CSE250A_HW6.4

November 14, 2017

1 6.4 Auxiliary function

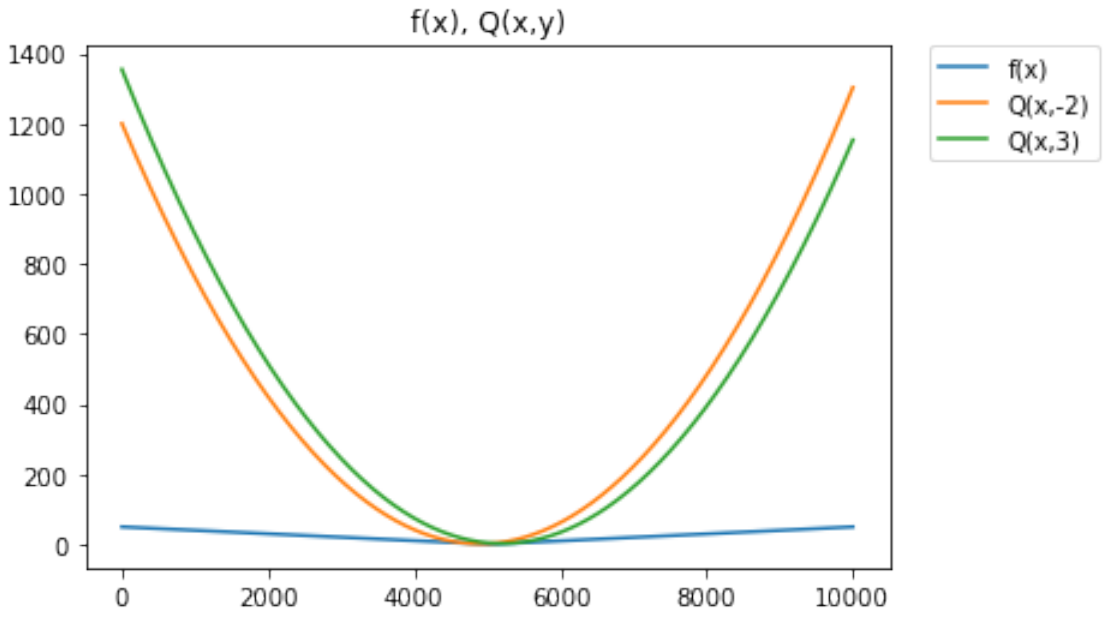
In [15]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

In [43]:
```
''' functions '''

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

def df(x):
    return(np.tanh(x))

def df2(x):
    return((1/np.cosh(x))**2)

def Q(x,y):
    out = f(y) + df(y)*(x-y) + 0.5*(x-y)**2
    return(out)
```

1.1 6.4c - plotting f(x), Q(x,y)

In [42]:
```
x_arr = np.arange(-50-0.01, 50+0.01, 0.01)

plt.subplot(111)
plt.plot(f(x_arr), label="f(x)")
plt.plot(Q(x_arr,-2), label="Q(x,-2)")
plt.plot(Q(x_arr,3), label="Q(x,3)")
plt.title('f(x), Q(x,y)')
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
```

Out[42]:
```
<matplotlib.legend.Legend at 0x7f2f517f6e80>
```

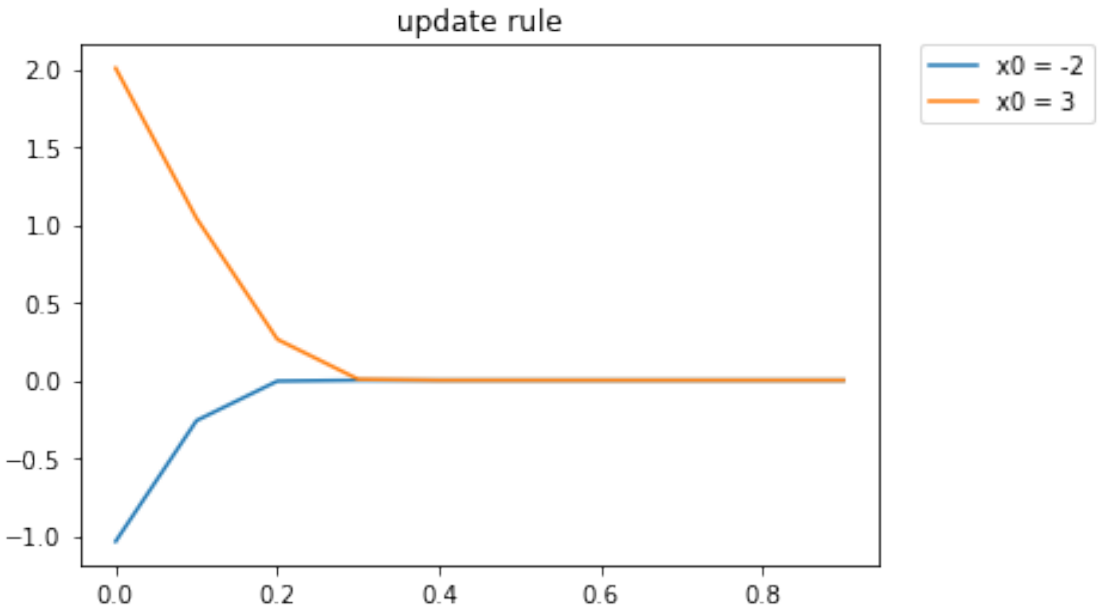


2 6.4f - update rule

```
In [47]: def update(x):
             return(x-np.tanh(x))

         x01=-2
         xn_arr1=[]
         n=np.arange(0,1,0.1)
         for i in range(len(n)):
             if i == 0:
                 xn_arr1.append(update(x01))
             else:
                 xn_arr1.append(update(xn_arr1[i-1]))

         x02=3
         xn_arr2=[]
         for i in range(len(n)):
             if i == 0:
                 xn_arr2.append(update(x02))
             else:
                 xn_arr2.append(update(xn_arr2[i-1]))

         plt.subplot(111)
         plt.plot(n, xn_arr1, label="x0 = -2")
         plt.plot(n, xn_arr2, label="x0 = 3")
```

```
        plt.title('update rule')
        plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
```

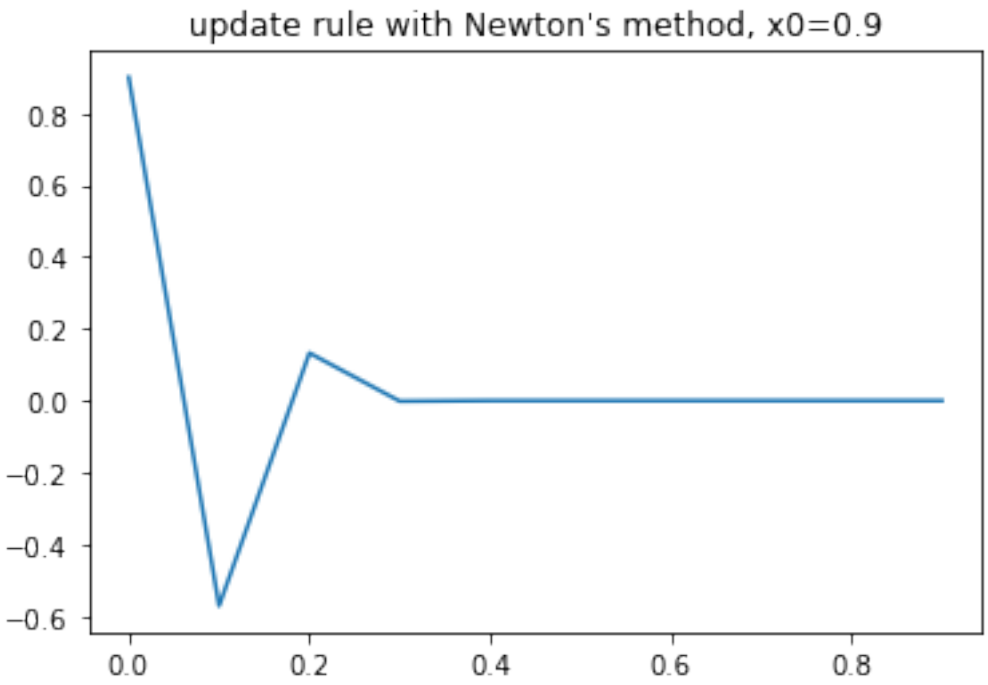
```
Out[47]: <matplotlib.legend.Legend at 0x7f2f51b24208>
```



3 6.4g - update rule with Newton's method

```
In [83]: def update_newton(xn):
             return(xn-np.sinh(xn)*np.cosh(xn))

         # converging example
         x1_arr=[]
         n_arr = np.arange(0,1,0.1)
         x01 = 0.9
         for i in range(len(n_arr)):
             if i == 0:
                 x1_arr.append(x01)
             else:
                 x1_arr.append(update_newton(x1_arr[i-1]))

         plt.plot(n_arr, x1_arr)
         plt.title("update rule with Newton's method, x0=0.9")
```

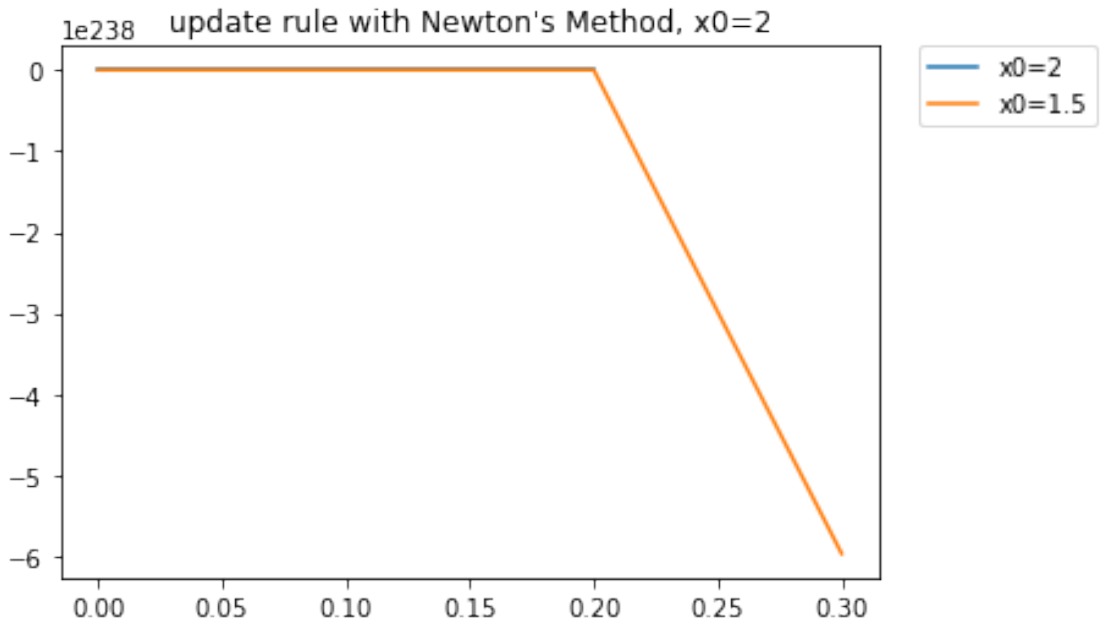
```
Out[83]: <matplotlib.text.Text at 0x7f2f50e1d080>
```



In [85]:
```
# non-converging example
x2_arr=[]
x02 = 2
for i in range(len(n_arr)):
    if i == 0:
        x2_arr.append(x02)
    else:
        x2_arr.append(update_newton(x2_arr[i-1]))

plt.plot(n_arr,x2_arr,label='x0=2')
plt.title("update rule with Newton's Method, x0=2")
```

```
/home/jlzhou/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:2: RuntimeWarning: over

/home/jlzhou/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:2: RuntimeWarning: over

/home/jlzhou/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:2: RuntimeWarning: inva
```

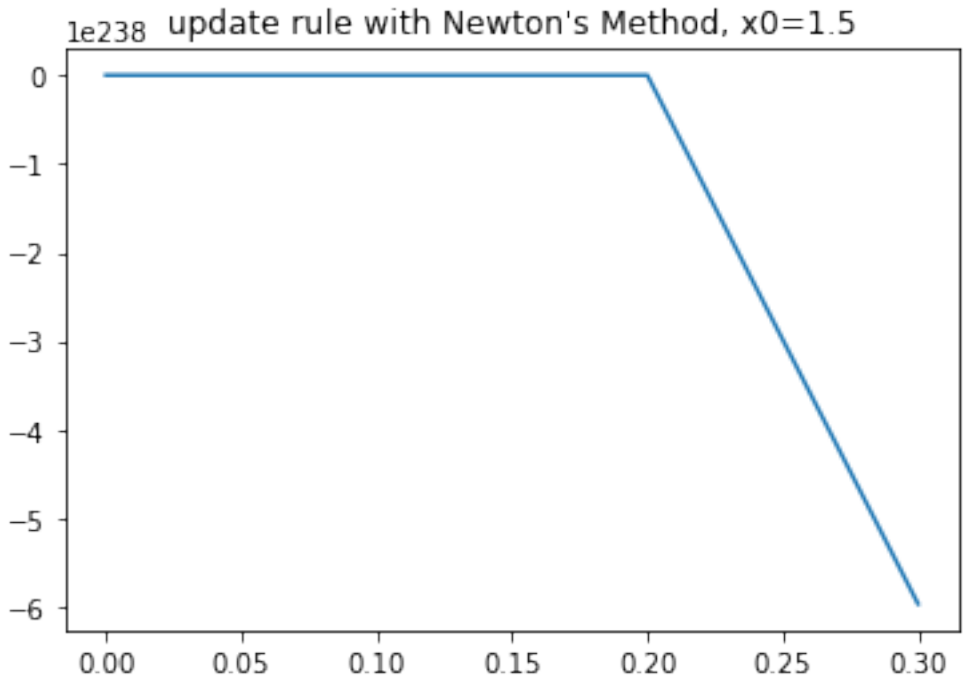
Out[85]: <matplotlib.legend.Legend at 0x7f2f50c8ef98>



In [87]:
```
# non-converging example
x3_arr=[]
x03 = 1.5
for i in range(len(n_arr)):
    if i == 0:
        x3_arr.append(x03)
    else:
        x3_arr.append(update_newton(x3_arr[i-1]))

plt.plot(n_arr,x3_arr)
plt.title("update rule with Newton's Method, x0=1.5")
```

```
/home/jlzhou/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:2: RuntimeWarning: over

/home/jlzhou/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:2: RuntimeWarning: over

/home/jlzhou/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:2: RuntimeWarning: inva
```

Out[87]: <matplotlib.text.Text at 0x7f2f510482b0>



3.0.1 The update rule using Newton's method does not always converge, because when |x1|>=|x0|, the value of x_n becomes very large or very small.

```
In [75]: def test_newton(x0):
             xn_arr = []
             xn = x0
             for i in range(20):
                 xn_arr.append(xn)
                 xn_new = xn - np.sinh(xn)*np.cosh(xn)
                 xn = xn_new
             return xn_arr

         # find upper bound for convergence
         test_range = np.arange(0, 2, 0.0001)
         for i in test_range:
             t = test_newton(i)
             if 'nan' in str(t) or '-inf' in str(t):
                 break
         print('upper bound for |x0| = %f' % i)
```

```
upper bound for |x0| = 1.088700
```

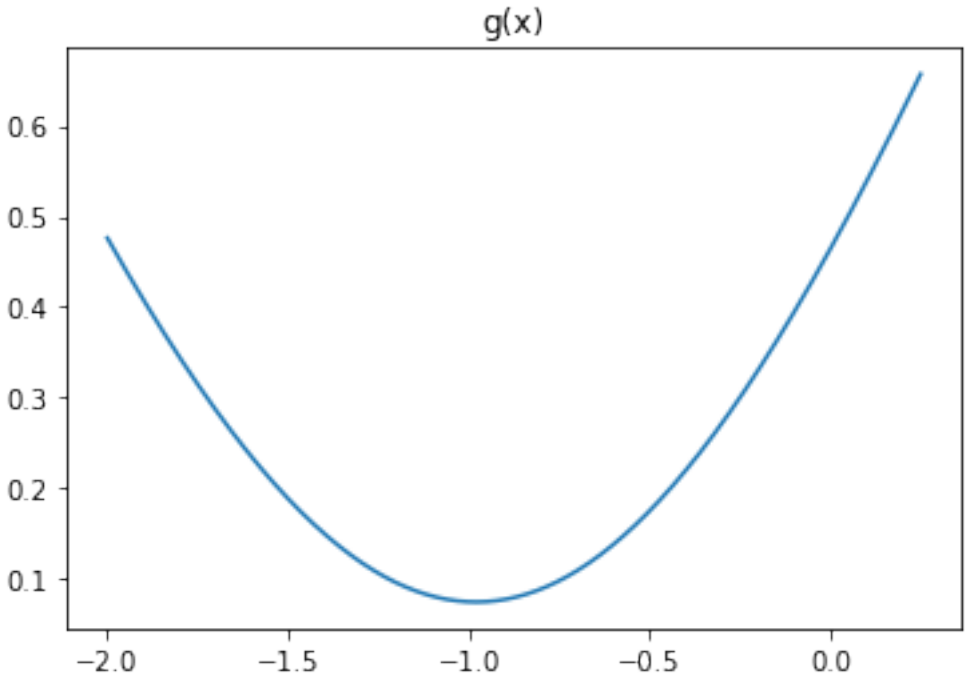
```
/home/jlzhou/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:7: RuntimeWarning: over
  import sys
```

```
/home/jlzhou/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:7: RuntimeWarning: over
  import sys
/home/jlzhou/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:7: RuntimeWarning: inva
  import sys
```

4 6.4h - plot g(x)

In [103]:
```
def g(x):
    k_arr = np.arange(1,11)
    return(np.sum(np.log(np.cosh(x+2/(k_arr**0.5))))/10)

x_arr = np.arange(-2,0.25,0.0001)
g_arr = []
for i in range(len(x_arr)):
    g_arr.append(g(x_arr[i]))

plt.plot(x_arr,g_arr)
plt.title('g(x)')
```

Out[103]: `<matplotlib.text.Text at 0x7f2f506c4d68>`



4.0.1 Due to the summation, taking the derivative of the function and determining the exact value which gives the extrema of the function becomes very difficult

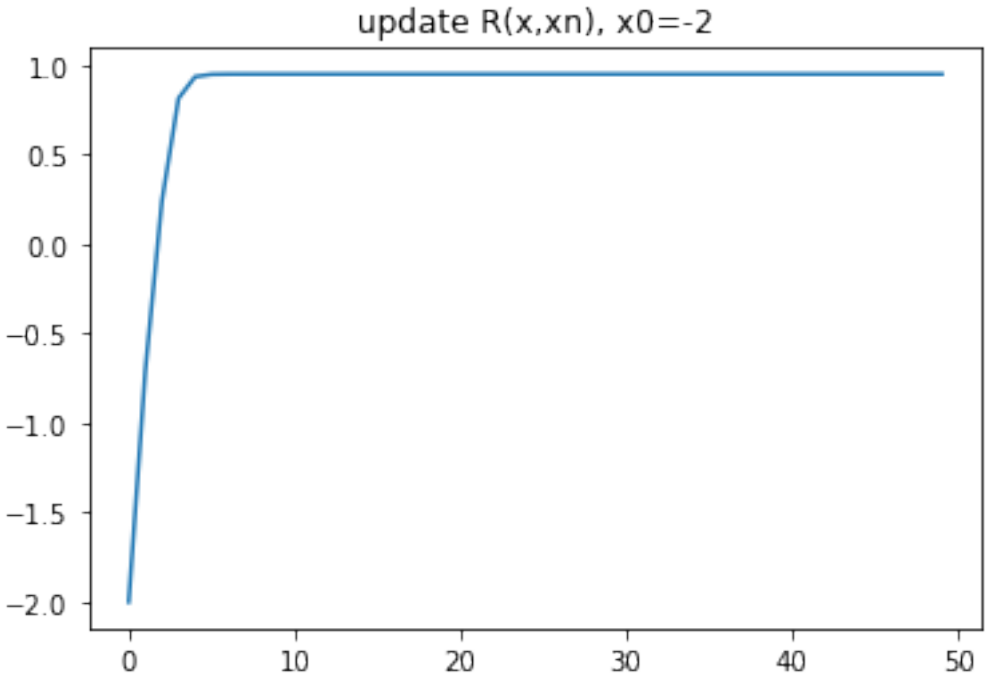
5 6.4k - R(x,xn)

```
In [127]: def update_R(xn):
              k_arr = np.arange(1,11)
              return(xn-np.sum(np.tanh(xn+2/np.sqrt(k_arr)))/10)

          def test(x0):
              xn_arr = []
              xn = x0
              for i in range(50):
                  xn_arr.append(xn)
                  xn_new = xn-update_R(xn)
                  xn = xn_new
              return xn_arr
```

```
In [132]: # test 1 - x0=-2
          g_arr1 = test(-2)
          print(np.around(g_arr1[-3:],4))
          plt.plot(g_arr1)
          plt.title('update R(x,xn), x0=-2')
```

```
[ 0.9513  0.9513  0.9513]
```

```
Out[132]: <matplotlib.text.Text at 0x7f2f4fe6fc88>
```



In [131]:
```
# test 2 - x0=3
g_arr2 = test(3)
print(np.around(g_arr2[-3:],4))
plt.plot(g_arr2)
plt.title('update R(x,xn), x0=3')
```

```
[ 0.9513  0.9513  0.9513]
```

Out[131]: <matplotlib.text.Text at 0x7f2f4fee0630>