Raphael Razafindralambo
raphael.razafin@gmail.com

Homework # 3
Convex Optimization 2023

11/20/23

Let's define $f : x \mapsto ||x||_1$. The conjugate function of f is:

$$f^*(y) = \sup_x y^T x - ||x||_1 = \begin{cases} 0 & \text{if } ||y||_\infty \leq 1 \\ +\infty & \text{otherwise} \end{cases} \tag{1}$$

Indeed if we denote by $||y||_* = \sup_{||x||_1 \leq 1} x^T y = \sup_{x \neq 0} \frac{x^T y}{||x||_1}$ for all y the dual norm of l_1 norm, then

- if $||y||_* \leq 1$, $\forall x \in \mathbb{R}^d_*$, $x^T y - ||x||_1 \leq 0 \implies \sup_x y^T x - ||x||_1 = 0$ since it's 0 maps to 0 and then $f^*(x) = 0$
- if $||y||_* > 1$, from Weierstrass' theorem the minimum of $x \mapsto x^T y$ is attained in some u with $||u|| \leq 1$, and so $u^T y = ||y||_*$. If $x = tu$ with $t > 0$ then $y^T x - ||x||_1 = t(y^T u - ||u||_1) = t(||y||_* - ||u||_1) \xrightarrow[t \longrightarrow +\infty]{} +\infty$, then $f^*(y) = +\infty$

We deduce the result from $||y||_* = ||y||_\infty$. Let's show that $\forall y \in \mathbb{R}^d, \ \sup_{||x||_1 \leq 1} x^T y = ||y||_\infty$.

- $\forall x$ s.t $||x||_1 \leq 1, \ x^T y \leq \sum_{i=1}^d |x_i||y_i| \leq \max\{|y_i|\} \sum_{i=1}^d |x_i| \leq \max\{|y_i|\} = ||y||_\infty$.
- Let's chose $x = sign(y_j)e_j$ where $(e_1, \ldots, e_d)$ is the canonical base of $\mathbb{R}^d$ and $j = \arg\max |y_i|$. We have $||x||_1 = 1$ and $x^T y = y^T sign(y_j)e_j = |y_j| = ||y||_\infty$.

1 Question 1

We consider the following problem:

$$\min_{w \in \mathcal{R}^d} \frac{1}{2}||Xw - y||_2^2 + \lambda||w||_1$$

It can be reformuled as

$$\min_{w, z = Xw - y} \frac{1}{2}||z||_2^2 + \lambda||w||_1$$

The Lagrangian function of this problem is, for all (w, z, ν) in $\mathbb{R}^d \times \mathbb{R}^n \times \mathbb{R}^n$:

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

Let's compute the dual function. It is

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

For the first infimum, we have

$$\begin{aligned} \inf_w \nu^T Xw + \lambda||w||_1 &= \inf_w (X^T\nu)^T w + \lambda||w||_1 \\ &= -\sup_w -(X^T\nu)^T w - \lambda||w||_1 \\ &= -\lambda \sup_w -(\frac{1}{\lambda}X^T\nu)^T w - ||w||_1 \\ &= -\lambda f^*(-\frac{1}{\lambda}X^T\nu) \\ &= \begin{cases} 0 & \text{if } \frac{1}{\lambda}||X^T\nu||_\infty \leq 1 \\ -\infty & \text{otherwise} \end{cases} \end{aligned}$$

where f^* is the conjugate of the $||.||_1$ norm.

For the second infimum, since $z \mapsto \frac{1}{2}||z||_2^2 - \nu^T z$ is twice differentiable and

$$\begin{cases} \nabla_z \frac{1}{2}||z||_2^2 - \nu^T z = 0 \iff z = \nu \\ \nabla_z^2 \frac{1}{2}||z||_2^2 - \nu^T z = 2I_n \succeq 0 \text{ (convexity)} \end{cases}$$

then the minimum is attained in $z = \nu$ and

$$\inf_z \frac{1}{2}||z||_2^2 - \nu^T z = -\frac{1}{2}\nu^T \nu.$$

We deduce that

$$g(\nu) = \begin{cases} -\frac{1}{2}\nu^T\nu - \nu^T & \text{if } ||X^T\nu||_\infty \leq \lambda \\ -\infty & \text{otherwise} \end{cases}.$$

Thus, a dual problem is

$$\max_{||X^T\nu||_\infty \leq \lambda} -\frac{1}{2}\nu^T\nu - \nu^T y$$

which has the same solutions as

$$\min_{||X^T\nu||_\infty \leq \lambda} \frac{1}{2}\nu^T\nu + \nu^T y.$$

Let's rewrite the constraints. if we pose

$$A = \begin{pmatrix} X^T \\ -X^T \end{pmatrix}, \quad b = \begin{pmatrix} \lambda \\ \lambda \\ \vdots \\ \lambda \end{pmatrix} \in \mathbb{R}^{2d}$$

we retrieve

$$A\nu \preceq b \iff \begin{pmatrix} X^T \\ -X^T \end{pmatrix} \nu \preceq \begin{pmatrix} \lambda \\ \lambda \\ \vdots \\ \lambda \end{pmatrix} \iff -\lambda < (X^T\nu)_i < \lambda \; \forall i \in [\![1; 2n]\!] \iff ||X^T\nu||_\infty \leq \lambda$$

Thus, we retrieve the quadratic problem (QP) with

$$Q = \frac{1}{2}I_n, \quad A = \begin{pmatrix} X^T \\ -X^T \end{pmatrix}, \quad b = \lambda \mathbf{1}_{2d}$$

2 Question 2

(QP) problem is:

$$\min_{A\nu \preceq b} \nu^T Q \nu + p^T \nu \tag{2}$$

The associated centering problem is

$$\min_\nu t(\nu^T Q\nu + p^T\nu) - \sum_{i=1}^m log(-(A\nu - b)_i) \tag{3}$$

We need to compute the gradient. $\mu \mapsto t(\nu^T Q\nu + p^T\nu) - \sum_{i=1}^m log(-(A\nu - b)_i)$ is twice differentiable and the gradient is

$$t(Q + Q^T)\nu + tp - \sum_{i=1}^m \frac{A_i}{(A\nu - b)_i} \tag{4}$$

The hessian is equal to

$$t(Q + Q^T) + \sum_{i=1}^m \frac{A_i^T A_i}{(A\nu - b)_i^2}. \tag{5}$$

We want to solve the centering problem with Newton algorithm. It's an unconstrained minimization.
In order to solve the barrier problem, we use these settings and hyper-parameters:

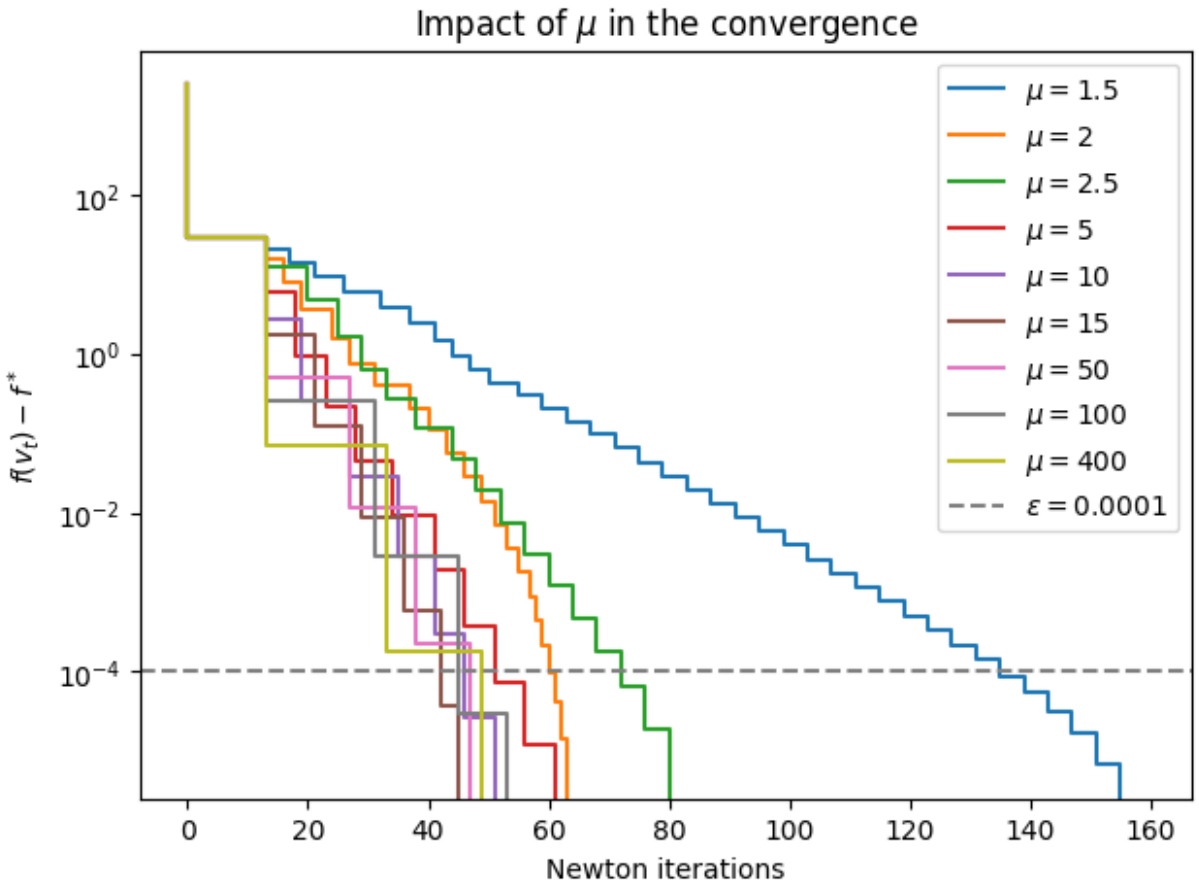
- $n = 30 \ll d = 100$ ($m = 2d = 200$).
- Q,A, b given by the question 1 ($p = y$).
- Initializations $v_0 = 0_{\mathbb{R}^n}$, $t = 1$.
- $\alpha = 0.1$, $\beta = 0.5$.

We observe from the plots that the choice of μ involves a trade-off: large μ means fewer outer iterations, more inner (Newton) iterations. For exemple, for $\mu = 2$ we have much more outer iterations but fewer inner iterations. The reverse is observed for $\mu = 400$.
The total number of steps experiences a significant reduction for minor adjustments in the value of μ up to approximately 15, after which the impact of these changes becomes progressively less pronounced.
We chose μ which minimize this total number of iterations (inner + outer) without being too high, which seems to be $\mu = 15$ according to the plots.

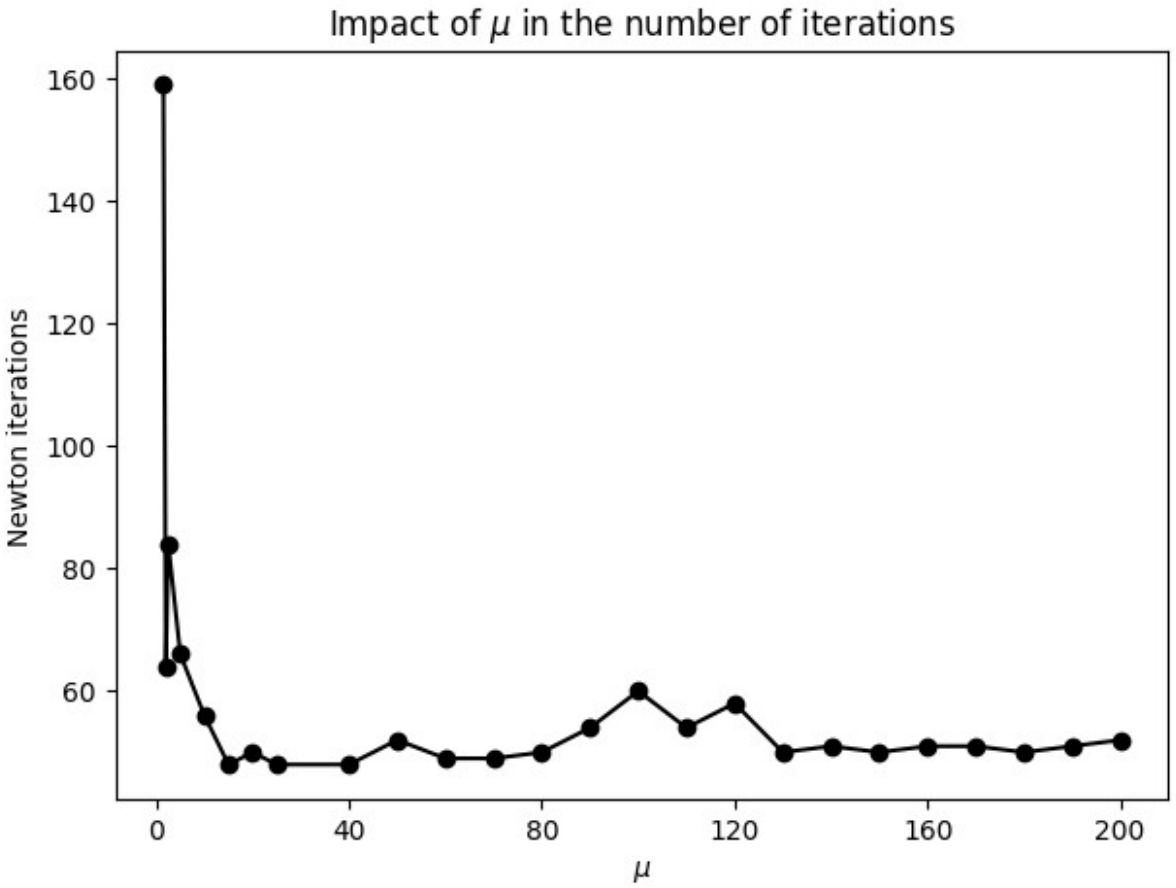
```
import numpy as np
import scipy.optimize
import matplotlib.pyplot as plt
import sklearn.linear_model
import sympy as sp
from sklearn.datasets import make_regression

def QP(v,Q,p):
  return v.T@(Q@v) + p.T@v

def f_barrier(v,Q,p,t,A,b):
  if not (-A@v+b>0).all():
        raise Exception('The problem is not feasible.')
  return t*QP(v,Q,p) - sum(np.log(-A@v+b))

def grad_barrier(v,Q,p,t,A,b):
  return t*(Q+Q.T)@v + t*p - np.sum(A.T*(1/(A@v-b)), axis=1)

def hessian_barrier(v,Q,p,t,A,b):
  diff = A@v-b
  return t*(Q.T+Q) + sum([1/(diff[i])**2 * A[i,:].reshape(-
1,1)@A[i,:].reshape(1,-1) for i in range(A.shape[0])])


def delta_newton(grad,hess):
  return -np.linalg.inv(hess)@grad


def line_search(v,Q,p,t,A,b,delta,grad,alpha=0.01,beta=0.5):
    step = 1
    while not(((-A@(v+step*delta)+b)>0).all()) or
(f_barrier(v+step*delta,Q,p,t,A,b) >= f_barrier(v,Q,p,t,A,b) +
alpha*step*(grad.T@delta)):
      step *= beta
    return step



def centering_step(Q,p,A,b,t,v0,eps=1e-3):
    '''implements the Newton method to solve the centering step given
the inputs (Q; p; A; b), the
    barrier method parameter t (see lectures), initial variable v0 and
a target precision
    eps. The function outputs the sequence of variables iterates
(vi)i=1;:::;n , where n_eps is
    the number of iterations to obtain the epsilon precision. Use a
backtracking line search
    with appropriate parameters.'''
    v_seq = [v0]
    v = np.copy(v0)
```

```
    alpha = 0.1
    beta = 0.5
    it = 0
    while True:


      #1 Compute the Newton step and decrement
      grad,hess=grad_barrier(v,Q,p,t,A,b),hessian_barrier(v,Q,p,t,A,b)
      delta = delta_newton(grad,hess)
      lambda2 = -grad.T@delta

      #2 Stopping criterion
      if lambda2/2 <= eps: break

      #3 Line search
      step = line_search(v,Q,p,t,A,b,delta,grad,alpha,beta)

      #4 Update
      it += 1
      v = np.copy(v+step*delta)
      v_seq.append(v)

    return v_seq,it

def barr_method(Q,p,A,b,v0,mu,eps=1e-3):
    '''implements the barrier method to solve QP using precedent
function given the data inputs (Q, p, A, b),
    a feasible point v0, a precision criterion eps. The function
outputs the sequence of
    variables iterates (vi)i=1,...,n_eps , where n_eps is the number
of iterations to obtain the eps precision.'''
    v_seq = [v0]
    v = np.copy(v0)
    t = 1
    m = len(b)
    iters = [0]
    while True:

      #1 Centering step
      vs,it = centering_step(Q,p,A,b,t,v,eps)
      v = vs[-1]
      iters.append(it+iters[-1])

      #2 Update
      v_seq.append(v)
      #3 Stopping criterion
      if m/t < eps: break

      #4 Increase t
```

```
      t *= mu
    return v_seq,iters

n = 30
d = 100
X,y =
make_regression(n_samples=n,n_features=d,noise=1,random_state=42)
Q = (1/2)*np.eye(n); p = y
A = np.vstack((X.T,-X.T)); b = 5*np.ones(2*d)
v0 = np.zeros(n)
eps = 1e-4

plt.figure(figsize=(7,5))
mus = [1.5,2, 2.5, 5, 10,15, 50, 100,400]
for mu in mus:
    v_seq, iters = barr_method(Q,p,A,b,v0,mu,eps)
    #print(iters)
    fs = np.array([QP(v, Q, p) for v in v_seq])
    plt.step(iters, fs - fs[-1], label='$\mu = $' + str(mu))
    plt.semilogy()
plt.axhline(y=eps, color='grey', linestyle='--',label="$\epsilon = $"
+ str(eps))
plt.xlabel('Newton iterations')
plt.ylabel('$f(v_t)-f^*$')
plt.title('Impact of $\mu$ in the convergence')
plt.legend()
plt.show()
```




```
plt.figure(figsize=(7,5))
mus =
[1.5,2,2.5,5,10,15,20,25,40,50,60,70,80,90,100,110,120,130,140,150,160
,170,180,190,200]
iters_total = [barr_method(Q,p,A,b,v0,mu,eps)[1][-1] for mu in mus]
plt.plot(mus, iters_total, 'o-',c="black")  # 'o-' pour des lignes
avec des ronds aux points
plt.xticks(np.arange(0, 201, 40))  # Définit les labels de l'axe des x
de 0 à 100 avec un pas de 10
plt.xlabel('$\mu$')
plt.ylabel('Newton iterations')
plt.title('Impact of $\mu$ in the number of iterations')
plt.show()
```


Impact of μ in the number of iterations
Newton iterations
160
140
120
100
80
60
0
40
80
120
160
200
μ