ANLY561 Homework 3

Hongyang Zheng

1.

a.

$g(x) = f(x) - log(x) - log(3-x) = \frac{1}{4}x^2 - log(x) - log(3-x)$

$g'(x) = \frac{1}{2}x - \frac{1}{x} + \frac{1}{3-x}$

$g''(x) = \frac{1}{2} + \frac{1}{x^2} + \frac{1}{(3-x)^2}$

Since $g''(x) > 0$ for all x, $g(x)$ is strictly convex.

Plug $x^{(0)} = 1$ into $g'(x)$ and then $g'(x) = 0$. Therefore, $x^{(0)} = 1$ is the minimizing solution for $g(x)$.

Therefore, the optimal solution to the centering step for the log-barrier method is $x^{(0)} = 1$

b.

In the outer loop 1: $t = M * t = 2$

(1)inner loop 1: $x_0 = 1$

$flb = \frac{1}{4}x^2 - \frac{1}{2}log(x) - \frac{1}{2}log(3-x)$

$dflb = \frac{1}{2}x - \frac{1}{2x} + \frac{1}{2(3-x)}$

$dx = -dflb$

In the backtraking: $\alpha = 0.5, \beta = 0.5, df = 0.25, \delta = -0.03125, f_0 = -0.09657359$

$n = 0, t = \beta^n = 1, dx = -0.25, x = x_0 + dx * t = 0.75$

$C = f_0 + \delta * t = -0.1278236$

$f(x) = flb(0.75) = -0.1209991 > C$

$n = 1, t = \beta^n = 0.5, x = x_0 + dx * t = 0.875$

$C = f_0 + \delta * t = -0.1121986$

$f(x) = flb(0.875) = -0.118714$

since $f(x) < C, x^{1\_1} = 0.875$

(2) inner loop 2: $x_0 = 0.875$

In the backtraking: $\alpha = 0.5, \beta = 0.5, \delta = -0.005137482, f_0 = -0.118714$

$t = \beta^n = 1, dx = -0.1013655, x = x_0 + dx * t = 0.7736345$

$C = f_0 + \delta * t = -0.1238514$

$f(x) = flb(0.7736345) = -0.1222298 > C$

$n = 1, t = \beta^n = 0.5, x = x_0 + dx * t = 0.8243173$

$C = f_0 + \delta * t = -0.1212827$

$f(x) = flb(0.8243173) = -0.1221966$

since $f(x) < C, x^{1\_2} = 0.8243173$

Therefore, $x^{(1)}_{outer} = 0.8243173$

In the outer loop 2: $t = M * t = 4$

(1)inner loop 1: $x_0 = 0.8243173$

$flb = \frac{1}{4}x^2 - \frac{1}{4}log(x) - \frac{1}{4}log(3 - x)$

$dflb = \frac{1}{2}x - \frac{1}{4x} + \frac{1}{4(3-x)}$

$dx = -dflb$

In the backtraking: $\alpha = 0.5, \beta = 0.5, df = 0.2237838, \delta = -0.02503959, f_0 = 0.02383907$

$n = 0, t = \beta^n = 1, dx = -0.2237838, x = x_0 + dx * t = 0.6005335$

$C = f_0 + \delta * t = -0.00120052$

$f(x) = flb(0.6005335) = -0.001167271 > C$

$n = 1, t = \beta^n = 0.5, x = x_0 + dx * t = 0.7124254$

$C = f_0 + \delta * t = 0.01131928$

$f(x) = flb(0.7124254) = 0.004784474$

since $f(x) < C, x^{2\_1} = 0.7124254$

(2) inner loop 2: $x_0 = 0.7124254$

In the backtraking: $\alpha = 0.5, \beta = 0.5, \delta = -0.006564838, f_0 = 0.004784474$

$t = \beta^n = 1, dx = -0.1145848, x = x_0 + dx * t = 0.5978406$

$C = f_0 + \delta * t = -0.001780364$

$f(x) = flb(0.5978406) = -0.001130896 > C$

$n = 1, t = \beta^n = 0.5, x = x_0 + dx * t = 0.655133$

$C = f_0 + \delta * t = 0.001502055$

$f(x) = flb(0.655133) = -0.00002810641$

since $f(x) < C, x^{2\_2} = 0.655133$

Therefore, $x^{(2)}_{outer} = 0.655133$

2.

(1) calculate $\alpha^{(1)}$

$$-\phi'_{Q1}(\alpha^{(0)}) = \frac{1}{4}((-1) * (-1) * logit(-\alpha^{(0)} * (-1) * (-1))$$

$$+0 + 1 * 1 * logit(-\alpha^{(0)} * 1 * 1) + 1 * 1 * logit(-\alpha^{(0)} * 1 * 1))$$

$$-\phi'_{Q1}(\alpha^{(0)}) = \frac{3}{4}(logit(-\alpha^{(0)}))$$

$$= \frac{3}{4} * \frac{1}{1+e^{\alpha^{(0)}}} = \frac{3}{4} * \frac{1}{1+e} = 0.2017061$$

$$\alpha^{(1)} = \alpha^{(0)} + \gamma^{(0)} * 0.2017061 = 1.100853$$

(2) calculate $\alpha^{(2)}$

$$-\phi'_{Q2}(\alpha^{(1)}) = \frac{1}{4}((-1) * (-1) * logit(-\alpha^{(1)} * (-1) * (-1))$$

$$+(-1) * (-1) * logit(-\alpha^{(1)} * (-1) * (-1)) + 0 + 1 * (-1) * logit(-\alpha^{(1)} * 1 * (-1)))$$

$$-\phi'_{Q2}(\alpha^{(1)}) = \frac{2}{4}(logit(-\alpha^{(1)})) - \frac{1}{4}(logit(\alpha^{(1)}))$$

$$= \frac{2}{4} * \frac{1}{1+e^{\alpha^{(1)}}} - \frac{1}{4} * \frac{1}{1+e^{-\alpha^{(1)}}}$$

$$= \frac{2}{4} * \frac{1}{1+e^{1.100853}} - \frac{1}{4} * \frac{1}{1+e^{-1.100853}} = 0.12479 - 0.187605$$

$$= -0.062815$$

$$\alpha^{(2)} = \alpha^{(1)} + \gamma^{(1)} * (-0.062815) = 1.075727$$

In [11]:

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

def backtracking1D(x0, dx, f, df0, alpha=0.5, beta=0.5, verbose=False):

    print('In backtracking...')

    if verbose:
        n=0
        xs = [x0 + dx] * 3

    ######################################
    # The core of the algorithm
    ######################################
    delta = alpha * dx * df0 # Just precomputing the alpha times increment times
 derivative factor
    t = 1 # Initialize t=beta**0; beta**n in the loop
    f0 = f(x0) # Evaluate for future use
    x = x0 + dx # Initialize x_{0, inner}, $x = x^{(0)}+\beta^0\Delta x$
    fx = f(x)
    print(fx)
    while (not np.isfinite(fx)) or fx > f0 + delta * t:
        print(fx)
        t = beta * t
        x = x0 + t * dx
        print(x)
        fx = f(x)
    ######################################

        if verbose:
            n += 1
            xs.append(x)
            xs.pop(0)

    if verbose:
        u = 1.1 * np.abs(xs[0] - x0)
        l = 0.1 * np.abs(xs[0] - x0)
        if dx < 0:
            s = np.linspace(x0 - u, x0 + l, 100)
            xi = [x0-u, x0]
            fxi = [f(x0) - alpha*u*df0, f(x0)]
        else:
            s = np.linspace(x0 - l, x0 + u, 100)
            xi = [x0, x0 + u]
            fxi = [f(x0), f(x0) + alpha*u*df0]

        y = np.zeros(len(s))
        for i in range(len(s)):
            y[i] = f(s[i]) # Slow for vectorized functions

        plt.figure('Backtracking illustration')
        arm, =plt.plot(xi, fxi, '--', label='Armijo Criterion')
        fcn, =plt.plot(s, y, label='Objective Function')
        plt.plot([s[0], s[-1]], [0, 0], 'k--')
        pts =plt.scatter(xs, [0 for p in xs], label='Backtracking points for n=%
d, %d, %d' % (n, n+1, n+2))
        plt.scatter(xs, [f(p) for p in xs], label='Backtracking points for n=%d,
 %d, %d' % (n, n+1, n+2))
        init =plt.scatter([x0, x0], [0, f(x0)], color='black', label='Initial po
```

```
int')
        plt.xlabel('x')
        plt.ylabel('f(x)')
        plt.legend(handles=[arm, fcn, pts, init])
        plt.show()

    return x

def lb1D(x, a, b):

    return -np.log(x-a)-np.log(b-x)

def dlb1D(x, a, b):

    return 1/(b-x) - 1/(x-a)

def d2lb1D(x, a, b):

    return 1/((b-x)**2) + 1/((x-a)**2)

def log_barrier_opt_1D(a, b, x0, f, df, d2f=None, al=0.5, be=0.5, M=2, init_iter
=1, out_iter=2, in_iter=2, verbose=False):

    # First, approximate the solution with t=1
    x = x0
    if verbose:
        pts = [x0]

    # Centering step
    for i in range(init_iter):
        flb = lambda z: f(z) + lb1D(z, a, b)
        dflb0 = df(x) + dlb1D(x, a, b)
        dx = -dflb0
        if d2f is not None:
            dx = dx / (d2f(x) + d2lb1D(x, a, b))
        x = backtracking1D(x, dx, flb, dflb0, alpha=al, beta=be)
        if verbose:
            pts.append(x)

    if verbose:
        s = np.linspace(a+1e-6, b-1e-6, 100)
        y = np.zeros(100)
        q = np.zeros(len(pts))
        for i in range(100):
            y[i] = flb(s[i])
        for i in range(len(pts)):
            q[i] = flb(pts[i])

        fl = min(np.min(q), 0)
        fu = max(np.max(q), 0)

        interval_length = np.max(pts) - np.min(pts)
        range_length = fu - fl

        l = np.min(pts) - 0.1*interval_length
        u = np.max(pts) + 0.1*interval_length
        fl = np.min(q) - 0.1*range_length
        fu = np.max(q) + 0.1*range_length

        plt.plot([s[0], s[-1]], [0, 0], 'k--')
        obj, =plt.plot(s, y, label='Objective plus barrier')
```

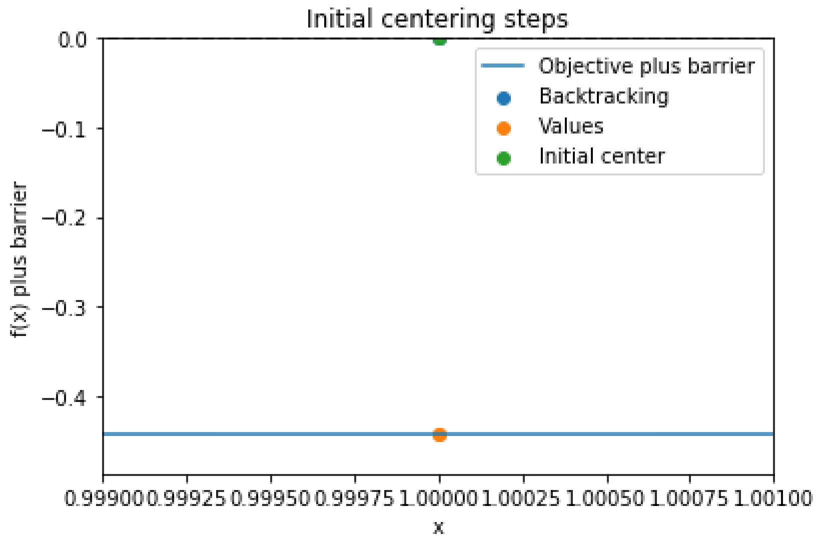
```
        bt =plt.scatter(pts, np.zeros(len(pts)), label='Backtracking')
        vals =plt.scatter(pts, q, label='Values')
        init =plt.scatter([pts[-1]], 0, label='Initial center')
        plt.axis([l, u, min(fl,0), max(fu,0)])
        plt.legend(handles=[obj, bt, vals, init])
        plt.xlabel('x')
        plt.ylabel('f(x) plus barrier')
        plt.title('Initial centering steps')
        plt.show()

    # Now begin the outer iterations
    t=1
    for i in range(out_iter):
        t = M * t
        if verbose:
            pts = [x]
        for j in range(in_iter):
            flb = lambda z: f(z) + lb1D(z, a, b)/t
            dflb0 = df(x) + dlb1D(x, a, b)/t
            dx = -dflb0
            if d2f is not None:
                dx = dx / (d2f(x) + d2lb1D(x, a, b)/t)
            x = backtracking1D(x, dx, flb, dflb0, alpha=al, beta=be)
            pts.append(x)

        if verbose:
            s = np.linspace(a+1e-6, b-1e-6, 100)
            y = np.zeros(100)
            q = np.zeros(len(pts))
            for k in range(100):
                y[k] = flb(s[k])
            for k in range(len(pts)):
                q[k] = flb(pts[k])

            fl = min(np.min(q), 0)
            fu = max(np.max(q), 0)

            interval_length = np.max(pts) - np.min(pts)
            range_length = fu - fl

            l = np.min(pts) - 0.1*interval_length
            u = np.max(pts) + 0.1*interval_length
            fl = np.min(q) - 0.1*range_length
            fu = np.max(q) + 0.1*range_length

            obj, =plt.plot(s, y, label=('Objective plus barrier at t=%f' % t))
            bt =plt.scatter(pts, np.zeros(len(pts)), label='Inner loop iterates'
)
            outer =plt.scatter([pts[-1]], 0, label='Outer loop iterate', color=
'red')
            vals =plt.scatter(pts, q, label='Values at iterates')
            plt.axis([l, u, min(fl, 0), max(fu, 0)])
            plt.legend(handles=[obj, bt, outer, vals])
            plt.xlabel('x')
            plt.ylabel('f(x) plus barrier')
            plt.title('Log barrier steps at outer iteration %d' % i)
            plt.show()

    return x
```

```
a=0
```

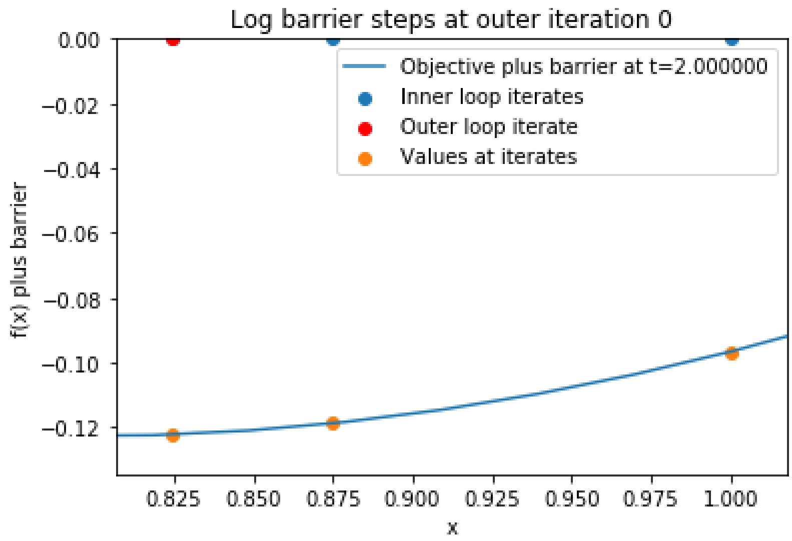
```
b=3
x0=1
f=lambda x: 0.25*(x)**2
df=lambda x: 0.5*x

x_approx = log_barrier_opt_1D(a, b, x0, f, df, verbose=True, out_iter=2)
```

```
In backtracking...
-0.4431471805599453
```

```
/anaconda3/lib/python3.6/site-packages/matplotlib/axes/_base.py:312
4: UserWarning: Attempting to set identical left==right results
in singular transformations; automatically expanding.
left=1.0, right=1.0
  'left=%s, right=%s') % (left, right))
```




```
In backtracking...
-0.12099907188227393
-0.12099907188227393
0.875
In backtracking...
-0.12222976153686824
-0.12222976153686824
0.8243172268907563
```




```
In backtracking...
-0.0011672705214830353
-0.0011672705214830353
0.7124253413299524
In backtracking...
-0.0011308957614862003
-0.0011308957614862003
0.6551329622681582
```