EE364a: Convex Optimization I S. Boyd
March 14–15, 15–16, or 16–17, 2019

# Final Exam Solutions

1. *Fitting a periodic Poisson distribution to data.* We model the (random) number of times that some type of event occurs in each hour of the day as independent Poisson variables, with

$$\mathbf{Prob}(k \text{ events occur}) = e^{-\lambda_t}\frac{\lambda_t^k}{k!}, \quad k = 0, 1, \ldots,$$

with parameter $\lambda_t \geq 0$, $t = 1, \ldots, 24$. (For $\lambda_t = 0$, $k = 0$ events occur with probability one.) Here $t$ denotes the hour, with $t = 1$ corresponding to the hour from midnight to 1AM, and $t = 24$ the hour between 11PM and midnight. (This is the periodic Poisson distribution in the title.) The parameter $\lambda_t$ is the expected value of the number of events that occur in hour $t$; it can be thought of as the rate of occurence of the events in hour $t$.

Over one day we observe the numbers of events $N_1, \ldots, N_{24}$.

   (a) *Maximum likelihood estimate of parameters.* What is the maximum likelihood estimate of the parameters $\lambda_1, \ldots, \lambda_{24}$? *Hint.* There is a simple analytical solution. You should consider the cases $N_t > 0$ and $N_t = 0$ separately.

   (b) *Regularized maximum likelihood estimate of parameters.* In many applications it is reasonable to assume that $\lambda_t$ varies smoothly over the day; for example, the rate of occurence of events for 3PM–4PM is not too different from the rate of occurence for 4PM–5PM. To obtain a smooth estimate of $\lambda_t$ we maximize the log likelihood minus the regularization term

$$\rho\left(\sum_{t=1}^{23}(\lambda_{t+1} - \lambda_t)^2 + (\lambda_1 - \lambda_{24})^2\right),$$

   where $\rho \geq 0$. Explain how to find the values $\lambda_1, \ldots, \lambda_{24}$ using convex optimization. If you change variables, explain.

   (c) What happens as $\rho \to \infty$? You can give a very short answer, with an informal argument. *Hint.* As in part (a), there is a simple analytical solution.

   (d) *Numerical example.* Over one day, we observe

$$N = (0, 4, 2, 2, 3, 0, 4, 5, 6, 6, 4, 1, 4, 4, 0, 1, 3, 4, 2, 0, 3, 2, 0, 1).$$

   Find the regularized maximum likelihood parameters for $\rho \in \{0.1, 1, 10, 100\}$ using CVX*, and plot $\lambda_t$ versus $t$ for each value of $\rho$.

(e) *Choosing the hyper-parameter value by out-of-sample test.* One way to choose the value of $\rho$ is to see which of the models found in part (d) has the highest log likelihood on a test set, *i.e.*, another day's data, that was not used to create the model. For each of the 4 values of the parameters you estimated in part (d), evaluate the log likelihood of another day's number of events,

$$N^{\text{test}} = (0, 1, 3, 2, 3, 1, 4, 5, 3, 1, 4, 3, 5, 5, 2, 1, 1, 1, 2, 0, 1, 2, 1, 0).$$

Which hyper-parameter value $\rho$ would you choose?

**Solution.**

(a) The log-likelihood of observing $N_t$ events is $-\lambda_t + N_t \log \lambda_t - \log N_t!$. Since these events are independent, we add these to get the log-likelihood of the collection of observed events, so the negative log-likelihood of the collection of observed events is

$$\sum_{t=1}^{24} \left(\lambda_t - N_t \log \lambda_t + \log N_t!\right).$$

The maximum likelihood estimate of $\lambda_t$ minimizes the negative log-likelihood of observing $N_1, \ldots, N_{24}$, which can be found by solving the optimization problem

$$\begin{array}{ll} \text{minimize} & \sum_{t=1}^{24} \left(\lambda_t - N_t \log \lambda_t + \log N_t!\right) \\ \text{subject to} & \lambda \succeq 0, \end{array}$$

with variables $\lambda_1, \ldots, \lambda_{24}$. Note that when $N_t > 0$, the $\log \lambda_t$ term imposes an implicit constraint that $\lambda_t > 0$; we need the explicit inequality $\lambda \succeq 0$ to cover the case when $N_t = 0$. (We did not penalize those who omitted the explicit inequality.)

This problem is separable in $t$, so we can separately minimize over each $\lambda_t$. Let's first assume that $N_t > 0$. Then the minimizer satisfies $1 - N_t/\lambda_t = 0$, so $\lambda_t = N_t$. For $N_t = 0$, we minimize $\lambda_t$, subject to $\lambda_t \geq 0$. This gives $\lambda_t = 0$. So in all cases the ML estimate is $\lambda_t = N_t$. This is very natural: If you observe $N_t$ events, you guess that this is the mean of the number of events that occur.

(b) We solve the convex optimization problem

$$\begin{array}{ll} \text{minimize} & \sum_{t=1}^{24} \left(\lambda_t - N_t \log \lambda_t\right) + \rho \left(\sum_{t=1}^{23} (\lambda_{t+1} - \lambda_t)^2 + (\lambda_1 - \lambda_{24})^2\right) \\ \text{subject to} & \lambda \succeq 0, \end{array}$$

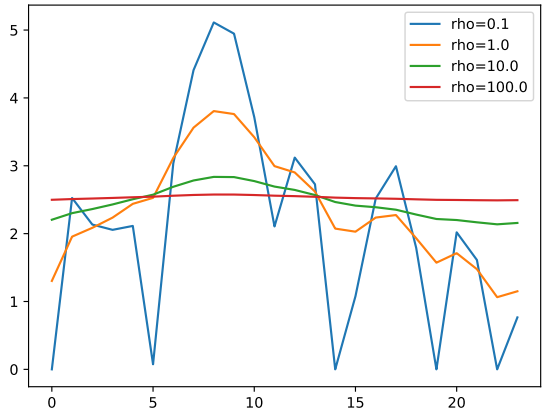
with variable $\lambda \in \mathbf{R}^{24}$. (We drop the constant term that does not depend on $\lambda_t$.)

(c) As $\rho \to \infty$, the $\lambda_t$ all become equal, so we have $\lambda = \tilde{\lambda}\mathbf{1}$, resulting in the problem

$$\begin{array}{ll} \text{minimize} & 24\tilde{\lambda} - \left(\sum_{t=1}^{24} N_t\right) \log \tilde{\lambda} \\ \text{subject to} & \tilde{\lambda} \geq 0. \end{array}$$

The solution is the constant Poisson model with $\lambda_t = \tilde{\lambda} = \overline{N} = \sum_\tau N_\tau/24$, the average rate of hourly occurence of the events over the whole day.

(d) The estimated rates are shown below. As expected, the larger $\rho$ is, the smoother $\lambda$ is.




(e) We get the following log likelihood values on the test set: $(-83.44, -37.75, -41.71, -43.76)$. We would choose $\rho = 1$, since it gets the highest log-likelihood on the test set.

Python solution for all parts:

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

T = 24
N = np.array([0, 4, 2, 2, 3, 0, 4, 5, 6, 6, 4, 1,
              4, 4, 0, 1, 3, 4, 2, 0, 3, 2, 0, 1])

lambd = cvx.Variable(T, nonneg=True)
rho = cvx.Parameter(nonneg=True)
objective = cvx.sum(lambd) - N@cvx.log(lambd) +\
    rho * (lambd[0] - lambd[-1])**2 +\
    rho * cvx.sum_squares(cvx.diff(lambd))
problem = cvx.Problem(cvx.Minimize(objective))

lambdas, rhovalues = [], [0.1, 1, 10, 100]
for rhovalue in rhovalues:
    rho.value = rhovalue
    problem.solve()
    plt.plot(np.arange(T), lambd.value, label="rho=%.1f" % rhovalue)
    lambdas.append(lambd.value)
plt.legend()
```

```
plt.savefig("../figures/time_varying_poisson.pdf")

from scipy.special import factorial
Ntest = np.array([0, 1, 3, 2, 3, 1, 4, 5, 3, 1, 4, 3,
                  5, 5, 2, 1, 1, 1, 2, 0, 1, 2, 1, 0])
for rhovalue, lambd in zip(rhovalues, lambdas):
    test_logprob = np.sum(np.log(
        np.exp(-lambd) * lambd**Ntest / factorial(Ntest)
    ))
    print(f"rho = {rhovalue}, log likelihood = {test_logprob}")
```

Julia solution for all parts:

```
using Convex, ECOS, PSPlot

# problem data
T = 24
N = [0, 4, 2, 2, 3, 0, 4, 5, 6, 6, 4, 1,
     4, 4, 0, 1, 3, 4, 2, 0, 3, 2, 0, 1]
N_test = [0, 1, 3, 2, 3, 1, 4, 5, 3, 1, 4, 3,
          5, 5, 2, 1, 1, 1, 2, 0, 1, 2, 1, 0]

# circular difference
circdiff(x) = x - [x[2:end]; x[1]]

# solve problems
lambd = Variable(T)
lambds, rhos = [], [0.1, 1, 10, 100]
for rho in rhos
    ll = sum(lambd - N .* log(lambd))
    reg = rho * sumsquares(circdiff(lambd))
    prob = minimize(ll + reg)
    solve!(prob, ECOSSolver(eps=1e-8))
    plot(1:T, lambd.value, label="rho = $(rho)")
    push!(lambds, lambd.value)
end
legend()
printfig("time_varying_poisson.eps")

# compute log likelihoods
log_fact(n) = sum(log.(1:n))
for (rho, lambd) in zip(rhos, lambds)
    ll = sum(-lambd + N_test .* log.(lambd) - log_fact.(N_test))
```

```
    println("rho = $(rho), log likelihood = $(ll)")
end
```

Matlab solution for all parts:

```
T = 24;
N = [0,4,2,2,3,0,4,5,6,6,4,1,4,4,0,1,3,4,2,0,3,2,0,1];
hold on
lambds = [];
rhos = [0.1,1,10,100];
for rho = rhos
cvx_begin
    variable lambd(T)
minimize(sum(lambd)-N*log(lambd) +
 rho*(lambd(1)-lambd(T))^2+ rho*sum((lambd(1:T-1)-lambd(2:T)).^2))
    subject to
        lambd >= 0
cvx_end
plot(lambd)
lambds = [lambds lambd];
end
legend(["rho=0.1","rho=1","rho=10","rho=100"])
saveas(gcf,'time_varying_poisson.eps','epsc')

Ntest = [0,1,3,2,3,1,4,5,3,1,4,3,5,5,2,1,1,1,2,0,1,2,1,0]';
for i = 1:4
    lam =lambds(:,i);
    test_logprob = sum(log(exp(-lam).*lam.^Ntest./factorial(Ntest)))
    rhos(i)
end
```

2. *Currency exchange.* An entity (such as a multinational corporation) holds $n = 10$ currencies, with $c_i^{\text{init}} \geq 0$ denoting the number of units of currency $i$. The currencies are, in order, USD, EUR, GBP, CAD, JPY, CNY, RUB, MXN, INR, and BRL. Our goal is to exchange currencies on a market so that, after the exchanges, we hold at least $c_i^{\text{req}}$ units of each currency $i$.

   The exchange rates are given by $F \in \mathbf{R}^{n\times n}$, where $F_{ij}$ is the units of currency $j$ it costs to buy one unit of currency $i$. We call $1/F_{ij}$ the bid price for currency $j$ in terms of currency $i$, and $F_{ji}$ the ask price for currency $j$ in terms of currency $i$.

   For example, suppose that $F_{12} = 0.88$ and $F_{21} = 1.18$. This means that it takes 0.88 EUR to buy one USD, and it takes 1.18 USD to buy one EUR; the bid and ask prices for EUR in USD are 1.1364 USD and 1.1800 USD, respectively.

   We will value a set of currency holdings in USD, by valuing each unit of currency $j$ at the geometric mean of the bid and ask price in USD, $\sqrt{F_{j1}/F_{1j}}$. In our example above, we would value one EUR as $\sqrt{1.1364 \cdot 1.1800} = 1.1580$ USD.

   We let $X \in \mathbf{R}_+^{n\times n}$ denote the currency exchanges that we carry out, with $X_{ij} \geq 0$ the amount of currency $j$ we exchange on the market for currency $i$, for which we obtain $X_{ij}/F_{ij}$ of currency $i$. (You can assume that $X_{ii} = 0$.) The total of each currency $j$ that we exchange into other currencies cannot exceed our initial holdings, $c_j^{\text{init}}$. After the currency exchange, we must end up with at least $c_i^{\text{req}}$ of currency $i$. (The post-exchange amount we hold of currency $i$ is our original holding $c_i^{\text{init}}$, minus the total we exchange into other currencies, plus the total amount we obtain from exchanging other currencies into currency $i$.)

   The cost of the exchanges is the decrease in value between the currency holdings before and after the exchanges, in USD. The cost can be interpreted as the transaction costs incurred by crossing the bid-ask spread (*i.e.*, if the bid and the ask were the same, there would be no cost.)

   Find the currency exchanges $X^\star$ that minimize the currency exchange cost for the data in `currency_exchange_data.*`. (These data are based on real exchange rates, but with artificially large spreads, to make sure that you don't encounter any numerical issues.) Explain your method, and give the optimal value, *i.e.*, the cost obtained.

   **Solution.**

   We exchange $X_{ij}$ of currency $j$ into currency $i$, so the total we exchange of currency $j$ is $\sum_i X_{ij}$. So the vector of totals of currencies we exchange is $X^T\mathbf{1}$.

   For the exchange of $X_{ij}$ of currency $j$, we receive $X_{ij}/F_{ij}$ of currency $i$. The total we receive of currency $i$ is $\sum_j X_{ij}/F_{ij}$. The vector of proceeds from our exchanges is given by $(X/F)\mathbf{1}$, where $F/X$ is meant elementwise. The post-exchange currency holdings $z \in \mathbf{R}_+^n$ are given by

$$z = (X/F)\mathbf{1} - X^T\mathbf{1} + c^{\text{init}}.$$

The cost of the exchanges is the decrease in value after the exchanges, which is given by

$$\sum_{j=1}^{n}(c^{\text{init}}-z)_j\sqrt{F_{j1}/F_{1j}}.$$

We can now assemble the problem, which turns out to be an LP,

$$\begin{array}{ll} \text{minimize} & \sum_{j=1}^{n}(c^{\text{init}}-z)_j\sqrt{F_{j1}/F_{1j}} \\ \text{subject to} & X_{ij}\geq 0, \quad \mathbf{diag}(X)=0, \\ & z=(X/F)\mathbf{1}-X^T\mathbf{1}+c^{\text{init}}, \\ & X^T\mathbf{1}\preceq c^{\text{init}}, \\ & z\succeq c^{\text{req}}, \end{array} \tag{1}$$

where $X$ and $z$ are the optimization variables.

For the given data, the minimum currency exchange cost is 7.7 USD. The optimal $X$ is

$$X^\star=\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 545 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 17 & 0 & 0 & 0 & 0 & 0 & 727 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 369 & 0 & 0 \\ 16 & 0 & 0 & 0 & 182 & 0 & 0 & 0 & 0 & 182 \\ 19 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 510 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The initial value $c^{\text{init}}$ is $(1818, 1636, 1455, 1273, 1091, 909, 727, 545, 364, 182)$. The total of each currency that we exchange are $X^{\star T}\mathbf{1}=(561, 0, 0, 545, 182, 0, 727, 369, 0, 182)$. This means that we don't exchange any of our EUR, GBP, CNY, or INR. The optimal final value $z$ is $(1257, 1636, 1772, 727, 909, 1091, 1273, 1455, 1636, 1818)$.

The following Python code implements the solution:

```
import numpy as np
import cvxpy as cp
from currency_exchange_data import *

X = cp.Variable((n,n))
z = (X/F)@np.ones(n) - X.T@np.ones(n) + c_init
cost = (c_init - z)@np.sqrt(F[:,0]/F[0,:])
prob = cp.Problem(cp.Minimize(cost),
                  [X >= 0,
                   cp.diag(X) == 0,
```

```
                  X.T@np.ones(n) <= c_init,
                  z >= c_req])
result = prob.solve()
print("The minimum cost is ", result, " USD")
```

The following MATLAB code implements the solution:

```
currency_exchange_data;

cvx_begin
    variable X(n,n);
    z = (X./F)*ones(n, 1) - X.'*ones(n, 1) + c_init;
    cost = sqrt(F(:, 1).'./F(1, :))*(c_init - z);
    minimize (cost);
    X >= 0;
    diag(X) == 0;
    X.'*ones(n, 1) <= c_init;
    z >= c_req;
cvx_end

fprintf("The minimum cost is %d USD\n", cvx_optval)
```

The following Julia code implements the solution:

```
using Convex;
using SCS;
using LinearAlgebra;

include("currency_exchange_data.jl")

X = Convex.Variable((n, n));
z = (X ./ F) * ones(n) - X' * ones(n) + c_init;
cost = Convex.dot((c_init - z), sqrt.(F[1:end, 1] ./ F[1, 1:end]));

constraints = [
    X >= 0,
    Convex.diag(X) == 0,
    X' * ones(n) <= c_init,
    z >= c_req
];

problem = Convex.minimize(cost, constraints);
```

```
solve!(problem, SCSSolver());

print("The minimum cost is ", problem.optval, " USD");
```

3. *Optimal operation of a microgrid.* We consider a small electrical microgrid that consists of a photo-voltaic (PV) array, a storage device (battery), a load, and a connection to an external grid. We will optimize the operation of the microgrid over one day, in 15 minute increments, so all powers, and the battery charge, are represented as vectors in $\mathbf{R}^{96}$. The load power is $p^{\mathrm{ld}}$, which is nonnegative and known. The power that we take from the external grid is $p^{\mathrm{grid}}$; $p_i^{\mathrm{grid}} \geq 0$ means we are consuming power from the grid, and $p_i^{\mathrm{grid}} < 0$ means we are sending power back into the grid, in time period $i$. The PV array output, which is nonnegative and known, is denoted as $p^{\mathrm{pv}}$. The battery power is $p^{\mathrm{batt}}$, with $p_i^{\mathrm{batt}} \geq 0$ meaning the battery is discharging, and $p_i^{\mathrm{batt}} < 0$ meaning the battery is charging. These powers must balance in all periods, *i.e.*, we have

$$p^{\mathrm{ld}} = p^{\mathrm{grid}} + p^{\mathrm{batt}} + p^{\mathrm{pv}}.$$

(This is called the power balance constraint. The lefthand side is the load power, and the righthand side is the sum of the power coming from the grid, the battery, and the PV array.) All powers are given in kW.

The battery state of charge is given by $q \in \mathbf{R}^{96}$. It must satisfy $0 \leq q_i \leq Q$ for all $i$, where $Q$ is the battery capacity (in kWh). The battery dynamics are

$$q_{i+1} = q_i - (1/4)p_i^{\mathrm{batt}}, \quad i = 1, \ldots, 95, \quad q_1 = q_{96} - (1/4)p_{96}^{\mathrm{batt}}.$$

(The last equation means that we seek a periodic operation of the microgrid.) The battery power must satisfy $-C \leq p_i^{\mathrm{batt}} \leq D$ for all $i$, where $C$ and $D$ are (positive) known maximum charge and maximum discharge rates.

When we buy power (*i.e.*, $p_i^{\mathrm{grid}} \geq 0$) we pay for it at the rate of $R_i^{\mathrm{buy}}$ (in \$/kWh). When we sell power to the grid (*i.e.*, $p_i^{\mathrm{grid}} < 0$) we are paid for it at the rate of $R_i^{\mathrm{sell}}$. These (positive) prices vary with time period, and are known. The total cost of the grid power (in \$) is

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

where $(p^{\mathrm{grid}})_+ = \max\{p^{\mathrm{grid}}, 0\}$ and $(p^{\mathrm{grid}})_- = \max\{-p^{\mathrm{grid}}, 0\}$ (elementwise). You can assume that $R_i^{\mathrm{buy}} > R_i^{\mathrm{sell}} > 0$, *i.e.*, in every period, you pay at a higher rate to consume power from the grid than you are paid when you send power back into the grid.

The data for the problem are

$$p^{\mathrm{ld}}, \quad p^{\mathrm{pv}}, \quad Q, \quad C, \quad D, \quad R^{\mathrm{buy}}, \quad R^{\mathrm{sell}}.$$

(a) Explain how to find the powers and battery state of charge that minimize the total cost of the grid power. Carry out your method using the data given in `microgrid_data.*`. Report the optimal cost of the grid power. Plot $p^{\mathrm{grid}}$, $p^{\mathrm{load}}$, $p^{\mathrm{pv}}$, $p^{\mathrm{batt}}$, and $q$ versus $i$. *Note.* For CVXPY, you might need to specify `solver=cvx.ECOS` when you call the `solve()` method.

(b) *Price and payments.* Let $\nu \in \mathbf{R}^{96}$ denote the optimal dual variable associated with the power balance constraint. The vector $4\nu$ can be interpreted as the (time-varying) price of electricity at the microgrid, and is called the *locational marginal price* (LMP). The LMP is in \$/kWh, and is generally positive; the factor 4 converts between 15 minute power intervals and per kWh prices. Find and plot the LMP, along with the grid buy and sell prices, versus $i$. Make a very brief comment comparing the LMP prices with the buy and sell grid prices. *Hint.* Depending on how you express the power balance constraint, your software might return $-\nu$ instead of $\nu$. Feel free to use $-4\nu$ instead of $\nu$, or to switch the left-hand and righ-hand sides of your power balance constraint.

(c) The LMPs can be used as a system for payments among the load, the PV array, the battery, and the grid. The load pays $\nu^T p^{\mathrm{ld}}$; the PV array is paid $\nu^T p^{\mathrm{pv}}$; the battery is paid $\nu^T p^{\mathrm{batt}}$; and the grid is paid $\nu^T p^{\mathrm{grid}}$. Note carefully the directions of these payments. Also note that the battery and grid, whose powers can have either sign, can be paid in some time intervals and pay in others.

Use this pricing scheme to calculate the LMP payments made by the load, and to the PV array, the battery, and the grid. If all goes well, these payments will balance, *i.e.*, the load will pay an amount equal to the sum of the others.

When you execute the script that contains the data, it will create plots showing the various powers and prices versus time. Your are welcome to use these as templates for plotting your results. You are very welcome to look inside the script to see how the data is generated.

*Remark.* (Not needed to solve the problem.) The given data is approximately consistent with a group of ten houses, a common or pooled PV array of around 100 panels, and two Tesla Powerwall batteries.

**Solution.**

(a) With the objective and constraints directly as they are written in the problem statement, we get the problem

$$\begin{array}{ll} \text{minimize} & (1/4)\left(R^{\mathrm{buy}}\right)^T \max\{p^{\mathrm{grid}},0\} - (1/4)\left(R^{\mathrm{sell}}\right)^T \max\{-p^{\mathrm{grid}},0\}. \\ \text{subject to} & p^{\mathrm{ld}} = p^{\mathrm{grid}} + p^{\mathrm{batt}} + p^{\mathrm{pv}} \\ & 0 \preceq q \preceq Q\mathbf{1} \\ & q_{i+1} = q_i - (1/4)p_i^{\mathrm{batt}} \quad i = 1,\ldots,95 \\ & q_1 = q_{96} - (1/4)p_{96}^{\mathrm{batt}} \\ & -C\mathbf{1} \preceq p^{\mathrm{batt}} \preceq D\mathbf{1}, \end{array}$$

with variables $p^{\mathrm{grid}}$, $p^{\mathrm{batt}}$, and $q$.

All of the constraints are convex. The objective is also convex, but it's not in DCP form, since the second term (including the minus sign) is concave. To see

that the total objective is convex, we note that it's a sum of functions of $p_i^{\text{grid}}$,

$$f_i(p_i^{\text{grid}}) = \begin{cases} (1/4)R^{\text{sell}}p_i^{\text{grid}} & p_i^{\text{grid}} \leq 0 \\ (1/4)R^{\text{buy}}p_i^{\text{grid}} & p_i^{\text{grid}} \geq 0. \end{cases}$$

This piecewise affine function 'kinks upward' at the kink point 0, so it's convex. (This depends on the inequality $R_i^{\text{buy}} \geq R_i^{\text{sell}}$.)

So the problem is convex as stated, but it's not in DCP form. There are two ways to get around this. The first is to express the functions $f_i$ as

$$f_i(u) = R_i^{\text{buy}}u_i + (R_i^{\text{buy}} - R_i^{\text{sell}})\max\{-u_i, 0\},$$

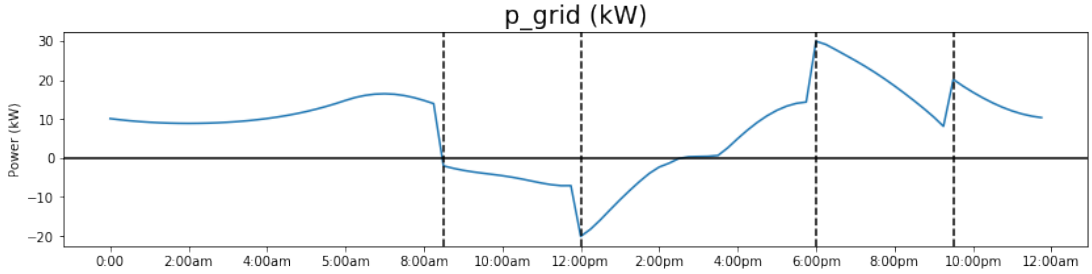
which is DCP. Note the interpretation: Power, bought or sold, is charged at the (cheaper) rate $R_i^{\text{buy}}$; but you also pay an additional amount if you are selling power back to the grid.

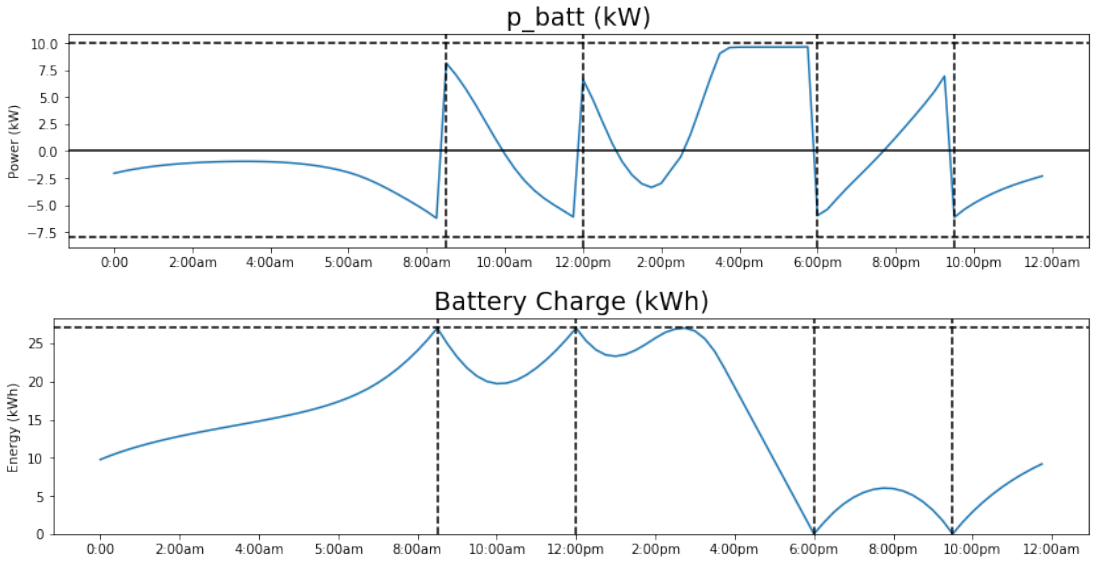
The second method is to break up the grid power into two components, which we call $p^{\text{buy}}$ and $p^{\text{sell}}$, and then constrain both variables to be nonnegative, and set $p^{\text{grid}} = p^{\text{buy}} - p^{\text{sell}}$. Our problem then becomes:

$$\begin{array}{ll} \text{minimize} & (1/4)\left(R^{\text{buy}}\right)^T p^{\text{buy}} - (1/4)\left(R^{\text{sell}}\right)^T p^{\text{sell}} \\ \text{subject to} & p^{\text{ld}} = p^{\text{grid}} + p^{\text{batt}} + p^{\text{pv}} \\ & 0 \preceq q \preceq Q\mathbf{1} \\ & q_{i+1} = q_i - (1/4)p_i^{\text{batt}} \quad i = 1, \ldots, 95 \\ & q_1 = q_{96} - (1/4)p_{96}^{\text{batt}} \\ & -C\mathbf{1} \preceq p^{\text{batt}} \preceq D\mathbf{1} \\ & p^{\text{grid}} = p^{\text{buy}} - p^{\text{sell}} \\ & p^{\text{buy}} \succeq 0 \\ & p^{\text{sell}} \succeq 0 \end{array}$$

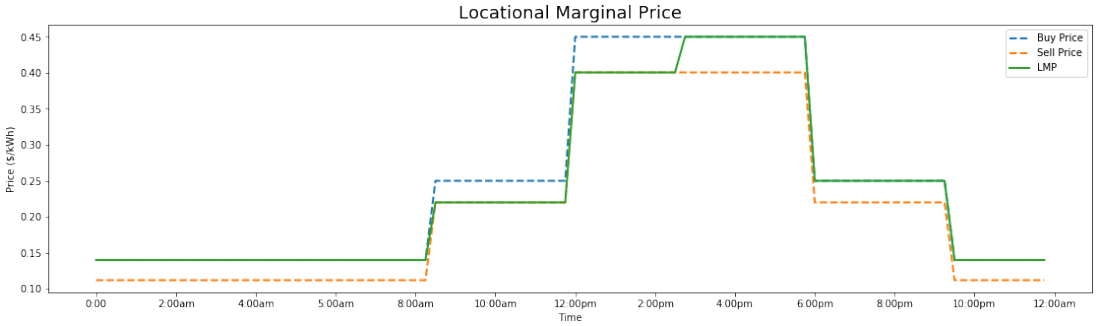
With variables $p^{\text{grid}}$, $p^{\text{buy}}$, $p^{\text{sell}}$, $p^{\text{batt}}$, and $q$.

This problem is convex, in fact an LP, and we can recover our desired variables, $p^{\text{grid}}$, $p^{\text{batt}}$, and $q$, directly from the result. Depending on which programming language and which solver used, the optimal value (minimum cost paid to the grid) ranged between \$33.10 and \$33.16, due to numerical roundoff errors and small differences in the way the problem is transformed to an LP by the different systems.






(b) To get the LMP, as the problem explains, we simply get the value of the dual variable associated with the power balance equality constraint, multiplied by 4 to convert the prices back into \$/kWh (rather than per 15 minute interval).




(c) Depending on what language and solver is used, the total costs here could vary by around \$0.05. One particular set of values we get is load cost = \$107.98, battery cost = \$-8.37, PV cost = \$-66.45, and the effective grid cost is \$33.16. If you take the effective grid cost minus the sum of the other three values, it equals 0, which is what we expect.

Note also that the LMP is fixed to the buy price of the current period when we're buying power and the sell price of the current period if we're selling power.

The following Python code solves the problem.

```
# First, run imports.
import cvxpy as cvx
import numpy as np
import matplotlib.pyplot as plt
from microgrid_data import *
```

```
##############################################
# part (a) - Solving the optimization problem
##############################################

p_batt = cvx.Variable(N) # battery charge and discharge (pos is charging, neg i
p_buy = cvx.Variable(N)  # non-negative vector of power bought from grid
p_sell = cvx.Variable(N) # non-negative vector of power sold to grid
p_grid = p_buy - p_sell  # net power from grid pos is buying, neg is selling)
q = cvx.Variable(N)      # Energy in battery (in KWh)

payments = (.25)*p_buy.T * R_buy
income = (.25)*p_sell.T * R_sell
obj = cvx.Minimize(payments - income) # minimize net cost

constraints = []

# Power balance constraint
constraints += [p_ld == p_grid + p_batt + p_pv]

# Constrain p_buy and p_sell to be non-negative
constraints += [p_buy >= 0,
                p_sell >= 0]

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

constraints += [q[0] == q[N-1] - (.25)*p_batt[N-1]]
for i in range(1, N):
    constraints += [q[i] == q[i-1] - (.25)*p_batt[i-1]]

prob = cvx.Problem(obj, constraints)
prob.solve(solver=cvx.ECOS)

print('Minimum Cost: ', obj.value)

##############################################
# Plotting the variables
```

```
##############################################

q = q.value
p_grid = p_grid.value
p_batt = p_batt.value

# Plot boundaries indicating the intervals at which prices change, and the limi
# for each value, to make graphs more interpretable

# Plotting p_grid:
plt.figure(figsize=fig_size)
plt.plot(p_grid)
plt.ylabel('Power (kW)')
plt.title('p_grid (kW)', fontsize=19)
plt.xticks(xtick_vals, xtick_labels )
plt.axvline(partial_peak_start, linestyle='--', color='black')
plt.axvline(peak_start, linestyle='--', color='black')
plt.axvline(peak_end, linestyle='--', color='black')
plt.axvline(partial_peak_end, linestyle='--', color='black')
plt.axhline(0, color='black')
plt.show()

# Plotting p_batt:
plt.figure(figsize=fig_size)
plt.plot(p_batt)
plt.ylabel('Power (kW)')
plt.title('p_batt (kW)', fontsize=19)
plt.xticks(xtick_vals, xtick_labels )
plt.axvline(partial_peak_start, linestyle='--', color='black')
plt.axvline(peak_start, linestyle='--', color='black')
plt.axvline(peak_end, linestyle='--', color='black')
plt.axvline(partial_peak_end, linestyle='--', color='black')
plt.axhline(D, linestyle='--', color='black')
plt.axhline(-C, linestyle='--', color='black')
plt.axhline(0, color='black')
plt.show()

# Plotting q:
plt.figure(figsize=fig_size)
plt.plot(q)
plt.ylabel('Energy (kWh)')
plt.title('Battery Charge (kWh)', fontsize=19)
```

```
plt.xticks(xtick_vals, xtick_labels )
plt.axvline(partial_peak_start, linestyle='--', color='black')
plt.axvline(peak_start, linestyle='--', color='black')
plt.axvline(peak_end, linestyle='--', color='black')
plt.axvline(partial_peak_end, linestyle='--', color='black')
plt.axhline(Q, linestyle='--', color='black')
plt.ylim(bottom=0)
plt.show()

##############################################
# part (b) - Plotting LMP
##############################################
# Get the dual variable values
dual_vals = -constraints[0].dual_value
# We multiply by negative 1 because the sign was flipped
# If we wrote the equality constraint in a different way,
# we wouldn't have had to use the negative sign (i.e. the
# sign flip was an artifact of how we wrote the constraint)
LMP = 4*dual_vals

# Plot the price over time
plt.figure(figsize=(19,5))
plt.plot(R_buy, '--', label='Buy Price', linewidth=2)
plt.plot(R_sell, '--', label='Sell Price', linewidth=2)
plt.plot(LMP, linewidth=2, label='LMP')
plt.xlabel('Time')
plt.ylabel('Price ($/kWh)')
plt.title('Locational Marginal Price', fontsize=18)
plt.legend()
plt.xticks(xtick_vals, xtick_labels )
plt.show()


##############################################
# part (c) - Calculating Payments
##############################################
# Confirm that the prices balance when using the LMP
# Expecting load to have paid, battery and PV to have been paid
load_cost = p_ld @ dual_vals
batt_cost = -p_batt @ dual_vals
PV_cost = -p_pv @ dual_vals
grid_costs = p_grid @ dual_vals
```

```
print('Load cost: %.2f' % (load_cost))
print('Battery cost: %.2f' % (batt_cost))
print('PV cost: %.2f' % (PV_cost))
print('Effective grid cost: %.2f' % (grid_costs))
print()
net_cost = -load_cost - batt_cost - PV_cost + grid_costs
print('Grid costs minus other three: %.2f' % (net_cost))
```

The following Julia code solves the problem.

```
# First, run the data script
include("./microgrid_data.jl");

using Convex
using PyPlot

################################################
# part (a) - Solving the optimization problem
################################################

p_batt = Variable(N) # battery charge and discharge (pos is charging, neg is di
p_buy = Variable(N)  # non-negative vector of power bought from grid
p_sell = Variable(N) # non-negative vector of power sold to grid
p_grid = p_buy - p_sell  # net power from grid pos is buying, neg is selling)
q = Variable(N)      # Energy in battery (in KWh)

payments = (.25)*p_buy' * R_buy
income = (.25)*p_sell' * R_sell
prob = minimize(payments - income) # minimize net cost


# Power balance constraint
prob.constraints += [p_ld == p_grid + p_batt + p_pv]

# Constrain p_buy and p_sell to be non-negative
prob.constraints += [p_buy >= 0,
                p_sell >= 0]

# Battery Constraints
prob.constraints += [q <= Q,
                q >= 0,
                p_batt <= D,
                p_batt >= -C]
```

```
prob.constraints += [q[1] == q[N] - (.25)*p_batt[N]]
for i = 2:N
    prob.constraints += [q[i] == q[i-1] - (.25)*p_batt[i-1]]
end

solve!(prob, SCSSolver(verbose=0))

println("Minimum Cost: ", prob.optval)

q = q.value;
p_grid = p_buy.value - p_sell.value;
p_batt = p_batt.value;

################################################
# Plotting the variables
################################################
# Plot boundaries indicating the intervals at which prices change, and the limi
# for each value, to make graphs more interpretable

# Plotting p_grid:
figure(figsize=fig_size)
plot(p_grid)
ylabel("Power (kW)")
title("p_grid (kW)", fontsize=19)
xticks(xtick_vals, xtick_labels )
axvline(partial_peak_start, linestyle="--", color="black")
axvline(peak_start, linestyle="--", color="black")
axvline(peak_end, linestyle="--", color="black")
axvline(partial_peak_end, linestyle="--", color="black")
axhline(0, color="black")

# Plotting p_batt:
figure(figsize=fig_size)
plot(p_batt)
ylabel("Power (kW)")
title("p_batt (kW)", fontsize=19)
xticks(xtick_vals, xtick_labels )
axvline(partial_peak_start, linestyle="--", color="black")
axvline(peak_start, linestyle="--", color="black")
axvline(peak_end, linestyle="--", color="black")
axvline(partial_peak_end, linestyle="--", color="black")
```

```
axhline(D, linestyle="--", color="black")
axhline(-C, linestyle="--", color="black")
axhline(0, color="black")


# Plotting q:
figure(figsize=fig_size)
plot(q)
ylabel("Energy (kWh)")
title("Battery Charge (kWh)", fontsize=19)
xticks(xtick_vals, xtick_labels )
axvline(partial_peak_start, linestyle="--", color="black")
axvline(peak_start, linestyle="--", color="black")
axvline(peak_end, linestyle="--", color="black")
axvline(partial_peak_end, linestyle="--", color="black")
axhline(Q, linestyle="--", color="black")
ylim(bottom=0);


##############################################
# part (b) - Plotting LMP
##############################################
# Get the dual variable values
dual_vals = -prob.constraints[1].dual
# We multiply by negative 1 because the sign was flipped
# If we wrote the equality constraint in a different way,
# we wouldn't have had to use the negative sign (i.e. the
# sign flip was an artifact of how we wrote the constraint)
LMP = 4*dual_vals

# Plot the price over time
figure(figsize=(19,5))
plot(R_buy, "--", label="Buy Price", linewidth=2)
plot(R_sell, "--", label="Sell Price", linewidth=2)
plot(LMP, linewidth=2, label="LMP")
xlabel("Time")
ylabel("Price (\$/kWh)")
title("Locational Marginal Price", fontsize=18)
legend()
xticks(xtick_vals, xtick_labels );
```

```
##############################################
# part (c) - Calculating Payments
##############################################
# Confirm that the prices balance when using the LMP
# Expecting load to have paid, battery and PV to have been paid
load_cost = p_ld' * dual_vals
batt_cost = -p_batt' * dual_vals
PV_cost = -p_pv' * dual_vals
grid_costs = p_grid' * dual_vals

println("Load cost: " , load_cost)
println("Battery cost: " , batt_cost)
println("PV cost: " , PV_cost)
println("Effective grid cost: " , grid_costs)
net_cost = -load_cost - batt_cost - PV_cost + grid_costs
println("Grid costs minus other three: ", net_cost)
```

The following Matlab code solves the problem.

```
clc; clear; close all;
% Get the data
run('micro_grid_data.m')

%% Part (a) - Solving Optimization
cvx_begin quiet
    variables p_batt(N) p_buy(N) p_sell(N) p_grid(N) q(N)
    variables payments income
    dual variable v
    minimize(payments - income)
    subject to
        % Grid constraints
        payments == (.25)*p_buy' * R_buy
        income == (.25)*p_sell' * R_sell
        p_grid == p_buy - p_sell
        v : p_ld == p_grid + p_batt + p_pv % associate the dual variable
        p_buy >= 0
        p_sell >= 0
        % Battery constraints
        q >= 0
        q <= Q
        p_batt <= D
        p_batt >= -C
        q(1) == q(N) - (.25)*p_batt(N)
```

```
        for i=2:N
            q(i) == q(i-1) - (.25)*p_batt(i-1)
        end
cvx_end
fprintf('Optimal Value: $%.2f\n', cvx_optval);

%% Plot things
pp_start = 34;
p_start = 48;
p_end = 72;
pp_end = 86;

% Plot vertical lines where prices change and horizontal lines
% where the limits of each value are, for interpretability

figure()
plot(p_grid)
title('Grid Power (kW)');
title('Power (kW)');
xlabel('Interval');
line([pp_start, pp_start], [-35, 35], 'Color', 'black', 'LineStyle', '--');
line([p_start, p_start], [-35, 35], 'Color', 'black', 'LineStyle', '--');
line([p_end, p_end], [-35, 35], 'Color', 'black', 'LineStyle', '--');
line([pp_end, pp_end], [-35, 35], 'Color', 'black', 'LineStyle', '--');
line([1,N], [0, 0], 'color', 'black');
ylim([-35, 35])

figure()
plot(p_batt)
title('Battery Power (kW)');
ylabel('Power (kW)');
xlabel('Interval');
line([pp_start, pp_start], [-C, D], 'Color', 'black', 'LineStyle', '--');
line([p_start, p_start], [-C, D], 'Color', 'black', 'LineStyle', '--');
line([p_end, p_end], [-C, D], 'Color', 'black', 'LineStyle', '--');
line([pp_end, pp_end], [-C, D], 'Color', 'black', 'LineStyle', '--');
line([1,N], [D, D], 'color', 'black', 'linestyle', '--');
line([1,N], [-C, -C], 'color', 'black', 'linestyle', '--');
line([1,N], [0, 0], 'color', 'black');
ylim([-C-2, D+2])

figure()
```

```
plot(q)
title('Battery Charge (kWh)');
title('Energy (kWh)');
xlabel('Interval');
line([pp_start, pp_start], [0, Q], 'Color', 'black', 'LineStyle', '--');
line([p_start, p_start], [0, Q], 'Color', 'black', 'LineStyle', '--');
line([p_end, p_end], [0, Q], 'Color', 'black', 'LineStyle', '--');
line([pp_end, pp_end], [0, Q], 'Color', 'black', 'LineStyle', '--');
line([1,N], [Q, Q], 'color', 'black', 'linestyle', '--');
line([1,N], [0, 0], 'color', 'black');
ylim([0, Q+2])

%% Part (b) - Getting LMP
if v(1) < 0
    v = -v;
end
LMP = 4*v;
% We multiply by negative 1 because the sign was flipped
% If we wrote the equality constraint in a different way,
% we wouldn't have had to use the negative sign (i.e. the
% sign flip was an artifact of how we wrote the constraint)

figure()
hold('on')
plot(R_buy, 'linestyle', '--')
plot(R_sell, 'linestyle', '--')
plot(LMP)
legend('Buy Price', 'Sell Price', 'LMP');
title('LMP vs Nominal Prices');
xlabel('Interval');
ylabel('Price ($/kWh)');

%% Part (c) - Calculating Payments
load_cost = p_ld' * v
batt_cost = -p_batt' * v
PV_cost = -p_pv' * v
grid_costs = p_grid' * v

net_cost = -load_cost - batt_cost - PV_cost + grid_costs
```

4. *Curvature of some order statistics.* For $x \in \mathbf{R}^n$, with $n > 1$, $x_{[k]}$ denotes the $k$th largest entry of $x$, for $k = 1, \ldots, n$, so, for example, $x_{[1]} = \max_{i=1,\ldots,n} x_i$ and $x_{[n]} = \min_{i=1,\ldots,n} x_i$. Functions that depend on these sorted values are called order statistics or order functions. Determine the curvature of the order statistics below, from the choices convex, concave, or neither. For each function, explain why the function has the curvature you claim. If you say it is neither convex nor concave, give a counterexample showing it is not convex, and a counterexample showing it is not concave. All functions below have domain $\mathbf{R}^n$.

   (a) $\text{median}(x) = x_{[(n+1)/2]}$. (You can assume that $n$ is odd.)

   (b) The range of values, $x_{[1]} - x_{[n]}$.

   (c) The midpoint of the range, $(x_{[1]} + x_{[n]})/2$.

   (d) Interquartile range, defined as $x_{[n/4]} - x_{[3n/4]}$. (You can assume that $n/4$ is an integer.)

   (e) Symmetric trimmed mean, defined as

$$\frac{x_{[n/10]} + x_{[n/10+1]} + \cdots + x_{[9n/10]}}{0.8n + 1},$$

   the mean of the values between the 10th and 90th percentiles. (You can assume that $n/10$ is an integer.)

   (f) Lower trimmed mean, defined as

$$\frac{x_{[1]} + x_{[2]} + \cdots + x_{[9n/10]}}{0.9n + 1},$$

   the mean of the entries, excluding the bottom decile. (You can assume that $n/10$ is an integer.)

*Remark.* For the functions defined in (d)–(f), you might find slightly different definitions in the literature. Please use the formulas above to answer each question.

**Solution.**

(a) *Neither convex nor concave.* The medians of $(0, 2, 0)$ and $(2, 0, 0)$ are both 0, but the median of their average $(1, 1, 0)$ is 1, which violates convexity. The medians of $(0, -2, 0)$ and $(-2, 0, 0)$ are both 0, but the median of their average, $(-1, -1, 0)$ is $-1$, which violates concavity.

(b) *Convex.* The function $x_{[n]}$ is concave for $x \in \mathbf{R}^n$, so the given function is the sum of two convex functions.

(c) *Neither.* The midpoints of the ranges of $(0, 2, 2)$ and $(2, 2, 0)$ are 1, but the midpoint of the range of their average, $(1, 2, 1)$, is $3/2$, which violates convexity. On the other hand, the midpoints of the ranges of $(2, 0, 0)$ and $(0, 2, 0)$ are 1, but the midpoint of the range of their average, $(1, 1, 0)$, is $1/2$, which violates concavity.

(d) *Neither.* Suppose that $n = 7$, so the interquartile range of $z \in \mathbf{R}^7$ is $z_{[2]} - z_{[6]}$. If $x = 2e_1 - 2e_7$ and $y = 2e_2 - 2e_6$, then the interquartile ranges of both $x$ and $y$ are 0, but their average, $(x+y)/2 = (e_1 + e_2) - (e_6 + e_7)$ has an interquartile range of 2, which violates convexity. Similarly, if $u = 2(e_1 + e_2) - 2(e_6 + e_7)$ and $v = -u$, then the interquartile ranges of $u$ and $v$ are 2, but their average, $(u + v)/2 = 0$ has an interquartile range of 0, which violates concavity.

(e) *Neither.* (This counterexample assumes a shift by one in the indexing, correct if necessary.) For $n = 20$, if $x = e_1 + e_2$ and $y = e_3 + e_4$, their symmetric trimmed mean is exactly zero. However, the symmetric mean of their average, $z = (e_1+e_2+e_3+e_4)/2$, will be $1/16$, which violates convexity. On the other hand, if $x = \mathbf{1}-e_1-e_2$ and $y = \mathbf{1}-e_3-e_4$, their symmetric trimmed mean will be exactly one. However, the symmetric mean of their average, $z = \mathbf{1} - (e_1 + e_2 + e_3 + e_4)/2$, will be $15/16$, which violates concavity.

(f) *Convex.* It's $10/9n$ times the sum of the $9n/10$ largest elements of $x$, which is a convex function for $x \in \mathbf{R}^n$.

5. *Control with various objectives.* We consider a standard optimal control problem, with dynamics $x_{t+1} = Ax_t + Bu_t$, $t = 0, 1, \ldots, T-1$. Here $x_t \in \mathbf{R}^n$ is the state, and $u_t \in \mathbf{R}^m$ is the control or input, at time period $t$, $A \in \mathbf{R}^{n\times n}$ is the dynamics matrix, and $B \in \mathbf{R}^{n\times m}$ is the input matrix. We are given the initial state, $x_0 = x^{\text{init}}$, and we require that the final state be zero, $x_T = 0$. (In applications, the state 0 corresponds to some desirable state.) Your job is to choose the sequence of inputs $u_0, \ldots, u_{T-1}$ that minimize an objective. Values for $x^{\text{init}}$, $A$, $B$, and $T$ are given in `various_obj_regulator_data.*`.

   We consider various objectives, all of which measure the size of the inputs (or, in control dialect, the *control effort*).

   (a) *Sum of squares of 2-norms.* $\sum_{t=0}^{T-1} \|u_t\|_2^2$. This is the traditional objective.

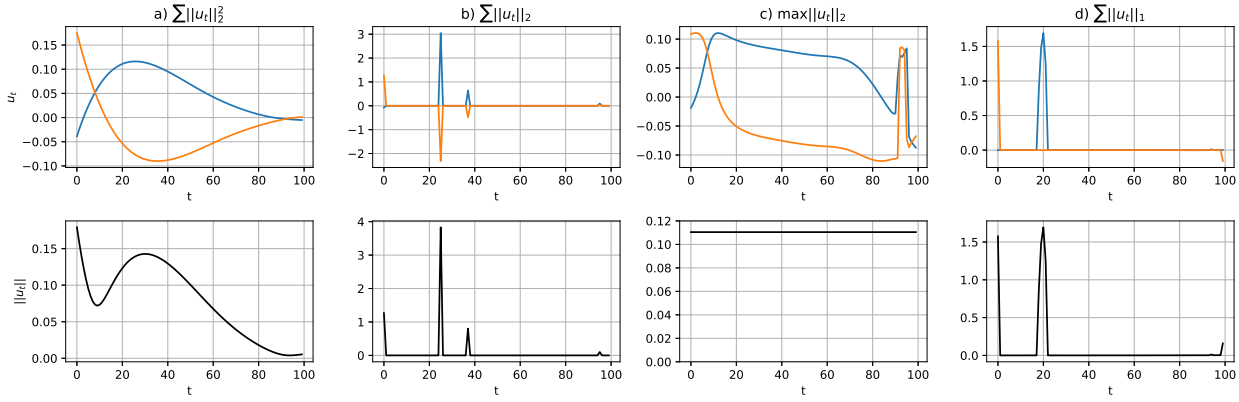
   (b) *Sum of 2-norms.* $\sum_{t=0}^{T-1} \|u_t\|_2$.

   (c) *Max of 2-norms.* $\max_{t=0,\ldots,T-1} \|u_t\|_2$.

   (d) *Sum of 1-norms.* $\sum_{t=0}^{T-1} \|u_t\|_1$. In some applications this is an approximation of the fuel use.

   For each objective, plot (the components of) optimal input, as well as $\|u_t\|_2$, versus $t$. Make a very brief comment on each plot of optimal control inputs, explaining why you might expect what happened.

   **Solution.** The following plot shows controller inputs from python




   (a) The control inputs are small, but not sparse. This is what we expect with least squares.

   (b) The control input is sparse; and when the control is nonzero, both components are nonzero.

   (c) The $\ell_2$ norm of the control input is constant; the direction of the control input changes over time.

   (d) The control input is sparse; the different components are nonzero in different times.

The following Python code solves the problem.

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

from various_obj_regulator_data import *

u_ = cp.Variable((m,T))
x_ = cp.Variable((n,T+1))
objs = [
(cp.Minimize(cp.sum_squares(u_))            , "a) $\\sum||u_t||_2^2$"),
(cp.Minimize(cp.sum(cp.norm(u_,2,axis=0)))  , "b) $\\sum||u_t||_2$"),
(cp.Minimize(cp.max(cp.norm(u_,axis=0)))    , "c) $\\max||u_t||_2$"),
(cp.Minimize(cp.sum(cp.norm(u_,1,axis=0)))  , "d) $\\sum||u_t||_1$")
]
plt.figure(figsize=(15,5))
for i,obj in enumerate(objs):
    const = [x_[:,-1] == np.zeros(n)]
    const.append(x_[:,0] == x_init)
    for t in range(1,T+1):
        const.append(x_[:,t] == A@x_[:,t-1] + B@u_[:,t-1])
    prob = cp.Problem(obj[0],const)
    prob.solve()
    plt.subplot(2,4,i+1)
    plt.plot(u_.value.T)
    if i == 0:
        plt.ylabel("$u_t$")
    plt.title(obj[1])
    plt.grid()
    plt.xlabel("t")
    plt.subplot(2,4,i+5)
    plt.xlabel("t")
    plt.plot(np.linalg.norm(u_.value,axis=0),c="black",label="$||u||_2$")
    if i == 2:
        plt.ylim(ymax = .12,ymin=0)
    if i == 0:
        plt.ylabel("$||u_t||$")
    plt.grid()
plt.tight_layout()
plt.savefig('../figures/various_obj_regulator.eps',bbox_inches='tight')
#plt.show()
```

The following Matlab code solves the problem.

```
n = 4;
m = 2;

A = [
 0.95,  0.16,  0.12,  0.01;
-0.12,  0.98, -0.11, -0.03;
-0.16,  0.02,  0.98,  0.03;
-0.  ,  0.02, -0.04,  1.03;
];

B = [
 0.8 , 0. ;
 0.1 , 0.2;
 0.  , 0.8;
-0.2 , 0.1;
];

x_init = ones(n,1);

T = 100;

cvx_begin
    variable u(m,T)
    variable x(n,T+1)
    minimize(sum(sum(u.^2)));
    subject to
        x(:,end) == zeros(n,1)
        x(:,1) == x_init
    for t = 2:(T+1)
        subject to
            x(:,t) == A*x(:,t-1) + B*u(:,t-1)
    end
cvx_end

subplot(2,4,1)
plot(u')
ylabel("u_t")
subplot(2,4,4+1)
plot(sqrt(sum(u.^2,1)))
xlabel("t")
ylabel("||u_t||")


cvx_begin
    variable u(m,T)
```

```
    variable x(n,T+1)
    minimize(sum(norms(u,2,1)));
    subject to
        x(:,end) == zeros(n,1)
        x(:,1) == x_init
    for t = 2:(T+1)
        subject to
            x(:,t) == A*x(:,t-1) + B*u(:,t-1)
    end
cvx_end

subplot(2,4,2)
plot(u')
ylabel("u_t")
subplot(2,4,4+2)
plot(sqrt(sum(u.^2,1)))
xlabel("t")
ylabel("||u_t||")


cvx_begin
    variable u(m,T)
    variable x(n,T+1)
    minimize(max(norms(u,2,1)));
    subject to
        x(:,end) == zeros(n,1)
        x(:,1) == x_init
    for t = 2:(T+1)
        subject to
            x(:,t) == A*x(:,t-1) + B*u(:,t-1)
    end
cvx_end

subplot(2,4,3)
plot(u')
ylabel("u_t")
subplot(2,4,4+3)
plot(sqrt(sum(u.^2,1)))
xlabel("t")
ylabel("||u_t||")

cvx_begin
    variable u(m,T)
    variable x(n,T+1)
    minimize(sum(norms(u,1,1)));
```

```
    subject to
        x(:,end) == zeros(n,1)
        x(:,1) == x_init
    for t = 2:(T+1)
        subject to
            x(:,t) == A*x(:,t-1) + B*u(:,t-1)
    end
cvx_end

subplot(2,4,4)
plot(u')
ylabel("u_t")
subplot(2,4,4+4)
plot(sqrt(sum(u.^2,1)))
xlabel("t")
ylabel("||u_t||")
```

The following Julia code solves the problem.

6. *Morphing between two discrete distributions.* Consider two distributions for a random variable that takes values in $\{1, 2, \ldots, n\}$, given by $q, r \in \mathbf{R}^n$, with $q \succeq 0$, $\mathbf{1}^T q = 1$, and $r \succeq 0$, $\mathbf{1}^T r = 1$. We seek a sequence of distributions $p^{(i)}$, $i = 1, \ldots, N$, that 'morph' between $q$ and $r$. This means that $p^{(1)} = q$, $p^{(N)} = r$, and $p^{(i+1)}$ is close to $p^{(i)}$ for $i = 1, \ldots, (N-1)$, in some sense. Specifically we will minimize

$$\sum_{i=1}^{N-1} d(p^{(i)}, p^{(i+1)})$$

where $d$ is a distance function.

(a) *Euclidean morphing.* What is the solution when the distance function is the sum of squares, $d^{\text{sq}}(u, v) = \|u - v\|_2^2$? The solution is simple; you can just give it without justification.

(b) *Hellinger morphing.* Now we use the Hellinger distance function

$$d^{\text{hel}}(u, v) = \sum_{i=1}^{n} (\sqrt{u_i} - \sqrt{v_i})^2.$$

Explain how to solve the Hellinger morphing problem using convex optimization.

(c) *Kolmogorov morphing.* Now we use the Kolmogorov distance function

$$d^{\text{kol}}(u, v) = \max_{i=1,\ldots,n} \left| \sum_{j=1}^{i} u_j - \sum_{j=1}^{i} v_j \right|,$$

which is the $\ell_\infty$ distance between the respective cumulative distributions (using the order of the outcomes). Explain how to solve the Kolmogorov morphing problem using convex optimization.

(d) Find the Euclidean, Hellinger, and Kolmogorov morphings for $N = 10$, $n = 100$. Use $q$ and $r$ provided in `morphing_data.*`. Plot each $p^{(i)}$ versus $n$. Produce one figure for each choice of distance function.

*Note.* In Python and Julia, you should use the ECOS solver.

**Solution.**

(a) As you might suspect, the optimal distributions satisfy

$$p^{(i)} = \left(1 - \frac{i-1}{N-1}\right) q + \left(\frac{i-1}{N-1}\right) r, \quad i = 1, \ldots, N.$$

(In other words, $p^{(i)}$ are $N$ evenly spaced points along the line segment between $q$ and $r$.) You can verify this by differentiating the objective (ignoring the constriants), which yields

$$2p^{(i)} = p^{(i-1)} + p^{(i+1)}, \qquad i = 2, \ldots, N-1.$$

Solving this recurrence yields the solution above (and the unconstrained minimum also satisfies the constraints).

(b) The problem as given is convex, since $d^{\text{hel}}(u, v)$ is (jointly) convex. This is because

$$d^{\text{hel}}(u, v) = (u + v)^T \mathbf{1} - 2 \sum_{i=1}^{n} \sqrt{u_i v_i},$$

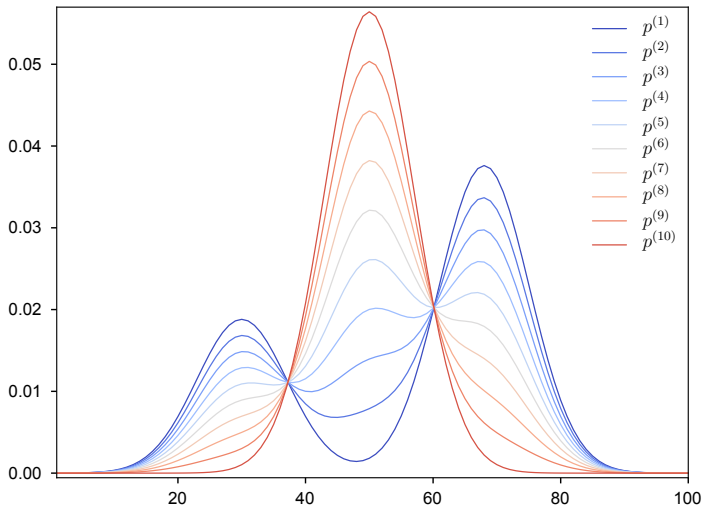
which is convex since the geometric mean is concave.

(c) The problem as given is convex, since $d^{\text{kol}}(u, v)$ is (jointly) convex. This is because for $i = 1, \ldots, n$, the function $\left| \sum_{j=1}^{i} u_j - \sum_{j=1}^{i} v_j \right|$ is the composition of a convex function (absolute value) with a linear function of $u$ and $v$. Since $d^{\text{kol}}$ is the maximum of these $n$ functions, it is convex.

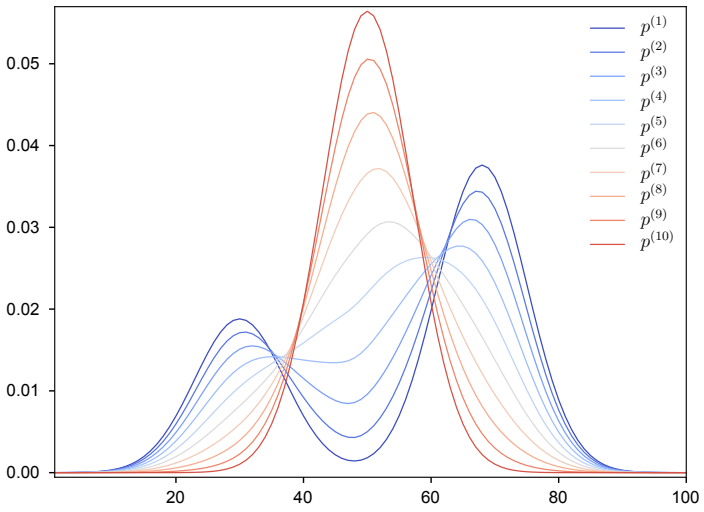
(d) For each choice of $d$ above, we simply solve

$$\begin{array}{lll} \text{minimize} & \sum_{i=1}^{N-1} d(p^{(i)}, p^{(i+1)}) & \\ \text{subject to} & p^{(i)} \succeq 0, \quad \mathbf{1}^T p^{(i)} = 1, & i = 1, 2, \ldots N, \\ & p^{(1)} = q, \quad p^{(N)} = r. & \end{array}$$

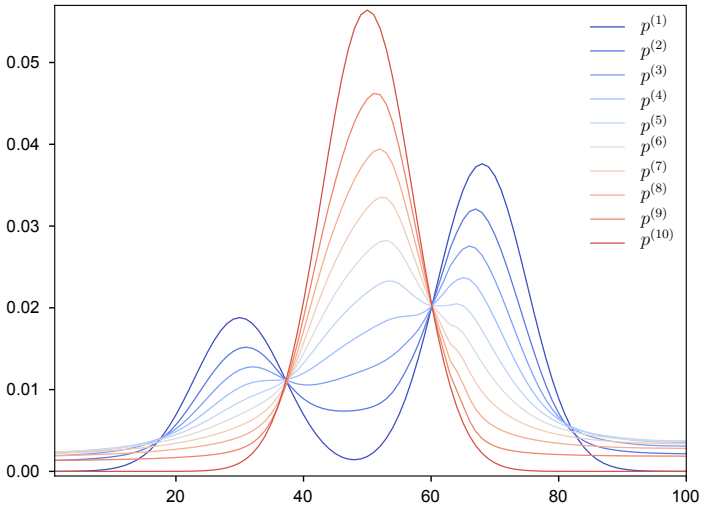
Here's the sequence of distributions obtained by using the Euclidean distance.




Here's the sequence of distributions obtained by using the Hellinger distance.




Here's the sequence of distributions obtained by using the Kolmogorov distance.




The following Julia code solves the problem.

```
using Convex, ECOS, PSPlot
include("morphing_data.jl")

# plotting colors
cm = ColorMap("coolwarm"); colors = [cm(i) for i in linspace(0, 0.9, N)];

# distance functions
euc(a, b) = sumsquares(a - b)
hel(a, b) = -sum(geomean(a[i], b[i]) for i in 1:n)
kol(a, b) = maximum(abs(sum(a[1:i] - b[1:i])) for i in 1:n)
functions = [euc, hel, kol]

for d in functions
    p = Variable(n, N, Positive())
```

```
    obj = sum(d(p[:, i], p[:, i+1]) for i in 1:(N-1))
    prob = minimize(obj, [p[:, 1] == q, p[:, N] == r, p' * ones(n) == ones(N)])
    solve!(prob, ECOSSolver(verbose=false), verbose=false)
    p = p.value
    # plot
    figure()
    for i in 1:N
        plot(1:n, p[:, i], label="       ", color=colors[i])
    end
    legend()
    printfig("../figures/morphing_$(d)")
end
```

The following Python code solves the problem.

```
from morphing_data import *
import cvxpy as cp

import matplotlib.pyplot as plt
from matplotlib import cm

coolwarm = cm.get_cmap('coolwarm')
colors = [coolwarm(i) for i in np.linspace(0, 0.9, N)]

# distance functions
functions = [
lambda a,b : cp.sum_squares(a - b),
lambda a,b : -cp.sum([cp.geo_mean(cp.vstack([a[i], b[i]])) for i in range(n)]),
lambda a,b : cp.max(cp.vstack([cp.abs(cp.sum(a[:i] - b[:i])) for i in range(1,n)])),
]

for d in functions:
    p = cp.Variable((n, N), nonneg=True)
    obj = cp.Minimize(cp.sum([d(p[:, i], p[:, i+1]) for i in range(N-1)]))
    prob = cp.Problem(obj, [p[:, 0] == q, p[:, -1] == r, p.T@np.ones(n) == np.ones(N)])
    prob.solve(solver="ECOS")
    p = p.value
    for i in range(N):
        plt.plot(p[:, i], label="$p^{(%d)}$"%(i+1), color=colors[i])
    plt.legend()
    plt.show()
```

The following MATLAB code solves the problem.

```
N = 10;
n = 100;
clf
colormap cool;
q = [exp(-((1:50) -25).^2 ./ 2) exp(-((51:100) -75).^2 ./ 2)];
r = exp(-((1:100) - 50).^2 ./ 2);
q = q/sum(q);
r = r/sum(r);
fns = {@euc,@hel,@kol}
```

```
for i = 1:3
    cvx_begin
        variable p(n,N) nonnegative
        expression x(N-1);
        for k=1:N-1
            x(k) = fns{i}(p(:,k), p(:,k+1));
        end
        minimize(sum(x))
        subject to
            p(:,1) == q';
            p(:,end) == r';
            p'*ones(n,1) == ones(N,1);
    cvx_end

    plot(p)
end
function r=euc(a,b)
    r=sum(sum((a-b).^2));
end
function r=hel(a,b)
    r = 0;
    for j = 1:100
        r = r-geo_mean([a(j),b(j)]);
    end
end
function r=kol(a,b)
    m = [];
    for k = 1:100
       m = [m, abs(sum(a(1:k) - b(1:k)))];
    end
    r = max(m);
end
```

7. *Constrained maximum likelihood estimation of mean and covariance.* You are given some independent samples $x_1, \ldots, x_N \in \mathbf{R}^n$ from a Gaussian distribution $\mathcal{N}(\mu, \Sigma)$. Explain how to find the maximum-likelihood estimate of $\mu$ and $\Sigma$, subject to the constraint that $\Sigma^{-1}\mu \succeq 0$, using convex optimization. You must fully justify any change of variables.

   *Finance interpretation.* (Not needed to solve the problem.) Suppose $x \sim \mathcal{N}(\mu, \Sigma)$ is the return of $n$ assets. The portfolio vector $h$ that maximizes the risk-adjusted return $\mu^T h - \gamma h^T \Sigma h$, where $\gamma > 0$ is the risk aversion parameter, is $h = (1/2\gamma)\Sigma^{-1}\mu$. So the constraint in the problem above is that the optimal portfolio has nonnegative entries, *i.e.*, is a long-only portfolio. The constrained maximum-likelihood estimate finds the maximum likelihood mean and covariance of the return distribution, subject to the constraint that the associated optimal portfolio is long-only.

   *Probability interpretation.* (Not needed to solve the problem.) The constraint $\Sigma^{-1}\mu \succeq 0$ is the same as $\nabla p(0) \succeq 0$, where $p$ is the density of the $\mathcal{N}(\mu, \Sigma)$ distribution. In other words, at 0, the density is nondecreasing in each coordinate.

   **Solution.** The negative log-likelihood is

$$\frac{1}{2}\sum_{i=1}^{N}\left(n\log(2\pi) + \log\det\Sigma + (x_i-\mu)^T\Sigma^{-1}(x_i-\mu)\right),$$

   which is not (in general) convex in the variable $\Sigma$ and $\mu$, and nor is the constraint $\Sigma^{-1}\mu \succeq 0$.

   However, we use the standard change of variables, to the so-called natural parameters for a Gaussian distribution, $\theta = \Sigma^{-1}$ and $\omega = \Sigma^{-1}\mu$. We can recover the original parameters from these using $\Sigma = \theta^{-1}$ and $\mu = \theta^{-1}\omega$. Using these parameters, the negative log-likelihood is (dropping a constant that doesn't matter)

$$\begin{aligned}\ell(\theta,\omega) &= -\frac{N}{2}\log\det\theta + \frac{1}{2}\sum_{i=1}^{N}(x_i-\theta^{-1}\omega)^T\theta(x_i-\theta^{-1}\omega)\\ &= -\frac{N}{2}\log\det\theta + \frac{1}{2}\sum_{i=1}^{N}\left(x_i^T\theta x_i - 2x_i^T\omega + \omega^T\theta^{-1}\omega\right),\end{aligned}$$

   which is convex. Our constraint is very simple, and convex in the new variables: $\omega \succeq 0$. So the constrained maximum likelihood problem is convex in the new variables $\theta, \omega$.

8. *Minimizing tax liability.* You will liquidate (sell) some stocks that you hold to raise a given amount of cash $C$. The stocks are divided into $n$ tax lots; a tax lot is a group of stocks you bought at the same time. For each tax lot $i$, you have the cost basis $b_i > 0$, the current market value $v_i > 0$ (both in \$), and its short term / long term status. (Long term means that you acquired the stock in the tax lot more than one year ago, and short term means that you acquired it less than one year ago.) We assume that tax lots $i = 1, \ldots, L$ are long term, and tax lots $i = L+1, \ldots, n$ are short term.

   The goal is to choose how much of each lot to sell. We let $s_i$ denote the amount of tax lot $i$ we sell (in \$). These must satisfy $0 \leq s_i \leq v_i$, and we must have $\mathbf{1}^T s = C$.

   When $v_i < b_i$, the sale is called a loss, and when $v_i > b_i$, the sale is called a gain. The amount of the gain or loss is given by $g_i = (s_i/v_i)(v_i - b_i)$, with positive values meaning a gain, and negative values meaning a loss. We define the (net) long and short term gains as

$$N^{\mathrm{l}} = \sum_{i=1}^{L} g_i, \qquad N^{\mathrm{s}} = \sum_{i=L+1}^{n} g_i.$$

   When $N^{\mathrm{l}} > 0$ ($N^{\mathrm{l}} < 0$), we say that we have had a long term capital gain (loss), and similary for short term gain.

   These two net gains determine the total tax liability. The long and short term net gains are taxed at two different rates, $\rho^{\mathrm{l}}$ and $\rho^{\mathrm{s}}$, respectively, which satisfy $0 < \rho^{\mathrm{l}} < \rho^{\mathrm{s}}$.

   The simplest case is when both net gains are nonnegative, in which case the tax is $\rho^{\mathrm{l}} N^{\mathrm{l}} + \rho^{\mathrm{s}} N^{\mathrm{s}}$. Another simple case occurs when both net gains are nonpositive, in which case the tax is zero.

   In the case when one of the net gains is positive and the other is negative, you are allowed to use the net loss in one to offset the net gain in the other, up to the value of the net gain. Specifically, if $N^{\mathrm{l}} < 0$ (you have a long term loss), the tax is $\rho^{\mathrm{s}}(N^{\mathrm{s}} + N^{\mathrm{l}})_+$; if $N^{\mathrm{s}} < 0$ (you have a short term loss), the tax is $\rho^{\mathrm{l}}(N^{\mathrm{s}} + N^{\mathrm{l}})_+$. (Here $(u)_+ = \max\{u, 0\}$.) Note that you have zero tax liability if $N^{\mathrm{s}} + N^{\mathrm{l}} \leq 0$, *i.e.*, your total long and short net gains is less than or equal to zero.

   *Apology.* Sorry this sounds complicated. In fact, this is a highly simplified version of the way taxes really work.

   *Hint.* The tax liability is neither a convex nor quasiconvex function of the long and short term net gains $N^{\mathrm{l}}$ and $N^{\mathrm{s}}$.

   (a) Explain how to find $s$ that minimizes the tax liability, subject to the constraints listed above, using convex optimization. Your solution can involve solving a modest number of convex problems.

   (b) Suppose you want to raise $C = 2300$ dollars from $n = 10$ tax lots, and the cost

basis and values of each lot are given by

$$\begin{aligned} b &= (400, 80, 400, 200, 400, 400, 80, 400, 100, 500), \\ v &= (500, 100, 500, 200, 700, 300, 120, 300, 150, 600). \end{aligned}$$

Carry out your method on this data with $L = 4$, $\rho_l = 0.2$, and $\rho_s = 0.3$. Give optimal values of $s_i$, and the optimal value of the tax liability. Compare this to the tax liability when you liquidate all tax lots proportionally, *i.e.*, $s = (C/\mathbf{1}^T v)v$.

**Solution.** The problem is not convex or quasiconvex as stated, since the tax liability function is not convex or quasiconvex. (The hint tells you that.)

We first work out an expression for the tax liability $T(N^{\mathrm{l}}, N^{\mathrm{s}})$ as a convex function of its two arguments. We observe that the net long and short term gains are linear in the variable $s$. $T(x, y)$ is a piecewise linear function, in regions defined by the signs of $x$, $y$, and $x + y$. From the description above, we have

$$T(x,y) = \begin{cases} \rho^{\mathrm{l}} x + \rho^{\mathrm{s}} y & x \geq 0,\ y \geq 0 \\ \rho^{\mathrm{l}}(x+y)_+ & x \geq 0,\ y < 0 \\ \rho^{\mathrm{s}}(x+y)_+ & x < 0,\ y \geq 0 \\ 0 & x \leq 0,\ y \leq 0. \end{cases}$$

This function is neither convex nor quasiconvex. It is piecewise linear, however, with four regions.

Here is a simple method to solve the problem: We optimize $s$ over each of the four regions, adding constraints that require the long and short term losses to have the right signs, and then choose the solution among these that gives us the smallest tax liability. Sometimes brute force is just the right thing to do.

This results in four different convex problems, which are readily converted to LPs. In each one we minimize $T(N^{\mathrm{l}}, N^{\mathrm{s}})$ subject to the the linear constraints relating these variables and $s$, along with the inequalities given above. In each of the four cases, we add the additional constraints on the signs of the two gains (and their sum), to ensure that we are in one region of $T$.

The first problem we solve is the Gain-Gain problem (*i.e.*, long and short term gains, the first of the four regions listed above.)

$$\begin{array}{ll} \text{minimize} & \rho^{\mathrm{l}} x + \rho^{\mathrm{s}} y \\ \text{subject to} & x = \sum_{i=1}^{L} g_i, \quad y = \sum_{i=L+1}^{n} g_i, \\ & g_i = (s_i/v_i)(v_i - b_i), \quad i = 1, \ldots, n, \\ & 0 \preceq s \preceq v, \\ & \mathbf{1}^T s = C, \\ & x \geq 0, \quad y \geq 0, \end{array}$$

with variables $s$, $g$, $x$, $y$. If this is infeasible, we record the tax liability in this case as $+\infty$.

The second problem we solve is the Gain-Loss problem (*i.e.*, long term gain, short term loss.)

$$\begin{array}{ll} \text{minimize} & \rho^{\mathrm{l}}(x+y)_+ \\ \text{subject to} & x = \sum_{i=1}^{L} g_i, \quad y = \sum_{i=L+1}^{n} g_i, \\ & g_i = (s_i/v_i)(v_i - b_i), \quad i = 1, \ldots, n, \\ & 0 \preceq s \preceq v, \\ & \mathbf{1}^T s = C, \\ & x \geq 0, \quad y \leq 0. \end{array}$$

If this is infeasible, we record the tax liability in this case as $+\infty$.

The third problem we solve is the Loss-Gain problem (*i.e.*, long term loss, short term gain.)

$$\begin{array}{ll} \text{minimize} & \rho^{\mathrm{s}}(x+y)_+ \\ \text{subject to} & x = \sum_{i=1}^{L} g_i, \quad y = \sum_{i=L+1}^{n} g_i, \\ & g_i = (s_i/v_i)(v_i - b_i), \quad i = 1, \ldots, n, \\ & 0 \preceq s \preceq v, \\ & \mathbf{1}^T s = C, \\ & x \leq 0, \quad y \geq 0. \end{array}$$

If this is infeasible, we record the tax liability in this case as $+\infty$.

The fourth problem we solve is the Loss-Loss problem (*i.e.*, long term loss, short term loss.)

$$\begin{array}{ll} \text{minimize} & 0 \\ \text{subject to} & x = \sum_{i=1}^{L} g_i, \quad y = \sum_{i=L+1}^{n} g_i, \\ & g_i = (s_i/v_i)(v_i - b_i), \quad i = 1, \ldots, n, \\ & 0 \preceq s \preceq v, \\ & \mathbf{1}^T s = C, \\ & x \leq 0, \quad y \leq 0. \end{array}$$

This is really just a feasibility problem; if it is feasible, then we have no tax liability, which is optimal (since in all cases the taxes are nonnegative). If it is not feasible, we take the tax to be $+\infty$.

Finally, we see which of the four problems has the lowest tax liability. That's our solution. The code for this is given below.

The four values of optimal tax liability are \$24.27 (gain-gain), \$16 (gain-loss), \$78 (loss-gain), and $+\infty$ (loss-loss). We choose the second one, with a long term gain and and short term loss. The optimal tax liability is \$16. The tax liability when you liquidate all tax lots proportionally is \$72.91.

The optimal liquidations are

$$s^\star = (419.43, 61.14, 419.43, 200, 0, 300, 0, 300, 0, 600).$$

(Most of these are either fully liquidated or not liquidated at all.)

*Remark.* It's possible, but not fun, to work out an 'analytical' solution for $s$ in each of the regions above. We did not ask you to do that.

Python solution for all parts:

```
import cvxpy as cp
import numpy as np
np.set_printoptions(precision=2, suppress=True)

n = 10
L = 4
C = 2300
rho_long = .2
rho_short = .3
b = np.array([400, 80, 400, 200, 400, 400, 80, 400, 100, 500])
v = np.array([500, 100, 500, 200, 700, 300, 120, 300, 150, 600])

s = cp.Variable(n)
g = cp.multiply(cp.multiply(s, 1. / v), v - b)
x = cp.sum(g[:L])
y = cp.sum(g[L:])
base_constraints = [s >= 0, s <= v, cp.sum(s) == C]

gaingain = cp.Problem(cp.Minimize(rho_long * x + rho_short * y),
                      base_constraints + [x >= 0, y >= 0])
result = gaingain.solve()
print("gaingain has", "%.3f" % result,
      "tax liability, selling", s.value)

gainloss = cp.Problem(cp.Minimize(rho_long * cp.pos(x + y)),
                      base_constraints + [x >= 0, y <= 0])
result = gainloss.solve()
print("gainloss has", "%.3f" % result,
      "tax liability, selling", s.value)

lossgain = cp.Problem(cp.Minimize(rho_short * cp.pos(x + y)),
                      base_constraints + [x <= 0, y >= 0])
result = lossgain.solve()
print("lossgain has", "%.3f" % result,
      "tax liability, selling", s.value)

lossloss = cp.Problem(cp.Minimize(0),
                      base_constraints + [x <= 0, y <= 0])
result = lossloss.solve()
```

```
print("lossloss has", "%.3f" % result,
      "tax liability, selling", s.value)

s.value = C / v.sum() * v
liability = 0.
if x.value >= 0 and y.value >= 0:
    liability = rho_long * x.value + rho_short * x.value
elif x.value >= 0 and y.value <= 0:
    liability = rho_long * np.maximum(x.value + y.value, 0)
elif x.value <= 0 and y.value >= 0:
    liability = rho_short * np.maximum(x.value + y.value, 0)
print("proportional has", "%.3f" %
      liability, "tax liability, selling", s.value)
```

Julia solution for all parts:

```
using Convex
using ECOS
using PyPlot

n = 10
L = 4
C = 2300
rho_long = 0.2
rho_short = 0.3

b = [400; 80; 400; 200; 400; 400; 80; 400; 100; 500]
v = [500; 100; 500; 200; 700; 300; 120; 300; 150; 600]

s = Variable(n)
g = dot(*)(dot(/)(s, v), v - b)
x = sum(g[1:L])
y = sum(g[L+1:n])

constr = Constraint[]
push!(constr, s >= 0)
push!(constr, s <= v)
push!(constr, sum(s) == C)

gaingain = minimize(rho_long * x + rho_short * y, constr + [x >= 0, y >= 0])
solve!(gaingain, ECOSSolver())
println("gaingain has $(gaingain.optval) tax liability, selling $(s.value)")
```

```
gainloss = minimize(rho_long * pos(x+y), constr + [x >= 0, y <= 0])
solve!(gainloss, ECOSSolver())
println("gainloss has $(gainloss.optval) tax liability, selling $(s.value)")

lossgain = minimize(rho_short * pos(x+y), constr + [x <= 0, y >= 0])
solve!(lossgain, ECOSSolver())
println("lossgain has $(lossgain.optval) tax liability, selling $(s.value)")

lossloss = minimize(0, constr + [x <= 0, y <= 0])
solve!(lossloss, ECOSSolver())
println("lossloss has $(lossloss.optval) tax liability, selling $(s.value)")

proportional = C/sum(v) * v
g = (proportional./v).*(v-b)
prop_liability = rho_long * sum(g[1:L]) + rho_short * sum(g[L+1:n])
println("proportional has $(prop_liability), selling $(proportional)")
```

Matlab solution for all parts:

```
n = 10;
L = 4;
C = 2300;
rho_long = 0.2;
rho_short = 0.3;
b = [400; 80; 400; 200; 400; 400; 80; 400; 100; 500];
v = [500; 100; 500; 200; 700; 300; 120; 300; 150; 600];

cvx_begin
    variable s(n);
    g = s.*(1./v).*(v - b);
    x = sum(g(1:L));
    y = sum(g(L+1:n));
    minimize (rho_long*x + rho_short*y);
    x >= 0;
    y >= 0;
    s >= 0;
    s <= v;
    sum(s) == C;
cvx_end
gg = cvx_optval;

cvx_begin
    variable s(n);
```

```
    g = s.*(1./v).*(v - b);
    x = sum(g(1:L));
    y = sum(g(L+1:n));
    minimize (rho_long*subplus(x + y));
    x >= 0;
    y <= 0;
    s >= 0;
    s <= v;
    sum(s) == C;
cvx_end
gl = cvx_optval;
s_opt = s;

cvx_begin
    variable s(n);
    g = s.*(1./v).*(v - b);
    x = sum(g(1:L));
    y = sum(g(L+1:n));
    minimize (rho_short*subplus(x + y));
    x <= 0;
    y >= 0;
    s >= 0;
    s <= v;
    sum(s) == C;
cvx_end
lg = cvx_optval;

cvx_begin
    variable s(n);
    g = s.*(1./v).*(v - b);
    x = sum(g(1:L));
    y = sum(g(L+1:n));
    minimize (0);
    x <= 0;
    y <= 0;
    s >= 0;
    s <= v;
    sum(s) == C;
cvx_end
ll = cvx_optval;

s = (C / sum(v)) * v;
```

```
g = s.*(1./v).*(v - b);
x = sum(g(1:L));
y = sum(g(L+1:n));
liability = 0.0;
if x >= 0 && y >= 0
    liability = rho_long * x + rho_short * y;
elseif x >= 0 && y <= 0
    liability = rho_long * subplus(x + y);
elseif x <= 0 && y >= 0
    liability = rho_short * subplus(x + y);
end

fprintf("gain-gain optimal tax liability: %d\n", gg);
fprintf("gain-loss optimal tax liability: %d\n", gl);
fprintf("loss-gain optimal tax liability: %d\n", lg);
fprintf("loss-loss optimal tax liability: %d\n", ll);
fprintf("proportional tax liability %d\n", liability);
fprintf("The optimal liquidations are \n");
disp(s_opt);
```

9. *Fitting with a nonnegative combination of vectors from ellipsoids.* You are given ellipsoids $\mathcal{E}_1, \ldots, \mathcal{E}_n \subset \mathbf{R}^k$, and the vector $b \in \mathbf{R}^k$. Explain how to use convex optimization to choose $a_i \in \mathcal{E}_i$, $i = 1, \ldots, n$, and nonnegative $x_1, \ldots, x_n \in \mathbf{R}$, that minimize

$$\left\| \sum_{i=1}^{n} x_i a_i - b \right\|_2 .$$

You can use any parametrization of the ellipsoids you like, for example,

$$\mathcal{E}_i = \{a \mid \|P_i a + q_i\|_2 \leq 1\},$$

or

$$\mathcal{E}_i = \{P_i u + q_i \mid \|u\|_2 \leq 1\},$$

or

$$\mathcal{E}_i = \left\{a \mid (a - c_i)^T P_i^{-1} (a - c_i) \leq 1\right\},$$

with $P_i \in \mathbf{S}^k_{++}$ and $c_i \in \mathbf{R}^k$.

*Remark.* This is the opposite situation from robust approximation. In robust approximation, the $a_i$'s would be chosen to maximize the objective, once you choose $x$. Here, however, the $a_i$'s are chosen to minimize the objective, along with $x$.

**Solution.** Our hint about the parameterization of the ellipsoids was really a red herring; it turns out all of the parameterizations we suggested work.

**First parametrization.** If we use the first ellipsoid parameterization above,

$$\mathcal{E}_i = \{a \mid \|P_i a + q_i\|_2 \leq 1\},$$

we can formulate a convex problem in the following way. Define $z_i = x_i a_i$, so the objective is $\| \sum_{i=1}^n z_i - b\|_2$ is convex. For $x_i > 0$, we have $z_i/x_i = a_i \in \mathcal{E}_i$, which can be written as $\|P_i(z_i/x_i) + q_i\|_2 \leq 1$, which can then be expressed as $\|P_i z_i + x_i q_i\|_2 \leq x_i$. This is a convex constraint, in fact, a second order cone constraint. When $x_i = 0$, we have $z_i = 0$, so in this case too we can express the constraint as $\|P_i z_i + x_i q_i\|_2 \leq x_i$. So, the problem is equivalent to the convex problem

$$\begin{array}{ll} \text{minimize} & \| \sum_{i=1}^n z_i - b\|_2 \\ \text{subject to} & \|P_i z_i + x_i q_i\|_2 \leq x_i, \quad i = 1, \ldots, n, \end{array}$$

with variables $z_i \in \mathbf{R}^k$ and $x \in \mathbf{R}^n$. (The constraint $x_i \geq 0$ is implicit.) We solve this problem to get $z_i^\star$ and $x^\star$, and then we recover the solution of the original problem as $a_i^\star = x_i^\star z_i^\star$.

**Second parameterization.** If we used the ellipsoid parameterization

$$\mathcal{E}_i = \{P_i u + q_i \mid \|u\|_2 \leq 1\},$$

then the problem is equivalent to

$$\begin{array}{ll} \text{minimize} & \|\sum_{i=1}^n (P_i z_i + x_i q_i) - b\|_2 \\ \text{subject to} & \|z_i\|_2 \leq x_i, \quad i = 1, \ldots, n. \end{array}$$

**Third parameterization.** If we used the ellipsoid parameterization

$$\mathcal{E}_i = \left\{a \mid (a - c_i)^T P_i^{-1} (a - c_i) \leq 1\right\},$$

then the problem is equivalent to

$$\begin{array}{ll} \text{minimize} & \|\sum_{i=1}^n z_i - b\|_2 \\ \text{subject to} & \|P_i^{-1/2}(z_i - x_i c_i)\|_2 \leq x_i, \quad i = 1, \ldots, n. \end{array}$$

In addition to the three solutions above, some students made different changes of variables (for example, multiplying $z_i$ by $P_i^{-1/2}$) that result in equivalent convex problems. We accepted these as long as they were clear and mathematically correct.