A6.2 Minimax rational fit to the exponential

As shown in problem 6.9, this is a quasiconvex optimization problem. It can be solved using bisection on the objective value α, combined with a feasibility LP at each step.

For a given $\alpha > 0$, the feasibility problem is:

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

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:

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

Expanding:

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

Rearranging, for each i:

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

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

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

Results.

Bisection yields:

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

Optimal objective value: $\alpha^\star \approx 0.0233$. The following plots are also obtained.

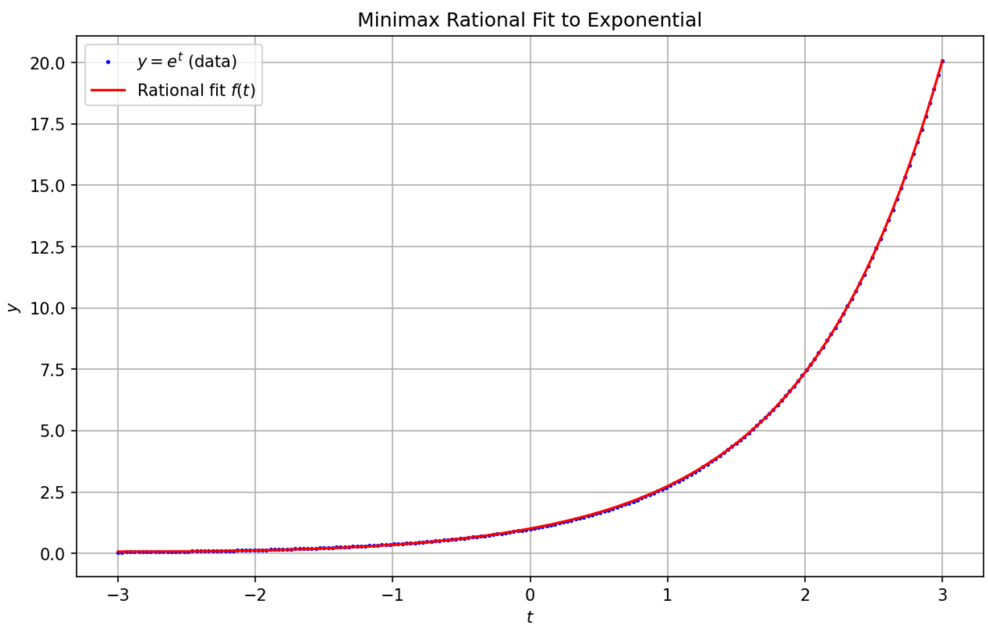

Figure 1: Data (blue dots) and rational fit $f(t)$ (red line).

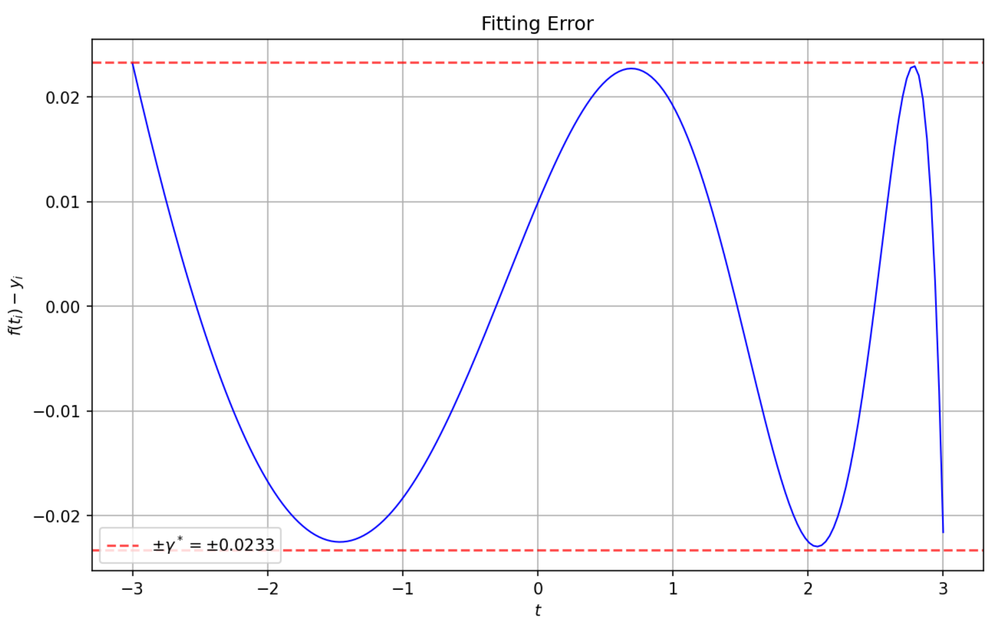

Figure 2: Fitting error $f(t_i) - y_i$.

Code:

```
def check_feasibility(alpha, Tpowers, y):
    a = cp.Variable(3)
    b = cp.Variable(2)
    q_coeffs = cp.hstack([1, b])
    p_vals = Tpowers @ a
    q_vals = Tpowers @ q_coeffs
```

```python
    constraints = [
        p_vals <= cp.multiply(y + alpha, q_vals),
        p_vals >= cp.multiply(y - alpha, q_vals),
    ]
    prob = cp.Problem(cp.Minimize(0), constraints)
    prob.solve()
    return prob.status == 'optimal', a.value, b.value

# Bisection
l, u = 0.0, np.exp(3)
while u - l >= 1e-3:
    alpha = (l + u) / 2
    feasible, a, b = check_feasibility(alpha, Tpowers, y)
    if feasible:
        u, a_opt, b_opt, alpha_opt = alpha, a, b, alpha
    else:
        l = alpha
```

A6.27 Properties of least-p-norm solutions

(a) Unreasonable. The ℓ_2 norm is smooth, so the solution $x^\star = A^\dagger b$ spreads effort across all components. No sparsity is expected.

(b) Reasonable. The ℓ_1 ball has corners on the axes. The optimal point typically lies at a corner, giving at most m nonzeros. Since $m \ll n$, most components are zero.

(c) Reasonable. The ℓ_∞ ball is a box. The optimal point lies on a face, where many components hit the bounds $\pm\|x^\star\|_\infty$.

A16.11 Control with various objectives

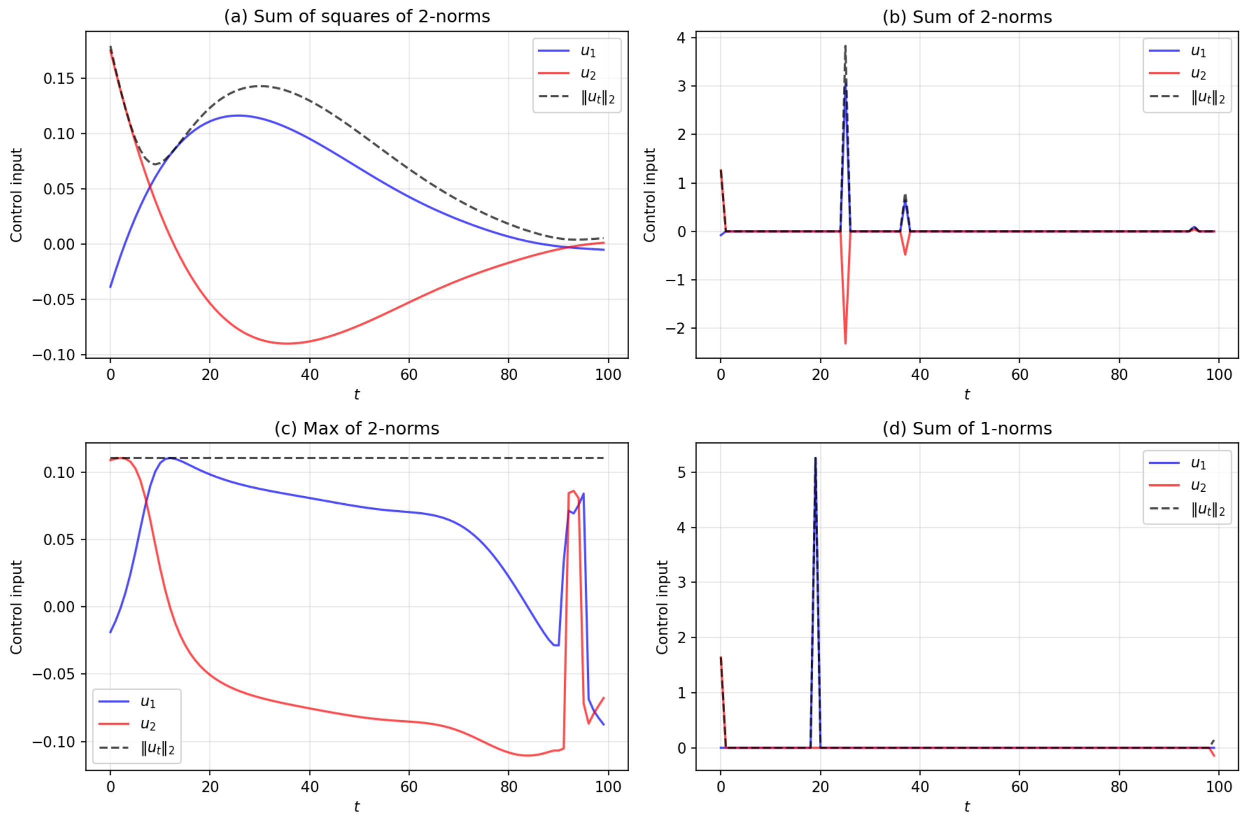

Figure 3: Optimal control inputs for each objective. Blue and red show the two components u_1, u_2; black dashed shows $\|u_t\|_2$.

- **(a) Sum of squares:** As seen in subplot (a), the control is smooth and spreads effort across all time steps. The quadratic penalty discourages large deviations.
- **(b) Sum of 2-norms:** As seen in subplot (b), many time steps have $u_t \approx 0$ (group sparsity). The control acts in bursts at the beginning and end.
- **(c) Max of 2-norms:** As seen in subplot (c), the control is spread evenly with $\|u_t\|_2 \approx 0.11$ for most t.
- **(d) Sum of 1-norms:** As seen in subplot (d), many individual components are zero (element-wise sparsity). The control uses one actuator at a time.

Code:

```
X = cp.Variable((n, T+1))  # states x_0, ..., x_T
U = cp.Variable((m, T))    # inputs u_0, ..., u_{T-1}

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
```

```
obj_a = cp.sum_squares(U)

# (b) Sum of 2-norms
obj_b = cp.sum([cp.norm(U[:, t], 2) for t in range(T)])

# (c) Max of 2-norms
obj_c = cp.max(cp.norm(U, 2, axis=0))

# (d) Sum of 1-norms
obj_d = cp.sum([cp.norm(U[:, t], 1) for t in range(T)])
```

A17.18 Option price bounds

Given a risk-free asset, stock, two calls ($K = 1.1, 1.2$), and two puts ($K = 0.8, 0.7$) with known prices, find the arbitrage-free price bounds for a collar with floor $F = 0.9$ and cap $C = 1.15$.

Solution: A state-price vector $\pi \in \mathbf{R}^m$ with $\pi \succeq 0$ assigns a "price" to each scenario. For arbitrage-free pricing:

$$p_j = \sum_{i=1}^{m} \pi_i V_{ij} = v_j^T \pi$$

where V_{ij} is the payoff of asset j in scenario i.

Let $V_{\text{known}} \in \mathbf{R}^{m \times 6}$ be the payoff matrix for the 6 known assets, $p_{\text{known}} \in \mathbf{R}^6$ their prices, and $v_{\text{collar}} \in \mathbf{R}^m$ the collar payoffs. The price bounds are:

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: Same with maximize.

Results:

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

Code:

```python
# Payoff matrix for known assets
V_known[:, 0] = r                          # Risk-free
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))
v_collar = np.minimum(1.15, np.maximum(0.9, S))

# State-price vector and constraints
pi = cp.Variable(m)
constraints = [V_known.T @ pi == prices_known, pi >= 0]

# Price bounds
prob_lower = cp.Problem(cp.Minimize(v_collar @ pi), constraints)
prob_upper = cp.Problem(cp.Maximize(v_collar @ pi), constraints)
```

A20.14 Optimal operation of a microgrid

A microgrid with PV array, battery storage, and grid connection is optimized over one day ($N = 96$ periods of 15 minutes).

Variables: p^{grid} (grid power, positive = buying), p^{batt} (battery power, positive = discharging), q (state of charge).

Constraints:

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

Objective: Minimize grid cost $\frac{1}{4}\left((R^{\text{buy}})^T(p^{\text{grid}})_+ - (R^{\text{sell}})^T(p^{\text{grid}})_-\right)$
Data: $Q = 27$ kWh, $C = 8$ kW, $D = 10$ kW.

(a) Optimal grid cost

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

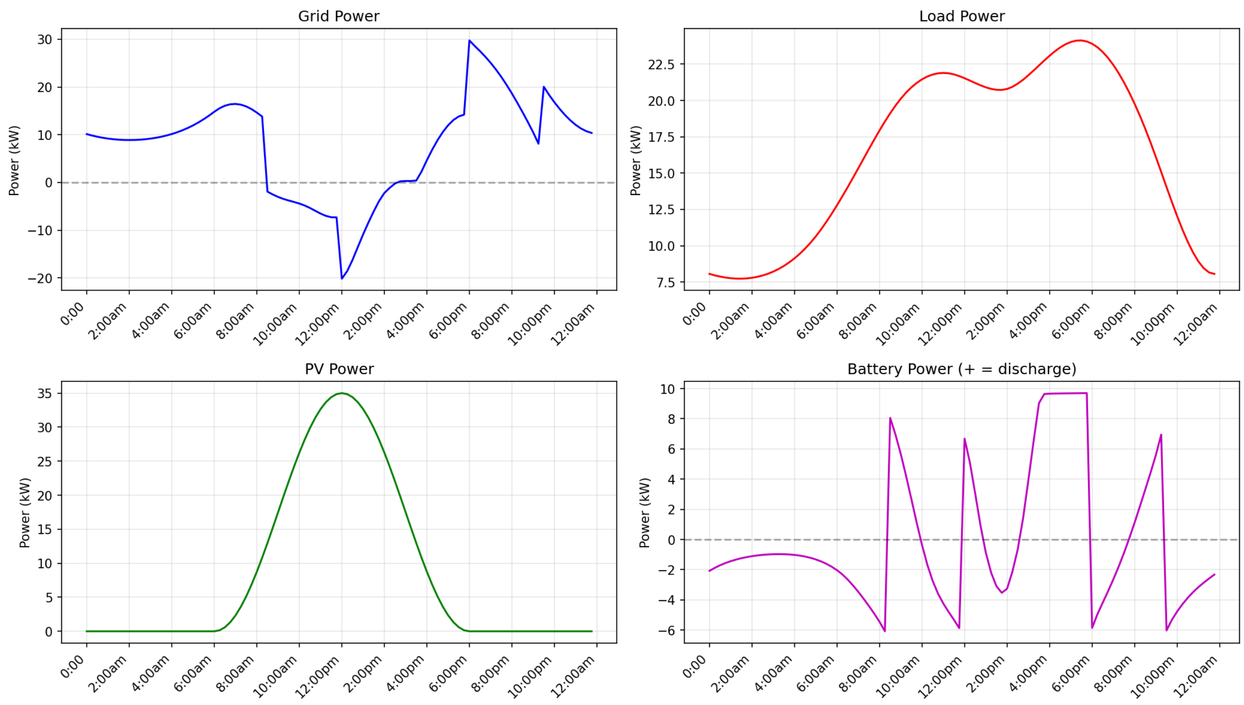

Figure 4: Optimal power profiles.

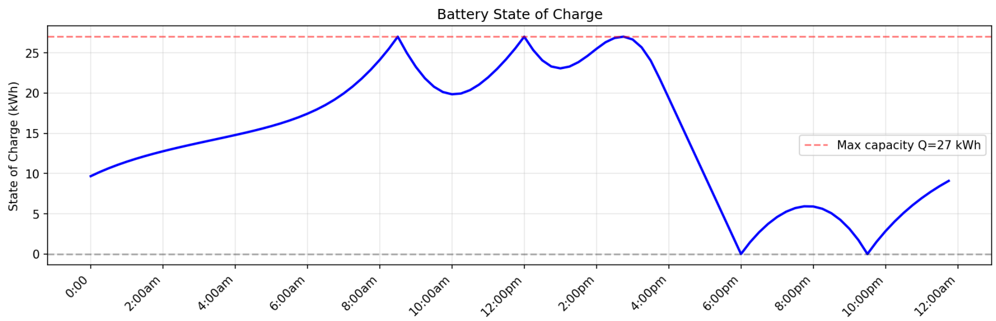

Figure 5: Battery state of charge.

(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

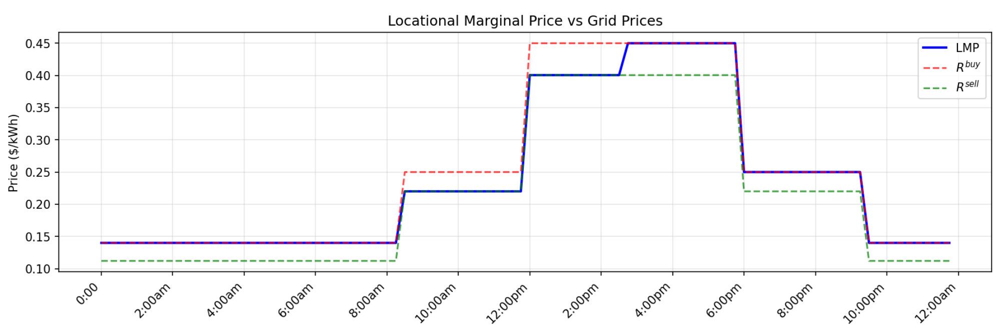

Figure 6: LMP compared to grid buy/sell prices.

The LMP equals R^{buy} when buying and R^{sell} when selling. The battery arbitrages by charging when LMP is low and discharging when high.

(c) LMP Payments

Each entity pays/receives based on LMP $\times$ power:

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

Balance: $430.35 = 265.80 + 33.48 + 131.07$

Code:

```python
# Variables (p_grid = p_buy - p_sell for DCP compliance)
p_buy = cp.Variable(N, nonneg=True)
p_sell = cp.Variable(N, nonneg=True)
p_batt = cp.Variable(N)
q = cp.Variable(N)

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

# Battery dynamics (periodic)
for i in range(1, N):
    constraints.append(q[i] == q[i-1] - 0.25 * p_batt[i-1])
constraints.append(q[0] == q[N-1] - 0.25 * p_batt[N-1])

# Bounds
constraints += [q >= 0, q <= Q, p_batt >= -C, p_batt <= D]

# Objective
cost = (1/4) * (R_buy @ p_buy - R_sell @ p_sell)
prob = cp.Problem(cp.Minimize(cost), constraints)

# LMP from dual variable
LMP = -4 * constraints[0].dual_value
```