EE364a: Convex Optimization I N. Moehle
August 13–14 or August 14–15, 2016

Final Exam Solutions

This is a 24 hour take-home final. Please turn it in at Bytes Cafe in the Packard building, 24 hours after you pick it up.

You may use any books, notes, or computer programs, but you may not discuss the exam with anyone until August 16, after everyone has taken the exam. The only exception is that you can ask us for clarification, via the course staff email address. We've tried pretty hard to make the exam unambiguous and clear, so we're unlikely to say much.

Please make a copy of your exam, or scan it, before handing it in.

Please attach the cover page to the front of your exam. Assemble your solutions in order (problem 1, problem 2, problem 3, ...), starting a new page for each problem. Put everything associated with each problem (*e.g.*, text, code, plots) together; do not attach code or plots at the end of the final.

We will deduct points from long, needlessly complex solutions, even if they are correct. Our solutions are not long, so if you find that your solution to a problem goes on and on for many pages, you should try to figure out a simpler one. We expect neat, legible exams from everyone, including those enrolled Cr/N.

When a problem involves computation you must give all of the following: a clear discussion and justification of exactly what you did, the source code that produces the result, and the final numerical results or plots.

Files containing problem data can be found in the usual place,

`http://www.stanford.edu/~boyd/cvxbook/cvxbook_additional_exercises/`

Please respect the honor code. Although we allow you to work on homework assignments in small groups, you cannot discuss the final with anyone, at least until everyone has taken it.

All problems have equal weight. Some are easy. Others, not so much.

Be sure you are using the most recent version of CVX, CVXPY, or Convex.jl. Check your email often during the exam, just in case we need to send out an important announcement.

Some problems involve applications. But you do not need to know *anything* about the problem area to solve the problem; the problem statement contains everything you need.

1. *Funding an expense stream.* Your task is to fund an expense stream over n time periods. We consider an expense stream $e \in \mathbf{R}^n$, so that e_t is our expenditure at time t.

   One possibility for funding the expense stream is through our bank account. At time period t, the account has balance b_t and we withdraw an amount w_t. The value of our bank account accumulates with an interest rate ρ per time period, less withdrawals:

   $$b_{t+1} = (1+\rho)b_t - w_t.$$

   We assume the account value must be nonnegative, so that $b_t \geq 0$ for all t.

   We can also use other investments to fund our expense stream, which we purchase at the initial time period $t = 1$, and which pay out over the n time periods. The amount each investment type pays out over the n time periods is given by the *payout matrix* P, defined so that P_{tj} is the amount investment type j pays out at time period t per dollar invested. There are m investment types, and we purchase $x_j \geq 0$ dollars of investment type j. In time period t, the total payout of all investments purchased is therefore given by $(Px)_t$.

   In each time period, the sum of the withdrawals and the investment payouts must cover the expense stream, so that

   $$w_t + (Px)_t \geq e_t$$

   for all $t = 1, \ldots, n$.

   The total amount we invest to fund the expense stream is the sum of the initial account balance, and the sum total of the investments purchased: $b_1 + \mathbf{1}^T x$.

   (a) Show that the minimum initial investment that funds the expense stream can be found by solving a convex optimization problem.

   (b) Using the data in `expense_stream_data.*`, carry out your method in part (a). On three graphs, plot the expense stream, the payouts from the m investment types (so m different curves), and the bank account balance, all as a function of the time period t. Report the minimum initial investment, and the initial investment required when no investments are purchased (so $x = 0$).

Solution.

(a) The problem is

$$\begin{array}{ll} \text{minimize} & b_1 + \mathbf{1}^T x \\ \text{subject to} & w_t + (Px)_t \geq e_t \quad \text{for all } t = 1, \ldots, n \\ & w + Px \succeq e \\ & x \succeq 0 \\ & b \succeq 0 \end{array}$$

with variables $x \in \mathbf{R}^n$, $w \in \mathbf{R}^n$, and $b \in \mathbf{R}^n$. This problem is convex as stated. Many people had the urge to make $w \succeq 0$, which was not stated anywhere in the problem (and has strange effects, including that it is optimal to throw money away, which you can verify out by checking $w_t + (Px)_t > e_t$ for some time periods). We decided to give full amnesty on this issue, so you could get full credit either way.

(b) The investments are 177.5 (197.9 if you took $w \succeq 0$) and 336.5
The following Matlab code solves the problem:

```
expense_stream_data

cvx_begin % no investments
    variables x(m) b(n) w(n)
    minimize(b(1) + sum(x))
    w + P*x == e
    x == 0
    b >= 0
    for t = 1:n-1
      b(t+1) == (1+rho)*b(t) - w(t)
    end
cvx_end

total_cost_no_investments = cvx_optval

cvx_begin % with investments
    variables x(m) b(n) w(n)
    minimize(b(1) + sum(x))
    w + P*x >= e
    x >= 0
    b >= 0
    for t = 1:n-1
      b(t+1) == (1+rho)*b(t) - w(t)
    end
```

```
cvx_end

total_cost = cvx_optval

clf
hold all
subplot(311)
stairs(e)
stairs(w + P*x)
ylabel('expense stream')
subplot(312)
stairs(P.*(ones(n,1)*x'), 'k')
ylabel('investment payouts')
subplot(313)
stairs(b)
xlabel('time period')
ylabel('balance')
print -depsc expense_stream.eps
```

The following Python code solves the problem:

```
import numpy as np
import numpy.matlib as ml
import cvxpy as cp
#import matplotlib.pyplot as ppt

from expense_stream_data import *

x = cp.Variable(m)
b = cp.Variable(n)
w = cp.Variable(n)

constr =  [w + P*x == e]
constr += [x >= 0]
constr += [w >= 0]
constr += [b >= 0]
constr += [b[t+1] == (1+rho)*b[t] - w[t] for t in range(n-1)]
p = cp.Problem(cp.Minimize(b[1] + sum(x)), constr)
p.solve(verbose=True)
print(p.value) # with investments

#ppt.subplot(311)
#ppt.step(range(n), e)
#ppt.step(range(n), w.value + P*x.value)
```

```
#ppt.ylabel("expense stream")
#ppt.subplot(312)
#[ppt.step(range(n), P[:,j]*x.value[j]) for j in range(m)]
#ppt.ylabel("investment payments")
#ppt.subplot(313)
#ppt.step(range(n), b.value)
#ppt.xlabel("time period")
#ppt.ylabel("balance")
#ppt.show()

constr += [x == 0]
p = cp.Problem(cp.Minimize(b[1] + sum(x)), constr)
p.solve()
print(p.value) # no investments
```

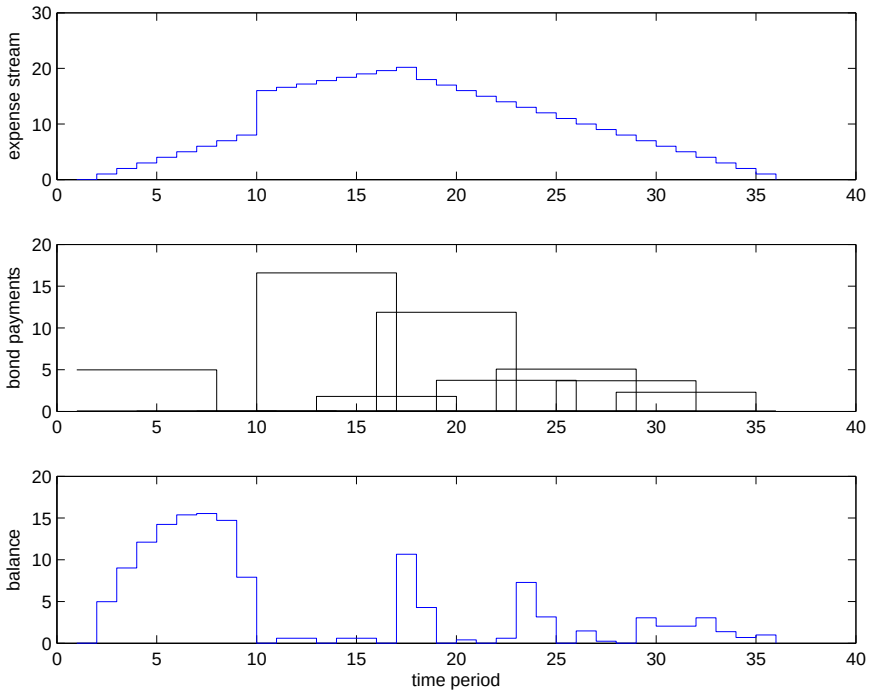
The following Julia code solves the problem:

```
using Convex
using PyPlot

include("expense_stream_data.jl")

x = Variable(m)
b = Variable(n)
w = Variable(n)
prob = minimize(b[1] + sum(x))
prob.constraints += w + P*x == e
prob.constraints += x  0
prob.constraints += b  0
for t in 1:n-1
    prob.constraints += b[t+1] == (1+)*b[t] - w[t]
end
solve!(prob)
display(prob.optval) # with investments

clf
hold(true)
subplot(311)
step(1:n, e)
step(1:n, w.value + P*x.value)
ylabel("expense stream")
subplot(312)
step(1:n, P.*(ones(n,1)*x.value'))
ylabel("investment payments")
```




```
subplot(313)
step(1:n, b.value)
xlabel("time period")
ylabel("balance")


prob.constraints += x == 0
solve!(prob)
display(prob.optval) # no investments
```

2. *Approximations of the PSD cone.* A symmetric matrix is positive semidefinite if and only if all its principal minors are nonnegative. Here we consider approximations of the positive-semidefinite cone produced by partially relaxing this condition.

   Denote by $K_{1,n}$ the cone of matrices whose 1×1 principal minors (*i.e.*, diagonal elements) are nonnegative, so that

   $$K_{1,n} = \{X \in \mathbf{S}^n \mid X_{ii} \geq 0 \text{ for all } i\}.$$

   Similarly, denote by $K_{2,n}$ the cone of matrices whose 1×1 and 2×2 principal minors are nonnegative:

   $$K_{2,n} = \left\{ X \in \mathbf{S}^n \;\middle|\; \begin{bmatrix} X_{ii} & X_{ij} \\ X_{ij} & X_{jj} \end{bmatrix} \succeq 0, \text{ for all } i \neq j \right\},$$

   i.e., the cone of symmetric matrices with positive semidefinite 2×2 principal submatrices. These two cones are convex (and in fact, proper), and satisfy the relation:

   $$K_{1,n}^* \subseteq K_{2,n}^* \subseteq \mathbf{S}_+^n \subseteq K_{2,n} \subseteq K_{1,n},$$

   where $K_{1,n}^*$ and $K_{2,n}^*$ are the dual cones of $K_{1,n}$ and $K_{2,n}$, respectively. (The last two inclusions are immediate, and the first two inclusions follow from the second bullet on page 53 of the text.)

   (a) Give an explicit characterization of $K_{1,n}^*$.

   (b) Give an explicit characterization of $K_{2,n}^*$.
   Hint: You can use the fact that if $K = K_1 \cap \cdots \cap K_m$, then $K^* = K_1^* + \cdots + K_m^*$.

   (c) Consider the problem

   $$\begin{array}{ll} \text{minimize} & \mathbf{Tr}\, CX \\ \text{subject to} & \mathbf{Tr}\, AX = b \\ & X \in K \end{array}$$

   with variable $X \in \mathbf{S}^n$. The problem parameters are $C \in \mathbf{S}^n$, $A \in \mathbf{S}^n$, $b \in \mathbf{R}$, and the cone $K \subseteq \mathbf{S}^n$. Using the data in `psd_cone_approx_data.*`, solve this problem five times, each time replacing K with one of the five cones $K_{1,n}$, $K_{2,n}$, $\mathbf{S}_+^n$, $K_{2,n}^*$, and $K_{1,n}^*$. Report the five different optimal values you obtain.
   Note: Python users who run into numerical difficulties might want to use the SCS solver by using `prob.solve(solver=cvxpy.SCS)`.

Note: For parts (a) and (b), the shorter and clearer your description is, the more points you will receive. At the very least, it should be possible to implement your description in CVX.*.

Solution:

(a) We have

$$K_{1,n}^* = \{Y \in \mathbf{S}^n \mid \mathbf{Tr}\, XY \geq 0,\ X_{ii} \geq 0 \text{ for all } i\}.$$

Since there are no restrictions on the off-diagonal components of X, the corresponding off-diagonal components of Y must all be zero. Because the diagonal elements of X are nonnegative, the diagonal elements of Y must be nonnegative as well. Therefore, $K_{1,n}^*$ is

$$K_{1,n}^* = \{Y \in \mathbf{S}^n \mid Y_{ii} \geq 0 \text{ for all } i,\ Y_{ij} = 0 \text{ for all } i \neq j\},$$

i.e., the set of positive semidefinite diagonal matrices.

(b) For fixed i and j, define

$$K_{2,n}^{ij} = \left\{ X \in \mathbf{S}^n \;\middle|\; \begin{bmatrix} X_{ii} & X_{ij} \\ X_{ij} & X_{jj} \end{bmatrix} \succeq 0 \right\}.$$

This cone is the subset of $n \times n$ symmetric matrices, in which the 2×2 subblock given by indices i and j is positive semidefinite. Note that we can rewrite $K_{2,n}$ as $K_{2,n} = \bigcap_{i \neq j} K_{2,n}^{ij}$.

For an member X of $K_{2,n}^{ij}$, the 2×2 block given by indices i and j is positive semidefinite, so the corresponding block of any member Y of the dual cone must also be positive semidefinite. All other elements of X are unrestricted, so the corresponding elements of Y must all be zero. The dual cone is therefore

$$(K_{2,n}^{ij})^* = \left\{ Y \in \mathbf{S}^n \;\middle|\; \begin{bmatrix} Y_{ii} & Y_{ij} \\ Y_{ij} & Y_{jj} \end{bmatrix} \succeq 0,\ Y_{kl} = 0 \text{ if } (k,l) \neq (i,i), (i,j), (j,i) \text{ or } (j,j) \right\}.$$

This cone is set of positive semidefinite matrices Y with at most four nonzero elements Y_{ii}, Y_{ij}, Y_{ji}, and Y_{jj}. Using the hint, we have

$$K_{2,n}^* = \left(\sum_{i \neq j} \left(K_{2,n}^{ij} \right)^* \right).$$

Therefore, any $Y \in K_{2,n}^*$ can be written as a sum of sparse semidefinite matrices, with at most four nonzero elements each.

There are some equivalent characterizations of this cone. For example, a symmetric matrix Y is in $K_{2,n}^*$ if and only if for some diagonal matrix $D \succeq 0$, DYD is diagonal dominant. Equivalently, Y can be written as MM^T, where the columns of M have no more than two nonzero elements.

(c) We can solve the problem directly using the solutions to parts (a) and (b). It is also possible to solve this part without completing parts (a) or (b), by solving the dual problem:

$$\begin{array}{ll} \text{maximize} & bz \\ \text{subject to} & X = C - Az \\ & X \in K^* \end{array}$$

with variables $z \in \mathbf{R}$ and $X \in \mathbf{S}^n$. Because $(K_{1,n}^*)^* = K_{1,n}$ and $(K_{2,n}^*)^* = K_{2,n}$, to solve the original problem with $K = K_{2,n}^*$, we can solve the dual with $K^* = K_{2,n}$. The following code solves the primal and dual problems with all five cones, in all three languages. The optimal values, in increasing order, are 0, 2.74, 3.04, 5.09, and 9.18.

The following Matlab code solves the problem:

```
psd_cone_approx_data

for primal = [1]  % Solve primal and dual problem,
    for cone = 5  % for all five cones.
        cvx_begin
            variable X(n,n) symmetric
            switch cone
                case 1  % K = K1
                    diag(X) >= 0
                case 2  % K = K2
                    for i = 1:n, for j = 1:n
                        if i ~= j
                            [X(i,i) X(j,i); X(i,j) X(j,j)] ...
                                 == semidefinite(2);
                        end
                    end, end
                case 3  % K = S++
                    X == semidefinite(n);
                case 4  % K = K1star
                    variable Z(n,n,n,n)
                    for i = 1:n, for j = 1:n, for k = 1:n
                            if k ~= i && k ~= j
                                Z(k,k,i,j) == 0
                            end
                        Z(:,:,i,j) == semidefinite(n);
                    end, end, end
                    X == sum(sum(Z, 3), 4)
                case 5  % K = K2star
                    variable x(n)
                    X == diag(x);
                    x >= 0;
            end

            if primal  % solve primal problem
                minimize(trace(X*C))
                trace(A*X) == b
```

```
            else  % solve dual problem
                variable z
                maximize(z*b)
                X == C - A*z;
            end
        cvx_end
        if primal
            pstar(cone) = cvx_optval;
        else
            dstar(5 - cone + 1) = cvx_optval;
        end
    end
end

% print the primal and dual opt values, for all 5 cones:
pstar
dstar
```

The following Python code solves the problem:

```
import numpy as np
import numpy.matlib as ml
import cvxpy as cp

from psd_cone_approx_data import *

pstar = []
dstar = []
for primal in [False,True]:  # Solve primal and dual problem,
  for cone in range(5):    # for all five cones.

    X = cp.Variable(n,n)
    constr = []

    # CONE CONSTRAINT
    if cone == 0:
      constr += [cp.diag(X) >= 0]
    elif cone == 1:
      for i in range(n):
        for j in range(n):
          if i != j:
            constr += [cp.vstack(cp.hstack(X[i,i], X[j,i]),
                       cp.hstack(X[i,j], X[j,j])) == cp.Semidef(2)]
    elif cone == 2:
```

```
      constr += [X == cp.Semidef(n)]
    elif cone == 3:
      Zsum = ml.zeros((n,n))
      for i in range(n):
        for j in range(n):
          Z = cp.Semidef(n,n)
          Zsum = Zsum + Z
          for k in range(n):
            if k != i and k != j:
              constr += [Z[k,k] == 0]
      constr += [X == Zsum]
    elif cone == 4:
      x = cp.Variable(n)
      constr += [X == cp.diag(x)]
      constr += [x >= 0]

    # OBJECTIVE AND EQUALITY CONSTRAINT
    if primal:  # solve primal problem
      constr += [cp.trace(A*X) == b]
      p = cp.Problem(cp.Minimize(cp.trace(X.T*C)), constr)
    else:  # solve dual problem
      z = cp.Variable()
      constr += [X == C - A*z]
      p = cp.Problem(cp.Maximize(z*b), constr)

    # SOLVE
    p.solve(solver=cp.SCS)
    if primal:  # solve primal problem
      pstar.append(p.value)
    else:  # solve dual problem
      dstar.append(p.value)

# print the primal and dual opt values, for all 5 cones:
print(pstar)
print(dstar)
```

The following Julia code solves the problem:

```
using Convex

include("psd_cone_approx_data.jl")

pstar = zeros(5)
dstar = zeros(5)
```

```
for primal in [false,true]  # solve primal and dual problem, for all five cones
  for cone in 1:5

    # OBJECTIVE AND EQUALITY CONSTRAINT
    X = Variable(n,n)
    if primal  # solve primal problem
      p = minimize(trace(X'*C))
      p.constraints += trace(A*X) == b
    else  # solve dual problem
      z = Variable()
      p = maximize(z*b)
      p.constraints += X == C - A*z
    end

    # CONE CONSTRAINT
    if cone == 1
      p.constraints += diag(X)  0
    elseif cone == 2
      for i in 1:n, j in 1:n
        if i != j
          p.constraints +=
              [X[i,i] X[j,i]; X[i,j] X[j,j]] == Semidefinite(2)
        end
      end
    elseif cone == 3
      p.constraints += X == Semidefinite(n)
    elseif cone == 4
      Zsum = zeros(n,n)
      for i in 1:n, j in 1:n
        Z = Semidefinite(n)
        for k in 1:n
          if k != i && k != j
            p.constraints += Z[k,k] == 0
          end
        end
        Zsum += Z
      end
      p.constraints += X == Zsum
    elseif cone == 5
      x = Variable(n)
      p.constraints += X == diagm(x)
      p.constraints += x  0
```

```
    end

    # SOLVE
    solve!(p)
    if primal  # solve primal problem
      pstar[cone] = p.optval
    else  # solve dual problem
      dstar[5 - cone + 1] = p.optval
    end
  end
end

# print the primal and dual opt values, for all 5 cones:
display(pstar)
display(dstar)
```

3. *Direct standardization.* Consider a random variable $(x, y) \in \mathbf{R}^n \times \mathbf{R}$, and N samples $(x_1, y_1), \ldots, (x_N, y_N) \in \mathbf{R}^n \times \mathbf{R}$, which we will use to estimate the (marginal) distribution of y. If the given samples were chosen according to the joint distribution of (x, y), a reasonable estimate for the distribution of y would be the uniform empirical distribution, which takes on values $y_1, \ldots, y_N$ each with probability $1/N$. (If y is Boolean, *i.e.*, $y \in \{0, 1\}$, we are using the fraction of samples with $y = 1$ as our estimate of $\mathbf{Prob}(y = 1)$.)

   The bad news is that the samples $(x_1, y_1), \ldots, (x_N, y_N) \in \mathbf{R}^n \times \mathbf{R}$ were *not* chosen from the distribution of (x, y), but instead from another (unknown, but presumably similar) distribution. The good news is that we know $\mathbf{E}\, x$, the expected value of x. We will use our knowledge of $\mathbf{E}\, x$, together with the samples, to estimate the distribution of y. *Direct standardization* replaces the uniform empirical distribution with a weighted one, which takes on values y_i with probability π_i, where $\pi \succeq 0$, $\mathbf{1}^T \pi = 1$. The weights or sample probabilities π are found by maximizing the entropy $-\sum_{i=1}^N \pi_i \log \pi_i$, subject to the requirement that the weighted sample expected value of x matches the known probabilities of x in the distribution, $\mathbf{E}\, x$. This can be expressed as $\sum_{i=1}^N \pi_i x_i = \mathbf{E}\, x$. (Both x_i and $\mathbf{E}\, x$ are known.)

   (a) Explain why choosing π is a convex optimization problem.

   (b) Consider the simple case with $n = 1$, and $x \in \{0, 1\}$, so $\mathbf{E}\, x = \mathbf{Prob}(x = 1)$. Find the optimal sample weights $\pi_i^\star$ (analytically). Explain your solution in the following case. The samples are people, with $x = 0$ meaning the person is male, and $x = 1$ meaning the person is female. The overall population is known to have equal numbers of females and males, but in the sample population the male : female proportions are $0.7 : 0.3$.

   (c) The data in `direct_std_data.*` contain the samples $x^{(i)}$ and $y^{(i)}$, as well as $\mathbf{E}\, x$. Find the weights $\pi^\star$, and report the weighted empirical distribution. On the same plot, compare the cumulative distributions of

   - the uniform empirical distribution,
   - the weighted empirical distribution using $\pi^\star$, and
   - the true distribution of y.

   The true and empirical distributions are provided in the data file. (For example, the 20 elements of `p_true` give $\mathbf{Prob}(y = 1)$ up to $\mathbf{Prob}(y = 20)$, in order).

   Note: Julia users might want to use the ECOS solver, by including `using ECOS`, and solving by using `solve!(prob, ECOSSolver())`.

   Note: You don't need to know this to solve the problem, but the data for part (c) are real. The random variable x is a vector of a student's gender, age, and mother's and father's educational attainment, and y is the student's score on a standardized test.

Solution.

(a) We choose π as the solution to the convex optimization problem

$$\begin{array}{ll} \text{maximize} & -\sum_{i=1}^n \pi_i \log \pi_i \\ \text{subject to} & \pi \succeq 0 \\ & \mathbf{1}^T \pi = 1 \\ & \sum_{i=1}^n x_i \pi_i = \mathbf{E}\, x \end{array}$$

The variable is π, and the parameters are $x^{(i)}$ and $\mathbf{E}\, x$.

(b) The intuition here is that high entropy distributions are "very random" or "very uniform," so we want the "most uniform" distribution that satisfies the constraints, *i.e.*, one that puts equal weight on all the females, and equal weight on all the males. The solution is that if $x_i = 1$, then $\pi_i = 5/(3N)$, and if $x_i = 0$, then $\pi_i = 5/(7N)$. (We didn't really care how you came up with this, as long as there was some clear, logical argument.)

We didn't ask you to show optimality, but here it is anyway. The point is feasible, since we have $\pi \succeq 0$, and also

$$\mathbf{1}^T \pi = \sum_{i=1}^N \pi_i = (0.3N)\frac{5}{3N} + (0.7N)\frac{5}{7N} = 1,$$

and

$$\sum_{i=1}^N x^{(i)} \pi_i = \sum_{i=1}^{0.3N} \frac{5}{3N} = (0.3N)\frac{5}{3N} = 0.5 = \mathbf{E}\, x.$$

Apart from primal feasibility, the other KKT conditions are

$$\log \pi_i + 1 + \nu - \lambda_i + \mu x^{(i)} = 0, \qquad \lambda_i \succeq 0, \qquad \lambda_i \pi_i = 0.$$

(All equations hold for $i = 1, \ldots, N$.) We can simply take $\lambda = 0$ to eliminate the last two conditions. When we in our value for π, the remaining N equations are reduced to two:

$$\log\left(\frac{5}{7N}\right) + 1 + \nu = 0, \qquad \log\left(\frac{5}{3N}\right) + 1 + \nu + \mu = 1$$

These are two equations with two unknowns, which we can solve to obtain the optimal dual variables. (Note that the key step In reducing these N equations to two equations was that there are only two unique values in the vector π, which was what our intuition was telling us.)

Some people gave the more general solution, which was: $\pi_i = (\mathbf{E}\, x)/M$ if $x_i = 1$, and $\pi_i = (1 - \mathbf{E}\, x)/(N - M)$, if $x_i = 0$, where M is the number of samples for which $x^{(i)} = 1$.

(c) The probability distribution we were looking for is given by

$$\begin{bmatrix} 0.0884 \\ 0 \\ 0 \\ 0 \\ 0.0135 \\ 0.0486 \\ 0.0322 \\ 0.0819 \\ 0.0888 \\ 0.1383 \\ 0.1439 \\ 0.0601 \\ 0.0374 \\ 0.0838 \\ 0.0550 \\ 0.0512 \\ 0.0231 \\ 0.0404 \\ 0 \\ 0.0133 \end{bmatrix},$$

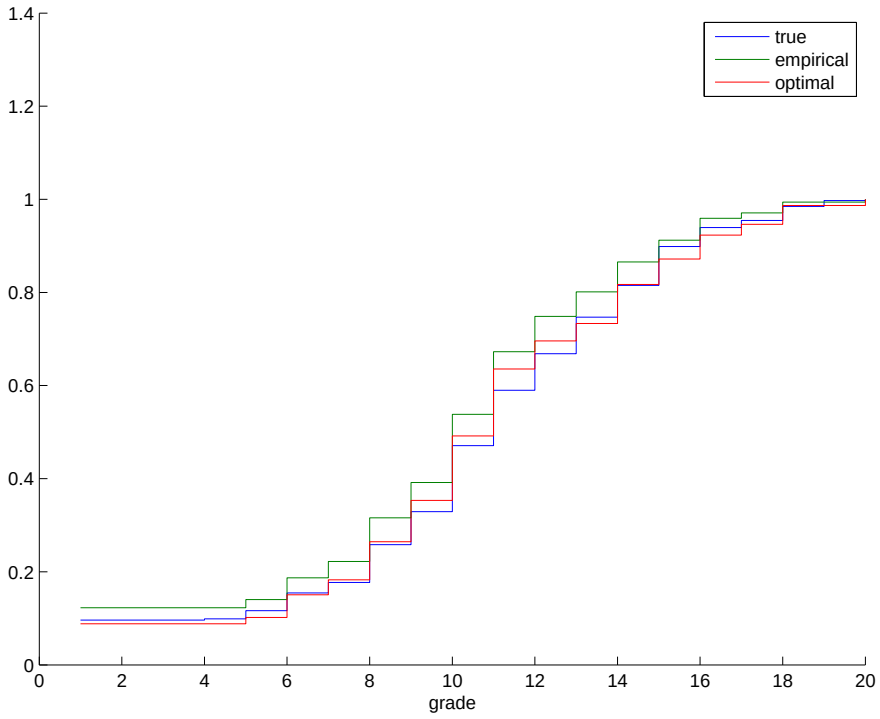
where element i gives $\mathbf{Prob}(y = i)$. The three CDFs we asked for are plotted below.

The following Matlab code solves the problem:

```
direct_std_data

cvx_begin
    variables Pi(N)
    maximize(sum(entr(Pi)))
    Pi'*x == Ex
    Pi >= 0
    sum(Pi) == 1
cvx_end

p = zeros(20,1);
for grade = 1:20
    for i = 1:N
        if y(i) == grade
```




```
            p(grade) = p(grade) + Pi(i);
        end
    end
end

figure
hold all
plot(p_true)
plot(p_empirical)
plot(p)
xlabel('grade')
legend('true','empirical','optimal')

figure
hold all
stairs(cumsum(p_true))
stairs(cumsum(p_empirical))
stairs(cumsum(p))
xlabel('grade')
legend('true','empirical','optimal')
print -depsc direct_std.eps
```

The following Python code solves the problem:

```
import numpy as np
import numpy.matlib as ml
import cvxpy as cp
import matplotlib.pyplot as ppt

from direct_std_data import *

pi = cp.Variable(N)
constr = []
constr += [pi.T*x == Ex.T]
constr += [pi >= 0]
constr += [sum(pi) == 1]
prob = cp.Problem(cp.Maximize(sum(cp.entr(pi))), constr)
prob.solve()

p = ml.zeros((20,1))
for i in range(N):
    p[y[i,0]-1] += pi.value[i,0]

ppt.hold(True)
ppt.step(range(1,21), np.cumsum(p_true).T)
ppt.step(range(1,21), np.cumsum(p_empirical).T)
ppt.step(range(1,21), np.cumsum(p).T)
ppt.xlabel("grade")
ppt.legend(["true","empirical","optimal"])
ppt.show()
```

The following Julia code solves the problem:

```
using Convex
using ECOS
using PyPlot

include("direct_std_data.jl")

 = Variable(N)
prob = maximize(sum(entropy()))
prob.constraints += '*x == Ex
prob.constraints +=   0
prob.constraints += sum() == 1
solve!(prob, ECOSSolver())

p = zeros(20)
for i in 1:N
```

```
    p[y[i]] += .value[i]
end

hold(true)
step(1:20, cumsum(p_true))
step(1:20, cumsum(p_empirical))
step(1:20, cumsum(p))
xlabel("grade")
legend(["true","empirical","optimal"])
```

4. *Thermodynamic potentials.* We consider a mixture of k chemical species. The *internal energy* of the mixture is

$$U(S, V, N_1, \ldots, N_k),$$

where S is the entropy of the mixture, V is the volume occupied by the mixture, and N_i is the quantity (in moles) of chemical species i. We assume the function U is convex. (Real internal energy functions satisfy this and other interesting properties, but we won't need any others for this problem.) The *enthalpy* H, the *Helmoltz free energy* A, and the *Gibbs free energy* G are defined as

$$\begin{aligned} H(S, P, N_1, \ldots, N_k) &= \inf_V U(S, V, N_1, \ldots, N_k) - PV, \\ A(T, V, N_1, \ldots, N_k) &= \inf_S U(S, V, N_1, \ldots, N_k) + TS, \\ G(T, P, N_1, \ldots, N_k) &= \inf_{S,V} U(S, V, N_1, \ldots, N_k) + TS - PV. \end{aligned}$$

The variables T and P can be interpreted physically as the temperature and pressure of the mixture. These four functions are called *thermodynamic potentials*. We refer to the arguments S, V, and $N_1, \ldots N_k$ as the extensive variables, and the arguments T and P as the intensive variables.

(a) Show that H, A, and G are convex in the extensive variables, when the intensive variables are fixed.

(b) Show that H, A, and G are concave in the intensive variables, when the extensive variables are fixed.

(c) We consider a simple reaction involving three species,

$$2[\text{species 1}] \rightleftharpoons [\text{species 2}] + [\text{species 3}],$$

carried out at temperature T_{react} and volume V_{react}. The Helmholtz free energy of the mixture is

$$A(T, V, N_1, N_2, N_3) = T \sum_{j=1}^{3} N_j (s_{0,j} - Rc_j) + TR \sum_{j=1}^{3} N_j \log \left(N_j \left(\frac{V_0}{V} \right) \left(\frac{T_0}{T} \right)^{c_j} \right),$$

where R, V_0, T_0, $s_{0,j}$, and c_j, for $j = 1, \ldots, k$, are known, positive constants. The equilibrium molar quantities $N_1^\star$, $N_2^\star$, and $N_3^\star$ of the three species are those that minimize $A(T_{\text{react}}, V_{\text{react}}, N_1, N_2, N_3)$ subject to the stoichiometry constraints

$$N_1 = N_{1,\text{init}} - 2z, \quad N_2 = N_{2,\text{init}} + z, \quad N_3 = N_{3,\text{init}} + z,$$

where $N_{j,\text{init}}$ is the initial quantity of species j, and the variable z gives the amount of the reaction that has proceeded. For the values of T_{react}, V_{react}, R, V_0, T_0, $s_{0,j}$, and c_j given in `thermo_potentials_data.*`, report the equilibrium molar quantities $N_1^\star$, $N_2^\star$, and $N_3^\star$.

Note: Julia users might want the ECOS solver. Include `using ECOS`, and solve by using `solve!(prob, ECOSSolver())`.

Solution:

(a) Consider the enthalpy H, which has extensive variables S and $N_1, \ldots, N_k$, and intensive variable P. We fix the extensive variable P, and consider convexity in S and $N_1, \ldots, N_k$.

Note that for fixed P, the function $U(S, V, N_1, \ldots, N_k) - PV$ is (jointly) convex in S, V, and $N_1, \ldots N_k$, so by performing partial minimization over V, (joint) convexity in S and $N_1, \ldots, N_k$ is maintained.

A similar argument shows convexity of the other two potentials in their extensive variables.

(b) Consider again the enthalpy H, which has extensive variables S and $N_1, \ldots, N_k$, and intensive variable P. We fix the extensive variables S and $N_1, \ldots, N_k$, and consider concavity in V.

Note that the function $U(S, V, N_1, \ldots, N_k) - PV$ is affine in P. The infimum over affine functions is concave. A similar argument shows concavity of the other two potentials in their intensive variables.

(c) The problem is convex as stated. The only difficulty is expressing the objective in a DCP-compliant form using the functions available in CVX.*.

The function A looks complicated, but when the constants are grouped together, it has the simpler form

$$\sum_{j=1}^{k} \alpha_j N_j + \sum_{j=1}^{k} \beta N_j \log \gamma_j N_j,$$

where $\alpha_j = T(s_{0,j} - Rc_j)$, $\beta = TR$, and $\gamma_j = (V_0/V_{\text{react}})(T_0/T_{\text{react}})^{c_j}$. This is equivalent to

$$\sum_{j=1}^{k} \alpha_j N_j + \sum_{j=1}^{k} (\beta/\gamma_j)(\gamma_j N_j) \log(\gamma_j N_j)$$

This is the sum of a linear functions (in N_j) and negative entropy functions. (the entropy function is given by $f(x) = -x \log x$). The equilibrium quantities are $N_1^\star = 0.1$ mol, $N_2^\star = 0.45$ mol, and $N_3^\star = 0.45$ mol.

The following Matlab code solves the problem:

```
thermo_potentials_data

alpha = Treact*(s0 - R*c)
beta = Treact*R
gamma = (V0/Vreact) * (T0/Treact).^c

cvx_begin
    variables N(3) z
```

```
    minimize(alpha'*N - (beta * gamma)'*entr(gamma .* N))
    N == [-2; 1; 1]*z + Ninit
cvx_end

display(N)
```

The following Python code solves the problem:

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

from thermo_potentials_data import *

alpha = [Treact*(s0[j] - R*c[j]) for j in range(3)]
beta = Treact*R
gamma = [(V0/Vreact) * (T0/Treact)**c[j] for j in range(3)]

N = cp.Variable(3)
z = cp.Variable()

constr  = [N[0] == -2*z + Ninit[0]]
constr += [N[1] ==    z + Ninit[1]]
constr += [N[2] ==    z + Ninit[2]]
obj = cp.Minimize(sum([ alpha[j]*N[j] - (beta/gamma[j]) *
                  cp.entr(gamma[j]*N[j]) for j in range(3)]))
p = cp.Problem(obj, constr)
p.solve()

print(N.value)
```

The following Julia code solves the problem:

```
using Convex
using ECOS

include("thermo_potentials_data.jl")

 = Treact*(s0 - R*c)
 = Treact*R
 = (V0/Vreact) * (T0/Treact).^c

N = Variable(3)
z = Variable()
```

```
p = minimize('*N - sum([ (/[j]) * entropy([j]*N[j]) for j in 1:3]))
p.constraints += N == [-2.0; 1.0; 1.0]*z + Ninit
solve!(p, ECOSSolver())

display(N)
```

5. *Controlling a switched linear system via duality.* We consider a discrete-time dynamical system with state $x_t \in \mathbf{R}^n$. The state propagates according to the recursion

$$x_{t+1} = A_t x_t, \quad t = 0, 1, \ldots, T-1,$$

where the matrices A_t are to be chosen from a finite set $\mathcal{A} = \{A^{(1)}, \ldots, A^{(K)}\}$ in order to control the state x_t over a finite time horizon of length T. More formally, the switched-linear control problem is

$$\begin{array}{ll} \text{minimize} & \sum_{t=1}^{T} f(x_t) \\ \text{subject to} & x_{t+1} = A^{(u_t)} x_t, \quad \text{for } t = 0, \ldots, T-1 \end{array}$$

The problem variables are $x_t \in \mathbf{R}^n$, for $t = 1, \ldots, T$, and $u_t \in \{1, \ldots, K\}$, for $t = 0, \ldots, T-1$. We assume the initial state, $x_0 \in \mathbf{R}^n$ is a problem parameter (*i.e.*, is known and fixed). You may assume the function f is convex, though it isn't necessary for this problem.

Note that, to find a feasible point, we take any sequence $u_0, \ldots, u_{T-1} \in \{1, \ldots, K\}$; we then generate a feasible point according to the recursion

$$x_{t+1} = A^{(u_t)} x_t, \quad t = 0, 1, \ldots, T-1.$$

The switched-linear control problem is *not* convex, and is hard to solve globally. Instead, we consider a heuristic based on Lagrange duality.

(a) Find the dual of the switched-linear control problem explicitly in terms of x_0, $A^{(1)}, \ldots, A^{(K)}$, the function f, and its conjugate f^*. Your formulation cannot involve a number of constraints or objective terms that is exponential in K or T. (This includes minimization or maximization with an exponential number of terms.)

(b) Given optimal dual variables $\nu_1^\star, \ldots, \nu_T^\star$ corresponding to the T constraints of the switched-linear control problem, a heuristic to choose u_t is to minimize the Langrangian using these optimal dual variables:

$$(\tilde{u}_0, \ldots, \tilde{u}_{T-1}) \in \operatorname*{argmin}_{u_0, \ldots, u_{T-1} \in \{1, \ldots, K\}} \inf_{x_1, \ldots, x_T} L(x_1, \ldots, x_T, u_0, \ldots, u_{T-1}, \nu_1^\star, \ldots, \nu_T^\star),$$

Given the optimal dual variables, show (explicitly) how to find $\tilde{u}_0, \ldots, \tilde{u}_{T-1}$.

(c) Consider the case $f(x) = (1/2)x^T Q x$. with $Q \in \mathbf{S}^n_{++}$. For the data given in `sw_lin_sys_data.*`, solve the dual problem and report its optimal value $d^\star$, which is a lower bound on $p^\star$. (As a courtesy, we also included $p^\star$ in the data file, so you can check your bound.)

Note: Julia users might want to use the ECOS solver, by including `using ECOS`, and solving by using `solve!(prob, ECOSSolver())`.

(d) Using the same data as is part (c), carry out the heuristic method of part (b) to compute $\tilde{u}_0, \ldots \tilde{u}_{T-1}$. Use these values to generate a feasible point. Report the value of the objective at this feasible point, which is an upper bound on $p^\star$.

Solution.

(a) You actually don't have to do any trickery to get this to work. The domain of the problem is

$$\mathcal{D} = (\mathbf{dom}\, f)^T \times \{1, \ldots, K\}^T$$

(*i.e.*, the Cartesian product of the domain of f T times with $\{1, \ldots, K\}$ T times). We just have to form the Lagrangian (by "dualizing" the T equality constraints), and minimize it over vectors $(x_1, \ldots, x_T, u_0, \ldots, u_{T-1})$ in $\mathcal{D}$.

We consider the dual variables $\nu_1, \ldots \nu_T$ for the T equality constraints. The dual function is

$$g(\nu_1, \ldots, \nu_T) = \inf \left(\sum_{t=1}^{T} f(x_t) + \nu_t^T \left(x_t - A^{(u_{t-1})} x_{t-1} \right) \right)$$

where the infimum is taken over $x_1, \ldots, x_T \in \mathbf{dom}\, f$, and $u_0, \ldots, u_{T-1} \in \{1, \ldots K\}$. We can rearrange the terms in the sum to be

$$\inf \left(f(x_T) + \nu_T^T x_T - \nu_1^T A^{(u_0)} x_0 + \sum_{t=1}^{T-1} f(x_t) + \left(\nu_t - A^{(u_t)T} \nu_{t+1} \right)^T x_t \right).$$

By minimizing over each x_t, for $t = 1 \ldots, T$, we obtain

$$\inf \left(-x_0^T A^{(u_0)T} \nu_1 - f^*(-\nu_T) - \sum_{t=1}^{T-1} f^* \left(A^{(u_t)T} \nu_{t+1} - \nu_t \right) \right),$$

where now the infimum is over $u_0, \ldots, u_{T-1} \in \{1, \ldots K\}$. Because the infimum is in fact taken over a finite set, and each u_t only affects one term in the sum, we can rewrite this as

$$-f^*(-\nu_T) - \max_{k \in \{1,\ldots,K\}} x_0^T A^{(k)T} \nu_1 - \sum_{t=1}^{T-1} \max_{k \in \{1,\ldots,K\}} f^* \left(A^{(k)T} \nu_{t+1} - \nu_t \right).$$

Note that this function is concave (as all dual functions are), because the increasing, convex function max is composed with the convex function f^*, preserving convexity (and thus concavity of g).

The dual problem is therefore:

$$\begin{array}{ll} \text{maximize} & -f^*(-\nu_T) - \mathbf{1}^T s \\ \text{subject to} & x_0^T A^{(k)T} \nu_1 \leq s_0 \\ & f^* \left(A^{(k)T} \nu_{t+1} - \nu_t \right) \leq s_{t+1} \end{array}$$

where the first constraint holds for $k = 1, \ldots, K$, and the second constraint holds for $t = 0, \ldots, T-1$ and $k = 1, \ldots, K$. The variables are $\nu_1, \ldots, \nu_T \in \mathbf{R}^n$ and $s \in \mathbf{R}^T$.

(b) Following the reasoning that led up to the dual problem, we have

$$\tilde{u}_0 \in \operatorname*{argmax}_{k\in\{1,\ldots,K\}} x_0^T A^{(k)T} \nu_1^\star,$$

and

$$\tilde{u}_t \in \operatorname*{argmax}_{k\in\{1,\ldots,K\}} f^*\left(A^{(k)T}\nu_{t+1}^\star - \nu_t^\star\right).$$

for $t = 1, \ldots, T-1$. With $u_t = \tilde{u}_t$, for $t = 0, \ldots T-1$, we can generate a feasible point $\tilde{x}_1 \ldots \tilde{x}_T$ according to the recursion $\tilde{x}_{t+1} = A^{(u_t)}\tilde{x}_t$, for $t = 0, \ldots, T-1$, with $\tilde{x}_0 = x_0$.

(c) The dual was derived in part (a). The only thing left to do is find the conjugate of f, which is $f^*(x) = (1/2)x^T Q^{-1} x$. For the data given, $d^\star = 5069$.

(d) For the data given, there are actually multiple values you can get here, because there are multiple minimizers of L over $u_0, \ldots, u_{t-1}$. (And as far as we know, getting the best one requires trying all of them.) The one we got was 52413. If you got this far (few did), then you got credit for any reasonable bound you produced.

The following Matlab code solves the problem:

```
sw_lin_ctrl_data
fstar = @(x) quad_form(x, inv(Q))/2;

% DUAL PROBLEM
cvx_begin
    variables nu(n,T) s(T)
    maximize(-fstar(nu(:,T)) - sum(s))
    for k = 1:K
        x0'*A{k}'*nu(:,1) <= s(1)
        for t = 1:T-1
            fstar(A{k}'*nu(:,t+1) - nu(:,t)) <= s(t+1)
        end
    end
cvx_end

% USE HEURISTIC TO FIND UTILDE
u = zeros(T,1);
val = -inf;
for k = 1:K
    max_argument = x0'*A{k}'*nu(:,1);
    if val <= max_argument
        u(1) = k;
        val = max_argument;
```

```
    end
end

for t = 1:T-1
    val = -inf;
    for k = 1:K
        max_argument = fstar(A{k}'*nu(:,t+1) - nu(:,t));
        if val <= max_argument
            u(t+1) = k;
            val = max_argument;
        end
    end
end

% USE RECURSION TO FIND XTILDE
x = repmat(A{u(1)}*x0,1,T);
obj = x(:,1)'*Q*x(:,1)/2;
for t = 1:T-1
    x(:,t+1) = A{u(t+1)}*x(:,t);
    obj = obj + f(x(:,t+1));
end
lower_bound = cvx_optval
opt_value = pstar
upper_bound = obj
```

The following Python code solves the problem:

```
import numpy as np
import numpy.matlib as ml
import cvxpy as cp

from sw_lin_ctrl_data import *

def f(x):
    return cp.quad_form(x, Q)/2
def fstar(x):
    return cp.quad_form(x, Q.I)/2

#% DUAL PROBLEM
nu = cp.Variable(n,T)
s = cp.Variable(T)
constr = []
for k in range(K):
    constr += [x0.T@A[k].T*nu[:,0] <= s[0]]
```

```
    for t in range(T-1):
        constr += [fstar(A[k].T*nu[:,t+1] - nu[:,t]) <= s[t+1]]
p = cp.Problem(cp.Maximize(-fstar(nu[:,T-1]) - sum(s)), constr)
p.solve()

# USE HEURISTIC TO FIND UTILDE
u = []
u.append( np.argmax([(x0.T@A[k].T@nu.value[:,0])[0,0]
                     for k in range(K)]) )
for t in range(T-1):
    print([fstar(A[k].T*nu[:,t+1] - nu[:,t]).value for k in range(K)])
    u.append( np.argmax([fstar(A[k].T*nu[:,t+1]
                         - nu[:,t]).value for k in range(K)]) )

## USE RECURSION TO FIND XTILDE
x = []
x.append( A[u[1]]*x0 )
for t in range(T-1):
    x.append( A[u[t+1]]*x[t] )
obj = sum([f(x[t]).value for t in range(T)])

print(p.value)
print(pstar)
print(obj)
```

The following Julia code solves the problem:

```
using Convex
using ECOS

include("sw_lin_ctrl_data.jl")
fstar(x) = quad_form(x, inv(Q) + inv(Q)')/4
fstar(x::Vector{Float64}) = (x'/Q*x)[1]/2

#% DUAL PROBLEM
nu = Variable(n,T)
s = Variable(T)
p = maximize(-fstar(nu[:,T]) - sum(s))
for k = 1:K
    p.constraints += x0'*A[k]'*nu[:,1] <= s[1]
    for t = 1:T-1
        p.constraints += fstar(A[k]'*nu[:,t+1] - nu[:,t]) <= s[t+1]
    end
end
```

```
solve!(p, ECOSSolver())

# USE HEURISTIC TO FIND UTILDE
u = zeros(T);
u[1] = findmax([(x0'*A[k]'*nu.value[:,1])[1] for k in 1:K])[2]
for t = 1:T-1
    u[t+1] = findmax([fstar(A[k]'*nu.value[:,t+1]
                        - nu.value[:,t]) for k in 1:K])[2]
end

# USE RECURSION TO FIND XTILDE
x = cell(T)
x[1] = A[u[1]]*x0
for t = 1:T-1
    x[t+1] = A[u[t+1]]*x[t]
end
obj = sum([f(x[t]) for t in 1:T])

display(p.optval)
display(pstar)
display(obj)
```

6. *Elastic stored energy in a spring.* A spring is a mechanical device that exerts a force F that depends on its extension x: $F = \phi(x)$, where $\phi : \mathbf{R} \to \mathbf{R}$. The domain $\mathbf{dom}\,\phi$ is an interval $[x^{\min}, x^{\max}]$ containing 0, where $x^{\min}$ ($x^{\max}$) is the minimum (maximum) possible extension of the spring. When $x > 0$, the spring is said to be extended, and when $x < 0$, it is said to be in compression. The force exerted by the spring must be *restoring*, which means that $F \geq 0$ when $x \geq 0$, and $F \leq 0$ when $x \leq 0$. (Our sign convention is that a positive force F opposes a positive extension x.) This implies that $F = 0$ when $x = 0$, *i.e.*, zero force is developed when the spring is not extended or compressed.

   The simplest spring is a Hooke (linear) spring, with $\phi(x) = Kx$, where $K > 0$ is the *spring constant*. (The constant $1/K$ is called the spring *compliance*.)

   A spring is called *monotonic* if the function ϕ is nondecreasing, *i.e.*, larger extension leads to a stronger restoring force. Many, but not all, springs are monotonic. A classic example is a compound bow, which has a force that first increases with x, and then decreases to a small value at the extension x where it is fully drawn. (This decrease in force from the maximum is called the *let off* of the bow.)

   The elastic stored energy in the spring is

$$E(x) = \int_0^x \phi(x)\, dx,$$

   with domain $[x^{\min}, x^{\max}]$.

   Show that E is quasi-convex. Show that E is convex if and only if the spring is monotonic. You may assume ϕ is differentiable.

Solution.

(a) First note that ϕ is nonnegative, and thus E is nondecreasing, over the interval $[0, x^{\max}]$. Similarly, ϕ is nonpositive, and thus E is nonincreasing, over the interval $[x^{\min}, 0]$. So for $x \leq 0$, E is nonincreasing, and for $x \geq 0$, E is nondecreasing, which means E is quasi-convex. (This is the third bullet of page 99 of the text.) This is all you had to say to get full credit.

If you didn't remember this fact, then you could say that, because the restriction of E to $[0, x^{\max}]$ is nondecreasing, its α-sublevel set is an interval $[0, \beta]$. Similarly, because the restriction of E to $[x^{\min}, 0]$ is nonincreasing, its sublevel set is an interval $[\gamma, 0]$. The α-sublevel set of E is the union of these intervals, which is $[\gamma, \beta]$, a convex set.

You could have also use the (modified) Jensen's inequality. Take $z = \theta x + (1-\theta)y$. Then:

- If $y \geq x \geq 0$, then because E is nondecreasing over nonnegative numbers, $E(z) \leq \max(E(y), E(x)) = E(y)$.
- If $y \leq x \leq 0$, then because E is nonincreasing over nonpositive numbers, $E(z) \leq \max(E(y), E(x)) = E(y)$.
- If $y \leq 0 \leq x$ and $z \leq 0$, then z is also a convex combination of y and 0, so $E(z) \leq \max(E(y), 0) = E(y)$.
- If $y \leq 0 \leq x$ and $z \geq 0$, then z is also a convex combination of x and 0, so $E(z) \leq \max(E(x), 0) = E(x)$.

Unfortunately, second-order conditions for quasiconvexity don't help us here.

A shocking number of people claimed that E was nondecreasing, which is false. (A counter example is the linear (Hooke) spring, for which $E(x) = (1/2)Kx^2$.) This is an easy mistake to make, but it also revealed to us that you didn't really understand what E looks like at all. (Also, we allowed for a lot of mathematically sloppy language on this problem, but if you said something like "E is increasing in both directions", or "E is increasing around 0," you were heavily penalized, because these statements are undeniably false.)

(b) ϕ is nondecreasing if and only if ϕ' is nonnegative. Because $\phi'(x) = E''(x)$, this is equivalent to convexity.

7. *Recovering latent periodic signals.* First, a definition: a signal $x \in \mathbf{R}^n$ is *p-periodic* with $p < n$ if $x_{i+p} = x_i$ for $i = 1, \ldots, n-p$.

   In this problem, we consider a noisy, measured signal $y \in \mathbf{R}^n$ which is (approximately) the sum of a several periodic signals, with unknown periods. Given only the noisy signal y, our task is to recover these latent periodic signals. In particular, y is given as

$$y = v + \sum_{p \in \mathcal{P}} x^{(p)},$$

   where $v \in \mathbf{R}^n$ is a (small) random noise term, and $x^{(p)}$ is a p-periodic signal. The set $\mathcal{P} \subset \{1, \ldots, p_{\max}\}$ contains the periods of the latent periodic signals that compose y.

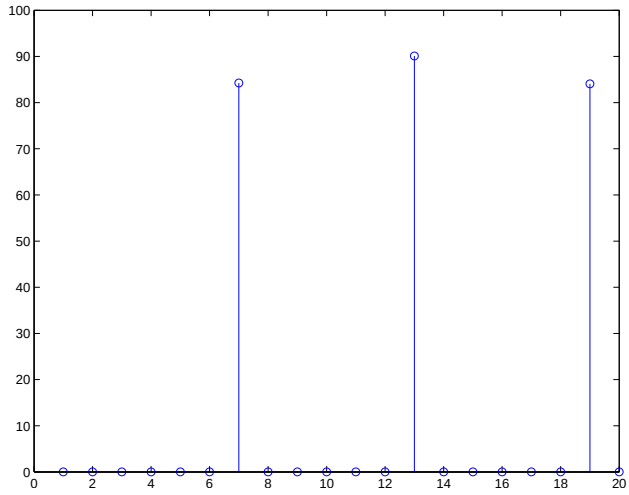
   If $\mathcal{P}$ were known, we could approximately recover the latent periodic signals $x^{(p)}$ using, say, least squares. Because $\mathcal{P}$ is *not* known, we instead propose to recover the latent periodic signals $x^{(p)}$ by solving the following optimization problem:

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \sum_{p=1}^{p_{\max}} w_p \|\hat{x}^{(p)}\|_2 \\ \text{subject to} & \hat{y} = \sum_{p=1}^{p_{\max}} \hat{x}^{(p)} \\ & \hat{x}^{(p)} \text{ is } p\text{-periodic, for } p = 1, \ldots, p_{\max}. \end{array}$$

   The variables are $\hat{y}$ and $\hat{x}^{(p)}$, for $p = 1, \ldots, p_{\max}$. The first sum in the objective penalizes the squared deviation of the measured signal y from our estimate $\hat{y}$, and the second sum is a heuristic for producing vectors $\hat{x}^{(p)}$ that contain only zeros. The weight vector $w \succeq 0$ is increasing in its indices, which encodes our desire that the latent periodic signals have small period.

   (a) Explain how to solve the given optimization problem using convex optimization, and how to use it to (approximately) recover the set $\mathcal{P}$ and the latent periodic signals $x^{(p)}$, for $p \in \mathcal{P}$.

   (b) The file `periodic_signals_data.*` contains a signal y, as well as a weight vector w. Return your best guess of the set $\mathcal{P}$. plot the measured signal y, as well as the different periodic components that (approximately) compose it. (Use separate graphs for each signal, so you should have $|\mathcal{P}| + 1$ graphs.)



Solution.

- The problem is convex as stated; the only difficulty is expressing the constraint

$$x^{(p)} \text{ is } p\text{-periodic}$$

in a CVX-friendly way. We choose a free-parameter representation: We define a matrix $S^{(p)} \in \mathbf{R}^{n\times p}$ such that the i-th row of $S^{(p)}$ is $e_{(p+1) \bmod (n+1)}$. (Informally, the matrix $S^{(p)}$ contains several vertically stacked $p \times p$ identity matrices, with the last one (possibly) truncated.)

The vectors $\hat{x}^{(i)}$ that are nonzero (or simply those with large norm) are our candidates for the latent periodic signals.

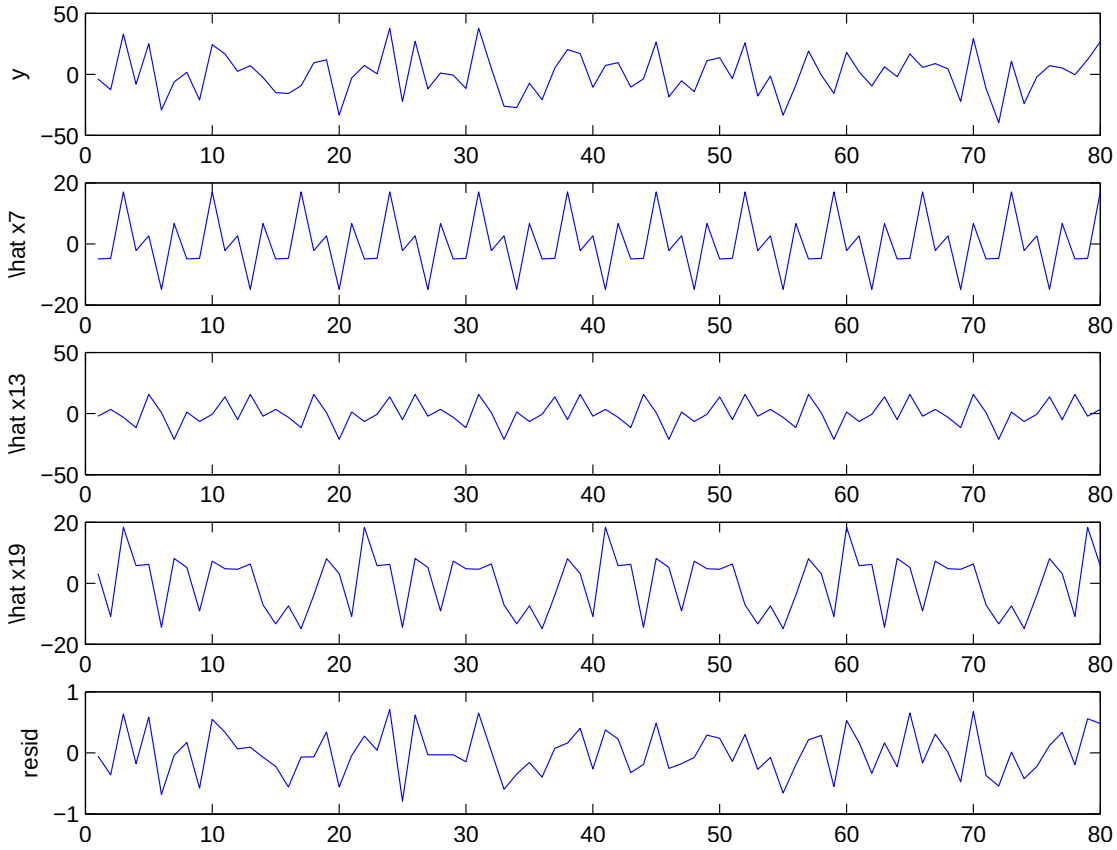
- We can pick out the periods by plotting $\|\hat{x}^{(p)}\|$ as a function of p, given here. The set $\mathcal{P}$ is clearly $\{7, 13, 19\}$.

We also give the requested plot of the measured signal y, and the latent periodic signals $\hat{x}^{(p)}$. We also show the (small) residual, which strongly implies that we have identified the latent periodic signals correctly.

The following Matlab code solves the problem:

```
periodic_signal_data

S = cell(pmax,1);
for p = 1:pmax
    S{p} = kron(ones(n,1), eye(p));
    S{p} = S{p}(1:n,:);
```


y
\hat x7
\hat x13
\hat x19
resid

```
end

cvx_begin
    variables yhat(n) xhat(n,pmax) z(n,pmax)
    minimize(sum_square(y - yhat) + norms(xhat)*w)
    yhat == sum(xhat, 2)
    for p = 1:pmax
        xhat(:,p) == S{p}*z(1:p,p)
    end
cvx_end

close all
figure
stem(norms(xhat))
xlabel('p')
ylabel('norm of x^{p}')
print -depsc periodic_signal_norms.eps

figure
subplot(511)
plot(y)
ylabel('y')
subplot(512)
plot(xhat(:,7))
ylabel('\hat x7')
subplot(513)
plot(xhat(:,13))
ylabel('\hat x13')
subplot(514)
plot(xhat(:,19))
ylabel('\hat x19')
subplot(515)
plot(y - xhat(:,7) - xhat(:,13) - xhat(:,19))
ylabel('resid')
print -depsc periodic_signal.eps
```

The following Python code solves the problem:

```
import numpy as np
import numpy.matlib as ml
import cvxpy as cp
import matplotlib.pyplot as ppt

from periodic_signal_data import *
```

```
S = []
for p in range(pmax):
    Sp = ml.kron(ml.ones(n), ml.eye(p+1)).T
    Sp = Sp[:n,:]
    S.append(Sp)

yhat = cp.Variable(n)
xhat = [cp.Variable(n) for p in range(pmax)]

constr = [yhat == sum(xhat)]
constr += [xhat[p] == S[p]*cp.Variable(p+1) for p in range(pmax)]

obj = cp.sum_squares( yhat - y )
obj += sum([w[p] * cp.norm(xhat[p]) for p in range(pmax)])

prob = cp.Problem(cp.Minimize(obj), constr)
prob.solve()

ppt.close("all")
ppt.figure()
ppt.plot([la.norm(xhat[p].value) for p in range(pmax)])
ppt.xlabel("period")
ppt.ylabel("norm of xhat")
ppt.show()

ppt.figure()
ppt.subplot(511)
ppt.plot(y)
ppt.xlabel("y")
ppt.subplot(512)
ppt.plot(xhat[7].value)
ppt.xlabel("x7")
ppt.subplot(513)
ppt.plot(xhat[13].value)
ppt.xlabel("x13")
ppt.subplot(514)
ppt.plot(xhat[19].value)
ppt.xlabel("x19")
ppt.subplot(515)
ppt.plot(y - xhat[6].value - xhat[12].value - xhat[18].value)
ppt.xlabel("resid")
```

```
ppt.show()
```

The following Julia code solves the problem:

```
using Convex
using PyPlot

include("periodic_signal_data.jl")

S = cell(pmax)
for p in 1:pmax
    S[p] = kron(ones(n,1), eye(p))
    S[p] = S[p][1:n,:]
end

yhat = Variable(n)
xhat = [Variable(n) for p in 1:pmax]

constr = [yhat == sum(xhat)]
append!(constr, [xhat[p] == S[p]*Variable(p) for p in 1:pmax])

obj = sumsquares( yhat - y )
obj += sum([w[p] * norm(xhat[p]) for p in 1:pmax])

prob = minimize(obj, constr)
solve!(prob)

close("all")
figure()
plot(Float64[norm(xhat[p].value) for p in 1:pmax])
xlabel("period")
ylabel("norm of xhat")

figure()
subplot(511)
plot(y)
xlabel("y")
subplot(512)
plot(xhat[7].value)
xlabel("x7")
subplot(513)
plot(xhat[13].value)
xlabel("x13")
subplot(514)
```

```
plot(xhat[19].value)
xlabel("x19")
subplot(515)
plot(y - xhat[7].value - xhat[13].value - xhat[19].value)
xlabel("resid")
```