Convex Optimization
Homework 3 — November 29

Justin AYIVI

justin.ayivi@yahoo.com

Homework 3 – November 29th

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

$X=(x_1^T,\dots,x_n^T)\in\mathbb{R}^{n\times d}$, $y=(y_1,\dots,y_n)\in\mathbb{R}^n$

1) Trouvons le dual de la fonction $f(x)=\|x\|_1$. $f=\|\cdot\|_1$

$$f^*(y)=\sup_{x\in \operatorname{dom} f}\left(y^Tx-f(x)\right)=\sup_{x\in\operatorname{dom}(f)}\left(y^Tx-\|x\|_1\right)$$

$$=\sup_{x\in\operatorname{dom} f}\left(y^Tx-\sum_{i=1}^{d}|x_i|\right)=\sup_{x\in\operatorname{dom} f}\left(\sum_{i=1}^{d}y_ix_i-\sum_{i=1}^{d}|x_i|\right)$$

On avait déjà prouvé dans le Homework 2 que cette fonction s'écrivait ainsi :

$$f^*(y)=\begin{cases}0 & \text{si } \|y\|_\infty\leq 1\\ +\infty & \text{sinon}\end{cases}$$

Revenons maintenant à notre problème d'optimisation

$$\min_{w}\frac{1}{2}\|Xw-y\|_2^2+\lambda\|w\|_1 \iff \min_{w}\frac{1}{2}\|z\|^2+\lambda\|w\|_1 \quad \text{s/c } Xw-y=z$$

Soit $\nu\in\mathbb{R}^n$, $z\in\mathbb{R}^n$, $w\in\mathbb{R}^d$

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

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

$$=\frac{1}{2}\|z\|_2^2+\nu^Tz+\lambda\|w\|_1-(X^T\nu)^Tw+\nu^Ty$$

et la fonction duale s'écrit

$$g(\gamma) = \inf_{w, z} \mathcal{L}(w, z, \gamma)$$

$$= y^T\gamma + \inf_{z}\left(\frac{1}{2}\|z\|_2^2 + \gamma^T z\right) + \inf_{w}\left(\lambda\|w\|_1 - (X^T\gamma)^T w\right)$$

La fonction $h : z \longmapsto \frac{1}{2}\|z\|_2^2 + \gamma^T z$ est convexe et différentiable.
Son gradient est donné par $\nabla h(z) = z + \gamma$.

$\nabla h(z) = 0 \iff z = -\gamma$ alors le minimum de la fonction h est atteint pour $z = -\gamma$

$$h(z) = h(-\gamma) = \frac{1}{2}\|-\gamma\|_2^2 + \gamma^T(-\gamma) = \frac{1}{2}\|\gamma\|_2^2 - \|\gamma\|^2 = -\frac{1}{2}\|\gamma\|_2^2.$$

$$\inf_{z}\left(\frac{1}{2}\|z\|_2^2 + \gamma^T z\right) = -\frac{1}{2}\|\gamma\|_2^2 \quad (1)$$

Trouvons maintenant $\inf_{w}\left(\lambda\|w\|_1 - (X^T\gamma)^T w\right)$

$$\inf_{w}\left(\lambda\|w\|_1 - (X^T\gamma)^T w\right) = -\sup_{w}\left((X^T\gamma)^T w - \lambda\|w\|_1\right)$$

$$= -\sup_{w} \lambda\left(\frac{1}{\lambda}(X^T\gamma)^T w - \|w\|_1\right)$$

$$= -\sup_{w} \lambda\left[\left(\frac{1}{\lambda} X^T\gamma\right)^T w - \|w\|_1\right]$$

$$= \lambda\left(-\sup_{w} \lambda\left[\left(\frac{1}{\lambda} X^T\gamma\right)^T w - \|w\|_1\right]\right)$$

$$= -\lambda f^*\left(\frac{1}{\lambda} X^T\gamma\right)$$ où f^* désigne la fonction conjugée.

$$\inf_w \left(\lambda \|w\|_1 - (X^T\gamma)^T w\right) = \begin{cases} 0 \text{ si } \|\frac{1}{\lambda} X^T\gamma\|_\infty \leq 1 \\ -\infty \text{ sinon} \end{cases}$$

$$= \begin{cases} 0 \text{ si } \|X^T\gamma\|_\infty \leq \lambda \\ -\infty \text{ sinon} \end{cases}$$

Donc le problème dual Lasso est le suivant :

$$\boxed{\begin{cases} \max_\gamma y^T\gamma - \frac{1}{2}\|\gamma\|_2^2 \\ s/c \ \|\frac{1}{\lambda} X^T\gamma\|_\infty \leq 1 \end{cases} \text{ ie } \begin{cases} \max_\gamma \frac{1}{2} y^T\gamma - \frac{1}{2}\|\gamma\|_2^2 \\ s/c \ \|X^T\gamma\|_\infty \leq \lambda \end{cases}}$$

Montrons que cela peut se réécrire sous la forme quadratique suivante :

$$\min v^T Q v + p^T v$$
$$s/c \ Av \leq b$$

$$\left\|\frac{X^T\gamma}{\lambda}\right\|_\infty \leq 1 \iff \forall i \in [\![1, n]\!], \ -1 \leq \left[\frac{1}{\lambda} X^T\gamma\right]_i \leq 1$$

$$\iff \forall i \in [\![1, n]\!] \ \left[\frac{1}{\lambda} X^T\gamma\right]_i \leq 1 \text{ et } \left[-\frac{1}{\lambda} X^T\gamma\right]_i \leq 1$$

$$\iff A\gamma \leq \lambda \mathbb{1}_{2d}$$

avec $A = \begin{pmatrix} X^T \\ -X^T \end{pmatrix}$ et $b = (\lambda, \lambda, \dots, \lambda)^T$

En a donc $\max_\gamma y^T\gamma - \frac{1}{2}\|\gamma\|_2^2 = \max_\gamma -\frac{1}{2}\|\gamma\|_2^2 + y^T\gamma$

$$= \max_\gamma \gamma^T\left(-\frac{1}{2} I_n\right)\gamma + y^T\gamma$$

$$= \max_\gamma -\left[-y^T\gamma - \gamma^T\left(-\frac{1}{2} I_n\right)\gamma\right]$$

$$\max_{\gamma} y^T\gamma - \frac{1}{2}\|\gamma\|_2^2 = \max_{\gamma}\left[-\left(-y^T\gamma - \gamma^T\left(-\frac{1}{2}I_n\right)\gamma\right)\right]$$

$$= \min_{\gamma} \gamma^T\left(\frac{1}{2}I_n\right)\gamma - y^T\gamma$$

Par identification, on a : $Q = \frac{1}{2}I_n$ et $p = -y$.

En somme,

$$\begin{cases} \max_{\gamma} -\frac{1}{2}\|\gamma\|_2^2 + y^T\gamma \\ s/c \ \|\frac{1}{\lambda}X^T\gamma\|_\infty \leq 1 \end{cases} \text{ est équivalent à } \begin{cases} \min_{\gamma} \gamma^T Q\gamma + p^T\gamma \\ s/c \ A\gamma \leq b \end{cases}$$

Avec $A = \begin{pmatrix} X^T \\ -X^T \end{pmatrix} \in \mathbb{R}^{2d\times n}$, $Q = \frac{1}{2}I_n \in \mathbb{R}^n$

$b = (\underbrace{\lambda, \dots, \lambda}_{2d}) \in \mathbb{R}^{2d}$ et $p = -y \in \mathbb{R}^n$

$\gamma \in \mathbb{R}^n$

2 - Implementation de la methode de Barrier pour résoudre le problème QP.

La methode de barrier consiste à réécrire la ~~fonction type~~ problème d'optimisation avec contraintes sous la forme d'un problème d'optimisation sans contraintes.

Ainsi le problème d'optimisation $\min_{v} v^T Q v + p^T v$ est
$s.c \quad Av \leq b$

équivalent au problème

$$\boxed{\min_{v} v^T Q v + p^T v + \sum_{i=1}^{2d} -\left(\frac{1}{t}\right)\log\left(b_i - [Av]_i\right) =}$$

$$\min_{v} v^T Q v + p^T v + \sum_{i=1}^{2d} -\frac{1}{t}\log\left(b_i - a_i^T v\right)$$

Si nous considérons $A = \begin{pmatrix} a_1^T \\ \vdots \\ a_{2d}^T \end{pmatrix}$

Posons $\phi(v) = -\sum_{i=1}^{2d} \log\left(b_i - a_i^T v\right) =$

$\mathrm{dom}\,\phi = \{v \mid Av \preceq b\}$. Calculons le hessien et le gradient de la fonction ϕ.

$$\nabla\phi(v) = \frac{\partial\phi(v)}{\partial v} = \sum_{i=1}^{d} \frac{a_i}{b_i - a_i^T v}$$

$$\nabla^2\phi(v) = \frac{\partial^2\phi(v)}{\partial v^2} = \sum_{i=1}^{d} \frac{a_i a_i^T}{\left(b_i - a_i^T v\right)^2}$$

Donc on ~~on~~ a en resumé

$$\nabla\phi(\gamma) = \sum_{i=1}^{2d} \frac{a_i}{b_i - a_i^T\gamma} = A^T d$$ avec les element de $d \in \mathbb{R}^{2d}$

qui sont obtenus par $d_i = \dfrac{1}{b_i - a_i^T\gamma}$. $d \in \mathbb{R}^{2d}$

$$\nabla^2\phi(\gamma) = \sum_{i=1}^{2d} \frac{a_i a_i^T}{(b_i - a_i^T\gamma)^2} = \sum_{i=1}^{2d} d_i^2 a_i a_i^T = A^T[\mathrm{diag}(d)]^2 A$$

Le problème d'optimisation initial s'écrivait.

$$\min_\gamma \left[\gamma^T Q\gamma + p^T\gamma + \frac{1}{t}\sum_{i=1}^{2d} -\log(b_i - a_i^T\gamma)\right]$$

$$= \min_\gamma \left[\gamma^T Q\gamma + p^T\gamma + \frac{1}{t}\phi(\gamma)\right]$$

$$= \min_\gamma \left[f_0(\gamma) + \frac{1}{t}\phi(\gamma)\right]$$. avec $f_0(\gamma) = \gamma^T Q\gamma + p^T\gamma$.

$$\nabla f_0(\gamma) = \frac{\partial}{\partial\gamma}\left[\gamma^T Q\gamma + p^T\gamma\right] = (Q + Q^T)\gamma + p = 2Q\gamma + p$$

$$\nabla^2 f_0(\gamma) = \frac{\partial^2 f_0(\gamma)}{\partial\gamma^2} = 2Q .$$

et

$$\nabla\phi(\gamma) = A^T d$$

$$\nabla^2\phi(\gamma) = A^T[\mathrm{diag}(d)]^2 A$$

Ainsi,

Si on pose

$$h(\gamma) = \min_{\gamma}\left[\left(\gamma^T Q \gamma + p^T\gamma + \frac{1}{t}\phi(\gamma)\right)\times t\right]$$

$$= \min_{\gamma}\left[\left(\gamma^T Q\gamma + p^T\gamma\right)t + \phi(\gamma)\right] \quad \text{Alors}$$

$$\nabla h(\gamma) = t\,\nabla f_0(\gamma) + \nabla\phi(\gamma) = (2Q\gamma + p)t + A^T d$$

$$\nabla^2 h(\gamma) = t\,\nabla^2 f_0(\gamma) + \nabla^2\phi(\gamma) = 2Qt + A^T[\mathrm{diag}(d)]^2 A$$

$$\boxed{\begin{aligned} \nabla h(\gamma) &= (2Q\gamma + p)t + A^T d \\ \nabla^2 h(\gamma) &= 2Q.t + A^T[\mathrm{diag}(d)]^2 A. \end{aligned}}$$

In [ ]:

```
import numpy as np
import math
from time import time
import matplotlib.pyplot as plt
from sklearn.linear_model import Lasso
```

Question 1 up to Question 2 : Preliminaries

Write the constrained problem minimization

$$\begin{cases} \min_{\nu} \left(\nu^T Q \nu + p^T \nu\right) \\ s.t. A\nu \preceq b \end{cases}$$

as unconstrained problem minimization

$$\min_{\nu} g_t(\nu)$$

$$\triangleq t\left(\nu^T Q \nu + p^T \nu\right) -$$

$$\sum_{i=1}^{2d} \log(b_i - [A\nu]_i)$$

In [ ]:

```
# objective without constraint
def dual(v, Q, p):
  return v.T.dot(Q).dot(v) + p.T.dot(v)
```

In [ ]:

```
class PbUnConstraint(object):
  def __init__(self, Q, p, A, b, t):
    self.Q = Q
    self.p = p
    self.A = A
    self.b = b
    # self.D, self.N = A.shape # N= n et D = 2d
    self.p = p
    self.t = t

  # objective without constraint
  def obj(self, v):
    return v.T.dot(Q).dot(v) + p.T.dot(v)

  # objective with constraint
  def barobj(self, v):
    obj_function = self.t * (v.T.dot(Q).dot(v) + p.T.dot(v))
    if (b - np.dot(A, v) <= 0).any():
      return float("NaN")
    else :
      constraint_function = A.shape[0] * np.mean(np.log(b - np.dot(A, v))) # on enlève 1/
2d
      return obj_function - constraint_function


  # gradient
  def grad(self, v):
    phi = (1. / (b - np.dot(A, v)))
    # diagm = diag * np.eye(A.shape[0])
    return (2 * np.dot(Q, v) + p) * self.t + (A.T).dot(phi)

  # hessian
  def hess(self, v):
    phi = b - np.dot(A, v)
```

```
    # diag = 1 / (d**2)
    # diagm = diag * np.eye(A.shape[0])
    diagm = np.diag(phi)**2
    return 2 * Q * self.t + (A.T).dot(diagm).dot(A)
```

In [ ]:

```
def lineSearch(f, df, v, dv, alpha, beta):
  t = 1
  while (np.isnan(f(v + t * dv)) or f(v + t * dv) >=  f(v) + alpha * t * (df(v).T.dot(dv
))) and ( t > 1e-6):
    t *= beta
    if np.any(b - np.dot(A, v + t * dv) <= 0):
            return t
  return t
```

Question 2

Write a function v_seq which implements the Newton method to solve the centering step given the inputs (Q, p, A, b), the barrier method parameter t (see lectures), initial variables $v0$ and a target precision \epsilon. The function outputs the sequence of variables iterates (v_i), $i = 1, \ldots, n_\epsilon$, where n_ϵ is the number of iterations to obtain the ϵ precision. Use a backtracking line search with appropriate parameters

In [ ]:

```
def centering_step(v0, Q, p, A, b, t, eps=1e-9, alpha=.5, beta=.9, max_iter=500):
  v_seq = [v0]
  i=0
  v = v0

  # Class instanciate
  pb = PbUnConstraint(Q, p, A, b, t)
  # to simplify notations
  f = lambda v :  pb.barobj(v)
  df = lambda v : pb.grad(v)
  dfdf = lambda v : pb.hess(v)

  while i < max_iter :

    # Newton method
    dv = np.linalg.pinv(dfdf(v)).dot(df(v))
    lambda2 = df(v).T.dot(dv)
    if ((0.5 * lambda2) <= eps).any():
      break

    # step size by backtracking line search
    t = lineSearch(f, df, v, dv, alpha = alpha, beta = beta)
    v = v - t * dv
    v_seq.append(v)
    i+=1

  return v_seq
```

Write a function v_seq which implements the barrier method to solve QP using precedent function given the data inputs (Q, p, A, b), a feasible point v_0, a precision criterion ϵ. The function outputs the sequence of variables iterates (v_i), $i = 1, \ldots, n_\epsilon$ where n_ϵ is the number of iterations to obtain the ϵ precision

In [ ]:

```
def barr_method(v0, Q, p, A, b, mu, eps = 1e-9, max_iter=500):
```

```
    #Initialization
    v = v0
    v_seq = [v0]
    m = A.shape[0]
    t = 1

    # Class instanciate
    pb = PbUnConstraint(Q, p, A, b, t)
    # to simplify notations
    f = lambda v : pb.obj(v) # Centering step Barrier method
    df = lambda v : pb.grad(v)
    dfdf = lambda v : pb.hess(v)


    # centering step
    while (m / t) >= eps:
      v = centering_step(v_seq[-1], Q, p, A, b, t, eps=1e-9, alpha=.5, beta=.9, max_iter=50
0)[-1]
      v_seq.append(v)
      t *= mu

    return v_seq
```

Question 3

Test your function on randomly generated matrices X and observations y with $\lambda = 10$. Plot precision criterion and gap $f(v_t) - f^*$ in semilog scale (using the best value found for f as a surrogate for f^*). Repeat for different values of the barrier method parameter $\mu = 2, 15, 50, 100 \ldots$ and check the impact on w. What would be an appropriate choice for μ?

In [ ]:

```
def make_data(n, d, lamda=10):

    X = 3 * np.random.randn(n, d)
    y = 5 + 1.5 * np.random.randn(n)

    Q = np.eye(n) / 2
    p = - y

    A = np.concatenate((X.T, - X.T), axis=0)
    b = lamda * np.ones(2 * d)

    v0 = np.zeros(n)

    return X, y, Q, p, A, b, v0
```

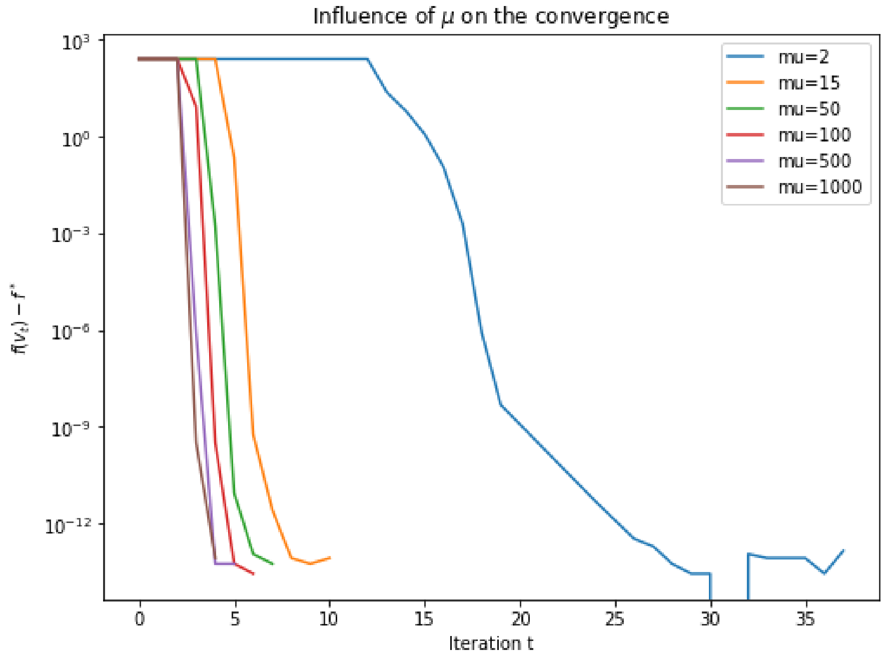
In [ ]:

```
n, d, lamda = 20, 50, 10
X, w,  y, Q, p, A, b, v0 = make_data(n, d, lamda)
eps = 1e-9
alpha, beta = .5, .9
max_iter = 500

mu_values = [2, 15, 50, 100, 500, 1000]
results = [barr_method(v0, Q, p, A, b, mu, eps = 1e-9, max_iter=500) for mu in mu_values]
f_values = [[dual(v, Q, p) for v in results[i]] for i in range(len(results))]
f_star = np.infty
for i in range(len(results)):
    for v in f_values[i]:
        if f_star > v:
            f_star = v

plt.figure(figsize=(8, 6))
plt.xlabel('Iteration t')
plt.ylabel('$f(v_t) - f^*$')
plt.title('Influence of $\\mu$ on the convergence')
```

```
for i in range(len(results)):
    plt.semilogy(f_values[i] - f_star, label='mu={}'.format(mu_values[i]))
plt.legend()
plt.show()
```



We see that the number of iterations increases if μ decreases. We also notice that that the different initializations give almost the same result f^*, except for $\mu = 2$, which gives a better result than the others.