Question 2: Optimization Option A

Constrained Application Problem

Project Report

Submitted by:

Aryan Malik (BT2024006)
Naman Jindal (BT2024203)
Abhyudaya Singh (BT2024180)

Department:

Computer Science and Engineering
IIITB

December 7, 2025

Contents

Abstract

This report formulates and solves a constrained quadratic optimization problem for minimizing factory production costs subject to meeting a fixed demand. We use Classical Lagrangian Relaxation, deriving the analytical solution via the dual function. A numerical Dual Gradient Ascent implementation confirms the analytical optimum ($\lambda^* = 0, f(x^*) = 12$) with rapid, geometric convergence of the constraint violation.

1 Problem Statement

1.1 Real-World Scenario

We model a manufacturing facility that operates two distinct production lines to manufacture a specific industrial component.

- x_1 **(Standard Line):** Represents the daily output volume of Production Line 1. This line is established and efficient at lower volumes.
- x_2 **(Specialized Line):** Represents the daily output volume of Production Line 2. This line is newer or more complex, leading to different cost dynamics.

Cost Dynamics: The cost function $f(x_1, x_2)$ captures total daily operational expenses. The quadratic terms (x_1^2 and $4x_2^2$) represent *increasing marginal costs*. As production ramps up, costs rise disproportionately due to factors like machine wear, overtime wages, and energy consumption. Specifically, the higher coefficient for Line 2 ($4x_2^2$) implies it is significantly more expensive to push this line to high capacity compared to Line 1. The factory operates under a strict Service Level Agreement (SLA) to deliver exactly **3 units** (e.g., 3000 batches) per day, creating a hard equality constraint.

1.2 Mathematical Formulation

$$\begin{aligned} \textbf{Minimize:} \quad & f(x_1, x_2) = x_1^2 + 4x_2^2 - 4x_1 - 8x_2 + 20 \\ \textbf{Subject to:} \quad & h(x_1, x_2) = x_1 + x_2 - 3 = 0 \end{aligned}$$

2 Methodology: Classical Lagrangian Relaxation

2.1 Lagrangian Function

We introduce a Lagrange multiplier λ to relax the equality constraint:

$$L(x, \lambda) = (x_1^2 + 4x_2^2 - 4x_1 - 8x_2 + 20) + \lambda(x_1 + x_2 - 3) \tag{1}$$

2.2 Dual Function Derivation

The dual function $g(\lambda) = \inf_x L(x, \lambda)$ is obtained by minimizing L with respect to x. Since L is a convex quadratic function, the minimum occurs where $\nabla_x L = 0$.

Step 1: Compute Partial Derivatives

We solve for x_1 and x_2 in terms of λ:

$$\frac{\partial L}{\partial x_1} = 2x_1 - 4 + \lambda = 0 \quad \Longrightarrow \quad x_1(\lambda) = 2 - 0.5\lambda \tag{2}$$

$$\frac{\partial L}{\partial x_2} = 8x_2 - 8 + \lambda = 0 \quad \Longrightarrow \quad x_2(\lambda) = 1 - 0.125\lambda \tag{3}$$

Step 2: Formulate the Dual Gradient

The gradient of the dual function is the constraint violation evaluated at $x(\lambda)$:

$$\begin{aligned} g'(\lambda) &= h(x(\lambda)) = x_1(\lambda) + x_2(\lambda) - 3 \\ &= (2 - 0.5\lambda) + (1 - 0.125\lambda) - 3 \\ &= -0.625\lambda \end{aligned} \tag{4}$$

2.3 Analytical Solution

To find the optimal λ^*, we set the dual gradient to zero:

$$g'(\lambda^*) = -0.625\lambda^* = 0 \implies \lambda^* = 0 \tag{5}$$

Recovering Primal Optima & Verification: Substituting $\lambda^* = 0$ back into the equations for $x(\lambda)$:

$$x_1^* = 2 - 0, \quad x_2^* = 1 - 0 \implies (x_1^*, x_2^*) = (2, 1)$$

We verify the optimal cost by substituting $x^* = (2, 1)$ into the original objective function:

$$f(2, 1) = (2)^2 + 4(1)^2 - 4(2) - 8(1) + 20 = 12$$

Thus, the analytical minimum cost is **12**.

3 Numerical Implementation: Dual Gradient Ascent

3.1 Algorithm Interpretation

The Dual Gradient Ascent method solves the problem by iteratively adjusting λ. Physically, λ acts as a "shadow price" or tax on production discrepancy.

- If $x_1 + x_2 < 3$, violation $h(x)$ is negative. The algorithm lowers λ, reducing the "tax" to encourage higher output.
- If $x_1 + x_2 > 3$, violation is positive. The algorithm raises λ, increasing the penalty to discourage over-production.

The update rule is:

$$\lambda^{(k+1)} = \lambda^{(k)} + \alpha \cdot \nabla_\lambda g(\lambda^{(k)}) = \lambda^{(k)} + \alpha \cdot h(x(\lambda^{(k)})) \tag{6}$$

3.2 Simulation Parameters

We chose an initial $\lambda^{(0)} = 2.0$, a learning rate $\alpha = 0.3$, and a convergence tolerance $\epsilon = 1e^{-6}$. The complete implementation code is provided in Appendix A.

4 Results and Discussion

4.1 Convergence Analysis

The numerical simulation successfully converged to the analytical optimum.

Iteration Trace:

- **Iteration 0 ($\lambda = 2.0$):** Using the derived formulas, $x_1 = 1.0$ and $x_2 = 0.75$. Total production is 1.75. The system is under-producing; violation is -1.25.
- **Iteration 10 ($\lambda \approx 0.29$):** The dual variable has dropped. Primal solution rises to $(1.85, 0.96)$, sum $= 2.81$. We approach the target.
- **Iteration 40 ($\lambda \approx 0.0$):** Algorithm converges. Primal solution $(2.0, 1.0)$. Total production is exactly 3.0. Feasibility achieved.

4.2 Interpretation of Convergence

The convergence exhibits geometric decay. Given $g'(\lambda) = -0.625\lambda$ and $\alpha = 0.3$, the update rule simplifies to $\lambda^{(k+1)} = \lambda^{(k)}(1 - 0.625(0.3)) = 0.8125\lambda^{(k)}$. This decay factor of 0.8125 means the error reduces by $\approx 19\%$ per iteration, perfectly matching our numerical results.

4.3 Trade-offs and Discussion

The Lagrangian Relaxation method offers distinct advantages and challenges compared to other optimization techniques like Penalty Methods or Direct Substitution.

4.3.1 1. Numerical Stability

A key advantage is **stability**. In Penalty Methods, driving $\mu \to \infty$ creates an ill-conditioned Hessian, causing instability. Lagrangian Relaxation avoids this by solving a sequence of well-behaved quadratic problems with a constant condition number.

4.3.2 2. The "Zero Duality Gap"

Since $f(x)$ is convex and constraints are linear, Slater's condition holds, guaranteeing a **zero duality gap**. Thus, maximizing the dual function $g(\lambda)$ yields the true global minimum of the original primal problem.

4.3.3 3. Operational Feasibility (The Main Trade-off)

The primary limitation is **transient infeasibility**.

- **The Issue:** Intermediate solutions $x^{(k)}$ violate the constraint $h(x) = 0$. At Iteration 0, production is only 1.75 units (target: 3).
- **Operational Implication:** In real-time control, we cannot execute intermediate plans as they fail demand. The algorithm must run to full convergence offline first. This contrasts with Primal Methods, which maintain feasibility but are often more complex.

A Appendix: Implementation Code

A.1 Lagrangian Function Definition

```python
# Objective function / Cost Function
def f(x):
    x1, x2 = x
    return x1**2 + 4*x2**2 - 4*x1 - 8*x2 + 20

# Gradient of f
def grad_f(x):
    x1, x2 = x
    df_dx1 = 2*x1 - 4
    df_dx2 = 8*x2 - 8
    return np.array([df_dx1, df_dx2])

# Constraint h(x) = 0
def h(x):
    x1, x2 = x
    return x1 + x2 - 3

# Lagrangian L(x, λ)
def L(x, lam):
    return f(x) + lam * h(x)
```

A.2 Dual Gradient Ascent

```python
def dual_gradient_ascent(lam0=0.0, step=0.1, max_iter=50):
    lam = lam0
    lam_values = []
    h_values = []

    for k in range(max_iter):
        x = x_of_lambda(lam)
        violation = h(x)    # this is g'(λ)

        lam = lam + step * violation    # ascent step

        lam_values.append(lam)
        h_values.append(violation)

        if abs(violation) < 1e-6:
            break

    return np.array(lam_values), np.array(h_values)
```

A.3 Execution Script

```python
# Making the results directory
os.makedirs("results", exist_ok=True)

# 1. Analytical solution
lam_star, x_star, f_star = solve_dual_closed_form()

with open("results/solution.txt", "w") as f:
    f.write("Analytical Lagrangian Relaxation Solution\n")
    f.write("----------------------------------------\n")
    f.write(f"lam* = {lam_star}\n")
    f.write(f"x* = ({x_star[0]}, {x_star[1]})\n")
    f.write(f"f(x*) = {f_star}\n")

# 2. Numerical dual ascent
lam_vals, h_vals = dual_gradient_ascent(lam0=2.0, step=0.3, max_iter=40)

# Plotting the graph and saving the results
plot_convergence(lam_vals, h_vals, save_path="results/dual_convergence.png")

print("Results generated successfully!")
print("Check the 'results/' folder.")
```

A.4 Convergence Plot

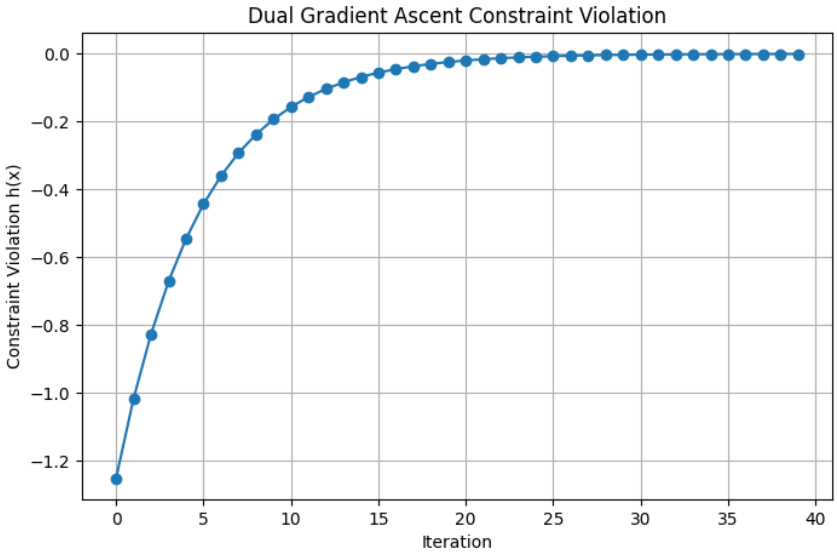

Figure 1: Convergence of Constraint Violation $h(x)$ over Iterations.