5.13 Lagrangian relaxation of Boolean LP

A *Boolean linear program* is an optimization problem of the form

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Ax \preceq b \\ & x_i \in \{0,1\}, \quad i=1,\ldots,n, \end{array}$$

which is, in general, very difficult to solve. The *LP relaxation* of this problem is

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Ax \preceq b \\ & 0 \le x_i \le 1, \quad i=1,\ldots,n, \end{array} \tag{5.107}$$

which gives a lower bound on the optimal value of the Boolean LP.

1. **Lagrangian relaxation.** The Boolean LP can be reformulated as the problem

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Ax \preceq b \\ & x_i(1-x_i) = 0, \quad i=1,\ldots,n, \end{array}$$

   which has quadratic equality constraints. Find the Lagrange dual of this problem.

2. Show that the lower bound obtained via Lagrangian relaxation, and via the LP relaxation (5.107), are the same.

Solution.

(a) Deriving the Lagrange dual

Step 1: Form the Lagrangian. With $\lambda \succeq 0$ for the inequality constraints and $\nu \in \mathbf{R}^n$ for the equality constraints:

$$\begin{aligned} L(x,\lambda,\nu) &= c^T x + \lambda^T(Ax-b) + \sum_{i=1}^n \nu_i x_i(1-x_i) \\ &= c^T x + \lambda^T A x - \lambda^T b + \sum_{i=1}^n \nu_i x_i - \sum_{i=1}^n \nu_i x_i^2 \\ &= (c + A^T\lambda + \nu)^T x - \sum_{i=1}^n \nu_i x_i^2 - \lambda^T b \end{aligned}$$

Step 2: Compute the dual function. The dual function is

$$g(\lambda,\nu) = \inf_{x\in\mathbf{R}^n} L(x,\lambda,\nu).$$

Since the Lagrangian separates across components of x:

$$g(\lambda,\nu) = -\lambda^T b + \sum_{i=1}^n \inf_{x_i\in\mathbf{R}} \underbrace{\left[(c_i + (A^T\lambda)_i + \nu_i)x_i - \nu_i x_i^2\right]}_{\ell_i(x_i)}$$

Step 3: Compute $\inf_{x_i} \ell_i(x_i)$ for each component. Define $\ell_i(x_i) = (c_i + (A^T\lambda)_i + \nu_i)x_i - \nu_i x_i^2$. This is a quadratic in x_i.

Case 1: $\nu_i > 0$. The coefficient of x_i^2 is $-\nu_i < 0$, so ℓ_i is concave (opens downward):

$$\inf_{x_i \in \mathbf{R}} \ell_i(x_i) = -\infty.$$

Case 2: $\nu_i = 0$. Then $\ell_i(x_i) = (c_i + (A^T\lambda)_i)x_i$ is linear:

$$\inf_{x_i \in \mathbf{R}} \ell_i(x_i) = -\infty \quad (\text{unless } c_i + (A^T\lambda)_i = 0).$$

Case 3: $\nu_i < 0$. The coefficient of x_i^2 is $-\nu_i > 0$, so ℓ_i is convex (opens upward). Let $\mu_i = -\nu_i > 0$. Then:

$$\ell_i(x_i) = (c_i + (A^T\lambda)_i - \mu_i)x_i + \mu_i x_i^2$$

To find the minimum, take the derivative and set to zero:

$$\frac{d\ell_i}{dx_i} = (c_i + (A^T\lambda)_i - \mu_i) + 2\mu_i x_i = 0$$

$$\Rightarrow \quad x_i^\star = -\frac{c_i + (A^T\lambda)_i - \mu_i}{2\mu_i}$$

Substituting back to find the minimum value:

$$\begin{aligned}
\inf_{x_i} \ell_i(x_i) &= \ell_i(x_i^\star) \\
&= (c_i + (A^T\lambda)_i - \mu_i) \cdot \left(-\frac{c_i + (A^T\lambda)_i - \mu_i}{2\mu_i}\right) + \mu_i \left(-\frac{c_i + (A^T\lambda)_i - \mu_i}{2\mu_i}\right)^2 \\
&= -\frac{(c_i + (A^T\lambda)_i - \mu_i)^2}{2\mu_i} + \frac{(c_i + (A^T\lambda)_i - \mu_i)^2}{4\mu_i} \\
&= -\frac{(c_i + (A^T\lambda)_i - \mu_i)^2}{4\mu_i}
\end{aligned}$$

Step 4: Assemble the dual function. For the dual function to be finite, we need $\nu_i < 0$ for all i. Using $\mu = -\nu \succ 0$:

$$g(\lambda, \mu) = -\lambda^T b - \sum_{i=1}^n \frac{(c_i + (A^T\lambda)_i - \mu_i)^2}{4\mu_i}$$

Step 5: Write the Lagrange dual problem.

$$\boxed{\begin{array}{ll} \text{maximize} & -\lambda^T b - \displaystyle\sum_{i=1}^n \frac{(c_i + (A^T\lambda)_i - \mu_i)^2}{4\mu_i} \\ \text{subject to} & \lambda \succeq 0, \quad \mu \succ 0 \end{array}}$$

(b) Equivalence with LP relaxation

The dual of the LP relaxation (5.107) is:

$$\begin{array}{ll} \text{maximize} & -b^T\lambda - \mathbf{1}^T\mu \\ \text{subject to} & c + A^T\lambda + \mu \succeq 0, \quad \lambda \succeq 0, \quad \mu \succeq 0 \end{array}$$

To show equivalence, we optimize the Lagrangian dual over μ_i for fixed λ. Let $a_i = c_i + (A^T\lambda)_i$. We compute:

$$\sup_{\mu_i > 0} \left(-\frac{(a_i - \mu_i)^2}{4\mu_i} \right)$$

Taking the derivative with respect to μ_i and setting to zero:

$$\frac{d}{d\mu_i} \left(-\frac{(a_i - \mu_i)^2}{4\mu_i} \right) = -\frac{2(a_i - \mu_i)(-1) \cdot 4\mu_i - (a_i - \mu_i)^2 \cdot 4}{16\mu_i^2} = \frac{a_i^2 - \mu_i^2}{4\mu_i^2} = 0$$

This gives $\mu_i^\star = |a_i|$ (taking positive root since $\mu_i > 0$).

Evaluating:

$$\sup_{\mu_i > 0} \left(-\frac{(a_i - \mu_i)^2}{4\mu_i} \right) = \begin{cases} 0 & \text{if } a_i \geq 0 \\ a_i & \text{if } a_i < 0 \end{cases} = \min(0, a_i)$$

Summing over all i:

$$\sup_{\mu \succ 0} g(\lambda, \mu) = -\lambda^T b + \sum_{i=1}^n \min(0, c_i + (A^T\lambda)_i)$$

This matches the LP relaxation dual objective $-b^T\lambda - \mathbf{1}^T\mu$ when we set $\mu_i = \max(0, -(c_i + (A^T\lambda)_i))$ to satisfy $c + A^T\lambda + \mu \succeq 0$.

Therefore:

$$\boxed{d^\star_{\text{Lagrangian}} = d^\star_{\text{LP relaxation}}}$$

The LP relaxation provides the same lower bound on the Boolean LP as the Lagrangian relaxation approach.

6.9 Minimax rational function fitting

Show that the following problem is quasiconvex:

$$\text{minimize} \quad \max_{i=1,\ldots,k} \left| \frac{p(t_i)}{q(t_i)} - y_i \right|$$

where

$$p(t) = a_0 + a_1 t + a_2 t^2 + \cdots + a_m t^m, \qquad q(t) = 1 + b_1 t + \cdots + b_n t^n,$$

and the domain of the objective function is defined as

$$D = \{(a, b) \in \mathbf{R}^{m+1} \times \mathbf{R}^n \mid q(t) > 0,\ \alpha \le t \le \beta\}.$$

In this problem we fit a rational function $p(t)/q(t)$ to given data, while constraining the denominator polynomial to be positive on the interval $[\alpha, \beta]$. The optimization variables are the numerator and denominator coefficients a_i, b_i. The interpolation points $t_i \in [\alpha, \beta]$ and desired function values y_i, $i = 1, \ldots, k$, are given.

Solution.

1. Domain is convex: For fixed t, $q(t) > 0$ is a strict linear inequality in b. The domain D is the intersection of open halfspaces over $t \in [\alpha, \beta]$, hence convex.

2. Sublevel sets are convex: For $\gamma > 0$, the constraint $|p(t_i)/q(t_i) - y_i| \le \gamma$ with $q(t_i) > 0$ becomes:

$$-\gamma\, q(t_i) \le p(t_i) - y_i\, q(t_i) \le \gamma\, q(t_i)$$

Since $p(t_i)$ is linear in a and $q(t_i)$ is affine in b, these are **linear inequalities** in (a, b). Thus $S_\gamma = D \cap \{2k \text{ halfspaces}\}$ is convex.

Conclusion: Convex domain + convex sublevel sets $\Rightarrow$ **quasiconvex**.

Remark: Solve via bisection on γ, checking LP feasibility at each step.

A5.37 Properties under Slater's condition

Consider a convex optimization problem

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & f_i(x) \leq 0, \quad i = 1, \dots, m \\ & Ax = b, \end{array}$$

with variable $x \in \mathbf{R}^n$, that satisfies Slater's constraint qualification. Determine whether each of the statements below is true or false. True means it holds with no further assumptions.

(a) The primal and dual problems have the same objective value.

True.

Slater's constraint qualification states that there exists a point $\tilde{x}$ in the relative interior of the domain such that:

- $f_i(\tilde{x}) < 0$ for all $i = 1, \dots, m$ (strict inequality for inequality constraints)
- $A\tilde{x} = b$

For convex optimization problems, **Slater's condition implies strong duality**. Therefore:

$$p^\star = d^\star$$

The primal and dual optimal values are equal.

(b) The primal problem has a unique solution.

False.

Slater's condition guarantees strong duality, but it does *not* guarantee uniqueness of the primal solution.

Counterexample: Consider the LP

$$\begin{array}{ll} \text{minimize} & x_1 \\ \text{subject to} & x_1 + x_2 \leq 2 \\ & x_1 \geq 0, \ x_2 \geq 0 \end{array}$$

This is convex and satisfies Slater's condition (e.g., $\tilde{x} = (0.5, 0.5)$ is strictly feasible). However, the optimal value is $p^\star = 0$, achieved by *any* point of the form $(0, t)$ where $0 \leq t \leq 2$. The solution is not unique.

(c) The dual problem is not unbounded.

True.

For a convex optimization problem satisfying Slater's condition:

- The primal problem is feasible (Slater's condition provides a feasible point).
- By weak duality, $d^\star \leq p^\star$.

The dual problem is a *maximization* problem:

$$\text{maximize} \quad g(\lambda, \nu) \quad \text{subject to} \quad \lambda \succeq 0$$

Since $g(\lambda, \nu) \leq p^\star$ for all dual feasible (λ, ν), the dual objective is **bounded above** by $p^\star$. Therefore, the dual problem cannot be unbounded (i.e., $d^\star \neq +\infty$).

(d) Suppose $x^\star$ is optimal, with $f_1(x^\star) = -0.2$. Then for every dual optimal point $(\lambda^\star, \nu^\star)$, we have $\lambda_1^\star = 0$.

True.

This follows from **complementary slackness**. Under Slater's condition, strong duality holds, and the KKT conditions are necessary and sufficient for optimality. One of the KKT conditions is:

$$\lambda_i^\star f_i(x^\star) = 0, \quad i = 1, \dots, m$$

For $i = 1$: Since $f_1(x^\star) = -0.2 \neq 0$, we must have:

$$\lambda_1^\star \cdot (-0.2) = 0 \quad \Longrightarrow \quad \lambda_1^\star = 0$$

Intuitively, $f_1(x^\star) = -0.2 < 0$ means the first inequality constraint is **inactive** (strictly satisfied) at the optimal point. By complementary slackness, the corresponding dual variable must be zero.

This holds for *every* dual optimal point $(\lambda^\star, \nu^\star)$.

A6.2 Minimax rational fit to the exponential

We consider the specific problem instance with data

$$t_i = -3 + \frac{6(i-1)}{k-1}, \quad y_i = e^{t_i}, \quad i = 1, \dots, k,$$

where $k = 201$. (In other words, the data are obtained by uniformly sampling the exponential function over the interval $[-3, 3]$.)

Find a function of the form

$$f(t) = \frac{a_0 + a_1 t + a_2 t^2}{1 + b_1 t + b_2 t^2}$$

that minimizes $\max_{i=1,\dots,k} |f(t_i) - y_i|$. (We require that $1 + b_1 t_i + b_2 t_i^2 > 0$ for $i = 1, \dots, k$.)

Solution.

As shown in problem 6.9, this is a **quasiconvex optimization problem**. We solve it using **bisection** on the objective value γ, combined with a feasibility LP at each step.

For a given $\gamma > 0$, we check feasibility of:

$$\left| \frac{p(t_i)}{q(t_i)} - y_i \right| \leq \gamma, \quad i = 1, \dots, k$$

where $p(t) = a_0 + a_1 t + a_2 t^2$ and $q(t) = 1 + b_1 t + b_2 t^2$.

Since $q(t_i) > 0$, multiplying by $q(t_i)$ gives the linear constraints:

$$-\gamma\, q(t_i) \leq p(t_i) - y_i\, q(t_i) \leq \gamma\, q(t_i), \quad i = 1, \dots, k$$

Expanding:

$$-\gamma(1 + b_1 t_i + b_2 t_i^2) \leq (a_0 + a_1 t_i + a_2 t_i^2) - y_i(1 + b_1 t_i + b_2 t_i^2) \leq \gamma(1 + b_1 t_i + b_2 t_i^2)$$

Rearranging, for each i:

$$a_0 + a_1 t_i + a_2 t_i^2 - (y_i + \gamma) b_1 t_i - (y_i + \gamma) b_2 t_i^2 \leq y_i + \gamma$$

$$a_0 + a_1 t_i + a_2 t_i^2 - (y_i - \gamma) b_1 t_i - (y_i - \gamma) b_2 t_i^2 \geq y_i - \gamma$$

These are linear in the variables $(a_0, a_1, a_2, b_1, b_2)$.

We also need $q(t_i) > 0$, which we enforce as $q(t_i) \geq \epsilon$ for a small $\epsilon > 0$.

Results.

Using bisection with tolerance 0.001, we obtain:

$$\begin{aligned} a_0 &= 1.0099 \\ a_1 &= 0.6117 \\ a_2 &= 0.1134 \\ b_1 &= -0.4147 \\ b_2 &= 0.0485 \end{aligned}$$

Optimal objective value: $\gamma^\star \approx 0.0233$

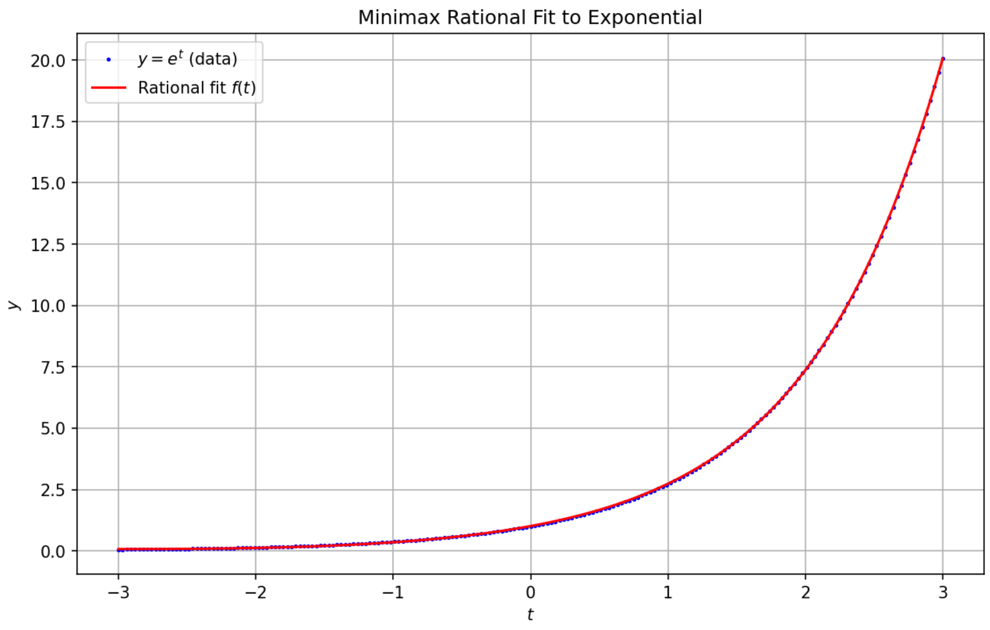

Figure 1: Exponential function $y = e^t$ (blue dots) and rational fit $f(t)$ (red line) over $[-3, 3]$.

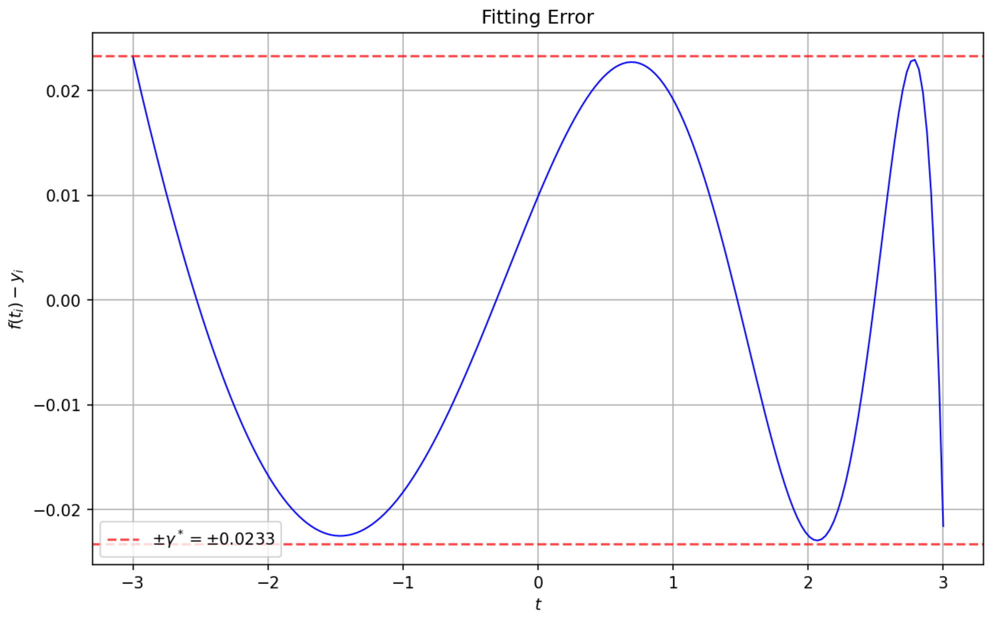

Figure 2: Fitting error $f(t_i) - y_i$ showing the equioscillation property characteristic of minimax approximation.

The fitting error plot shows the characteristic *equioscillation* behavior of minimax (Chebyshev) approximation: the error oscillates between approximately $\pm\gamma^\star$. This is consistent with the optimality conditions for minimax polynomial/rational approximation.

Python code:

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

# Data generation
k = 201
t = np.linspace(-3, 3, k)
y = np.exp(t)

# Build the Vandermonde-like matrix for powers of t
# Tpowers = [1, t, t^2] for each data point
Tpowers = np.column_stack([np.ones(k), t, t**2])

def check_feasibility(gamma, Tpowers, y):
    """
    Check if there exists a rational function f(t) = p(t)/q(t) such that
    |f(t_i) - y_i| <= gamma for all i, with q(t_i) > 0.

    p(t) = a0 + a1*t + a2*t^2  (coefficients a = [a0, a1, a2])
    q(t) = 1 + b1*t + b2*t^2   (coefficients [1, b1, b2], so b = [b1, b2])

    The constraint |p(t_i)/q(t_i) - y_i| <= gamma becomes:
    |p(t_i) - y_i*q(t_i)| <= gamma*q(t_i)

    With Tpowers @ a = p(t) and Tpowers @ [1; b] = q(t):
    |Tpowers @ a - y * (Tpowers @ [1; b])| <= gamma * (Tpowers @ [1; b])
    """
    k = len(y)

    # Variables: a = [a0, a1, a2], b = [b1, b2]
    a = cp.Variable(3)
    b = cp.Variable(2)

    # q_coeffs = [1, b1, b2]
    q_coeffs = cp.hstack([1, b])

    # p(t_i) = Tpowers @ a
    # q(t_i) = Tpowers @ q_coeffs
    p_vals = Tpowers @ a
    q_vals = Tpowers @ q_coeffs

    # Constraints: |p - y*q| <= gamma*q, which splits into two linear constraints
    # p - y*q <= gamma*q  =>  p <= (y + gamma)*q
    # p - y*q >= -gamma*q =>  p >= (y - gamma)*q
    constraints = [
        p_vals <= cp.multiply(y + gamma, q_vals),
        p_vals >= cp.multiply(y - gamma, q_vals),
    ]

    prob = cp.Problem(cp.Minimize(0), constraints)
    prob.solve()

    if prob.status == 'optimal':
        return True, a.value, b.value
    else:
        return False, None, None

# Bisection to find optimal gamma
l = 0.0  # lower bound
u = np.exp(3)  # upper bound (conservative)
```

```python
bisection_tol = 1e-3

a_opt = None
b_opt = None
gamma_opt = None

while u - l >= bisection_tol:
    gamma = (l + u) / 2
    feasible, a, b = check_feasibility(gamma, Tpowers, y)
    if feasible:
        u = gamma
        a_opt = a
        b_opt = b
        gamma_opt = gamma
    else:
        l = gamma

print(f"Optimal␣gamma:␣{gamma_opt:.4f}")
print(f"a0␣=␣{a_opt[0]:.4f}")
print(f"a1␣=␣{a_opt[1]:.4f}")
print(f"a2␣=␣{a_opt[2]:.4f}")
print(f"b1␣=␣{b_opt[0]:.4f}")
print(f"b2␣=␣{b_opt[1]:.4f}")

# Compute the rational function fit
def f(t_val, a, b):
    p = a[0] + a[1]*t_val + a[2]*t_val**2
    q = 1 + b[0]*t_val + b[1]*t_val**2
    return p / q

y_fit = f(t, a_opt, b_opt)

# Plot 1: Data and fit
plt.figure(figsize=(10, 6))
plt.plot(t, y, 'b.', markersize=3, label=r'$y␣=␣e^t$␣(data)')
plt.plot(t, y_fit, 'r-', linewidth=1.5, label=r'Rational␣fit␣$f(t)$')
plt.xlabel('$t$')
plt.ylabel('$y$')
plt.title('Minimax␣Rational␣Fit␣to␣Exponential')
plt.legend()
plt.grid(True)
plt.savefig('latex/img/a6_2_fit.png', dpi=150, bbox_inches='tight')
plt.close()

# Plot 2: Fitting error
error = y_fit - y
plt.figure(figsize=(10, 6))
plt.plot(t, error, 'b-', linewidth=1)
plt.axhline(y=gamma_opt, color='r', linestyle='--', alpha=0.7, label=f'$\\pm\\
    gamma^*␣=␣\\pm{gamma_opt:.4f}$')
plt.axhline(y=-gamma_opt, color='r', linestyle='--', alpha=0.7)
plt.xlabel('$t$')
plt.ylabel('$f(t_i)␣-␣y_i$')
plt.title('Fitting␣Error')
plt.legend()
plt.grid(True)
plt.savefig('latex/img/a6_2_error.png', dpi=150, bbox_inches='tight')
plt.close()

```

```python
print(f"\nMax absolute error: {np.max(np.abs(error)):.4f}")
print("Plots saved to latex/img/")
```

A6.27 Properties of least-p-norm solutions

Let $x^\star$ be optimal for the least-p-norm problem

$$\begin{array}{ll} \text{minimize} & \|x\|_p \\ \text{subject to} & Ax = b, \end{array}$$

with variable $x \in \mathbf{R}^n$, where $A \in \mathbf{R}^{m\times n}$, with $m \ll n$. (And of course, $p \in [1,\infty]$.) Determine if the statements below are reasonable or unreasonable.

(a) For $p = 2$, we would expect to see many components of $x^\star$ equal to zero.
Unreasonable.
For $p = 2$, the problem becomes:

$$\begin{array}{ll} \text{minimize} & \|x\|_2 \\ \text{subject to} & Ax = b \end{array}$$

The optimal solution is the *minimum-norm least squares solution*:

$$x^\star = A^T(AA^T)^{-1}b = A^\dagger b$$

where $A^\dagger$ is the Moore-Penrose pseudoinverse.

This solution is a **linear function** of b. Since $x^\star = A^T y$ for some $y \in \mathbf{R}^m$, the solution is a linear combination of the rows of A. There is no mechanism that would make components exactly zero unless there is special structure in A or b.

The ℓ_2 norm is *strictly convex* and *smooth*, which means the optimal solution "spreads" the effort across all components rather than concentrating it. We would **not** expect many zeros.

(b) For $p = 1$, we would expect to see many components of $x^\star$ equal to zero.
Reasonable.
For $p = 1$, the problem becomes:

$$\begin{array}{ll} \text{minimize} & \|x\|_1 \\ \text{subject to} & Ax = b \end{array}$$

This is known as *basis pursuit* and is fundamental to compressed sensing and sparse recovery.

The ℓ_1 norm **promotes sparsity**. Geometrically, the ℓ_1 ball has "corners" (vertices) that lie on the coordinate axes. The feasible set $\{x \mid Ax = b\}$ is an affine subspace of dimension $n-m$. The optimal solution occurs where this subspace first touches the expanding ℓ_1 ball, which generically happens at a vertex or low-dimensional face.

Since $m \ll n$, the affine subspace has high dimension ($n - m \approx n$), and the optimal solution typically has **at most m nonzero components**. This means $n - m$ components are zero, which is "many" when $m \ll n$.

(c) For $p = \infty$, we would expect many components of $x^\star$ to take on the values $\pm\|x^\star\|_\infty$.
Reasonable.
For $p = \infty$, the problem becomes:

$$\begin{array}{ll} \text{minimize} & \|x\|_\infty \\ \text{subject to} & Ax = b \end{array}$$

The ℓ_∞ norm is $\|x\|_\infty = \max_i |x_i|$. Minimizing this means we want to make the largest component (in absolute value) as small as possible.

Geometrically, the ℓ_∞ ball is a *hypercube* $\{x \mid -t \leq x_i \leq t\}$. The optimal solution occurs where the affine subspace $\{x \mid Ax = b\}$ first touches the expanding hypercube. This generically happens on a *face* of the hypercube, where many components are at their bounds $\pm t$.

This is analogous to the **equioscillation** property in Chebyshev approximation: the optimal solution "spreads the load" so that many components achieve the maximum absolute value $\|x^\star\|_\infty$.

Since $m \ll n$, we expect **at least** $n - m$ **components** to be at the bounds $\pm\|x^\star\|_\infty$.

Summary:

p	Statement	Reasoning
2	Unreasonable	ℓ_2 is smooth; solution spreads across components
1	Reasonable	ℓ_1 promotes sparsity; expect $\leq m$ nonzeros
∞	Reasonable	ℓ_∞ spreads load; many components at $\pm\|x^\star\|_\infty$

A16.11 Control with various objectives

We consider a discrete-time linear dynamical system:

$$x_{t+1} = Ax_t + Bu_t, \quad t = 0, 1, \dots, T-1$$

with state $x_t \in \mathbf{R}^n$ and control input $u_t \in \mathbf{R}^m$. We are given initial state $x_0 = x^{\text{init}}$ and require $x_T = 0$.

Problem data: $n = 4$, $m = 2$, $T = 100$, with A, B, and x^{init} from `various_obj_regulator_data.py`.

We solve four optimization problems with different objectives measuring the control effort:

1. **Sum of squares of 2-norms:** $\sum_{t=0}^{T-1} \|u_t\|_2^2$

   This is the traditional LQR objective (with $R = I$, $Q = 0$). It penalizes large control inputs quadratically.

2. **Sum of 2-norms:** $\sum_{t=0}^{T-1} \|u_t\|_2$

   This promotes *group sparsity* — entire time steps where $u_t = 0$.

3. **Max of 2-norms:** $\max_{t=0,\dots,T-1} \|u_t\|_2$

   This minimizes the peak control effort, spreading the control action evenly across time.

4. **Sum of 1-norms:** $\sum_{t=0}^{T-1} \|u_t\|_1$

   This approximates fuel use in some applications and promotes *element-wise sparsity.*

Results:

Objective	Optimal value
(a) Sum of squares of 2-norms	0.8595
(b) Sum of 2-norms	5.9969
(c) Max of 2-norms	0.1105
(d) Sum of 1-norms	7.0443

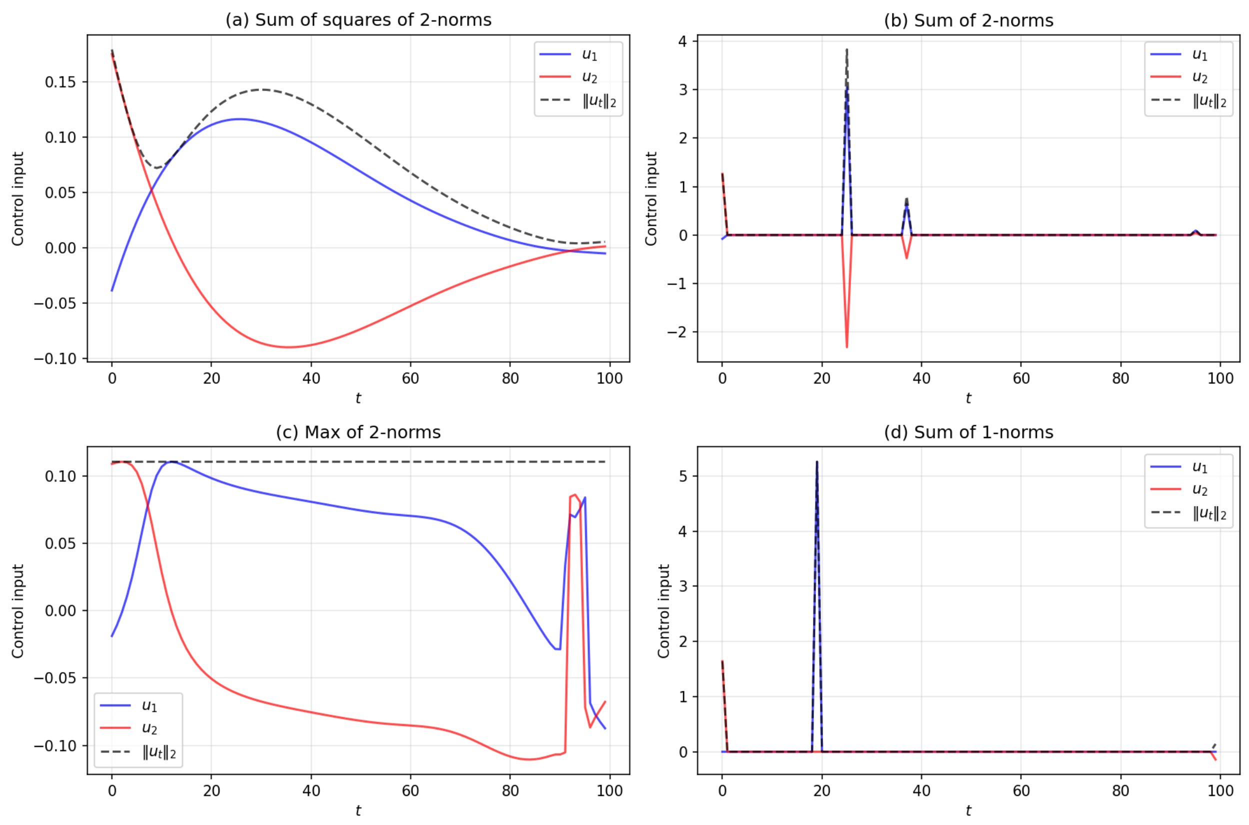

Figure 3: Optimal control inputs for each objective. Blue and red show the two components u_1, u_2; black dashed shows $\|u_t\|_2$.

Comments:

- **(a) Sum of squares:** The control is smooth and spreads effort across all time steps. This is expected since the quadratic penalty discourages large deviations.
- **(b) Sum of 2-norms:** We observe many time steps where $u_t \approx 0$ (group sparsity). The control acts in bursts, with most effort at the beginning and end.
- **(c) Max of 2-norms:** The control is spread very evenly, with $\|u_t\|_2 \approx 0.11$ for most t. This "equioscillation" behavior is characteristic of ℓ_∞-type objectives.
- **(d) Sum of 1-norms:** Many individual components are zero (element-wise sparsity). The control tends to use one actuator at a time rather than both simultaneously.

Python code:

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

# Import problem data
from various_obj_regulator_data import A, B, x_init, T, n, m

# Variables
X = cp.Variable((n, T+1))  # states x_0, ..., x_T
```

```python
U = cp.Variable((m, T))     # inputs u_0, ..., u_{T-1}

# Common constraints: dynamics, initial and final state
constraints = [
    X[:, 0] == x_init,
    X[:, T] == 0,
]
for t in range(T):
    constraints.append(X[:, t+1] == A @ X[:, t] + B @ U[:, t])

# (a) Sum of squares of 2-norms: sum ||u_t||_2^2
obj_a = cp.sum_squares(U)
prob_a = cp.Problem(cp.Minimize(obj_a), constraints)
prob_a.solve()
U_a = U.value.copy()
print(f"(a) Sum of squares of 2-norms: optimal value = {prob_a.value:.4f}")

# (b) Sum of 2-norms: sum ||u_t||_2
obj_b = cp.sum([cp.norm(U[:, t], 2) for t in range(T)])
prob_b = cp.Problem(cp.Minimize(obj_b), constraints)
prob_b.solve()
U_b = U.value.copy()
print(f"(b) Sum of 2-norms: optimal value = {prob_b.value:.4f}")

# (c) Max of 2-norms: max ||u_t||_2
obj_c = cp.max(cp.norm(U, 2, axis=0))
prob_c = cp.Problem(cp.Minimize(obj_c), constraints)
prob_c.solve()
U_c = U.value.copy()
print(f"(c) Max of 2-norms: optimal value = {prob_c.value:.4f}")

# (d) Sum of 1-norms: sum ||u_t||_1 (fuel use approximation)
obj_d = cp.sum([cp.norm(U[:, t], 1) for t in range(T)])
prob_d = cp.Problem(cp.Minimize(obj_d), constraints)
prob_d.solve()
U_d = U.value.copy()
print(f"(d) Sum of 1-norms: optimal value = {prob_d.value:.4f}")

# Plotting
t_range = np.arange(T)

fig, axes = plt.subplots(2, 2, figsize=(12, 8))

def plot_control(ax, U_opt, title, label):
    u_norms = np.linalg.norm(U_opt, axis=0)
    ax.plot(t_range, U_opt[0, :], 'b-', label='$u_1$', alpha=0.7)
    ax.plot(t_range, U_opt[1, :], 'r-', label='$u_2$', alpha=0.7)
    ax.plot(t_range, u_norms, 'k--', label=r'$\|u_t\|_2$', alpha=0.7)
    ax.set_xlabel('$t$')
    ax.set_ylabel('Control input')
    ax.set_title(title)
    ax.legend()
    ax.grid(True, alpha=0.3)

plot_control(axes[0, 0], U_a, '(a) Sum of squares of 2-norms', 'a')
plot_control(axes[0, 1], U_b, '(b) Sum of 2-norms', 'b')
plot_control(axes[1, 0], U_c, '(c) Max of 2-norms', 'c')
plot_control(axes[1, 1], U_d, '(d) Sum of 1-norms', 'd')

```

```python
plt.tight_layout()
plt.savefig('../latex/img/a16_11_controls.png', dpi=150, bbox_inches='tight')
plt.close()

print("\nPlot saved to ../latex/img/a16_11_controls.png")
```

A17.18 Option price bounds

We use arbitrage-free pricing to find bounds on the price of a collar option.

Setup:

- Risk-free asset with payoff $r = 1.05$ in every scenario, price = 1
- Underlying stock with current price $S_0 = 1$
- $m = 200$ scenarios with $S^{(i)}$ uniformly spaced from 0.5 to 2
- Call options: $K = 1.1$ (price 0.06), $K = 1.2$ (price 0.03)
- Put options: $K = 0.8$ (price 0.02), $K = 0.7$ (price 0.01)
- Collar with floor $F = 0.9$ and cap $C = 1.15$ (price unknown)

Arbitrage-free pricing framework:

A state-price vector $\pi \in \mathbf{R}^m$ satisfies $\pi \succeq 0$ and gives the "price" of each scenario. For a portfolio to be arbitrage-free, there must exist such a π where the price of each asset equals $\pi^T v$, where v is the vector of payoffs.

Let $V \in \mathbf{R}^{m \times n}$ be the payoff matrix where V_{ij} is the payoff of asset j in scenario i. The arbitrage-free condition requires:

$$V^T \pi = p$$

where p is the vector of known asset prices.

Assets and payoffs:

For scenario i with stock price $S^{(i)}$:

1. Risk-free: payoff = 1.05
2. Stock: payoff = $S^{(i)}$
3. Call $K = 1.1$: payoff = $\max(0, S^{(i)} - 1.1)$
4. Call $K = 1.2$: payoff = $\max(0, S^{(i)} - 1.2)$
5. Put $K = 0.8$: payoff = $\max(0, 0.8 - S^{(i)})$
6. Put $K = 0.7$: payoff = $\max(0, 0.7 - S^{(i)})$
7. Collar $F = 0.9$, $C = 1.15$: payoff = $\min(C, \max(F, S^{(i)}))$

Finding price bounds:

To find the range of arbitrage-free prices for the collar, we solve two LPs:

Lower bound:

$$\begin{array}{ll} \text{minimize} & v_{\text{collar}}^T \pi \\ \text{subject to} & V_{\text{known}}^T \pi = p_{\text{known}} \\ & \pi \succeq 0 \end{array}$$

Upper bound:

$$\begin{array}{ll} \text{maximize} & v_{\text{collar}}^T \pi \\ \text{subject to} & V_{\text{known}}^T \pi = p_{\text{known}} \\ & \pi \succeq 0 \end{array}$$

Results:

Quantity	Value
Lower bound on collar price	0.9850
Upper bound on collar price	1.0173

The arbitrage-free price range for the collar with $F = 0.9$ and $C = 1.15$ is:

$$\boxed{0.9850 \le p_{\text{collar}} \le 1.0173}$$

Interpretation: The collar acts like a "capped and floored" version of the stock. It provides downside protection (floor at 0.9) while giving up upside potential (cap at 1.15). The price range is fairly tight, reflecting that the traded options provide good information about the state-price distribution.

Python code:

```python
import numpy as np
import cvxpy as cp

# Problem data
r = 1.05  # risk-free rate
S0 = 1.0  # current stock price
m = 200   # number of scenarios

# Stock prices in each scenario (uniformly spaced from 0.5 to 2)
S = np.linspace(0.5, 2.0, m)

# Known asset prices
prices_known = np.array([1.0, 1.0, 0.06, 0.03, 0.02, 0.01])

# Payoff matrix for known assets (m x 6)
V_known = np.zeros((m, 6))
V_known[:, 0] = r                          # Risk-free asset
V_known[:, 1] = S                          # Stock
V_known[:, 2] = np.maximum(0, S - 1.1)     # Call K=1.1
V_known[:, 3] = np.maximum(0, S - 1.2)     # Call K=1.2
V_known[:, 4] = np.maximum(0, 0.8 - S)     # Put K=0.8
V_known[:, 5] = np.maximum(0, 0.7 - S)     # Put K=0.7

# Collar payoff: min(C, max(F, S)) with F=0.9, C=1.15
F, C = 0.9, 1.15
v_collar = np.minimum(C, np.maximum(F, S))

# State-price vector variable
pi = cp.Variable(m)

# Constraints: prices match and pi >= 0
constraints = [
    V_known.T @ pi == prices_known,
    pi >= 0
]

# Lower bound on collar price
prob_lower = cp.Problem(cp.Minimize(v_collar @ pi), constraints)
prob_lower.solve()
price_lower = prob_lower.value

# Upper bound on collar price
prob_upper = cp.Problem(cp.Maximize(v_collar @ pi), constraints)
prob_upper.solve()
```

```python
price_upper = prob_upper.value

print("Option Price Bounds for Collar (F=0.9, C=1.15)")
print("=" * 50)
print(f"Lower bound: {price_lower:.4f}")
print(f"Upper bound: {price_upper:.4f}")
print(f"\nArbitrage-free price range: [{price_lower:.4f}, {price_upper:.4f}]")

# Verify with a feasible state-price vector
print("\n--- Verification ---")
print(f"Sum of pi (should be close to 1/r = {1/r:.4f}): {np.sum(pi.value):.4f}")
print(f"All pi >= 0: {np.all(pi.value >= -1e-8)}")
```

A20.14 Optimal operation of a microgrid

We consider a microgrid with a photovoltaic (PV) array, battery storage, and grid connection. The optimization is over one day with $N = 96$ periods (15-minute intervals).

Variables:

- $p^{\text{grid}} \in \mathbf{R}^{96}$: power from grid (positive = buying, negative = selling)
- $p^{\text{batt}} \in \mathbf{R}^{96}$: battery power (positive = discharging, negative = charging)
- $q \in \mathbf{R}^{96}$: battery state of charge

Constraints:

- Power balance: $p^{\text{ld}} = p^{\text{grid}} + p^{\text{batt}} + p^{\text{pv}}$
- Battery dynamics: $q_{i+1} = q_i - \frac{1}{4}p_i^{\text{batt}}$ (with periodic condition $q_1 = q_{96} - \frac{1}{4}p_{96}^{\text{batt}}$)
- Battery limits: $0 \le q_i \le Q$, $-C \le p_i^{\text{batt}} \le D$

Objective: Minimize grid cost

$$\frac{1}{4}\left((R^{\text{buy}})^T(p^{\text{grid}})_+ - (R^{\text{sell}})^T(p^{\text{grid}})_-\right)$$

Problem data: $Q = 27$ kWh, $C = 8$ kW (max charge), $D = 10$ kW (max discharge).

(a) Optimal grid cost

$$\boxed{\text{Optimal grid cost} = \$32.77}$$

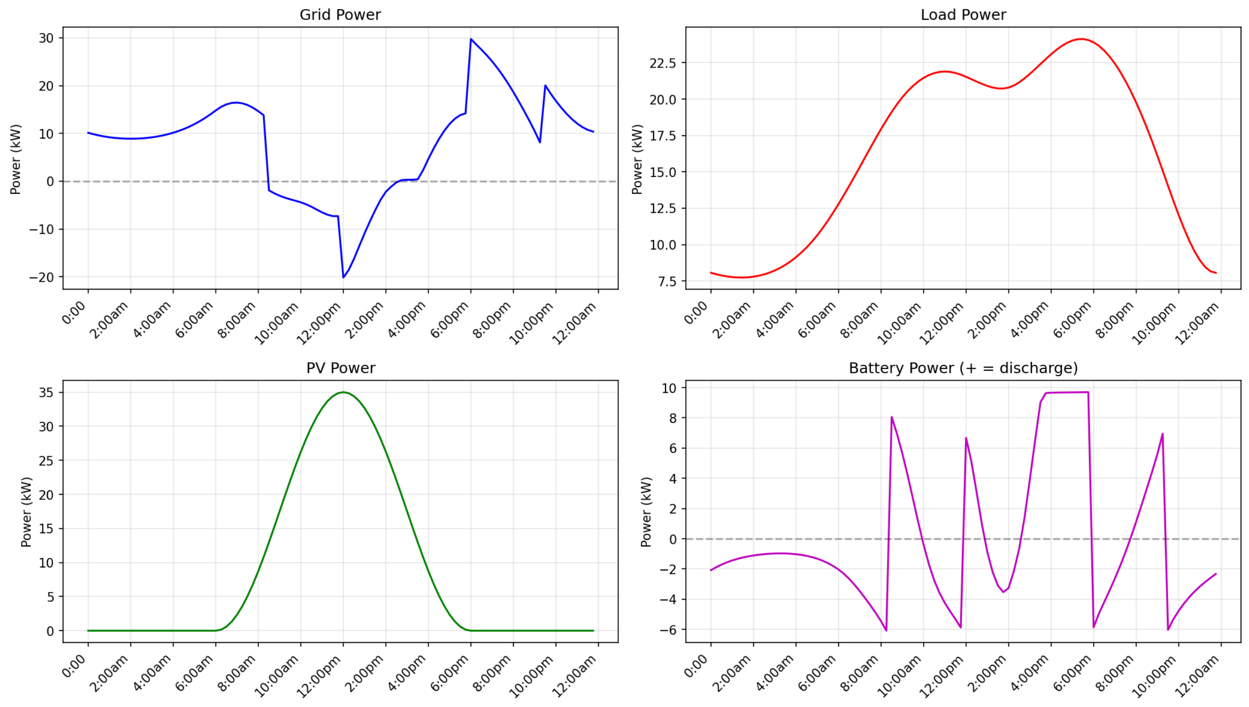

Figure 4: Optimal power profiles: grid power, load, PV output, and battery power.

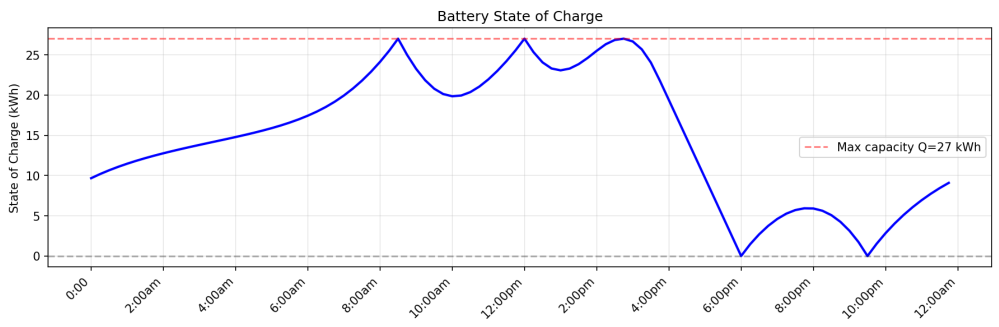

Figure 5: Battery state of charge over the day.

(b) Locational Marginal Price (LMP)

The LMP is 4ν, where ν is the dual variable for the power balance constraint:

Statistic	Value
Average LMP	\$0.2395/kWh
Max LMP	\$0.4500/kWh
Min LMP	\$0.1400/kWh

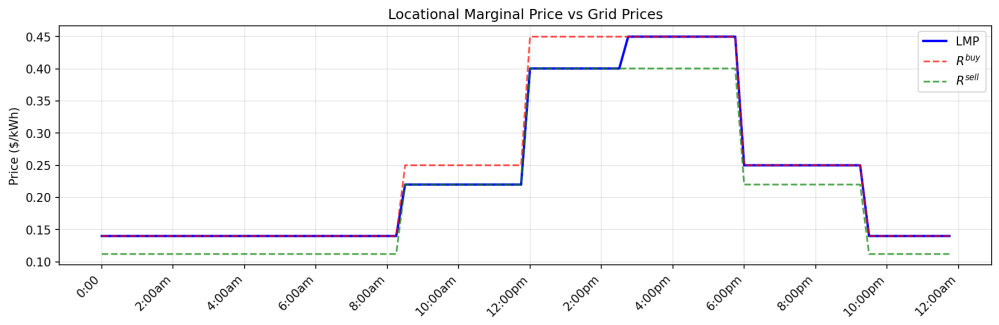

Figure 6: LMP compared to grid buy/sell prices.

Comment: The LMP equals R^{buy} when we're buying from the grid and R^{sell} when we're selling. In between, when the battery is charging/discharging or constraints are active, the LMP takes intermediate values. The LMP is always between R^{sell} and R^{buy}.

(c) LMP Payments

Using the pricing scheme where each entity pays/receives based on $\nu^T p$:

Entity	Payment (\$)
Load pays	\$430.35
PV array is paid	\$265.80
Battery is paid	\$33.48
Grid is paid	\$131.07

Balance check: $430.35 = 265.80 + 33.48 + 131.07$ ✓

The load pays for all the power it consumes at the LMP. The PV array receives payment for all the power it generates. The battery earns a profit by arbitraging — charging when LMP is low and discharging when LMP is high. The grid receives net payment for the energy imported over the day.

Python code:

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

# Import problem data (suppress plots during import)
import matplotlib
matplotlib.use('Agg')

# Manually set PLOT_FIGURES to False before importing
import sys
import importlib.util
spec = importlib.util.spec_from_file_location("microgrid_data", "microgrid_data.py")
# Load the module but override PLOT_FIGURES
exec(open("microgrid_data.py").read().replace("PLOT_FIGURES␣=␣True", "PLOT_FIGURES␣=␣False"))

# Now we have: N, R_buy, R_sell, p_pv, p_ld, D, C, Q

print(f"Problem␣size:␣N␣=␣{N}␣periods")
print(f"Battery:␣Q␣=␣{Q}␣kWh,␣C␣=␣{C}␣kW␣(max␣charge),␣D␣=␣{D}␣kW␣(max␣discharge)")

# Variables
p_buy = cp.Variable(N, nonneg=True)   # power bought from grid
p_sell = cp.Variable(N, nonneg=True)  # power sold to grid
p_grid = p_buy - p_sell               # net grid power
p_batt = cp.Variable(N)               # battery power (positive = discharging)
q = cp.Variable(N)                    # battery state of charge

# Power balance: p_ld = p_grid + p_batt + p_pv
constraints = [
    p_ld == p_buy - p_sell + p_batt + p_pv
]

# Battery dynamics: q_{i+1} = q_i - (1/4) p_batt_i
# For i = 1, ..., N-1: q[i] = q[i-1] - 0.25 * p_batt[i-1]
for i in range(1, N):
    constraints.append(q[i] == q[i-1] - 0.25 * p_batt[i-1])

# Periodicity: q[0] = q[N-1] - 0.25 * p_batt[N-1]
constraints.append(q[0] == q[N-1] - 0.25 * p_batt[N-1])

# Battery constraints
constraints.append(q >= 0)
constraints.append(q <= Q)
constraints.append(p_batt >= -C)  # charging limit
constraints.append(p_batt <= D)   # discharging limit

# Grid cost: (1/4) * (R_buy^T p_buy - R_sell^T p_sell)
cost = (1/4) * (R_buy @ p_buy - R_sell @ p_sell)

```

```python
prob = cp.Problem(cp.Minimize(cost), constraints)
prob.solve()

print(f"\n--- Part (a) Results ---")
print(f"Optimal grid cost: ${prob.value:.2f}")

p_grid_opt = p_buy.value - p_sell.value
p_batt_opt = p_batt.value
q_opt = q.value

# Get dual variable for power balance (LMP)
# Note: sign may be flipped depending on constraint formulation
nu = constraints[0].dual_value
LMP = -4 * nu  # Convert to $/kWh (negate due to constraint form)

print(f"\n--- Part (b) LMP Analysis ---")
print(f"Average LMP: ${np.mean(LMP):.4f}/kWh")
print(f"Max LMP: ${np.max(LMP):.4f}/kWh")
print(f"Min LMP: ${np.min(LMP):.4f}/kWh")

# Part (c) - LMP payments
payment_load = np.dot(LMP, p_ld)
payment_pv = np.dot(LMP, p_pv)
payment_batt = np.dot(LMP, p_batt_opt)
payment_grid = np.dot(LMP, p_grid_opt)

print(f"\n--- Part (c) LMP Payments ---")
print(f"Load pays: ${payment_load:.2f}")
print(f"PV array is paid: ${payment_pv:.2f}")
print(f"Battery is paid: ${payment_batt:.2f}")
print(f"Grid is paid: ${payment_grid:.2f}")
print(f"Balance check (should be ~0): ${payment_load - payment_pv - payment_batt -
    payment_grid:.4f}")

# Plotting
fig_size = (12, 3)
xtick_vals = [0, 8, 16, 24, 32, 40, 48, 56, 64, 72, 80, 88, 96]
xtick_labels = ["0:00", "2:00am", "4:00am", "6:00am", "8:00am", "10:00am",
                "12:00pm", "2:00pm", "4:00pm", "6:00pm", "8:00pm", "10:00pm", "
                    12:00am"]
t = np.arange(N)

# Plot 1: Powers
fig, axes = plt.subplots(2, 2, figsize=(14, 8))

axes[0, 0].plot(t, p_grid_opt, 'b-', label='$p^{grid}$')
axes[0, 0].axhline(0, color='k', linestyle='--', alpha=0.3)
axes[0, 0].set_ylabel('Power (kW)')
axes[0, 0].set_title('Grid Power')
axes[0, 0].set_xticks(xtick_vals)
axes[0, 0].set_xticklabels(xtick_labels, rotation=45, ha='right')
axes[0, 0].grid(True, alpha=0.3)

axes[0, 1].plot(t, p_ld, 'r-', label='$p^{load}$')
axes[0, 1].set_ylabel('Power (kW)')
axes[0, 1].set_title('Load Power')
axes[0, 1].set_xticks(xtick_vals)
axes[0, 1].set_xticklabels(xtick_labels, rotation=45, ha='right')
axes[0, 1].grid(True, alpha=0.3)
```

```python

axes[1, 0].plot(t, p_pv, 'g-', label='$p^{pv}$')
axes[1, 0].set_ylabel('Power (kW)')
axes[1, 0].set_title('PV Power')
axes[1, 0].set_xticks(xtick_vals)
axes[1, 0].set_xticklabels(xtick_labels, rotation=45, ha='right')
axes[1, 0].grid(True, alpha=0.3)

axes[1, 1].plot(t, p_batt_opt, 'm-', label='$p^{batt}$')
axes[1, 1].axhline(0, color='k', linestyle='--', alpha=0.3)
axes[1, 1].set_ylabel('Power (kW)')
axes[1, 1].set_title('Battery Power (+ = discharge)')
axes[1, 1].set_xticks(xtick_vals)
axes[1, 1].set_xticklabels(xtick_labels, rotation=45, ha='right')
axes[1, 1].grid(True, alpha=0.3)

plt.tight_layout()
plt.savefig('../latex/img/a20_14_powers.png', dpi=150, bbox_inches='tight')
plt.close()

# Plot 2: Battery state of charge
plt.figure(figsize=(12, 4))
plt.plot(t, q_opt, 'b-', linewidth=2)
plt.axhline(Q, color='r', linestyle='--', alpha=0.5, label=f'Max capacity Q={Q} 
    kWh')
plt.axhline(0, color='k', linestyle='--', alpha=0.3)
plt.ylabel('State of Charge (kWh)')
plt.title('Battery State of Charge')
plt.xticks(xtick_vals, xtick_labels, rotation=45, ha='right')
plt.legend()
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.savefig('../latex/img/a20_14_soc.png', dpi=150, bbox_inches='tight')
plt.close()

# Plot 3: LMP vs grid prices
plt.figure(figsize=(12, 4))
plt.plot(t, LMP, 'b-', linewidth=2, label='LMP')
plt.plot(t, R_buy, 'r--', alpha=0.7, label='$R^{buy}$')
plt.plot(t, R_sell, 'g--', alpha=0.7, label='$R^{sell}$')
plt.ylabel('Price ($/kWh)')
plt.title('Locational Marginal Price vs Grid Prices')
plt.xticks(xtick_vals, xtick_labels, rotation=45, ha='right')
plt.legend()
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.savefig('../latex/img/a20_14_lmp.png', dpi=150, bbox_inches='tight')
plt.close()

print("\nPlots saved to ../latex/img/")
```