Convex Optimization - Homework 3

Boubacar Sow
Ecole Normale supérieure - Paris-Saclay

November 20, 2023

Exercise 1. Given $x_1, \dots, x_n \in \mathbb{R}^d$ data vectors and $y_1, \dots, y_n \in \mathbb{R}$ observations, we are searching for regression parameters $w \in \mathbb{R}^d$ which fit data inputs to observations y by minimizing their squared difference. In a high dimensional setting (when $n \ll d$) a 1-norm penalty is often used on the regression coefficients w in order to enforce sparsity of the solution (so that w will only have a few non-zeros entries). Such penalization has well known statistical properties, and makes the model both more interpretable, and faster at test time.

From an optimization point of view we want to solve the following problem called LASSO (which stands for Least Absolute Shrinkage Operator and Selection Operator)

$$\min_{w \in \mathbb{R}^d} \frac{1}{2}\|Xw - y\|_2^2 + \lambda \|w\|_1 \quad \text{(LASSO)}$$

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

1. Derive the dual problem of LASSO and format it as a general Quadratic Problem as follows

$$\min_{v \in \mathbb{R}^n} v^T Q v + p^T v \quad \text{subject to} \quad Av \preceq b \quad \text{(QP)}$$

   where $Q \succeq 0$.

2. Implement the barrier method to solve QP.

   - Write a function $v_{\text{seq}} = \text{centering step}(Q, p, A, b, t, v_0, \epsilon)$ which implements the Newton method to solve the centering step given the inputs (Q, p, A, b), the barrier method parameter t, initial variable v_0 and a target precision ϵ. The function outputs the sequence of variables iterates $(v_i)_{i=1,\dots,n_\epsilon}$, where n_ϵ is the number of iterations to obtain the ϵ precision. Use a backtracking line search with appropriate parameters.
   - Write a function $v_{\text{seq}} = \text{barr method}(Q, p, A, b, v_0, \epsilon)$ 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 $(v_i)_{i=1,\dots,n_\epsilon}$, where n_ϵ is the number of iterations to obtain the ϵ precision.

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 $\mu = 2, 15, 50, 100, \dots$ and check the impact on w. What would be an appropriate choice for μ?

Solution

1. **Reformulation of the Dual Problem of LASSO as a General Quadratic Problem**

The LASSO problem can be expressed as:

$$\min_w \frac{1}{2}\|z\|_2^2 + \lambda\|w\|_1 \quad \text{subject to} \quad z = Xw - y \tag{1}$$

where $w \in \mathbb{R}^d$, $z \in \mathbb{R}^n$, and $\mu \in \mathbb{R}^n$. The Lagrangian associated with this problem is:

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

Here, $w \in \mathbb{R}^d$, $z \in \mathbb{R}^n$, and $\mu \in \mathbb{R}^n$. The term $\frac{1}{2}\|z\|_2^2$ is the squared Euclidean norm of z, $\lambda\|w\|_1$ is the L1 norm of w scaled by the regularization parameter λ, and $\mu^T(z - Xw + y)$ is the inner product of the dual variable μ and the residual $z - Xw + y$.

The Lagrangian can be expanded and rearranged as follows:

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

Here, $\mu^T z$ is the inner product of μ and z, $w^T X^T \mu$ is the inner product of w and $X^T \mu$, and $y^T \mu$ is the inner product of y and μ.

The dual function is then:

$$g(\mu) = \inf_{w,z} L(w, z, \mu) = \inf_z \left(\frac{1}{2}\|z\|_2^2 + z^T \mu\right) + \inf_w \left(\lambda\|w\|_1 - w^T X^T \mu\right) + y^T \mu$$

Step 1: Compute the infimum over z

The function $h : z \mapsto \frac{1}{2}\|z\|_2^2 + z^T\mu$ is convex and differentiable. Its gradient is $\nabla h(z) = z + \mu$. Setting the gradient to zero gives the condition for the minimum:

$$\nabla h(z) = 0 \iff z = -\mu$$

Substituting $z = -\mu$ into $h(z)$ gives the minimum value of $h(z)$:

$$h(-\mu) = \frac{1}{2}\| - \mu\|_2^2 - \mu^T\mu = -\frac{1}{2}\|\mu\|_2^2$$

Step 2: Compute the infimum over w

The term involving w can be rewritten using the conjugate of the $\|\cdot\|_1$ norm:

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

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

Step 3: Combine the results

Substituting the results from steps 1 and 2 into the expression for $g(\mu)$ gives:

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

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

$$\begin{aligned} \underset{\mu}{\text{maximize}} \quad & y^T\mu - \frac{1}{2}\|\mu\|_2^2 \\ \text{subject to} \quad & \left\|\frac{1}{\lambda} X^T \mu\right\|_\infty \leq 1 \end{aligned} \tag{1}$$

We can reformulate the constraint as an affine one:

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

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

$$\iff A\mu \preceq b$$

where $A = \begin{bmatrix} X^T \\ -X^T \end{bmatrix}$ and $b = \lambda \cdot \mathbf{1}_{2d}$.

The dual problem is thus:

$$\begin{aligned} \underset{\mu}{\text{minimize}} \quad & \frac{1}{2}\mu^T\mu - y^T\mu \\ \text{subject to} \quad & A\mu \preceq b \quad \text{(Quadratic Problem)} \end{aligned} \tag{2}$$

Where : $A = \begin{bmatrix} X^T \\ -X^T \end{bmatrix}$, $b = \lambda \cdot \mathbf{1}_{2d}$, $\mu \in \mathbb{R}^n$.

2. **Implementation of the barrier method to solve QP.**

The QP problem we are trying to solve has a quadratic objective function and linear constraints. The objective function is given by:

$$f_0(v) = v^T Q v + p^T v \tag{3}$$

The barrier method involves two main steps: the centering step and the update step.

1. **Centering step**: In this step, we solve a sequence of approximate problems by using the Newton method. Each approximate problem adds a barrier term to the objective function of the original problem, which penalizes points near the boundary of the feasible region. The barrier parameter t controls the weight of the barrier term in the objective function. A backtracking line search is used to ensure sufficient decrease in the objective function value at each step of the Newton method.

The objective function for the centering step is given by:

$$f(v) = t f_0(v) - \sum_{i=1}^{2d} \log(b_i - [Av]_i) \tag{4}$$

The gradient and Hessian of this function are computed as follows:

- Gradient:

$$\nabla f(v) = t(Qv + p) - A^T(1/(b - Av)) \tag{5}$$

- Hessian:

$$\nabla^2 f(v) = 2tQ - A^T \mathrm{Diag}((1/(b - Av))^2) A \tag{6}$$

See the notebook for the code.

3. **Test of the barrier method on a random generated dataset**

Figure 1: Evolution of the dual gap, see more comments on the notebook

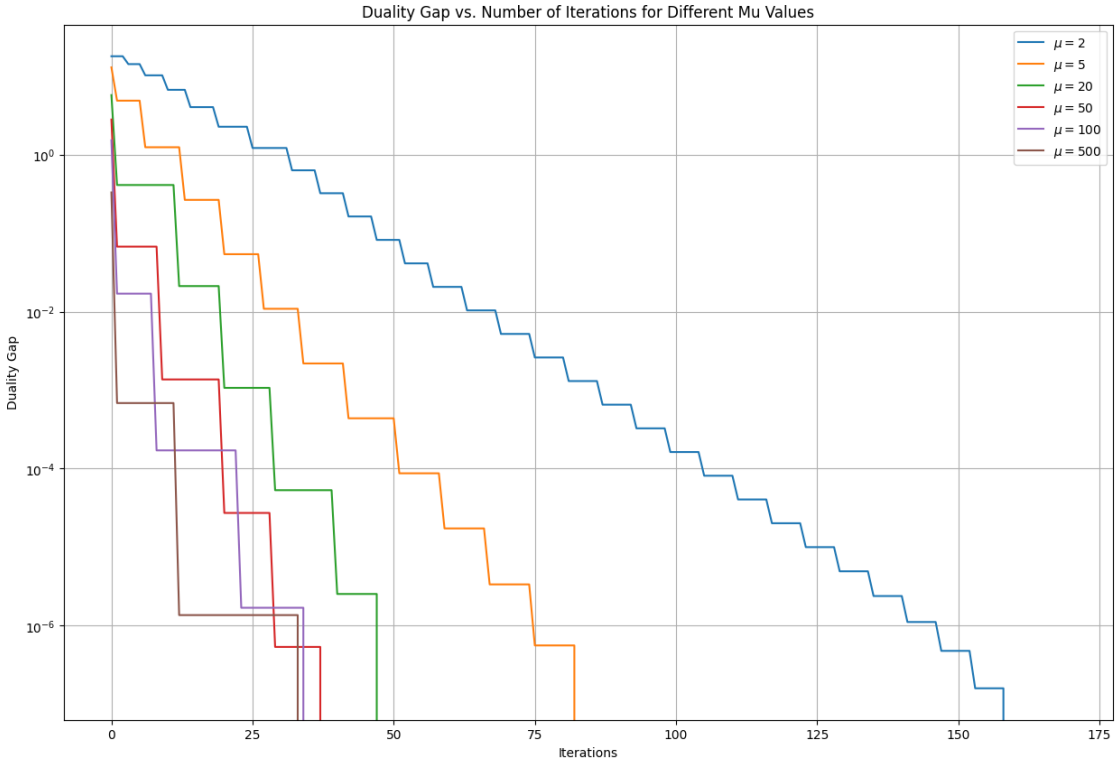


Figure 2: Impact of the different values of μ on the weights w, see more comments in the notebook

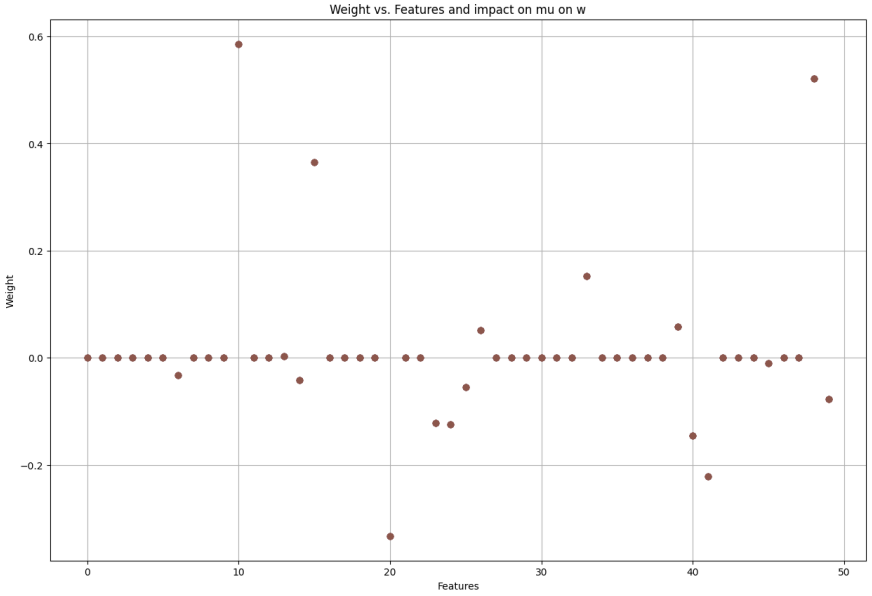
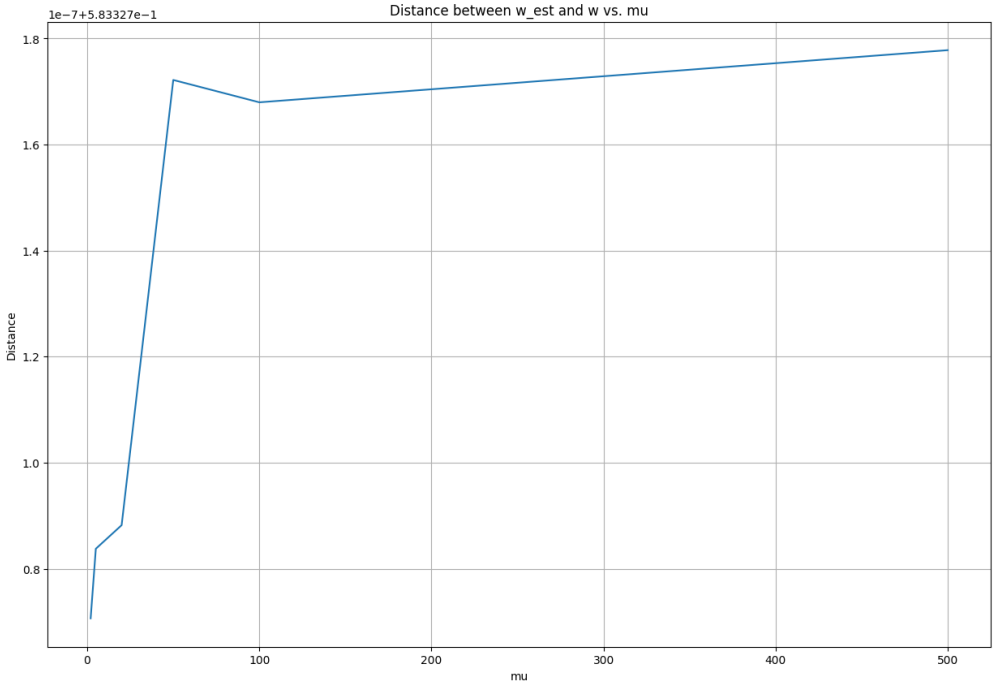


Figure 3: Difference between true w and estimated weights of a random sample, see more comments in the notebook



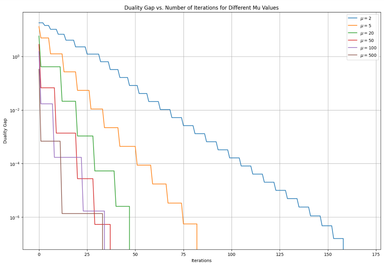
Convex Optimization - Homework 3

Boubacar Sow, boubacar.sow@ens-paris-saclay.fr

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [676]:
```
import numpy as np

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

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

def f0(Q, p, v):
    return v.T @ Q @ v + p.T @ v

def f(Q, p, A, b, t, v):
    diff = b - A @ v
    return None if np.any(diff <= 0) else t * f0(Q, p, v) - np.sum(np.log(diff))

def backtracking_line_search(Q, p, A, b, t, v, dv, alpha=0.1, beta=0.7):
    step_size = 1
    # Precompute some values
    f_v = f(Q, p, A, b, t, v)
    grad_dot_dv = np.dot(compute_gradient(Q, p, A, b, t, v).T, dv)
    while True:
        new_v = v + step_size * dv
        f_new_v = f(Q, p, A, b, t, new_v)
        if step_size > 1e-10 and (f_new_v is None or f_new_v > f_v + alpha * step_size * grad_dot_dv):
            step_size *= beta
        else:
            break
    return step_size


def centering_step(Q, p, A, b, t, v0, eps):
    v_seq = []
    k = 0
    while True:
        v_seq.append(v0.copy())
        grad = compute_gradient(Q, p, A, b, t, v0)
        H = compute_hessian(Q, A, b, t, v0)
        dv = - np.linalg.solve(H, grad)
        newton_decrement = - np.dot(grad.T, dv)
        if (newton_decrement / 2) <= eps:
            return v_seq, k
        step_size = backtracking_line_search(Q, p, A, b, t, v0, dv)
        v0 += step_size * dv

def barrier_method(Q, p, A , b ,v0 , eps=1e-6, mu=15):
    t = 0.1
    v_seq = []
    total_iters = 0
    # Feasibility check or we can assume that v0 is feasible
    while True:
        v_inner_seq, k = centering_step(Q, p, A, b, t, v0, eps)
        v_seq.append(v_inner_seq)
        total_iters += k
        v0 = v_inner_seq[-1]
        if b.shape[0] / t < eps:
            return v_seq, total_iters
        t *= mu
```

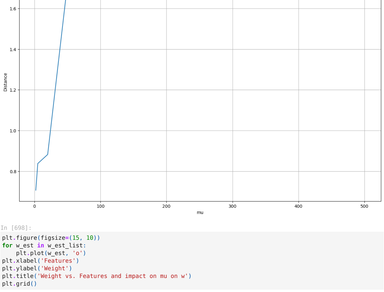
In [683]:
```
import numpy as np
import matplotlib.pyplot as plt

# Seed for reproducibility
np.random.seed(42)

# Generate synthetic data
num_samples = 100
num_features = 50
X, y = np.random.randn(num_samples, num_features), np.random.randn(num_samples)

# Define problem parameters
lambda_ = 10
Q = 0.5 * np.eye(num_samples)
p = -y
A = np.concatenate([X, -X], axis=1).T
b = lambda_ * np.ones(2 * num_features)
v0 = np.zeros(num_samples)
eps = 1e-6

# Set mu values
muses = [2, 5, 20, 50, 100, 500]

plt.figure(figsize=(15, 10))

gaps = []
objective = []
iterations = []

# Loop over mu values
for mu in muses:
    v_list, nb_iterations = barrier_method(Q, p, A, b, v0, eps, mu)

    v_sequence = [outer[-1] for outer in v_list for _ in outer]
    optimal_v = v_list[-1][-1]
    optimal_f = f0(Q, p, optimal_v)
    gap = [f0(Q, p, vi) - optimal_f for vi in v_sequence[:-1]]
    gaps.append(gap)
    iterations.append(nb_iterations)
    objective.append(optimal_f)


for g in gaps:
    plt.semilogy(g)

plt.yscale('log')
plt.xlabel('Newton iterations')
plt.ylabel('Duality Gap')
plt.legend([f'$\mu = {mu}$' for mu in muses])
plt.title('Duality Gap vs. Number of Iterations for Different Mu Values')
plt.grid()
plt.show()
```



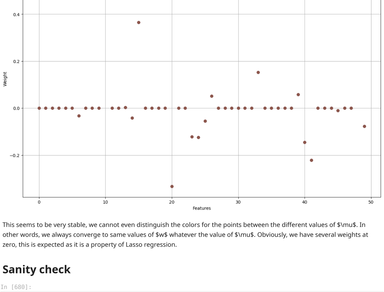
We can see here that we have more iterations if μ is small i.e. $\mu <= 5$ w When $\mu = 20$, we can still have many outer iterations. We can consider based on the observations that a μ between 50 and 100 as if we choose a high value of μ, we observe the opposite and we will have more inner iterations.

Impact of μ on w

In [687]:
```
dist = []
w_est_list = []
for m in muses:
    v_list, nb_iterations = barrier_method(Q, p, A, b, v0, eps, m)
    w_est = np.dot(np.linalg.pinv(X), y - v_list[-1][-1]) # KKT condition
    w_est /= np.linalg.norm(w_est)
    w = np.dot(np.linalg.pinv(X), y)
    w /= np.linalg.norm(w)
    dist.append(np.linalg.norm(w_est - w))
    w_est_list.append(w_est)

plt.figure(figsize=(15, 10))
plt.plot(muses, dist)
plt.xlabel('mu')
plt.ylabel('Distance')
plt.title('Distance between w_est and w vs. mu')
plt.grid()
plt.show()
```



In [688]:
```
plt.figure(figsize=(15, 10))
for w_est in w_est_list:
    plt.plot(w_est, 'o')
plt.xlabel('Features')
plt.ylabel('Weight')
plt.title('Weight vs. Features and impact on mu on w')
plt.grid()
```



This seems to be very stable, we cannot even distinguish the colors for the points between the different values of μ. In other words, we always converge to same values of w whatever the value of μ. Obviously, we have several weights at zero, this is expected as it is a property of Lasso regression.

Sanity check

In [680]:
```
import cvxpy as cp
import numpy as np
from sklearn.datasets import make_regression

# Seed for reproducibility
np.random.seed(42)

# Generate synthetic data
num_samples = 20
num_features = 100
X_data, y_data, coef_data = make_regression(n_samples=num_samples, n_features=num_features, coef=True)

# Define problem parameters
lambda_ = 10
Q = 0.5 * np.eye(num_samples)
p = y_data
A = np.vstack([X_data.T, -X_data.T])
b = lambda_ * np.ones(2 * num_features)
initial_v = np.zeros(num_samples)

# Run barrier method
solution_sequence,_ = barrier_method(Q, p, A, b, initial_v)

# Define and solve the CVXPY problem
variable = cp.Variable(num_samples)
objective = cp.Minimize((cp.quad_form(variable, Q) + p.T @ variable))
constraints = [A @ variable <= b]
problem = cp.Problem(objective, constraints)
problem.solve()

# Display results
print("\n" + "-" * 50)
barrier_solution = solution_sequence[-1][-1]
cvxpy_solution = variable.value
objective_barrier = f0(Q, p, barrier_solution)
objective_cvxpy = problem.value
print('Difference in objective function values: ', np.abs(objective_cvxpy - objective_barrier))
print()
print("Solution from barrier method:", barrier_solution)
print("Solution from CVXPY:", cvxpy_solution)
```

```
----------------------------------------------------
Difference in objective function values:  7.802327672834508e-08

Solution from barrier method: [ 0.9759365  -1.70267913  0.38030792  0.31964372  3.98787202 -1.04912036
  0.40283373  0.7250498   -0.36334495 -0.58708938 -2.27209172  1.91771785
  3.63524759  1.38911772 -2.61848496  0.42661692  3.10612045  0.31221236
 -0.70460549  0.24032645]
Solution from CVXPY: [ 0.9759365  -1.70267913  0.38030792  0.31964372  3.98787202 -1.04912036
  0.40283373  0.7250498   -0.36334495 -0.58708938 -2.27209172  1.91771784
  3.63524759  1.38911772 -2.61848496  0.42661692  3.10612045  0.31221236
 -0.70460549  0.24032645]
```

We can definitely conclude that the implemented algorithm is working perfectly

In [ ]: