EE364 Convex Optimization Prof. S. Boyd
June 7 – 8 or June 8 – 9, 2006.

Final exam solutions

1. *Optimizing processor speed.* A set of n tasks is to be completed by n processors. The variables to be chosen are the processor speeds $s_1, \ldots, s_n$, which must lie between a given minimum value $s_{\min}$ and a maximum value $s_{\max}$. The computational load of task i is α_i, so the time required to complete task i is $\tau_i = \alpha_i/s_i$.

   The power consumed by processor i is given by $p_i = f(s_i)$, where $f : \mathbf{R} \to \mathbf{R}$ is positive, increasing, and convex. Therefore, the total energy consumed is

   $$E = \sum_{i=1}^{n} \frac{\alpha_i}{s_i} f(s_i).$$

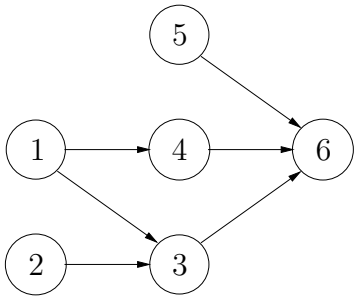
   (Here we ignore the energy used to transfer data between processors, and assume the processors are powered down when they are not active.)

   There is a set of *precedence constraints* for the tasks, which is a set of m ordered pairs $\mathcal{P} \subseteq \{1, \ldots, n\} \times \{1, \ldots, n\}$. If $(i, j) \in \mathcal{P}$, then task j cannot start until task i finishes. (This would be the case, for example, if task j requires data that is computed in task i.) When $(i, j) \in \mathcal{P}$, we refer to task i as a *precedent* of task j, since it must precede task j. We assume that the precedence constraints define a directed acyclic graph (DAG), with an edge from i to j if $(i, j) \in \mathcal{P}$.

   If a task has no precedents, then it starts at time $t = 0$. Otherwise, each task starts as soon as all of its precedents have finished. We let T denote the time for all tasks to be completed.

   To be sure the precedence constraints are clear, we consider the very small example shown below, with $n = 6$ tasks and $m = 6$ precedence constraints.

   $$\mathcal{P} = \{(1,4),\ (1,3),\ (2,3),\ (3,6),\ (4,6),\ (5,6)\}.$$



In this example, tasks 1, 2, and 5 start at time $t = 0$ (since they have no precedents). Task 1 finishes at $t = \tau_1$, task 2 finishes at $t = \tau_2$, and task 5 finishes at $t = \tau_5$. Task 3 has tasks 1 and 2 as precedents, so it starts at time $t = \max\{\tau_1, \tau_2\}$, and ends τ_3 seconds later, at $t = \max\{\tau_1, \tau_2\} + \tau_3$. Task 4 completes at time $t = \tau_1 + \tau_4$. Task 6 starts when tasks 3, 4, and 5 have finished, at time $t = \max\{\max\{\tau_1, \tau_2\} + \tau_3, \tau_1 + \tau_4, \tau_5\}$. It finishes τ_6 seconds later. In this example, task 6 is the last task to be completed, so we have

$$T = \max\{\max\{\tau_1, \tau_2\} + \tau_3, \tau_1 + \tau_4, \tau_5\} + \tau_6.$$

(a) Formulate the problem of choosing processor speeds (between the given limits) to minimize completion time T, subject to an energy limit $E \leq E_{\max}$, as a convex optimization problem. The data in this problem are $\mathcal{P}$, $s_{\min}$, $s_{\max}$, $\alpha_1, \ldots, \alpha_n$, $E_{\max}$, and the function f. The variables are $s_1, \ldots, s_n$.

Feel free to change variables or to introduce new variables. Be sure to explain clearly why your formulation of the problem is convex, and why it is equivalent to the problem statement above.

Important:

- Your formulation must be convex for any function f that is positive, increasing, and convex. You cannot make any further assumptions about f.
- This problem refers to the general case, not the small example described above.

(b) Consider the specific instance with data given in `ps_data.m`, and processor power

$$f(s) = 1 + s + s^2 + s^3.$$

The precedence constraints are given by an $m \times 2$ matrix `prec`, where m is the number of precedence constraints, with each row giving one precedence constraint (the first column gives the precedents).

Plot the optimal trade-off curve of energy E versus time T, over a range of T that extends from its minimum to its maximum possible value. (These occur when all processors operate at $s_{\max}$ and $s_{\min}$, respectively, since T is monotone nonincreasing in s.) On the same plot, show the energy-time trade-off obtained when all processors operate at the same speed $\bar{s}$, which is varied from $s_{\min}$ to $s_{\max}$.

Note: In this part of the problem there is no limit $E^{\max}$ on E as in part (a); you are to find the optimal trade-off of E versus T.

Solution.

(a) First let's look at the energy E. In general it is *not* a convex function of s. For example consider $f(s) = s^{1.5}$, which is increasing and convex. But $(1/s)f(s) = \sqrt{s}$, which is not convex. So we're going to need to reformulate the problem somehow.

We introduce the variable $\tau \in \mathbf{R}^n$, defined as

$$\tau_i = \alpha_i / s_i.$$

The variable τ_i is the time required to complete task i. We can recover s_i from τ_i as $s_i = \alpha_i/\tau_i$. We'll use τ_i instead of s_i.

The energy E, as a function of τ, is

$$E = \sum_{i=1}^{n} \tau_i f(\alpha_i/\tau_i).$$

This is a convex function of τ, since each term is the perspective of f, $yf(x/y)$, evaluated at $y = \tau_i$ and $x = \alpha_i$. (This shows that E is jointly convex in τ and α, but we take α constant here.)

The processor speed limits $s_{\min} \leq s_i \leq s_{\max}$ are equivalent to

$$\alpha_i/s_{\max} \leq \tau_i \leq \alpha_i/s_{\min}, \quad i = 1, \ldots, n.$$

Now let's look at the precedence constraints. To tackle these, we introduce the variable $t \in \mathbf{R}^n$, where t_i is an upper bound on the completion time of task i. Thus, we have

$$T \leq \max_i t_i.$$

Task i cannot start before all its precedents have finished; after that, it takes at least τ_i more time. Thus, we have

$$t_j \geq t_i + \tau_j, \quad (i,j) \in \mathcal{P}.$$

Tasks that have no precedent must satisfy $t_i \geq \tau_i$. In fact, this holds for all tasks, so we have

$$t_i \geq \tau_i, \quad i = 1, \ldots, n.$$

We formulate the problem as

$$\begin{array}{lll} \text{minimize} & \max_i t_i & \\ \text{subject to} & \sum_{i=1}^n \tau_i f(\alpha_i/\tau_i) \leq E_{\max} & \\ & \alpha_i/s_{\max} \leq \tau_i \leq \alpha_i/s_{\min}, & i = 1, \ldots, n \\ & t_i \geq \tau_i, \quad i = 1, \ldots, n & \\ & t_j \geq t_i + \tau_j, \quad (i,j) \in \mathcal{P}, & \end{array}$$

with variables t and τ. The energy constraint is convex, and the other constraints are linear. The objective is convex.

(b) For this particular problem, we have

$$\tau_i f(\alpha_i/\tau_i) = \tau_i + \alpha_i + \alpha_i^2/\tau_i + \alpha_i^3/\tau_i^2.$$

To generate the optimal tradeoff curve we scalarize, and minimize $T + \lambda E$ for λ varying over some range that gives us the full range of T. Thus, we solve the problem

$$\begin{array}{ll} \text{minimize} & \max_i t_i + \lambda \sum_{i=1}^n (\tau_i + \alpha_i + \alpha_i^2/\tau_i + \alpha_i^3/\tau_i^2) \\ \text{subject to} & \alpha_i/s_{\max} \leq \tau_i \leq \alpha_i/s_{\min}, \quad i = 1, \ldots, n \\ & t_i \geq \tau_i, \quad i = 1, \ldots, n \\ & t_j \geq t_i + \tau_j, \quad (i,j) \in \mathcal{P}, \end{array}$$

for λ taking a values in some range.

If we constrain all processors to have the same speed $\bar{s}$, we are in effect adding the constraint $\tau = (1/\bar{s})\alpha$. In this case we can find the time required to complete all processes by solving the problem

$$\begin{array}{ll} \text{minimize} & \max_i t_i \\ \text{subject to} & t_i \geq \alpha_i/\bar{s}, \quad i = 1, \ldots, n \\ & t_j \geq t_i + \alpha_j/\bar{s}, \quad (i,j) \in \mathcal{P}. \end{array}$$

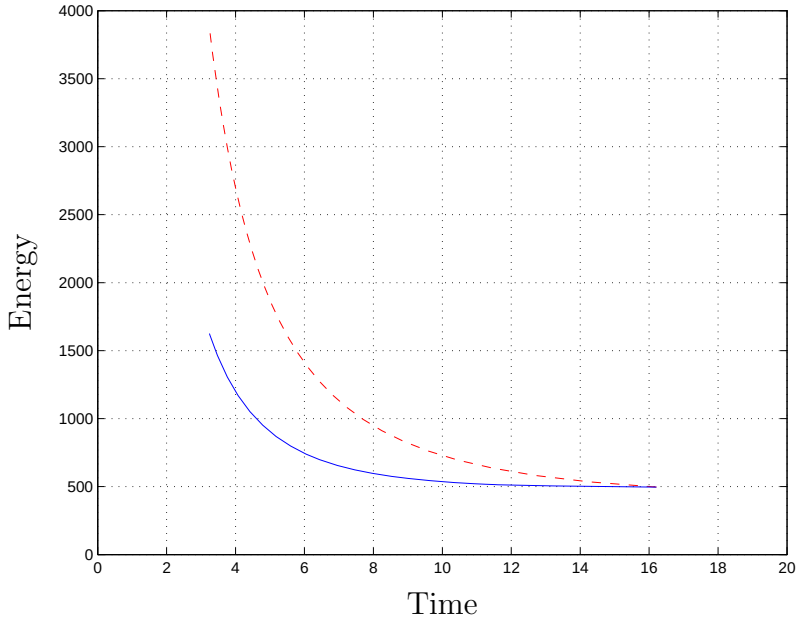
(We don't really need to solve an optimization problem here; but it's easier to solve it than to write the code to evaluate T.) To generate the tradeoff curve for the case when all processors are running at the same speed, we solve the problem above for $\bar{s}$ ranging between $s_{\min} = 1$ and $s_{\max} = 5$. This gives us the full range of possible values of T: when $\bar{s} = s_{\max}$ we find $T = 3.243$; when $\bar{s} = s_{\min}$ we find $T = 16.212$.

The following matlab code was used to plot the two tradeoff curves:

```
cvx_quiet(true);
ps_data

% Optimal power-time tradeoff curve
Eopt = []; Topt = [];
fprintf(1,'Optimal tradeoff curve\n')
for lambda = logspace(0,-3,30);
   fprintf(1,'Solving for lambda = %1.3f\n',lambda);
   cvx_begin
       variables t(n) tau(n)
       E = sum(tau+alpha+alpha.^2.*inv_pos(tau)+...
           alpha.^3.*square_pos(inv_pos(tau)));
       minimize(lambda*E+max(t))
       subject to
           t(prec(:,2)) >= t(prec(:,1))+tau(prec(:,2))
           t >= tau
           tau >= alpha/s_max
           tau <= alpha/s_min
```

```
   cvx_end
   E = sum(tau+alpha+alpha.^2./tau+alpha.^3./(tau.^2));
   T = max(t);
   Eopt = [Eopt E];
   Topt = [Topt T];
end

% Tradeoff-curve for constant speed
fprintf(1,'\nConstant speed tradeoff curve\n')
Econst = []; Tconst = [];
for s_const = linspace(s_min,s_max,30);
   fprintf(1,'Solving for s = %1.3f\n',s_const);
   cvx_begin
       variables t(n)
       minimize(max(t))
       subject to
           t(prec(:,2)) >= t(prec(:,1))+alpha(prec(:,2))/s_const
           t >= alpha/s_const
   cvx_end
   E = sum(alpha*(1/s_const+1+s_const+s_const^2));
   T = max(t);
   Econst = [Econst E];
   Tconst = [Tconst T];
end

plot(Tconst,Econst,'r--')
hold on
plot(Topt,Eopt,'b-')
xlabel('Time')
ylabel('Energy')
grid on
axis([0 20 0 4000])
print -depsc processor_speed.eps
```

The two tradeoff curves are shown in the following plot. The solid line corresponds to the optimal tradeoff curve, while the dotted line corresponds to the tradeoff curve with constant processor speed.



We see that the optimal processor speeds use significantly less energy than when all processors have the same speed, adjusted to give the same T, especially when T is small.

We note that this particular problem can be solved without using the formulation given in part (a). For this particular power function we can actually use s as the optimization variable; we don't need to change coordinates to τ. This is because E is a convex function of s; it has the form

$$E = \sum_{i=1}^{n} \alpha_i \left(1/s_i + 1 + s_i + s_i^2\right).$$

To get the tradeoff curve we can solve the problem

$$\begin{array}{ll} \text{minimize} & \max_i t_i + \lambda \sum_{i=1}^n \alpha_i \left(1/s_i + 1 + s_i + s_i^2\right) \\ \text{subject to} & s_{\min} \leq s_i \leq s_{\max}, \quad i = 1, \ldots, n \\ & t_i \geq \alpha_i / s_i, \quad i = 1, \ldots, n \\ & t_j \geq t_i + \alpha_j / s_j, \quad (i,j) \in \mathcal{P}, \end{array}$$

with variables t and s, for a range of positive values of λ.

```
cvx_begin
    variables s(n) t(n)
    E = alpha'*(inv_pos(s)+1+s+square_pos(s));
    minimize(lambda*E+max(t))
    subject to
        t(prec(:,2)) >= t(prec(:,1))+alpha(prec(:,2)).*...
               inv_pos(s(prec(:,2)))
        t >= alpha.*inv_pos(s)
```

```
        s >= s_max
        s <= s_min
cvx_end
```

Finally, we note that this specific problem can also be cast as a GP, since E is a posynomial function of the speeds, and all the constraints can be written as posynomial inequalities.

2. *Exploring nearly optimal points.* An optimization algorithm will find *an* optimal point for a problem, provided the problem is feasible. It is often useful to explore the set of nearly optimal points. When a problem has a 'strong minimum', the set of nearly optimal points is small; all such points are close to the original optimal point found. At the other extreme, a problem can have a 'soft minimum', which means that there are many points, some quite far from the original optimal point found, that are feasible and have nearly optimal objective value. In this problem you will use a typical method to explore the set of nearly optimal points.

   We start by finding the optimal value $p^\star$ of the given problem

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

   as well as an optimal point $x^\star \in \mathbf{R}^n$. We then pick a small positive number ϵ, and a vector $c \in \mathbf{R}^n$, and solve the problem

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & f_i(x) \leq 0, \quad i = 1, \ldots, m \\ & h_i(x) = 0, \quad i = 1, \ldots, p \\ & f_0(x) \leq p^\star + \epsilon. \end{array}$$

   Note that any feasible point for this problem is ϵ-suboptimal for the original problem. Solving this problem multiple times, with different c's, will generate (perhaps different) ϵ-suboptimal points. If the problem has a strong minimum, these points will all be close to each other; if the problem has a weak minimum, they can be quite different.

   There are different strategies for choosing c in these experiments. The simplest is to choose the c's randomly; another method is to choose c to have the form $\pm e_i$, for $i = 1, \ldots, n$. (This method gives the 'range' of each component of x, over the ϵ-suboptimal set.)

   You will carry out this method for the following problem, to determine whether it has a strong minimum or a weak minimum. You can generate the vectors c randomly, with enough samples for you to come to your conclusion. You can pick $\epsilon = 0.01p^\star$, which means that we are considering the set of 1% suboptimal points.

   The problem is a minimum fuel optimal control problem for a vehicle moving in $\mathbf{R}^2$. The position at time kh is given by $p(k) \in \mathbf{R}^2$, and the velocity by $v(k) \in \mathbf{R}^2$, for $k = 1, \ldots, K$. Here $h > 0$ is the sampling period. These are related by the equations

$$p(k+1) = p(k) + hv(k), \quad v(k+1) = (1-\alpha)v(k) + (h/m)f(k), \quad k = 1, \ldots, K-1,$$

   where $f(k) \in \mathbf{R}^2$ is the force applied to the vehicle at time kh, $m > 0$ is the vehicle mass, and $\alpha \in (0, 1)$ models drag on the vehicle; in the absense of any other force, the vehicle velocity decreases by the factor $1 - \alpha$ in each discretized time interval.

(These formulas are approximations of more accurate formulas that involve matrix exponentials.)

The force comes from two thrusters, and from gravity:

$$f(k) = \begin{bmatrix} \cos\theta_1 \\ \sin\theta_1 \end{bmatrix} u_1(k) + \begin{bmatrix} \cos\theta_2 \\ \sin\theta_2 \end{bmatrix} u_2(k) + \begin{bmatrix} 0 \\ -mg \end{bmatrix}, \quad k = 1, \ldots, K-1.$$

Here $u_1(k) \in \mathbf{R}$ and $u_2(k) \in \mathbf{R}$ are the (nonnegative) thruster force magnitudes, θ_1 and θ_2 are the directions of the thrust forces, and $g = 10$ is the constant acceleration due to gravity.

The total fuel use is

$$F = \sum_{k=1}^{K-1} \left(u_1(k) + u_2(k) \right).$$

(Recall that $u_1(k) \geq 0$, $u_2(k) \geq 0$.)

The problem is to minimize fuel use subject to the initial condition $p(1) = 0$, $v(1) = 0$, and the way-point constraints

$$p(k_i) = w_i, \quad i = 1, \ldots, M.$$

(These state that at the time hk_i, the vehicle must pass through the location $w_i \in \mathbf{R}^2$.) In addition, we require that the vehicle should remain in a square operating region,

$$\|p(k)\|_\infty \leq P^{\max}, \quad k = 1, \ldots, K.$$

Both parts of this problem concern the specific problem instance with data given in `thrusters_data.m`.

(a) Find an optimal trajectory, and the associated minimum fuel use $p^\star$. Plot the trajectory $p(k)$ in $\mathbf{R}^2$ (*i.e.*, in the p_1, p_2 plane). Verify that it passes through the way-points.

(b) Generate several 1% suboptimal trajectories using the general method described above, and plot the associated trajectories in $\mathbf{R}^2$. Would you say this problem has a strong minimum, or a weak minimum?

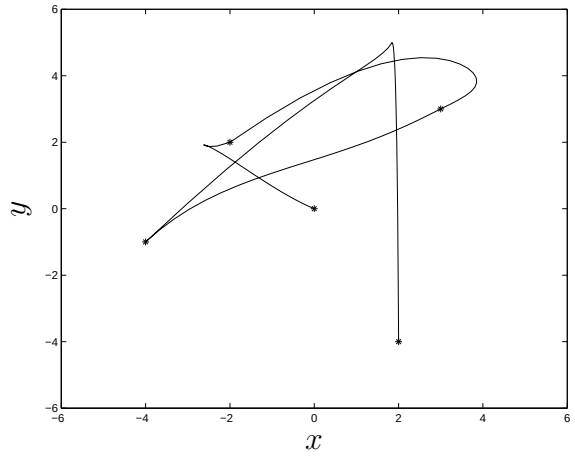
Solution.

(a) The following Matlab script finds the optimal solution.

```
cvx_quiet(true);
thrusters_data;
F = [ cos(theta1) cos(theta2);...
      sin(theta1) sin(theta2)];
```

```
% finding optimal solution
cvx_begin
    variables u(2,K-1) p(2,K) v(2,K)
    minimize ( sum(sum(u)))
    p(:,1) == 0;            % initial position
    v(:,1) == 0;            % initial velocity
    % way-point constraints
    p(:,k1) == w1;
    p(:,k2) == w2;
    p(:,k3) == w3;
    p(:,k4) == w4;
    for i=1:K-1
        p(:,i+1) == p(:,i) + h*v(:,i);
        v(:,i+1) == (1-alpha)*v(:,i) + h*F*u(:,i)/m + [0; -g*h];
    end
    u >= 0;
    % constaints on positions (x,y)
    p <= pmax;
    p >= -pmax;
cvx_end

display('The optimal fuel use is: ');
optval = cvx_optval
plot(p(1,:),p(2,:));
hold on
ps = [zeros(2,1) w1 w2 w3 w4];
plot(ps(1,:),ps(2,:),'*');
xlabel('x'); ylabel('y'); title('optimal');
axis([-6 6 -6 6]);
```

This Matlab script generates the following optimal trajectory.

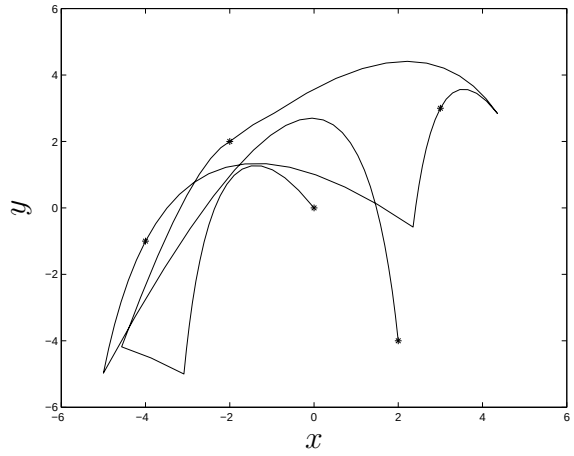


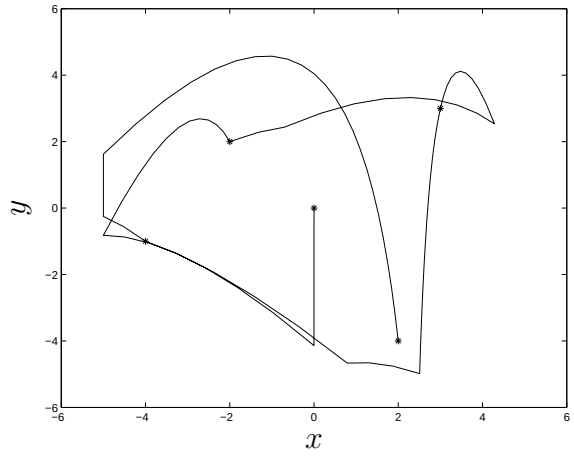
The optimal value fuel use is found to be 1055.3.

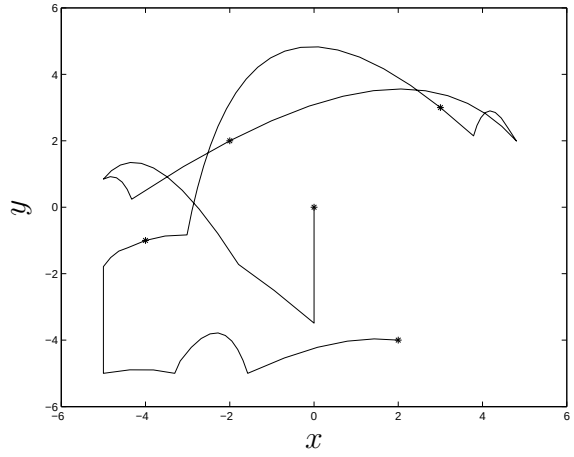
(b) The following script finds 1% suboptimal solutions.

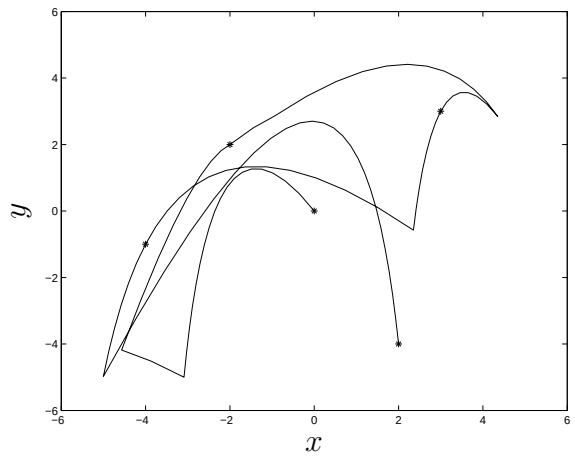
```
% finding nearly optimal solutions
cvx_begin
    variables u(2,K-1) p(2,K) v(2,K)
    minimize ( sum ( sum ( randn(2,K-1).*u ) ) + ...
               sum ( sum ( randn(2,K).*p ) ) + ...
               sum ( sum ( randn(2,K).*v ) ) )
    p(:,1) == 0;            % initial position
    v(:,1) == 0;            % initial velocity
    % way-point constraints
    p(:,k1) == w1;
    p(:,k2) == w2;
    p(:,k3) == w3;
    p(:,k4) == w4;
    for i=1:K-1
        p(:,i+1) == p(:,i) + h*v(:,i);
        v(:,i+1) == (1-alpha)*v(:,i) + F*u(:,i) + [0; -g*h];
    end
    u >= 0;
    sum(sum(u))<=1.01*optval;
    % constaints on positions (x,y)
    p <= pmax;
    p >= -pmax;
cvx_end

figure;
plot(p(1,:),p(2,:));
hold on
ps = [zeros(2,1) w1 w2 w3 w4];
plot(ps(1,:),ps(2,:),'*');
xlabel('x'); ylabel('y'); title('suboptimal');
axis([-6 6 -6 6]);
```

The MATLAB script returns 4 randomly-generated nearly optimal trajectories.


y
x


y
x


y
x



We see that these nearly optimal trajectories are very, very different. So in this problem there is a weak minimum, *i.e.*, a very large 1%-suboptimal set.

3. *Estimating a vector with unknown nonlinear measurement nonlinearity.* We want to estimate a vector $x \in \mathbf{R}^n$, given some measurements

$$y_i = \phi(a_i^T x + v_i), \quad i = 1, \ldots, m.$$

Here $a_i \in \mathbf{R}^n$ are known, v_i are IID $\mathcal{N}(0, \sigma^2)$ random noises, and $\phi : \mathbf{R} \to \mathbf{R}$ is an unknown monotonic increasing function, known to satisfy

$$\alpha \leq \phi'(u) \leq \beta,$$

for all u. (Here α and β are known positive constants, with $\alpha < \beta$.) We want to find a maximum likelihood estimate of x and ϕ, given y_i. (We also know a_i, σ, α, and β.)

This sounds like an infinite-dimensional problem, since one of the parameters we are estimating is a function. In fact, we only need to know the m numbers $z_i = \phi^{-1}(y_i)$, $i = 1, \ldots, m$. So by estimating ϕ we really mean estimating the m numbers $z_1, \ldots, z_m$. (These numbers are not arbitrary; they must be consistent with the prior information $\alpha \leq \phi'(u) \leq \beta$ for all u.)

(a) Explain how to find a maximum likelihood estimate of x and ϕ (*i.e.*, $z_1, \ldots, z_m$) using convex optimization.

(b) Carry out your method on the data given in `nonlin_meas_data.m`, which includes a matrix $A \in \mathbf{R}^{m \times n}$, with rows $a_1^T, \ldots, a_m^T$. Give $\hat{x}_{\mathrm{ml}}$, the maximum likelihood estimate of x. Plot your estimated function $\hat{\phi}_{\mathrm{ml}}$. (You can do this by plotting $(\hat{z}_{\mathrm{ml}})_i$ versus y_i, with y_i on the vertical axis and $(\hat{z}_{\mathrm{ml}})_i$ on the horizontal axis.)

Hint. You can assume the measurements are numbered so that y_i are sorted in nondecreasing order, *i.e.*, $y_1 \leq y_2 \leq \cdots \leq y_m$. (The data given in the problem instance for part (b) is given in this order.)

Solution.

(a) The measurement equations can be written

$$z_i = \phi^{-1}(y_i), \quad i = 1, \ldots, m.$$

The function ϕ^{-1} is unknown (indeed, it is to be estimated), but it has derivative that lies between $1/\beta$ and $1/\alpha$. In terms of the z_i, this means

$$(1/\beta)(y_{i+1} - y_i) \leq z_{i+1} - z_i \leq (1/\alpha)(y_{i+1} - y_i), \quad i = 1, \ldots, m-1,$$

assuming that the data are given with y_i in nondecreasing order.

The log-likelihood function has the form

$$l(z, x) = -(1/\sigma^2) \sum_{i=1}^{m} (z_i - a_i^T x)^2$$

(plus a constant that isn't relevant). Thus to find a maximum likelihood estimate of x and z we solve the problem

$$\begin{array}{ll} \text{maximize} & -(1/\sigma^2)\sum_{i=1}^{m}(z_i - a_i^T x)^2 \\ \text{subject to} & (1/\beta)(y_{i+1}-y_i) \leq z_{i+1}-z_i \leq (1/\alpha)(y_{i+1}-y_i), \quad i = 1,\ldots,m-1, \end{array}$$

with variables $z \in \mathbf{R}^m$ and $x \in \mathbf{R}^n$. This is a QP.

(b) The following Matlab code solve the given problem

```
nonlin_meas_data

row=zeros(1,m);
row(1)=-1;
row(2)=1;
col=zeros(1,m-1);
col(1)=-1;
B=toeplitz(col,row);


cvx_begin
    variable x(n);
    variable z(m);
    minimize(norm(z-A*x));
    subject to
       1/beta*B*y<=B*z;
       B*z<=1/alpha*B*y;
cvx_end

disp('estimated x:'); disp(x);

plot(z,y)
ylabel('y')
xlabel('z')
title('ML estimate of \phi')
```

The estimated x is $x = (0.4819,\ -0.4657,\ 0.9364,\ 0.9297)$. Figure 1 shows the estimated z versus the measured value y.

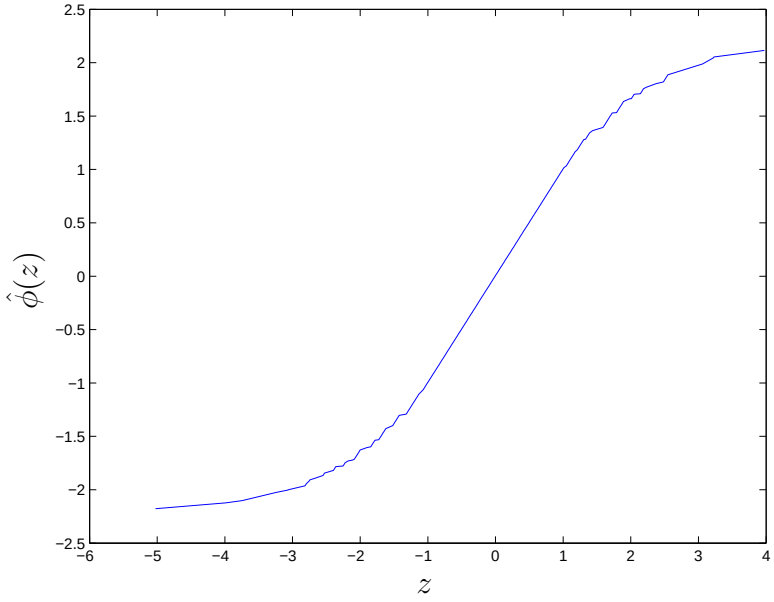


Figure 1: Maximum likelihood estimate of ϕ.

4. *Optimizing rates and time slot fractions.* We consider a wireless system that uses time-domain multiple access (TDMA) to support n communication flows. The flows have (nonnegative) rates $r_1, \ldots, r_n$, given in bits/sec. To support a rate r_i on flow i requires transmitter power

$$p = a_i(e^{br} - 1),$$

where b is a (known) positive constant, and a_i are (known) positive constants related to the noise power and gain of receiver i.

TDMA works like this. Time is divided up into periods of some fixed duration T (seconds). Each of these T-long periods is divided into n time-slots, with durations $t_1, \ldots, t_n$, that must satisfy $t_1 + \cdots + t_n = T$, $t_i \geq 0$. In time-slot i, communications flow i is transmitted at an instantaneous rate $r = Tr_i/t_i$, so that over each T-long period, Tr_i bits from flow i are transmitted. The power required during time-slot i is $a_i(e^{bTr_i/t_i} - 1)$, so the average transmitter power over each T-long period is

$$P = (1/T)\sum_{i=1}^{n} a_i t_i (e^{bTr_i/t_i} - 1).$$

When t_i is zero, we take $P = \infty$ if $r_i > 0$, and $P = 0$ if $r_i = 0$. (The latter corresponds to the case when there is zero flow, and also, zero time allocated to the flow.)

The problem is to find rates $r \in \mathbf{R}^n$ and time-slot durations $t \in \mathbf{R}^n$ that maximize the log utility function

$$U(r) = \sum_{i=1}^{n} \log r_i,$$

subject to $P \leq P^{\max}$. (This utility function is often used to ensure 'fairness'; each communication flow gets at least some positive rate.) The problem data are a_i, b, T and $P^{\max}$; the variables are t_i and r_i.

(a) Formulate this problem as a convex optimization problem. Feel free to introduce new variables, if needed, or to change variables. Be sure to justify convexity of the objective or constraint functions in your formulation.

(b) Give the optimality conditions for your formulation. Of course we prefer simpler optimality conditions to complex ones. *Note:* We do not expect you to *solve* the optimality conditions; you can give them as a set of equations (and possibly inequalities).

Hint. With a log utility function, we cannot have $r_i = 0$, and therefore we cannot have $t_i = 0$; therefore the constraints $r_i \geq 0$ and $t_i \geq 0$ cannot be active or tight. This will allow you to simplify the optimality conditions.

Solution. The problem is

$$\begin{array}{ll} \text{maximize} & \sum_{i=1}^{n} \log r_i \\ \text{subject to} & \mathbf{1}^T t = T \\ & P = (1/T)\sum_{i=1}^{n} a_i t_i (e^{bTr_i/t_i} - 1) \leq P^{\max}, \end{array}$$

with variables $r \in \mathbf{R}^n$ and $t \in \mathbf{R}^n$. There is an implicit constraint that $r_i > 0$, and also that $t_i > 0$.

In fact, we don't need to introduce any new variables, or to change any variables. This is a convex optimization problem just as it stands. The objective is clearly concave, and so can be maximized. The only question is whether or not the function P is convex in r and t. To show this, we need to show that the function $f(x, y) = xe^{x/y}$ is convex in x and y, for $y > 0$. But this is nothing more than the perspective of the exponential function, so it's convex. The function P is just a positive weighted sum of functions of this form (plus an affine function), so it's convex.

We introduce a Lagrange multiplier $\nu \in \mathbf{R}$ for the equality constraint, and $\lambda \in \mathbf{R}_+$ for the inequality constraint. We don't need Lagrange multipliers for the implicit constraints $t \succeq 0$, $r \succeq 0$; even if we did introduce them they'd be zero at the optimum, since these constraints cannot be tight.

The KKT conditions are: primal feasibility,

$$\mathbf{1}^T t = T, \qquad (1/T)\sum_{i=1}^{n} a_i t_i (e^{bTr_i/t_i} - 1) \leq P^{\max},$$

dual feasibility, $\lambda \geq 0$,

$$\frac{\partial L}{\partial r_i} = -1/r_i + \lambda a_i b e^{bTr_i/t_i} = 0, \quad i = 1, \ldots, n,$$

$$\frac{\partial L}{\partial t_i} = \lambda (a_i/T)\left(e^{bTr_i/t_i} - 1 - (bTr_i/t_i)e^{bTr_i/t_i}\right) + \nu = 0, \quad i = 1, \ldots, n,$$

and the complementarity condition $\lambda(P - P^{\max}) = 0$.

In fact, the constraint $P \leq P^{\max}$ must be tight at the optimum, because the utility is monotonic increasing in r, and if the power constraint were slack, we could increase rates slightly, without violating the power limit, and get more utility. In other words, we can replace $P \leq P^{\max}$ with $P = P^{\max}$. This means we can replace the second primal feasibility condition with an equality, and also, we conclude that the complementarity condition always holds.

Thus, the KKT conditions are

$$\begin{array}{l}
\mathbf{1}^T t = T, \\
(1/T)\sum_{i=1}^{n} a_i t_i (e^{bTr_i/t_i} - 1) = P^{\max}, \\
-1/r_i + \lambda a_i b e^{bTr_i/t_i} = 0, \quad i = 1, \ldots, n, \\
(\lambda a_i/T)\left(e^{bTr_i/t_i} - 1 - (bTr_i/t_i)e^{bTr_i/t_i}\right) + \nu = 0, \quad i = 1, \ldots, n, \\
\lambda \geq 0.
\end{array}$$

We didn't ask you to solve these equations. As far as we know, there's no analytical solution. But, after a huge and bloody algebra battle, it's possible to solve the KKT

conditions using a one-parameter search, as in water-filling. Although this appears to be a great solution, it actually has no better computational complexity than a standard method, such as Newton's method, for solving the KKT conditions, provided the special structure in the Newton step equations is exploited properly. Either way, you end up with a method that involves say a few tens of iterations, each one requiring $O(n)$ flops.

Remember, we didn't ask you to solve the KKT equations. And you should be grateful that we didn't, because we certainly could have.

5. *Feature selection and sparse linear separation.* Suppose $x^{(1)}, \ldots, x^{(N)}$ and $y^{(1)}, \ldots, y^{(M)}$ are two given nonempty collections or classes of vectors in $\mathbf{R}^n$ that can be (strictly) separated by a hyperplane, *i.e.*, there exists $a \in \mathbf{R}^n$ and $b \in \mathbf{R}$ such that

$$a^T x^{(i)} - b \geq 1, \quad i = 1, \ldots, N, \qquad a^T y^{(i)} - b \leq -1, \quad i = 1, \ldots, M.$$

This means the two classes are (weakly) separated by the slab

$$S = \{z \mid |a^T z - b| \leq 1\},$$

which has thickness $2/\|a\|_2$. You can think of the components of $x^{(i)}$ and $y^{(i)}$ as *features*; a and b define an affine function that combines the features and allows us to distinguish the two classes.

To find the thickest slab that separates the two classes, we can solve the QP

$$\begin{array}{ll} \text{minimize} & \|a\|_2 \\ \text{subject to} & a^T x_i - b \geq 1, \quad i = 1, \ldots, N \\ & a^T y_i - b \leq -1, \quad i = 1, \ldots, M, \end{array}$$

with variables $a \in \mathbf{R}^n$ and $b \in \mathbf{R}$. (This is equivalent to the problem given in (8.23), p424, §8.6.1; see also exercise 8.23.)

In this problem we seek (a, b) that separate the two classes with a thick slab, and also has a sparse, *i.e.*, there are many j with $a_j = 0$. Note that if $a_j = 0$, the affine function $a^T z - b$ does not depend on z_j, *i.e.*, the jth feature is not used to carry out classification. So a sparse a corresponds to a classification function that is parsimonius; it depends on just a few features. So our goal is to find an affine classification function that gives a thick separating slab, and also uses as few features as possible to carry out the classification.

This is in general a hard combinatorial (bi-criterion) optimization problem, so we use the standard heuristic of solving

$$\begin{array}{ll} \text{minimize} & \|a\|_2 + \lambda \|a\|_1 \\ \text{subject to} & a^T x_i - b \geq 1, \quad i = 1, \ldots, N \\ & a^T y_i - b \leq -1, \quad i = 1, \ldots, M, \end{array}$$

where $\lambda \geq 0$ is a weight vector that controls the trade-off between separating slab thickness and (indirectly, through the ℓ_1 norm) sparsity of a.

Get the data in `sp_ln_sp_data.m`, which gives x_i and y_i as the columns of matrices `X` and `Y`, respectively. Find the thickness of the maximum thickness separating slab. Solve the problem above for 100 or so values of λ over an appropriate range (we recommend log spacing). For each value, record the separation slab thickness $2/\|a\|_2$ and $\mathbf{card}(a)$, the cardinality of a (*i.e.*, the number of nonzero entries). In computing the cardinality, you can count an entry a_j of a as zero if it satisfies $|a_j| \leq 10^{-4}$. Plot these data with slab thickness on the vertical axis and cardinality on the horizontal axis.

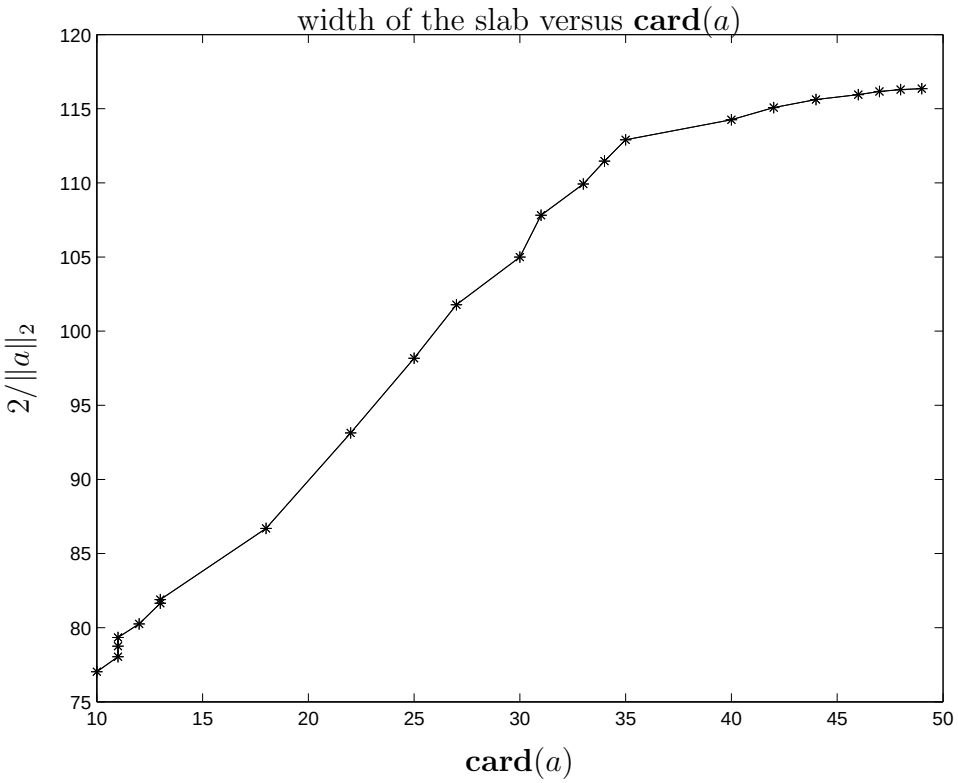
Use this data to choose a set of 10 features out of the 50 in the data. Give the indices of the features you choose. You may have several choices of sets of features here; you can just choose one. Then find the maximum thickness separating slab that uses only the chosen features. (This is standard practice: once you've chosen the features you're going to use, you optimize again, using only those features, and without the ℓ_1 regularization.

Solution: The MATLAB script used to solve this problem is

```
cvx_quiet(true);
sp_ln_sp_data;

% thickest slab
cvx_begin
    variables a(n) b
    minimize ( norm(a) )
    a'*X - b >= 1
    a'*Y - b <= -1
cvx_end
w_thickest = 2./norm(a);
disp('The thickness of the maximum thickness separating slab is: ');
disp(w_thickest);

% generating the trade-off curve
lambdas = logspace(-2,5);
A = zeros(n,length(lambdas));

for i=1:length(lambdas)
    cvx_begin
        variables a(n) b
        minimize ( norm(a) + lambdas(i)*norm(a,1) )
        a'*X - b >= 1
        a'*Y - b <= -1
    cvx_end
    A(:,i) = a;
end
w = 2./norms(A);        % width of the slab
card = sum((abs(A) > 1e-4));
plot(card,w)
hold on;
plot(card,w,'*')
xlabel('card(a)');
ylabel('w');
title('width of the slab versus cardinality of a');
```

```
% feature selection (fixing card(a) to 10)
indices = find(card == 10);
idx = indices(end);
w_before = w(idx);
a_selected = A(:,idx);
features = find(abs(a_selected) > 1e-4);
num_feat = length(features);
X_sub = X(features,:);
Y_sub = Y(features,:);
cvx_begin
    variables a(num_feat) b
    minimize ( norm(a) )
    a'*X_sub - b >= 1
    a'*Y_sub - b <= -1
cvx_end
w_after = 2/norm(a);
disp('Using only the following 10 features');
disp(features');
disp('the width of the thickest slab returned by the regularized');
disp('optimization problem was: ');
disp(w_before);
disp('after reoptimizing, the width of the thickest slab is: ');
disp(w_after)
```

The thickness of the maximum thickness separating slab is found to be 116.4244. The script also generates the following trade-off curve



We find that, using only the features

$$1,\ 7,\ 8,\ 18,\ 19,\ 21,\ 23,\ 26,\ 27,\ 46,$$

the width of the thickest slab found from the regularized optimization problem is 77.0246. After re-optimizing over this subset of variables, we find that the width of the thickest slab increases to 78.4697.

6. *Bounding object position from multiple camera views.* A small object is located at unknown position $x \in \mathbf{R}^3$, and viewed by a set of m cameras. Our goal is to find a box in $\mathbf{R}^3$,

$$\mathcal{B} = \{z \in \mathbf{R}^3 \mid l \preceq z \preceq u\},$$

for which we can guarantee $x \in \mathcal{B}$. We want the smallest possible such bounding box. (Although it doesn't matter, we can use volume to judge 'smallest' among boxes.)

Now we describe the cameras. The object at location $x \in \mathbf{R}^3$ creates an image on image plane of camera i at location

$$v_i = \frac{1}{c_i^T x + d_i}(A_i x + b_i) \in \mathbf{R}^2.$$

The matrices $A_i \in \mathbf{R}^{2\times 3}$, vectors $b_i \in \mathbf{R}^2$ and $c_i \in \mathbf{R}^3$, and real numbers $d_i \in \mathbf{R}$ are known, and depend on the camera positions and orientations. We assume that $c_i^T x + d_i > 0$. The 3×4 matrix

$$P_i = \begin{bmatrix} A_i & b_i \\ c_i^T & d_i \end{bmatrix}$$

is called the *camera matrix* (for camera i). It is often (but not always) the case the that the first 3 columns of P_i (*i.e.*, A_i stacked above c_i^T) form an orthogonal matrix, in which case the camera is called *orthographic*.

We do not have direct access to the image point v_i; we only know the (square) pixel that it lies in. In other words, the camera gives us a measurement $\hat{v}_i$ (the center of the pixel that the image point lies in); we are guaranteed that

$$\|v_i - \hat{v}_i\|_\infty \leq \rho_i/2,$$

where ρ_i is the pixel width (and height) of camera i. (We know nothing else about v_i; it could be any point in this pixel.)

Given the data A_i, b_i, c_i, d_i, $\hat{v}_i$, ρ_i, we are to find the smallest box $\mathcal{B}$ (*i.e.*, find the vectors l and u) that is guaranteed to contain x. In other words, find the smallest box in $\mathbf{R}^3$ that contains all points consistent with the observations from the camera.

(a) Explain how to solve this using convex or quasiconvex optimization. You must explain any transformations you use, any new variables you introduce, etc. If the convexity or quasiconvexity of any function in your formulation isn't obvious, be sure justify it.

(b) Solve the specific problem instance given in the file `camera_data.m`. Be sure that your final numerical answer (*i.e.*, l and u) stands out.

Solution:

(a) We get a subset $\mathcal{P} \subseteq \mathbf{R}^3$ (which we'll soon see is a polyhedron) of locations x that are consistent with the camera measurements. To find the smallest box that covers *any* subset in $\mathbf{R}^3$, all we need to do is maximize and minimize the (linear) functions x_1, x_2, and x_3 to get l and u. Here $\mathcal{P}$ is a polyhedron, so we'll end up solving 6 LPs, one to get each of l_1, l_2, l_3, u_1, u_2, and u_3.

Now let's look more closely at $\mathcal{P}$. Our measurements tell us that

$$\hat{v}_i - \rho_i/2 \leq \frac{1}{c_i^T x + d_i}(A_i x + b_i) \leq \hat{v}_i + \rho_i/2, \quad i = 1, \ldots, m.$$

We multiply through by $c_i^T x + d_i$, which is positive, to get

$$(\hat{v}_i - \rho_i/2)(c_i^T x + d_i) \leq A_i x + b_i \leq (\hat{v}_i + \rho_i/2)(c_i^T x + d_i), \quad i = 1, \ldots, m,$$

as set of $2m$ linear inequalities on x. In particular, it defines a polyhedron.

To get l_k we solve the LP

$$\begin{array}{ll} \text{minimize} & x_k \\ \text{subject to} & (\hat{v}_i - \rho_i/2)(c_i^T x + d_i) \leq A_i x + b_i, \quad i = 1, \ldots, m, \\ & A_i x + b_i \leq (\hat{v}_i + \rho_i/2)(c_i^T x + d_i), \quad i = 1, \ldots, m, \end{array}$$

for $k = 1, 2, 3$; to get u_k we maximize the same objective.

(b) Here is a MATLAB script that solves given instance:

```
% load the data
camera_data;
A1 = P1(1:2,1:3); b1 = P1(1:2,4); c1 = P1(3,1:3); d1 = P1(3,4);
A2 = P2(1:2,1:3); b2 = P2(1:2,4); c2 = P2(3,1:3); d2 = P2(3,4);
A3 = P3(1:2,1:3); b3 = P3(1:2,4); c3 = P3(3,1:3); d3 = P3(3,4);
A4 = P4(1:2,1:3); b4 = P4(1:2,4); c4 = P4(3,1:3); d4 = P4(3,4);

cvx_quiet(true);
for bounds = 1:6
    cvx_begin
        variable x(3)
        switch bounds
            case 1
                minimize x(1)
            case 2
                maximize x(1)
            case 3
                minimize x(2)
            case 4
                maximize x(2)
```

```
            case 5
                minimize x(3)
            case 6
                maximize x(3)
        end
        % constraints for 1st camera
        (vhat(:,1)-rho(1)/2)*(c1*x + d1) <= A1*x + b1;
        A1*x + b1 <= (vhat(:,1)+rho(1)/2)*(c1*x + d1);
        % constraints for 2ns camera
        (vhat(:,2)-rho(2)/2)*(c2*x + d2) <= A2*x + b2;
        A2*x + b2 <= (vhat(:,2)+rho(2)/2)*(c2*x + d2);
        % constraints for 3rd camera
        (vhat(:,3)-rho(3)/2)*(c3*x + d3) <= A3*x + b3;
        A3*x + b3 <= (vhat(:,3)+rho(3)/2)*(c3*x + d3);
        % constraints for 4th camera
        (vhat(:,4)-rho(4)/2)*(c4*x + d4) <= A4*x + b4;
        A4*x + b4 <= (vhat(:,4)+rho(4)/2)*(c4*x + d4);
    cvx_end
    val(bounds) = cvx_optval;
end
disp(['l1 = ' num2str(val(1))]);
disp(['l2 = ' num2str(val(3))]);
disp(['l3 = ' num2str(val(5))]);
disp(['u1 = ' num2str(val(2))]);
disp(['u2 = ' num2str(val(4))]);
disp(['u3 = ' num2str(val(6))]);
```

The MATLAB script returns the following results:

```
l1 = -0.99561
l2 = 0.27531
l3 = -0.67899
u1 = -0.8245
u2 = 0.37837
u3 = -0.57352
```

7. $\ell_{1.5}$ *optimization.* Optimization and approximation methods that use both an ℓ_2-norm (or its square) and an ℓ_1-norm are currently very popular in statistics, machine learning, and signal and image processing. Examples include Huber estimation, LASSO, basis pursuit, SVM, various ℓ_1-regularized classification methods, total variation denoising, etc. Very roughly, an ℓ_2-norm corresponds to Euclidean distance (squared), or the negative log-likelihood function for a Gaussian; in contrast the ℓ_1-norm gives 'robust' approximation, *i.e.*, reduced sensitivity to outliers, and also tends to yield sparse solutions (of whatever the argument of the norm is). (All of this is just background; you don't need to know any of this to solve the problem.)

   In this problem we study a natural method for blending the two norms, by using the $\ell_{1.5}$-norm, defined as

$$\|z\|_{1.5} = \left(\sum_{i=1}^{k} |z_i|^{3/2}\right)^{2/3}$$

   for $z \in \mathbf{R}^k$. We will consider the simplest approximation or regression problem:

$$\text{minimize} \quad \|Ax - b\|_{1.5},$$

   with variable $x \in \mathbf{R}^n$, and problem data $A \in \mathbf{R}^{m\times n}$ and $b \in \mathbf{R}^m$. We will assume that $m > n$ and the A is full rank (*i.e.*, rank n). The hope is that this $\ell_{1.5}$-optimal approximation problem should share some of the good features of ℓ_2 and ℓ_1 approximation.

   (a) Give optimality conditions for this problem. Try to make these as simple as possible. Your solution should have the form 'x is optimal for the $\ell_{1.5}$-norm approximation problem if and only if ...'.

   (b) Explain how to formulate the $\ell_{1.5}$-norm approximation problem as an SDP. (Your SDP can include linear equality and inequality constraints.)

   (c) Solve the specific numerical instance generated by the following code:

```
randn('state',0);
A=randn(100,30);
b=randn(100,1);
```

   Numerically verify the optimality conditions. Give a histogram of the residuals, and repeat for the ℓ_2-norm and ℓ_1-norm approximations. You can use any method you like to solve the problem (but of course you must explain how you did it); in particular, you do not need to use the SDP formulation found in part (b).

Solution:

   (a) We can just as well minimize the objective to the 3/2 power, *i.e.*, solve the problem

$$\text{minimize} \quad f(x) = \textstyle\sum_{i=1}^{m} |a_i^T x - b_i|^{3/2}$$

This objective is differentiable, in fact, so the optimality condition is simply that the gradient should vanish. (But it is not twice differentiable.) The gradient is

$$\nabla f(x) = \sum_{i=1}^{m} (3/2)\,\mathbf{sign}(a_i^T x - b_i)|a_i^T x - b_i|^{1/2} a_i,$$

so the optimality condition is just

$$\sum_{i=1}^{m} (3/2)\,\mathbf{sign}(r_i)|r_i|^{1/2} a_i = 0,$$

where $r_i = a_i^T x - b_i$ is the ith residual. We can, of course, drop the factor $3/2$.

(b) We can write an equivalent problem

$$\begin{array}{ll} \text{minimize} & \mathbf{1}^T t \\ \text{subject to} & s^{3/2} \preceq t, \\ & -s_i \preceq a_i^T x - b_i \preceq s_i \quad i = 1, \ldots, m, \end{array}$$

with new variables $t, s \in \mathbf{R}^m$.

We need a way to express $s_i^{3/2} \leq t_i$ using LMIs. We first write it as $s_i^2 \leq t_i \sqrt{s_i}$. We're going to express this using some LMIs. Recall that the general 2×2 LMI

$$\left[\begin{array}{cc} u & v \\ v & w \end{array} \right] \succeq 0$$

is equivalent to $u \geq 0$, $uw \geq v^2$. So we can write $s_i^2 \leq t_i \sqrt{s_i}$ as

$$\left[\begin{array}{cc} \sqrt{s}_i & s_i \\ s_i & t \end{array} \right] \succeq 0.$$

Now this is not yet an LMI, because the $1,1$ entry is not affine in the variables. To deal with this, we introduce a new variable $y \in \mathbf{R}^m$, which satisfies $0 \preceq y \preceq \sqrt{s}$:

$$\left[\begin{array}{cc} y_i & s_i \\ s_i & t_i \end{array} \right] \succeq 0, \quad \left[\begin{array}{cc} s_i & y_i \\ y_i & 1 \end{array} \right] \succeq 0.$$

These are LMIs in the variables. The first LMI is equivalent to $y_i \geq 0$, $y_i t_i \geq s_i^2$. The second LMI is equivalent to $s_i \geq y_i^2$, *i.e.*, $\sqrt{s}_i \geq |y_i|$. These two together are equivalent to $s_i^2 \geq t_i \sqrt{s}_i$. (Here we use the fact that if we increase the $1,1$ entry of a matrix, it gets more positive semidefinite. (That's informal, but you know what we mean.)

Now we can assemble an SDP to solve our $\ell_{1.5}$-norm approximation problem:

$$\begin{array}{ll} \text{minimize} & \mathbf{1}^T t \\ \text{subject to} & -s_i \preceq a_i^T x - b_i \preceq s_i, \quad i = 1, \ldots, m \\ & \left[\begin{array}{cc} y_i & s_i \\ s_i & t_i \end{array} \right] \succeq 0, \quad \left[\begin{array}{cc} s_i & y_i \\ y_i & 1 \end{array} \right] \succeq 0, \quad i = 1, \ldots, m, \end{array}$$

with variables $x \in \mathbf{R}^n$, $t \in \mathbf{R}^m$, $y \in \mathbf{R}^m$, $s \in \mathbf{R}^m$.

Here is another solution several of you used, which we like. The final SDP is

$$\begin{array}{ll} \text{minimize} & z \\ \text{subject to} & -s_i \preceq a_i^T x - b_i \preceq s_i, \quad i = 1, \ldots, m \\ & \begin{bmatrix} s_i & y_i \\ y_i & 1 \end{bmatrix} \succeq 0, \quad i = 1, \ldots, m, \\ & \begin{bmatrix} z & s^T \\ s & \mathbf{diag}(y) \end{bmatrix} \succeq 0, \end{array}$$

with variables $x \in \mathbf{R}^n$, $y \in \mathbf{R}^m$, $s \in \mathbf{R}^m$, and $z \in \mathbf{R}$.

The first set of inequalities is equivalent to $|a_i^T x - b_i| \leq s_i$; the set of 2×2 LMIs is equivalent to $s_i \geq y_i^2$, and the last $(2m+1) \times (2m+1)$ LMI is equivalent to

$$z \geq s^T(\mathbf{diag}(y)^{-1})s = \sum_{i=1}^m s_i^2/y_i.$$

Evidently we minimize z, and therefore the righthand side above. For s_i fixed, the choice $y_i = \sqrt{s_i}$ minimizes the objective, so we are minimizing

$$\sum_{i=1}^m s_i^2/y_i = \sum_{i=1}^m s_i^{3/2}.$$

(c) We're going to use `cvx` to solve the problem. The function `norm(r,1.5)` isn't implemented yet, so we'll have to do it ourselves. One simple way is to note that $|r|^{3/2} = r^2/\sqrt{r}$, which is the composition of the quadratic over linear function x_1^2/x_2 with $x_1 = r$, $x_2 = \sqrt{r}$. Fortunately, the result is convex, since the quadratic over linear function is convex and decreasing in its second argument, so it can accept a concave positive function there. In other words, `cvx` will accept `quad_over_lin(s,sqrt(s))`, and recognize it as convex. So we have a snappy, short way to express $s^{3/2}$ for $s > 0$. Now we have to form the composition of this with the convex function $r_i = a_i^T x - b_i$. Here is one way to do this.

```
cvx_begin
  variables x(n) s(m);
  s >= abs(A*x-b);
  minimize(sum(quad_over_lin(s,sqrt(s),0)));
cvx_end
```

The following code solve the problem for the different norms and plot histograms of the residuals.

```
n=30;
m=100;
randn('state',0);
```

```
A=randn(m,n);
b=randn(m,1);


%l1.5 solution
cvx_begin
  variables x(n) s(m);
  s >= abs(A*x-b);
  minimize(sum(quad_over_lin(s,sqrt(s),0)));
cvx_end

%l2 solution
xl2=A\b;

%l1 solution
cvx_begin
  variables xl1(n);
  minimize(norm(A*xl1-b,1));
cvx_end

r=A*x-b; %residuals
rl2=A*xl2-b;
rl1=A*xl1-b;

%check optimality condition
A'*(3/2*sign(r).*sqrt(abs(r)))

subplot(3,1,1)
hist(r)
axis([-2.5 2.5 0 50])
xlabel('r')
subplot(3,1,2)
hist(rl2)
axis([-2.5 2.5 0 50])
xlabel('r2')
subplot(3,1,3)
hist(rl1)
axis([-2.5 2.5 0 50])
xlabel('r1')

%solution using SDP
cvx_begin sdp
```

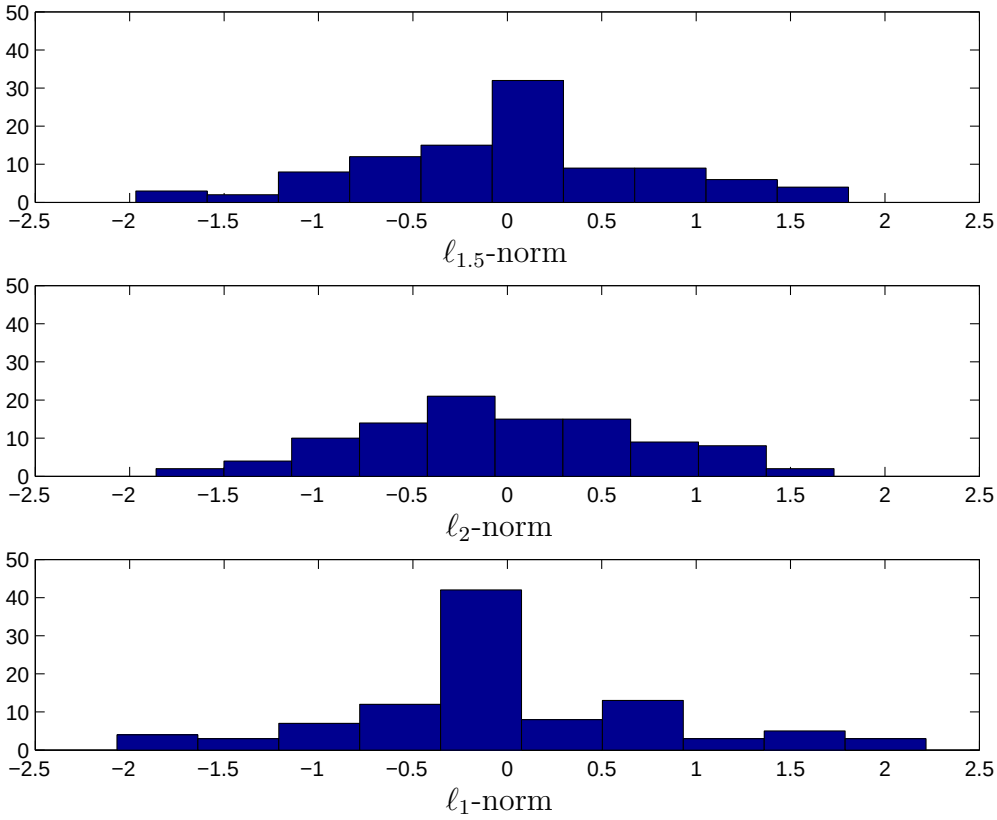


Figure 2: Histogram of the residuals for $\ell_{1.5}$-norm, ℓ_2-norm, and ℓ_1-norm

```
  variables xdf(n) r(m) y(m) t(m);
  A*xdf-b<=r;
  -r<=A*xdf-b;
  minimize(sum(t));
  for i=1:m
     [y(i), r(i); r(i), t(i)]>=0;
     [r(i), y(i); y(i), 1]>=0;
  end
cvx_end
```

Figure 2 shows the histograms of the residuals for the three norms.

8. *Three-way linear classification.* We are given data

$$x^{(1)}, \ldots, x^{(N)}, \quad y^{(1)}, \ldots, y^{(M)}, \quad z^{(1)}, \ldots, z^{(P)},$$

three nonempty sets of vectors in $\mathbf{R}^n$. We wish to find three affine functions on $\mathbf{R}^n$,

$$f_i(z) = a_i^T z - b_i, \quad i = 1, 2, 3,$$

that satisfy the following properties:

$$\begin{array}{l} f_1(x^{(j)}) > \max\{f_2(x^{(j)}), f_3(x^{(j)})\}, \quad j = 1, \ldots, N, \\ f_2(y^{(j)}) > \max\{f_1(y^{(j)}), f_3(y^{(j)})\}, \quad j = 1, \ldots, M, \\ f_3(z^{(j)}) > \max\{f_1(z^{(j)}), f_2(z^{(j)})\}, \quad j = 1, \ldots, P. \end{array}$$

In words: f_1 is the largest of the three functions on the x data points, f_2 is the largest of the three functions on the y data points, f_3 is the largest of the three functions on the z data points. We can give a simple geometric interpretation: The functions f_1, f_2, and f_3 partition $\mathbf{R}^n$ into three regions,

$$\begin{array}{rcl} R_1 & = & \{z \mid f_1(z) > \max\{f_2(z), f_3(z)\}\}, \\ R_2 & = & \{z \mid f_2(z) > \max\{f_1(z), f_3(z)\}\}, \\ R_3 & = & \{z \mid f_3(z) > \max\{f_1(z), f_2(z)\}\}, \end{array}$$

defined by where each function is the largest of the three. Our goal is to find functions with $x^{(j)} \in R_1$, $y^{(j)} \in R_2$, and $z^{(j)} \in R_3$.

Pose this as a convex optimization problem. You may not use strict inequalities in your formulation.

Solution: We need

$$\begin{array}{rcll} f_1(x^{(j)}) & > & f_2(x^{(j)}), & j = 1, \ldots, N, \\ f_1(x^{(j)}) & > & f_3(x^{(j)}), & j = 1, \ldots, N, \\ f_2(y^{(j)}) & > & f_1(y^{(j)}), & j = 1, \ldots, M, \\ f_2(y^{(j)}) & > & f_3(y^{(j)}), & j = 1, \ldots, M, \\ f_3(z^{(j)}) & > & f_1(z^{(j)}), & j = 1, \ldots, P, \\ f_3(z^{(j)}) & > & f_2(z^{(j)}), & j = 1, \ldots, P, \end{array}$$

which is a set of $2(M+N+P)$ strict linear inequalities in the variables a_i and b_i. They are homogeneous in a_i and b_i so we can express them as

$$\begin{array}{rcll} f_1(x^{(j)}) & \geq & f_2(x^{(j)}) + 1, & j = 1, \ldots, N, \\ f_1(x^{(j)}) & \geq & f_3(x^{(j)}) + 1, & j = 1, \ldots, N, \\ f_2(y^{(j)}) & \geq & f_1(y^{(j)}) + 1, & j = 1, \ldots, M, \\ f_2(y^{(j)}) & \geq & f_3(y^{(j)}) + 1, & j = 1, \ldots, M, \\ f_3(z^{(j)}) & \geq & f_1(z^{(j)}) + 1, & j = 1, \ldots, P, \\ f_3(z^{(j)}) & \geq & f_2(z^{(j)}) + 1, & j = 1, \ldots, P, \end{array}$$

an LP feasibility problem. More explicitly we have

$$\begin{array}{rcll}
(a_1 - a_2)^T x^{(j)} - (b_1 - b_2) & \geq & 1, & j = 1, \ldots, N, \\
(a_1 - a_3)^T x^{(j)} - (b_1 - b_3) & \geq & 1, & j = 1, \ldots, N, \\
(a_2 - a_1)^T y^{(j)} - (b_2 - b_1) & \geq & 1, & j = 1, \ldots, M, \\
(a_2 - a_3)^T y^{(j)} - (b_2 - b_3) & \geq & 1, & j = 1, \ldots, M, \\
(a_3 - a_1)^T z^{(j)} - (b_3 - b_1) & \geq & 1, & j = 1, \ldots, P, \\
(a_3 - a_2)^T z^{(j)} - (b_3 - b_2) & \geq & 1, & j = 1, \ldots, P.
\end{array}$$

The first and third state that the affine function $(a_1 - a_2)^T z - (b_1 - b_2)$ strictly separates the $x^{(j)}$ and $y^{(j)}$. The second and fifth state that the affine function $(a_1 - a_3)^T z - (b_1 - b_3)$ strictly separates the $x^{(j)}$ and $z^{(j)}$. The fourth and sixth state that the affine function $(a_2 - a_3)^T z - (b_2 - b_3)$ strictly separates the $y^{(j)}$ and $z^{(j)}$.

Note that we can add any vector α to each of the a_i, without affecting these inequalities (which only refer to difference between a_i's), and we can add any number β to each of the b_i's for the same reason. We can use this observation to normalize or simplify the a_i and b_i. For example, we can assume without loss of generality that $a_1 = 0$, $b_1 = 0$. Or, in a more symmetric way, we can require that $a_1 + a_2 + a_3 = 0$ and $b_1 + b_2 + b_3 = 0$.

We didn't ask you to try it out on numerical data. But we did, just for fun, and to be sure our theory was really correct.

The following MATLAB script implements this method for 3-way classification and tests it on a small separable data set

```
sep3way_data;

cvx_begin
    variables a1(2) a2(2) a3(2) b1 b2 b3
    (a1 - a2)'*X - (b1 - b2) >= 1
    (a1 - a3)'*X - (b1 - b3) >= 1
    (a2 - a1)'*Y - (b2 - b1) >= 1
    (a2 - a3)'*Y - (b2 - b3) >= 1
    (a3 - a1)'*Z - (b3 - b1) >= 1
    (a3 - a2)'*Z - (b3 - b2) >= 1
    a1 + a2 + a3 == 0
    b1 + b2 + b3 == 0
cvx_end

% maximally confusing point
p = [(a1-a2)';(a1-a3)']\[(b1-b2);(b1-b3)];

% plot
```

```
plot(X(1,:),X(2,:),'*');
hold on;
plot(Y(1,:),Y(2,:),'ro');
plot(Z(1,:),Z(2,:),'g+');
plot(p(1),p(2),'ks');
t = [-5:0.01:8];
u1 = a1 - a2; v1 = b1 - b2;
u2 = a2 - a3; v2 = b2 - b3;
u3 = a3 - a1; v3 = b3 - b1;
line1 = (-t*u1(1) + v1)/u1(2);
idx1 = find(u2'*[t;line1] - v2 > 0);
plot(t(idx1),line1(idx1));
line2 = (-t*u2(1) + v2)/u2(2);
idx2 = find(u3'*[t; line2] - v3 > 0);
plot(t(idx2),line2(idx2),'r');
line3 = (-t*u3(1) + v3)/u3(2);
idx3 = find(u1'*[t; line3] - v1 > 0);
plot(t(idx3),line3(idx3),'g');
```

The following figure is generated.