HW9-2 (1)

April 12, 2022

1 Assinment 9

Daniel González Esparza

Zainab Alsuwaykit

2 Q1

1. Central path.

KKT conditions for the log barrier problem:

$$\begin{aligned}
&\nabla f_0(x) + \tau \textstyle\sum_{i=1}^{m} \frac{1}{-f_i(x)} \nabla f_i(x) + A^t \nu = 0 \\
&Ax = b \\
&-\lambda_i f_i(x) = \tau \\
&f(x) \leq 0 \\
&\lambda \geq 0 \\
&\tau \frac{1}{-f(x)} \nabla f(x) \geq 0
\end{aligned}$$

f(x) should be negative to satisfy the log and not equal to zero constraints to avoid the singularity on the hessain.

The centeral path KKT condition:

$$\begin{aligned}
&\nabla f_0(x) + \textstyle\sum_{i=1}^{m} \lambda_i \nabla f_i(x) + A^t \nu = 0 \\
&\lambda \geq 0 \\
&-\lambda f(x) = \tau \\
&\lambda = \frac{-\tau}{f(x)}
\end{aligned}$$

The two problem have the same KKT conditions.

3 Q2

2. Interior Point Method for LP.

$$\begin{array}{ll}
\text{minimize} & -\sum_{i=1}^{n} \log x_i \\
\text{subject to} & Ax = b
\end{array}$$

Minimizing the log barrier with x makes all x elements positive thus, the solution to this problem can be used as an feasible initial point because the optimization problem gurantees the constraints are satisies.

We can also use the feasibilty method that can be written as follwing:

$$\begin{array}{ll} \text{minimize} & s \\ \text{subject to} & Ax = b \\ & -x \leq s \end{array}$$

s is the upper bound of the infeasibility of the inequality condition, however, we want s<0. if s<=0 then x is feasible, otherwise, it is infeasible.

Equality constrained problem:

$$\begin{array}{ll} \text{minimize} & p^t x - \tau \sum_{i=1}^m \log{(x_i)} \\ \text{subject to} & Ax = b \end{array}$$

KKT conditions:

$$\begin{array}{l} p - \tau \sum_{i=1}^m \frac{1}{x_i} + A^t \nu = 0 \\ Ax = b \\ x \geq 0 \end{array}$$

The central path KKT conditon:

$$\begin{array}{l} p + A^t \nu + \lambda = 0 \\ -\lambda_i x_i = \tau \\ f(x) \leq 0 \\ x \geq 0 \\ \lambda \geq 0 \\ \therefore \lambda = \frac{-\tau}{x} \end{array}$$

Therefore, the KKT conditions are the same.

Every step we solve:

$$\begin{bmatrix} diag(\frac{\tau}{(x^k)^2}) & A^t \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta \nu \end{bmatrix} = - \begin{bmatrix} p - \frac{\tau}{x^k} + A^t \nu^k \\ Ax^k - b \end{bmatrix}$$

[14]:
```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from numpy import linalg as la


import scipy.optimize

np.random.seed(9727)              # seed the random number generator
n = 150
p = 100

# Generate random data
A = np.hstack( (np.random.randn(p, n-p), np.eye(p)) )
b = A @ np.random.rand(n)
```

```python
print(A.shape)
pobj = np.concatenate( (np.random.randn(n-p), np.zeros(p)) )


# Solution may be compared to the one generated by linprog
# https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.linprog.
↪html
solution = scipy.optimize.linprog(pobj, method='interior-point', A_eq = A, b_eq␣
↪= b)
print(solution)
```

```
(100, 150)
     con: array([-3.37774253e-12, -1.14432908e-11,  1.24833477e-12,
2.15996110e-11,
        2.77027290e-11, -5.08537656e-12,  8.99147423e-12, -2.16413554e-11,
       -1.19344534e-11, -2.16697771e-11, -3.47801787e-11,  1.84448012e-11,
       -2.82085466e-11,  3.83248988e-12,  1.11279874e-11, -8.33999536e-12,
       -7.05666636e-11, -2.82431856e-11, -1.60840230e-11,  1.08073550e-11,
        1.75992554e-12, -1.48743240e-11, -1.97447614e-11, -9.09716746e-12,
       -2.27275976e-11, -1.50599533e-11,  2.56417110e-11, -3.04209991e-11,
        3.06688008e-12,  1.13136167e-11, -1.47926116e-11, -2.73452372e-11,
       -2.10169659e-11,  2.07824868e-11,  3.16373594e-11, -5.87252469e-12,
        1.99711359e-11,  4.91946484e-11,  1.47957202e-11,  8.16224865e-12,
        1.34114941e-11,  1.34656730e-11, -8.35154168e-12,  4.43134418e-12,
        1.10840226e-11, -4.20818935e-11, -2.14086526e-11, -3.71498388e-11,
       -5.11035658e-12, -3.41948692e-13, -1.95656824e-11, -3.98596711e-11,
        3.55981911e-11, -4.03788114e-12, -1.90203409e-11, -3.55513396e-11,
       -1.45234935e-11, -1.19078081e-11, -3.76951803e-11, -3.90216748e-11,
       -2.18838281e-11,  2.38724596e-11, -2.54474219e-12, -1.89857019e-11,
        1.89213090e-11, -2.96056513e-11,  1.13495879e-11, -2.94324565e-11,
        1.28084210e-11,  2.43192133e-11,  1.00819353e-11,  2.15472085e-11,
        4.07851530e-12,  2.21505037e-11,  9.93516380e-12,  1.24233956e-11,
        1.84217086e-11, -7.41984252e-12, -5.58220137e-12, -4.13775680e-11,
        7.05080438e-12,  6.22435437e-12, -5.26068078e-12,  3.64752673e-12,
       -2.48259191e-11, -5.07887066e-11,  1.06155085e-11,  3.02877723e-11,
       -7.36566363e-12, -1.45868873e-11, -1.69446679e-11, -2.26600960e-11,
       -1.88880023e-11,  9.59854418e-12, -5.45030687e-11, -4.49000837e-11,
        2.61382027e-11, -1.54862789e-11, -3.83906240e-11, -1.99973371e-12])
     fun: -3.718631950458553
 message: 'Optimization terminated successfully.'
     nit: 10
   slack: array([], dtype=float64)
  status: 0
 success: True
       x: array([3.19113347e-01, 1.23789828e-12, 1.40267652e+00, 1.02711427e+00,
       1.07991208e-02, 1.06453868e+00, 1.91100942e-01, 1.42157824e+00,
       1.24133754e-02, 9.41924038e-01, 4.40352987e-01, 1.86271954e-01,
```

```
       3.57248774e-01, 5.60421394e-01, 2.89119813e-01, 1.98969969e-01,
       5.32761763e-01, 5.65638861e-01, 5.33224765e-01, 7.79170816e-01,
       2.01684002e-01, 8.36668858e-01, 3.85641927e-01, 3.75520296e-01,
       2.74166282e-01, 1.81482834e-01, 1.03364873e-01, 1.11404813e+00,
       1.18671898e-11, 5.63656412e-01, 6.88530511e-01, 9.22901410e-01,
       9.56819641e-01, 2.79526016e-12, 1.42491419e-01, 2.83386018e-01,
       7.12063344e-02, 7.83780739e-01, 7.39582721e-01, 1.40928005e-12,
       3.80660211e-01, 9.07770805e-01, 4.18934003e-01, 6.46374204e-01,
       4.48985120e-01, 1.15137691e-11, 4.23638348e-12, 1.91466336e-01,
       4.26490857e-12, 7.85892033e-01, 1.10150337e-11, 2.37729488e-10,
       6.31784556e-02, 6.32967605e-12, 2.16038536e-11, 9.41845553e-12,
       1.76866003e-01, 9.44341368e-01, 2.39234283e+00, 9.54346374e-01,
       2.24974504e-10, 6.09753879e-11, 2.53179874e-01, 4.18265557e-01,
       3.79829835e-11, 1.27970509e-11, 4.39189449e-11, 2.95147029e-11,
       2.85545987e+00, 2.88122869e-11, 7.65993083e-11, 2.17292593e-11,
       3.29098912e-01, 1.39026619e+00, 1.35856876e+00, 3.43503939e+00,
       7.53624050e-02, 2.45898654e+00, 3.84191798e-01, 6.41670025e-01,
       4.05868444e-01, 3.05949388e-11, 3.47630273e-11, 1.14477804e-11,
       1.66063168e-11, 2.66151201e-01, 5.53946228e-01, 2.87823740e-11,
       1.94898950e+00, 2.46698490e-11, 1.27034765e+00, 2.01069657e+00,
       6.78452230e-12, 2.41147101e+00, 1.06870665e+00, 2.56804863e-01,
       1.69867297e+00, 1.40961081e-11, 2.10105524e-11, 1.28933734e-01,
       1.89491902e+00, 2.71734030e-09, 1.82423884e-11, 1.04830677e-11,
       3.76216366e-01, 2.36386282e-11, 1.22447144e+00, 1.61419184e-11,
       2.49066909e-11, 4.18462163e-11, 1.26683216e+00, 2.71139634e-10,
       1.38065399e+00, 2.63441604e+00, 7.17426309e-01, 1.09579254e-11,
       2.58379962e+00, 1.12275112e-11, 2.25323575e-10, 3.11387426e-01,
       1.03029027e-10, 1.46377803e-11, 2.95946524e-01, 3.79553821e-11,
       2.02289066e-11, 1.94389975e-01, 1.15797702e+00, 3.74514153e+00,
       1.36915447e+00, 1.23546513e+00, 1.60478248e+00, 1.23773068e-01,
       4.18734998e+00, 1.82757378e+00, 1.21647492e-11, 9.86892052e-01,
       1.63842866e+00, 1.62422063e+00, 7.92650874e-01, 2.43613638e-11,
       7.10519065e-11, 2.19489497e+00, 2.38795239e+00, 8.51659025e-01,
       7.28238235e-01, 3.50935123e+00, 5.34366170e-11, 2.21632426e+00,
       3.98243742e-01, 7.46569283e-11])
```

[15]:
```python
def res (A,b, x,v,tau,p,pobj):

    r = np.zeros(n+p)
    r[0:n] = (pobj + (A.T @ v)) - (tau/x)
    r[n:n+p] = A@x-b

    return r
```

[16]:
```python
la.norm(res (A,b, np.ones(n),np.ones(p),10,p,pobj))
```

[16]: 129.07746412659736

[17]:

```
def hess(x,A,tau,p):
    hess_ = np.zeros((n+p,n+p))
    hess_[0:n,0:n]= np.diag(tau/(x**2))
    hess_[n:n+p,0:n] = A
    hess_[0:n,n:n+p] =A.T
    hess_[n:n+p,n:n+p] = 0
    return hess_
```

[18]:

```
def grad(x,A,b,v,tau,p,pobj):

    grad_ = np.zeros(n+p)
    grad_[0:n] = (pobj + (A.T @ v)) - (tau/x)
    grad_[n:n+p] = A@x-b

    return grad_
```

[19]:

```
import cvxpy as cp
def feasible(A,b,n):
    #s = cp.Variable(1)
    xf=cp.Variable(n)

    prob = cp.Problem(cp.Minimize(-cp.sum(cp.log(xf))),[A@xf==b])
    prob.solve()
    return xf.value
```

[20]:

```
def newton(n,p,f,res, gf, Hf, A, b, x0,v0,tau,Xs,Ts,pobj):



    k =0
    x =x0
    v =v0

    while(la.norm(res(A,b, x,v,tau,p,pobj))>=1e-5):

        d =  la.solve(Hf(x,A,tau,p),-gf(x,A,b,v,tau,p,pobj))
        #print(la.norm(d))
        t = backtrack(f,res,A,b,d,x,v,tau,n,p,pobj)
        dx = d[0:n]
        dv = d[n:n+p]
        x = x+ t*dx
        v = v+ t*dv

        Xs=np.vstack((Xs,x))
        Ts=np.vstack((Ts,tau))
        k+=1
        print("k",k)
```

```
    return Xs,Ts,x,v
```

[21]:
```
def f (pobj,x,tau):
    for element in x:
        if element < 0:
            return -float('Inf')
    f = pobj.T@x -  tau*np.sum(np.log(x))
    return f
```

[22]:
```
def backtrack(f,res,A,b,d,x0,v0,tau,n,p,pobj):
    t=1
    alpha = 0.8
    beta = 0.9
    dx = d[0:n]
    dv = d[n:n+p]



    while(la.norm(res(A,b,x0+t*dx,v0+t*dv,tau,p,pobj)) > (1-alpha*t)* la.
 ↪norm(res(A,b,x0,v0,tau,p,pobj)) ) :
        t*=beta

    return t
```

[23]:
```
def interiorLP(p, A, b,n,pobj,mu, tol= 1e-5):
    x = feasible(A,b,n)
    v = -1* la.inv(A@A.T)@A@(pobj - (1./x))

    tau = 10
    Xs=np.array([x])
    Ts=np.array([tau])
    l=0
    while( p*tau >= tol):
        #Xs,Ts,x0,v0 = newton(n,p,f,res,grad,hess,A,b,x,v,tau,Xs0,Ts0,pobj)

        k =0
        while(la.norm(res(A,b, x,v,tau,p,pobj))>=1e-5):

            d =  la.solve(hess(x,A,tau,p),-grad(x,A,b,v,tau,p,pobj))
            #print(la.norm(d))
            t = backtrack(f,res,A,b,d,x,v,tau,n,p,pobj)
            dx = d[0:n]
            dv = d[n:n+p]
            x = x+ t*dx
            v = v+ t*dv
```

```
            Xs=np.vstack((Xs,x))
            Ts=np.vstack((Ts,tau))
            k+=1
            #print("k",k)

        tau = tau/mu
        l+=1
        print("inner iteration ",l)
        print("tau",mu*tau)
        print("res", la.norm(res(A,b, x,v,tau,p,pobj)))

    return Xs,Ts,x,v
```

[24]:
```
Xs,Ts,x,v = interiorLP(p, A, b,n,pobj,mu=2, tol= 1e-5)
```

```
inner iteration  1
tau 10.0
res 239.40071621805555
inner iteration  2
tau 5.0
res 119.43419632968426
inner iteration  3
tau 2.5
res 59.46535251249626
inner iteration  4
tau 1.25
res 29.509965588276135
inner iteration  5
tau 0.625
res 14.59070418317551
inner iteration  6
tau 0.3125
res 7.245898476712353
inner iteration  7
tau 0.15625
res 3.7728827618830953
inner iteration  8
tau 0.078125
res 2.2896140483409146
inner iteration  9
tau 0.0390625
res 1.8520924516521822
inner iteration  10
tau 0.01953125
res 2.014162609415054
inner iteration  11
```

```
tau 0.009765625
res 2.3016942165615957
inner iteration  12
tau 0.0048828125
res 2.4619491091088226
inner iteration  13
tau 0.00244140625
res 2.5212821356676423
inner iteration  14
tau 0.001220703125
res 2.55135743711082
inner iteration  15
tau 0.0006103515625
res 2.583385945976659
inner iteration  16
tau 0.00030517578125
res 2.6134956513286904
inner iteration  17
tau 0.000152587890625
res 2.6366240717361444
inner iteration  18
tau 7.62939453125e-05
res 2.650163953644759
inner iteration  19
tau 3.814697265625e-05
res 2.6569772304745918
inner iteration  20
tau 1.9073486328125e-05
res 2.660236305912203
inner iteration  21
tau 9.5367431640625e-06
res 2.6617211718745373
inner iteration  22
tau 4.76837158203125e-06
res 2.662377891382453
inner iteration  23
tau 2.384185791015625e-06
res 2.662675934890767
inner iteration  24
tau 1.1920928955078125e-06
res 2.662816793045655
inner iteration  25
tau 5.960464477539062e-07
res 2.6628851846909263
inner iteration  26
tau 2.980232238769531e-07
res 2.6629188769657066
inner iteration  27
```

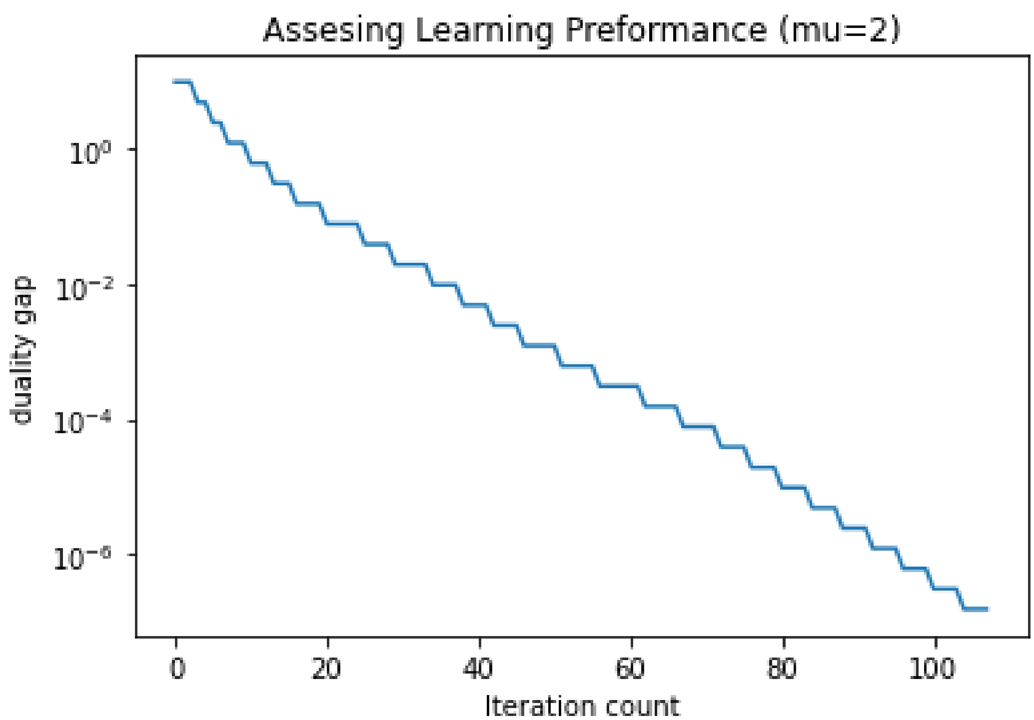
```
tau 1.4901161193847656e-07
res 2.66293559826736
```

The problem does not stop for both mu=2 and mu=10

[25]:
```python
Xlen = Xs.shape[0]
print(Xlen)
f_history = np.zeros(25)
for i in range(25):
    f_history[i] = f(pobj, Xs[i], Ts[i])

plt.figure('Contours')
plt.title('Assesing Learning Preformance (mu=2)')
plt.semilogy(np.arange(0, Xlen), np.absolute(Ts))
plt.xlabel('Iteration count')
plt.ylabel('duality gap')
plt.show()
```

```
108
```



[182]:
```python
Xs,Ts,x,v = interiorLP(p, A, b,n,pobj,mu=10, tol= 1e-5)
```

```
inner iteration  1
tau 10.0
res 430.92128948691806
```

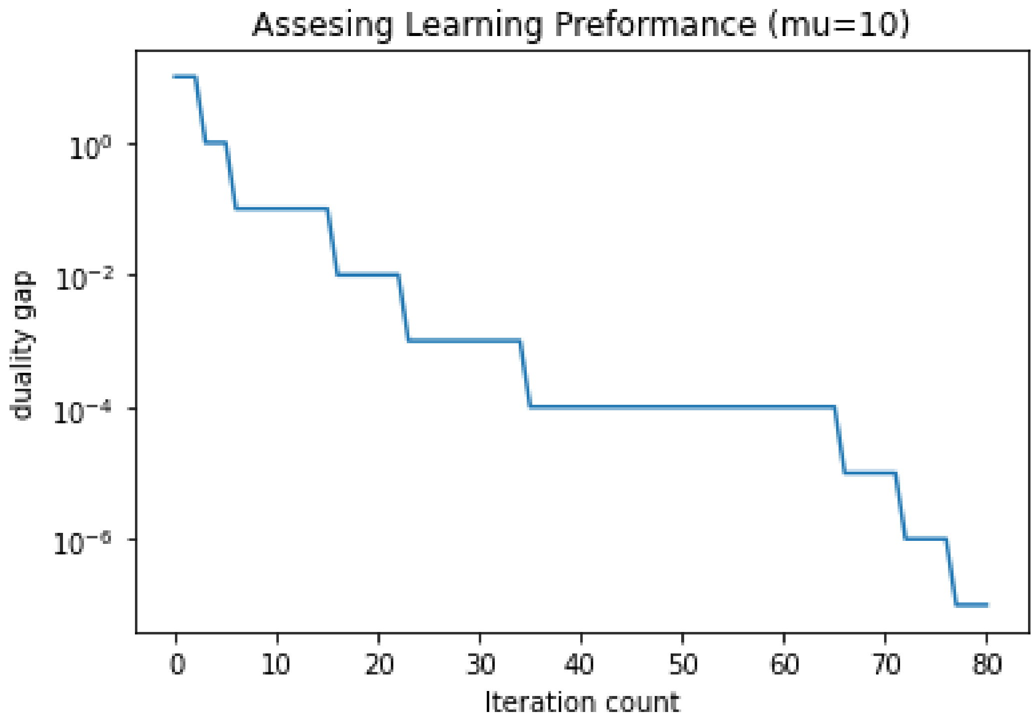
```
inner iteration  2
tau 1.0
res 42.355013212983366
inner iteration  3
tau 0.1
res 4.801183182508477
inner iteration  4
tau 0.01
res 3.516926899703453
inner iteration  5
tau 0.001
res 4.319497848263243
inner iteration  6
tau 0.0001
res 4.446749371833353
inner iteration  7
tau 1.0000000000000003e-05
res 4.506982904688478
inner iteration  8
tau 1.0000000000000002e-06
res 4.520832108687231
inner iteration  9
tau 1.0000000000000002e-07
res 4.522195432821212
```

[178]:

```
Xlen = Xs.shape[0]
print(Xlen)
f_history = np.zeros(Xlen)
for i in range(Xlen):
    f_history[i] = f(pobj, Xs[i], Ts[i])

plt.figure('Contours')
plt.title('Assesing Learning Preformance (mu=10)')
plt.semilogy(np.arange(0, Xlen), np.absolute(Ts))
plt.xlabel('Iteration count')
plt.ylabel('duality gap')
plt.show()
```

```
81
```



the run with =2 took 27 inner steps while the run with =10

took 9 iterations to reach a duality gap almost zero

4 Q3

[27]:
```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from numpy import linalg as la
import cvxpy as cp
```

[28]:
```
np.random.seed(421)             # seed the random number generator, for repeatability
n = 200
m = 100
# Generate a random p-by-n matrix with independent rows
A = np.random.randn(m, n)
S = np.random.randn(n, n)
H=S@np.transpose(S)

while np.linalg.matrix_rank(A) != m :
    print('generating another data set with independent rows')
```

```
    A = np.random.rand(m, n)
while np.linalg.matrix_rank(H) != n :
    print('generating another data set with independent rows')
    S = np.random.randn(n, n)
    H=S@np.transpose(S)
# Generate a random right hand side
b =np.random.randn(m,1)
g =np.random.randn(n,1)
```

[29]:
```
def feasible(A,b):
    m=len(A)
    n=len(A[0])
    x = cp.Variable((n,1))
    s = cp.Variable((1,1))
    objective = cp.Minimize(s)
    one=np.ones((m,1))
    constraints = [A@x-b-s*one<=0, s>=-1]
    prob = cp.Problem(objective, constraints)
    result = prob.solve()
    xx=x.value
    if max(A@xx-b)<0:
        print('Feasible starting point')
    else:
        print('Infeasible starting point')
    return x.value
```

[30]:
```
def grad(x,H,g,A,b,T):
    grad=np.zeros((len(x),1))
    grad=(H@x+g)/T
    for i in range(0,len(x)):
        grad[i]+=A[:,i]@(1./(-(A@x-b)))
    return grad
```

[31]:
```
def Hess(x,H,g,A,b,T):
    hess=np.zeros((len(x),len(x)))
    hess=(1/T)*H
    for i in range(0,len(x)):
        for j in range(0,len(x)):
            hess[i,j]+=(A[:,j]*A[:,i])@(1./(A@x-b)**2)
    return hess
```

[32]:
```
def fun(x,H,g,A,b,T):
    f=(0.5/T)*np.transpose(x)@H@x+np.transpose(g)@x-sum(np.log(-(A@x-b)))
    return f
```

[33]:
```
def DualGap(x,H,g,T,m):
    f=T*m
```

```
    #f=(0.5)*np.transpose(x)@H@x+np.transpose(g)@x-T*m
    return f.reshape(1,1)
```

[46]:
```
def newton_m(fun, grad, hess, x0, H, g, A, b, T, M,tol,it):
    x=x0
    print('Newton')
    n=len(x)
    Fs=np.array([x])
    Ts=np.array(T)
    E=1
    count=0
    while((E> tol) or (T>1e-05)):
        p= la.solve(hess(x,H,g,A,b,T),-grad(x,H,g,A,b,T))
        t = backtrack_r(fun,grad(x,H,g,A,b,T),p,x,H,g,A,b,T,alpha=0.1,beta=0.9)
        x+=t*p
        Fs=np.vstack((Fs,[x]))
        Ts=np.vstack((Ts,T))
        E=np.absolute (T*fun(Fs[-1],H,g,A,b,T)-Ts[-2]*fun(Fs[-2],H,g,A,b,T))
        print('k= ',len(Ts),'Fun',T*fun(x,H,g,A,b,T),'Error',E,'Tau',T)
        count+=1
        if E<tol:
            T=T/M
            count=0
    return Fs,x,Ts
```

[51]:
```
def backtrack_r(fun,grad, p,xk,H,g,A,b,T,alpha,beta):
    #print('back')
    t=1
    n=len(xk)
    while(max(A@(xk+t*p)-b)>0):
        t *=beta
    while(((fun(xk+t*p,H,g,A,b,T)> fun(xk,H,g,A,b,T) + alpha* t * (np.
 ↪transpose(grad) @ p)))):
        t *=beta
    return t
```

[52]:
```
x0=feasible(A,b)
```

```
Feasible starting point
```

[54]:
```
T=100
M=20
tol=1e-08
it=3
Sol=newton_m(fun, grad, Hess, x0, H, g, A, b, T, M, tol,it)
```

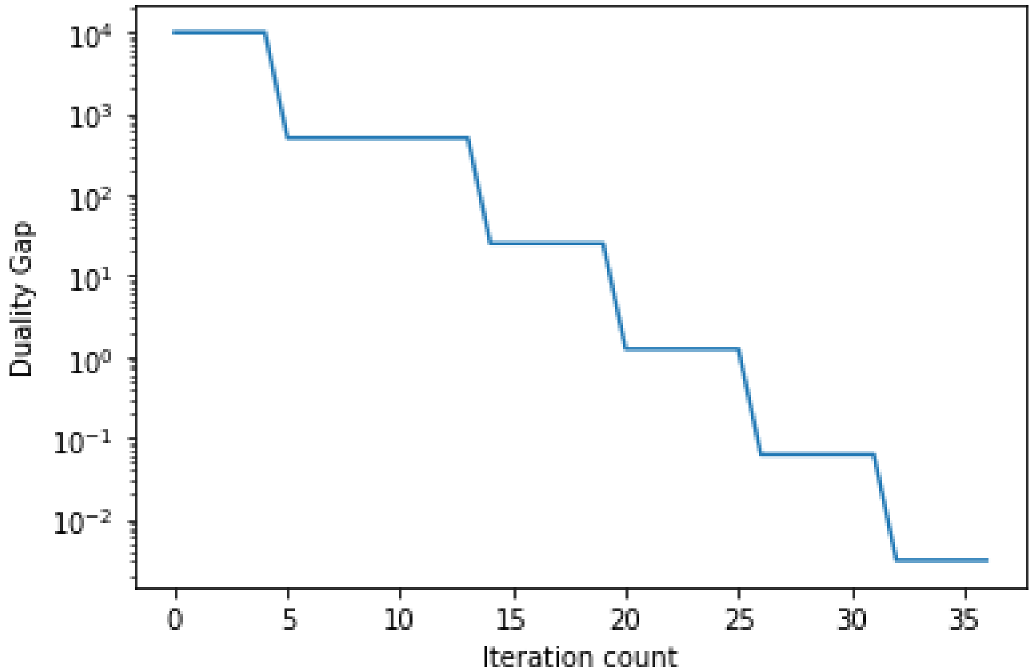
```
Newton
```

```
k=  2 Fun [[-23197.8035988]] Error [[7130.16043383]] Tau 100
k=  3 Fun [[-28126.41396322]] Error [[4928.61036442]] Tau 100
k=  4 Fun [[-29458.65253954]] Error [[1332.23857632]] Tau 100
k=  5 Fun [[-29458.65253954]] Error [[0.]] Tau 100
k=  6 Fun [[-427.71772788]] Error [[59463.86041612]] Tau 5.0
k=  7 Fun [[-443.30998474]] Error [[15.59225686]] Tau 5.0
k=  8 Fun [[-448.08287218]] Error [[4.77288744]] Tau 5.0
k=  9 Fun [[-449.09180386]] Error [[1.00893168]] Tau 5.0
k=  10 Fun [[-449.46619064]] Error [[0.37438678]] Tau 5.0
k=  11 Fun [[-449.53783742]] Error [[0.07164678]] Tau 5.0
k=  12 Fun [[-449.54232626]] Error [[0.00448883]] Tau 5.0
k=  13 Fun [[-449.54235596]] Error [[2.97074813e-05]] Tau 5.0
k=  14 Fun [[-449.54235597]] Error [[1.37868028e-09]] Tau 5.0
k=  15 Fun [[102.93241579]] Error [[4853.4559165]] Tau 0.25
k=  16 Fun [[43.93201675]] Error [[59.00039904]] Tau 0.25
k=  17 Fun [[33.9480603]] Error [[9.98395645]] Tau 0.25
k=  18 Fun [[33.425474]] Error [[0.5225863]] Tau 0.25
k=  19 Fun [[33.41179835]] Error [[0.01367565]] Tau 0.25
k=  20 Fun [[33.41179835]] Error [[7.10542736e-15]] Tau 0.25
k=  21 Fun [[24.28604866]] Error [[543.29195732]] Tau 0.0125
k=  22 Fun [[20.62138439]] Error [[3.66466428]] Tau 0.0125
k=  23 Fun [[18.22895808]] Error [[2.3924263]] Tau 0.0125
k=  24 Fun [[16.77681291]] Error [[1.45214518]] Tau 0.0125
k=  25 Fun [[16.29506545]] Error [[0.48174746]] Tau 0.0125
k=  26 Fun [[16.29506545]] Error [[0.]] Tau 0.0125
k=  27 Fun [[13.79292126]] Error [[267.32122034]] Tau 0.000625
k=  28 Fun [[13.65607439]] Error [[0.13684687]] Tau 0.000625
k=  29 Fun [[13.57849216]] Error [[0.07758224]] Tau 0.000625
k=  30 Fun [[13.47592508]] Error [[0.10256707]] Tau 0.000625
k=  31 Fun [[13.43055597]] Error [[0.04536912]] Tau 0.000625
k=  32 Fun [[13.43055597]] Error [[0.]] Tau 0.000625
k=  33 Fun [[13.20512579]] Error [[251.26709302]] Tau 3.125e-05
k=  34 Fun [[13.19980492]] Error [[0.00532087]] Tau 3.125e-05
k=  35 Fun [[13.19143797]] Error [[0.00836695]] Tau 3.125e-05
k=  36 Fun [[13.18613509]] Error [[0.00530288]] Tau 3.125e-05
k=  37 Fun [[13.18613509]] Error [[1.77635684e-15]] Tau 3.125e-05
```

[55]:

```python
xi=Sol[0]
Ti=Sol[2]
Dual=np.zeros((len(xi),1))
itera=np.zeros((len(xi),1))
for i in range(len(xi)):
    Dual[i]=DualGap(xi[i],H,g,Ti[i],m)
    itera[i]=i
plt.semilogy(itera,np.abs(Dual))
plt.xlabel('Iteration count')
plt.ylabel(r'Duality Gap')
```

[55]: Text(0, 0.5, 'Duality Gap')



5 Q4

4. Inverse barrier.

- Considering that the barrier function is given by the resprical value of f(x), then the objective function with the barrier function is as follow:

$$\frac{1}{\tau}f_0(x) + \sum_{i=1}^{m} \frac{-1}{f_i(x)}$$

Along the central path, the function

$$\frac{1}{\tau}f_0(x)$$

is the one that is minimized.

- a dual feasible from x∗( ):

$$\frac{1}{\tau}\nabla f_0(x) + \sum_{i=1}^{m} \frac{\nabla f_i(x)}{f_i(x)^2} = 0$$
$$\nabla f_0(x) + \tau\sum_{i=1}^{m} \frac{\nabla f_i(x)}{f_i(x)^2} = 0$$
$$\nabla f_0(x) + \lambda^t \nabla f_i(x^*) = 0$$
$$\lambda_i = \frac{\tau}{f_i(x^*)^2}$$

To find the duality gap:

$$
\begin{aligned}
&minimize \qquad f_0(x)\\
&s.t \quad f_i(x) \leq 0\\
&L = f_0(x) + \textstyle\sum_{i=1}^{m} \lambda f_i(x)
\end{aligned}
$$

From the previous exercise we found that

$$\lambda_i = \frac{\tau}{f_i(x^*)^2}$$

, then

$$
\begin{aligned}
&f^* = minL(x^*, \lambda(\tau))\\
&f^* = f_0(x^*) + \textstyle\sum_{i=1}^{m} \lambda f_i(x)\\
&f^* = f_0(x^*) + \textstyle\sum_{i=1}^{m} \frac{\tau}{f_i(x^*)}
\end{aligned}
$$

- The pseudo code is the following:

$$
\begin{aligned}
&Chooseafeasiblex^0\\
&whileDualityGap > Tol_1\\
&\qquad while(||\nabla f_\phi(x^k)||)\\
&\qquad\qquad solve \quad \nabla^2 f_\phi(x^k)\Delta x = -\nabla f_\phi(x^k)\\
&\qquad\qquad linesearcht = \alpha tuntil f_\phi(x^k + t\Delta x) \leq f_\phi(x^k) + \alpha t \nabla f_\phi(x^k)^T \Delta x\\
&\qquad\qquad update : x^{k+1} = x^k + \Delta x; \quad k = k+1\\
&\qquad end\\
&\qquad \tau = \tau/\mu \quad l = l+1\\
&end
\end{aligned}
$$

  Where the Duality Gap is given by

$$f_0(x^*) + \sum_{i=1}^{m} \frac{\tau}{f_i(x^*)}$$

  .

- First, for calculating the gradient and Hessian, we must define the function $f_\phi(x)$ as follows:

$$f_\phi(x) = \frac{1}{\tau} f_0(x) + \phi(x)$$

  Then, applying the nambla operator:

$$\nabla f_\phi(x) = \frac{1}{\tau}\nabla f_0(x) + \nabla\phi(x)$$

  if $\phi = \sum_{i=1}^{m} \frac{-1}{f_i(x)}$, then the gradient is given by:

$$\nabla\phi(x) = \sum_{i=1}^{m} \frac{\nabla f_i(x)}{f_i(x)^2}$$

  Finally, the gradient of the fuction is:

$$\nabla f_\phi(x) = \frac{1}{\tau}\nabla f_0(x) + \sum_{i=1}^{m} \frac{\nabla f_i(x)}{f_i(x)^2}$$

For finding the Hessian we follow the same steps applying the nabla operator as follows

$$\nabla^2 f_\phi(x) = \frac{1}{\tau}\nabla^2 f_0(x) + \nabla^2\phi(x) = \frac{1}{\tau}\nabla^2 f_0(x) + \sum_{i=1}^{m}(\frac{\nabla^2 f_i(x)}{f_i(x)^2} - 2\frac{\nabla f_i(x)^t \nabla f_i(x)}{f_i(x)^3})$$

In conclusion:

$$\nabla\phi(x) = \sum_{i=1}^{m}\frac{\nabla f_i(x)}{f_i(x)^2}$$

and

$$\nabla^2\phi(x) = \sum_{i=1}^{m}(\frac{\nabla^2 f_i(x)}{f_i(x)^2} - 2\frac{\nabla f_i(x)^t \nabla f_i(x)}{f_i(x)^3})$$

In this exercise we corrected the function $\phi(x)$ as in the references to have a Positive definite Hessian, we used $\phi = \sum_{i=1}^{m}\frac{-1}{f_i(x)}$ instead of $\phi = -\sum_{i=1}^{m}\frac{-1}{f_i(x)}$ The calculation will not be different than the log barrier, we only must calulate the new hessian and carry out the same steps as we showed in the previous pseudo-code. It is important keep in mind that the Value τ have to be changed once the inner while has reached the convergence until the duality gap was close to zero.

```
[ ]:
```