Homework 2 Convex Optimization

For Monday the 2nd ofDecember

Exercise 1

In the previous homework, one has shown that for $u \in \mathbb{R}^n$,

$$\| \cdot \|_1^*(u) = \begin{cases} 0 & \text{if } \|u\|_\infty \leq 1, \\ +\infty & \text{otherwise.} \end{cases}$$

The problem of LASSO is equivalent to that:

$$\min_w \frac{1}{2}\|z\|_2^2 + \lambda \|w\|_1 \quad \text{with constraint:} z = Xw - y.$$

So we basically transformed the problem into a Optimization problem.

So now we just compute the Lagrangian function of the problem and we have: Let $w \in \mathbb{R}^d, z \in \mathbb{R}^n, \nu \in \mathbb{R}^n$.

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

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

So the dual function is:

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

$$g(\nu) = y^T\nu + \inf_z \left(\underbrace{\frac{1}{2}\|z\|_2^2 + \nu^T z} \right) + \inf_w \left(\underbrace{\lambda\|w\|_1 - (X^T\nu)^T w} \right).$$

The function $h : z \mapsto \frac{1}{2}\|z\|_2^2 + \nu^T z$ is convex and differentiable. Its gradient is given by $\nabla h(z) = z + \nu$, and we have $\nabla h(z) = 0$ iff $z = -\nu$. Hence, the minimum of h is:

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

We can express the last term using the conjugate of $\| \cdot \|_1$:

$$\inf_w \lambda\|w\|_1 - (X^T\nu)^T w = \sup_w \left(\frac{1}{\lambda}X^T\nu\right)^T w - \|w\|_1 = \| \cdot \|_1^* \left(\frac{1}{\lambda}X^T\nu\right).$$

Thus,

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

where $\| \cdot \|_1^*$ is the indicator function of the $\| \cdot \|_\infty$-unit ball. Hence, the dual problem of LASSO is:

$$\max_\nu y^T\nu - \frac{1}{2}\|\nu\|_2^2 \quad \text{s.t.} \quad \left\|\frac{1}{\lambda}X^T\nu\right\|_\infty \leq 1.$$

Let's reformulate the constraint as an affine one:

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

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

$$\iff A\nu \preceq \lambda\mathbf{1}_{2d},$$

where $A = \begin{pmatrix} X^T \\ -X^T \end{pmatrix}$. Hence, multiplying the problem by -1, one has:

$$\min_\nu \nu^T Q\nu + p^T\nu,$$

where $Q = \frac{1}{2}I_n, p = -y, b = \lambda\mathbf{1}_{2d}$, and the constraints are $A\nu \leq b$.

The dual problem is given by:

$$\min_\nu \nu^T Q\nu + p^T\nu \quad \text{with } Q = \frac{1}{2}I_n,\ p = -y,\ b = \lambda \cdot \mathbf{1}_{2d},$$

knowing that $A\nu \preceq b$.

Exercise 2

let's just write down a few remarks:
Let's compute the gradient and the Hessian matrix of the objective function:

$$\nabla g_t(\nu) = t\big(2Q\nu + p\big) - \sum_{i=1}^{2d} \frac{-a_i}{b_i - a_i^T \nu},$$

$$\nabla g_t(\nu) = t\big(2Q\nu + p\big) + \sum_{i=1}^{2d} \big(b_i - a_i^T \nu\big)^{-1} a_i.$$

Defining:

$$\phi = \Big(\big(b_1 - a_1^T \nu\big)^{-1}, \ldots, \big(b_{2d} - a_{2d}^T \nu\big)^{-1}\Big)^T,$$

we have:

$$\nabla g_t(\nu) = t\,(2Q\nu + p) + A^T \phi.$$

Besides:

$$\nabla^2 g_t(\nu) = 2t \cdot Q + \sum_{i=1}^{2d} \big(b_i - a_i^T \nu\big)^{-2} a_i a_i^T,$$

and:

$$\nabla^2 g_t = 2t \cdot Q + A^T \,\mathrm{Diag}(\phi)^2 A.$$

For the code, see the Appendix page 4 to 5.

Exercise 3

See the jupyter notebook here `Appendix page 4 to 5.`.

Our remarks and conclusion on which μ to use:

1. mu = 2 is reaching precision but takes time
2. mu = 50 or 100 converges fast but it is at the expense of precision
3. mu = 15 seems to be a good tradeoff

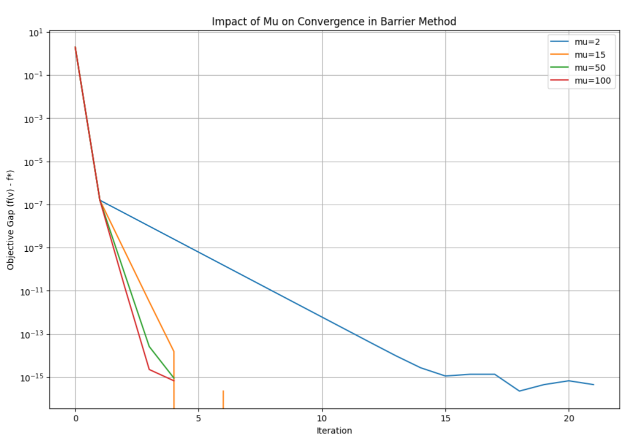


Figure 1: results

Appendix

```
import numpy as np
import pandas as pd
```

Question 2

Barrier Method

- centering_step
- barr_method

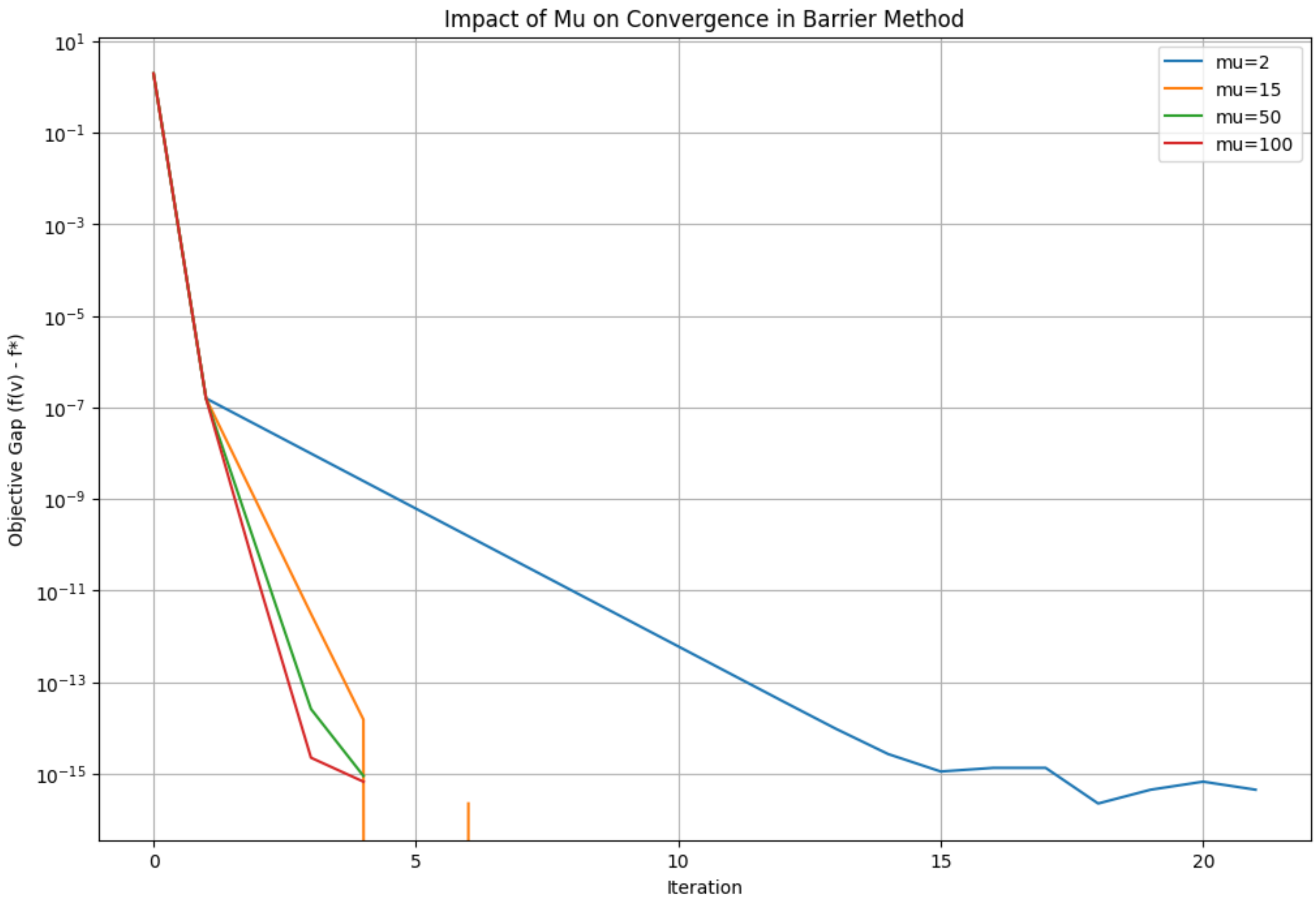
Centering step algorithm

- First compute the hessian and gradient
- then use it for the Newton method
- And we need to be sure that we still respect the condition ($<$ \varespsilon)

```
import numpy as np

def centering_step(Q, p, A, b, t, v0, eps):
    v = v0
    max_iter = 100
    for _ in range(max_iter):

        grad = 2 * Q @ v + p + np.sum((A.T / (b - A @ v)), axis=1) / t
        hess = 2 * Q + np.sum([(Ai[:, None] @ Ai[None, :]) / (bi - Ai @ v)**2 for A

        delta_v = np.linalg.solve(hess, -grad)

        alpha = 1
        while np.any(b - A @ (v + alpha * delta_v) <= 0):
            alpha *= 0.5
        v = v + alpha * delta_v

        if np.linalg.norm(grad) < eps:
            break
    return v



def barr_method(Q, p, A, b, v0, eps, mu):
    t = 1

    m = len(b)
    v = v0
    v_seq = [v0]

    while m / t > eps:

        v = centering_step(Q, p, A, b, t, v, eps)
        v_seq.append(v)

        t *= mu

    return v_seq



import matplotlib.pyplot as plt
```

```
def generate_random_problem(n, d, lam=10):
    """Generate random matrices X, observations y, and other problem parameters."
    np.random.seed(42)
    X = np.random.randn(n, d)
    y = np.random.randn(n)
    Q = X.T @ X + lam * np.eye(d)
    p = -X.T @ y
    A = np.vstack([np.eye(d), -np.eye(d)])
    b = lam * np.ones(2 * d)
    return Q, p, A, b


def objective_function(Q, p, v):
    """Evaluate the quadratic objective function."""
    return v.T @ Q @ v + p.T @ v



def test_barrier_method(Q, p, A, b, v0, eps, mu_values):
    """Test the barrier method for different values of mu and plot results."""
    f_star = None
    results = {}

    for mu in mu_values:
        v_seq = barr_method(Q, p, A, b, v0, eps, mu)
        f_values = [objective_function(Q, p, v) for v in v_seq]
        if f_star is None or min(f_values) < f_star:
            f_star = min(f_values)
        results[mu] = f_values


    plt.figure(figsize=(12, 8))
    for mu, f_values in results.items():
        gaps = [f - f_star for f in f_values]
        plt.semilogy(range(len(gaps)), gaps, label=f"mu={mu}")

    plt.xlabel("Iteration")
    plt.ylabel("Objective Gap (f(v) - f*)")
    plt.title("Impact of Mu on Convergence in Barrier Method")
    plt.legend()
    plt.grid()
    plt.show()

n, d = 50, 10
lam = 10
```

```
Q, p, A, b = generate_random_problem(n, d, lam)
v0 = np.zeros(d)
eps = 1e-5
mu_values = [2, 15, 50, 100]


test_barrier_method(Q, p, A, b, v0, eps, mu_values)
```



Start coding or generate with AI.

mu = 2 is reaching precision but takes time mu = 50 or 100 converges fast but it is athe expense of precision mu = 15 seems to be a good tradeoff

Start coding or generate with AI.

Start coding or generate with AI.