MATH 173B, HW6

Victor Pekkari — epekkari@ucsd.edu

March 8, 2025

Problem 1

(a)

$$\mathcal{L}(x, \lambda, \nu) = \left[x_1^2 + 2x_2^2\right] + \lambda_1(4 - x_1 - 2x_2) - \lambda_2(x_1) - \lambda_3(x_2)$$

$$= x_1^2 + 2x_2^2 - (\lambda_1 + \lambda_2)x_1 - (2\lambda_1 + \lambda_3)x_2 + 4\lambda_1$$

(b)

Since the function is convex in x_1, x_2, setting the gradient equal to zero will give us the minimizer.

$$\nabla_{x_1}\mathcal{L} = 2x_1 - \lambda_1 - \lambda_2 = 0 \implies x_1^* = \frac{1}{2}(\lambda_1 + \lambda_2)$$

$$\nabla_{x_2}\mathcal{L} = 4x_2 - 2\lambda_1 - \lambda_3 = 0 \implies x_2^* = \frac{1}{4}(2\lambda_1 + \lambda_3)$$

Lagrangian dual function $F(\lambda, \nu)$ is:

$$F(\lambda, \nu) = \min_x \mathcal{L}(x, \lambda, \nu) = \mathcal{L}(x^*, \lambda, \nu)$$

Putting in x_1^*, x_2^* into $\mathcal{L}$ yields the Lagrangian dual function:

$$F(\lambda, \nu) = \frac{1}{4}(\lambda_1 + \lambda_2)^2 + \frac{1}{8}(2\lambda_1 + \lambda_3)^2 - \frac{1}{2}(\lambda_1 + \lambda_2)^2 - \frac{1}{4}(2\lambda_1 + \lambda_3)^2 + 4\lambda_1$$

$$= -\frac{1}{2}(\lambda_1 + \lambda_2)^2 - \frac{1}{4}(2\lambda_1 + \lambda_3)^2 + 4\lambda_1$$

(c)

If the Hessian of $F(\lambda, \nu)$ is negative semidefinite the function is concave:

$$\frac{\partial F}{\partial \lambda_1} = -(\lambda_1 + \lambda_2) - \frac{1}{2}(2\lambda_1 + \lambda_3) + 4$$
$$\frac{\partial F}{\partial \lambda_2} = -(\lambda_1 + \lambda_2)$$
$$\frac{\partial F}{\partial \lambda_3} = -\frac{1}{2}(2\lambda_1 + \lambda_3)$$

We compute the Hessian matrix:

$$H = \begin{bmatrix} -2 & -1 & -\frac{1}{2} \\ -1 & -1 & 0 \\ -1 & 0 & -\frac{1}{2} \end{bmatrix} \implies \lambda(H) = \{0, \frac{1}{4}(\sqrt{17} - 7), \frac{1}{4}(-7 - \sqrt{17})\}$$

Conclusion: We can see that all eigenvalues of the hessian are less than or equal to zero, This means that the Hessian is Negative Semidefinite. The NSD Hessian implies that the function $F(\lambda, \nu)$ is concave.

(d)

Since the Dual function is convex in λ, ν we can take it's gradient and set it equal to zero

$$\frac{\partial F}{\partial \lambda_1} = -(\lambda_1 + \lambda_2) - \frac{1}{2}(2\lambda_1 + \lambda_3) + 4 = 0$$
$$\frac{\partial F}{\partial \lambda_2} = -(\lambda_1 + \lambda_2) = 0$$
$$\frac{\partial F}{\partial \lambda_3} = -\frac{1}{2}(2\lambda_1 + \lambda_3) = 0$$

The equations above give us the following linear system to solve for λ_i:

$$\begin{cases} -2\lambda - \lambda_2 - \frac{1}{2}\lambda_3 + 4 & = 0 \\ -\lambda_1 + \lambda_2 & = 0 \\ -\lambda_1 - \frac{1}{2}\lambda_3 & = 0 \end{cases} \implies \begin{cases} \lambda_1^* = -8 \\ \lambda_2^* = 8 \\ \lambda_3^* = 24 \end{cases}$$

Putting λ_i^* back into the dual function yields the following:

$$F(\lambda^*, \nu) = -48$$

Answer: -48 is the minimizer for the dual optimization problem

(e)

Strong duality implies that the maximizer for the dual optimization problem is equal to the minimizer for the primal optimization problem. We can check wether strong duality holds between the primal and dual optimization problems by checking if Slater's condition holds.

Slater's condition is a sufficient condition for strong duality to hold for a convex optimization problem. We can see that Slater's condition holds for our optimization problem since:

(1) f(x) is convex:

$$f(x) = x_1^2 + 4x_2^2$$

(2) Conditions for the $g_i(x) \leq 0$ constraints holds:

$$g_1(x) = 4 - x_1 - 2x_2, \quad g_2(x) = -x_1, \quad g_3(x) = -x_2$$

- **(2.1)** All $g_i(x)$ are linear, which means they are convex
- **(2.2)** $\exists x \quad [g_i(x) < 0 \quad \forall g_i]$ for example $x = (1, 1)$ satisfies this condition.

(3) Doesn't exist $h_i(x) = 0$ constraints

This means all conditions that must hold for the $h_i(x)$ automatically holds

Answer: Since strong duality holds, the primal objective and the dual objective are equal, which means the primal minimizer will be the same as the dual maximizer

Problem 2

$$\min_{x\in\mathbb{R}^n} x^T x$$

$$\text{s.t.} \quad Ax = b, \quad A \in \mathbb{R}^{m\times n}$$

(a)

We have the following Lagrangian of the optimization problem:

$$\mathcal{L}(x, y) = f(x) + y^T (Ax - b)$$

where $f(x) = x^T x$, so the Lagrangian becomes:

$$\mathcal{L}(x, y) = x^T x + y^T (Ax - b)$$

The dual function is obtained by minimizing $\mathcal{L}(x, y)$ over x:

$$F(y) = \min_x \mathcal{L}(x, y) = \min_x \left(x^T x + y^T (Ax - b)\right)$$

$$\nabla_x \mathcal{L}(x, y) = 2x + A^T y = 0$$

Solving for x: $\implies$ $x^* = -\frac{1}{2} A^T y$

Dual Ascent Algorithm:

$$\begin{cases} x^{(t+1)} = -\frac{1}{2} A^T y^{(t)} & (= arg\min_x \mathcal{L}(x, y^{(t)})) \\ y^{(t+1)} = y^{(t)} + \alpha_t (Ax^{(t+1)} - b) \end{cases}$$

(b)

$$\mathcal{L}_\rho(x, y) = x^T x + y^T (Ax - b) + \frac{\rho}{2}\|Ax - b\|^2$$

$$\min_x \mathcal{L}(x, y) \implies \nabla_x \mathcal{L}_\rho = 2x + A^T y + \rho A^T (Ax - b) = 0$$

Solving for x:

$$x^{(t+1)} = \frac{\rho A^T b - A^T y}{2 + \rho\|A\|^2}$$

$$y^{(t+1)} = y^{(t)} + \rho(Ax^{(t+1)} - b)$$

Method of multipliers:

$$\begin{cases} x^{(t+1)} = \frac{\rho A^T b - A^T y}{2+\rho\|A\|^2} & (= arg\min_x \mathcal{L}_\rho(x, y^{(t)}) \\ y^{(t+1)} = y^{(t)} + \rho(Ax^{(t+1)} - b) \end{cases}$$

(c)

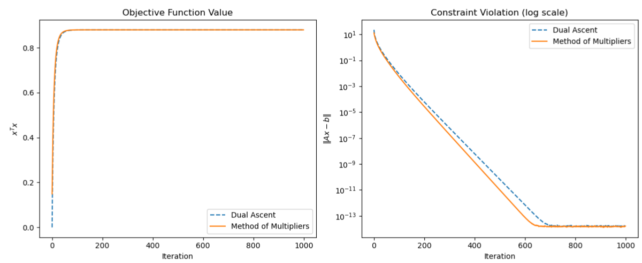

Figure 1: rho=270, alpha=0.001

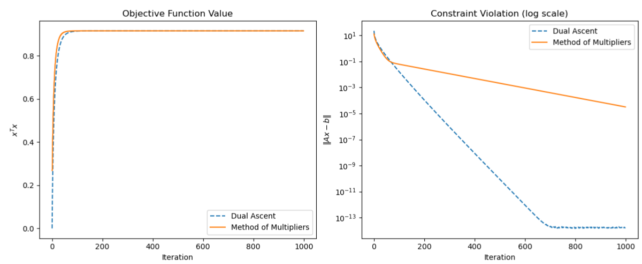

Figure 2: rho=350, alpha=0.001

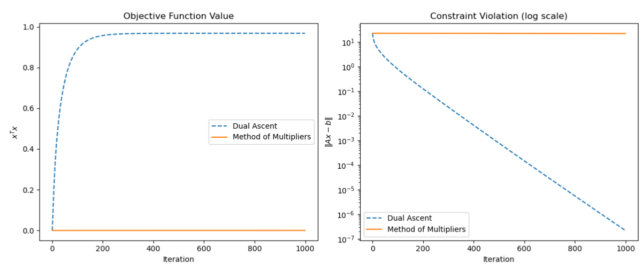

Figure 3: default start values

Answer: From my results I can deduce that MM performed well for a wider range of ρ-values. For example when I put the α-value too small the function value and the norm exploded for the dual ascent (didn't include this graph). MM didn't perform well for a small $\rho = 0.01$ I had too increase it a lot for MM to perform well. The optimum contained from the the two algorithms was approximately equal.
A note about computational efficiency is that Dual ascent is highly parallelisable. It would be possible to parallelise this function, i didn't though. This

parallelizability is indeed one advantage of dual ascent over MM, as MM requires sequential computations that cannot be easily distributed.

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

# Problem parameters
m, n = 500, 1000
A = np.random.randn(m, n)
b = np.ones(m)
alpha = 0.001
rho = 270
max_iters = 1000


def dual_ascent(A, b, alpha, max_iters):
    x_vals = []
    infeasibility = []

    y = np.zeros(m)
    x = np.zeros(n)

    for _ in range(max_iters):
        x = -0.5 * A.T @ y
        y = y + alpha * (A @ x - b)
        x_vals.append(x.T @ x)
        infeasibility.append(np.linalg.norm(A @ x - b))

    return x_vals, infeasibility

def method_of_multipliers(A, b, rho, max_iters):
    x_vals = []
    infeasibility = []

    y = np.zeros(m)
    x = np.zeros(n)

    for _ in range(max_iters):
        x = (rho * (A.T @ b) - A.T @ y) / (2 + (np.linalg.norm(A)**2))
        y = y + rho * (A @ x - b)

        x_vals.append(x.T @ x)
        infeasibility.append(np.linalg.norm(A @ x - b))

    return x_vals, infeasibility

x_vals_dual, infeasibility_dual = dual_ascent(A, b, alpha, max_iters)
x_vals_mm, infeasibility_mm = method_of_multipliers(A, b, rho, max_iters)

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(x_vals_dual, label='Dual Ascent', linestyle='dashed')
plt.plot(x_vals_mm, label='Method of Multipliers')
plt.xlabel("Iteration")
plt.ylabel(r"$x^T x$")
plt.title("Objective Function Value")
plt.legend()
```

```
plt.subplot(1, 2, 2)
plt.plot(infeasibility_dual, label='Dual Ascent', linestyle='dashed')
plt.plot(infeasibility_mm, label='Method of Multipliers')
plt.xlabel("Iteration")
plt.ylabel(r"$\|Ax - b\|$")
plt.yscale("log")
plt.title("Constraint Violation (log scale)")
plt.legend()

plt.tight_layout()
plt.show()
```