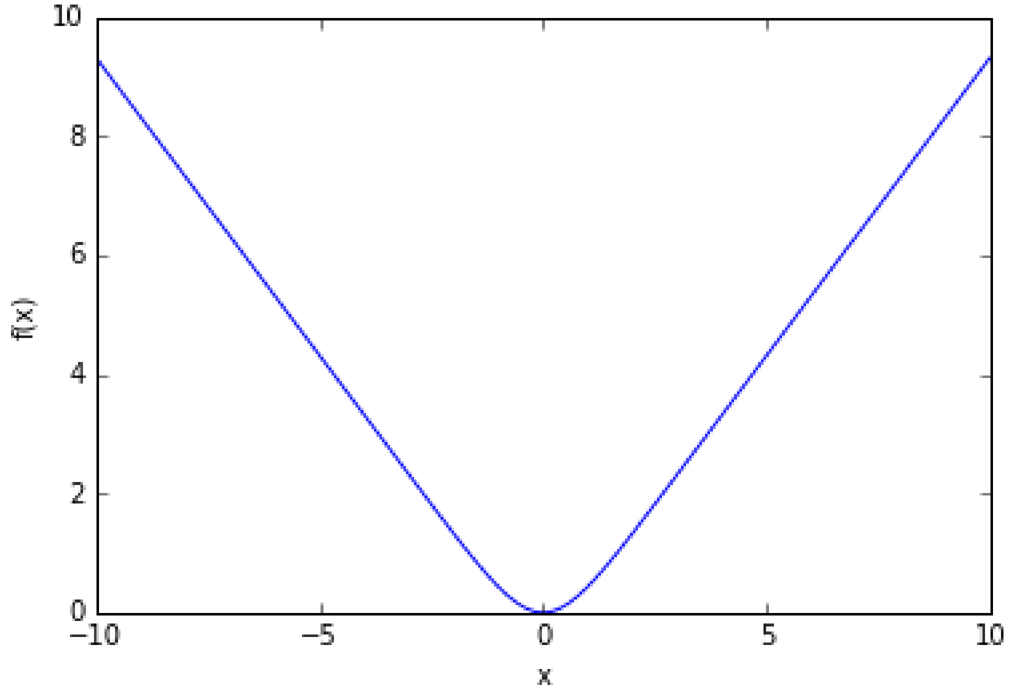
6.2 d) & e)

iteration	number of mistakes	M log-likelihood L
0	195	-1.044559748
1	60	-0.50494051
2	43	-0.410763774
4	42	-0.365127174
8	44	-0.347663212
16	40	-0.334676667
32	37	-0.322592689
64	37	-0.314831062
128	36	-0.311155817
256	36	-0.310161104

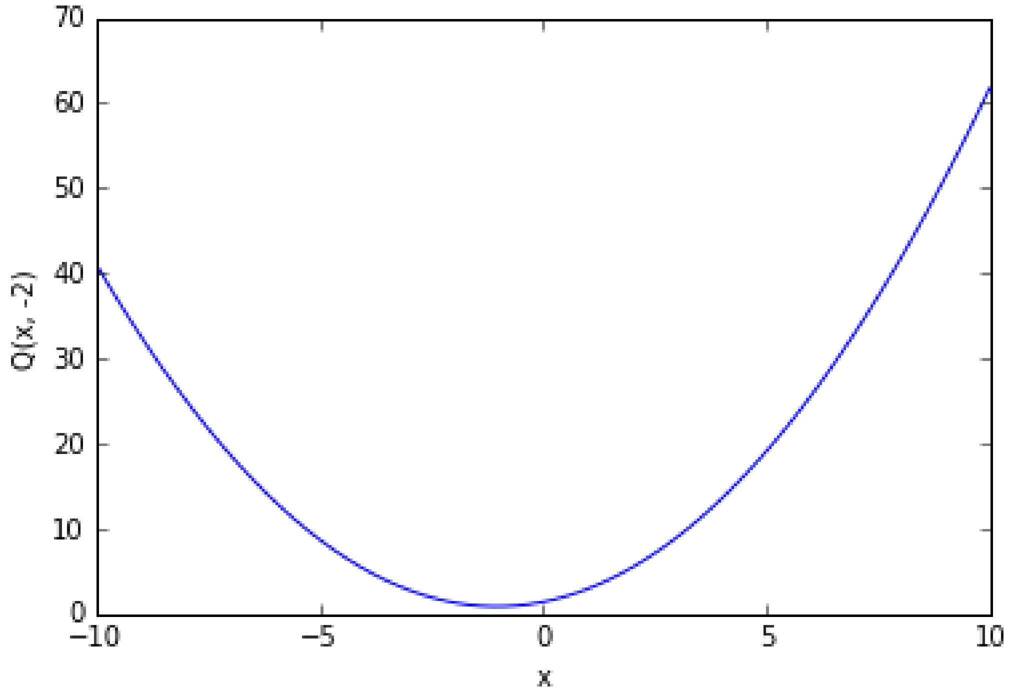
```
import numpy as np

OX = np.loadtxt('spectX.txt', dtype=int)
OY = np.loadtxt('spectY.txt', dtype=int)

PZ_X = np.zeros(OX.shape[1], dtype=int)
PZ_X = PZ_X + (1/len(PZ_X))

#P(Y|X)
def getPY_X(PZ_X, X, Y):
    result = 1.0
    for i in range(len(PZ_X)):
        result = result * (np.power((1-PZ_X[i]), X[i]))
    if(Y==1):
        result = 1-result
    return result

def LLhood(OX, OY, PZ_X):
    llhood = 0.0
    for i in range (len(OX)):
        llhood = llhood + np.log(getPY_X(PZ_X, OX[i], OY[i]))
    llhood = llhood/len(OX)
    return llhood

def countMistakes(PY_X):
    return np.count_nonzero(PY_X<0.5)

def run(OX,OY,PZ_X):
    #PZX_XY E-step
    for loop in range(257):
        PY_X = np.array([getPY_X(PZ_X, OX[i],OY[i]) for i in
range(len(OX))])
        a = (OX.T/PY_X).T * PZ_X
        PnewZ_X = np.inner(OY, a.T)
        PnewZ_X = np.array([PnewZ_X[i]/np.count_nonzero(OX.T[i]) for
i in range(len(PnewZ_X))])
        if (loop & loop-1) == 0:
            print (loop, countMistakes(PY_X), LLhood(OX, OY, PZ_X))
        PZ_X = PnewZ_X #M-step
```

6.3 c)

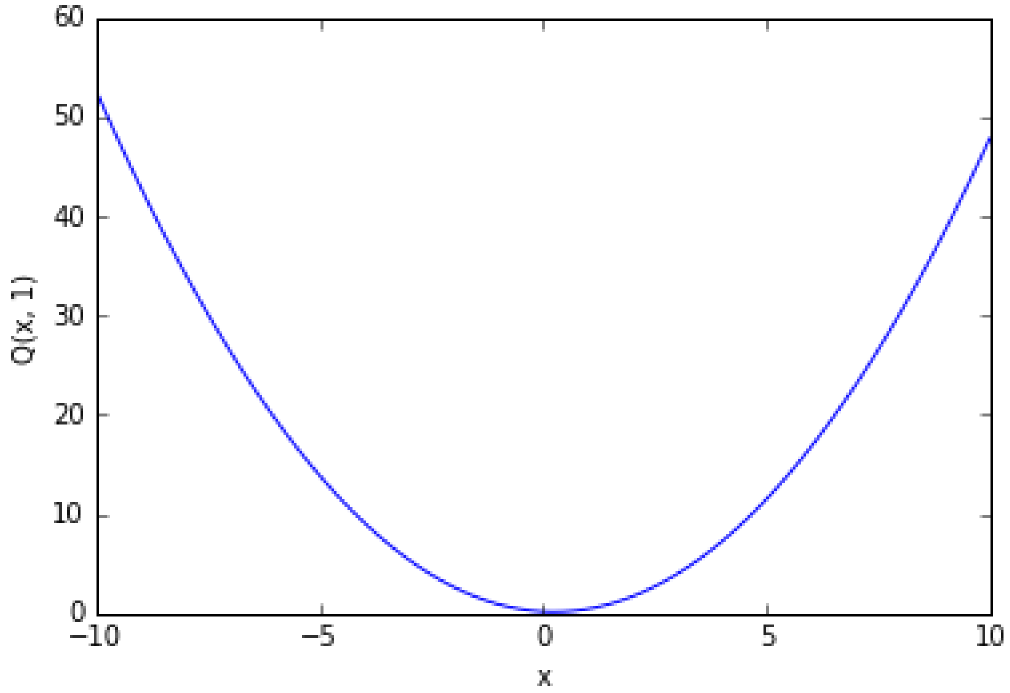
```
>>> plot(-10,10, 0.001, f)
```

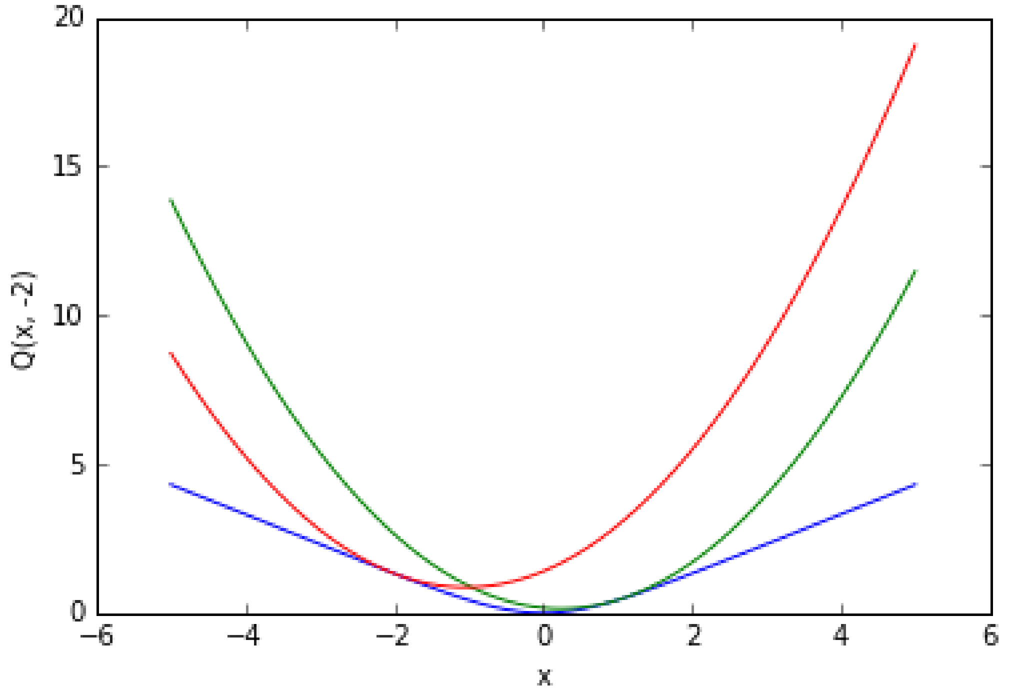



```
>>> plot(-10,10, 0.001, Q, -2)
```



```
>>> plot(-10,10, 0.001, Q, 1)
```





6.3 f)

```
>>> ConvergerAux(-2, 10, f, fd)
```

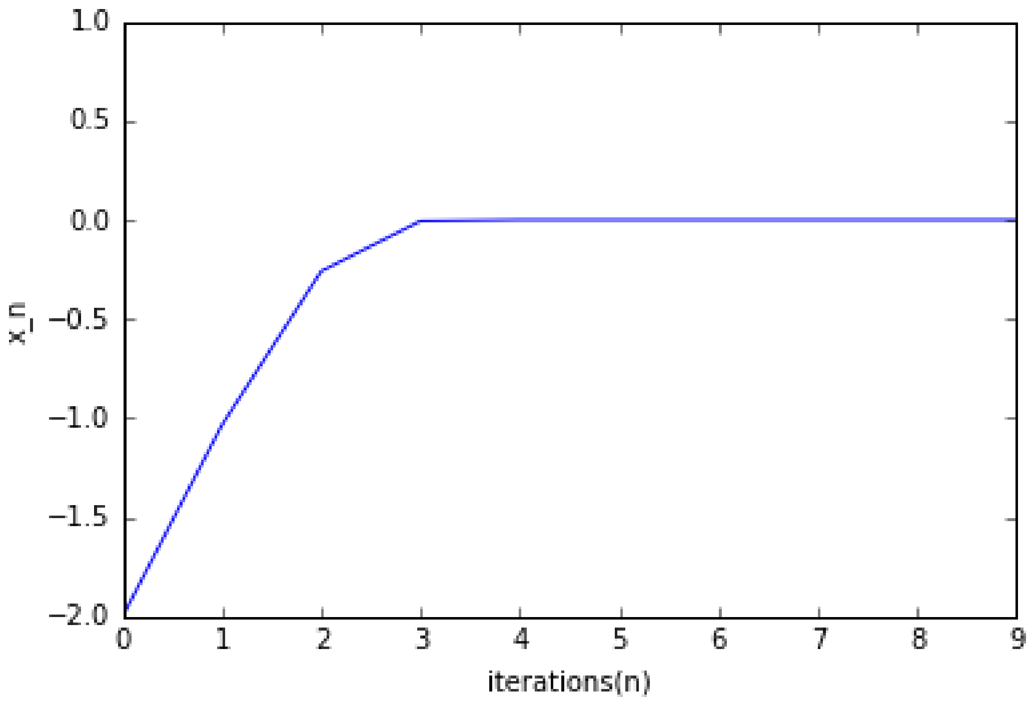


Table showing values of x_n after every update, with respect to minimizing Q(x, y). It converges to 0 after 5 iterations. We start at x_0 = -2.

iteration	x_n
0	-2
1	-1.03597
2	-0.25968
3	-0.00568
4	-6.12E-08
5	-7.94E-23
6	0
7	0
8	0
9	0

```
>>> ConvergerAux(1, 10, f, fd)
```

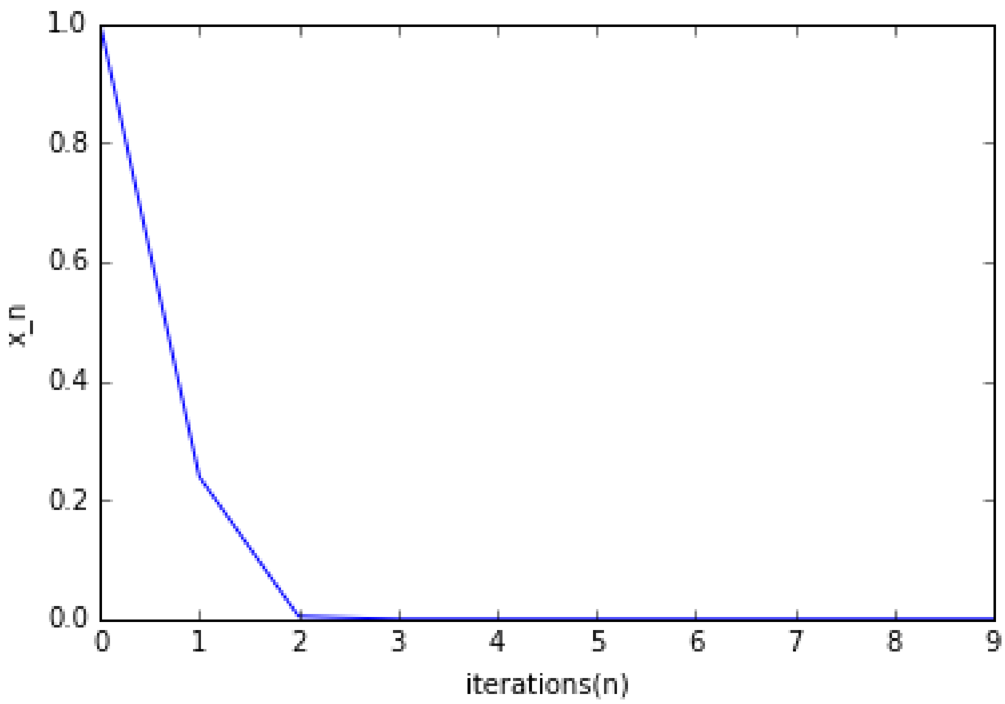


Table showing values of x_n after every update, with respect to minimizing Q(x, y), It converges to zero after 5 iterations. We start at x_0 = 1.

iteration	x_n
0	1
1	0.238406
2	0.004416
3	2.87E-08
4	6.62E-24
5	0.00E+00
6	0
7	0
8	0
9	0

6.3 g)

```
>>> ConvergerNewton(1, 10)
```

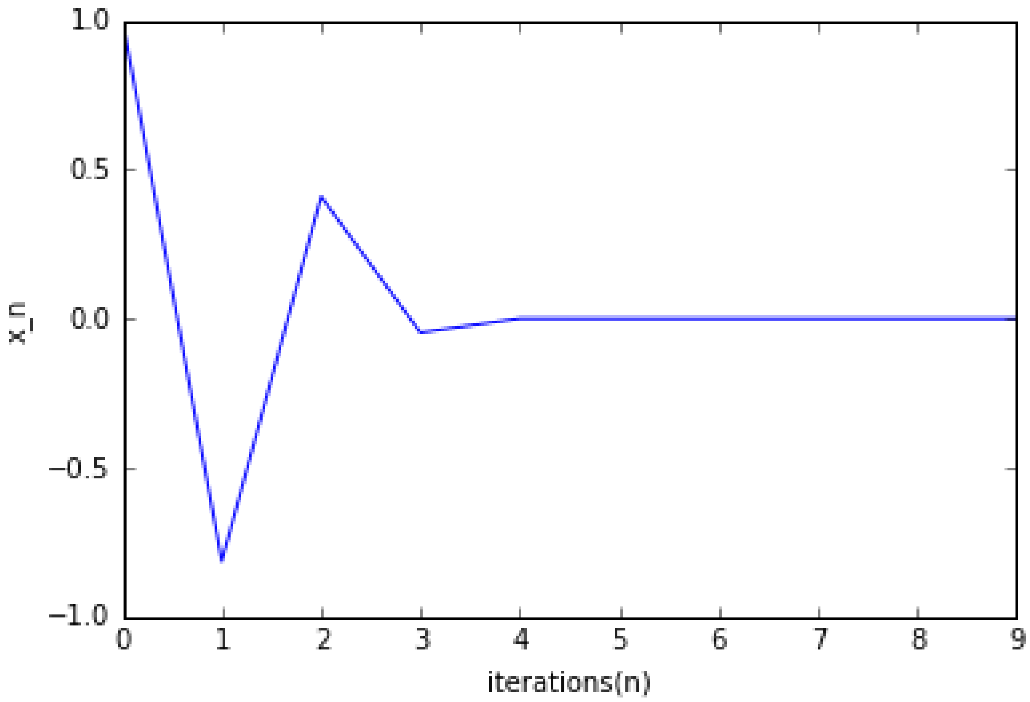
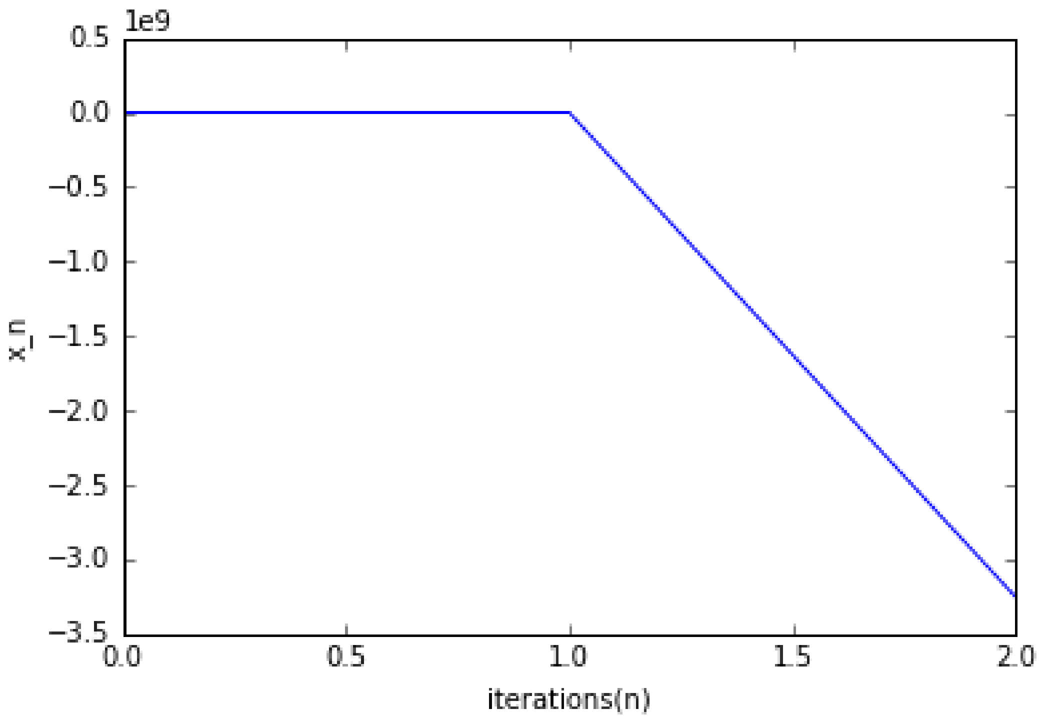


Table showing values of x_n after every update, with respect to minimizing f(x) using Newton method with x_0 = 1. It converges to zero after 5 iterations.

iteration	x_n
0	1
1	-0.81343
2	0.409402
3	-4.73E-02
4	7.06E-05
5	-2.35E-13
6	0
7	0
8	0
9	0

```
>>> ConvergerNewton(-2, 10)
```



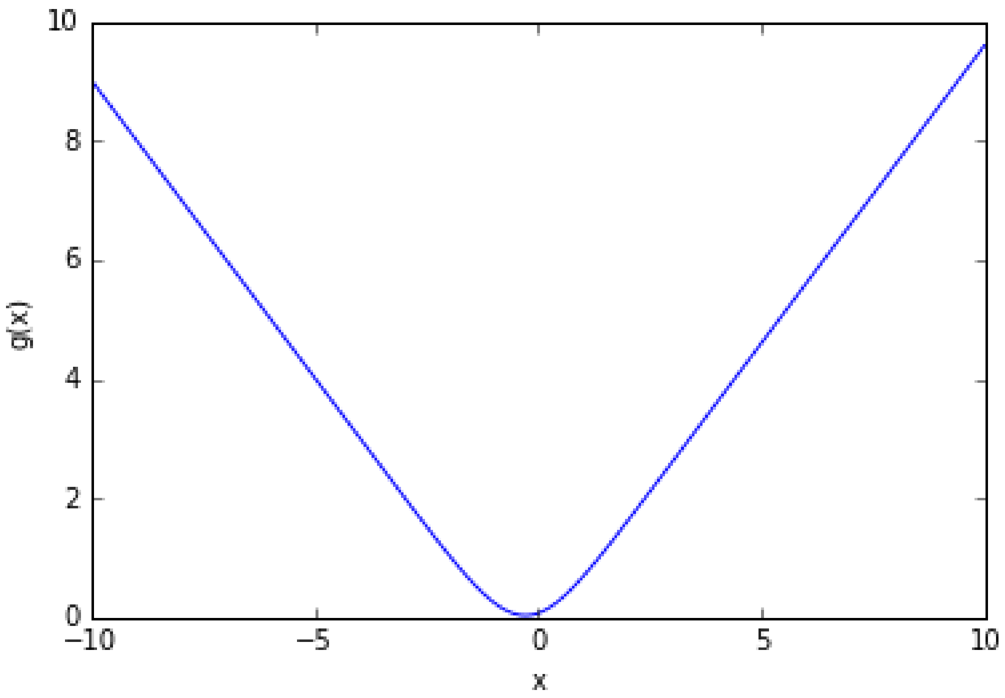
After 2 iteration values goes out of bound. So not shown on graph. (not converging)

Table showing values of x_n after every update, with respect to minimizing f(x) using Newton method with x_0 = -2. Values are not defined after 3 iteration as it never converges.

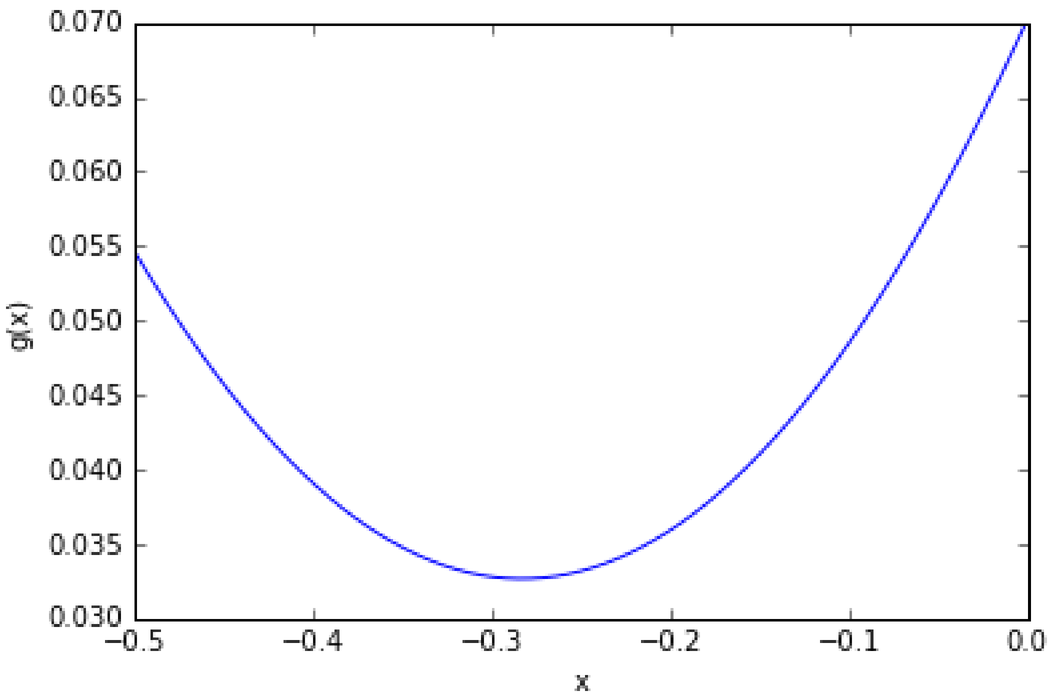
iteration	x_n
0	-2
1	11.64496
2	-3.3E+09
3	inf
4	nan
5	nan
6	nan
7	nan
8	nan
9	nan

6.3 h)

```
>>> plot(-10,10, 0.001, g)
```



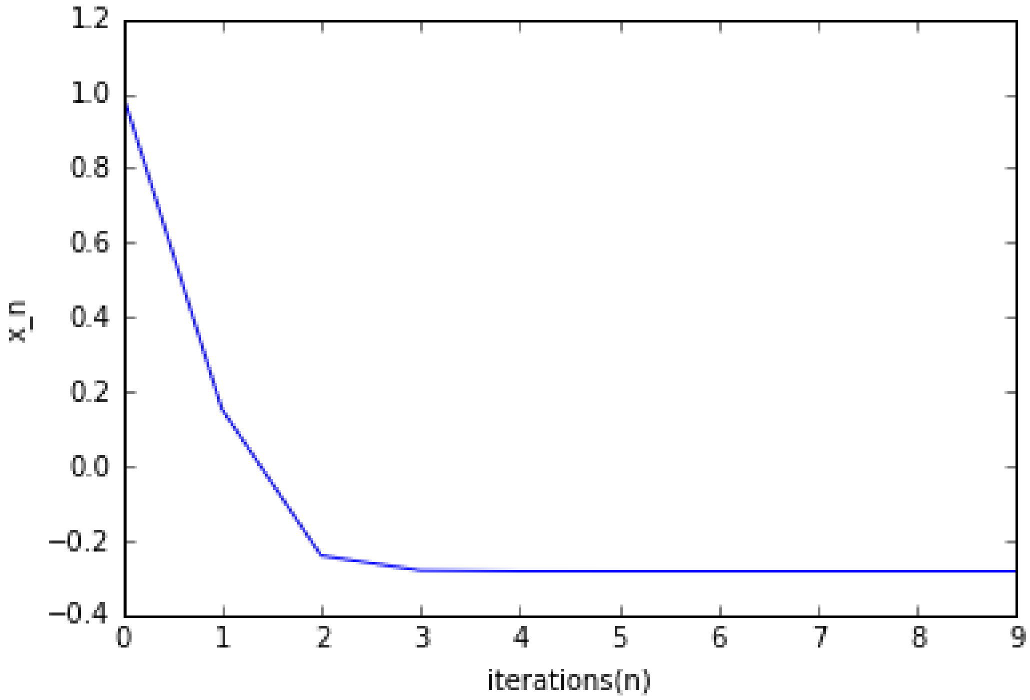

```
>>> plot(-0.5,0, 0.001, g)
```



A closer look to g(x)

6.3 k)

```
>>> ConvergerAux(1, 10, g, gd)
```



$$x^* = -0.283027022745$$

$$g_{min} = g(x^*) = 0.0326571259075$$

Table showing values of x_n after every iteration with respect to minimizing R(x, y), with x_0 =1.

iteration	x_n
0	1
1	0.152599478
2	-0.240966615
3	-0.280717926
4	-0.28289933
5	-0.283019955
6	-0.283026632
7	-0.283027001
8	-0.283027022
9	-0.283027023

```python
import numpy as np
import matplotlib.pyplot as plt

def g(x):
    result = 0.0
    for k in range(10):
        result += np.log(np.math.cosh(x+ (1/(k+1))))
    return result/10;

def gd(x):
    result=0.0
    for k in range(10):
        result += np.tanh(x + (1/(k+1)))
    return result/10;

def f(x):
    return np.log(np.cosh(x));

def fd(x):
    return np.tanh(x);

def fdd(x):
    return 1/(np.cosh(x) * np.cosh(x))

def Q(x,y):
    return f(y)+fd(y)*(x-y) + np.power((x-y),2)/2

def ConvergerAux(x0, loops, func, funcd):
    x = x0;
    it = np.zeros(loops, dtype=int)
    xVal = np.zeros(loops)
    for i in range(loops):
        xVal[i]= x
        it[i] = i
        print (it[i], x)
        x = x - funcd(x)
    plt.plot(it, xVal)
    plt.xlabel("iterations(n)")
    plt.ylabel("x_n")
    print (x, func(x))

def ConvergerNewton(x0, loops):
    x = x0;
    it = np.zeros(loops, dtype=int)
    xVal = np.zeros(loops)
    for i in range(loops):
        xVal[i]= x
        it[i] = i
        print (it[i], x)
        x = x - (fd(x)/fdd(x))
    plt.plot(it, xVal)
    plt.xlabel("iterations(n)")
    plt.ylabel("x_n")
    print (x, f(x))
```

```python
def plot(l,r,step,func, y0=0):
    xRange = np.arange(l,r,step)
    if func==Q:
        y = np.array([func(x,y0) for x in xRange])
    else:
        y = np.array([func(x) for x in xRange])
    plt.plot(xRange,y)
    plt.xlabel("x")
    ylabel = func.__name__ + '(x'
    if y0!=0: ylabel+= (', '+str(y0))
    ylabel += ')'
    plt.ylabel(ylabel)
```