CONVEX OPTIMIZATION

HOMEWORK 3

1. We have the following problem:

$$(\text{LASSO}) \qquad \min_{w} \ \frac{1}{2}\|Xw - y\|_2^2 + \lambda \|w\|_1$$

where $w \in \mathbb{R}^d$, $X = (x_1^T, \ldots, x_n^T) \in \mathbb{R}^{n \times d}$, $y = (y_1, \ldots, y_n) \in \mathbb{R}^n$ and $\lambda > 0$ is a regularization parameter.

In order to derive the dual problem of LASSO, we introduce a new variable in the problem, as follows:

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

The lagrangian of (LASSO) is given by:

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

It follows that the dual function is given by:

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

* We first minimize over a: it is patent that the function $[a \longmapsto \frac{1}{2}\|a\|_2^2 + \nu^T a]$ is a convex function (since it is the sum of 2 convex functions). Moreover, the function $[a \longmapsto \frac{1}{2}\|a\|_2^2 + \nu^T a]$ is differentiable.

Therefore we minimize over a by setting the gradient of L with respect to a equal to zero, that is:

$$\nabla_a L(w,a,\nu) = a + \nu = 0 \iff a = -\nu$$

* Now we minimize over ω : $\inf_{\omega} \{ \lambda \|\omega\|_1 - (X^T\nu)^T \omega \}$

$$B = \inf_{\omega} \{ \lambda \|\omega\|_1 - (X^T\nu)^T \omega \}$$ since $\lambda > 0$, we have :

$$= \inf_{\omega} \{ \lambda (\|\omega\|_1 - \frac{1}{\lambda} (X^T\nu)^T \omega) \}$$

$$= \lambda \inf_{\omega} \{ \|\omega\|_1 - \frac{1}{\lambda} (X^T\nu)^T \omega \}$$

$$= -\lambda \sup_{\omega} \{ (\frac{1}{\lambda} X^T\nu)^T \omega - \|\omega\|_1 \}$$

$$= -\lambda \|.\|_1^* (\frac{1}{\lambda} X^T \nu)$$ where $\|.\|_1^*$ is the conjugate function of $\|.\|_1$.

From homework 2 we know that the conjugate of $\|.\|_1$ is given by:

$$\forall x \in \mathbb{R}^n, \quad \|.\|_1^*(x) = \begin{cases} 0 & \text{if } \|x\|_\infty \leq 1 \\ +\infty & \text{otherwise} \end{cases}$$

Therefore we obtain :

$$B = \inf_{\omega} \{ \lambda \|\omega\|_1 - (X^T\nu)^T \omega \}$$

$$B = \begin{cases} 0 & \text{if } \| \frac{1}{\lambda} X^T \nu \|_\infty \leq 1 \\ -\infty & \text{otherwise} \end{cases}$$

The dual function is therefore given by:

$$g(\nu) = \begin{cases} \frac{1}{2} \|-\nu\|_2^2 + \nu^T(-\nu) + \nu^T y & \text{if } \| \frac{1}{\lambda} X^T \nu \|_\infty \leq 1 \\ -\infty & \text{otherwise} \end{cases}$$

$$= \begin{cases} \frac{1}{2} \|\nu\|_2^2 - \|\nu\|_2^2 + \nu^T y & \text{if } \| \frac{1}{\lambda} X^T \nu \|_\infty \leq 1 \\ -\infty & \text{otherwise} \end{cases}$$

$$g(\nu) = \begin{cases} -\frac{1}{2} \|\nu\|_2^2 + \nu^T y & \text{if } \| \frac{1}{\lambda} X^T \nu \|_\infty \leq 1 \\ -\infty & \text{otherwise} \end{cases}$$

HOMEWORK 3

Hence, the dual problem of LASSO is:

(Dual)
$$\boxed{\begin{array}{ll} \underset{\nu}{\text{Max}} & -\frac{1}{2}\|\nu\|_2^2 + y^T\nu \\ \text{s.t.} & \|\frac{1}{\lambda}X^T\nu\|_\infty \leq 1 \end{array}}$$

Which can be written as:

(Dual)
$$\boxed{\begin{array}{ll} \underset{\nu}{\text{Min}} & \nu^T Q \nu + p^T\nu \\ \text{s.t.} & \|\frac{1}{\lambda}X^T\nu\|_\infty \leq 1 \end{array}}$$

where $Q = \frac{1}{2}I_n \succeq 0$ and $p = -y$

N.B.: $\|\frac{1}{\lambda}X^T\nu\|_\infty \leq 1 \iff \max\limits_{1\leq i\leq d} |\frac{1}{\lambda}(X^T\nu)_i| \leq 1$

$\iff \forall i = 1,\dots,d, \quad |\frac{1}{\lambda}(X^T\nu)_i| \leq 1$

$\iff \forall i = 1,\dots,d \quad -1 \leq \frac{1}{\lambda}(X^T\nu)_i \leq 1$

$\iff \forall i = 1,\dots,d \quad -\lambda \leq (X^T\nu)_i \leq \lambda$

$\iff \forall i = 1,\dots,d, \quad -(X^T\nu)_i \leq \lambda$ and $(X^T\nu)_i \leq \lambda$

$\iff A\nu \leq \lambda \times 1_{2d}$ where $A = \begin{pmatrix} -X^T \\ X^T \end{pmatrix}$

Hence, if we denote $A = \begin{pmatrix} -X^T \\ X^T \end{pmatrix}$, $b = \lambda \times 1_{2d}$, $Q = \frac{1}{2}I_n \succeq 0$ and $p = -y$,
we have that the dual of (LASSO) is:

(Dual)
$$\boxed{\begin{array}{ll} \underset{\nu}{\text{Min}} & \nu^T Q \nu + p^T\nu \\ \text{s.t.} & A\nu \leq b \end{array}} \quad (QP)$$

2. Our problem (QP) has no equality constraint.

If we define $f_t(v) = t(v^TQv + p^Tv) - \sum_{i=1}^{2d} \log(b_i - A_i^Tv)$

where A_i is the i-th row of matrix A, ie: $A = \begin{pmatrix} — A_1^T — \\ \vdots \\ — A_{2d}^T — \end{pmatrix}$

we have that the centering problem is: $\min_v f_t(v)$

The goal of centering step is to solve this problem.

→ In order to do this, it is necessary to compute the gradient and the hessian of f_t as follows:

$$\nabla f_t(v) = 2tQv + tp - \sum_{i=1}^{2d} \frac{-A_i}{b_i - A_i^Tv}$$

$$\nabla f_t(v) = t(2Qv + p) + \sum_{i=1}^{2d} (b_i - A_i^Tv)^{-1} A_i.$$

$$\text{Hess} f_t(v) = 2tQ + \sum_{i=1}^{2d} (-1) \times \frac{-A_i}{(b_i - A_i^Tv)^2} A_i^T$$

$$\text{Hess} f_t(v) = 2tQ + \sum_{i=1}^{2d} (b_i - A_i^Tv)^{-2} A_i A_i^T.$$

In order to simplify the notations and afterwards the code, we will denote $D = \left((b_1 - A_1^Tv)^{-1}, \dots, (b_{2d} - A_{2d}^Tv)^{-1}\right)^T$

Hence, the gradient and the hessian of f_t can be rewritten as:

$$\nabla f_t(v) = t(2Qv + p) + A^T D$$

and $\text{Hess} f_t(v) = 2tQ + A^T \operatorname{diag}(D)^2 A.$

Question 2

In [ ]:

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

In [ ]:

```
# We start by defining the functions of ft, the gradient and the hessian of ft.

def ft(v, Q, p, A, b, t):
    if np.any(b - A @ v <= 0):
        return float("NaN")
    return t * (v.T @ Q @ v + p.T @ v) - np.sum(np.log(b - A @ v))


def gradient_ft(v, Q, p, A, b, t):
    D = 1. / (b - A @ v)
    return t * (2 * Q @ v + p) + A.T @ D


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

In [ ]:

```
# We define our line_search function

def line_search(f, grad_f, v, dv, alpha=.5, beta=.9):
    step = 1

    while f(v + step * dv) > f(v) + alpha * step * grad_f(v).T @ dv and step > 1e-6:
        step *= beta

        if np.any(b - A @ (v + step * dv) <= 0):
            return step

    return step
```

In [ ]:

```
# Now we can define our function centering_step

def centering_step(Q, p, A, b, t, v0, eps=1e-9, alpha=.5, beta=.9, max_iter=500):


    # to simplify notations
    obj = lambda v: ft(v, Q, p, A, b, t)
    grad = lambda v: gradient_ft(v, Q, p, A, b, t)
    hess = lambda v: hessian_ft(v, Q, p, A, b, t)

    # initialization
    v_seq = [v0]
    v = v0
    i = 0

    while i < max_iter:
        i += 1

        # Newton's method
        dv = np.linalg.pinv(hess(v)) @ grad(v)
        step = line_search(obj, grad, v, dv, alpha=alpha, beta=beta)
        v = v - step * dv
        v_seq.append(v)
```

```
        # stopping criterion
        l = grad(v).T @ dv
        if l < 2 * eps:
            break

    return v_seq
```

In [ ]:

```
# Now that we have our centering_step function we can define our function barr_method

def barr_method(Q, p, A, b, v0, mu, eps=1e-9, alpha=.5, beta=.9, max_iter=500):

    # initialization
    v_seq = [v0]
    t = 1
    m = len(A)

    while m/t > eps:
        x = centering_step(Q, p, A, b, t, v_seq[-1], eps=eps, alpha=alpha, beta=beta, ma
x_iter=max_iter)[-1]
        v_seq.append(x)

        t *= mu

    return v_seq
```

Question 3

In [ ]:

```
##### We start by defining the dimensions and parameters
n = 10
d = 50
reg = 10     # N.B : we set the regularization parameter lambda = 10 as the question aske
d.


# We randomly generate a matrix X and observations y
X = 5 * np.random.randn(n, d)
y = 5 + 1.5 * np.random.randn(n)

p = - y
Q = np.eye(n) * 0.5
A = np.concatenate((X.T, - X.T), axis=0)
b = reg * np.ones(2 * d)
v0 = np.zeros(n)

eps = 1e-9
alpha, beta = .5, .9
max_iter = 500

mu_values = [2, 15, 50, 100, 500]
```

In [ ]:

```
results = [barr_method(Q, p, A, b, v0, mu, eps, alpha, beta, max_iter) for mu in mu_valu
es]

f_values = [[ v.T @ Q @ v + p.T @ v 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('$f(v_t) - f^*$')

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