Convex Optimization - Homework 3

Name :Amer Essakine

email : amer.essakine@ens-paris-saclay.fr

Exercise 1

Consider the LASSO problem given by :

$$\text{minimize} \quad \frac{1}{2}||Xw - y||_2^2 - \lambda||w||_1$$

Let's derive its dual problem. First we formulate the LASSO problem as the equivalent problem :

$$\begin{aligned} \min_{z,w} \quad & \frac{1}{2}||z||_2^2 + \lambda||w||_1 \\ \text{s.t.} \quad & z = Xw - y \end{aligned}$$

We write the Lagrangian for this problem as :

$$\begin{aligned} \mathcal{L}(z,w,\nu) =& \frac{1}{2}||z||_2^2 + \lambda||w||_1 + \nu^T(z - Xw - y) \\ =& (\frac{1}{2}||z||_2^2 + \nu^T z) + (\lambda||w||_1 - (Xw)^T\nu) + \nu^T y \end{aligned}$$

Hence the dual function is given by :

$$\begin{aligned} g(\nu) &= \min_{z,w} \mathcal{L}(z,w,\nu) \\ &= \min_{z}(\frac{1}{2}||z||_2^2 + \nu^T z) + \min_{w}(\lambda||w||_1 - (Xw)^T\nu) + \nu^T y \end{aligned}$$

The function $h : z \rightarrow \frac{1}{2}||z||_2^2 + \nu^T z$ is convex and differentiable, and its gradient is given by :

$$\nabla h(z) = z + \nu$$

Hence it attains it minimum in $z = -v$ and the minumum is $\frac{1}{2}\|\nu\|_2^2$.

For the second function, we can write :

$$\min_w(\lambda\|w\|_1 - w^T X^T \nu) = \lambda(\|.\|)^*(\frac{1}{\lambda}X^T\nu)$$

Where $(\|.\|)^*$ is the conjugate function of the L1 norm,whose formula is given by the derivation from homework 2 :

$$(\|.\|)^*(\nu) = \begin{cases} 0 & \text{if } \ \|\nu\|_\infty \leq 1, \\ +\infty & \text{otherwise .} \end{cases}$$

Thus the dual problem is :

$$\begin{aligned} \max_v \quad & -\frac{1}{2}\|\nu\|_2^2 + \nu^T y \\ \text{s.t.} \quad & \|X^T\nu\|_\infty \leq \lambda \end{aligned}$$

However, we know that :

$$\begin{aligned} \|X^T\nu\|_\infty \leq \lambda & \Leftrightarrow \forall\, i\,,\ -\lambda \leq X^T\nu \leq \lambda \\ & \Leftrightarrow \forall\, i\,, X^T\nu \leq \lambda \text{ and } -X^T v \leq \lambda \\ & \Leftrightarrow \begin{pmatrix} X^T \\ -X^T \end{pmatrix} \nu \preceq \lambda I_{2d} \end{aligned}$$

Hence, the dual problem is equivalent to :

$$\begin{aligned} \min_v \quad & \nu^T Q \nu + p^T \nu \\ \text{s.t.} \quad & A\nu \preceq b \end{aligned}$$

Where :

$$Q = \frac{1}{2} I_n\,, p = -y\,, A = \begin{pmatrix} X^T \\ -X^T \end{pmatrix}\,, b = \lambda$$

Exercise 2

In [64]:
```
import numpy as np
import scipy as sc
import matplotlib.pyplot as plt
from scipy.linalg import inv
```

The goal of the centring step is to solve the unconstrained problem :

$$f_t(\nu) = t(\nu^T Q\nu + p^T\nu) - \sum_{i=1}^{2d} \log(b_i - (A\nu)_i)$$

The gradient of this function is given by :

$$\nabla f_t(\nu) = t(2Qv + p) + \sum_{i=1}^{2d} \frac{1}{b_i - (A\nu)_i} A_i \tag{1}$$

Where A_i is the i-th row of A. And the Hessian matrix is given by :

$$\nabla^2 f_t(\nu) = 2tQ + \sum_{i=1}^{2d} \frac{1}{(b_i - (A\nu)_i)^2} A_i^T A_i \tag{2}$$

Using this, let's first implement the centring step :

First let's define some useful function to be used in the two exercises

In [118…
```
def primal(w,X,y,lamb) :
  return 0.5*np.linalg.norm(X.T@w - y)**2 + lamb*np.linalg.norm(w,1)

def dual(v,Q,p,A,b) :
  return v.T @ Q @ v + p.T @ v

def f(Q,p,A,b,t,v) :
    return t*(v.T@Q@v + p.T@v) - np.sum(np.log(b - A@v))

def gradient(Q,p,A,b,t,v) :
```

```
  return t*(2*Q@v + p) - A.T @ (1/(b - A@v))

def hessian(Q,p,A,b,t,v) :
  weights = 1 / ((b - A@v)**2)
  return 2*t*Q + A.T @ np.diag(weights.flatten()) @ A

def line_search(Q,p,A,b,t,v,dv,alpha,beta,max_iter=100) :
  rate = 1
  iter = 0
  while f(Q,p,A,b,t,v+rate*dv) > f(Q,p,A,b,t,v) + alpha*rate*(dv.T@gradient(Q,p,A,b,t,dv)) and iter < max_iter  :
    rate *= beta
    iter +=1
  return rate
```

Then we implement the centring step :

In [119…

```
def centering_step(Q,p,A,b,t,v0,eps,max_iter=1000) :
  #line search parameters
  alpha = 0.5
  beta = 0.9

  #Initialization
  v_seq = [v0]
  v = v0
  iter= 0
  while iter < max_iter :
    hess = hessian(Q,p,A,b,t,v)
    grad = gradient(Q,p,A,b,t,v)
    dv = -inv(hess) @ grad
    rate = line_search(Q,p,A,b,t,v,dv,alpha,beta)
    v_new = v + rate*dv
    v_seq.append(v_new)
    lamb = - grad.T @ dv
    if (lamb < eps/2).all() :
      break
    v = v_new
    iter +=1
  return v_seq
```

Using the previous function, we implement the barrier method :

In [120…

```
def barr_method(Q,p,A,b,v0,eps,mu) :
  #Initialization
  t = 1
  v_seq = [v0]
  m = A.shape[0]
  v = v0

  while m/t > eps :
    v_new = centering_step(Q,p,A,b,t,v,eps)[-1]
    t = t*mu
    v_seq.append(v_new)
    v = v_new
  return v_seq
```

Now, let's generate some data to test our function :

In [121…

```
#Dimension parameters
d = 10
n = 10
alpha = 0.01
beta = 0.5

#Data
lamb = 10
X =  np.random.randn(n, d)
y =  np.random.randn(n,1)
t = 1

#LASSO dual parameters
Q = 0.5*np.eye(n)
p = -y
A = np.vstack((X.T,-X.T))
b = lamb*np.ones((2*d,1))
v0 = np.zeros((n,1))
eps = 1e-6
```

In [122…

```
plt.figure(figsize=(7,4))
v_secs = []
for mu in [2,15,50,100,300]:
    v_sec = barr_method(Q,p,A,b,v0,eps,mu)
```

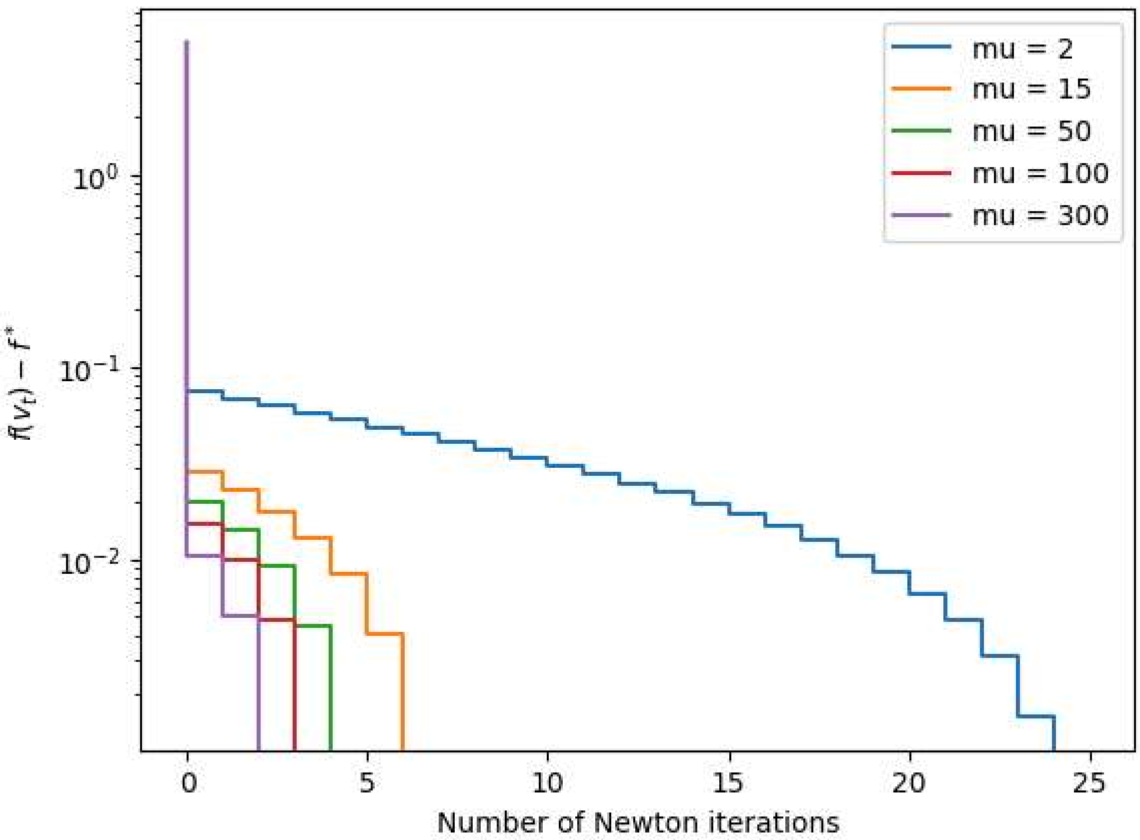
```
    v_secs.append(v_sec)
    print('done')
```

```
done
done
done
done
done
<Figure size 700x400 with 0 Axes>
```

In [125…

```
mu_values = [2,15,50,100,300]
for i in range(5):
    v_traj = v_secs[i]
    v_last = v_traj[-1]
    iters_newt = np.arange(len(v_traj))
    values = [(v0.T@Q@v0 + p.T.dot(v0))[0,0] - (v_last.T@Q@v_last + p.T.dot(v_last))[0,0] for v0 in v_traj]
    plt.step(iters_newt, values, label='mu = '+str(mu_values[i]))
plt.legend(loc = 'upper right')
plt.semilogy()
plt.xlabel('Number of Newton iterations')
plt.ylabel('$f(v_t)-f^*$')
plt.savefig("plot.eps")
plt.show()
```

```
WARNING:matplotlib.backends.backend_ps:The PostScript backend does not support transparency; partially transparent ar
tists will be rendered opaque.
```



We can see that the choice of μ significantly affects the convergence speed and stability of the barrier method. Smaller values of μ, such as $\mu = 2$, require more Newton iterations to achieve a desired precision, resulting in slower convergence. Conversely, larger values, such as $\mu = 100$ or $\mu = 300$, exhibit rapid convergence but may risk numerical instability or increased sensitivity to parameter tuning. A balanced choice, such as $\mu = 15$ or $\mu = 50$, provides an effective trade-off between convergence speed and stability, making them ideal for practical implementation.