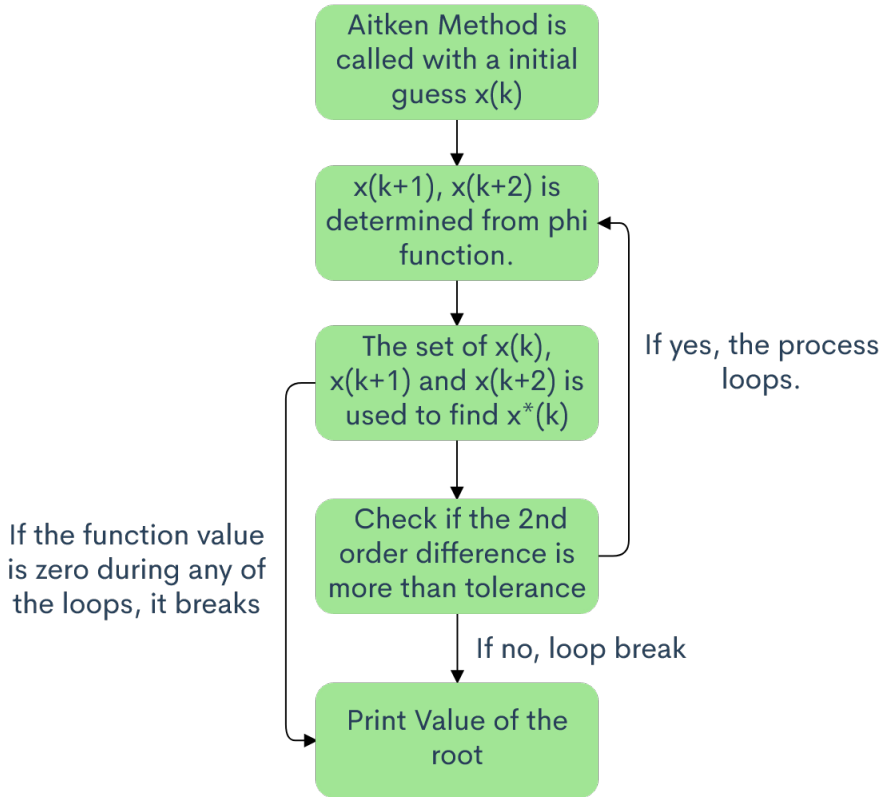

Figure 9: Flowchart of Aitken method code

3.2 Code and Output

```
File Edit Options Buffers Tools Python Virtual Envs Elpy Flymake YASnippet Help
import numpy as np
def f(x):
  return x**3 + x**2 - 1
def p(x):
  return 1/(1+x)**0.5
def iteration(x):
  i=1
  while i < 20:
    y = p(x)
    x = y
    i=i+1
    if f(x)==0:
      break
    else:
      continue
  print ("Root of x**3 + x**2 - 1 by iteration method is {}".format(round(y,4)) )
def aitken(x):
  a=0
  b=0
  c=5
  while abs(c - 2*b + a)>0.0001:
    a=x
    b=p(x)
    c=p(b)
    d= a - ((b - a)**2)/(c - 2*b + a)
    x = d
    if f(x)==0:
      break
    else:
      continue
  print ("Root of x**3 + x**2 - 1 by aitken's method is {}".format(round(d,4)))
```

Figure 10: Iteration method and Aitken method for $f(x) = x^3 + x^2 - 1 = 0$

```
def f1(x):
  return x - np.exp(-x)
def p1(x):
  return np.exp(-x)
def iteration1(x):
  i=1
  while i < 20:
    y = p1(x)
    x = y
    i=i+1
    if f1(x)==0:
      break
    else:
      continue
  print ("Root of x - e**(-x) by iteration method is {}".format(round(y,4)) )
def aitken1(x):
  a=0
  b=0
  c=5
  while abs(c - 2*b + a)>0.0001:
    a=x
    b=p1(x)
    c=p1(b)
    d= a - ((b - a)**2)/(c - 2*b + a)
    x = d
    if f1(x)==0:
      break
    else:
      continue
  print ("Root of x - e**(-x) by aitken's method is {}".format(round(d,4)))
iteration(2)
aitken(2)
iteration1(2)
aitken1(2)
```

Figure 11: Iteration method and Aitken's Δ^2 method for $f(x) = x - e^{-x} = 0$

```
ayakamlan@DESKTOP-C21G1MH:/mnt/c/Users/Dell/Desktop/CP1/Lab3$ python3 MS18197_3_code3.py
Root of x**3 + x**2 - 1 by iteration method is 0.7549
Root of x**3 + x**2 - 1 by aitken's method is 0.7549
Root of x - e**(-x) by iteration method is 0.5671
Root of x - e**(-x) by aitken's method is 0.5671
ayakamlan@DESKTOP-C21G1MH:/mnt/c/Users/Dell/Desktop/CP1/Lab3$
```

Figure 12: Output