VE485, Optimization in Machine Learning (Summer 2020) Homework: 5. Solving Non-constrained Problem

5. Solving Non-constrained Problem

Lecturer: Xiaolin Huang xiaolinhuang@sjtu.edu.cn

Student: Chongdan pandddda@sjtu.edu.cn

Problem 1

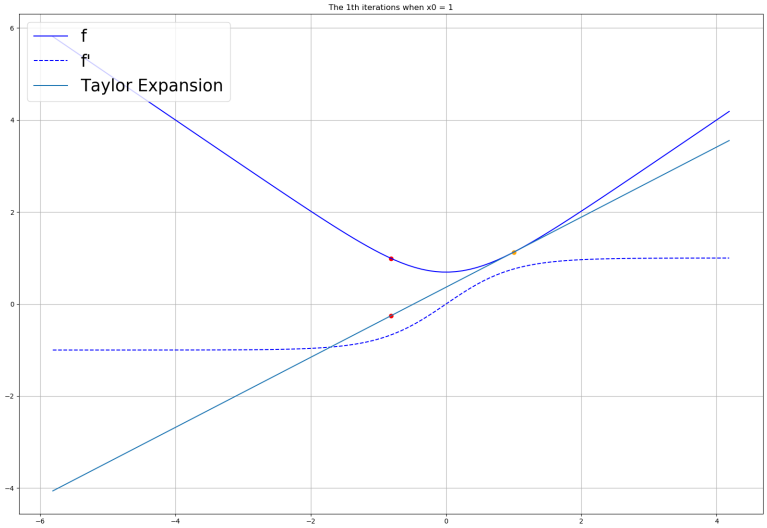
The pure Newton method. Newton's method with fixed step size $t = 1$ can diverge if the initial point is not close to $x^\star$. In this problem we consider two examples.

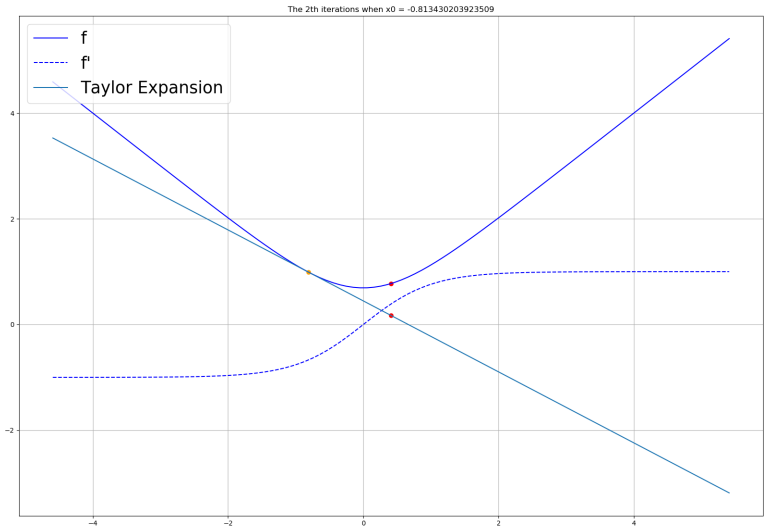
1. $f(x) = \log(e^x + e^{-x})$ has a unique minimizer $x^\star = 0$. Run Newton's method with fixed step size $t = 1$, starting at $x^{(0)} = 1$ and at $x^{(0)} = 1.1$.
2. $f(x) = -\log x + x$ has a unique minimizer $x^\star = 1$. Run Newton's method with fixed step size $t = 1$, starting at $x^{(0)} = 3$.

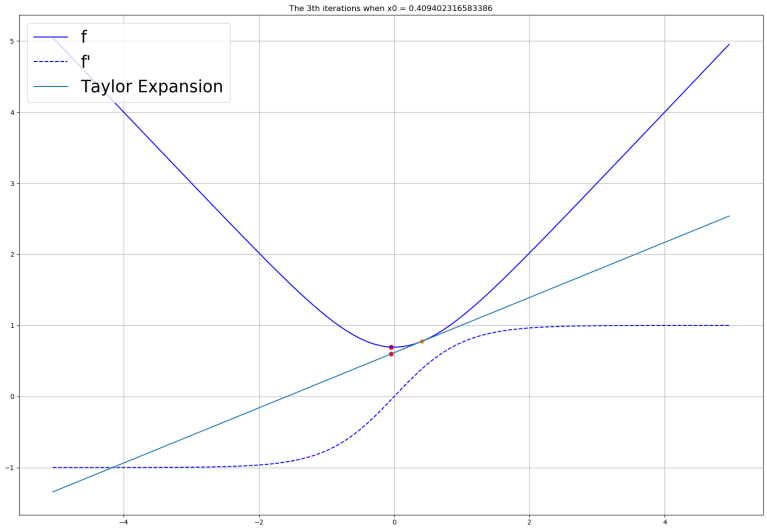
Plot f and f', and show the first few iterates.

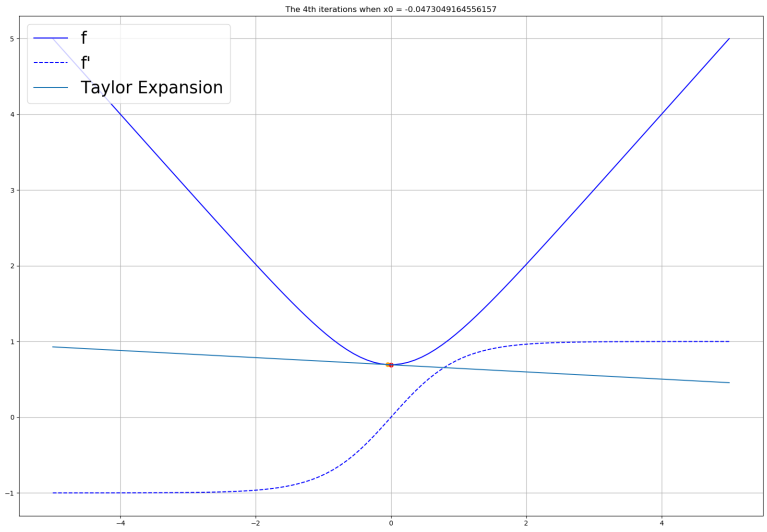
Answer.

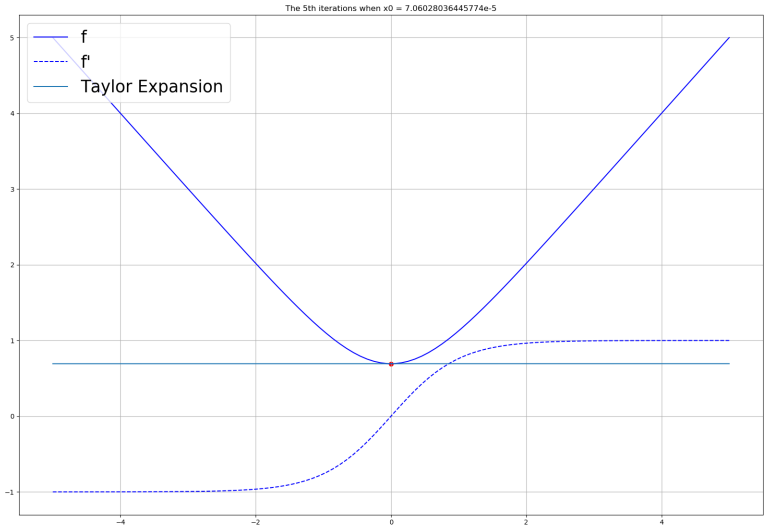
1. When $f(x) = \log(e^x + e^{-x}), x^{(0)} = 1, t = 1$, we can find x is closer to the minimize point after each iteration.



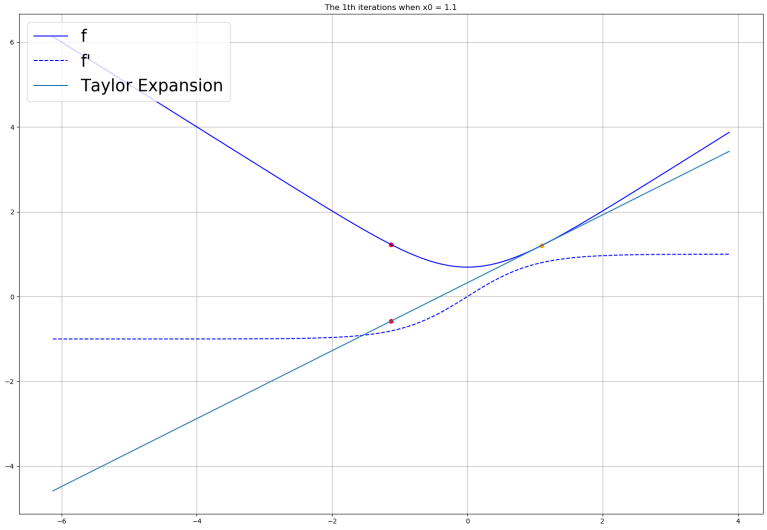

The 2th iterations when x0 = -0.813430203923509
f
f'
Taylor Expansion

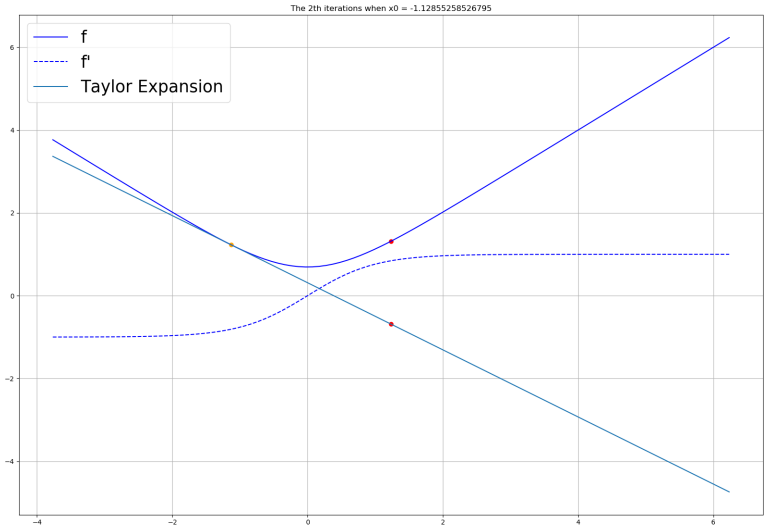

The 3th iterations when x0 = 0.409402316583386
f
f'
Taylor Expansion

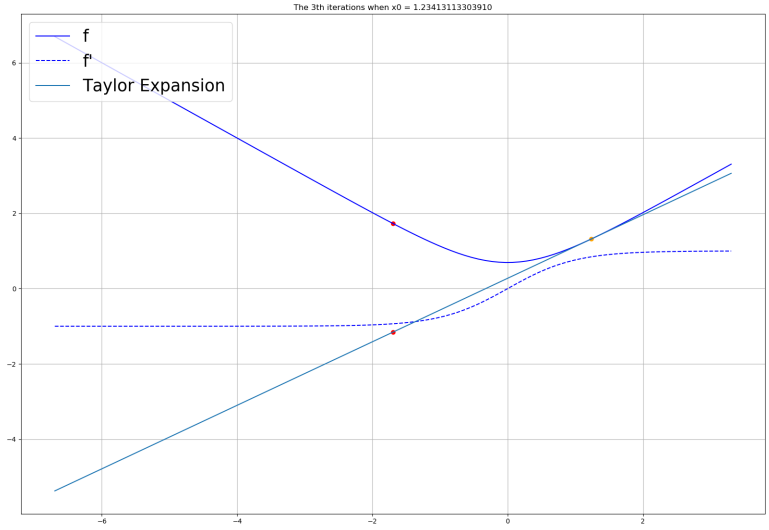


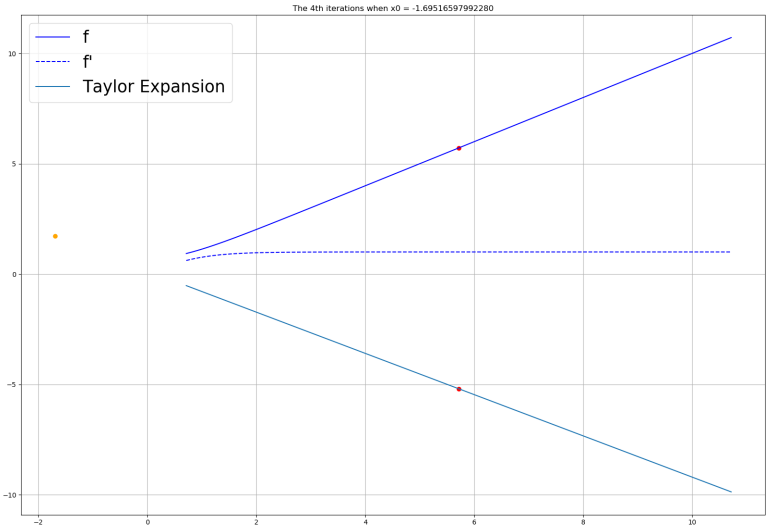


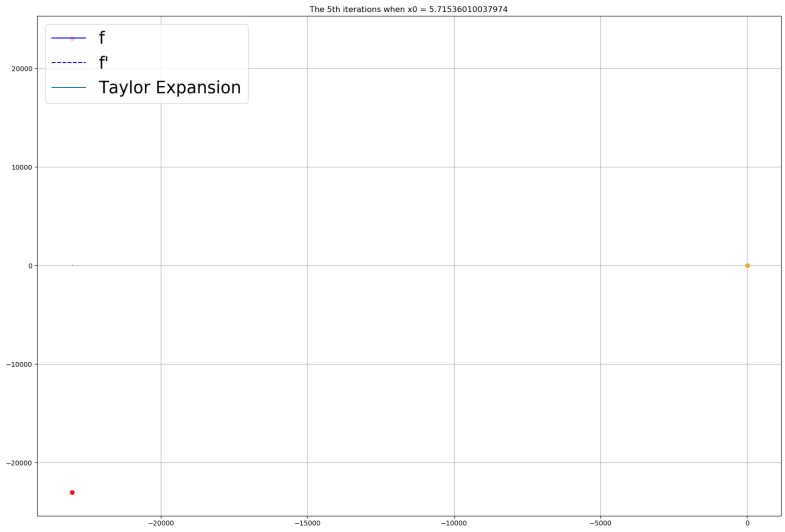
However, when $f(x) = \log(e^x + e^{-x}), x^{(0)} = 1.1, t = 1$, we can find x is farther from the minimize point after each iteration, its absolute value also grows larger.


The 1th iterations when x0 = 1.1
f
f'
Taylor Expansion

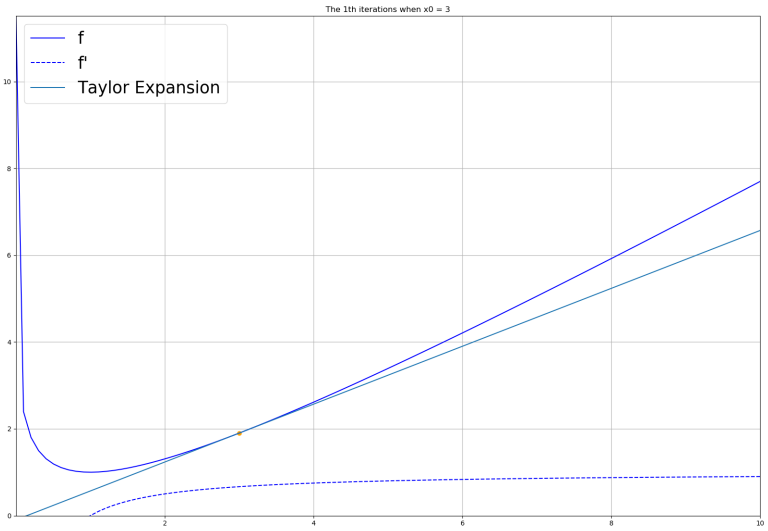

The 2th iterations when x0 = -1.12855258526795
f
f'
Taylor Expansion


The 3th iterations when x0 = 1.23413113303910
f
f'
Taylor Expansion

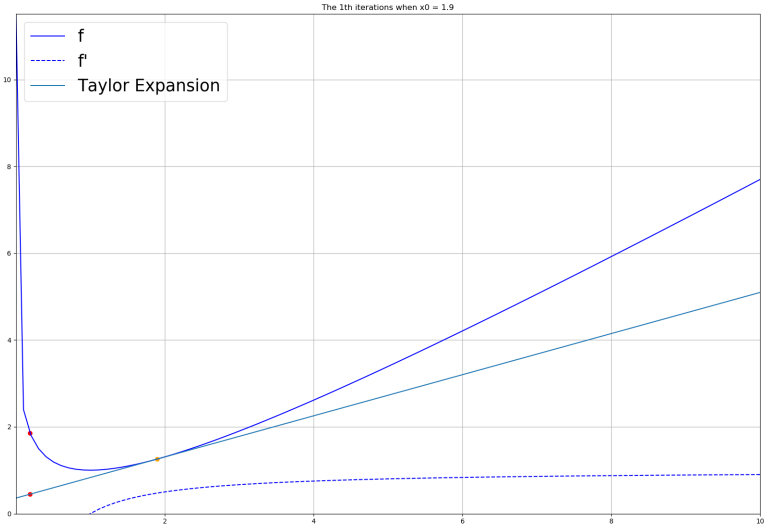

The 4th iterations when x0 = -1.69516597992280
f
f'
Taylor Expansion

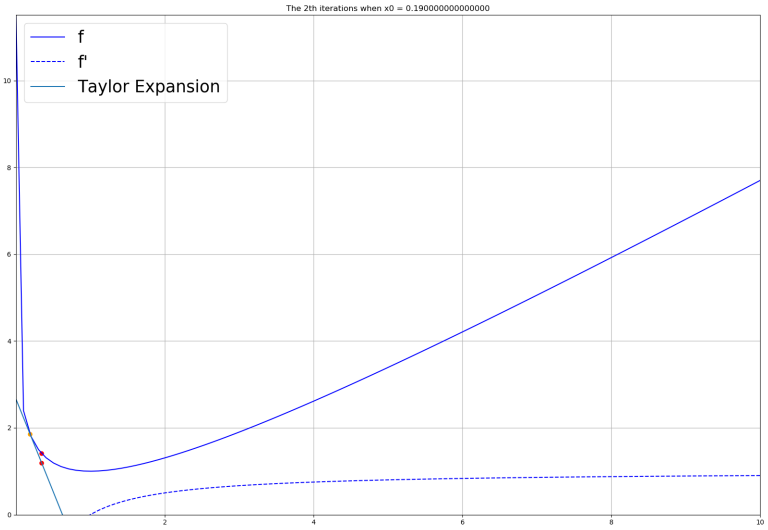


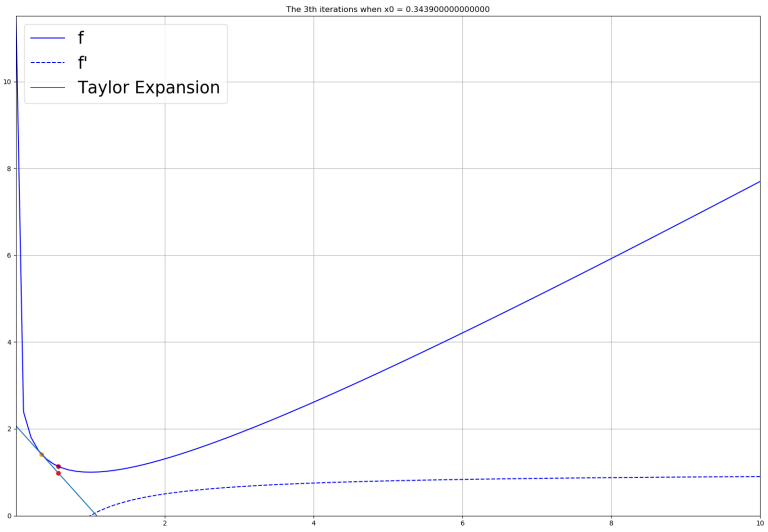
2. When $f(x) = -\log x + x, x^{(0)} = 3, t = 1$, we can find after the first iteration, $x^{(1)} = 3 - \frac{1-1/3}{1/9} = -3$, which is not in $f(x)$ **dom**, so the process has to be interrupted.

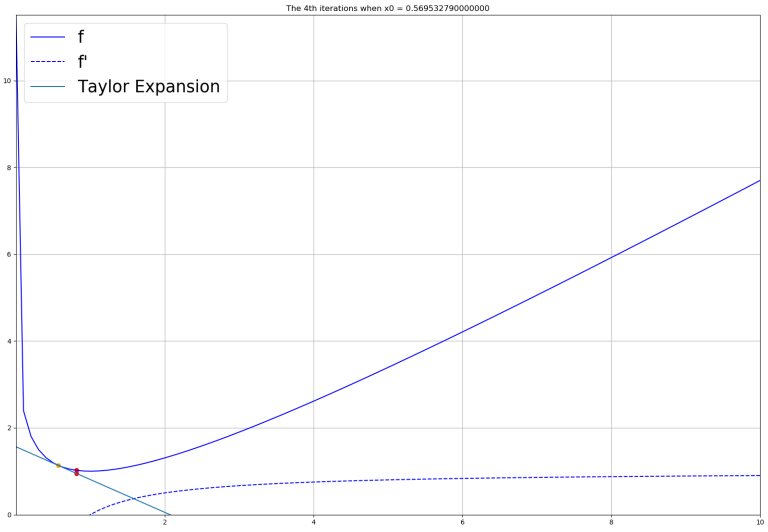


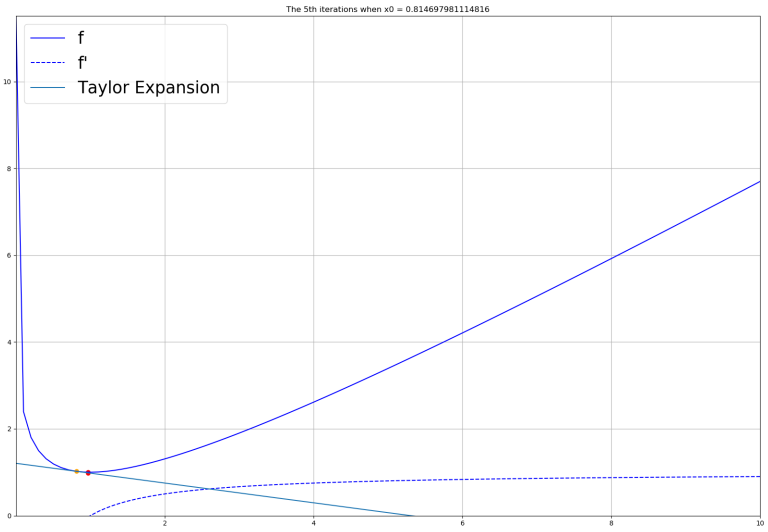
However, if I change $x^{(0)}$'s value to 1.9, the ***Newton Method*** still converges to the minimize point $x = 1$

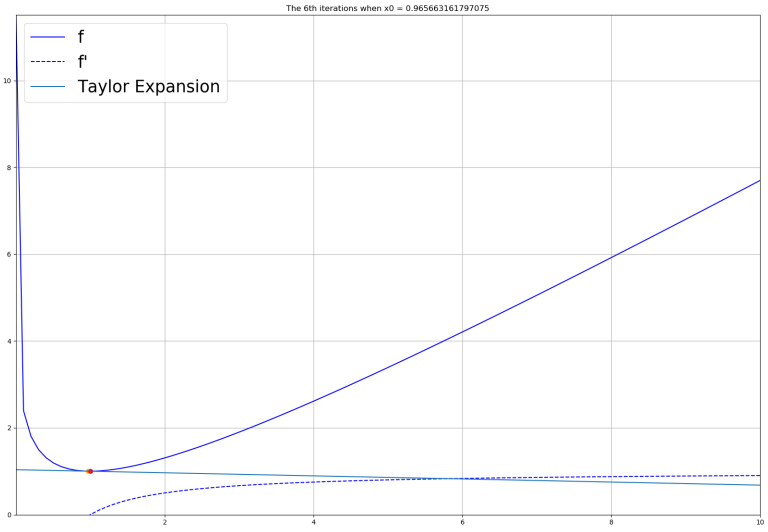

The 1th iterations when x0 = 1.9
f
f'
Taylor Expansion
0
2
4
6
8
10

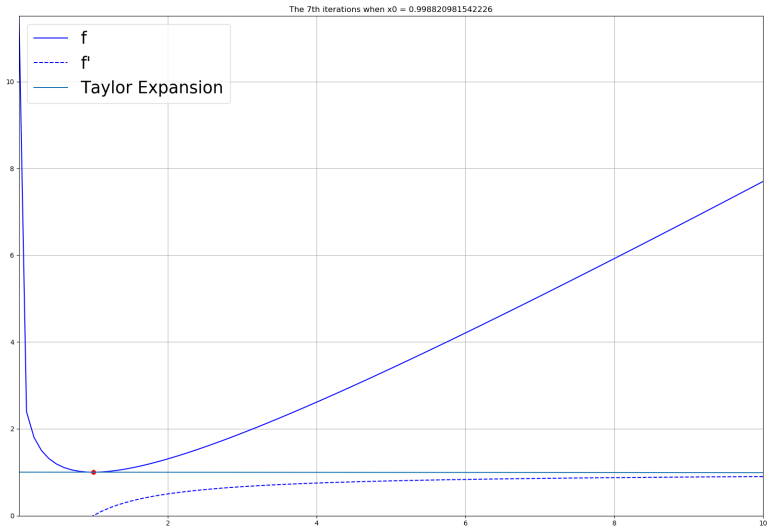

The 2th iterations when x0 = 0.190000000000000
f
f'
Taylor Expansion
0
2
4
6
8
10


The 3th iterations when x0 = 0.343900000000000
f
f'
Taylor Expansion
0
2
4
6
8
10


The 4th iterations when x0 = 0.569532790000000
f
f'
Taylor Expansion
0
2
4
6
8
10


The 5th iterations when x0 = 0.814697981114816
f
f'
Taylor Expansion
0
2
4
6
8
10


The 6th iterations when x0 = 0.965663161797075
f
f'
Taylor Expansion
0
2
4
6
8
10



Therefore, according to my observation, for the ***Pure Newton Method*** where t is constant, the closer x to the minimize point, the more possible it's going to converge. In our two cases, the converge distance both happen to be 1.

P1 Python Code

```
from sympy import *
import numpy as np
import matplotlib.pyplot as plt
x = Symbol('x')
fx = log(exp(x)+exp(-x))
# fx = -log(x) + x
t = 1
x0 = 3
fdx = diff(fx,x)
fddx = diff(fdx,x)
e = 1e-5
k=1
while 1 :

    tx = fx.subs(x,x0)+fdx.subs(x,x0)*(x-x0)+fddx.subs(x,x0)*(x0-x0)**2/2
    x_nt = -fdx.evalf(subs = {'x':x0})/fddx.evalf(subs = {'x':x0})
    l=(x_nt**2)**0.5
    plt.figure(figsize=(20,20))
    plt.grid(1)
    # draw the points
    plt.scatter(x0,fx.evalf(subs = {'x':x0}),color='orange')
    title='The ' + str(k) + 'th iterations when x0 = ' + str(x0)
```

```
    x0 = x0+t*x_nt
    plt.scatter(x0,fx.evalf(subs = {'x':x0}),color='red')
    plt.scatter(x0,tx.evalf(subs = {'x':x0}),color='red')
    # draw the graph of functions
    n = np.linspace(float(x0)-5,float(x0)+5,100)
    Tx=[]
    Fx=[]
    Fdx=[]
    for i in range(len(n)) :
        Tx.append(tx.subs(x,n[i]))
        Fx.append(fx.subs(x,n[i]))
        Fdx.append(fdx.subs(x,n[i]))
    plt.plot(n, Fx, color='blue', label='f')
    plt.plot(n, Fdx, color='blue', label="f'", linestyle='--')
    plt.plot(n, Tx, label="Taylor Expansion")
    plt.legend(loc='upper left', fontsize=25)
    print(n)
    plt.title(title)
    plt.show()

    if l**2/2 <= e:
        break
    k=k+1
```

P2 Python Code

```
from sympy import *
import numpy as np
import matplotlib.pyplot as plt
x = Symbol('x')
# fx = log(exp(x)+exp(-x))
fx = -log(x) + x
t = 1
x0 = 3
fdx = diff(fx,x)
fddx = diff(fdx,x)
e = 1e-5
k=1
while 1 :

    tx = fx.subs(x,x0)+fdx.subs(x,x0)*(x-x0)+fddx.subs(x,x0)*(x0-x0)**2/2
    x_nt = -fdx.evalf(subs = {'x':x0})/fddx.evalf(subs = {'x':x0})
    l=(x_nt**2)**0.5
    plt.figure(figsize=(20,20))
    plt.grid(1)
    # draw the points
    plt.scatter(x0,fx.evalf(subs = {'x':x0}),color='orange')
    title='The ' + str(k) + 'th iterations when x0 = ' + str(x0)
```

```
x0 = x0+t*x_nt
plt.scatter(x0,fx.evalf(subs = {'x':x0}),color='red')
plt.scatter(x0,tx.evalf(subs = {'x':x0}),color='red')
# draw the graph of functions
left=float(x0)-5
right=float(x0)+5
if left<=0:
    left=1e-5
    right=left+10
n = np.linspace(left,right,100)
Tx=[]
Fx=[]
Fdx=[]
for i in range(len(n)) :
    Tx.append(tx.subs(x,n[i]))
    Fx.append(fx.subs(x,n[i]))
    Fdx.append(fdx.subs(x,n[i]))
plt.plot(n, Fx, color='blue', label='f')
plt.plot(n, Fdx, color='blue', label="f'", linestyle='--')
plt.plot(n, Tx, label="Taylor Expansion")
plt.legend(loc='upper left', fontsize=25)
plt.axis([left,right,0,float(max(Fx))])
plt.title(title)
plt.show()
if l**2/2 <= e:
    break
k=k+1
```