

Additional Exercises for *Convex Optimization*

Stephen Boyd

Lieven Vandenberghe

March 18, 2016

This is a collection of additional exercises, meant to supplement those found in the book *Convex Optimization*, by Stephen Boyd and Lieven Vandenberghe. These exercises were used in several courses on convex optimization, EE364a (Stanford), EE236b (UCLA), or 6.975 (MIT), usually for homework, but sometimes as exam questions. Some of the exercises were originally written for the book, but were removed at some point. Many of them include a computational component using CVX, a Matlab package for convex optimization; files required for these exercises can be found at the book web site www.stanford.edu/~boyd/cvxbook/. We are in the process of adapting many of these problems to be compatible with two other packages for convex optimization: CVXPY (Python) and Convex.jl (Julia). Some of the exercises require a knowledge of elementary analysis.

You are free to use these exercises any way you like (for example in a course you teach), provided you acknowledge the source. In turn, we gratefully acknowledge the teaching assistants (and in some cases, students) who have helped us develop and debug these exercises. Pablo Parrilo helped develop some of the exercises that were originally used in 6.975.

Course instructors can obtain solutions by email to us. Please specify the course you are teaching and give its URL.

We'll update this document as new exercises become available, so the exercise numbers and sections will occasionally change. We have categorized the exercises into sections that follow the book chapters, as well as various additional application areas. Some exercises fit into more than one section, or don't fit well into any section, so we have just arbitrarily assigned these.

Stephen Boyd and Lieven Vandenberghe

Contents

1 Convex sets	3
2 Convex functions	8
3 Convex optimization problems	29
4 Duality	95
5 Approximation and fitting	136
6 Statistical estimation	204
7 Geometry	235
8 Unconstrained and equality constrained minimization	279
9 Interior point methods	298
10 Mathematical background	321
11 Circuit design	324
12 Signal processing and communications	350
13 Finance	375
14 Mechanical and aerospace engineering	455
15 Graphs and networks	513
16 Energy and power	533
17 Miscellaneous applications	576

1 Convex sets

1.1 Is the set $\{a \in \mathbf{R}^k \mid p(0) = 1, |p(t)| \leq 1 \text{ for } \alpha \leq t \leq \beta\}$, where

$$p(t) = a_1 + a_2t + \cdots + a_k t^{k-1},$$

convex?

Solution. Yes, it is convex; it is the intersection of an infinite number of slabs,

$$\{a \mid -1 \leq a_1 + a_2t + \cdots + a_k t^{k-1} \leq 1\},$$

parametrized by $t \in [\alpha, \beta]$, and the hyperplane

$$\{a \mid a_0 = 1\}.$$

1.2 Set distributive characterization of convexity. [?, p21], [?, Theorem 3.2] Show that $C \subseteq \mathbf{R}^n$ is convex if and only if $(\alpha + \beta)C = \alpha C + \beta C$ for all nonnegative α, β .

Solution. The equality is trivially true for $\alpha = \beta = 0$, so we will assume that $\alpha + \beta \neq 0$, and dividing by $\alpha + \beta$, we can rephrase the theorem as follows: C is convex if and only if

$$C = \theta C + (1 - \theta)C$$

for $0 \leq \theta \leq 1$.

$C \subseteq \theta C + (1 - \theta)C$ for all $\theta \in [0, 1]$ is true for any set C , because $x = \theta x + (1 - \theta)x$.

$C \supseteq \theta C + (1 - \theta)C$ for all $\theta \in [0, 1]$ is just a rephrasing of Jensen's inequality.

1.3 Composition of linear-fractional functions. Suppose $\phi : \mathbf{R}^n \rightarrow \mathbf{R}^m$ and $\psi : \mathbf{R}^m \rightarrow \mathbf{R}^p$ are the linear-fractional functions

$$\phi(x) = \frac{Ax + b}{c^T x + d}, \quad \psi(y) = \frac{Ey + f}{g^T y + h},$$

with domains $\mathbf{dom} \phi = \{x \mid c^T x + d > 0\}$, $\mathbf{dom} \psi = \{y \mid g^T y + h > 0\}$. We associate with ϕ and ψ the matrices

$$\begin{bmatrix} A & b \\ c^T & d \end{bmatrix}, \quad \begin{bmatrix} E & f \\ g^T & h \end{bmatrix},$$

respectively.

Now consider the composition Γ of ψ and ϕ , i.e., $\Gamma(x) = \psi(\phi(x))$, with domain

$$\mathbf{dom} \Gamma = \{x \in \mathbf{dom} \phi \mid \phi(x) \in \mathbf{dom} \psi\}.$$

Show that Γ is linear-fractional, and that the matrix associated with it is the product

$$\begin{bmatrix} E & f \\ g^T & h \end{bmatrix} \begin{bmatrix} A & b \\ c^T & d \end{bmatrix}.$$

Solution. We have, for $x \in \text{dom } \Gamma$,

$$\Gamma(x) = \frac{E((Ax + b)/(c^T x + d)) + f}{g^T(Ax + b)/(c^T x + d) + h}.$$

Multiplying numerator and denominator by $c^T x + d$ yields

$$\begin{aligned}\Gamma(x) &= \frac{EAx + Eb + fc^T x + fd}{g^T Ax + g^T b + hc^T x + hd} \\ &= \frac{(EA + fc^T)x + (Eb + fd)}{(g^T A + hc^T)x + (g^T b + hd)},\end{aligned}$$

which is the linear fractional function associated with the product matrix.

1.4 Dual of exponential cone. The exponential cone $K_{\exp} \subseteq \mathbf{R}^3$ is defined as

$$K_{\exp} = \{(x, y, z) \mid y > 0, ye^{x/y} \leq z\}.$$

Find the dual cone K_{\exp}^* .

We are not worried here about the fine details of what happens on the boundaries of these cones, so you really needn't worry about it. But we make some comments here for those who do care about such things.

The cone K_{\exp} as defined above is not closed. To obtain its closure, we need to add the points

$$\{(x, y, z) \mid x \leq 0, y = 0, z \geq 0\}.$$

(This makes no difference, since the dual of a cone is equal to the dual of its closure.)

Solution. The dual cone can be expressed as

$$\begin{aligned}K_{\exp}^* &= \mathbf{cl}\{(u, v, w) \mid u < 0, -ue^{v/u} \leq ew\} \\ &= \{(u, v, w) \mid u < 0, -ue^{v/u} \leq ew\} \cup \{(0, v, w) \mid v \geq 0, w \geq 0\},\end{aligned}$$

where **cl** means closure. We didn't expect people to add the points on the boundary, *i.e.*, to work out the second set in the second line.

The dual cone can be expressed several other ways as well. The conditions $u < 0, -ue^{v/u} \leq ew$ can be expressed as

$$(u/w)\log(-u/w) + v/w - u/w \geq 0,$$

or

$$u\log(-u/w) + v - u \geq 0,$$

or

$$\log(-u/w) \leq 1 - (v/u).$$

Now let's derive the result. For (u, v, w) to be in K_{\exp}^* , we need to have $ux + vy + wz \geq 0$ whenever $y > 0$ and $ye^{x/y} \leq z$. Thus, we must have $w \geq 0$; otherwise we can make $ux + vy + wz$ negative

by choosing a large enough z . Now let's see what happens when $w = 0$. In this case we need $ux + vy \geq 0$ for all $y > 0$. This happens only if $u = 0$ and $v \geq 0$. So points with

$$u = 0, \quad v \geq 0, \quad w = 0$$

are in K_{exp}^* .

Let's now consider the case $w > 0$. We'll minimize $ux + vy + wz$ over all x, y and z that satisfy $y > 0$ and $ye^{x/y} \leq z$. Since $w > 0$, we minimize over z by taking $z = ye^{x/y}$, which yields

$$ux + vy + wye^{x/y}.$$

Now we will minimize over x . If $u > 0$, this is unbounded below as $x \rightarrow -\infty$. If $u = 0$, the minimum is not achieved, but occurs as $x \rightarrow -\infty$; the minimum value is then vy . This has a nonnegative minimum value over all $y > 0$ only when $v \geq 0$. Thus we find that points satisfying

$$u = 0, \quad v \geq 0, \quad w > 0$$

are in K_{exp}^* .

If $u < 0$, the minimum of $ux + vy + wye^{x/y}$ occurs when its derivative with respect to x vanishes. This leads to $u + we^{x/y} = 0$, i.e., $x = y \log(-u/w)$. For this value of x the expression above becomes

$$y(u \log(-u/w) + v - u).$$

Now we minimize over $y > 0$. We get $-\infty$ if $u \log(-u/w) + v - u < 0$; we get 0 if $u \log(-u/w) + v - u \geq 0$.

So finally, we have our condition:

$$u \log(-u/w) + v - u \geq 0, \quad u < 0, \quad w > 0.$$

Dividing by u and taking the exponential, we can write this as

$$-ue^{v/u} \leq ew, \quad u < 0.$$

(The condition $w > 0$ is implied by these two conditions.)

Finally, then, we have

$$K_{\text{exp}}^* = \{(u, v, w) \mid u < 0, -ue^{v/u} \leq ew\} \cup \{(0, v, w) \mid v \geq 0, w \geq 0\}.$$

1.5 Dual of intersection of cones. Let C and D be closed convex cones in \mathbf{R}^n . In this problem we will show that

$$(C \cap D)^* = C^* + D^*.$$

Here, $+$ denotes set addition: $C^* + D^*$ is the set $\{u + v \mid u \in C^*, v \in D^*\}$. In other words, the dual of the intersection of two closed convex cones is the sum of the dual cones.

- (a) Show that $C \cap D$ and $C^* + D^*$ are convex cones. (In fact, $C \cap D$ and $C^* + D^*$ are closed, but we won't ask you to show this.)

- (b) Show that $(C \cap D)^* \supseteq C^* + D^*$.
(c) Now let's show $(C \cap D)^* \subseteq C^* + D^*$. You can do this by first showing

$$(C \cap D)^* \subseteq C^* + D^* \iff C \cap D \supseteq (C^* + D^*)^*.$$

You can use the following result:

If K is a closed convex cone, then $K^{**} = K$.

Next, show that $C \cap D \supseteq (C^* + D^*)^*$ and conclude $(C \cap D)^* = C^* + D^*$.

- (d) Show that the dual of the polyhedral cone $V = \{x \mid Ax \succeq 0\}$ can be expressed as

$$V^* = \{A^T v \mid v \succeq 0\}.$$

Solution.

- (a) Suppose $x \in C \cap D$. This implies that $x \in C$ and $x \in D$, which implies $\theta x \in C$ and $\theta x \in D$ for any $\theta \geq 0$. Thus, $\theta x \in C \cap D$ for any $\theta \geq 0$, which shows $C \cap D$ is a cone. We know $C \cap D$ is convex since the intersection of convex sets is convex.
To show $C^* + D^*$ is a closed convex cone, note that both C^* and D^* are convex cones, thus $C^* + D^*$ is the conic hull of $C^* \cup D^*$, which is a convex cone.
- (b) Suppose $x \in C^* + D^*$. We can write x as $x = u + v$, where $u \in C^*$ and $v \in D^*$. We know $u^T y \geq 0$ for all $y \in C$ and $v^T y \geq 0$ for all $y \in D$, which implies that $x^T y = u^T y + v^T y \geq 0$ for all $y \in C \cap D$. This shows $x \in (C \cap D)^*$, and so $(C \cap D)^* \supseteq C^* + D^*$.
- (c) We showed in part (a) that $C \cap D$ and $C^* + D^*$ are closed convex cones. This implies $(C \cap D)^{**} = C \cap D$ and $(C^* + D^*)^{**} = (C^* + D^*)$, and so

$$(C \cap D)^* \subseteq C^* + D^* \iff C \cap D \supseteq (C^* + D^*)^*.$$

Suppose $x \in (C^* + D^*)^*$. $x^T y \geq 0$ for all $y = u + v$, where $u \in C^*$, $v \in D^*$. This can be written as $x^T u + x^T v \geq 0$, for all $u \in C^*$ and $v \in D^*$. Since $0 \in C^*$ and $0 \in D^*$, taking $v = 0$ we get $x^T u \geq 0$ for all $u \in C^*$, and taking $u = 0$ we get $x^T v \geq 0$ for all $v \in D^*$. This implies $x \in C^{**} = C$ and $x \in D^{**} = D$, i.e., $x \in C \cap D$.

So we have shown both $(C \cap D)^* \supseteq C^* + D^*$ and $(C \cap D)^* \subseteq C^* + D^*$, which implies $(C \cap D)^* = C^* + D^*$.

- (d) Using the result we just proved, we can write

$$V^* = \{x \mid a_1^T x \geq 0\}^* + \cdots + \{x \mid a_m^T x \geq 0\}^*.$$

The dual of $\{x \mid a_i^T x \geq 0\}$ is the set $\{\theta a_i \mid \theta \geq 0\}$, so we get

$$\begin{aligned} V^* &= \{\theta a_1 \mid \theta \geq 0\} + \cdots + \{\theta a_m \mid \theta \geq 0\} \\ &= \{\theta_1 a_1 + \cdots + \theta_m a_m \mid \theta_i \geq 0, i = 1, \dots, m\}. \end{aligned}$$

This can be more compactly written as

$$V^* = \{A^T v \mid v \succeq 0\}.$$

1.6 Polar of a set. The *polar* of $C \subseteq \mathbf{R}^n$ is defined as the set

$$C^\circ = \{y \in \mathbf{R}^n \mid y^T x \leq 1 \text{ for all } x \in C\}.$$

- (a) Show that C° is convex (even if C is not).
- (b) What is the polar of a cone?
- (c) What is the polar of the unit ball for a norm $\|\cdot\|$?
- (d) Show that if C is closed and convex, with $0 \in \text{int } C$, then $(C^\circ)^\circ = C$.

Solution.

- (a) The polar is the intersection of hyperplanes $\{y \mid y^T x \leq 1\}$, parametrized by $x \in C$, so it is convex.
- (b) If C is a cone, then we have $y^T x \leq 1$ for all $x \in C$ if and only if $y^T x \leq 0$ for all $x \in C$. (To see this, suppose that $y^T x > 0$ for some $x \in C$. Then $\alpha x \in C$ for all $\alpha > 0$, so $\alpha y^T x$ can be made arbitrarily large, and in particular, exceeds one for α large enough.) Therefore the polar of a cone K is $-K^*$, the negative of the dual cone.
- (c) The polar of the unit ball in the norm $\|\cdot\|$ is the unit ball for the dual norm $\|\cdot\|_*$.
- (d) Suppose that $x \in C$ and $y \in C^\circ$. Then we have $y^T x \leq 1$. Since this is true for any $y \in C^\circ$, we $x^T y \leq 1$ for all $y \in C^\circ$. Thus we have $x \in (C^\circ)^\circ$. So $C \subseteq (C^\circ)^\circ$.

Now suppose that $x \in (C^\circ)^\circ \setminus C$. Then we can find a separating hyperplane for $\{x\}$ and C , i.e., $a \neq 0$ and b with $a^T x > b$, and $a^T z \leq b$ for $z \in C$. Since $z = 0 \in \text{int } C$, we have $b > 0$. By scaling a and b , we can assume that $a^T x > 1$ and $a^T z \leq 1$ for all $z \in C$. Thus, $a \in C^\circ$. Our assumption $x \in (C^\circ)^\circ$ then tells us that $a^T x \leq 1$, a contradiction.

1.7 Dual cones in \mathbf{R}^2 . Describe the dual cone for each of the following cones.

- (a) $K = \{0\}$.
- (b) $K = \mathbf{R}^2$.
- (c) $K = \{(x_1, x_2) \mid |x_1| \leq x_2\}$.
- (d) $K = \{(x_1, x_2) \mid x_1 + x_2 = 0\}$.

Solution.

- (a) $K^* = \mathbf{R}^2$. To see this:

$$\begin{aligned} K^* &= \{y \mid y^T x \geq 0 \text{ for all } x \in K\} \\ &= \{y \mid y^T 0 \geq 0\} \\ &= \mathbf{R}^2. \end{aligned}$$

- (b) $K^* = \{0\}$. To see this, we need to identify the values of $y \in \mathbf{R}^2$ for which $y^T x \geq 0$ for all $x \in \mathbf{R}^2$. But given any $y \neq 0$, consider the choice $x = -y$, for which we have $y^T x = -\|y\|_2^2 < 0$. So the only possible choice is $y = 0$ (which indeed satisfies $y^T x \geq 0$ for all $x \in \mathbf{R}^2$).
- (c) $K^* = K$. (This cone is self-dual.)
- (d) $K^* = \{(x_1, x_2) \mid x_1 - x_2 = 0\}$. Here K is a line, and K^* is the line orthogonal to it.

2 Convex functions

2.1 *Maximum of a convex function over a polyhedron.* Show that the maximum of a convex function f over the polyhedron $\mathcal{P} = \text{conv}\{v_1, \dots, v_k\}$ is achieved at one of its vertices, *i.e.*,

$$\sup_{x \in \mathcal{P}} f(x) = \max_{i=1, \dots, k} f(v_i).$$

(A stronger statement is: the maximum of a convex function over a closed bounded convex set is achieved at an extreme point, *i.e.*, a point in the set that is not a convex combination of any other points in the set.) *Hint.* Assume the statement is false, and use Jensen's inequality.

Solution. Let's assume the statement is false: We have $z \in \mathcal{P}$ with $f(z) > f(v_i)$ for $i = 1, \dots, k$. We can represent z as

$$z = \lambda_1 v_1 + \dots + \lambda_k v_k,$$

where $\lambda \succeq 0$, $\mathbf{1}^T \lambda = 1$. Jensen's inequality tells us that

$$\begin{aligned} f(z) &= f(\lambda_1 v_1 + \dots + \lambda_k v_k) \\ &\leq \lambda_1 f(v_1) + \dots + \lambda_k f(v_k) \\ &< f(z), \end{aligned}$$

so we have a contradiction.

2.2 *A general vector composition rule.* Suppose

$$f(x) = h(g_1(x), g_2(x), \dots, g_k(x))$$

where $h : \mathbf{R}^k \rightarrow \mathbf{R}$ is convex, and $g_i : \mathbf{R}^n \rightarrow \mathbf{R}$. Suppose that for each i , one of the following holds:

- h is nondecreasing in the i th argument, and g_i is convex
- h is nonincreasing in the i th argument, and g_i is concave
- g_i is affine.

Show that f is convex. (This composition rule subsumes all the ones given in the book, and is the one used in software systems such as CVX.) You can assume that $\text{dom } h = \mathbf{R}^k$; the result also holds in the general case when the monotonicity conditions listed above are imposed on \tilde{h} , the extended-valued extension of h .

Solution. Fix x, y , and $\theta \in [0, 1]$, and let $z = \theta x + (1 - \theta)y$. Let's re-arrange the indexes so that g_i is affine for $i = 1, \dots, p$, g_i is convex for $i = p + 1, \dots, q$, and g_i is concave for $i = q + 1, \dots, k$. Therefore we have

$$\begin{aligned} g_i(z) &= \theta g_i(x) + (1 - \theta)g_i(y), & i &= 1, \dots, p, \\ g_i(z) &\leq \theta g_i(x) + (1 - \theta)g_i(y), & i &= p + 1, \dots, q, \\ g_i(z) &\geq \theta g_i(x) + (1 - \theta)g_i(y), & i &= q + 1, \dots, k. \end{aligned}$$

We then have

$$\begin{aligned} f(z) &= h(g_1(z), g_2(z), \dots, g_k(z)) \\ &\leq h(\theta g_1(x) + (1 - \theta)g_1(y), \dots, \theta g_k(x) + (1 - \theta)g_k(y)) \\ &\leq \theta h(g_1(x), \dots, g_k(x)) + (1 - \theta)h(g_1(y), \dots, g_k(y)) \\ &= \theta f(x) + (1 - \theta)f(y). \end{aligned}$$

The second line holds since, for $i = p + 1, \dots, q$, we have increased the i th argument of h , which is (by assumption) nondecreasing in the i th argument, and for $i = q + 1, \dots, k$, we have decreased the i th argument, and h is nonincreasing in these arguments. The third line follows from convexity of h .

- 2.3** *Logarithmic barrier for the second-order cone.* The function $f(x, t) = -\log(t^2 - x^T x)$, with $\text{dom } f = \{(x, t) \in \mathbf{R}^n \times \mathbf{R} \mid t > \|x\|_2\}$ (i.e., the second-order cone), is convex. (The function f is called the logarithmic barrier function for the second-order cone.) This can be shown many ways, for example by evaluating the Hessian and demonstrating that it is positive semidefinite. In this exercise you establish convexity of f using a relatively painless method, leveraging some composition rules and known convexity of a few other functions.

- (a) Explain why $t - (1/t)u^T u$ is a concave function on $\text{dom } f$. *Hint.* Use convexity of the quadratic over linear function.
- (b) From this, show that $-\log(t - (1/t)u^T u)$ is a convex function on $\text{dom } f$.
- (c) From this, show that f is convex.

Solution.

- (a) $(1/t)u^T u$ is the quadratic over linear function, which is convex on $\text{dom } f$. So $t - (1/t)u^T u$ is concave, since it is a linear function minus a convex function.
- (b) The function $g(u) = -\log u$ is convex and decreasing, so if u is a concave (positive) function, the composition rules tell us that $g \circ u$ is convex. Here this means $-\log(t - (1/t)u^T u)$ is a convex function on $\text{dom } f$.
- (c) We write $f(x, t) = -\log(t - (1/t)u^T u) - \log t$, which shows that f is a sum of two convex functions, hence convex.

- 2.4** *A quadratic-over-linear composition theorem.* Suppose that $f : \mathbf{R}^n \rightarrow \mathbf{R}$ is nonnegative and convex, and $g : \mathbf{R}^n \rightarrow \mathbf{R}$ is positive and concave. Show that the function f^2/g , with domain $\text{dom } f \cap \text{dom } g$, is convex.

Solution. Without loss of generality we can assume that $n = 1$. (The general case follows by restricting to an arbitrary line.) Let x and y be in the domains of f and g , and let $\theta \in [0, 1]$, and define $z = \theta x + (1 - \theta)y$. By convexity of f we have

$$f(z) \leq \theta f(x) + (1 - \theta)f(y).$$

Since $f(x)$ and $\theta f(x) + (1 - \theta)f(y)$ are nonnegative, we have

$$f(z)^2 \leq (\theta f(x) + (1 - \theta)f(y))^2.$$

(The square function is monotonic on \mathbf{R}_+ .) By concavity of g , we have

$$g(z) \geq \theta g(x) + (1 - \theta)g(y).$$

Putting the last two together, we have

$$\frac{f(z)^2}{g(z)} \leq \frac{(\theta f(x) + (1 - \theta)f(y))^2}{\theta g(x) + (1 - \theta)g(y)}.$$

Now we use convexity of the function u^2/v , for $v > 0$, to conclude

$$\frac{(\theta f(x) + (1 - \theta)f(y))^2}{\theta g(x) + (1 - \theta)g(y)} \leq \theta \frac{f(x)^2}{g(x)} + (1 - \theta) \frac{f(y)^2}{g(y)}.$$

This, together with the inequality above, finishes the proof.

2.5 A perspective composition rule. [?] Let $f : \mathbf{R}^n \rightarrow \mathbf{R}$ be a convex function with $f(0) \leq 0$.

- (a) Show that the perspective $tf(x/t)$, with domain $\{(x, t) \mid t > 0, x/t \in \mathbf{dom} f\}$, is nonincreasing as a function of t .
- (b) Let g be concave and positive on its domain. Show that the function

$$h(x) = g(x)f(x/g(x)), \quad \mathbf{dom} h = \{x \in \mathbf{dom} g \mid x/g(x) \in \mathbf{dom} f\}$$

is convex.

- (c) As an example, show that

$$h(x) = \frac{x^T x}{(\prod_{k=1}^n x_k)^{1/n}}, \quad \mathbf{dom} h = \mathbf{R}_{++}^n$$

is convex.

Solution.

- (a) Suppose $t_2 > t_1 > 0$ and $x/t_1 \in \mathbf{dom} f$, $x/t_2 \in \mathbf{dom} f$. We have $x/t_2 = \theta(x/t_1) + (1 - \theta)0$ where $\theta = t_1/t_2$. Hence, by Jensen's inequality,

$$f(x/t_2) \leq \frac{t_1}{t_2} f(x/t_1) + (1 - \frac{t_1}{t_2}) f(0) \leq \frac{t_1}{t_2} f(x/t_1)$$

if $f(0) \leq 0$. Therefore

$$t_2 f(x/t_2) \leq t_1 f(x/t_1).$$

If we assume that f is differentiable, we can also verify that the derivative of the perspective with respect to t is less than or equal to zero. We have

$$\frac{\partial}{\partial t} tf(x/t) = f(x/t) - \nabla f(x/t)^T (x/t)$$

This is less than or equal to zero, because, from convexity of f ,

$$0 \geq f(0) \geq f(x/t) + \nabla f(x/t)^T (0 - x/t).$$

- (b) This follows from a composition theorem: $tf(x/t)$ is convex in (x, t) and nonincreasing in t ; therefore if $g(x)$ is concave then $g(x)f(x/g(x))$ is convex. This composition rule is actually not mentioned in the lectures or the textbook, but it is easily derived as follows. Let us denote the perspective function $tf(x/t)$ by $F(x, t)$. Consider a convex combination $x = \theta u + (1 - \theta)v$ of two points in $\mathbf{dom} h$. Then

$$\begin{aligned} h(\theta u + (1 - \theta)v) &= F(\theta u + (1 - \theta)v, g(\theta u + (1 - \theta)v)) \\ &\leq F(\theta u + (1 - \theta)v, \theta g(u) + (1 - \theta)g(v)), \end{aligned}$$

because g is concave, and $F(x, t)$ is nonincreasing with respect to t . Convexity of F then gives

$$\begin{aligned} h(\theta u + (1 - \theta)v) &\leq \theta F(u, g(u)) + (1 - \theta)F(v, g(v)) \\ &= \theta h(u) + (1 - \theta)h(v). \end{aligned}$$

As an alternative proof, we can establish convexity of h directly from the definition. Consider two points $u, v \in \text{dom } g$, with $u/g(u) \in \text{dom } f$ and $v/g(v) \in \text{dom } f$. We take a convex combination $x = \theta u + (1 - \theta)v$. This point lies in $\text{dom } g$ because $\text{dom } g$ is convex. Also,

$$\begin{aligned} \frac{x}{g(x)} &= \frac{\theta g(u)}{g(x)} \frac{u}{g(u)} + \frac{(1 - \theta)g(v)}{g(x)} \frac{v}{g(v)} \\ &= \mu_1 \frac{u}{g(u)} + \mu_2 \frac{v}{g(v)} + \mu_3 \cdot 0 \end{aligned}$$

where

$$\mu_1 = \frac{\theta g(u)}{g(x)}, \quad \mu_2 = \frac{(1 - \theta)g(v)}{g(x)}, \quad \mu_3 = 1 - \mu_1 - \mu_2.$$

These coefficients are nonnegative and add up to one because $g(x) > 0$ on its domain, and $\theta g(u) + (1 - \theta)g(v) \leq g(x)$ by concavity of g . Therefore $x/g(x)$ is a convex combination of three points in $\text{dom } f$, and therefore in $\text{dom } f$ itself. This shows that $\text{dom } h$ is convex.

Next we verify Jensen's inequality:

$$\begin{aligned} f(x/g(x)) &\leq \mu_1 f(u/g(u)) + \mu_2 f(v/g(v)) + \mu_3 f(0) \\ &\leq \mu_1 f(u/g(u)) + \mu_2 f(v/g(v)) \end{aligned}$$

because $f(0) \leq 0$. Substituting the expressions for μ_1 and μ_2 we get

$$g(x)f(x/g(x)) \leq \theta g(u)f(u/g(u)) + (1 - \theta)g(v)f(v/g(v))$$

i.e., Jensen's inequality $h(x) \leq \theta h(u) + (1 - \theta)h(v)$.

(c) Apply part (b) to $f(x) = x^T x$ and $g(x) = (\prod_k x_k)^{1/n}$.

2.6 Perspective of log determinant. Show that $f(X, t) = nt \log t - t \log \det X$, with $\text{dom } f = \mathbf{S}_{++}^n \times \mathbf{R}_{++}$, is convex in (X, t) . Use this to show that

$$\begin{aligned} g(X) &= n(\mathbf{tr} X) \log(\mathbf{tr} X) - (\mathbf{tr} X)(\log \det X) \\ &= n \left(\sum_{i=1}^n \lambda_i \right) \left(\log \sum_{i=1}^n \lambda_i - \sum_{i=1}^n \log \lambda_i \right), \end{aligned}$$

where λ_i are the eigenvalues of X , is convex on \mathbf{S}_{++}^n .

Solution. This is the perspective function of $-\log \det X$:

$$f(X, t) = -t \log \det(X/t) = nt \log t - \log \det X.$$

Convexity of g follows from $g(X) = f(X, \mathbf{tr} X)$ and the fact that $\mathbf{tr} X$ is a linear function of X (and positive).

2.7 Pre-composition with a linear fractional mapping. Suppose $f : \mathbf{R}^m \rightarrow \mathbf{R}$ is convex, and $A \in \mathbf{R}^{m \times n}$, $b \in \mathbf{R}^m$, $c \in \mathbf{R}^n$, and $d \in \mathbf{R}$. Show that $g : \mathbf{R}^n \rightarrow \mathbf{R}$, defined by

$$g(x) = (c^T x + d)f((Ax + b)/(c^T x + d)), \quad \mathbf{dom} g = \{x \mid c^T x + d > 0\},$$

is convex.

Solution. g is just the composition of the perspective of f (which is convex) with the affine map that takes x to $(Ax + b, c^T x + d)$, and so is convex.

2.8 Scalar valued linear fractional functions. A function $f : \mathbf{R}^n \rightarrow \mathbf{R}$ is called *linear fractional* if it has the form $f(x) = (a^T x + b)/(c^T x + d)$, with $\mathbf{dom} f = \{x \mid c^T x + d > 0\}$. When is a linear fractional function convex? When is a linear fractional function quasiconvex?

Solution. Linear fractional functions are always quasiconvex, since the sublevel sets are convex, defined by one strict and one nonstrict linear inequality:

$$f(x) \leq t \iff c^T x + d > 0, \quad a^T x + b - t(c^T x + d) \leq 0.$$

To analyze convexity, we form the Hessian:

$$\nabla^2 f(x) = -(c^T x + d)^{-2}(ac^T + ca^T) + (a^T x + b)(c^T x + d)^{-3}cc^T.$$

First assume that a and c are not colinear. In this case, we can find x with $c^T x + d = 1$ (so, $x \in \mathbf{dom} f$) with $a^T x + b$ taking any desired value. By taking it as a large and negative, we see that the Hessian is not positive semidefinite, so f is not convex.

So for f to be convex, we must have a and c colinear. If c is zero, then f is affine (hence convex). Assume now that c is nonzero, and that $a = \alpha c$ for some $\alpha \in \mathbf{R}$. In this case, f reduces to

$$f(x) = \frac{\alpha c^T x + b}{c^T x + d} = \alpha + \frac{b - \alpha d}{c^T x + d},$$

which is convex if and only if $b \geq \alpha d$.

So a linear fractional function is convex only in some very special cases: it is affine, or a constant plus a nonnegative constant times the inverse of $c^T x + d$.

2.9 Show that the function

$$f(x) = \frac{\|Ax - b\|_2^2}{1 - x^T x}$$

is convex on $\{x \mid \|x\|_2 < 1\}$.

Solution. The epigraph is convex because

$$\frac{\|Ax - b\|_2^2}{t} \leq 1 - x^T x$$

defines a convex set.

Here's another solution: The function $\|Ax - b\|_2^2/u$ is convex in (x, u) , since it is the quadratic over linear function, pre-composed with an affine mapping. This function is decreasing in its second argument, so by a composition rule, we can replace the second argument with any concave function, and the result is convex. But $u = 1 = x^T x$ is concave, so we're done.

2.10 Weighted geometric mean. The geometric mean $f(x) = (\prod_k x_k)^{1/n}$ with $\mathbf{dom} f = \mathbf{R}_{++}^n$ is concave, as shown on page 74. Extend the proof to show that

$$f(x) = \prod_{k=1}^n x_k^{\alpha_k}, \quad \mathbf{dom} f = \mathbf{R}_{++}^n$$

is concave, where α_k are nonnegative numbers with $\sum_k \alpha_k = 1$.

Solution. The Hessian of f is

$$\nabla^2 f(x) = f(x) \left(q q^T - \mathbf{diag}(\alpha)^{-1} \mathbf{diag}(q)^2 \right)$$

where q is the vector $(\alpha_1/x_1, \dots, \alpha_n/x_n)$. We have

$$y^T \nabla^2 f(x) y = f(x) \left(\left(\sum_{k=1}^n \alpha_k y_k / x_k \right)^2 - \sum_{k=1}^n \alpha_k y_k^2 / x_k^2 \right) \leq 0$$

by applying the Cauchy-Schwarz inequality $(u^T v)^2 \leq (\|u\|_2 \|v\|_2)^2$ to the vectors

$$u = (\sqrt{\alpha_1} y_1 / x_1, \dots, \sqrt{\alpha_n} y_n / x_n), \quad v = (\sqrt{\alpha_1}, \dots, \sqrt{\alpha_n}).$$

2.11 Suppose that $f : \mathbf{R}^n \rightarrow \mathbf{R}$ is convex, and define

$$g(x, t) = f(x/t), \quad \mathbf{dom} g = \{(x, t) \mid x/t \in \mathbf{dom} f, t > 0\}.$$

Show that g is quasiconvex.

Solution. The γ -sublevel set of g is defined by

$$f(x, t) \leq \gamma, \quad t > 0, \quad x/t \in \mathbf{dom} f.$$

This is equivalent to

$$t f(x, t) \leq t \gamma, \quad t > 0, \quad x/t \in \mathbf{dom} f.$$

The function $t f(x, t)$, with domain $\{(x, t) \mid x/t \in \mathbf{dom} f, t > 0\}$, is the perspective function of g , and is convex. The first inequality above is convex, since the righthand side is an affine function of t (for fixed γ). So the γ -sublevel set of g is convex, and we're done.

2.12 Continued fraction function. Show that the function

$$f(x) = \cfrac{1}{x_1 - \cfrac{1}{x_2 - \cfrac{1}{x_3 - \cfrac{1}{x_4}}}}$$

defined where every denominator is positive, is convex and decreasing. (There is nothing special about $n = 4$ here; the same holds for any number of variables.)

Solution. We will use the composition rules and recursion. $g_4(x) = 1/x_4$ is clearly convex and decreasing in x_4 . The function $\frac{1}{x_3 - z}$ is convex in x_3 and z (over $\mathbf{dom} f$), and is decreasing in x_3 and

increasing in z ; it follows by the composition rules that $g_3(x) = \frac{1}{x_3 - g_4(x)}$ is convex and decreasing in all its variables. Repeating this argument for $g_k(x) = \frac{1}{x_k - g_{k+1}(x)}$ shows that f is convex and decreasing.

Here is another way: $g_1(x) = x_3 - 1/x_4$ is clearly concave and increasing in x_3 and x_4 . The function $x_2 - 1/z$ is concave and increasing in x_2 and z ; it follows by the composition rules that $g_2(x) = x_2 - 1/g_1(x)$ is concave and increasing. Repeating this argument shows that $-f$ is concave and increasing, so f is convex and decreasing.

2.13 *Circularly symmetric Huber function.* The scalar Huber function is defined as

$$f_{\text{hub}}(x) = \begin{cases} (1/2)x^2 & |x| \leq 1 \\ |x| - 1/2 & |x| > 1. \end{cases}$$

This convex function comes up in several applications, including robust estimation. This problem concerns generalizations of the Huber function to \mathbf{R}^n . One generalization to \mathbf{R}^n is given by $f_{\text{hub}}(x_1) + \dots + f_{\text{hub}}(x_n)$, but this function is not circularly symmetric, *i.e.*, invariant under transformation of x by an orthogonal matrix. A generalization to \mathbf{R}^n that *is* circularly symmetric is

$$f_{\text{cshub}}(x) = f_{\text{hub}}(\|x\|) = \begin{cases} (1/2)\|x\|_2^2 & \|x\|_2 \leq 1 \\ \|x\|_2 - 1/2 & \|x\|_2 > 1. \end{cases}$$

(The subscript stands for ‘circularly symmetric Huber function’.) Show that f_{cshub} is convex. Find the conjugate function f_{cshub}^* .

Solution. We can’t directly use the composition form given above, since f_{hub} is *not* nondecreasing. But we can write $f_{\text{cshub}} = h \circ g$, where $h : \mathbf{R} \rightarrow \mathbf{R}$ and $g : \mathbf{R}^n \rightarrow \mathbf{R}$ are defined as

$$h(x) = \begin{cases} 0 & x \leq 0 \\ x^2/2 & 0 \leq x \leq 1 \\ x - 1/2 & x > 1, \end{cases}$$

and $g(x) = \|x\|_2$. We can think of g as a version of the scalar Huber function, modified to be zero when its argument is negative. Clearly, g is convex and \tilde{h} is convex and increasing. Thus, from the composition rules we conclude that f_{cshub} is convex.

Now we will show that

$$f_{\text{cshub}}^*(y) = \begin{cases} (1/2)\|y\|_2^2 & \|y\|_2 \leq 1 \\ \infty & \text{otherwise.} \end{cases}$$

Suppose $\|y\|_2 > 1$. Taking $x = ty/\|y\|_2$, we see that for $t \geq 1$ we have

$$y^T x - f(x) = t\|y\|_2 - t + 1/2 = t(\|y\|_2 - 1) + 1/2.$$

Letting $t \rightarrow \infty$, we see that for any y with $\|y\|_2 > 1$, $\sup_x (y^T x - f(x)) = \infty$. Thus, $f_{\text{cshub}}^*(y) = \infty$ for $\|y\|_2 > 1$.

Now suppose $\|y\|_2 \leq 1$. We can write $f_{\text{cshub}}^*(y)$ as

$$f_{\text{cshub}}^*(y) = \max \left\{ \sup_{\|x\|_2 \leq 1} (y^T x - (1/2)\|x\|_2^2), \sup_{\|x\|_2 \geq 1} (y^T x - \|x\|_2 + 1/2) \right\}.$$

It is easy to show that $y^T x - (1/2)\|x\|_2^2$ is maximized over $\{x \mid \|x\|_2 \leq 1\}$ when $x = y$ (set the gradient of $y^T x - (1/2)\|x\|_2^2$ equal to zero). This gives

$$\sup_{\|x\|_2 \leq 1} (y^T x - (1/2)\|x\|_2^2) = (1/2)\|y\|_2^2.$$

To find $\sup_{\|x\|_2 \geq 1} (y^T x - \|x\|_2 + 1/2)$, notice that for $\|x\|_2 \geq 1$

$$y^T x - \|x\|_2 + 1/2 \leq \|y\|_2\|x\|_2 - \|x\|_2 + 1/2 = \|x\|_2(\|y\|_2 - 1) + 1/2 \leq \|y\|_2 - 1/2.$$

Here, the first inequality follows from Cauchy-Schwarz, and the second inequality follows from $\|y\|_2 \leq 1$ and $\|x\|_2 \geq 1$. Furthermore, if we choose $x = y/\|y\|_2$, then

$$y^T x - \|x\|_2 + 1/2 = \|y\|_2 - 1/2,$$

thus,

$$\sup_{\|x\|_2 \geq 1} (y^T x - \|x\|_2 + 1/2) = \|y\|_2 - 1/2.$$

For $\|y\|_2 \leq 1$

$$\sup_{\|x\|_2 \geq 1} (y^T x - \|x\|_2 + 1/2) = \|y\|_2 - 1/2 \leq (1/2)\|y\|_2^2 = \sup_{\|x\|_2 \leq 1} (y^T x - (1/2)\|x\|_2^2),$$

so we conclude that for $\|y\|_2 \leq 1$, $f_{\text{cshub}}^*(y) = (1/2)\|y\|_2^2$.

2.14 Reverse Jensen inequality. Suppose f is convex, $\lambda_1 > 0$, $\lambda_i \leq 0$, $i = 2, \dots, k$, and $\lambda_1 + \dots + \lambda_n = 1$, and let $x_1, \dots, x_n \in \text{dom } f$. Show that the inequality

$$f(\lambda_1 x_1 + \dots + \lambda_n x_n) \geq \lambda_1 f(x_1) + \dots + \lambda_n f(x_n)$$

always holds. *Hints.* Draw a picture for the $n = 2$ case first. For the general case, express x_1 as a convex combination of $\lambda_1 x_1 + \dots + \lambda_n x_n$ and x_2, \dots, x_n , and use Jensen's inequality.

Solution. Let $z = \lambda_1 x_1 + \dots + \lambda_n x_n$, with $\lambda_1 > 0$, $\lambda_i \leq 0$, $i = 2, \dots, n$, and $\lambda_1 + \dots + \lambda_n = 1$. Then we have

$$x_1 = \theta_0 z + \theta_2 x_2 + \dots + \theta_n x_n,$$

where

$$\theta_i = -\lambda_i/\lambda_1, \quad i = 2, \dots, n,$$

and $\theta_0 = 1/\lambda_1$. Since $\lambda_1 > 1$, we see that $\theta_0 > 0$; from $\lambda_i \leq 0$ we get $\theta_i \geq 0$ for $i = 2, \dots, n$. Simple algebra shows that $\theta_1 + \dots + \theta_n = 1$. From Jensen's inequality we have

$$f(x_1) \leq \theta_0 f(z) + \theta_2 f(x_2) + \dots + \theta_n f(x_n),$$

so

$$f(z) \geq \frac{1}{\theta_0} f(x_1) + \frac{\theta_2}{\theta_0} f(x_2) + \dots + \frac{\theta_n}{\theta_0} f(x_n),$$

Substituting for θ_i this becomes the inequality we want,

$$f(z) \geq \lambda_1 f(x_1) + \dots + \lambda_n f(x_n).$$

2.15 Monotone extension of a convex function. Suppose $f : \mathbf{R}^n \rightarrow \mathbf{R}$ is convex. Recall that a function $h : \mathbf{R}^n \rightarrow \mathbf{R}$ is monotone nondecreasing if $h(x) \geq h(y)$ whenever $x \succeq y$. The *monotone extension* of f is defined as

$$g(x) = \inf_{z \succeq 0} f(x + z).$$

(We will assume that $g(x) > -\infty$.) Show that g is convex and monotone nondecreasing, and satisfies $g(x) \leq f(x)$ for all x . Show that if h is any other convex function that satisfies these properties, then $h(x) \leq g(x)$ for all x . Thus, g is the maximum convex monotone underestimator of f .

Remark. For simple functions (say, on \mathbf{R}) it is easy to work out what g is, given f . On \mathbf{R}^n , it can be very difficult to work out an explicit expression for g . However, systems such as CVX can immediately handle functions such as g , defined by partial minimization.

Solution. The function $f(x + z)$ is jointly convex in x and z ; partial minimization over z in the nonnegative orthant yields g , which therefore is convex.

To show that g is monotone, suppose that $x \succeq y$. Then we have $x = y + \hat{z}$, where $\hat{z} = x - y \succeq 0$, and so

$$g(y) = \inf_{z \succeq 0} f(y + z) \leq f(y + \hat{z}) = f(x).$$

To show that $g(x) \leq f(x)$, we use

$$g(x) = \inf_{z \succeq 0} f(x + z) \leq f(x).$$

Now suppose that h is monotone, convex, and satisfies $h(x) \leq f(x)$ for all x . Then we have, for all x and z , $h(x + z) \leq f(x + z)$. Taking the infimum over $z \succeq 0$, we obtain

$$\inf_{z \succeq 0} h(x + z) \leq \inf_{z \succeq 0} f(x + z).$$

Since h is monotone nondecreasing, the lefthand side is $h(x)$; the righthand side is $g(x)$.

2.16 Circularly symmetric convex functions. Suppose $f : \mathbf{R}^n \rightarrow \mathbf{R}$ is convex and symmetric with respect to rotations, i.e., $f(x)$ depends only on $\|x\|_2$. Show that f must have the form $f(x) = \phi(\|x\|_2)$, where $\phi : \mathbf{R} \rightarrow \mathbf{R}$ is nondecreasing and convex, with $\text{dom } f = \mathbf{R}$. (Conversely, any function of this form is symmetric and convex, so this form characterizes such functions.)

Solution. Define $\psi(a) = f(ae_1)$, where e_1 is the first standard unit vector, and $a \in \mathbf{R}$. ψ is a convex function, and it is symmetric: $\psi(-a) = f(-ae_1) = f(ae_1) = \psi(a)$, since $\|-ae_1\|_2 = \|ae_1\|_2$. A symmetric convex function on \mathbf{R} must have its minimum at 0; for suppose that $\psi(a) < \psi(0)$. By Jensen's inequality, $\psi(0) \leq (1/2)\psi(-a) + (1/2)\psi(a) = \psi(a)$, a contradiction. Therefore ψ is nondecreasing for $a \geq 0$. Now we define $\phi(a) = \psi(a)$ for $a \geq 0$ and $\phi(a) = \psi(0)$ for $a < 0$. ψ is convex and nondecreasing. Evidently we have $f(x) = \phi(\|x\|_2)$, so we're done.

2.17 Infimal convolution. Let f_1, \dots, f_m be convex functions on \mathbf{R}^n . Their *infimal convolution*, denoted $g = f_1 \diamond \dots \diamond f_m$ (several other notations are also used), is defined as

$$g(x) = \inf\{f_1(x_1) + \dots + f_m(x_m) \mid x_1 + \dots + x_m = x\},$$

with the natural domain (*i.e.*, defined by $g(x) < \infty$). In one simple interpretation, $f_i(x_i)$ is the cost for the i th firm to produce a mix of products given by x_i ; $g(x)$ is then the optimal cost obtained if the firms can freely exchange products to produce, all together, the mix given by x . (The name ‘convolution’ presumably comes from the observation that if we replace the sum above with the product, and the infimum above with integration, then we obtain the normal convolution.)

- (a) Show that g is convex.
- (b) Show that $g^* = f_1^* + \dots + f_m^*$. In other words, the conjugate of the infimal convolution is the sum of the conjugates.

Solution.

- (a) We can express g as

$$g(x) = \inf_{x_1, \dots, x_m} (f_1(x_1) + \dots + f_m(x_m) + \phi(x_1, \dots, x_m, x)),$$

where $\phi(x_1, \dots, x_m, x)$ is 0 when $x_1 + \dots + x_m = x$, and ∞ otherwise. The function on the righthand side above is convex in x_1, \dots, x_m, x , so by the partial minimization rule, so is g .

- (b) We have

$$\begin{aligned} g^*(y) &= \sup_x (y^T x - f(x)) \\ &= \sup_x \left(y^T x - \inf_{x_1 + \dots + x_m = x} f_1(x_1) + \dots + f_m(x_m) \right) \\ &= \sup_{x=x_1+\dots+x_m} \left(y^T x_1 - f_1(x_1) + \dots + y^T x_m - f_m(x_m) \right), \end{aligned}$$

where we use the fact that $(-\inf S)$ is the same as $(\sup -S)$. The last line means we are to take the supremum over all x and all x_1, \dots, x_m that sum to x . But this is the same as just taking the supremum over all x_1, \dots, x_m , so we get

$$\begin{aligned} g^*(y) &= \sup_{x_1, \dots, x_m} \left(y^T x_1 - f_1(x_1) + \dots + y^T x_m - f_m(x_m) \right) \\ &= \sup_{x_1} (y^T x_1 - f_1(x_1)) + \dots + \sup_{x_m} (y^T x_m - f_m(x_m)) \\ &= f_1^*(y) + \dots + f_m^*(y). \end{aligned}$$

2.18 Conjugate of composition of convex and linear function. Suppose $A \in \mathbf{R}^{m \times n}$ with $\text{rank } A = m$, and g is defined as $g(x) = f(Ax)$, where $f : \mathbf{R}^m \rightarrow \mathbf{R}$ is convex. Show that

$$g^*(y) = f^*((A^\dagger)^T y), \quad \text{dom}(g^*) = A^T \text{dom}(f^*),$$

where $A^\dagger = (AA^T)^{-1}A$ is the pseudo-inverse of A . (This generalizes the formula given on page 95 for the case when A is square and invertible.)

Solution. Let $z = (A^\dagger)^T y$, so $y = A^T z$. Then we have

$$\begin{aligned} g^*(y) &= \sup_x (y^T x - f(Ax)) \\ &= \sup_x (z^T (Ax) - f(Ax)) \end{aligned}$$

$$\begin{aligned}
&= \sup_w (z^T w - f(w)) \\
&= f^*(z) \\
&= f^*((A^\dagger)^T y).
\end{aligned}$$

Now $y \in \text{dom}(g^*)$ if and only if $(A^\dagger)^T y = z \in \text{dom}(f^*)$. But since $y = A^T z$, we see that this is equivalent to $y \in A^T \text{dom}(f^*)$.

2.19 [?, p104] Suppose $\lambda_1, \dots, \lambda_n$ are positive. Show that the function $f : \mathbf{R}^n \rightarrow \mathbf{R}$, given by

$$f(x) = \prod_{i=1}^n (1 - e^{-x_i})^{\lambda_i},$$

is concave on

$$\text{dom } f = \left\{ x \in \mathbf{R}_{++}^n \mid \sum_{i=1}^n \lambda_i e^{-x_i} \leq 1 \right\}.$$

Hint. The Hessian is given by

$$\nabla^2 f(x) = f(x)(yy^T - \text{diag}(z))$$

where $y_i = \lambda_i e^{-x_i} / (1 - e^{-x_i})$ and $z_i = y_i / (1 - e^{-x_i})$.

Solution. We'll use Cauchy-Schwarz to show that the Hessian is negative semidefinite. The Hessian is given by

$$\nabla^2 f(x) = f(x)(yy^T - \text{diag}(z))$$

where $y_i = \lambda_i e^{-x_i} / (1 - e^{-x_i})$ and $z_i = y_i^2 / (\lambda_i e^{-x_i})$.

For any $v \in \mathbf{R}^n$, we can show

$$v^T \nabla^2 f(x) v = \left(\sum_{i=1}^n v_i y_i \right)^2 - \sum_{i=1}^n \frac{v_i^2 y_i^2}{\lambda_i e^{-x_i}} \leq 0,$$

by applying the Cauchy-Schwarz inequality, $(a^T b)^2 \leq (a^T a)(b^T b)$, to the vectors with components $a_i = v_i y_i / \sqrt{\lambda_i e^{-x_i}}$ and $b_i = \sqrt{\lambda_i e^{-x_i}}$. The result follows because $(b^T b) \leq 1$ on $\text{dom } f$. Thus the Hessian is negative semidefinite.

2.20 Show that the following functions $f : \mathbf{R}^n \rightarrow \mathbf{R}$ are convex.

- (a) The difference between the maximum and minimum value of a polynomial on a given interval, as a function of its coefficients:

$$f(x) = \sup_{t \in [a,b]} p(t) - \inf_{t \in [a,b]} p(t) \quad \text{where } p(t) = x_1 + x_2 t + x_3 t^2 + \cdots + x_n t^{n-1}.$$

a, b are real constants with $a < b$.

- (b) The ‘exponential barrier’ of a set of inequalities:

$$f(x) = \sum_{i=1}^m e^{-1/f_i(x)}, \quad \text{dom } f = \{x \mid f_i(x) < 0, i = 1, \dots, m\}.$$

The functions f_i are convex.

(c) The function

$$f(x) = \inf_{\alpha > 0} \frac{g(y + \alpha x) - g(y)}{\alpha}$$

if g is convex and $y \in \text{dom } g$. (It can be shown that this is the directional derivative of g at y in the direction x .)

Solution.

- (a) f is the difference of a convex and a concave function. The first term is convex, because it is the supremum of a family of linear functions of x . The second term is concave because it is the infimum of a family of linear functions.
- (b) $h(u) = \exp(1/u)$ is convex and decreasing on \mathbf{R}_{++} :

$$h'(u) = -\frac{1}{u^2}e^{1/u}, \quad h''(u) = \frac{2}{u^3}e^{1/u} + \frac{1}{u^4}e^{1/u}.$$

Therefore the composition $h(-f_i(x)) = \exp(-1/f_i(x))$ is convex if f_i is convex.

(c) Can be written as

$$f(x) = \inf_{t > 0} t \left(g(y + \frac{1}{t}x) - g(y) \right),$$

the infimum over t of the perspective of the convex function $g(y + x) - g(y)$.

2.21 Symmetric convex functions of eigenvalues. A function $f : \mathbf{R}^n \rightarrow \mathbf{R}$ is said to be *symmetric* if it is invariant with respect to a permutation of its arguments, i.e., $f(x) = f(Px)$ for any permutation matrix P . An example of a symmetric function is $f(x) = \log(\sum_{k=1}^n \exp x_k)$.

In this problem we show that if $f : \mathbf{R}^n \rightarrow \mathbf{R}$ is *convex* and *symmetric*, then the function $g : \mathbf{S}^n \rightarrow \mathbf{R}$ defined as $g(X) = f(\lambda(X))$ is convex, where $\lambda(X) = (\lambda_1(X), \lambda_2(X), \dots, \lambda_n(X))$ is the vector of eigenvalues of X . This implies, for example, that the function

$$g(X) = \log \mathbf{tr} e^X = \log \sum_{k=1}^n e^{\lambda_k(X)}$$

is convex on \mathbf{S}^n .

- (a) A square matrix S is *doubly stochastic* if its elements are nonnegative and all row sums and column sums are equal to one. It can be shown that every doubly stochastic matrix is a convex combination of permutation matrices.

Show that if f is convex and symmetric and S is doubly stochastic, then

$$f(Sx) \leq f(x).$$

- (b) Let $Y = Q \mathbf{diag}(\lambda) Q^T$ be an eigenvalue decomposition of $Y \in \mathbf{S}^n$ with Q orthogonal. Show that the $n \times n$ matrix S with elements $S_{ij} = Q_{ij}^2$ is doubly stochastic and that $\mathbf{diag}(Y) = S\lambda$.
- (c) Use the results in parts (a) and (b) to show that if f is convex and symmetric and $X \in \mathbf{S}^n$, then

$$f(\lambda(X)) = \sup_{V \in \mathcal{V}} f(\mathbf{diag}(V^T X V))$$

where \mathcal{V} is the set of $n \times n$ orthogonal matrices. Show that this implies that $f(\lambda(X))$ is convex in X .

Solution.

(a) Suppose S is expressed as a convex combination of permutation matrices:

$$S = \sum_k \theta_k P_k$$

with $\theta_k \geq 0$, $\sum_k \theta_k = 1$, and P_k permutation matrices. From convexity and symmetry of f ,

$$f(Sx) = f\left(\sum_k \theta_k P_k x\right) \leq \sum_k \theta_k f(P_k x) = \sum_k \theta_k f(x) = f(x).$$

(b) From $X = Q \mathbf{diag}(\lambda) Q^T$,

$$X_{ii} = \sum_{j=1}^n Q_{ij}^2 \lambda_j.$$

From $QQ^T = I$, we have $\sum_j Q_{ij}^2 = 1$. From $Q^T Q = I$, we have $\sum_i Q_{ij}^2 = 1$.

(c) Combining the results in parts (a) and (b), we conclude that for any symmetric X , the inequality

$$f(\mathbf{diag}(X)) \leq f(\lambda(X))$$

holds. Moreover, if V is orthogonal, then $\lambda(X) = \lambda(V^T X V)$. Therefore also

$$f(\mathbf{diag}(V^T X V)) \leq f(\lambda(X))$$

for all orthogonal V , with equality if $V = Q$. In other words

$$f(\lambda(X)) = \sup_{V \in \mathcal{V}} f(\mathbf{diag}(V^T X V)).$$

This shows that $f(\lambda(X))$ is convex because it is the supremum of a family of convex functions of X .

2.22 Convexity of nonsymmetric matrix fractional function. Consider the function $f : \mathbf{R}^{n \times n} \times \mathbf{R}^n \rightarrow \mathbf{R}$, defined by

$$f(X, y) = y^T X^{-1} y, \quad \mathbf{dom} f = \{(X, y) \mid X + X^T \succ 0\}.$$

When this function is restricted to $X \in \mathbf{S}^n$, it is convex.

Is f convex? If so, prove it. If not, give a (simple) counterexample.

Solution. The function is not convex. Restrict the function f to $g(s) = f(X, y)$, with

$$X = \begin{bmatrix} 1 & s \\ -s & 1 \end{bmatrix}, \quad y = \begin{bmatrix} 1 \\ 1 \end{bmatrix},$$

and $s \in \mathbf{R}$. (The domain of g is \mathbf{R} .) If f is convex then so is g . But we have

$$g(s) = \frac{2}{1 + s^2},$$

which is certainly not convex.

For a very specific counterexample, take (say) $s = +1$ and $s = -1$. Then we have $g(-1) = 1$, $g(+1) = 1$ and

$$g((-1+1)/2) = g(0) = 2 \not\leq (g(-1) + g(+1))/2 = 1.$$

2.23 Show that the following functions $f : \mathbf{R}^n \rightarrow \mathbf{R}$ are convex.

- (a) $f(x) = -\exp(-g(x))$ where $g : \mathbf{R}^n \rightarrow \mathbf{R}$ has a convex domain and satisfies

$$\begin{bmatrix} \nabla^2 g(x) & \nabla g(x) \\ \nabla g(x)^T & 1 \end{bmatrix} \succeq 0$$

for $x \in \text{dom } g$.

- (b) The function

$$f(x) = \max \{ \|APx - b\| \mid P \text{ is a permutation matrix}\}$$

with $A \in \mathbf{R}^{m \times n}$, $b \in \mathbf{R}^m$.

Solution.

- (a) The gradient and Hessian of f are

$$\begin{aligned} \nabla f(x) &= e^{-g(x)} \nabla g(x) \\ \nabla^2 f(x) &= e^{-g(x)} \nabla^2 g(x) - e^{-g(x)} \nabla g(x) \nabla g(x)^T \\ &= e^{-g(x)} (\nabla^2 g(x) - \nabla g(x) \nabla g(x)^T) \\ &\succeq 0. \end{aligned}$$

- (b) f is the maximum of convex functions $\|APx - b\|$, parameterized by P .

2.24 Convex hull of functions. Suppose g and h are convex functions, bounded below, with $\text{dom } g = \text{dom } h = \mathbf{R}^n$. The convex hull function of g and h is defined as

$$f(x) = \inf \{ \theta g(y) + (1 - \theta)h(z) \mid \theta y + (1 - \theta)z = x, 0 \leq \theta \leq 1 \},$$

where the infimum is over θ, y, z . Show that the convex hull of h and g is convex. Describe $\text{epi } f$ in terms of $\text{epi } g$ and $\text{epi } h$.

Solution. We note that $f(x) \leq t$ if and only if there exist $\theta \in [0, 1]$, y, z, t_1, t_2 such that

$$g(y) \leq t_1, \quad h(z) \leq t_2, \quad \theta y + (1 - \theta)z = x, \quad \theta t_1 + (1 - \theta)t_2 = t.$$

Thus

$$\text{epi } f = \text{conv}(\text{epi } g \cup \text{epi } h),$$

i.e., $\text{epi } f$ is the convex hull of the union of the epigraphs of g and h . This shows that f is convex.

As an alternative proof, we can make a change of variable $\tilde{y} = \theta y$, $\tilde{z} = (1 - \theta)z$ in the minimization problem in the definition, and note that $f(x)$ is the optimal value of

$$\begin{aligned} &\text{minimize} && \theta g(y/\theta) + (1 - \theta)h(z/(1 - \theta)) \\ &\text{subject to} && y + z = x \\ &&& 0 \leq \theta \leq 1, \end{aligned}$$

with variables θ, y, z . This is a convex problem, and therefore the optimal value is a convex function of the righthand side x .

2.25 Show that a function $f : \mathbf{R} \rightarrow \mathbf{R}$ is convex if and only if $\mathbf{dom} f$ is convex and

$$\det \begin{bmatrix} 1 & 1 & 1 \\ x & y & z \\ f(x) & f(y) & f(z) \end{bmatrix} \geq 0$$

for all $x, y, z \in \mathbf{dom} f$ with $x < y < z$.

Solution.

$$\begin{aligned} \det \begin{bmatrix} 1 & 1 & 1 \\ t_1 & t_2 & t_3 \\ f(t_1) & f(t_2) & f(t_3) \end{bmatrix} &= \det \begin{bmatrix} 1 & 1 & 1 \\ t_1 & t_2 & t_3 \\ f(t_1) & f(t_2) & f(t_3) \end{bmatrix} \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 & 0 \\ t_1 & t_2 - t_1 & t_3 - t_2 \\ f(t_1) & f(t_2) - f(t_1) & f(t_3) - f(t_2) \end{bmatrix} \\ &= (t_2 - t_1)(f(t_3) - f(t_2)) - (t_3 - t_2)(f(t_2) - f(t_1)). \end{aligned}$$

This is nonnegative if and only if

$$\frac{t_3 - t_1}{(t_2 - t_1)(t_3 - t_2)} f(t_2) \leq \frac{1}{t_2 - t_1} f(t_1) + \frac{1}{t_3 - t_2} f(t_3).$$

This is Jensen's inequality

$$f(\theta t_1 + (1 - \theta)t_3) \leq \theta f(t_1) + (1 - \theta)f(t_3)$$

with

$$\theta = \frac{t_2 - t_1}{t_3 - t_1}, \quad 1 - \theta = \frac{t_3 - t_2}{t_3 - t_1}.$$

2.26 Generalization of the convexity of $\log \det X^{-1}$. Let $P \in \mathbf{R}^{n \times m}$ have rank m . In this problem we show that the function $f : \mathbf{S}^n \rightarrow \mathbf{R}$, with $\mathbf{dom} f = \mathbf{S}_{++}^n$, and

$$f(X) = \log \det(P^T X^{-1} P)$$

is convex. To prove this, we assume (without loss of generality) that P has the form

$$P = \begin{bmatrix} I \\ 0 \end{bmatrix},$$

where I . The matrix $P^T X^{-1} P$ is then the leading $m \times m$ principal submatrix of X^{-1} .

- (a) Let Y and Z be symmetric matrices with $0 \prec Y \preceq Z$. Show that $\det Y \leq \det Z$.
- (b) Let $X \in \mathbf{S}_{++}^n$, partitioned as

$$X = \begin{bmatrix} X_{11} & X_{12} \\ X_{12}^T & X_{22} \end{bmatrix},$$

with $X_{11} \in \mathbf{S}^m$. Show that the optimization problem

$$\begin{aligned} & \text{minimize} && \log \det Y^{-1} \\ & \text{subject to} && \begin{bmatrix} Y & 0 \\ 0 & 0 \end{bmatrix} \preceq \begin{bmatrix} X_{11} & X_{12} \\ X_{12}^T & X_{22} \end{bmatrix}, \end{aligned}$$

with variable $Y \in \mathbf{S}^m$, has the solution

$$Y = X_{11} - X_{12}X_{22}^{-1}X_{12}^T.$$

(As usual, we take \mathbf{S}_{++}^m as the domain of $\log \det Y^{-1}$.)

Hint. Use the Schur complement characterization of positive definite block matrices (page 651 of the book): if $C \succ 0$ then

$$\begin{bmatrix} A & B \\ B^T & C \end{bmatrix} \succeq 0$$

if and only if $A - BC^{-1}B^T \succeq 0$.

- (c) Combine the result in part (b) and the minimization property (page 3-19, lecture notes) to show that the function

$$f(X) = \log \det(X_{11} - X_{12}X_{22}^{-1}X_{12}^T)^{-1},$$

with $\mathbf{dom} f = \mathbf{S}_{++}^n$, is convex.

- (d) Show that $(X_{11} - X_{12}X_{22}^{-1}X_{12}^T)^{-1}$ is the leading $m \times m$ principal submatrix of X^{-1} , i.e.,

$$(X_{11} - X_{12}X_{22}^{-1}X_{12}^T)^{-1} = P^T X^{-1} P.$$

Hence, the convex function f defined in part (c) can also be expressed as $f(X) = \log \det(P^T X^{-1} P)$.

Hint. Use the formula for the inverse of a symmetric block matrix:

$$\begin{bmatrix} A & B \\ B^T & C \end{bmatrix}^{-1} = \begin{bmatrix} 0 & 0 \\ 0 & C^{-1} \end{bmatrix} + \begin{bmatrix} -I \\ C^{-1}B^T \end{bmatrix} (A - BC^{-1}B^T)^{-1} \begin{bmatrix} -I \\ C^{-1}B^T \end{bmatrix}^T$$

if C and $A - BC^{-1}B^T$ are invertible.

Solution.

- (a) $Y \preceq Z$ if and only if $Y^{-1/2}ZY^{-1/2} \succeq 0$, which implies $\det(Y^{-1/2}ZY^{-1/2}) = \det Y^{-1} \det Z = \det Z / \det Y \geq 0$.
- (b) The optimal Y maximizes $\det Y$ subject to the constraint

$$\begin{bmatrix} X_{11} & X_{12} \\ X_{12}^T & X_{22} \end{bmatrix} - \begin{bmatrix} Y & 0 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} X_{11} - Y & X_{12} \\ X_{12}^T & X_{22} \end{bmatrix} \succeq 0.$$

By the Schur complement property in the hint this inequality holds if and only if

$$Y \preceq X_{11} - X_{12}X_{22}^{-1}X_{12}^T$$

and this implies $\det Y \leq \det(X_{11} - X_{12}X_{22}^{-1}X_{12}^T)$. Therefore $Y = X_{11} - X_{12}X_{22}^{-1}X_{12}^T$ is optimal.

- (c) Define a function $F : \mathbf{S}^n \times \mathbf{S}^m \rightarrow \mathbf{R}$ with $F(X, Y) = \log \det Y^{-1}$ on the domain

$$\mathbf{dom} F = \left\{ (X, Y) \mid Y \succ 0, \begin{bmatrix} X_{11} & X_{12} \\ X_{12}^T & X_{22} \end{bmatrix} \succeq \begin{bmatrix} Y & 0 \\ 0 & 0 \end{bmatrix} \right\}.$$

This function is convex because its domain is convex and $\log \det Y^{-1}$ is convex on the set of positive definite Y . In part (b) we proved that

$$f(X) = \inf_Y F(X, Y)$$

and convexity follows from the minimization property.

- (d) The formula for the inverse shows that $(A - BC^{-1}B^T)^{-1}$ is the 1,1 block of the inverse of the block matrix.

2.27 Functions of a random variable with log-concave density. Suppose the random variable X on \mathbf{R}^n has log-concave density, and let $Y = g(X)$, where $g : \mathbf{R}^n \rightarrow \mathbf{R}$. For each of the following statements, either give a counterexample, or show that the statement is true.

- (a) If g is affine and not constant, then Y has log-concave density.
- (b) If g is convex, then $\mathbf{prob}(Y \leq a)$ is a log-concave function of a .
- (c) If g is concave, then $\mathbf{E}((Y - a)_+)$ is a convex and log-concave function of a . (This quantity is called the tail expectation of Y ; you can assume it exists. We define $(s)_+$ as $(s)_+ = \max\{s, 0\}$.)

Solution.

- (a) This one is **true**. Let p be the density of X , and let $g(x) = c^T x + d$, with $c \neq 0$ (otherwise g would be constant). Since g is not constant, we conclude that Y has a density p_Y .

With $\delta a > 0$, define the function

$$h(x, a) = \begin{cases} 1 & a \leq g(x) \leq a + \delta a \\ 0 & \text{otherwise,} \end{cases}$$

which is the $0 - 1$ indicator function of the convex set $\{(x, a) \mid a \leq g(x) \leq a + \delta a\}$. The $0 - 1$ indicator function of a convex set is log-concave, so by the integration rule it follows that

$$\int p(x)h(x, a) dx = \mathbf{E} h(X, a) = \mathbf{prob}(a \leq Y \leq a + \delta a)$$

is log-concave in a . It follows that

$$\frac{\mathbf{prob}(a \leq Y \leq a + \delta a)}{\delta a}$$

is log-concave (since $\delta a > 0$). Taking $\delta a \rightarrow 0$, this converges to $p_Y(a)$, which we conclude is log-concave.

- (b) This one is **true**. Here we define the function

$$h(x, a) = \begin{cases} 1 & g(x) \leq a \\ 0 & \text{otherwise,} \end{cases}$$

which is the 0–1 indicator function of the convex set $\text{epi } g = \{(x, a) \mid g(x) \leq a\}$, and therefore log-concave. By the integration rule we get that

$$\int p(x)h(x, a) dx = \mathbf{E} h(X, a) = \mathbf{prob}(Y \leq a)$$

is log-concave in a .

If we assume that g is concave, and we switch the inequality, we conclude that $\mathbf{prob}(Y \geq a)$ is log-concave in a . (We'll use this below.)

- (c) This one is **true**. Convexity of the tail expectation holds for *any* random variable, so it has nothing to do with g , and it has nothing to do with log-concavity of the density of X . For *any* random variable Y on \mathbf{R} , we have

$$\frac{d}{da} \mathbf{E}(Y - a)_+ = -\mathbf{prob}(Y \geq a).$$

The righthand side is nondecreasing in a , so the tail expectation has nondecreasing derivative, which means it is a convex function.

Now let's show that the tail expectation is log-concave. One simple method is to use the formula above to note that

$$\mathbf{E}(Y - a)_+ = \int_a^\infty \mathbf{prob}(Y \geq b) db.$$

The integration rule for log-concave functions tells us that this is log-concave.

We can also give a direct proof following the style of the ones given above. We define g as $h(x, a) = (g(x) - a)_+$. This function is log-concave. First, its domain is $\{(x, a) \mid g(x) > a\}$, which is convex. Concavity of $\log h(x, a) = \log(g(x) - a)$ follows from the composition rule: \log is concave and increasing, and $g(x) - a$ is concave in (x, a) . So by the integration rule we get

$$\int p(x)h(x, a) dx = \mathbf{E}(g(x) - a)_+$$

is log-concave in a .

2.28 Majorization. Define C as the set of all permutations of a given n -vector a , *i.e.*, the set of vectors $(a_{\pi_1}, a_{\pi_2}, \dots, a_{\pi_n})$ where $(\pi_1, \pi_2, \dots, \pi_n)$ is one of the $n!$ permutations of $(1, 2, \dots, n)$.

- (a) The support function of C is defined as $S_C(y) = \max_{x \in C} y^T x$. Show that

$$S_C(y) = a_{[1]}y_{[1]} + a_{[2]}y_{[2]} + \dots + a_{[n]}y_{[n]}.$$

($u_{[1]}, u_{[2]}, \dots, u_{[n]}$ denote the components of an n -vector u in nonincreasing order.)

Hint. To find the maximum of $y^T x$ over $x \in C$, write the inner product as

$$\begin{aligned} y^T x &= (y_1 - y_2)x_1 + (y_2 - y_3)(x_1 + x_2) + (y_3 - y_4)(x_1 + x_2 + x_3) + \dots \\ &\quad + (y_{n-1} - y_n)(x_1 + x_2 + \dots + x_{n-1}) + y_n(x_1 + x_2 + \dots + x_n) \end{aligned}$$

and assume that the components of y are sorted in nonincreasing order.

- (b) Show that x satisfies $x^T y \leq S_C(y)$ for all y if and only if

$$s_k(x) \leq s_k(a), \quad k = 1, \dots, n-1, \quad s_n(x) = s_n(a),$$

where s_k denotes the function $s_k(x) = x_{[1]} + x_{[2]} + \dots + x_{[k]}$. When these inequalities hold, we say the vector a *majorizes* the vector x .

- (c) Conclude from this that the conjugate of S_C is given by

$$S_C^*(x) = \begin{cases} 0 & \text{if } x \text{ is majorized by } a \\ +\infty & \text{otherwise.} \end{cases}$$

Since S_C^* is the indicator function of the convex hull of C , this establishes the following result: x is a convex combination of the permutations of a if and only if a majorizes x .

Solution.

- (a) Suppose y is sorted. From the expression of the inner product it is clear that the permutation of a that maximizes the inner product with y is $x_k = a_{[k]}$, $k = 1, \dots, n$.
- (b) We first show that if a majorizes x , then $x^T y \leq S_C(y)$ for all y . Note that if x is majorized by a , then all permutations of x are majorized by a , so we can assume that the components of y are sorted in nonincreasing order. Using the results from part a,

$$\begin{aligned} S_C(y) - x^T y &= (y_1 - y_2)(s_1(a) - x_1) + (y_2 - y_3)(s_2(a) - x_1 - x_2) + \dots \\ &\quad + (y_{n-1} - y_n)(s_{n-1}(a) - x_1 - \dots - x_{n-1}) + y_n(s_n(a) - x_1 - \dots - x_n) \\ &\geq (y_1 - y_2)(s_1(a) - s_1(x)) + (y_2 - y_3)(s_2(a) - s_2(x)) + \dots \\ &\quad + (y_{n-1} - y_n)(s_{n-1}(a) - s_{n-1}(x)) + y_n(s_n(a) - s_n(x)) \\ &\geq 0. \end{aligned}$$

Next, we show that the conditions are necessary. We distinguish two cases.

- Suppose $s_k(x) > s_k(a)$ for some $k < n$. Assume the components of x are sorted in nonincreasing order and choose

$$y_1 = \dots = y_{k-1} = 1, \quad y_k = \dots = y_n = 0.$$

Then $S_C(y) - x^T y = s_k(a) - s_k(x) < 0$.

- Suppose $s_n(x) \neq s_n(a)$. Choose $y = \mathbf{1}$ if $s_n(x) > s_n(a)$ and $y = -\mathbf{1}$ if $s_n(x) < s_n(a)$. We have $S_C(y) - x^T y = y_1(s_n(a) - s_n(x)) < 0$.

- (c) The expression for the conjugate follows from the fact that if $S_C(y) - x^T y$ is positive for some y then it is unbounded above, and if x is majorized by a then $x = 0$ is the optimum.

- 2.29 Convexity of products of powers.** This problem concerns the product of powers function $f : \mathbf{R}_{++}^n \rightarrow \mathbf{R}$ given by $f(x) = x_1^{\theta_1} \cdots x_n^{\theta_n}$, where $\theta \in \mathbf{R}^n$ is a vector of powers. We are interested in finding values of θ for which f is convex or concave. You already know a few, for example when $n = 2$ and $\theta = (2, -1)$, f is convex (the quadratic-over-linear function), and when $\theta = (1/n)\mathbf{1}$, f is concave

(geometric mean). Of course, if $n = 1$, f is convex when $\theta \geq 1$ or $\theta \leq 0$, and concave when $0 \leq \theta \leq 1$.

Show each of the statements below. We will not read long or complicated proofs, or ones that involve Hessians. We are looking for short, snappy ones, that (where possible) use composition rules, perspective, partial minimization, or other operations, together with known convex or concave functions, such as the ones listed in the previous paragraph. Feel free to use the results of earlier statements in later ones.

- (a) When $n = 2$, $\theta \succeq 0$, and $\mathbf{1}^T \theta = 1$, f is concave.
- (b) When $\theta \succeq 0$ and $\mathbf{1}^T \theta = 1$, f is concave. (This is the same as part (a), but here it is for general n .)
- (c) When $\theta \succeq 0$ and $\mathbf{1}^T \theta \leq 1$, f is concave.
- (d) When $\theta \preceq 0$, f is convex.
- (e) When $\mathbf{1}^T \theta = 1$ and exactly *one* of the elements of θ is positive, f is convex.
- (f) When $\mathbf{1}^T \theta \geq 1$ and exactly *one* of the elements of θ is positive, f is convex.

Remark. Parts (c), (d), and (f) exactly characterize the cases when f is either convex or concave. That is, if none of these conditions on θ hold, f is neither convex nor concave. Your teaching staff has, however, kindly refrained from asking you to show this.

Solution. To shorten our proofs, when both x and θ are vectors, we overload notation so that

$$f(x) = x_1^{\theta_1} \cdots x_n^{\theta_n} = x^\theta.$$

- (a) Since $x_1^{\theta_1}$ is concave for $0 \leq \theta_1 \leq 1$, applying the perspective transformation gives that

$$x_2(x_1/x_2)^{\theta_1} = x_1^{\theta_1} x_2^{1-\theta_1}$$

is concave, which is what we wanted.

- (b) The proof is by induction on n . We know the base case with $n = 1$ holds. For the induction step, if $\theta \in \mathbf{R}_+^{n+1}$, $\tilde{\theta} = (\theta_1, \dots, \theta_n)$, $\tilde{x} = (x_1, \dots, x_n)$, and $\mathbf{1}^T \theta = 1$, then $\tilde{x}^{\tilde{\theta}/\mathbf{1}^T \tilde{\theta}}$ is concave by the induction assumption. The function $y^{\mathbf{1}^T \tilde{\theta}} z^{1-\mathbf{1}^T \tilde{\theta}}$ is concave by (a) and nondecreasing. The composition rules give that

$$(\tilde{x}^{\tilde{\theta}/\mathbf{1}^T \theta})^{\mathbf{1}^T \tilde{\theta}} x_{n+1}^{1-\mathbf{1}^T \tilde{\theta}} = \tilde{x}^{\tilde{\theta}} x_{n+1}^{\theta_{n+1}} = x^\theta$$

is concave.

- (c) If $\mathbf{1}^T \theta \leq 1$, then $x^{\theta/\mathbf{1}^T \theta}$ is concave by (b). The function $y^{\mathbf{1}^T \theta}$ is concave and nondecreasing. Composition gives that

$$(x^{\theta/\mathbf{1}^T \theta})^{\mathbf{1}^T \theta} = x^\theta$$

is concave.

- (d) If $\theta \preceq 0$, then $x^{\theta/\mathbf{1}^T \theta}$ is concave by part (b). (We can assume $\mathbf{1}^T \theta \neq 0$.) The function $y^{\mathbf{1}^T \theta}$ is convex and nonincreasing, since $\mathbf{1}^T \theta < 0$. Composition gives that

$$(x^{\theta/\mathbf{1}^T \theta})^{\mathbf{1}^T \theta} = x^\theta$$

is convex.

Here's another proof, that several people used, and which is arguably simpler than the one above. Since $\theta_i \leq 0$, $\theta_i \log x_i$ is a convex function of x_i , and therefore the sum $\sum_i \theta_i \log x_i$ is convex in x . By the composition rules, the exponential of a convex function is convex, so

$$\exp\left(\sum_i \theta_i \log x_i\right) = x^\theta$$

is convex.

- (e) If $\theta \in \mathbf{R}^{n+1}$ and $\mathbf{1}^T \theta = 1$, we can assume that the single positive element is $\theta_{n+1} > 0$, so that $\tilde{\theta} = (\theta_1, \dots, \theta_n) \preceq 0$. If $\tilde{x} = (x_1, \dots, x_n)$, then $\tilde{x}^{\tilde{\theta}}$ is convex by part (d). Applying the perspective transformation gives that

$$x_{n+1} (\tilde{x}/x_{n+1})^{\tilde{\theta}} = \tilde{x}^{\tilde{\theta}} x_{n+1}^{1-\mathbf{1}^T \tilde{\theta}} = \tilde{x}^{\tilde{\theta}} x_{n+1}^{\theta_{n+1}} = x^\theta$$

is convex.

- (f) If $\mathbf{1}^T \theta \geq 1$ and exactly one element of θ is positive, then $x^\theta/\mathbf{1}^T \theta$ is convex by part (e). The function $y^{\mathbf{1}^T \theta}$ is convex and nondecreasing. Composition gives us that

$$(x^{\theta/\mathbf{1}^T \theta})^{\mathbf{1}^T \theta} = x^\theta$$

is convex.

Remark. The proofs for (c), (d), and (f) are syntactically identical.

Remark. We can also prove (c) with the following self-contained argument. A syntactically identical self-contained argument also works for (f) by substituting "convex" for "concave".

The proof is by induction on n . We know the base case: $x_1^{\theta_1}$ is concave for $0 \leq \theta_1 \leq 1$. For the inductive step, if $\theta \in \mathbf{R}_+^{n+1}$ and $\mathbf{1}^T \theta \leq 1$, let $\tilde{\theta} = (\theta_1, \dots, \theta_n)$ and $\tilde{x} = (x_1, \dots, x_n)$. Note that $\tilde{x}^{\tilde{\theta}}/\mathbf{1}^T \theta$ is concave by the induction assumption. Applying the perspective transformation gives that

$$x_{n+1} (\tilde{x}/x_{n+1})^{\tilde{\theta}/\mathbf{1}^T \theta} = \tilde{x}^{\tilde{\theta}/\mathbf{1}^T \theta} x_{n+1}^{1-\mathbf{1}^T \tilde{\theta}/\mathbf{1}^T \theta}$$

is concave. The function $y^{\mathbf{1}^T \theta}$ is concave and nondecreasing, and composing it with the previous function shows that

$$(\tilde{x}^{\tilde{\theta}/\mathbf{1}^T \theta} x_{n+1}^{1-\mathbf{1}^T \tilde{\theta}/\mathbf{1}^T \theta})^{\mathbf{1}^T \theta} = \tilde{x}^{\tilde{\theta}} x_{n+1}^{\mathbf{1}^T \theta - \mathbf{1}^T \tilde{\theta}} = \tilde{x}^{\tilde{\theta}} x_{n+1}^{\theta_{n+1}} = x^\theta$$

is concave, completing the proof.

3 Convex optimization problems

- 3.1** *Minimizing a function over the probability simplex.* Find simple necessary and sufficient conditions for $x \in \mathbf{R}^n$ to minimize a differentiable convex function f over the probability simplex, $\{x \mid \mathbf{1}^T x = 1, x \succeq 0\}$.

Solution. The simple basic optimality condition is that x is feasible, i.e., $x \succeq 0$, $\mathbf{1}^T x = 1$, and that $\nabla f(x)^T(y - x) \geq 0$ for all feasible y . We'll first show this is equivalent to

$$\min_{i=1,\dots,n} \nabla f(x)_i \geq \nabla f(x)^T x.$$

To see this, suppose that $\nabla f(x)^T(y - x) \geq 0$ for all feasible y . Then in particular, for $y = e_i$, we have $\nabla f(x)_i \geq \nabla f(x)^T x$, which is what we have above. To show the other way, suppose that $\nabla f(x)_i \geq \nabla f(x)^T x$ holds, for $i = 1, \dots, n$. Let y be feasible, i.e., $y \succeq 0$, $\mathbf{1}^T y = 1$. Then multiplying $\nabla f(x)_i \geq \nabla f(x)^T x$ by y_i and summing, we get

$$\sum_{i=1}^n y_i \nabla f(x)_i \geq \left(\sum_{i=1}^n y_i \right) \nabla f(x)^T x = \nabla f(x)^T x.$$

The lefthand side is $y^T \nabla f(x)$, so we have $\nabla f(x)^T(y - x) \geq 0$.

Now we can simplify even further. The condition above can be written as

$$\min_{i=1,\dots,n} \frac{\partial f}{\partial x_i} \geq \sum_{i=1}^n x_i \frac{\partial f}{\partial x_i}.$$

But since $\mathbf{1}^T x = 1$, $x \succeq 0$, we have

$$\min_{i=1,\dots,n} \frac{\partial f}{\partial x_i} \leq \sum_{i=1}^n x_i \frac{\partial f}{\partial x_i},$$

and it follows that

$$\min_{i=1,\dots,n} \frac{\partial f}{\partial x_i} = \sum_{i=1}^n x_i \frac{\partial f}{\partial x_i}.$$

The right hand side is a mixture of $\partial f / \partial x_i$ terms and equals the minimum of all of the terms. This is possible only if $x_k = 0$ whenever $\partial f / \partial x_k > \min_i \partial f / \partial x_i$.

Thus we can write the (necessary and sufficient) optimality condition as $\mathbf{1}^T x = 1$, $x \succeq 0$, and, for each k ,

$$x_k > 0 \Rightarrow \frac{\partial f}{\partial x_k} = \min_{i=1,\dots,n} \frac{\partial f}{\partial x_i}.$$

In particular, for k 's with $x_k > 0$, $\partial f / \partial x_k$ are all equal.

- 3.2** ‘Hello World’ in CVX*. Use CVX, CVX.PY or Convex.jl to verify the optimal values you obtained (analytically) for exercise 4.1 in *Convex Optimization*.

Solution.

- (a) $p^* = 0.6$

- (b) $p^* = -\infty$
- (c) $p^* = 0$
- (d) $p^* = \frac{1}{3}$
- (e) $p^* = \frac{1}{2}$

```
%exercise 4.1 using CVX
```

```
%set up a vector to store optimal values of problems
optimal_values=zeros(5,1);
```

```
%part a
cvx_begin
variable x(2)
minimize(x(1)+x(2))
2*x(1)+x(2) >= 0;
x(1)+3*x(2) >= 1;
x >= 0;
cvx_end
```

```
optimal_values(1)=cvx_optval;
```

```
%part b
cvx_begin
variable x(2)
minimize(-sum(x))
2*x(1)+x(2) >= 0;
x(1)+3*x(2) >= 1;
x >= 0;
cvx_end
optimal_values(2)=cvx_optval;
```

```
%part c
cvx_begin
variable x(2)
minimize(x(1))
2*x(1)+x(2) >= 0;
x(1)+3*x(2) >= 1;
x >= 0;
cvx_end
optimal_values(3)=cvx_optval;
```

```
%part d
cvx_begin
variable x(2)
minimize(max(x))
```

```

2*x(1)+x(2) >= 0;
x(1)+3*x(2) >= 1;
x >= 0;
cvx_end
optimal_values(4)=cvx_optval;

%part e
cvx_begin
variable x(2,1)
minimize( square(x(1))+ 9*square(x(2)) )
2*x(1)+x(2) >= 0;
x(1)+3*x(2) >= 1;
x >= 0;
cvx_end
optimal_values(5)=cvx_optval;

import cvxpy as cvx
import numpy as np

x1 = cvx.Variable()
x2 = cvx.Variable()
constraints = [2*x1 + x2 >= 1, x1 + 3*x2 >= 1, x1 >= 0, x2 >= 0]
prob = cvx.Problem([], constraints)

# (a)
prob.objective = cvx.Minimize(x1+x2)
prob.solve()
print 'With obj. x1+x2, p* is %.2f, optimal x1 is %.2f, optimal x2 is %.2f' \
      % (prob.value, x1.value, x2.value)

# (b)
prob.objective = cvx.Minimize(-x1-x2)
prob.solve(solver=cvx.CVXOPT)
print 'With obj. -x1-x2, status is ' + prob.status

# (c)
prob.objective = cvx.Minimize(x1)
prob.solve()
print 'With obj. x1, p* is %.2f, x1* is %.2f, x2* is %.2f' \
      % (prob.value, x1.value, x2.value)

# (d)
prob.objective = cvx.Minimize(cvx.max_elemwise(x1, x2))
prob.solve()
print 'With obj. max(x1,x2), p* is %.2f, x1* is %.2f, x2* is %.2f' \
      % (prob.value, x1.value, x2.value)

```

```

# (e)
prob.objective = cvx.Minimize(cvx.square(x1) + 9*cvx.square(x2))
prob.solve()
print 'With obj. x1^2+9x2^2, p* is %.2f, x1* is %.2f, x2* is %.2f' \
    % (prob.value, x1.value, x2.value)

# exercise 4.1 using CVX
using Convex

# set up a vector to store optimal values of problems
optimal_values=zeros(5);

# part a
x = Variable(2)
obj = x[1] + x[2]
constr = [2*x[1]+x[2] >= 1, x[1]+3*x[2] >= 1, x >= 0]
p = minimize(obj, constr)
solve!(p)
optimal_values[1]=p.optval;

# part b
x = Variable(2)
obj = -sum(x)
constr = [2*x[1]+x[2] >= 1, x[1]+3*x[2] >= 1, x >= 0]
p = minimize(obj, constr)
solve!(p)
optimal_values[2]=p.optval;

# part c
x = Variable(2)
obj = x[1]
constr = [2*x[1]+x[2] >= 1, x[1]+3*x[2] >= 1, x >= 0]
p = minimize(obj, constr)
solve!(p)
optimal_values[3]=p.optval;

# part d
x = Variable(2)
obj = maximum(x)
constr = [2*x[1]+x[2] >= 1, x[1]+3*x[2] >= 1, x >= 0]
p = minimize(obj, constr)
solve!(p)
optimal_values[4]=p.optval;

# part e

```

```

x = Variable(2)
obj = square(x[1]) + 9*square(x[2])
constr = [2*x[1]+x[2] >= 1, x[1]+3*x[2] >= 1, x >= 0]
p = minimize(obj, constr)
solve!(p)
optimal_values[5]=p.optval;

```

- 3.3 Reformulating constraints in CVX***. Each of the following CVX code fragments describes a convex constraint on the scalar variables x , y , and z , but violates the CVX rule set, and so is invalid. Briefly explain why each fragment is invalid. Then, rewrite each one in an equivalent form that conforms to the CVX rule set. In your reformulations, you can use linear equality and inequality constraints, and inequalities constructed using CVX functions. You can also introduce additional variables, or use LMIs. Be sure to explain (briefly) why your reformulation is equivalent to the original constraint, if it is not obvious.

Check your reformulations by creating a small problem that includes these constraints, and solving it using CVX. Your test problem doesn't have to be feasible; it's enough to verify that CVX processes your constraints without error.

Remark. This looks like a problem about ‘how to use CVX software’, or ‘tricks for using CVX’. But it really checks whether you understand the various composition rules, convex analysis, and constraint reformulation rules.

- (a) `norm([x + 2*y, x - y]) == 0`
- (b) `square(square(x + y)) <= x - y`
- (c) `1/x + 1/y <= 1; x >= 0; y >= 0`
- (d) `norm(max(x,1), max(y,2)) <= 3*x + y`
- (e) `x*y >= 1; x >= 0; y >= 0`
- (f) `(x + y)^2/sqrt(y) <= x - y + 5`
- (g) `x^3 + y^3 <= 1; x >= 0; y >= 0`
- (h) `x + z <= 1 + sqrt(x*y - z^2); x >= 0; y >= 0`

Solution.

- (a) The lefthand side is correctly identified as convex, but equality constraints are only valid with affine left and right hand sides. Since the norm of a vector is zero if and only if the vector is zero, we can express the constraint as $x + 2*y == 0; x - y == 0$, or simply $x == 0; y == 0$.
- (b) The problem is that `square()` can only accept affine arguments, because it is convex, but not increasing. To correct this use `square_pos()` instead:

```
square_pos(square(x + y)) <= x - y
```

We can also reformulate this constraint by introducing an additional variable.

```

variable t
square(x + y) <= t
square(t) <= x - y

```

Note that, in general, decomposing the objective by introducing new variables doesn't need to work. It works in this case because the outer `square` function is convex and monotonic over \mathbf{R}_+ .

Alternatively, we can rewrite the constraint as

$$(x + y)^4 \leq x - y$$

- (c) $1/x$ isn't convex, unless you restrict the domain to \mathbf{R}_{++} . We can write this one as `inv_pos(x) + inv_pos(y)`. The `inv_pos` function has domain \mathbf{R}_{++} so the constraints $x > 0, y > 0$ are (implicitly) included.
- (d) The problem is that `norm()` can only accept affine argument since it is convex but not increasing. One way to correct this is to introduce new variables u and v :

```
norm([u, v]) <= 3*x + y
max(x, 1) <= u
max(y, 2) <= v
```

Decomposing the objective by introducing new variables works here because `norm` is convex and monotonic over \mathbf{R}_+^2 , and in particular over $[1, \infty) \times [2, \infty)$.

- (e) xy isn't concave, so this isn't going to work as stated. But we can express the constraint as $x \geq \text{inv_pos}(y)$. (You can switch around x and y here.) Another solution is to write the constraint as `geo_mean([x, y]) >= 1`. We can also give an LMI representation:

$$[x \ 1; 1 \ y] == \text{semidefinite}(2)$$

- (f) This fails when we attempt to divide a convex function by a concave one. We can write this as

$$\text{quad_over_lin}(x + y, \sqrt{y}) \leq x - y + 5$$

This works because `quad_over_lin` is monotone decreasing in the second argument, so it can accept a concave function here, and `sqrt` is concave.

- (g) The function $x^3 + y^3$ is convex for $x \geq 0, y \geq 0$. But x^3 isn't convex for $x < 0$, so CVX is going to reject this statement. One way to rewrite this constraint is

$$\text{quad_pos_over_lin}(\text{square}(x), x) + \text{quad_pos_over_lin}(\text{square}(y), y) \leq 1$$

This works because `quad_pos_over_lin` is convex and increasing in its first argument, hence accepts a convex function in its first argument. (The function `quad_over_lin`, however, is not increasing in its first argument, and so won't work.)

Alternatively, and more simply, we can rewrite the constraint as

$$\text{pow_pos}(x, 3) + \text{pow_pos}(y, 3) \leq 1$$

- (h) The problem here is that xy isn't concave, which causes CVX to reject the statement. To correct this, notice that

$$\sqrt{xy - z^2} = \sqrt{y(x - z^2/y)},$$

so we can reformulate the constraint as

$$x + z \leq 1 + \text{geo_mean}([x - \text{quad_over_lin}(z, y), y])$$

This works, since `geo_mean` is concave and nondecreasing in each argument. It therefore accepts a concave function in its first argument.

We can check our reformulations by writing the following feasibility problem in CVX (which is obviously infeasible)

```
cvx_begin
    variables x y u v z
    x == 0;
    y == 0;
    (x + y)^4 <= x - y;
    inv_pos(x) + inv_pos(y)  <= 1;
    norm([u; v]) <= 3*x + y;
    max(x,1) <= u;
    max(y,2) <= v;
    x >= inv_pos(y);
    x >= 0;
    y >= 0;
    quad_over_lin(x + y, sqrt(y)) <= x - y + 5;
    pow_pos(x,3) + pow_pos(y,3) <= 1;
    x+z <= 1+geo_mean([x-quad_over_lin(z,y), y])
cvx_end
```

3.4 Optimal activity levels. Solve the optimal activity level problem described in exercise 4.17 in *Convex Optimization*, for the instance with problem data

$$A = \begin{bmatrix} 1 & 2 & 0 & 1 \\ 0 & 0 & 3 & 1 \\ 0 & 3 & 1 & 1 \\ 2 & 1 & 2 & 5 \\ 1 & 0 & 3 & 2 \end{bmatrix}, \quad c^{\max} = \begin{bmatrix} 100 \\ 100 \\ 100 \\ 100 \\ 100 \end{bmatrix}, \quad p = \begin{bmatrix} 3 \\ 2 \\ 7 \\ 6 \end{bmatrix}, \quad p^{\text{disc}} = \begin{bmatrix} 2 \\ 1 \\ 4 \\ 2 \end{bmatrix}, \quad q = \begin{bmatrix} 4 \\ 10 \\ 5 \\ 10 \end{bmatrix}.$$

You can do this by forming the LP you found in your solution of exercise 4.17, or more directly, using CVX*. Give the optimal activity levels, the revenue generated by each one, and the total revenue generated by the optimal solution. Also, give the average price per unit for each activity level, *i.e.*, the ratio of the revenue associated with an activity, to the activity level. (These numbers should be between the basic and discounted prices for each activity.) Give a *very brief* story explaining, or at least commenting on, the solution you find.

Solution. For this part, we write the problem in a form close to its original statement, and let CVX* do the work of reformulating it as an LP.

The following CVX code implements this:

```
A=[ 1 2 0 1;
    0 0 3 1;
    0 3 1 1;
    2 1 2 5;
```

```

1 0 3 2];

cmax=[100;100;100;100;100];
p=[3;2;7;6];
pdisc=[2;1;4;2];
q=[4; 10 ;5; 10];

cvx_begin
    variable x(4)
    maximize( sum(min(p.*x,p.*q+pdisc.*(x-q))) )
    subject to
        x >= 0;
        A*x <= cmax
cvx_end

x
r=min(p.*x,p.*q+pdisc.*(x-q))
totr=sum(r)
avgPrice=r./x

import cvxpy as cvx
import numpy as np

A = np.matrix('1 2 0 1; \
                0 0 3 1; \
                0 3 1 1; \
                2 1 2 5; \
                1 0 3 2')

cmax = np.matrix('100; 100; 100; 100; 100')
p = np.matrix('3; 2; 7; 6')
pdisc = np.matrix('2; 1; 4; 2')
q = np.matrix('4; 10; 5; 10')

x = cvx.Variable(4)

t1 = cvx.mul_elemwise(p, x)
t2 = cvx.mul_elemwise(p, q) + cvx.mul_elemwise(pd़isc, x-q)
obj = cvx.Maximize(cvx.sum_entries(cvx.min_elemwise(t1, t2)))

cons = [x >= 0, A*x <= cmax]

prob = cvx.Problem(obj, cons)
prob.solve()

r = cvx.min_elemwise(t1, t2).value

```

```

totr = sum(r)
avgPrice = r / x.value

print x.value
print r
print totr
print avgPrice

using Convex

A = [ 1 2 0 1;
      0 0 3 1;
      0 3 1 1;
      2 1 2 5;
      1 0 3 2];

cmax = [100;100;100;100;100];
p = [3;2;7;6];
pdisc = [2;1;4;2];
q = [4; 10 ;5; 10];

x = Variable(4);
r = min(p.*x, p.*q + pdisc.*(x - q));
totr = sum(r);
constraints = [x >= 0, A*x <= cmax];
p = maximize(totr, constraints);
solve!(p);

avgPrice = evaluate(r)./evaluate(x);
println(x.value)
println(evaluate(r))
println(evaluate(totr))
println(avgPrice)

```

The result of the code is

```

x =
4.0000
22.5000
31.0000
1.5000

```

```
r =
```

```

12.0000
32.5000
139.0000
9.0000

```

```

totr =
192.5000

```

```

avgPrice =
3.0000
1.4444
4.4839
6.0000

```

We notice that the 3rd activity level is the highest and is also the one with the highest basic price. Since it also has a high discounted price its activity level is higher than the discount quantity level and it produces the highest contribution to the total revenue. The 4th activity has a discounted price which is substantially lower then the basic price and its activity is therefore lower than the discount quantity level. Moreover it requires the use of a lot of resources and therefore its activity level is low.

3.5 Minimizing the ratio of convex and concave piecewise-linear functions. We consider the problem

$$\begin{aligned} \text{minimize} \quad & \frac{\max_{i=1,\dots,m}(a_i^T x + b_i)}{\min_{i=1,\dots,p}(c_i^T x + d_i)} \\ \text{subject to} \quad & Fx \preceq g, \end{aligned}$$

with variable $x \in \mathbf{R}^n$. We assume that $c_i^T x + d_i > 0$ and $\max_{i=1,\dots,m}(a_i^T x + b_i) \geq 0$ for all x satisfying $Fx \preceq g$, and that the feasible set is nonempty and bounded. This problem is quasiconvex, and can be solved using bisection, with each iteration involving a feasibility LP. Show how the problem can be solved by solving *one* LP, using a trick similar to one described in §4.3.2.

Solution. We will show that the problem is equivalent to the optimization problem

$$\begin{aligned} \text{minimize} \quad & \max_{i=1,\dots,m} (a_i^T y + b_i t) \\ \text{subject to} \quad & \min_{i=1,\dots,p} (c_i^T y + d_i t) \geq 1 \\ & Fy \preceq gt \\ & t \geq 0 \end{aligned} \tag{1}$$

with variables y, t . This can be further expressed as an LP using the standard tricks, by introducing

an additional variable u :

$$\begin{aligned} & \text{minimize} && u \\ & \text{subject to} && a_i^T y + b_i t \leq u, \quad i = 1, \dots, m \\ & && c_i^T y + d_i t \geq 1, \quad i = 1, \dots, p \\ & && Fy \preceq gt \\ & && t \geq 0. \end{aligned}$$

To show that (1) is equivalent to the problem in the assignment, we first note that $t > 0$ for all feasible (y, t) . Indeed, the first constraint implies that $(y, t) \neq 0$. We must have $t > 0$ because otherwise $Fy \preceq 0$ and $y \neq 0$, which means that y defines an unbounded direction in the polyhedron $\{x \mid Fx \preceq g\}$, contradicting the assumption that this polyhedron is bounded. If $t > 0$ for all feasible y, t , we can rewrite problem (1) as

$$\begin{aligned} & \text{minimize} && t \max_{i=1,\dots,m} (a_i^T(y/t) + b_i) \\ & \text{subject to} && \min_{i=1,\dots,p} (c_i^T(y/t) + d_i) \geq 1/t \\ & && F(y/t) \preceq g \\ & && t \geq 0. \end{aligned} \tag{2}$$

Next we argue that the first constraint necessarily holds with equality at the optimum, *i.e.*, the optimal solution of (2) is also the solution of

$$\begin{aligned} & \text{minimize} && t \max_{i=1,\dots,m} (a_i^T(y/t) + b_i) \\ & \text{subject to} && \min_{i=1,\dots,p} (c_i^T(y/t) + d_i) = 1/t \\ & && F(y/t) \preceq g \\ & && t \geq 0. \end{aligned} \tag{3}$$

To see this, suppose we fix y/t in (2) and optimize only over t . Since $\max_i(a_i^T(y/t) + b_i) \geq 0$ if $F(y/t) \leq g$, we minimize the cost function by making t as small as possible, *i.e.*, choosing t such that

$$\min_{i=1,\dots,p} (c_i^T(y/t) + d_i) = 1/t.$$

The final step is to substitute this expression for the optimal t in the cost function of (3) to get

$$\begin{aligned} & \text{minimize} && \frac{\max_{i=1,\dots,m} (a_i^T(y/t) + b_i)}{\min_{i=1,\dots,p} (c_i^T(y/t) + d_i)} \\ & \text{subject to} && F(y/t) \preceq g \\ & && t \geq 0. \end{aligned}$$

This is the problem of the assignment with $x = y/t$.

3.6 Two problems involving two norms. We consider the problem

$$\text{minimize} \quad \frac{\|Ax - b\|_1}{1 - \|x\|_\infty}, \tag{4}$$

and the very closely related problem

$$\text{minimize} \quad \frac{\|Ax - b\|_1^2}{1 - \|x\|_\infty}. \quad (5)$$

In both problems, the variable is $x \in \mathbf{R}^n$, and the data are $A \in \mathbf{R}^{m \times n}$ and $b \in \mathbf{R}^m$. Note that the only difference between problem (4) and (5) is the square in the numerator. In both problems, the constraint $\|x\|_\infty < 1$ is implicit. You can assume that $b \notin \mathcal{R}(A)$, in which case the constraint $\|x\|_\infty < 1$ can be replaced with $\|x\|_\infty \leq 1$.

Answer the following two questions, for each of the two problems. (So you will answer four questions all together.)

- (a) Is the problem, exactly as stated (and for all problem data), convex? If not, is it quasiconvex? Justify your answer.
- (b) Explain how to solve the problem. Your method can involve an SDP solver, an SOCP solver, an LP solver, or any combination. You can include a one-parameter bisection, if necessary. (For example, you can solve the problem by bisection on a parameter, where each iteration consists of solving an SOCP feasibility problem.)

Give the best method you can. In judging best, we use the following rules:

- *Bisection methods are worse than ‘one-shot’ methods.* Any method that solves the problem above by solving *one* LP, SOCP, or SDP problem is better than any method that uses a one-parameter bisection. In other words, use a bisection method only if you cannot find a ‘one-shot’ method.
- *Use the simplest solver needed to solve the problem.* We consider an LP solver to be simpler than an SOCP solver, which is considered simpler than an SDP solver. Thus, a method that uses an LP solver is better than a method that uses an SOCP solver, which in turn is better than a method that uses an SDP solver.

Solution. First we discuss the problem

$$\text{minimize} \quad \frac{\|Ax - b\|_1}{1 - \|x\|_\infty}.$$

- (a) This problem is in general not convex. As an example, take $n = 1$,

$$A = \begin{bmatrix} 2 \\ 0 \end{bmatrix}, \quad b = \begin{bmatrix} 1 \\ 0.01 \end{bmatrix}.$$

The objective is

$$f(x) = \frac{|2x - 1| + 0.01}{1 - |x|},$$

which is not convex (as can be easily seen by plotting the function). However, the problem is quasiconvex: for $\alpha \geq 0$, the sublevel sets

$$\begin{aligned} S_\alpha &= \{x \mid \|Ax - b\|_1 / (1 - \|x\|_\infty) \leq \alpha\} \\ &= \{x \mid \|Ax - b\|_1 + \alpha\|x\|_\infty \leq \alpha\} \end{aligned}$$

are convex. For $\alpha < 0$ the α -sublevel set is empty, hence convex.

- (b) The problem can be solved using a one-parameter bisection and an LP solver. At each iteration we solve the feasibility problem

$$\begin{aligned} & \text{find} && x \\ & \text{subject to} && \|Ax - b\|_1 / (1 - \|x\|_\infty) \leq \alpha \end{aligned}$$

(with domain $\{x \mid \|x\|_\infty < 1\}$) for fixed α . The assumption $b \notin \mathcal{R}(A)$ implies that the optimal value is positive, so we only have to consider positive values of α . The feasibility problem can be written in convex form

$$\begin{aligned} & \text{find} && x \\ & \text{subject to} && \|Ax - b\|_1 + \alpha\|x\|_\infty \leq \alpha. \end{aligned}$$

(Note that if $\alpha > 0$ the constraint implies the implicit constraint $\|x\|_\infty < 1$.) It can be further rewritten as an LP feasibility problem

$$\begin{aligned} & \text{find} && x \\ & \text{subject to} && -y\mathbf{1} \preceq x \preceq y\mathbf{1} \\ & && -z \preceq Ax - b \preceq z \\ & && \mathbf{1}^T z + \alpha y \leq \alpha \end{aligned}$$

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

In fact, we can do better and pose the problem as a single LP. We first note that by introducing an auxiliary scalar variable t we can formulate the problem as

$$\begin{aligned} & \text{minimize} && \|Ax - b\|_1/t \\ & \text{subject to} && t + \|x\|_\infty \leq 1 \end{aligned}$$

with an implicit constraint $t > 0$. A change of variables $y = x/t$, $z = 1/t$ gives a convex problem

$$\begin{aligned} & \text{minimize} && \|Ay - bz\|_1 \\ & \text{subject to} && 1 + \|y\|_\infty \leq z. \end{aligned}$$

(Note that the constraint implies $z > 0$.) This problem now reduces to an LP

$$\begin{aligned} & \text{minimize} && \mathbf{1}^T u \\ & \text{subject to} && -u \preceq Ay - bz \preceq u \\ & && -v\mathbf{1} \preceq y \preceq v\mathbf{1} \\ & && 1 + v \leq z \end{aligned}$$

with variables $u \in \mathbf{R}^m$, $y \in \mathbf{R}^n$, $z \in \mathbf{R}$, $v \in \mathbf{R}$.

Next we consider the problem

$$\text{minimize } \frac{\|Ax - b\|_1^2}{1 - \|x\|_\infty}.$$

- (a) This problem is convex. The objective is the composition of the function

$$\Phi(s, t) = \begin{cases} s^2/(1-t) & s \geq 0 \\ 0 & s < 0, \end{cases}$$

with domain $\mathbf{R} \times [0, 1]$, with the functions $g_1(x) = \|Ax - b\|_1$ and $g_2(x) = \|x\|_\infty$. Since Φ is convex, and nondecreasing in each argument, and g_1 and g_2 are convex, the composition $\Phi(g_1(x), g_2(x))$ is a convex function. (We had to extend the function ϕ as zero for $s < 0$ in order to claim that Φ is nondecreasing in each argument.)

As another proof, one can note that the epigraph of the function is a convex set, since

$$\frac{\|Ax - b\|_1^2}{1 - \|x\|_\infty} \leq t, \quad \|x\|_\infty < 1 \iff \frac{\|Ax - b\|_1^2}{t} + \|x\|_\infty \leq 1, \quad t > 0$$

and the left-hand side of the second inequality is a convex function, jointly in x and t . (The function $\|y\|_1^2/t$ is convex because it is the perspective of the convex function $\|z\|_1^2$.)

- (b) The problem can be written as an SOCP or SDP. With the assumption that $b \notin \mathcal{R}(A)$, it is equivalent to the problem:

$$\begin{array}{ll} \text{minimize} & y \\ \text{subject to} & s^2/(1-t) \leq y \\ & \|Ax - b\|_1 \leq s \\ & \|x\|_\infty \leq t \\ & t \leq 1. \end{array}$$

The second and third constraints can be reformulated as linear inequalities. The first constraint is equivalent to the linear matrix inequality

$$\begin{bmatrix} 1-t & s \\ s & y \end{bmatrix} \succeq 0$$

and also (using the observation in problem T4.26) to a second-order cone constraint

$$\left\| \begin{bmatrix} 2s \\ 1-t-y \end{bmatrix} \right\|_2 \leq 1-t+y, \quad t \leq 1, \quad y \geq 0.$$

If we use the SOCP formulation we obtain

$$\begin{array}{ll} \text{minimize} & y \\ \text{subject to} & -z \preceq Ax - b \preceq z, \\ & \mathbf{1}^T z \leq s, \\ & -t\mathbf{1} \preceq x \preceq t\mathbf{1}, \\ & \left\| \begin{bmatrix} 2s \\ 1-t-y \end{bmatrix} \right\|_2 \leq 1-t+y \\ & t \leq 1 \\ & y \geq 0. \end{array}$$

which is an SOCP in the variables s, t, x, y and z .

The SDP formulation is

$$\begin{array}{ll} \text{minimize} & y \\ \text{subject to} & -z \preceq Ax - b \preceq z, \\ & \mathbf{1}^T z \leq s, \\ & -t\mathbf{1} \preceq x \preceq t\mathbf{1}, \\ & \begin{bmatrix} 1-t & s \\ s & y \end{bmatrix} \succeq 0. \end{array}$$

There are several variations on this; for example we can just substitute $\mathbf{1}^T z$ for s in the linear matrix inequality.

3.7 The illumination problem.

$$f(p) = \max_{i=1,\dots,n} |\log a_i^T p - \log I_{\text{des}}|$$

where $a_i \in \mathbf{R}^m$, and $I_{\text{des}} > 0$ are given, and $p \in \mathbf{R}_+^m$.

- (a) Show that $\exp f$ is convex on $\{p \mid a_i^T p > 0, i = 1, \dots, n\}$.
- (b) Show that the constraint ‘no more than half of the total power is in any 10 lamps’ is convex (*i.e.*, the set of vectors p that satisfy the constraint is convex).
- (c) Show that the constraint ‘no more than half of the lamps are on’ is (in general) *not* convex.

Solution. To simplify the notation, we assume that $I_{\text{des}} = 1$ (if not, we can simply redefine a_{ij} as a_{ij}/I_{des}). We write $I_i = a_i^T p$ (in the notation of page 1-6 of the lecture notes), so the objective function can be written as

$$f(p) = \max_i |\log a_i^T p|.$$

The domain of f is

$$\{p \mid a_i^T p > 0, i = 1, \dots, n\}.$$

- (a) First note that

$$\begin{aligned} |\log(a_i^T p)| &= \max\{\log a_i^T p, \log(1/a_i^T p)\} \\ &= \log \max\{a_i^T p, 1/a_i^T p\}, \end{aligned}$$

so we can write the objective function as

$$f(p) = \log \max_i \max\{a_i^T p, 1/a_i^T p\}.$$

Both $a_i^T p$ and $1/a_i^T p$ are convex on $\text{dom } f$, and therefore $\max_i \max\{a_i^T p, 1/a_i^T p\}$ is a convex function. In other words $\exp f$ is convex.

- (b) This constraint can be expressed as

$$\sum_{i=1}^l p_{[i]} - 0.5 \sum_{i=1}^m p_i \leq 0$$

where $p_{[i]}$ is the i th largest component of p (see page 4-11 of the lecture notes). The first term on the left side is the power in the l lamps with the highest power, the second term is one half of the total power. The function on the left hand side is convex, since it is the sum of $\sum_{i=1}^l p_{[i]}$, which is convex, and a linear function,

- (c) Consider two solutions p^1 and p^2 that satisfy the constraint. In the first solution, the first $m/2$ lamps are on and the rest is off (*i.e.*, the first $m/2$ components of p^1 are positive and the rest is zero); in the second solution the first $m/2$ lamps are off and the rest is on (*i.e.*, the first $m/2$ components of p^2 are zero, and the rest is positive). The number of nonzero components in a convex combination of p^1 and p^2 will be m , *i.e.*, the convex combination does not satisfy the constraint.

3.8 Schur complements and LMI representation. Recognizing Schur complements (see §A5.5) often helps to represent nonlinear convex constraints as linear matrix inequalities (LMIs). Consider the function

$$f(x) = (Ax + b)^T (P_0 + x_1 P_1 + \cdots + x_n P_n)^{-1} (Ax + b)$$

where $A \in \mathbf{R}^{m \times n}$, $b \in \mathbf{R}^m$, and $P_i = P_i^T \in \mathbf{R}^{m \times m}$, with domain

$$\mathbf{dom} f = \{x \in \mathbf{R}^n \mid P_0 + x_1 P_1 + \cdots + x_n P_n \succ 0\}.$$

This is the composition of the matrix fractional function and an affine mapping, and so is convex. Give an LMI representation of $\mathbf{epi} f$. That is, find a symmetric matrix $F(x, t)$, affine in (x, t) , for which

$$x \in \mathbf{dom} f, \quad f(x) \leq t \quad \iff \quad F(x, t) \succeq 0.$$

Remark. LMI representations, such as the one you found in this exercise, can be directly used in software systems such as CVX.

Solution. The epigraph of f is the set of points (x, t) that satisfy $P_0 + x_1 P_1 + \cdots + x_n P_n \succ 0$ and

$$(Ax + b)^T (P_0 + x_1 P_1 + \cdots + x_n P_n)^{-1} (Ax + b) \leq t.$$

Using Schur complements, we can write the second inequality as

$$\begin{bmatrix} t & (Ax + b)^T \\ (Ax + b) & P_0 + x_1 P_1 + \cdots + x_n P_n \end{bmatrix} \succeq 0.$$

This a linear matrix inequality in the variables x, t , i.e., a convex constraint.

3.9 Complex least-norm problem. We consider the complex least ℓ_p -norm problem

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

where $A \in \mathbf{C}^{m \times n}$, $b \in \mathbf{C}^m$, and the variable is $x \in \mathbf{C}^n$. Here $\|\cdot\|_p$ denotes the ℓ_p -norm on \mathbf{C}^n , defined as

$$\|x\|_p = \left(\sum_{i=1}^n |x_i|^p \right)^{1/p}$$

for $p \geq 1$, and $\|x\|_\infty = \max_{i=1,\dots,n} |x_i|$. We assume A is full rank, and $m < n$.

- (a) Formulate the complex least ℓ_2 -norm problem as a least ℓ_2 -norm problem with real problem data and variable. *Hint.* Use $z = (\Re x, \Im x) \in \mathbf{R}^{2n}$ as the variable.
- (b) Formulate the complex least ℓ_∞ -norm problem as an SOCP.
- (c) Solve a random instance of both problems with $m = 30$ and $n = 100$. To generate the matrix A , you can use the Matlab command `A = randn(m,n) + i*randn(m,n)`. Similarly, use `b = randn(m,1) + i*randn(m,1)` to generate the vector b . Use the Matlab command `scatter` to plot the optimal solutions of the two problems on the complex plane, and comment (briefly) on what you observe. You can solve the problems using the CVX functions `norm(x,2)` and `norm(x,inf)`, which are overloaded to handle complex arguments. To utilize this feature, you will need to declare variables to be `complex` in the `variable` statement. (In particular, you do not have to manually form or solve the SOCP from part (b).)

Solution.

- (a) Define $z = (\Re x, \Im x) \in \mathbf{R}^{2n}$, so $\|x\|_2^2 = \|z\|_2^2$. The complex linear equations $Ax = b$ is the same as $\Re(Ax) = \Re b$, $\Im(Ax) = \Im b$, which in turn can be expressed as the set of linear equations

$$\begin{bmatrix} \Re A & -\Im A \\ \Im A & \Re A \end{bmatrix} z = \begin{bmatrix} \Re b \\ \Im b \end{bmatrix}.$$

Thus, the complex least ℓ_2 -norm problem can be expressed as

$$\begin{array}{ll} \text{minimize} & \|z\|_2 \\ \text{subject to} & \begin{bmatrix} \Re A & -\Im A \\ \Im A & \Re A \end{bmatrix} z = \begin{bmatrix} \Re b \\ \Im b \end{bmatrix}. \end{array}$$

(This is readily solved analytically).

- (b) Using epigraph formulation, with new variable t , we write the problem as

$$\begin{array}{ll} \text{minimize} & t \\ \text{subject to} & \left\| \begin{bmatrix} z_i \\ z_{n+i} \end{bmatrix} \right\|_2 \leq t, \quad i = 1, \dots, n \\ & \begin{bmatrix} \Re A & -\Im A \\ \Im A & \Re A \end{bmatrix} z = \begin{bmatrix} \Re b \\ \Im b \end{bmatrix}. \end{array}$$

This is an SOCP with n second-order cone constraints (in \mathbf{R}^3).

```
(c) % complex minimum norm problem
%
randn('state',0);
m = 30; n = 100;
% generate matrix A
Are = randn(m,n); Aim = randn(m,n);
bre = randn(m,1); bim = randn(m,1);

A = Are + i*Aim;
b = bre + i*bim;

% 2-norm problem (analytical solution)
Atot = [Are -Aim; Aim Are];
btot = [bre; bim];
z_2 = Atot'*inv(Atot*Atot')*btot;
x_2 = z_2(1:100) + i*z_2(101:200);

% 2-norm problem solution with cvx
cvx_begin
variable x(n) complex
minimize( norm(x) )
subject to
```

```

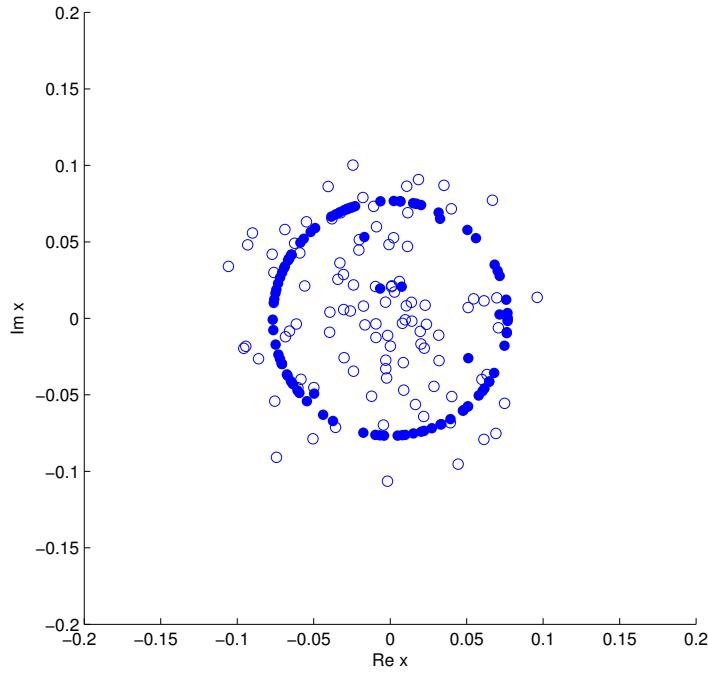
A*x == b;
cvx_end

% inf-norm problem solution with cvx
cvx_begin
variable xinf(n) complex
minimize( norm(xinf,Inf) )
subject to
    A*xinf == b;
cvx_end

% scatter plot
figure(1)
scatter(real(x),imag(x)), hold on,
scatter(real(xinf),imag(xinf),[],'filled'), hold off,
axis([-0.2 0.2 -0.2 0.2]), axis square,
xlabel('Re x'); ylabel('Im x');

```

The plot of the components of optimal $p = 2$ (empty circles) and $p = \infty$ (filled circles) solutions is presented below. The optimal $p = \infty$ solution minimizes the objective $\max_{i=1,\dots,n} |x_i|$ subject to $Ax = b$, and the scatter plot of x_i shows that almost all of them are concentrated around a circle in the complex plane. This should be expected since we are minimizing the maximum magnitude of x_i , and thus almost all of x_i 's should have about an equal magnitude $|x_i|$.



3.10 Linear programming with random cost vector. We consider the linear program

$$\begin{aligned} & \text{minimize} && c^T x \\ & \text{subject to} && Ax \preceq b. \end{aligned}$$

Here, however, the cost vector c is random, normally distributed with mean $\mathbf{E} c = c_0$ and covariance $\mathbf{E}(c - c_0)(c - c_0)^T = \Sigma$. (A , b , and x are deterministic.) Thus, for a given $x \in \mathbf{R}^n$, the cost $c^T x$ is a (scalar) Gaussian variable.

We can attach several different meanings to the goal ‘minimize $c^T x$ ’; we explore some of these below.

- (a) How would you minimize the expected cost $\mathbf{E} c^T x$ subject to $Ax \preceq b$?
- (b) In general there is a tradeoff between small expected cost and small cost variance. One way to take variance into account is to minimize a linear combination

$$\mathbf{E} c^T x + \gamma \mathbf{var}(c^T x) \quad (6)$$

of the expected value $\mathbf{E} c^T x$ and the variance $\mathbf{var}(c^T x) = \mathbf{E}(c^T x)^2 - (\mathbf{E} c^T x)^2$. This is called the ‘risk-sensitive cost’, and the parameter $\gamma \geq 0$ is called the *risk-aversion parameter*, since it sets the relative values of cost variance and expected value. (For $\gamma > 0$, we are willing to tradeoff an increase in expected cost for a decrease in cost variance). How would you minimize the risk-sensitive cost? Is this problem a convex optimization problem? Be as specific as you can.

- (c) We can also minimize the risk-sensitive cost, but with $\gamma < 0$. This is called ‘risk-seeking’. Is this problem a convex optimization problem?
- (d) Another way to deal with the randomness in the cost $c^T x$ is to formulate the problem as

$$\begin{aligned} &\text{minimize} && \beta \\ &\text{subject to} && \mathbf{prob}(c^T x \geq \beta) \leq \alpha \\ & && Ax \preceq b. \end{aligned}$$

Here, α is a fixed parameter, which corresponds roughly to the reliability we require, and might typically have a value of 0.01. Is this problem a convex optimization problem? Be as specific as you can. Can you obtain risk-seeking by choice of α ? Explain.

Solution.

- (a) Since $\mathbf{E} c^T x = c_0^T x$, the problem is an LP

$$\begin{aligned} &\text{minimize} && c_0^T x \\ &\text{subject to} && Ax \preceq b. \end{aligned}$$

- (b) We have

$$\begin{aligned} \mathbf{var}(c^T x) &= \mathbf{E}(c^T x - \mathbf{E} c^T x)^2 = \mathbf{E}((c - c_0)^T x)^2 \\ &= \mathbf{E} x^T (c - c_0)(c - c_0)^T x \\ &= x^T (\mathbf{E}(c - c_0)(c - c_0)^T) x \\ &= x^T \Sigma x. \end{aligned}$$

The risk-averse cost can therefore be minimized by solving

$$\begin{aligned} &\text{minimize} && c_0^T x + \gamma x^T \Sigma x \\ &\text{subject to} && Ax \preceq b, \end{aligned} \quad (7)$$

which is a (convex) quadratic program in x (since $\Sigma \succeq 0$ and $\gamma \geq 0$).

(c) Problem (7) is not convex if $\gamma < 0$ (the objective function is concave).

(d) The question is whether

$$\begin{aligned} & \text{minimize} && \beta \\ & \text{subject to} && \mathbf{prob}(c^T x \geq \beta) \leq \alpha \\ & && Ax \preceq b \end{aligned} \tag{8}$$

is a convex problem in x . For fixed x , $c^T x$ is a random variable, normally distribution with mean $c_0^T x$ and variance $x^T \Sigma x$. Therefore

$$\mathbf{prob}(c^T x \geq \beta) = \Phi\left(\frac{\beta - c_0^T x}{\|\Sigma^{1/2} x\|}\right),$$

where $\Phi(t) = \frac{1}{\sqrt{2\pi}} \int_x^\infty e^{-u^2/2} du$. The function Φ is monotonically decreasing, and therefore we can write

$$\begin{aligned} \mathbf{prob}(c^T x \geq \beta) \leq \alpha &\iff (\beta - c_0^T x)/\|\Sigma^{1/2} x\| \geq \Phi^{-1}(\alpha) \\ &\iff \Phi^{-1}(\alpha)\|\Sigma^{1/2} x\| + c_0^T x \leq \beta. \end{aligned}$$

If $\alpha \leq 0.5$, we have $\Phi^{-1}(\alpha) \geq 0$, so this is a convex constraint in x .

In summary, we can write (8) as

$$\begin{aligned} & \text{minimize} && \beta \\ & \text{subject to} && \Phi^{-1}(\alpha)\|\Sigma^{1/2} x\| + c_0^T x \leq \beta \\ & && Ax \preceq b, \end{aligned}$$

which is an SOCP in x and β (a linear objective, one second-order cone constraint, and a set of linear inequality constraints).

We obtain risk-seeking by choosing $\alpha > 0.5$. If $\alpha > 0.5$, we must have $\beta < \mathbf{E} c^T x$. If you plot the pdf of $c^T x$, it will be clear that there two ways to decrease $\mathbf{prob}(c^T x \geq \beta)$ if $\beta < \mathbf{E} c^T x$: we can decrease the expected value (this shifts the pdf to the left), or we can *increase* the variance, which is a risk-seeking choice.

3.11 Formulate the following optimization problems as semidefinite programs. The variable is $x \in \mathbf{R}^n$; $F(x)$ is defined as

$$F(x) = F_0 + x_1 F_1 + x_2 F_2 + \cdots + x_n F_n$$

with $F_i \in \mathbf{S}^m$. The domain of f in each subproblem is $\mathbf{dom} f = \{x \in \mathbf{R}^n \mid F(x) \succ 0\}$.

- (a) Minimize $f(x) = c^T F(x)^{-1} c$ where $c \in \mathbf{R}^m$.
- (b) Minimize $f(x) = \max_{i=1,\dots,K} c_i^T F(x)^{-1} c_i$ where $c_i \in \mathbf{R}^m$, $i = 1, \dots, K$.
- (c) Minimize $f(x) = \sup_{\|c\|_2 \leq 1} c^T F(x)^{-1} c$.
- (d) Minimize $f(x) = \mathbf{E}(c^T F(x)^{-1} c)$ where c is a random vector with mean $\mathbf{E} c = \bar{c}$ and covariance $\mathbf{E}(c - \bar{c})(c - \bar{c})^T = S$.

Solution.

(a) Using the Schur complement theorem we can write the problem as an SDP

$$\begin{aligned} & \text{minimize} && t \\ & \text{subject to} && \begin{bmatrix} F(x) & c \\ c^T & t \end{bmatrix} \succeq 0 \end{aligned}$$

with variables x, t . Note that the two problems are not quite equivalent at the boundary of the domain, *i.e.*, for points x with $F(x)$ positive semidefinite but not positive definite. The linear matrix inequality in the SDP given above is equivalent to

$$F(x) \succeq 0, \quad c \in \mathcal{R}(F(x)), \quad c^T F(x)^\dagger c \leq t$$

where $F(x)^\dagger$ is the pseudo-inverse (see page 651 of the textbook). The SDP is therefore equivalent to

$$\begin{aligned} & \text{minimize} && c^T F(x)^\dagger c \\ & \text{subject to} && F(x) \succeq 0 \\ & && c \in \mathcal{R}(F(x)). \end{aligned}$$

If $F(x)$ is positive semidefinite but singular, and $c \in \mathcal{R}(F(x))$, the objective function $c^T F(x)^\dagger c$ is finite, whereas it is $+\infty$ in the original problem. However this does not change the optimal value of the problem (unless the set $\{x \mid F(x) \succ 0\}$ is empty).

As an example, consider

$$c = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad F(x) = \begin{bmatrix} x & 0 \\ 0 & 1-x \end{bmatrix}.$$

Then the problem in the assignment is to minimize $1/x$, with domain $\{x \mid 0 < x < 1\}$. The optimal value is 1 and is not attained. The SDP reformulation is equivalent to minimizing $1/x$ subject to $0 \leq x \leq 1$. The optimal value is 1 and attained at $x = 1$.

(b)

$$\begin{aligned} & \text{minimize} && t \\ & \text{subject to} && \begin{bmatrix} F(x) & c_i \\ c_i^T & t \end{bmatrix} \succeq 0, \quad i = 1, \dots, K. \end{aligned}$$

(c) The cost function can be expressed as

$$f(x) = \lambda_{\max}(F(x)^{-1}),$$

so $f(x) \leq t$ if and only if $F(x)^{-1} \preceq tI$. Using a Schur complement we get

$$\begin{aligned} & \text{minimize} && t \\ & \text{subject to} && \begin{bmatrix} F(x) & I \\ I & tI \end{bmatrix} \succeq 0. \end{aligned}$$

(d) The cost function can be expressed as

$$f(x) = \bar{c}^T F(x)^{-1} \bar{c} + \mathbf{tr}(F(x)^{-1} S).$$

If we factor S as $S = \sum_{k=1}^m c_k c_k^T$ the problem is equivalent to

$$\text{minimize } \bar{c}^T F(x)^{-1} \bar{c} + \sum_{k=1}^m c_k^T F(x)^{-1} c_k,$$

which we can write as an SDP

$$\begin{aligned} & \text{minimize} && t_0 + \sum_k t_k \\ \text{subject to} & \begin{bmatrix} F(x) & \bar{c} \\ \bar{c}^T & t_0 \end{bmatrix} \succeq 0 \\ & \begin{bmatrix} F(x) & c_k \\ c_k^T & t_k \end{bmatrix} \succeq 0, \quad k = 1, \dots, m. \end{aligned}$$

3.12 A matrix fractional function. [?] Show that $X = B^T A^{-1} B$ solves the SDP

$$\begin{aligned} & \text{minimize} && \mathbf{tr} X \\ \text{subject to} & \begin{bmatrix} A & B \\ B^T & X \end{bmatrix} \succeq 0, \end{aligned}$$

with variable $X \in \mathbf{S}^n$, where $A \in \mathbf{S}_{++}^m$ and $B \in \mathbf{R}^{m \times n}$ are given.

Conclude that $\mathbf{tr}(B^T A^{-1} B)$ is a convex function of (A, B) , for A positive definite.

Solution. The constraint is equivalent to $X \succeq B^T A^{-1} B$. Therefore $\mathbf{tr} X \geq \mathbf{tr}(B^T A^{-1} B)$ for all feasible X , with equality if $X = B^T A^{-1} B$. This shows that $X = B^T A^{-1} B$ is optimal.

The optimal value can be expressed as $\mathbf{tr}(B^T A^{-1} B) = \inf_X F(X, A, B)$ where F is defined as

$$F(X, A, B) = \mathbf{tr} X,$$

with domain

$$\mathbf{dom} F = \left\{ (X, A, B) \in \mathbf{S}^m \times \mathbf{S}^m \times \mathbf{S}^m \mid A \succ 0, \quad \begin{bmatrix} A & B \\ B^T & X \end{bmatrix} \succeq 0 \right\}.$$

The function F is convex, jointly in A, B, X . Therefore its infimum over X , which is $\mathbf{tr}(B^T A^{-1} B)$, is convex in A, B .

3.13 Trace of harmonic mean of matrices. [?] The matrix $H(A, B) = 2(A^{-1} + B^{-1})^{-1}$ is known as the *harmonic mean* of positive definite matrices A and B . Show that $X = (1/2)H(A, B)$ solves the SDP

$$\begin{aligned} & \text{maximize} && \mathbf{tr} X \\ \text{subject to} & \begin{bmatrix} X & X \\ X & X \end{bmatrix} \preceq \begin{bmatrix} A & 0 \\ 0 & B \end{bmatrix}, \end{aligned}$$

with variable $X \in \mathbf{S}^n$. The matrices $A \in \mathbf{S}_{++}^m$ and $B \in \mathbf{S}_{++}^n$ are given. Conclude that the function $\mathbf{tr}((A^{-1} + B^{-1})^{-1})$, with domain $\mathbf{S}_{++}^m \times \mathbf{S}_{++}^n$, is concave.

Hint. Verify that the matrix

$$R = \begin{bmatrix} A^{-1} & I \\ B^{-1} & -I \end{bmatrix}$$

is nonsingular. Then apply the congruence transformation defined by R to the two sides of matrix inequality in the SDP, to obtain an equivalent inequality

$$R^T \begin{bmatrix} X & X \\ X & X \end{bmatrix} R \preceq R^T \begin{bmatrix} A & 0 \\ 0 & B \end{bmatrix} R.$$

Solution. We first show that the matrix is nonsingular. Assume

$$\begin{bmatrix} A^{-1} & I \\ B^{-1} & -I \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

From the first equation, $y = -A^{-1}x$. Substituting this in the second equation gives $B^{-1}x + A^{-1}x = 0$, and therefore $x^T(B^{-1} + A^{-1})x = 0$. Since A and B are positive definite this implies $x = 0$. If $x = 0$, then also $y = -A^{-1}x = 0$. This shows that the matrix has a zero nullspace, or, equivalently, its columns are linearly independent.

Following the hint, we write the constraint as

$$\begin{bmatrix} A^{-1} & B^{-1} \\ I & -I \end{bmatrix} \begin{bmatrix} X & X \\ X & X \end{bmatrix} \begin{bmatrix} A^{-1} & I \\ B^{-1} & -I \end{bmatrix} \preceq \begin{bmatrix} A^{-1} & B^{-1} \\ I & -I \end{bmatrix} \begin{bmatrix} A & 0 \\ 0 & B \end{bmatrix} \begin{bmatrix} A^{-1} & I \\ B^{-1} & -I \end{bmatrix}.$$

After working out the products we get

$$\begin{bmatrix} (A^{-1} + B^{-1})X(A^{-1} + B^{-1}) & 0 \\ 0 & 0 \end{bmatrix} \preceq \begin{bmatrix} A^{-1} + B^{-1} & 0 \\ 0 & A + B \end{bmatrix}.$$

This shows that the SDP is equivalent to

$$\begin{aligned} & \text{maximize} && \mathbf{tr} X \\ & \text{subject to} && X \preceq (A^{-1} + B^{-1})^{-1}. \end{aligned}$$

We have $\mathbf{tr} X \leq \mathbf{tr}((A^{-1} + B^{-1})^{-1})$ for all feasible X , with equality if $X = (A^{-1} + B^{-1})^{-1}$.

3.14 Trace of geometric mean of matrices. [?]

$$G(A, B) = A^{1/2} \left(A^{-1/2} B A^{-1/2} \right)^{1/2} A^{1/2}$$

is known as the *geometric mean* of positive definite matrices A and B . Show that $X = G(A, B)$ solves the SDP

$$\begin{aligned} & \text{maximize} && \mathbf{tr} X \\ & \text{subject to} && \begin{bmatrix} A & X \\ X & B \end{bmatrix} \succeq 0. \end{aligned}$$

The variable is $X \in \mathbf{S}^n$. The matrices $A \in \mathbf{S}_{++}^n$ and $B \in \mathbf{S}_{++}^n$ are given.

Conclude that the function $\mathbf{tr} G(A, B)$ is concave, for A, B positive definite.

Hint. The symmetric matrix square root is monotone: if U and V are positive semidefinite with $U \preceq V$ then $U^{1/2} \preceq V^{1/2}$.

Solution. Using Schur complements, we can write the constraint as

$$XA^{-1}X \preceq B,$$

and

$$(A^{-1/2}XA^{-1/2})^2 = A^{-1/2}XA^{-1}XA^{-1/2} \preceq A^{-1/2}BA^{-1/2}.$$

Using the hint, this implies that

$$\begin{aligned} A^{-1/2}XA^{-1/2} &\preceq (A^{-1/2}BA^{-1/2})^{1/2} \\ X &\preceq A^{1/2}(A^{-1/2}BA^{-1/2})^{1/2}A^{1/2}. \end{aligned}$$

We conclude that every feasible X satisfies $X \preceq G(A, B)$, and hence $\mathbf{tr} X \leq \mathbf{tr} G(A, B)$.

Moreover, $X = G(A, B)$ is feasible because

$$\begin{aligned} XA^{-1}X &= A^{1/2}(A^{-1/2}BA^{-1/2})^{1/2}A^{1/2}A^{-1}A^{1/2}(A^{-1/2}BA^{-1/2})^{1/2}A^{1/2} \\ &= A^{1/2}(A^{-1/2}BA^{-1/2})A^{1/2} \\ &= B. \end{aligned}$$

3.15 Transforming a standard form convex problem to conic form. In this problem we show that any convex problem can be cast in conic form, provided some technical conditions hold. We start with a standard form convex problem with linear objective (without loss of generality):

$$\begin{aligned} &\text{minimize} && c^T x \\ &\text{subject to} && f_i(x) \leq 0, \quad i = 1, \dots, m, \\ & && Ax = b, \end{aligned}$$

where $f_i : \mathbf{R}^n \rightarrow \mathbf{R}$ are convex, and $x \in \mathbf{R}^n$ is the variable. For simplicity, we will assume that $\mathbf{dom} f_i = \mathbf{R}^n$ for each i .

Now introduce a new scalar variable $t \in \mathbf{R}$ and form the convex problem

$$\begin{aligned} &\text{minimize} && c^T x \\ &\text{subject to} && tf_i(x/t) \leq 0, \quad i = 1, \dots, m, \\ & && Ax = b, \quad t = 1. \end{aligned}$$

Define

$$K = \mathbf{cl}\{(x, t) \in \mathbf{R}^{n+1} \mid tf_i(x/t) \leq 0, i = 1, \dots, m, t > 0\}.$$

Then our original problem can be expressed as

$$\begin{aligned} &\text{minimize} && c^T x \\ &\text{subject to} && (x, t) \in K, \\ & && Ax = b, \quad t = 1. \end{aligned}$$

This is a conic problem when K is proper.

You will relate some properties of the original problem to K .

- (a) Show that K is a convex cone. (It is closed by definition, since we take the closure.)
- (b) Suppose the original problem is strictly feasible, *i.e.*, there exists a point \bar{x} with $f_i(\bar{x}) < 0$, $i = 1, \dots, m$. (This is called Slater's condition.) Show that K has nonempty interior.
- (c) Suppose that the inequalities define a bounded set, *i.e.*, $\{x \mid f_i(x) \leq 0, i = 1, \dots, m\}$ is bounded. Show that K is pointed.

Solution.

- (a) The functions $tf_i(x/t)$ are convex, so the intersection of their 0-sublevel sets is convex. We see that it is a cone since if $(x, t) \in K$ and $\alpha > 0$ we have $(\alpha x, \alpha t) \in K$.
- (b) We have $(\bar{x}, 1) \in \text{int } K$, so $\text{int } K \neq \emptyset$.
- (c) To show K is pointed, assume that $(x, t) \in K$ and $-(x, t) \in K$. Since the second component is always nonnegative in K , we see that $t = 0$. So $(x, 0) \in K$, $(-x, 0) \in K$. Assume that $x \neq 0$. $(x, 0) \in K$ means that there are sequences $x_k \rightarrow x$, $t_k \rightarrow 0$, $t_k > 0$, with $t_k f(x_k/t_k) \leq 0$, $i = 1, \dots, m$. This is the same as $f(x_k/t_k) \leq 0$. But $x_k \rightarrow x \neq 0$, so x_k/t_k is an unbounded sequence, and therefore cannot be in $\{x \mid f_i(x) \leq 0, i = 1, \dots, m\}$. So we have a contradiction.

3.16 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^* of the given problem

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

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

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

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 $i = 1, \dots, n$. (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 $\epsilon = 0.01p^*$, 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 $k = 1, \dots, K$. Here $h > 0$ is the sampling period. These are related by the equations

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

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:

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

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

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

(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

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

(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,

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

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

- (a) Find an optimal trajectory, and the associated minimum fuel use p^* . 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.

```

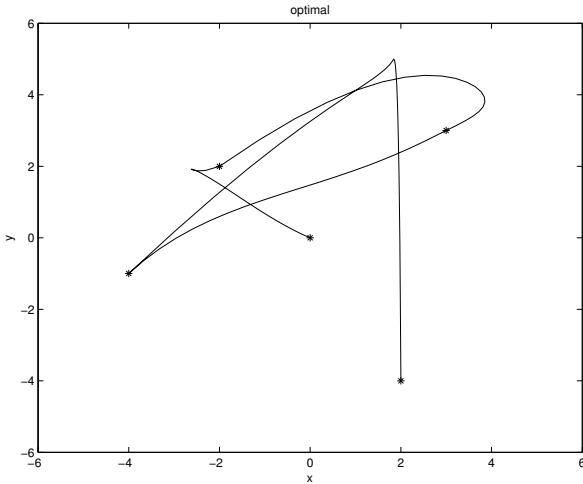
cvx_quiet(true);
thrusters_data;
F = [ cos(theta1) cos(theta2);...
       sin(theta1) sin(theta2)];

% 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;
    % constraints 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;
    % constraints on positions (x,y)
    p <= pmax;
    p >= -pmax;
cvx_end

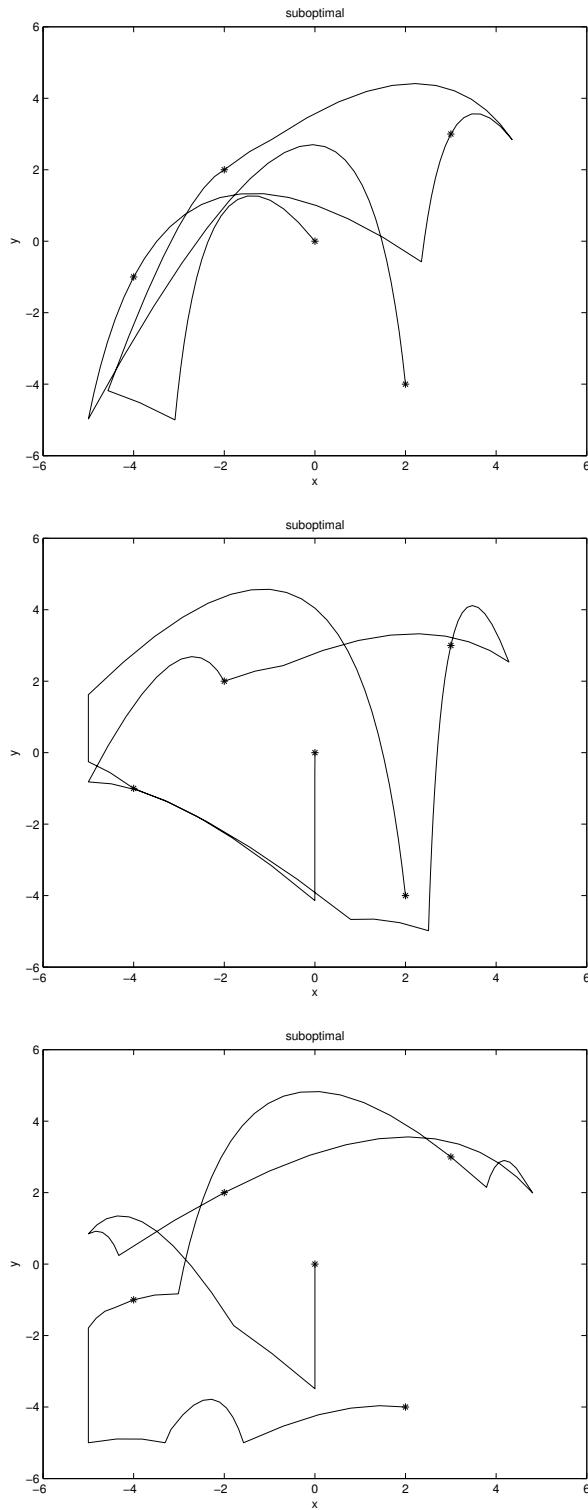
figure;
plot(p(1,:),p(2,:));
hold on
ps = [zeros(2,1) w1 w2 w3 w4];
plot(ps(1,:),ps(2,:),'*');
```

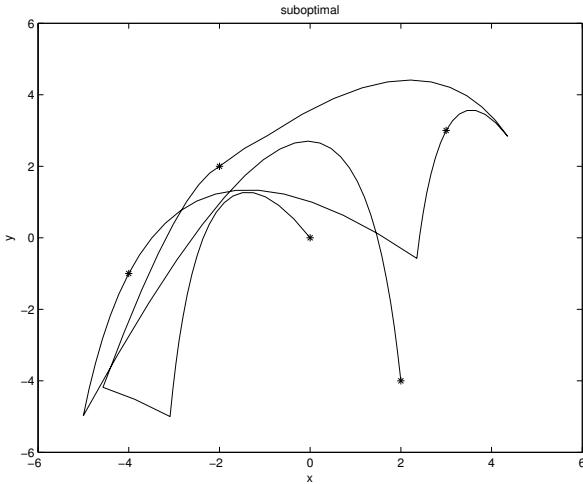
```

xlabel('x'); ylabel('y'); title('suboptimal');
axis([-6 6 -6 6]);

```

This script returns 4 randomly-generated nearly optimal trajectories.





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.

3.17 Minimum fuel optimal control. Solve the minimum fuel optimal control problem described in exercise 4.16 of *Convex Optimization*, for the instance with problem data

$$A = \begin{bmatrix} -1 & 0.4 & 0.8 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \quad b = \begin{bmatrix} 1 \\ 0 \\ 0.3 \end{bmatrix}, \quad x_{\text{des}} = \begin{bmatrix} 7 \\ 2 \\ -6 \end{bmatrix}, \quad N = 30.$$

You can do this by forming the LP you found in your solution of exercise 4.16, or more directly using CVX. Plot the actuator signal $u(t)$ as a function of time t .

Solution. The following Matlab code finds the solution

```

close all
clear all

n=3; % state dimension
N=30; % time horizon

A=[ -1 0.4 0.8; 1 0 0 ; 0 1 0];
b=[ 1 0 0.3]';
x0 = zeros(n,1);
xdes = [ 7 2 -6]';

cvx_begin
    variable X(n,N+1);
    variable u(1,N);
    minimize (sum(max(abs(u),2*abs(u)-1)))
    subject to
        X(:,2:N+1) == A*X(:,1:N)+b*u; % dynamics
        X(:,1) == x0;

```

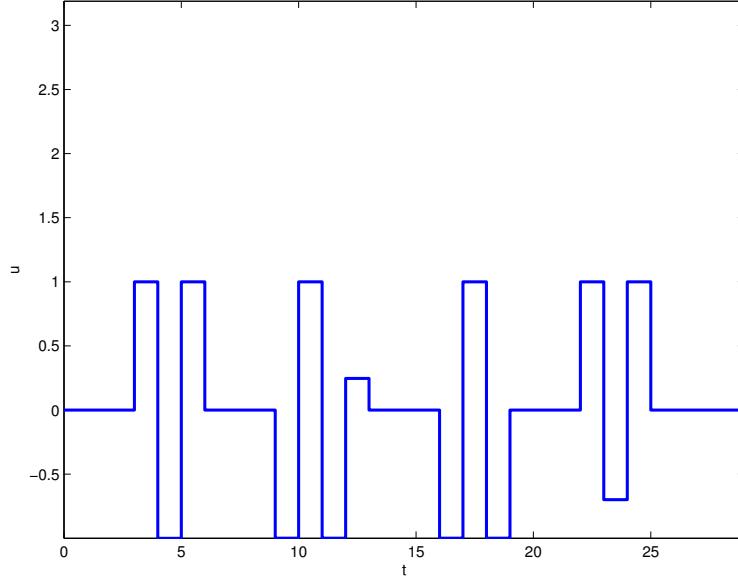


Figure 1: Minimum fuel actuator signal.

```
X(:,N+1) == xdes;
cvx_end

stairs(0:N-1,u,'linewidth',2)
axis tight
xlabel('t')
ylabel('u')
```

The optimal actuator signal is shown in figure 1.

3.18 Heuristic suboptimal solution for Boolean LP. This exercise builds on exercises 4.15 and 5.13 in *Convex Optimization*, which involve the Boolean LP

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

with optimal value p^* . Let x^{rlx} be a solution of the LP relaxation

$$\begin{aligned} & \text{minimize} && c^T x \\ & \text{subject to} && Ax \preceq b \\ & && 0 \preceq x \preceq \mathbf{1}, \end{aligned}$$

so $L = c^T x^{\text{rlx}}$ is a lower bound on p^* . The relaxed solution x^{rlx} can also be used to guess a Boolean point \hat{x} , by rounding its entries, based on a threshold $t \in [0, 1]$:

$$\hat{x}_i = \begin{cases} 1 & x_i^{\text{rlx}} \geq t \\ 0 & \text{otherwise,} \end{cases}$$

for $i = 1, \dots, n$. Evidently \hat{x} is Boolean (*i.e.*, has entries in $\{0, 1\}$). If it is feasible for the Boolean LP, *i.e.*, if $A\hat{x} \leq b$, then it can be considered a guess at a good, if not optimal, point for the Boolean LP. Its objective value, $U = c^T \hat{x}$, is an upper bound on p^* . If U and L are close, then \hat{x} is nearly optimal; specifically, \hat{x} cannot be more than $(U - L)$ -suboptimal for the Boolean LP.

This rounding need not work; indeed, it can happen that for all threshold values, \hat{x} is infeasible. But for some problem instances, it can work well.

Of course, there are many variations on this simple scheme for (possibly) constructing a feasible, good point from x^{rlx} .

Finally, we get to the problem. Generate problem data using one of the following.

Matlab code:

```
rand('state',0);
n=100;
m=300;
A=rand(m,n);
b=A*ones(n,1)/2;
c=-rand(n,1);
```

Python code:

```
import numpy as np
np.random.seed(0)
(m, n) = (300, 100)
A = np.random.rand(m, n); A = np.asmatrix(A)
b = A.dot(np.ones((n, 1)))/2; b = np.asmatrix(b)
c = -np.random.rand(n, 1); c = np.asmatrix(c)
```

Julia code:

```
srand(0);
n=100;
m=300;
A=rand(m,n);
b=A*ones(n,1)/2;
c=-rand(n,1);
```

You can think of x_i as a job we either accept or decline, and $-c_i$ as the (positive) revenue we generate if we accept job i . We can think of $Ax \leq b$ as a set of limits on m resources. A_{ij} , which is positive, is the amount of resource i consumed if we accept job j ; b_i , which is positive, is the amount of resource i available.

Find a solution of the relaxed LP and examine its entries. Note the associated lower bound L . Carry out threshold rounding for (say) 100 values of t , uniformly spaced over $[0, 1]$. For each value of t , note the objective value $c^T \hat{x}$ and the maximum constraint violation $\max_i (A\hat{x} - b)_i$. Plot the objective value and the maximum violation versus t . Be sure to indicate on the plot the values of t for which \hat{x} is feasible, and those for which it is not.

Find a value of t for which \hat{x} is feasible, and gives minimum objective value, and note the associated upper bound U . Give the gap $U - L$ between the upper bound on p^* and the lower bound on p^* .

In Matlab, if you define vectors `obj` and `maxviol`, you can find the upper bound as `U=min(obj(find(maxviol<=0)))`

Matlab Solution

The following Matlab code finds the solution

```
% generate data for boolean LP relaxation & heuristic
rand('state',0);
n=100;
m=300;
A=rand(m,n);
b=A*ones(n,1)/2;
c=-rand(n,1);

% solve LP relaxation
cvx_begin
    variable x(n)
    minimize (c'*x)
    subject to
        A*x <= b
        x>=0
        x<=1
cvx_end
xrlx = x;
L=cvx_optval;

% sweep over threshold & round
thres=0:0.01:1;
maxviol = zeros(length(thres),1);
obj = zeros(length(thres),1);
for i=1:length(thres)
    xhat = (xrlx>=thres(i));
    maxviol(i) = max(A*xhat-b);
    obj(i) = c'*xhat;
end

% find least upper bound and associated threshold
i_feas=find(maxviol<=0);
U=min(obj(i_feas))
%U=min(obj(find(maxviol <=0)))
t=min(i_feas);
min_thresh=thres(t)

% plot objective and max violation versus threshold
subplot(2,1,1)
```

```

plot(thres(1:t-1),maxviol(1:t-1),'r',thres(t:end),maxviol(t:end),'b','linewidth',2);
xlabel('threshold');
ylabel('max violation');
subplot(2,1,2)
hold on; plot(thres,L*ones(size(thres)),'k','linewidth',2);
plot(thres(1:t-1),obj(1:t-1),'r',thres(t:end),obj(t:end),'b','linewidth',2);
xlabel('threshold');
ylabel('objective');

```

The lower bound found from the relaxed LP is $L = -33.1672$. We find that the threshold value $t = 0.6006$ gives the best (smallest) objective value for feasible \hat{x} : $U = -32.4450$. The difference is 0.7222. So \hat{x} , with $t = 0.6006$, can be no more than 0.7222 suboptimal, *i.e.*, around 2.2% suboptimal.

In figure 2, the red lines indicate values for thresholding values which give infeasible \hat{x} , and the blue lines correspond to feasible \hat{x} . We see that the maximum violation decreases as the threshold is increased. This occurs because the constraint matrix A only has nonnegative entries. At a threshold of 0, all jobs are selected, which is an infeasible solution. As we increase the threshold, projects are removed in sequence (without adding new projects), which monotonically decreases the maximum violation. For a general boolean LP, the corresponding plots need not exhibit monotonic behavior.

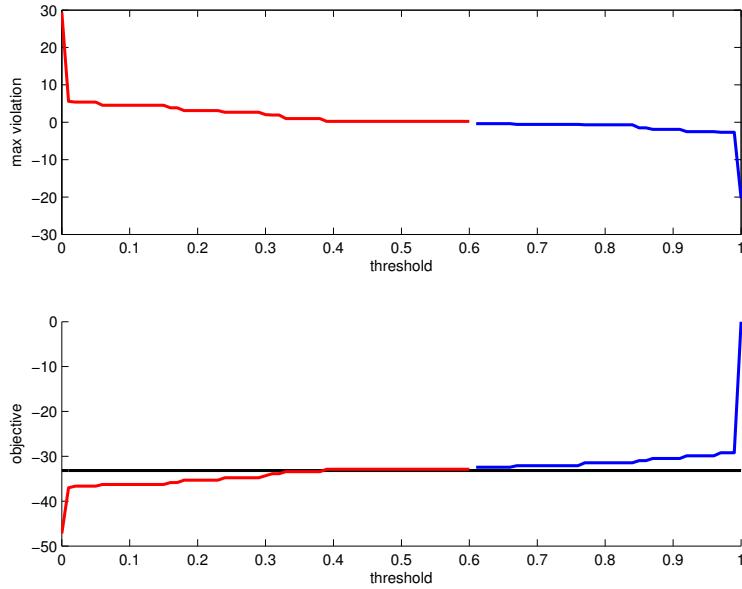


Figure 2: Plots of violation and objective vs threshold rule, for the Matlab code.

Python Solution

The following Python code finds the solution

```

import cvxpy as cvx
import numpy as np

```

```

import matplotlib.pyplot as plt

np.random.seed(0)
(m, n) = (300, 100)
A = np.random.rand(m, n); A = np.asmatrix(A)
b = A.dot(np.ones((n, 1)))/2; b = np.asmatrix(b)
c = -np.random.rand(n, 1); c = np.asmatrix(c)

#Solve relaxed LP
x = cvx.Variable(n)
objective = cvx.Minimize(c.T*x)
constraints = [0<=x, x<=1, A*x<=b]
cvx.Problem(objective, constraints).solve()
L = objective.value
x_rlx = x.value

#Rounding parameters
N = 100
t = np.linspace(0, 1, num=N).reshape(N, 1)
maxviol = np.zeros((N, 1))
obj = np.zeros((N, 1))
U = float('inf')
t_best = float('nan')

#Round
for i in range(N):
    x = np.matrix(x_rlx >= t[i], dtype = float)
    obj[i] = c.T*x
    maxviol[i] = max(A*x-b)
    if maxviol[i]<=0 and obj[i]<U:
        U = float(obj[i])
        x_best = x
        t_best = t[i]

#Plot
plt.figure(1)
plt.subplot(211)
plt.plot(t[maxviol<=0], maxviol[maxviol<=0], 'b')
plt.plot(t[maxviol>0], maxviol[maxviol>0], 'r')
plt.ylabel('max violation')
plt.xlabel('threshold')

plt.subplot(212)
plt.plot(t[maxviol<=0], obj[maxviol<=0], 'b')

```

```

plt.plot(t[maxviol>0], obj[maxviol>0], 'r')
plt.plot(t, objective.value*np.ones((N,1)), 'g')
plt.ylabel('objective')
plt.xlabel('threshold')
plt.savefig('figures/boolean_lp_heur_py.eps')
plt.show()

```

The lower bound found from the relaxed LP is $L = -34.42$. We find that the threshold value $t = 0.56$ gives the best (smallest) objective value for feasible \hat{x} : $U = -33.58$. The difference is 0.84. So \hat{x} , with $t = 0.56$, can be no more than 0.84 suboptimal, *i.e.*, around 2.5% suboptimal.

In figure 3, the red lines indicate values for thresholding values which give infeasible \hat{x} , and the blue lines correspond to feasible \hat{x} . We see that the maximum violation decreases as the threshold is increased. This occurs because the constraint matrix A only has nonnegative entries. At a threshold of 0, all jobs are selected, which is an infeasible solution. As we increase the threshold, projects are removed in sequence (without adding new projects), which monotonically decreases the maximum violation. For a general boolean LP, the corresponding plots need not exhibit monotonic behavior.

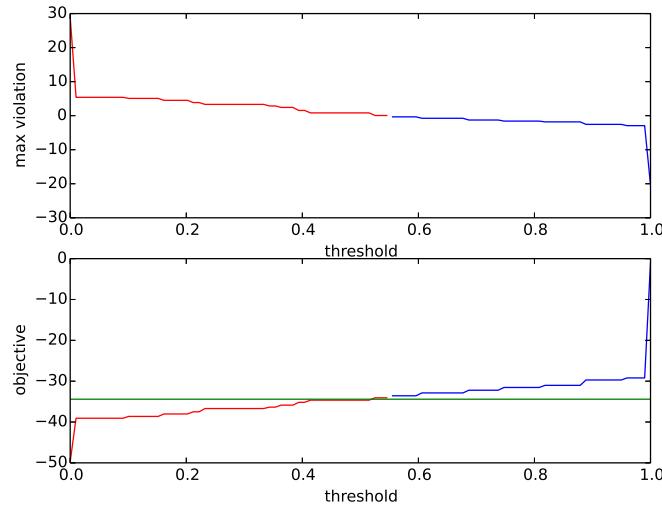


Figure 3: Plots of violation and objective vs threshold rule, for the Python code.

Julia Solution

The following Julia code finds the solution

```

using Convex, Gadfly

# generate data for boolean LP relaxation & heuristic
srand(0);
n=100;
m=300;

```

```

A=rand(m,n);
b=A*ones(n,1)/2;
c=-rand(n,1);

# solve LP relaxation
x = Variable(n);
p = minimize(c'*x, A*x <= b, x >= 0, x<= 1);
solve!(p);
xrlx = x.value;
L = p.optval;

# sweep over threshold & round
thres=0:0.01:1;
maxviol = zeros(length(thres),1);
obj = zeros(length(thres),1);
for i = 1:length(thres)
    xhat = xrlx .>= thres[i];
    maxviol[i] = maximum(A*xhat-b);
    obj[i] = (c'*xhat)[1];
end

# find least upper bound and associated threshold
i_feas = find(maxviol.<=0);
U = minimum(obj[i_feas])
t = minimum(i_feas);
min_thresh = thres[t];

# plot objective and max violation versus threshold
p1 = plot(
    layer(
        x = thres[1:t-1],
        y = maxviol[1:t-1],
        Geom.line,
        Theme(default_color = color("red"))
    ),
    layer(
        x = thres[t:end],
        y = maxviol[t:end],
        Geom.line,
        Theme(default_color = color("blue"))
    ),
    Guide.xlabel("threshold"),
    Guide.ylabel("max violation")
);
draw(PS("boolean_lp_heur_jl_1.eps", 6inch, 2inch), p1)

```

```

p2 = plot(
  layer(
    x = thres[1:t-1],
    y = obj[1:t-1],
    Geom.line,
    Theme(default_color = color("red"))
  ),
  layer(
    x = thres[t:end],
    y = obj[t:end],
    Geom.line,
    Theme(default_color = color("blue"))
  ),
  layer(
    x = thres,
    y = L*ones(length(thres)),
    Geom.line,
    Theme(default_color = color("black"))
  ),
  Guide.xlabel("threshold"),
  Guide.ylabel("objective")
);
draw(PS("boolean_lp_heur_jl_2.eps", 6inch, 2inch), p2)

```

The lower bound found from the relaxed LP is $L = -33.828$. We find that the threshold value $t = 0.54$ gives the best (smallest) objective value for feasible \hat{x} : $U = -32.601$. The difference is 1.23. So \hat{x} , with $t = 0.54$, can be no more than 1.23 suboptimal, *i.e.*, around 3.6% suboptimal.

In figure 4, the red lines indicate values for thresholding values which give infeasible \hat{x} , and the blue lines correspond to feasible \hat{x} . We see that the maximum violation decreases as the threshold is increased. This occurs because the constraint matrix A only has nonnegative entries. At a threshold of 0, all jobs are selected, which is an infeasible solution. As we increase the threshold, projects are removed in sequence (without adding new projects), which monotonically decreases the maximum violation. For a general boolean LP, the corresponding plots need not exhibit monotonic behavior.

- 3.19** *Optimal operation of a hybrid vehicle.* Solve the instance of the hybrid vehicle operation problem described in exercise 4.65 in *Convex Optimization*, with problem data given in the file `hybrid_veh_data.*`, and fuel use function $F(p) = p + \gamma p^2$ (for $p \geq 0$).

Hint. You will actually formulate and solve a *relaxation* of the original problem. You may find that some of the equality constraints you relaxed to inequality constraints do not hold for the solution found. This is not an error: it just means that there is no incentive (in terms of the objective) for the inequality to be tight. You can fix this in (at least) two ways. One is to go back and adjust certain variables, without affecting the objective and maintaining feasibility, so that the relaxed constraints hold with equality. Another simple method is to add to the objective a term of the

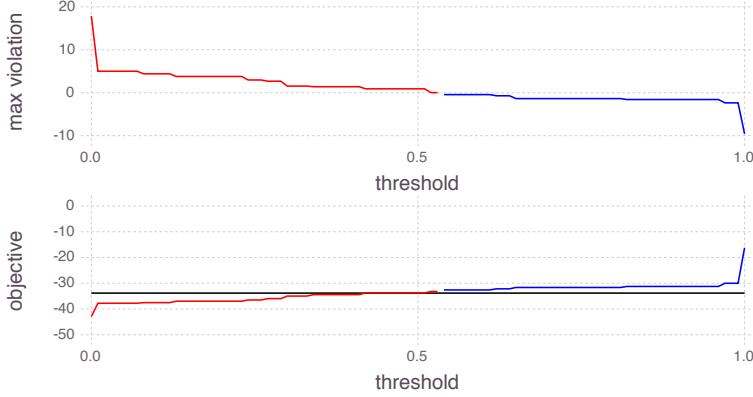


Figure 4: Plots of violation and objective vs threshold rule, for the Julia code.

form

$$\epsilon \sum_{t=1}^T \max\{0, -P_{mg}(t)\},$$

where ϵ is small and positive. This makes it more attractive to use the brakes to extract power from the wheels, even when the battery is (or will be) full (which removes any fuel incentive).

Find the optimal fuel consumption, and compare to the fuel consumption with a non-hybrid version of the same vehicle (*i.e.*, one without a battery). Plot the braking power, engine power, motor/generator power, and battery energy versus time.

How would you use optimal dual variables for this problem to find $\partial F_{\text{total}}/\partial E_{\text{batt}}^{\max}$, *i.e.*, the partial derivative of optimal fuel consumption with respect to battery capacity? (You can just assume that this partial derivative exists.) You do not have to give a long derivation or proof; you can just state how you would find this derivative from optimal dual variables for the problem. Verify your method numerically, by changing the battery capacity a small amount and re-running the optimization, and comparing this to the prediction made using dual variables.

Solution.

The code for solving this instance is given below. The optimal fuel consumption of the vehicle with the battery is $F_{\text{total}} = 5077.53$. Without the battery the fuel consumption goes up to $F_{\text{total}} = 5896.81$. The battery can shift energy in time and recover energy that would have been lost in friction braking, so we expect the fuel use to be smaller than without a battery.

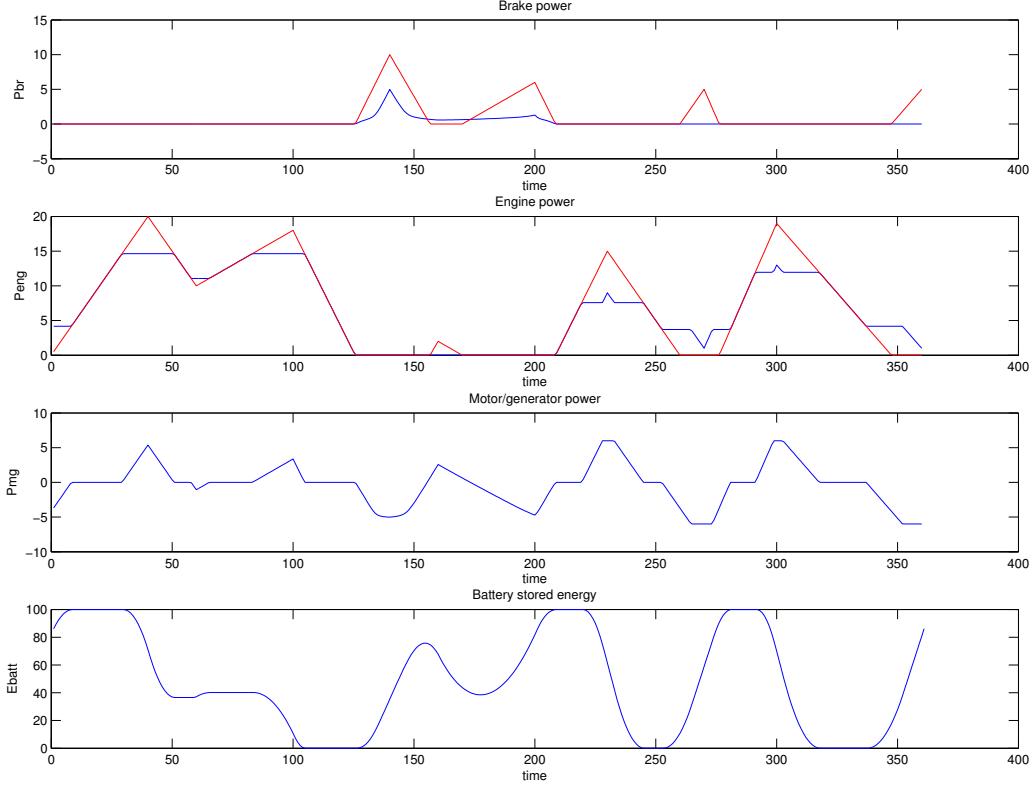
Closely examining the optimal variables reveals that there is a short period when the relaxed battery energy conservation constraint *does not hold*: We actually elect to throw battery energy away. This is a brief period before $t \approx 200$, when the battery is already full, so no further charging will make any difference at all. This is not a problem: The optimal fuel consumption is correct. The constraint can be made tight by following the method in the hint, without affecting fuel optimality.

To find $\partial F_{\text{total}}/\partial E_{\text{batt}}^{\max}$, we let $\lambda^*(t)$ be an optimal dual variable associated with the constraint $E(t) \leq E_{\text{batt}}^{\max}$. We interpret $-\lambda^*(t)$ as the partial derivative of optimal fuel use with respect to E_{batt}^{\max} at time t . It tells us how the fuel usage would improve if, at time t only, we were able to

store more energy in the battery. To get the full partial derivative $\partial F_{\text{total}}/\partial E_{\text{batt}}^{\max}$, we sum:

$$\frac{\partial F_{\text{total}}}{\partial E_{\text{batt}}^{\max}} = - \sum_{t=1}^T \lambda^*(t).$$

This formula gives us $\partial F_{\text{total}}/\partial E_{\text{batt}}^{\max} = -4.9680$. We can verify this numerically by perturbing the constraint by a small amount and running the optimization again. Setting $E(t) \leq E_{\text{batt}}^{\max} + \Delta E$, $\Delta E = 0.1$ we obtain $\partial F_{\text{total}}/\partial E_{\text{batt}}^{\max} \approx \frac{\Delta F}{\Delta E} = -4.9658$, where ΔF is the difference in fuel consumption between the perturbed and unperturbed cases. The numerical answer is indeed very close to the answer obtained from the dual variables.



```
% instance of hybrid vehicle optimization problem,
% exercise 4.65 in Boyd & Vandenberghe, Convex Optimization
```

```
hybrid_veh_data
```

```
% solution via CVX
```

```
epsilon=1e-5;
```

```

cvx_begin
cvx_quiet(true)
variables Peng(T) Pbr(T) Pmg(T) E(T+1)
dual variable lambda
% minimize (sum(Peng+gamma*square(Peng))) % total fuel use
minimize (sum(Peng+gamma*square(Peng))+epsilon*sum(pos(-Pmg)))
Preq == Peng + Pmg - Pbr; % power balance
E(1) == E(T+1); % starting and ending battery energy match
Pmg_min <= Pmg; % maximum generator power
Pmg <= Pmg_max; % maximum motor power
0 <= E;
% assign the dual variable to this set of constraints
% for use in sensitivity analysis
lambda: E <= Ebatt_max; % battery capacity
0 <= Peng;
Peng <= Peng_max; % max engine power
Pbr >= 0;
for t=1:T
    E(t+1)<=E(t)-Pmg(t)-eta*abs(Pmg(t));
    % this is a relaxation; true formula is below
    % E(t+1)==E(t)-Pmg(t)-eta*abs(Pmg(t));
    % extra term above is used to gaurantee tightness at optimal
end
cvx_end
Ftot = cvx_optval

% verify that the constraint holds with equality
DE=E(2:T+1)-E(1:T)+Pmg(1:T)+eta*abs(Pmg(1:T));
if max(abs(DE))<1e-3
    fprintf('battery energy constraint holds with equality \n')
end

% solve the problem for a vehicle without a battery
% can do this analytically, but no harm in using CVX
cvx_begin
cvx_quiet(true)
variables Peng_nobatt(T) Pbr_nobatt(T)
minimize (sum(Peng_nobatt+gamma*square(Peng_nobatt))) % total fuel use
Preq == Peng_nobatt - Pbr_nobatt;
0 <= Peng_nobatt;
Peng_nobatt <= Peng_max;
Pbr_nobatt >= 0;
cvx_end
Ftot_no_batt = cvx_optval

```

```

% generate the plots
scrsz = get(0,'ScreenSize');
figure('Position',[1 1 scrsz(3) scrsz(4)])
subplot(4,1,1);
plot(Pbr); hold on; plot(Pbr_nobatt,'r')
title('Brake power'); xlabel('time'); ylabel('Pbr');
%legend('With battery','Without battery')
subplot(4,1,2);
plot(Peng); hold on; plot(Peng_nobatt, 'r')
title('Engine power'); xlabel('time'); ylabel('Peng');
%legend('With battery','Without battery')
subplot(4,1,3);
plot(Pmg)
title('Motor/generator power'); xlabel('time'); ylabel('Pmg');
subplot(4,1,4);
plot(E)
title('Battery stored energy'); xlabel('time'); ylabel('Ebatt');

% numerical verification of sensitivity analysis
% perturb Ebatt_max by deltaE and examine the change in F
deltaE=0.1;
cvx_begin
cvx_quiet(true)
variables Peng(T) Pbr(T) Pmg(T) E(T+1)
% minimize (sum(Peng+gamma*square(Peng))) % total fuel use
minimize (sum(Peng+gamma*square(Peng))+epsilon*sum(pos(-Pmg)))
Preq == Peng + Pmg - Pbr;
E(1) == E(T+1);
Pmg_min <= Pmg;
Pmg <= Pmg_max;
0 <= E;
E <= Ebatt_max+deltaE;
0 <= Peng;
Peng <= Peng_max;
Pbr >= 0;
for t=1:T
    E(t+1)<=E(t)-Pmg(t)-eta*abs(Pmg(t));
end
cvx_end
Ftot_deltaE=cvx_optval;
% calculate sensitivity numerically
dFdE_num=(Ftot_deltaE-Ftot)/deltaE
% take the negative of the sum of the dual variables
% to obtain dFtot/dEbatt_max
dFdE_tot=-sum(lambda)

```

3.20 Optimal vehicle speed scheduling. A vehicle (say, an airplane) travels along a fixed path of n segments, between $n + 1$ waypoints labeled $0, \dots, n$. Segment i starts at waypoint $i - 1$ and terminates at waypoint i . The vehicle starts at time $t = 0$ at waypoint 0. It travels over each segment at a constant (nonnegative) speed; s_i is the speed on segment i . We have lower and upper limits on the speeds: $s^{\min} \leq s \leq s^{\max}$. The vehicle does not stop at the waypoints; it simply proceeds to the next segment. The travel distance of segment i is d_i (which is positive), so the travel time over segment i is d_i/s_i . We let τ_i , $i = 1, \dots, n$, denote the time at which the vehicle arrives at waypoint i . The vehicle is required to arrive at waypoint i , for $i = 1, \dots, n$, between times τ_i^{\min} and τ_i^{\max} , which are given. The vehicle consumes fuel over segment i at a rate that depends on its speed, $\Phi(s_i)$, where Φ is positive, increasing, and convex, and has units of kg/s.

You are given the data d (segment travel distances), s^{\min} and s^{\max} (speed bounds), τ^{\min} and τ^{\max} (waypoint arrival time bounds), and the fuel use function $\Phi : \mathbf{R} \rightarrow \mathbf{R}$. You are to choose the speeds s_1, \dots, s_n so as to minimize the total fuel consumed in kg.

- (a) Show how to pose this as a convex optimization problem. If you introduce new variables, or change variables, you must explain how to recover the optimal speeds from the solution of your problem. If convexity of the objective or any constraint function in your formulation is not obvious, explain why it is convex.
- (b) Carry out the method of part (a) on the problem instance with data in `veh_speed_sched_data.m`. Use the fuel use function $\Phi(s_i) = as_i^2 + bs_i + c$ (the parameters a , b , and c are defined in the data file). What is the optimal fuel consumption? Plot the optimal speed versus segment, using the matlab command `stairs` or the function `step` from matplotlib in Python and Julia to better show constant speed over the segments.

Solution.

- (a) The fuel consumed over the i th segment is $(d_i/s_i)\Phi(s_i)$, so the total fuel used is $\sum_{i=1}^n (d_i/s_i)\Phi(s_i)$. The vehicle arrives at waypoint i at time $\tau_i = \sum_{j=1}^i (d_j/s_j)$. Thus our problem is

$$\begin{aligned} & \text{minimize} && \sum_{i=1}^n (d_i/s_i)\Phi(s_i) \\ & \text{subject to} && s_i^{\min} \leq s_i \leq s_i^{\max} \quad i = 1, \dots, n \\ & && \tau_i^{\min} \leq \sum_{j=1}^i (d_j/s_j) \leq \tau_i^{\max} \quad i = 1, \dots, n, \end{aligned}$$

with variables s_1, \dots, s_n .

In this form, this is *not* a convex problem: the objective function need not be convex in s_i , and the inequalities $\tau_i^{\min} \leq \sum_{j=1}^i (d_j/s_j)$ are not convex.

However, we can formulate this as a convex problem by making a change of variables. We formulate the problem using the transit times of the segments, t_i , as the optimization variable, where $t_i = d_i/s_i$. (We then have $s_i = d_i/t_i$.) Our problem can be written as

$$\begin{aligned} & \text{minimize} && \sum_{i=1}^n t_i\Phi(d_i/t_i) \\ & \text{subject to} && d_i/s_i^{\max} \leq t_i \leq d_i/s_i^{\min} \quad i = 1, \dots, n \\ & && \tau_i^{\min} \leq \sum_{j=1}^i t_j \leq \tau_i^{\max} \quad i = 1, \dots, n, \end{aligned}$$

with variables t_1, \dots, t_n .

This is a convex problem. The function $t_i\Phi(d_i/t_i)$, the perspective of Φ , is convex jointly in d_i and t_i ; in particular, it is convex in t_i . Therefore the objective function is convex, since it

is a positive weighted sum of convex functions. The constraints are all linear in t . Once we have solved the problem for t_i^* we recover the optimal speeds using $s_i^* = d_i/t_i^*$.

- (b) The optimal fuel consumption is 2617.83 kg. The code below solves the problem in Matlab:

```
% solution to vehicle speed scheduling problem
veh_speed_sched_data
```

```
cvx_begin
    variable t(n)
    minimize(sum(a*d.^2.*inv_pos(t)+b*d+c*t))
    t<=d./smin;
    t>=d./smax;
    tau_min<=cumsum(t);
    tau_max>=cumsum(t);
cvx_end
s=d./t;

stairs(s)
title('speed over segment'); xlabel('i'); ylabel('s_i')
%print('-depsc','veh_speed.eps')
```

Here is a solution in Python.

```
# solution to vehicle speed scheduling problem

import numpy as np
import cvxpy as cvx
import matplotlib.pyplot as plt
from veh_speed_sched_data import *

t = cvx.Variable(n)
obj = cvx.sum_entries(a*cvx.mul_elementwise(cvx.square(d), cvx.inv_pos(t)) + b*d + c*t)
cons = [t <= d / smin, t >= d / smax]
cons += [tau_min[i] <= cvx.sum_entries(t[0:i+1]) for i in range(n)]
cons += [tau_max[i] >= cvx.sum_entries(t[0:i+1]) for i in range(n)]

cvx.Problem(cvx.Minimize(obj), cons).solve()

s = d / t.value

plt.step(np.arange(n), s)
plt.title('speed over segment')
plt.xlabel('$i$')
plt.ylabel('$s_i$')
plt.savefig("veh_speed_sched.eps")
plt.show()
```

Finally, here is a Julia solution.

```

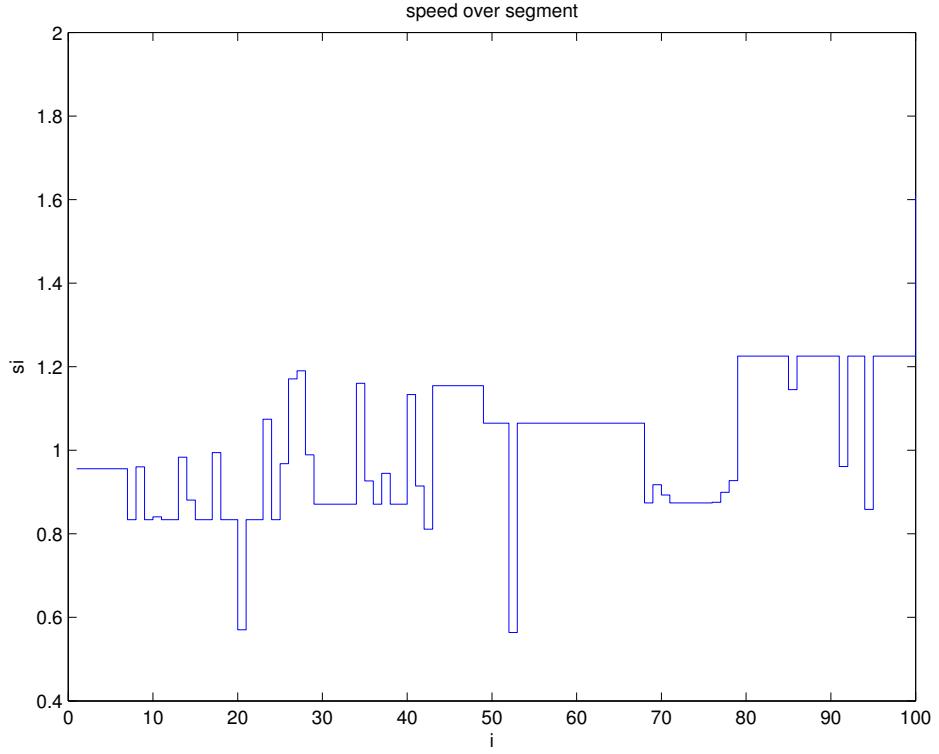
# solution to vehicle speed scheduling problem
include("veh_speed_sched_data.jl")

cumsum_mat = zeros(n, n);
for i = 1:n
    for j = 1:i
        cumsum_mat[i,j] = 1
    end
end

using Convex, SCS
t = Variable(n);
constraints = [
    t <= d./smin;
    t >= d./smax;
    tau_min <= cumsum_mat*t;
    tau_max >= cumsum_mat*t;
];
p = minimize(sum(a*(d.^2).*inv_pos(t) + b*d + c*t), constraints);
solve!(p, SCSSolver(max_iters=15000));
s = vec(d./t.value);

using PyPlot
step(1:length(s), s);
xlabel("i");
ylabel("s_i");
title("speed over segment");

```



3.21 Norm approximation via SOCP, for ℓ_p -norms with rational p .

- (a) Use the observation at the beginning of exercise 4.26 in *Convex Optimization* to express the constraint

$$y \leq \sqrt{z_1 z_2}, \quad y, z_1, z_2 \geq 0,$$

with variables y, z_1, z_2 , as a second-order cone constraint. Then extend your result to the constraint

$$y \leq (z_1 z_2 \cdots z_n)^{1/n}, \quad y \geq 0, \quad z \succeq 0,$$

where n is a positive integer, and the variables are $y \in \mathbf{R}$ and $z \in \mathbf{R}^n$. First assume that n is a power of two, and then generalize your formulation to arbitrary positive integers.

- (b) Express the constraint

$$f(x) \leq t$$

as a second-order cone constraint, for the following two convex functions f :

$$f(x) = \begin{cases} x^\alpha & x \geq 0 \\ 0 & x < 0, \end{cases}$$

where α is rational and greater than or equal to one, and

$$f(x) = x^\alpha, \quad \text{dom } f = \mathbf{R}_{++},$$

where α is rational and negative.

(c) Formulate the norm approximation problem

$$\text{minimize } \|Ax - b\|_p$$

as a second-order cone program, where p is a rational number greater than or equal to one. The variable in the optimization problem is $x \in \mathbf{R}^n$. The matrix $A \in \mathbf{R}^{m \times n}$ and the vector $b \in \mathbf{R}^m$ are given. For an m -vector y , the norm $\|y\|_p$ is defined as

$$\|y\|_p = \left(\sum_{k=1}^m |y_k|^p \right)^{1/p}$$

when $p \geq 1$.

Solution.

(a) The constraints are equivalent to

$$\left\| \begin{bmatrix} 2t \\ z_1 - z_2 \end{bmatrix} \right\| \leq z_1 + z_2, \quad t \geq 0.$$

Expanding the norm inequality gives $t^2 \leq z_1 z_2$ and $z_1 + z_2 \geq 0$. It is therefore equivalent to $|t| \leq \sqrt{z_1 z_2}$ and $z_1, z_2 \geq 0$. If we also constrain t to be nonnegative, we obtain the constraint in the problem statement.

For the second constraint, first suppose $n = 2^l$. Introduce variables y_{ij} for $i = 1, \dots, l-1$, and $j = 1, \dots, 2^i$, and write the constraint

$$y \leq (z_1 z_2 \cdots z_n)^{1/n}, \quad y \geq 0, \quad z \succeq 0,$$

as the following set of inequalities:

$$\begin{aligned} z &\succeq 0, \\ y_{l-1,j} &\leq (z_{2j-1} z_{2j})^{1/2}, \quad y_{l-1,j} \geq 0, \quad j = 1, \dots, 2^{l-1} \\ y_{ij} &\leq (y_{i+1,2*j-1} y_{i+1,2j})^{1/2}, \quad y_{ij} \geq 0, \quad i = 1, \dots, l-2, \quad j = 1, \dots, 2^i \\ y &\leq (y_{11} y_{12})^{1/2}, \quad y \geq 0. \end{aligned}$$

Then apply the second order cone formulation as in part (a).

For general n we write the constraint as

$$y \leq (y^{m-n} z_1 \cdots z_n)^{1/m}, \quad y \geq 0, \quad z \succeq 0,$$

where m is the smallest power of two that is greater than or equal to n , and then use the previous formulation.

(b) For the first function, write the constraint as

$$y \leq (t^s)^{1/r}, \quad y \geq x, \quad y \geq 0, \quad t \geq 0$$

where $\alpha = r/s$ and r and s are integers, and then apply part (a) with $z_k = t$ for $k = 1, \dots, s$ and $z_k = 1$ for $k = s+1, \dots, r$.

For the second function, we express the constraint as

$$1 \leq (x^r t^s)^{1/(r+s)}, \quad x \geq 0, \quad t \geq 0,$$

where $\alpha = -r/s$ and r and s are integers, and then apply the formulation of part (a).

(c) First express the problem as

$$\begin{aligned} & \text{minimize} && \sum_{k=1}^m t_k \\ & \text{subject to} && -y \preceq Ax - b \preceq y \\ & && y_k^p \leq t_k, \quad k = 1, \dots, m \end{aligned}$$

and use part (b).

3.22 *Linear optimization over the complement of a convex set.* Suppose $\mathcal{C} \subseteq \mathbf{R}_+^n$ is a closed bounded convex set with $0 \in \mathcal{C}$, and $c \in \mathbf{R}_+^n$. We define

$$\tilde{\mathcal{C}} = \mathbf{cl}(\mathbf{R}_+^n \setminus \mathcal{C}) = \mathbf{cl}\{x \in \mathbf{R}_+^n \mid x \notin \mathcal{C}\},$$

which is the closure of the complement of \mathcal{C} in \mathbf{R}_+^n .

Show that $c^T x$ has a minimizer over $\tilde{\mathcal{C}}$ of the form αe_k , where $\alpha \geq 0$ and e_k is the k th standard unit vector. (If you have not had a course on analysis, you can give an intuitive argument.)

It follows that we can minimize $c^T x$ over $\tilde{\mathcal{C}}$ by solving n one-dimensional optimization problems (which, indeed, can each be solved by bisection, provided we can check whether a point is in \mathcal{C} or not).

Solution. Suppose the claim is false, and let x^* be an optimal point, so $p^* = c^T x^*$ (we know the optimum is attained, since $\tilde{\mathcal{C}}$ is closed and $c \succeq 0$). If $c_i = 0$ for some i , we can find a $\gamma > 0$ for which $\gamma e_i \in \tilde{\mathcal{C}}$ (since \mathcal{C} is bounded) and $c^T(\gamma e_i) = 0$; so γe_i must be optimal. Thus, we can assume without loss of generality that $c > 0$. We can also assume $p^* > 0$, otherwise we must have $x^* = 0$.

For $i = 1, \dots, n$, let $\beta_i = p^*/c_i$, so $c^T(\beta_i e_i) = p^*$. If $\beta_i e_i \in \tilde{\mathcal{C}}$ for some i , then $\beta_i e_i$ is also an optimal point, so we can assume $\beta_i e_i \notin \tilde{\mathcal{C}}$, for all $i = 1, \dots, n$. This implies $\beta_i e_i \in \mathbf{int} \mathcal{C}$ (the interior of \mathcal{C} in \mathbf{R}_+^n). We can write x^* as a convex combination of the points $\beta_i e_i$,

$$x^* = \sum_{i=1}^n \lambda_i (\beta_i e_i),$$

where $\lambda_i = x_i^* c_i / p^*$. (It is easy to check that $\lambda \succeq 0$ and $\mathbf{1}^T \lambda = 1$.) Since $\beta_i e_i$ are all interior points, their convex combination must also be in the interior of \mathcal{C} , but this implies $x^* \notin \tilde{\mathcal{C}}$, which contradicts the assumption that it is optimal.

3.23 *Jensen's inequality for posynomials.* Suppose $f : \mathbf{R}^n \rightarrow \mathbf{R}$ is a posynomial function, $x, y \in \mathbf{R}_{++}^n$, and $\theta \in [0, 1]$. Define $z \in \mathbf{R}_{++}^n$ by $z_i = x_i^\theta y_i^{1-\theta}$, $i = 1, \dots, n$. Show that $f(z) \leq f(x)^\theta f(y)^{1-\theta}$.

Interpretation. We can think of z as a θ -weighted geometric mean between x and y . So the statement above is that a posynomial, evaluated at a weighted geometric mean of two points, is no more than the weighted geometric mean of the posynomial evaluated at the two points.

Solution.

Let

$$f(x) = \sum_{k=1}^K c_k x_1^{\alpha_{1k}} x_2^{\alpha_{2k}} \cdots x_n^{\alpha_{nk}}.$$

We apply the standard transformation

$$\hat{x}_i = \log x_i, \quad \hat{y}_i = \log y_i, \quad \hat{z}_i = \log z_i, \quad i = 1, \dots, n,$$

and

$$\hat{c}_k = \log c_k.$$

The inequality we want to prove is equivalent to

$$\log f(z) \leq \theta \log f(x) + (1 - \theta) \log f(y).$$

Since

$$\hat{z}_i = \theta \hat{x}_i + (1 - \theta) \hat{y}_i, \quad i = 1, \dots, n,$$

the inequality is

$$\log \sum_{k=1}^K e^{\theta \alpha_k^T \hat{x}_i + (1-\theta) \alpha_k^T \hat{y}_i + \hat{c}_k} \leq \theta \log \sum_{k=1}^K e^{\alpha_k^T \hat{x} + \hat{c}_k} + (1-\theta) \log \sum_{k=1}^K e^{\alpha_k^T \hat{y} + \hat{c}_k}.$$

Let

$$\begin{aligned}\tilde{x}_k &= \alpha_k^T \hat{x} + \hat{c}_k, \\ \tilde{y}_k &= \alpha_k^T \hat{y} + \hat{c}_k,\end{aligned}$$

so we finally have

$$\log \sum_{k=1}^K e^{\theta \tilde{x}_k + (1-\theta) \tilde{y}_k} \leq \theta \log \sum_{k=1}^K e^{\tilde{x}_k} + (1-\theta) \log \sum_{k=1}^K e^{\tilde{y}_k}.$$

This holds because the function log-sum-exp is convex (Chap. 3, page 72).

3.24 CVX implementation of a concave function. Consider the concave function $f : \mathbf{R} \rightarrow \mathbf{R}$ defined by

$$f(x) = \begin{cases} (x+1)/2 & x > 1 \\ \sqrt{x} & 0 \leq x \leq 1, \end{cases}$$

with $\text{dom } f = \mathbf{R}_+$. Give a CVX implementation of f , via a partially specified optimization problem. Check your implementation by maximizing $f(x) + f(a-x)$ for several interesting values of a (say, $a = -1$, $a = 1$, and $a = 3$).

Solution. You can't construct f using simple disciplined convex programming (DCP) rules, so you'll need to represent it as the optimal value of a convex problem. This can be done several ways, using various built-in CVX functions (which are themselves written as the optimal value of a cone problem!). Here is one:

$$\begin{aligned}&\text{maximize} && \sqrt{u} + v/2 \\ &\text{subject to} && x = u + v, \quad u \leq 1, \quad v \geq 0,\end{aligned}$$

with variables u, v . (Note that the squareroot function implies that $u \geq 0$.) This implementation relies on the built-in function `sqrt`.

Of course, you have to argue that the optimal value of this problem is indeed $f(x)$. First of all, the domain is correct: if $x < 0$, the problem above is infeasible, so $f(x) = -\infty$. Now suppose $x \in [0, 1]$. We have $u = x - v$, so the derivative of the objective with respect to v is $-(1/2)(x-v)^{-1/2} + 1/2$. This is negative, so the optimal v is $v = 0$, and in this range of x , the optimal value of the problem is \sqrt{x} , as required. Now consider the case $x > 1$. Setting the derivative above to zero yields

$v = x - 1$, which implies $u = 1$. Plugging this into the objective, we find that the optimal value of the problem is $\sqrt{1} + (1/2)(x - 1) = (x + 1)/2$ as required.

There are many other possible choices, of course.

Here is the implementation:

```
function result = ccv_f(x)
cvx_begin
    variables u v;
    maximize(sqrt(u)+v/2);
    subject to
        x == u+v
        v >= 0
    u <= 1
cvx_end
result = cvx_optval;
```

And here is the test script:

```
for a=[-1,1,3]
    cvx_begin
        variable x
        maximize(ccv_f(x) + ccv_f(a-x));
    cvx_end
    a
    x
    cvx_optval
end
```

When it is run, it does everything the right way. For $a = -1$, the problem is infeasible; when $a = 1$, the optimal x is $1/2$, and optimal value is $\sqrt{2}$; when $a = 3$, $x = 3/2$ is optimal, and the optimal value is $5/2$.

3.25 The following optimization problem arises in portfolio optimization:

$$\begin{aligned} \text{maximize} \quad & \frac{r^T x + d}{\|Rx + q\|_2} \\ \text{subject to} \quad & \sum_{i=1}^n f_i(x_i) \leq b \\ & x \succeq c. \end{aligned}$$

The variable is $x \in \mathbf{R}^n$. The functions f_i are defined as

$$f_i(x) = \alpha_i x_i + \beta_i |x_i| + \gamma_i |x_i|^{3/2},$$

with $\beta_i > |\alpha_i|$, $\gamma_i > 0$. We assume there exists a feasible x with $r^T x + d > 0$.

Show that this problem can be solved by solving an SOCP (if possible) or a sequence of SOCP feasibility problems (otherwise).

Solution. Since there is a feasible x with $r^T x + d > 0$, we can equivalently solve

$$\begin{aligned} \text{minimize} \quad & \frac{\|Rx + q\|_2}{r^T x + d} \\ \text{subject to} \quad & \sum_{i=1}^n f_i(x_i) \leq b \\ & x \succeq c \end{aligned}$$

and take as the domain of the cost function $\{x \mid r^T x + d > 0\}$. The objective function is quasiconvex and can be minimized via bisection. An alternative solution is to make a change of variables

$$y = \frac{1}{r^T x + d} x, \quad t = \frac{1}{r^T x + d}$$

by replacing x with y/t and adding the constraint $r^T y + dt = 1$, $t \geq 0$. This gives the equivalent problem

$$\begin{aligned} \text{minimize} \quad & \|Ry + qt\|_2 \\ \text{subject to} \quad & \sum_{i=1}^n (\alpha_i y_i + \beta_i |y_i| + \gamma_i t (|y_i|/t)^{3/2}) \leq tb \\ & y \succeq tc \\ & r^T y + dt = 1 \\ & t \geq 0. \end{aligned}$$

(Note that the first constraint implies that $y = 0$ if $t = 0$, but this is impossible because of the third constraint. Therefore $t > 0$ at the optimum.) The problem is further equivalent to

$$\begin{aligned} \text{minimize} \quad & w \\ \text{subject to} \quad & \|Ry + qt\|_2 \leq w \\ & \sum_{i=1}^n (\alpha_i y_i + \beta_i u_i + \gamma_i z_i) \leq tb \\ & -u \preceq y \preceq u \\ & u_i^{3/2} \leq z_i t^{1/2}, \quad i = 1, \dots, n \\ & y \succeq tc \\ & r^T y + dt = 1 \\ & t \geq 0. \end{aligned}$$

It remains to express the constraints $u_i^{3/2} \leq z_i t^{1/2}$ as second order cone constraints. These constraints are equivalent to

$$u_i^2 \leq z_i v_i, \quad v_i^2 \leq u_i t, \quad z_i, v_i, u_i, t \geq 0$$

which in turn are equivalent to second-order cone constraints

$$\left\| \begin{bmatrix} 2u_i \\ z_i - v_i \end{bmatrix} \right\|_2 \leq z_i + v_i, \quad \left\| \begin{bmatrix} 2v_i \\ u_i - t_i \end{bmatrix} \right\|_2 \leq u_i + t_i, \quad z_i, v_i, u_i, t \geq 0.$$

3.26 Positive nonconvex QCQP. We consider a (possibly nonconvex) QCQP, with nonnegative variable $x \in \mathbf{R}^n$,

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

where $f_i(x) = (1/2)x^T P_i x + q_i^T x + r_i$, with $P_i \in \mathbf{S}^n$, $q_i \in \mathbf{R}^n$, and $r_i \in \mathbf{R}$, for $i = 0, \dots, m$. We do not assume that $P_i \succeq 0$, so this need not be a convex problem.

Suppose that $q_i \preceq 0$, and P_i have nonpositive off-diagonal entries, *i.e.*, they satisfy

$$(P_i)_{jk} \leq 0, \quad j \neq k, \quad j, k = 1, \dots, n,$$

for $i = 0, \dots, m$. (A matrix with nonpositive off-diagonal entries is called a *Z-matrix*.) Explain how to reformulate this problem as a convex problem.

Hint. Change variables using $y_j = \phi(x_j)$, for some suitable function ϕ .

Solution. Let $y_j = x_j^2$. Because $x_j \geq 0$, we can recover x_j as $x_j = y_j^{1/2}$. The quadratic functions f_i can be written in terms of y as

$$f_i(x) = \frac{1}{2} \sum_{j=1}^n (P_i)_{jj} y_j + \frac{1}{2} \sum_{j \neq k} (P_i)_{jk} (y_j y_k)^{1/2} + \sum_{j=1}^n (q_i)_j y_j^{1/2}.$$

Since $y_j^{1/2}$ and $(y_j y_k)^{1/2}$ (the geometric mean of y_j and y_k) are concave and $(P_i)_{jk} \leq 0$, $q_i \preceq 0$, this is convex in y . Thus the QCQP becomes a convex problem in y .

3.27 Affine policy. We consider a family of LPs, parametrized by the random variable u , which is uniformly distributed on $\mathcal{U} = [-1, 1]^p$,

$$\begin{aligned} & \text{minimize} && c^T x \\ & \text{subject to} && Ax \preceq b(u), \end{aligned}$$

where $x \in \mathbf{R}^n$, $A \in \mathbf{R}^{m \times n}$, and $b(u) = b_0 + Bu \in \mathbf{R}^m$ is an affine function of u . You can think of u_i as representing a deviation of the i th parameter from its nominal value. The parameters might represent (deviations in) levels of resources available, or other varying limits.

The problem is to be solved many times; in each time, the value of u (*i.e.*, a sample) is given, and then the decision variable x is chosen. The mapping from u into the decision variable $x(u)$ is called the *policy*, since it gives the decision variable value for each value of u . When enough time and computing hardware is available, we can simply solve the LP for each new value of u ; this is an optimal policy, which we denote $x^*(u)$.

In some applications, however, the decision $x(u)$ must be made very quickly, so solving the LP is not an option. Instead we seek a suboptimal policy, which is affine: $x^{\text{aff}}(u) = x_0 + Ku$, where x_0 is called the *nominal decision* and $K \in \mathbf{R}^{n \times p}$ is called the *feedback gain matrix*. (Roughly speaking, x_0 is our guess of x before the value of u has been revealed; Ku is our modification of this guess, once we know u .) We determine the policy (*i.e.*, suitable values for x_0 and K) ahead of time; we can then evaluate the policy (that is, find $x^{\text{aff}}(u)$ given u) very quickly, by matrix multiplication and addition.

We will choose x_0 and K in order to minimize the expected value of the objective, while insisting that for any value of u , feasibility is maintained:

$$\begin{aligned} & \text{minimize} && \mathbf{E} c^T x^{\text{aff}}(u) \\ & \text{subject to} && Ax^{\text{aff}}(u) \preceq b(u) \quad \forall u \in \mathcal{U}. \end{aligned}$$

The variables here are x_0 and K . The expectation in the objective is over u , and the constraint requires that $Ax^{\text{aff}}(u) \preceq b(u)$ hold almost surely.

- (a) Explain how to find optimal values of x_0 and K by solving a standard explicit convex optimization problem (*i.e.*, one that does not involve an expectation or an infinite number of constraints, as the one above does.) The numbers of variables or constraints in your formulation should not grow exponentially with the problem dimensions n , p , or m .
- (b) Carry out your method on the data given in `affine_pol_data.m`. To evaluate your affine policy, generate 100 independent samples of u , and for each value, compute the objective value of the affine policy, $c^T x^{\text{aff}}(u)$, and of the optimal policy, $c^T x^*(u)$. Scatter plot the objective value of the affine policy (y -axis) versus the objective value of the optimal policy (x -axis), and include the line $y = x$ on the plot. Report the average values of $c^T x^{\text{aff}}(u)$ and $c^T x^*(u)$ over your samples. (These are estimates of $\mathbf{E} c^T x^{\text{aff}}(u)$ and $\mathbf{E} c^T x^*(u)$. The first number, by the way, can be found exactly.)

Solution. Let's start with the objective. We compute the expected value of the affine policy as

$$\mathbf{E} c^T (x_0 + Ku) = c^T x_0 + (K^T c)^T \mathbf{E} u = c^T x_0.$$

Now let's look at the constraints, which we write out in terms of its entries:

$$(Ax_0)_i + \sup_{u \in \mathcal{U}} ((AK - B)u)_i \leq (b_0)_i, \quad i = 1, \dots, m.$$

This turns into the explicit constraint

$$(Ax_0)_i + \|(AK - B)_i\|_1 \leq (b_0)_i, \quad i = 1, \dots, m.$$

Here $(AK - B)_i$ is the i th row of the matrix $A - BK$.

So we can find an optimal affine policy by solving the problem (which can be transformed to an LP)

$$\begin{aligned} & \text{minimize} && c^T x_0 \\ & \text{subject to} && (Ax_0)_i + \|(AK - B)_i\|_1 \leq (b_0)_i, \quad i = 1, \dots, m, \end{aligned}$$

with variables x_0 and K .

The code below implements this.

```
% Affine policy.
affine_pol_data;

% compute affine policy
cvx_begin
```

```

variables x0(n) K(n,p)
minimize (c'*x0)
subject to
    A*x0+norms(A*K-B,1,2) <= b0
cvx_end

% compare 100 samples
aff_obj = []; opt_obj = [];
for i=1:100
    u = 2*rand(p,1)-1;

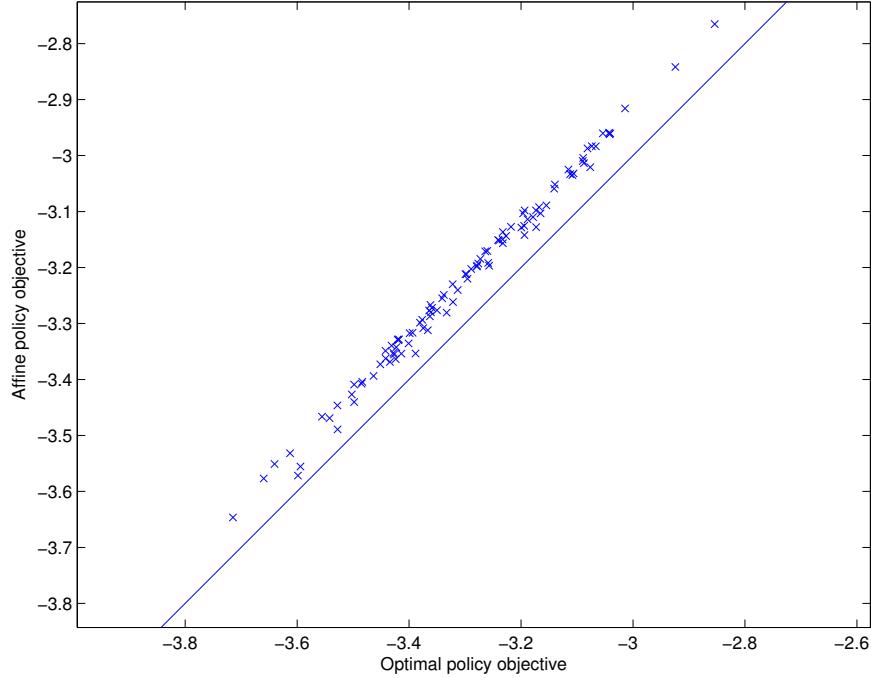
    cvx_begin quiet
        variable x_opt(n)
        minimize (c'*x_opt)
        subject to
            A*x_opt <= b0+B*u
    cvx_end

    aff_obj(i) = c'*(x0+K*u);
    opt_obj(i) = cvx_optval;
end

figure;
plot(opt_obj,aff_obj,'x'); hold on;
axis equal
plot(xlim, xlim);
xlabel('Optimal policy objective')
ylabel('Affine policy objective')

print -depsc affine_pol.eps

```



The (exact) expected cost of the affine policy is -3.219 , the mean objective of the affine policy is -3.223 for the sample, and the mean objective of the optimal policy is -3.301 for the sample. The plot shows that the affine policy produces points that are suboptimal, but not by much.

3.28 Probability bounds. Consider random variables X_1, X_2, X_3, X_4 that take values in $\{0, 1\}$. We are given the following marginal and conditional probabilities:

$$\begin{aligned}\mathbf{prob}(X_1 = 1) &= 0.9, \\ \mathbf{prob}(X_2 = 1) &= 0.9, \\ \mathbf{prob}(X_3 = 1) &= 0.1, \\ \mathbf{prob}(X_1 = 1, X_4 = 0 \mid X_3 = 1) &= 0.7, \\ \mathbf{prob}(X_4 = 1 \mid X_2 = 1, X_3 = 0) &= 0.6.\end{aligned}$$

Explain how to find the minimum and maximum possible values of $\mathbf{prob}(X_4 = 1)$, over all (joint) probability distributions consistent with the given data. Find these values and report them.

Hints. (You should feel free to ignore these hints.)

- Matlab:

- CVX supports multidimensional arrays; for example, `variable p(2,2,2,2)` declares a 4-dimensional array of variables, with each of the four indices taking the values 1 or 2.
- The function `sum(p,i)` sums a multidimensional array `p` along the `i`th index.
- The expression `sum(a(:))` gives the sum of all entries of a multidimensional array `a`. You might want to use the function definition `sum_all = @(A) sum(A(:));`, so `sum_all(a)` gives the sum of all entries in the multidimensional array `a`.

- Python:
 - Create a 1-d Variable and manually index the entries. You should come up with a reasonable scheme to avoid confusion.
- Julia:
 - You can create a multidimensional array of variables in Convex.jl. For example, the following creates a 4-dimensional array of variables, with each of the four indices taking the values 1 or 2.

```
p = [Variable() for i in 1:16];
p = reshape(p, 2, 2, 2, 2)
```

 - You can use the function `sum` to sum over various indices in the multidimesional array.

```
sum(p[:, :, :, :]) # sum all entries
sum(p[1, :, 2, :]) # fix first and third indices
```

 - To create constraints with the variables in the array, you need to access each variable independently. Something like `p >= 0` will not work.

Solution. The outcome space for the random variables has $2^4 = 16$ elements, corresponding to all possible 0/1 assignments to the variables. The variable is a probability disitribution on this set, which we can represent as a vector $p \in \mathbf{R}^{16}$. There are several reasonable choices for how we index the vector. One is to index the vector p by $i = 1, \dots, 16$, with some encoding of the index, such as

$$i = 1 + X_1 + 2X_2 + 4X_3 + 8X_4.$$

Another is to encode the index as 4-tuple p_{ijkl} , where $i, j, k, l \in \{1, 2\}$. (These are for Matlab encodings; in a real language, we'd start the indices at 0, or represent the probability distribution as a dictionary, *i.e.*, a set of key-value pairs.)

Now for any event $A \subseteq \{0, 1\}^4$, $\text{prob}(A)$ is a linear function of p . Indeed, the coefficient of each index is 0 if the index is not in A and 1 if the index is in A . So for any A , $\text{prob}(A)$ is a linear function of p .

It follows that the first three constraints, on marginal probabilities, are linear equalities on p : we simply sum the entries of p corresponding to the event (such as $X_1 = 1$) and constrain the sum to equal the given righthand side value. Conditional probabilities are ratios of probabilities of events, and so are linear-fractional functions of p . We can write the constraint

$$\text{prob}(A | B) = \frac{\text{prob}(A \cap B)}{\text{prob}(B)} = \alpha$$

as

$$\text{prob}(A \cap B) = \alpha \text{prob}(B),$$

which is a linear equality constraint on p . Since $\text{prob}(X_4 = 1)$ is a linear function of p , maximizing and minimizing it subject to the constraints is just an LP.

We find that the range of possible values of $\text{prob}(X_4 = 1)$ is between 0.48 and 0.61.

The following code solves the problem in Matlab, using the two different encodings described above.

```

sum_all = @(A) sum( A(:));

for direction=[-1, 1]
    % 1 in the ith entry of p corresponds to X_i being 0.
    % 2 corresponds to X_i being 1.
    cvx_begin
        variable p(2,2,2,2);
        maximize( direction*sum_all(p(:,:, :,2)))

        sum_all(p)==1;
        p >= 0;

        sum_all( p(2,:,:,:)) == .9;
        sum_all( p(:,2,:,:)) == .9;
        sum_all( p(:,:,2,:)) == .1;

        sum_all( p(2,:,2,1)) == .7* sum_all(p(:,:,2,:));
        sum_all( p(:,2,1,2)) == .6*sum_all(p(:,2,1,:));
    cvx_end
    sum_all(p(:,:, :,2))
end

for direction=[-1, 1]
    % p(k) corresponds to the arrangement x_1 x_2 x_3 x_4
    % where k = x_1 + 2 x_2 + 4 x_3+ 8 x_4 + +1
    cvx_begin
        variable p(16);
        maximize( direction*sum(p([9,10,11,12,13,14,15,16])))

        sum(p)==1;
        p >= 0;

        sum( p([1,3,5,7,9,11,13,15]+1)) == .9;
        sum( p([2,3,6,7,10,11,14,15]+1) ) == .9;
        sum( p([4,5,6,7,12,13,14,15]+1)) == .1;

        sum( p([5,7]+1)) == .7* sum(p([4,5,6,7,12,13,14,15]+1));
        sum( p([10,11]+1)) == .6*sum(p([2,3,10,11]+1));
    cvx_end
    sum(p([9,10,11,12,13,14,15,16]))
end

```

The following code solves the problem in Python.

```

import cvxpy as cvx

for direction in [-1, 1]:
    # 0 in the ith entry of p corresponds to X_i being 0.
    # 1 corresponds to X_i being 1.
    p = cvx.Variable(2, 2, 2, 2)
    obj = direction*cvx.sum_entries(p[:, :, :, 1])
    cons = [cvx.sum_entries(p) == 1, p >= 0]
    cons += [cvx.sum_entries(p[1, :, :, :]) == .9]
    cons += [cvx.sum_entries(p[:, 1, :, :]) == .9]
    cons += [cvx.sum_entries(p[:, :, 1, :]) == .1]

    cons += [cvx.sum_entries(p[1, :, 1, 0]) == .7 * cvx.sum_entries(p[:, :, 1, :])]
    cons += [cvx.sum_entries(p[:, 1, 0, 1]) == .6 * cvx.sum_entries(p[:, 1, 0, :])]

    cvx.Problem(cvx.Maximize(obj), cons).solve()
    print cvx.sum_entries(p[:, :, :, 1]).value

for direction in [-1, 1]:
    # p(k) corresponds to the arrangement x_1 x_2 x_3 x_4
    # where k = x_1 + 2 x_2 + 4 x_3+ 8 x_4
    p = cvx.Variable(16)
    obj = direction*cvx.sum_entries(p[8:])
    cons = [cvx.sum_entries(p) == 1, p >= 0]
    cons += [cvx.sum_entries(p[1::2]) == .9]
    cons += [cvx.sum_entries(p[2::4]) + cvx.sum_entries(p[3::4]) == .9]
    cons += [cvx.sum_entries(p[4:8]) + cvx.sum_entries(p[12:16]) == .1]

    cons += [p[5]+p[7] == .7 * (cvx.sum_entries(p[4:8]) + cvx.sum_entries(p[12:16]))]
    cons += [p[10]+p[11] == .6 * (p[2]+p[3]+p[10]+p[11])]

    cvx.Problem(cvx.Maximize(obj), cons).solve()
    print cvx.sum_entries(p[8:]).value

```

A solution in Julia follows.

```

using Convex

p = [Variable() for i in 1:16];
p = reshape(p, 2, 2, 2, 2);

constraints = [];
for i in 1:16
    constraints += p[i] >= 0;
end
constraints += sum(p) == 1;

```

```

constraints += sum(p[2,:,:,:]) == 0.9;
constraints += sum(p[:,2,:,:]) == 0.9;
constraints += sum(p[:, :, 2, :]) == 0.1;
constraints += sum(p[2, :, 2, 1]) == 0.7 * sum(p[:, :, 2, :]);
constraints += sum(p[:, 2, 1, 2]) == 0.6 * sum(p[:, 2, 1, :]);

objective = sum(p[:, :, :, 2]);

maxp = maximize(objective, constraints);
solve!(maxp);
println(maxp.optval);
minp = minimize(objective, constraints);
solve!(minp);
println(minp.optval);

```

- 3.29 Robust quadratic programming.** In this problem, we consider a robust variation of the (convex) quadratic program

$$\begin{aligned} & \text{minimize} && (1/2)x^T Px + q^T x + r \\ & \text{subject to} && Ax \preceq b. \end{aligned}$$

For simplicity we assume that only the matrix P is subject to errors, and the other parameters (q , r , A , b) are exactly known. The robust quadratic program is defined as

$$\begin{aligned} & \text{minimize} && \sup_{P \in \mathcal{E}} ((1/2)x^T Px + q^T x + r) \\ & \text{subject to} && Ax \preceq b \end{aligned}$$

where \mathcal{E} is the set of possible matrices P .

For each of the following sets \mathcal{E} , express the robust QP as a *tractable* convex problem. Be as specific as you can. (Here, tractable means that the problem can be reduced to an LP, QP, QCQP, SOCP, or SDP. But you do not have to work out the reduction, if it is complicated; it is enough to argue that it can be reduced to one of these.)

- (a) A finite set of matrices: $\mathcal{E} = \{P_1, \dots, P_K\}$, where $P_i \in \mathbf{S}_+^n$, $i = 1, \dots, K$.
- (b) A set specified by a nominal value $P_0 \in \mathbf{S}_+^n$ plus a bound on the eigenvalues of the deviation $P - P_0$:

$$\mathcal{E} = \{P \in \mathbf{S}^n \mid -\gamma I \preceq P - P_0 \preceq \gamma I\}$$

where $\gamma \in \mathbf{R}$ and $P_0 \in \mathbf{S}_+^n$.

- (c) An ellipsoid of matrices:

$$\mathcal{E} = \left\{ P_0 + \sum_{i=1}^K P_i u_i \mid \|u\|_2 \leq 1 \right\}.$$

You can assume $P_i \in \mathbf{S}_+^n$, $i = 0, \dots, K$.

Solution.

- (a) The objective function is a (finite) maximum of convex functions, hence convex. The formulation

$$\begin{aligned} & \text{minimize} && \max_{i=1,\dots,K} \left((1/2)x^T P_i x + q^T x + r \right) \\ & \text{subject to} && Ax \preceq b, \end{aligned}$$

with variable x , is tractable.

It can be expressed as the convex QCQP

$$\begin{aligned} & \text{minimize} && t \\ & \text{subject to} && (1/2)x^T P_i x + q^T x + r \leq t, \quad i = 1, \dots, K \\ & && Ax \preceq b, \end{aligned}$$

with variables x and t .

- (b) For given x , the supremum of $x^T \Delta P x$ over $-\gamma I \preceq \Delta P \preceq \gamma I$ is given by

$$\sup_{-\gamma I \preceq \Delta P \preceq \gamma I} x^T \Delta P x = \gamma x^T x.$$

Therefore we can express the robust QP as

$$\begin{aligned} & \text{minimize} && (1/2)x^T (P_0 + \gamma I)x + q^T x + r \\ & \text{subject to} && Ax \preceq b \end{aligned}$$

which is a QP.

- (c) For given x , the quadratic objective function is

$$\begin{aligned} & (1/2) \left(x^T P_0 x + \sup_{\|u\|_2 \leq 1} \sum_{i=1}^K u_i (x^T P_i x) \right) + q^T x + r \\ & = (1/2)x^T P_0 x + (1/2) \left(\sum_{i=1}^K (x^T P_i x)^2 \right)^{1/2} + q^T x + r. \end{aligned}$$

This is a convex function of x . To see this, observe that each of the functions $x^T P_i x$ is convex since $P_i \succeq 0$. The second term is a composition $h(g_1(x), \dots, g_K(x))$ of $h(y) = \|y\|_2$ with $g_i(x) = x^T P_i x$. The functions g_i are convex and nonnegative. The function h is convex and, for $y \in \mathbf{R}_+^K$, nondecreasing in each of its arguments. Therefore the composition is convex.

The resulting problem can be expressed as

$$\begin{aligned} & \text{minimize} && (1/2)x^T P_0 x + \|y\|_2 + q^T x + r \\ & \text{subject to} && (1/2)x^T P_i x \leq y_i, \quad i = 1, \dots, K \\ & && Ax \preceq b. \end{aligned}$$

- 3.30 Smallest confidence ellipsoid.** Suppose the random variable X on \mathbf{R}^n has log-concave density p . Formulate the following problem as a convex optimization problem: Find an ellipsoid \mathcal{E} that satisfies $\text{prob}(X \in \mathcal{E}) \geq 0.95$ and is smallest, in the sense of minimizing the sum of the squares of its semi-axis lengths. You do not need to worry about how to solve the resulting convex optimization problem; it is enough to formulate the smallest confidence ellipsoid problem as the problem of minimizing a convex function over a convex set involving the parameters that define \mathcal{E} .

Solution. We parametrize the ellipsoid as $\mathcal{E}(c, P) = \{x \mid (x - c)^T P^{-1}(x - c) \leq 1\}$, where $P \succ 0$. The semi-axis lengths are $\lambda_i^{1/2}$, where λ_i are the eigenvalues of P . The sum of the squares of the semi-axis lengths is $\sum_i \lambda_i = \text{tr } P$. So our job is to minimize $\text{tr } P$ subject to $\text{prob}(X \in \mathcal{E}) \geq 0.95$.

Define $g(x, c, P)$ as the 0-1 indicator function of \mathcal{E} , *i.e.*,

$$g(x, c, P) = \begin{cases} 1 & (x - c)^T P^{-1}(x - c) \leq 1 \\ 0 & \text{otherwise.} \end{cases}$$

Since $(x - c)^T P^{-1}(x - c)$ is a convex function of (x, c, P) (for $P \succ 0$), $\log g$ is concave in (x, c, P) , so g is log-concave in (x, c, P) . By the integration rule for log-concave functions,

$$\int p(x)g(x, c, P) dx = \text{prob}(X \in \mathcal{E}(c, P))$$

is log-concave in (c, P) . So for any η , $\text{prob}(X \in \mathcal{E}(c, P)) \geq \eta$ is a convex constraint in (c, P) . In particular,

$$\{(c, P) \mid \text{prob}(X \in \mathcal{E}(c, P)) \geq 0.95\}$$

is a convex set. We simply minimize (the linear function) $\text{tr } P$ over this set to get the smallest confidence ellipsoid.

- 3.31 Stochastic optimization via Monte Carlo sampling.** In (convex) stochastic optimization, the goal is to minimize a cost function of the form $F(x) = \mathbf{E} f(x, \omega)$, where ω is a random variable on Ω , and $f : \mathbf{R}^n \times \Omega \rightarrow \mathbf{R}$ is convex in its first argument for each $\omega \in \Omega$. (For simplicity we consider the unconstrained problem; it is not hard to include constraints.) Evidently F is convex. Let p^* denote the optimal value, *i.e.*, $p^* = \inf_x F(x)$ (which we assume is finite).

In a few very simple cases we can work out what F is analytically, but in general this is not possible. Moreover in many applications, we do not know the distribution of ω ; we only have access to an oracle that can generate independent samples from the distribution.

A standard method for approximately solving the stochastic optimization problem is based on Monte Carlo sampling. We first generate N independent samples, $\omega_1, \dots, \omega_N$, and form the empirical expectation

$$\hat{F}(x) = \frac{1}{N} \sum_{i=1}^N f(x, \omega_i).$$

This is a random function, since it depends on the particular samples drawn. For each x , we have $\mathbf{E} \hat{F}(x) = F(x)$, and also $\mathbf{E}(\hat{F}(x) - F(x))^2 \propto 1/N$. Roughly speaking, for N large enough, $\hat{F}(x) \approx F(x)$.

To (approximately) minimize F , we instead minimize $\hat{F}(x)$. The minimizer, \hat{x}^* , and the optimal value $\hat{p}^* = \hat{F}(\hat{x}^*)$, are also random variables. The hope is that for N large enough, we have $\hat{p}^* \approx p^*$. (In practice, stochastic optimization via Monte Carlo sampling works very well, even when N is not that big.)

One way to check the result of Monte Carlo sampling is to carry it out multiple times. We repeatedly generate different batches of samples, and for each batch, we find \hat{x}^* and \hat{p}^* . If the values of \hat{p}^* are near each other, it's reasonable to believe that we have (approximately) minimized F . If they are not, it means our value of N is too small.

Show that $\mathbf{E} \hat{p}^* \leq p^*$.

This inequality implies that if we repeatedly use Monte Carlo sampling and the values of \hat{p}^* that we get are all very close, then they are (likely) close to p^* .

Hint. Show that for any function $G : \mathbf{R}^n \times \Omega \rightarrow \mathbf{R}$ (convex or not in its first argument), and any random variable ω on Ω , we have

$$\inf_x \mathbf{E} G(x, \omega) \geq \mathbf{E} \inf_x G(x, \omega).$$

Solution. Let's show the hint. For each ω and any $z \in \mathbf{R}^n$, we have $G(z, \omega) \geq \inf_x G(x, \omega)$. Since expectation is monotone, we can take expectation and get

$$\mathbf{E} G(z, \omega) \geq \mathbf{E} \inf_x G(x, \omega).$$

This holds for all $z \in \mathbf{R}^n$, so we conclude that

$$\inf_z \mathbf{E} G(z, \omega) \geq \mathbf{E} \inf_x G(x, \omega),$$

which is what we wanted to show.

Then we have

$$\begin{aligned} p^* &= \inf_x F(x) \\ &= \inf_x \mathbf{E} \hat{F}(x) \\ &= \inf_x \mathbf{E} \frac{1}{N} \sum_{i=1}^N f(x, \omega_i) \\ &\geq \mathbf{E} \inf_x \frac{1}{N} \sum_{i=1}^N f(x, \omega_i) \\ &= \mathbf{E} \hat{p}^*. \end{aligned}$$

3.32 Satisfying a minimum number of constraints.

Consider the problem

$$\begin{aligned} &\text{minimize} && f_0(x) \\ &\text{subject to} && f_i(x) \leq 0 \text{ holds for at least } k \text{ values of } i, \end{aligned}$$

with variable $x \in \mathbf{R}^n$, where the objective f_0 and the constraint functions f_i , $i = 1, \dots, m$ (with $m \geq k$), are convex. Here we require that only k of the constraints hold, instead of all m of them. In general this is a hard combinatorial problem; the brute force solution is to solve all $\binom{m}{k}$ convex problems obtained by choosing subsets of k constraints to impose, and selecting one with smallest objective value.

In this problem we explore a convex restriction that can be an effective heuristic for the problem.

- (a) Suppose $\lambda > 0$. Show that the constraint

$$\sum_{i=1}^m (1 + \lambda f_i(x))_+ \leq m - k$$

guarantees that $f_i(x) \leq 0$ holds for at least k values of i . ($(u)_+$ means $\max\{u, 0\}$.)

Hint. For each $u \in \mathbf{R}$, $(1 + \lambda u)_+ \geq 1(u > 0)$, where $1(u > 0) = 1$ for $u > 0$, and $1(u > 0) = 0$ for $u \leq 0$.

(b) Consider the problem

$$\begin{aligned} & \text{minimize} && f_0(x) \\ & \text{subject to} && \sum_{i=1}^m (1 + \lambda f_i(x))_+ \leq m - k \\ & && \lambda > 0, \end{aligned}$$

with variables x and λ . This is a restriction of the original problem: If (x, λ) are feasible for it, then x is feasible for the original problem. Show how to solve this problem using convex optimization. (This may involve a change of variables.)

(c) Apply the method of part (b) to the problem instance

$$\begin{aligned} & \text{minimize} && c^T x \\ & \text{subject to} && a_i^T x \leq b_i \text{ holds for at least } k \text{ values of } i, \end{aligned}$$

with $m = 70$, $k = 58$, and $n = 12$. The vectors b , c and the matrix A with rows a_i^T are given in the file `satisfy_some_constraints_data.*`.

Report the optimal value of λ , the objective value, and the actual number of constraints that are satisfied (which should be larger than or equal to k). To determine if a constraint is satisfied, you can use the tolerance $a_i^T x - b_i \leq \epsilon^{\text{feas}}$, with $\epsilon^{\text{feas}} = 10^{-5}$.

A standard trick is to take this tentative solution, choose the k constraints with the smallest values of $f_i(x)$, and then minimize $f_0(x)$ subject to these k constraints (*i.e.*, ignoring the other $m - k$ constraints). This improves the objective value over the one found using the restriction. Carry this out for the problem instance, and report the objective value obtained.

Solution.

(a) We first prove the hint. If $u > 0$ then $1(u > 0) = 1$ and $1 + \lambda u > 1$, so $(1 + \lambda u)_+ > 1$. If $u \leq 0$ then $1(u > 0) = 0$ and $(1 + \lambda u)_+ \geq 0$. Hence $(1 + \lambda u)_+ \geq 1(u > 0)$ for all $u \in \mathbf{R}$.

Applying this to $u = f_i(x)$, we have that $(1 + \lambda f_i(x))_+ \geq 1(f_i(x) > 0)$ for all i . Hence

$$\sum_{i=1}^m 1(f_i(x) > 0) \leq \sum_{i=1}^m (1 + \lambda f_i(x))_+.$$

The constraint $\sum_{i=1}^m (1 + \lambda f_i(x))_+ \leq m - k$ guarantees that $\sum_{i=1}^m 1(f_i(x) > 0) \leq m - k$, so $f_i(x) > 0$ holds for at most $m - k$ values of i . In other words, $f_i(x) \leq 0$ holds for at least k values of i .

(b) If $\lambda > 0$ then $(\lambda u)_+ = \lambda(u)_+$ for all $u \in \mathbf{R}$. Hence $(1 + \lambda f_i(x))_+ = \lambda(1/\lambda + f_i(x))_+$. The constraint $\sum_{i=1}^m (1 + \lambda f_i(x))_+ \leq m - k$ can then be written as

$$\sum_{i=1}^m \lambda \left(\frac{1}{\lambda} + f_i(x) \right)_+ \leq m - k,$$

or equivalently,

$$\sum_{i=1}^m \left(\frac{1}{\lambda} + f_i(x) \right)_+ \leq (m - k) \frac{1}{\lambda}.$$

Letting $\mu = 1/\lambda$, the restricted problem can be expressed as

$$\begin{aligned} & \text{minimize} && f_0(x) \\ & \text{subject to} && \sum_{i=1}^m (\mu + f_i(x))_+ \leq (m-k)\mu \\ & && \mu > 0, \end{aligned}$$

with variables x and μ . Since the function $(\cdot)_+$ is convex and nondecreasing, and $\mu + f_i(x)$ is convex in both μ and x , $(\mu + f_i(x))_+$ is convex, so this is a convex optimization problem.

After solving this problem and obtaining an optimal value μ^* for μ , the optimal value of λ is $1/\mu^*$. Note that we can replace the constraint $\mu > 0$ by $\mu \geq 0$, since if $\mu = 0$ then the constraint $\sum_{i=1}^m (\mu + f_i(x))_+ \leq (m-k)\mu$ is the same as all the constraints $f_i(x) \leq 0$ hold. Certainly in this case at least k of them hold. Alternatively we can introduce a new variable t and replace the constraint $\mu > 0$ with $\mu \geq e^t$.

- (c) The optimal value of λ is 282.98 and the objective value is -8.45 . The number of constraints satisfied is 66, which exceeds our required minimum, $k = 58$.

When we take this tentative solution, choose the k constraints with the smallest values of $f_i(x)$, and then minimize $f_0(x)$ subject to these k constraints, we get an objective value between -8.75 and -8.86 (depending on the solver; these numbers *should* be the same ...). In any case, it gives a modest improvement in objective compared to the restriction.

The actual optimal value (which we obtained using branch and bound, a global optimization method) is -9.57 . To compute this takes far more effort than to solve the restriction; for larger problem sizes solving the global problem is prohibitively slow.

The following Matlab code solves the problem:

```
satisfy_some_constraints_data;

cvx_begin quiet
    variables x(n) mu_var
    minimize(c' * x)
    subject to
        sum(pos(mu_var + A * x - b)) <= (m - k) * mu_var
        mu_var >= 0
cvx_end
fprintf('Optimal value of lambda: %f\n', 1 / mu_var)
fprintf('Objective value: %f\n', cvx_optval)

fprintf('Number of constraints satisfied: %d\n', nnz(A * x - b <= 1e-5))

% Choose k least violated inequalities as constraints
[~, idx] = sort(A * x - b);
least_violated = idx(1:k);
cvx_begin quiet
    variables x(n)
    minimize(c' * x)
    subject to
        A(least_violated, :) * x <= b(least_violated)
```

```

cvx_end
fprintf('Objective after minimizing wrt k constraints: %f\n', cvx_optval)

```

The following Python code solves the problem:

```

import cvxpy as cvx

from satisfy_some_constraints_data import *

x = cvx.Variable(n)
mu = cvx.Variable()
constraints = [cvx.sum_entries(cvx.pos(mu + A * x - b)) <= (m - k) * mu,
               mu >= 0]
problem = cvx.Problem(cvx.Minimize(c.T * x), constraints)
problem.solve()
print('Optimal value of lambda: {}'.format(1 / mu.value))
print('Objective value: {}'.format(problem.value))

print('Number of constraints satisfied: {}'
      .format(np.count_nonzero(A.dot(x.value.A1) - b <= 1e-5)))

# Choose k least violated inequalities as constraints
least_violated = np.argsort(A.dot(x.value).A1 - b)[:k]
constraints = [A[least_violated] * x <= b[least_violated]]
problem = cvx.Problem(cvx.Minimize(c.T * x), constraints)
problem.solve()
print('Objective after minimizing wrt k constraints: {}'
      .format(problem.value))

```

The following Julia code solves the problem:

```

using Convex, SCS
set_default_solver(SCSSolver(verbose=false))

include("satisfy_some_constraints_data.jl")

x = Variable(n)
mu = Variable()
constraints = [sum(pos(mu + A * x - b)) <= (m - k) * mu,
               mu >= 0]
problem = minimize(c' * x, constraints)
solve!(problem)
println("Optimal value of lambda: $(1 / mu.value)")
println("Objective value: $(problem.optval)")

num_constraints_satisfied = countnz(A * x.value - b .<= 1e-5)
println("Number of constraints satisfied: $num_constraints_satisfied")

```

```
# Choose k least violated inequalities as constraints
least_violated = sortperm((A * x.value - b)[:,])[1:k]
constraints = [A[least_violated, :] * x <= b[least_violated]]
problem = minimize(c' * x, constraints)
solve!(problem)
println("Objective after minimizing wrt k constraints: $(problem.optval)")
```

4 Duality

4.1 Numerical perturbation analysis example. Consider the quadratic program

$$\begin{aligned} & \text{minimize} && x_1^2 + 2x_2^2 - x_1x_2 - x_1 \\ & \text{subject to} && x_1 + 2x_2 \leq u_1 \\ & && x_1 - 4x_2 \leq u_2, \\ & && 5x_1 + 76x_2 \leq 1, \end{aligned}$$

with variables x_1 , x_2 , and parameters u_1 , u_2 .

- (a) Solve this QP, for parameter values $u_1 = -2$, $u_2 = -3$, to find optimal primal variable values x_1^* and x_2^* , and optimal dual variable values λ_1^* , λ_2^* and λ_3^* . Let p^* denote the optimal objective value. Verify that the KKT conditions hold for the optimal primal and dual variables you found (within reasonable numerical accuracy).

Matlab hint: See §3.7 of the CVX users' guide to find out how to retrieve optimal dual variables. To specify the quadratic objective, use `quad_form()`.

- (b) We will now solve some perturbed versions of the QP, with

$$u_1 = -2 + \delta_1, \quad u_2 = -3 + \delta_2,$$

where δ_1 and δ_2 each take values from $\{-0.1, 0, 0.1\}$. (There are a total of nine such combinations, including the original problem with $\delta_1 = \delta_2 = 0$.) For each combination of δ_1 and δ_2 , make a prediction p_{pred}^* of the optimal value of the perturbed QP, and compare it to p_{exact}^* , the exact optimal value of the perturbed QP (obtained by solving the perturbed QP). Put your results in the two righthand columns in a table with the form shown below. Check that the inequality $p_{\text{pred}}^* \leq p_{\text{exact}}^*$ holds.

δ_1	δ_2	p_{pred}^*	p_{exact}^*
0	0		
	-0.1		
	0.1		
-0.1	0		
	-0.1		
	0.1		
0.1	0		
	-0.1		
	0.1		

Matlab solution

The following Matlab code sets up the simple QP and solves it using CVX.

Part (a):

```
Q = [1 -1/2; -1/2 2];
f = [-1 0]';
A = [1 2; 1 -4; 5 76];
```

```

b = [-2 -3 1]';

cvx_begin
    variable x(2)
    dual variable lambda
    minimize(quad_form(x,Q)+f'*x)
    subject to
        lambda: A*x <= b
cvx_end
p_star = cvx_optval

```

Part (b):

```

arr_i = [0 -1 1];
delta = 0.1;
pa_table = [];
for i = arr_i
    for j = arr_i
        p_pred = p_star - [lambda(1) lambda(2)]*[i; j]*delta;
        cvx_begin
            variable x(2)
            minimize(quad_form(x,Q)+f'*x)
            subject to
                A*x <= b+[i;j;0]*delta
        cvx_end
        p_exact = cvx_optval;

        pa_table = [pa_table; i*delta j*delta p_pred p_exact]
    end
end

```

When we run this, we find the optimal objective value is $p^* = 8.22$ and the optimal point is $x_1^* = -2.33$, $x_2^* = 0.17$. (This optimal point is unique since the objective is strictly convex.) A set of optimal dual variables is $\lambda_1^* = 2.13$, $\lambda_2^* = 3.31$ and $\lambda_3^* = 0.08$.

Python solution

The following Python code sets up the simple QP and solves it using CVXPY.

Part (a) and (b):

```

import cvxpy as cvx
import numpy as np

# part (a)
Q = np.matrix('1 -0.5; -0.5 2')
f = np.matrix('-1; 0')

```

```

A = np.matrix('1 2; 1 -4; 5 76')
b = np.matrix('-2; -3; 1')

x = cvx.Variable(2)
obj = cvx.quad_form(x, Q) + x.T*f
cons = [A*x <= b]

p_star = cvx.Problem(cvx.Minimize(obj), cons).solve()
lambdas = cons[0].dual_value

print p_star
print x.value
print lambdas
print A*x.value - b
print 2*Q*x.value + f + A.transpose()*cons[0].dual_value

# part(b)
arr_i = np.array([0, -1, 1])
delta = 0.1
pa_table = np.zeros((9, 4))
count = 0
for i in arr_i:
    for j in arr_i:
        p_pred = p_star - (lambdas[0]*i + lambdas[1]*j)*delta
        cons = [A*x <= b+np.matrix(np.array([i,j,0])).T*delta]
        p_exact = cvx.Problem(cvx.Minimize(obj), cons).solve()
        pa_table[count,:] = np.array([i*delta, j*delta, p_pred, p_exact])
        count += 1

print pa_table

```

When we run this, we find the optimal objective value is $p^* = 8.22$ and the optimal point is $x_1^* = -2.33$, $x_2^* = 0.17$. (This optimal point is unique since the objective is strictly convex.) A set of optimal dual variables is $\lambda_1^* = 1.60$, $\lambda_2^* = 3.67$ and $\lambda_3^* = 0.11$.

Julia solution

The following Julia code sets up the simple QP and solves it using Convex.jl.

Part (a) and (b):

```

using Convex

Q = [1 -1/2; -1/2 2];
f = [-1 0]';
A = [1 2; 1 -4; 5 76];
b = [-2 -3 1]';

```

```

# part (a)
x = Variable(2);
constr = A*x <= b;
p = minimize(quad_form(x,Q)+f'*x, constr);
solve!(p);
lambda = constr.dual;
p_star = p.optval;

println(lambda);
println(p_star);
println(x.value);
println(A*x.value - b);
println(2*Q*x.value + f + A'*lambda);

# part (b)
arr_i = [0 -1 1];
delta = 0.1;
pa_table = zeros(9, 4);
count = 1;
for i in arr_i
    for j in arr_i
        p_pred = p_star - dot(lambda[1:2], [i;j])*delta;
        x = Variable(2);
        p = minimize(quad_form(x,Q)+f'*x, A*x <= b+[1;j;0]*delta);
        solve!(p);
        p_exact = p.optval;
        pa_table[count, :] = [i*delta j*delta p_pred p_exact];
        count += 1;
    end
end

```

When we run this, we find the optimal objective value is $p^* = 8.22$ and the optimal point is $x_1^* = -2.33$, $x_2^* = 0.17$. (This optimal point is unique since the objective is strictly convex.) A set of optimal dual variables is $\lambda_1^* = 2.78$, $\lambda_2^* = 2.86$ and $\lambda_3^* = 0.04$.

All languages

The KKT conditions are

$$\begin{aligned}
x_1^* + 2x_2^* &\leq u_1, & x_1^* - 4x_2^* &\leq u_2, & 5x_1^* + 76x_2^* &\leq 1 \\
\lambda_1^* &\geq 0, & \lambda_2^* &\geq 0, & \lambda_3^* &\geq 0 \\
\lambda_1^*(x_1^* + 2x_2^* - u_1) &= 0, & \lambda_2^*(x_1^* - 4x_2^* - u_2) &= 0, & \lambda_3^*(5x_1^* + 76x_2^* - 1) &= 0, \\
2x_1^* - x_2^* - 1 + \lambda_1^* + \lambda_2^* + 5\lambda_3^* &= 0, \\
4x_2^* - x_1^* + 2\lambda_1^* - 4\lambda_2^* + 76\lambda_3^* &= 0.
\end{aligned}$$

We check these numerically. The dual variable λ_1^* , λ_2^* and λ_3^* are all greater than zero and the quantities

```
A*x-b
2*Q*x+f+A'*lambda
```

are found to be very small. Thus the KKT conditions are verified.

The predicted optimal value is given by

$$p_{\text{pred}}^* = p^* - \lambda_1^* \delta_1 - \lambda_2^* \delta_2.$$

The values obtained are

δ_1	δ_2	p_{pred}^*	p_{exact}^*
0	0	8.22	8.22
0	-0.1	8.55	8.70
0	0.1	7.89	7.98
-0.1	0	8.44	8.57
-0.1	-0.1	8.77	8.82
-0.1	0.1	8.10	8.32
0.1	0	8.01	8.22
0.1	-0.1	8.34	8.71
0.1	0.1	7.68	7.75

The inequality $p_{\text{pred}}^* \leq p_{\text{exact}}^*$ is verified to be true in all cases.

[Convex.jl with the SCS solver is less accurate, p_{pred}^* and p_{exact}^* coincide within a few percents.]

4.2 A determinant maximization problem. We consider the problem

$$\begin{aligned} & \text{minimize} && \log \det X^{-1} \\ & \text{subject to} && A_i^T X A_i \preceq B_i, \quad i = 1, \dots, m, \end{aligned}$$

with variable $X \in \mathbf{S}^n$, and problem data $A_i \in \mathbf{R}^{n \times k_i}$, $B_i \in \mathbf{S}_{++}^{k_i}$, $i = 1, \dots, m$. The constraint $X \succ 0$ is implicit.

We can give several interpretations of this problem. Here is one, from statistics. Let z be a random variable in \mathbf{R}^n , with covariance matrix X , which is unknown. However, we do have (matrix) upper bounds on the covariance of the random variables $y_i = A_i^T z \in \mathbf{R}^{k_i}$, which is $A_i^T X A_i$. The problem is to find the covariance matrix for z , that is consistent with the known upper bounds on the covariance of y_i , that has the largest volume confidence ellipsoid.

Derive the Lagrange dual of this problem. Be sure to state what the dual variables are (*e.g.*, vectors, scalars, matrices), any constraints they must satisfy, and what the dual function is. If the dual function has any implicit equality constraints, make them explicit. You can assume that $\sum_{i=1}^m A_i A_i^T \succ 0$, which implies the feasible set of the original problem is bounded.

What can you say about the optimal duality gap for this problem?

Solution. We introduce a dual variable $Z_i \in \mathbf{S}^{k_i}$ for each of the matrix inequalities in the original problem. These must satisfy $Z_i \succeq 0$. The Lagrangian is

$$L(X, Z_1, \dots, Z_m) = -\log \det X + \sum_{i=1}^m \mathbf{tr} Z_i (A_i^T X A_i - B_i).$$

As a function of X , this is strictly convex, and therefore X minimizes the Lagrangian if and only if the gradient of the Lagrangian with respect to X is zero, *i.e.*,

$$-X^{-1} + \sum_{i=1}^m A_i Z_i A_i^T = 0.$$

This equation has a solution if and only if

$$\sum_{i=1}^m A_i Z_i A_i^T \succ 0.$$

In fact, this is exactly the domain of the dual function; that is, if this condition doesn't hold, the Lagrangian is unbounded below. If the condition doesn't hold, then there is a nonzero vector v for which $A_i^T v = 0$ for all i . Take $X = I + t v v^T$, where $t > 0$. Then we have

$$L(X, Z_1, \dots, Z_m) = -\log(1 + t \|v\|^2) + \sum_i \mathbf{tr} Z_i (A_i^T A_i - B_i),$$

which tends to $-\infty$ as $t \rightarrow \infty$. So we can now move forward with the assumption

$$\sum_{i=1}^m A_i Z_i A_i^T \succ 0.$$

Thus we find that the X that minimizes the Lagrangian is

$$X^* = \left(\sum_{i=1}^m A_i Z_i A_i^T \right)^{-1}.$$

Therefore, the dual function is

$$\begin{aligned} g(Z_1, \dots, Z_m) &= L(X^*, Z_1, \dots, Z_m) \\ &= -\log \det \left(\sum_{i=1}^m A_i Z_i A_i^T \right)^{-1} + \sum_{i=1}^m \mathbf{tr} Z_i (A_i^T \left(\sum_{i=1}^m A_i Z_i A_i^T \right)^{-1} A_i - B_i) \\ &= \log \det \left(\sum_{i=1}^m A_i Z_i A_i^T \right) - \sum_{i=1}^m \mathbf{tr} Z_i B_i + \mathbf{tr} \left(\sum_{i=1}^m A_i Z_i A_i^T \right) \left(\sum_{i=1}^m A_i Z_i A_i^T \right)^{-1} \\ &= \log \det \left(\sum_{i=1}^m A_i Z_i A_i^T \right) - \sum_{i=1}^m \mathbf{tr} Z_i B_i + n, \end{aligned}$$

with domain

$$\mathbf{dom} g = \left\{ (Z_1, \dots, Z_m) \mid \sum_{i=1}^m A_i Z_i A_i^T \succ 0 \right\}.$$

(We didn't expect you to be so careful in identifying the domain of g .)

The dual problem is therefore

$$\begin{aligned} &\text{maximize} && \log \det \left(\sum_{i=1}^m A_i Z_i A_i^T \right) - \sum_{i=1}^m \mathbf{tr} Z_i B_i \\ &\text{subject to} && Z_i \succeq 0, \quad i = 1, \dots, m. \end{aligned}$$

Note that for $\epsilon > 0$ and small enough, $X = \epsilon I$ is strictly feasible for the primal problem. Specifically, choose $\epsilon < \min_{i=1,\dots,m} (\lambda_{\min}(B_i)/\sigma_{\max}^2(A_i))$ (since $B_i \succ 0$, $\epsilon > 0$ exists). Therefore by Slater's condition, strong duality *always* holds.

4.3 The relative entropy between two vectors $x, y \in \mathbf{R}_{++}^n$ is defined as

$$\sum_{k=1}^n x_k \log(x_k/y_k).$$

This is a convex function, jointly in x and y . In the following problem we calculate the vector x that minimizes the relative entropy with a given vector y , subject to equality constraints on x :

$$\begin{aligned} &\text{minimize} && \sum_{k=1}^n x_k \log(x_k/y_k) \\ &\text{subject to} && Ax = b \\ & && \mathbf{1}^T x = 1 \end{aligned}$$

The optimization variable is $x \in \mathbf{R}^n$. The domain of the objective function is \mathbf{R}_{++}^n . The parameters $y \in \mathbf{R}_{++}^n$, $A \in \mathbf{R}^{m \times n}$, and $b \in \mathbf{R}^m$ are given.

Derive the Lagrange dual of this problem and simplify it to get

$$\text{maximize } b^T z - \log \sum_{k=1}^n y_k e^{a_k^T z}$$

(a_k is the k th column of A).

Solution. The Lagrangian is

$$L(x, z, \mu) = \sum_k x_k \log(x_k/y_k) + b^T z - z^T Ax + \mu - \mu \mathbf{1}^T x.$$

Minimizing over x_k gives the conditions

$$1 + \log(x_k/y_k) - a_k^T z - \mu = 0, \quad k = 1, \dots, n,$$

with solution

$$x_k = y_k e^{a_k^T z + \mu - 1}.$$

Plugging this in in L gives the Lagrange dual function

$$g(z, \mu) = b^T z + \mu - \sum_{k=1}^n y_k e^{a_k^T z + \mu - 1}$$

and the dual problem

$$\text{maximize } b^T z + \mu - \sum_{k=1}^n y_k e^{a_k^T z + \mu - 1}.$$

This can be simplified a bit if we optimize over μ by setting the derivative equal to zero:

$$\mu = 1 - \log \sum_{k=1}^n y_k e^{a_k^T z}.$$

After this simplification the dual problem reduces to the problem in the assignment.

4.4 Source localization from range measurements. [?] A signal emitted by a source at an unknown position $x \in \mathbf{R}^n$ ($n = 2$ or $n = 3$) is received by m sensors at known positions $y_1, \dots, y_m \in \mathbf{R}^n$. From the strength of the received signals, we can obtain noisy estimates d_k of the distances $\|x - y_k\|_2$. We are interested in estimating the source position x based on the measured distances d_k .

In the following problem the error between the squares of the actual and observed distances is minimized:

$$\text{minimize } f_0(x) = \sum_{k=1}^m (\|x - y_k\|_2^2 - d_k^2)^2.$$

Introducing a new variable $t = x^T x$, we can express this as

$$\begin{aligned} \text{minimize} \quad & \sum_{k=1}^m (t - 2y_k^T x + \|y_k\|_2^2 - d_k^2)^2 \\ \text{subject to} \quad & x^T x - t = 0. \end{aligned} \tag{9}$$

The variables are $x \in \mathbf{R}^n$, $t \in \mathbf{R}$. Although this problem is not convex, it can be shown that strong duality holds. (It is a variation on the problem discussed on page 229 and in exercise 5.29 of *Convex Optimization*.)

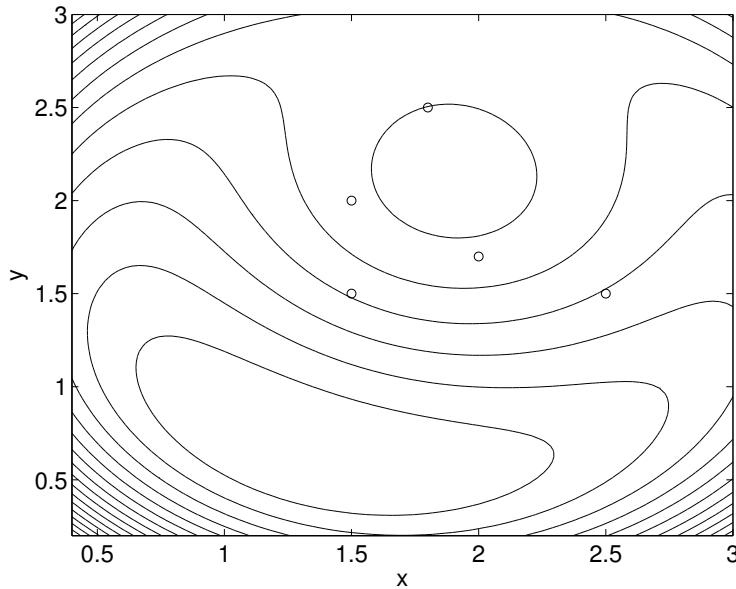
Solve (9) for an example with $m = 5$,

$$y_1 = \begin{bmatrix} 1.8 \\ 2.5 \end{bmatrix}, \quad y_2 = \begin{bmatrix} 2.0 \\ 1.7 \end{bmatrix}, \quad y_3 = \begin{bmatrix} 1.5 \\ 1.5 \end{bmatrix}, \quad y_4 = \begin{bmatrix} 1.5 \\ 2.0 \end{bmatrix}, \quad y_5 = \begin{bmatrix} 2.5 \\ 1.5 \end{bmatrix},$$

and

$$d = (2.00, 1.24, 0.59, 1.31, 1.44).$$

The figure shows some contour lines of the cost function f_0 , with the positions y_k indicated by circles.



To solve the problem, you can note that x^* is easily obtained from the KKT conditions for (9) if the optimal multiplier ν^* for the equality constraint is known. You can use one of the following two methods to find ν^* .

- Derive the dual problem, express it as an SDP, and solve it using CVX.
- Reduce the KKT conditions to a nonlinear equation in ν , and pick the correct solution (similarly as in exercise 5.29 of *Convex Optimization*).

Solution. Define

$$A = \begin{bmatrix} -2y_1^T & 1 \\ -2y_2^T & 1 \\ \vdots & \vdots \\ -2y_5^T & 1 \end{bmatrix}, \quad b = \begin{bmatrix} d_1^2 - \|y_1\|_2^2 \\ d_2^2 - \|y_2\|_2^2 \\ \vdots \\ d_5^2 - \|y_5\|_2^2 \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad f = \begin{bmatrix} 0 \\ 0 \\ -1/2 \end{bmatrix}$$

and $z = (x_1, x_2, t)$. With this notation, the problem is

$$\begin{aligned} & \text{minimize} && \|Az - b\|_2^2 \\ & \text{subject to} && z^T C z + 2f^T z = 0. \end{aligned}$$

The Lagrangian is

$$L(z, \nu) = z^T(A^T A + \nu C)z - 2(A^T b - \nu f)^T z + \|b\|_2^2,$$

which is bounded below as a function of z only if

$$A^T A + \nu C \succeq 0, \quad A^T b - \nu f \in \mathcal{R}(A^T A + \nu C).$$

The KKT conditions are therefore as follows.

- *Primal feasibility.*

$$z^T C z + 2f^T z = 0.$$

- *Dual feasibility.*

$$A^T A + \nu C \succeq 0, \quad A^T b - \nu f \in \mathcal{R}(A^T A + \nu C).$$

- *Gradient of Lagrangian is zero.*

$$(A^T A + \nu C)z = A^T b - \nu f.$$

(Note that this implies the range condition for dual feasibility.)

Method 1. We derive the dual problem. If ν is feasible, then

$$g(\nu) = -(A^T b - \nu f)^T (A^T A + \nu C)^\dagger (A^T b - \nu f) + \|b\|_2^2,$$

so the dual problem can be expressed as an SDP

$$\begin{aligned} & \text{maximize} && -t + \|b\|_2^2 \\ & \text{subject to} && \begin{bmatrix} A^T A + \nu C & A^T b - \nu f \\ (A^T b - \nu f)^T & t \end{bmatrix} \succeq 0. \end{aligned}$$

Solving this in CVX gives $\nu^* = 0.5896$. From ν^* , we get

$$z^* = (A^T A + \nu C)^{-1} (A^T b - \nu f) = (1.33, 0.64, 2.18).$$

Hence $x^* = (1.33, 0.64)$.

Method 2. Alternatively, we can solve the KKT equations directly. To simplify the equations, we make a change of variables

$$w = Q^T L^T z$$

where L is the Cholesky factor in the factorization $A^T A = LL^T$, and Q is the matrix of eigenvectors of $L^{-1}CL^{-T} = Q\Lambda Q^T$. This transforms the KKT equations to

$$w^T \Lambda w + 2g^T w = 0, \quad I + \nu \Lambda \succeq 0, \quad (I + \nu \Lambda)w = h - \nu g$$

where

$$g = Q^T L^{-1} f, \quad h = Q^T L^{-1} A^T b.$$

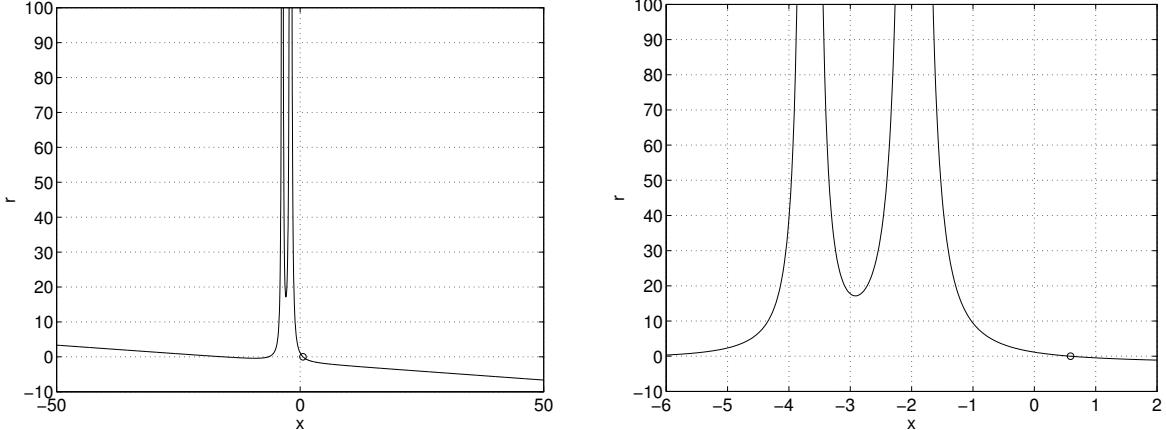
We can eliminate w from the last equation in the KKT conditions to obtain an equation in ν :

$$r(\nu) = \sum_{k=1}^{n+1} \left(\frac{\lambda_k(h_k - \nu g_k)^2}{(1 + \nu \lambda_k)^2} + \frac{2g_k(h_k - \nu g_k)}{1 + \nu \lambda_k} \right) = 0$$

In our example, the eigenvalues are

$$\lambda_1 = 0.5104, \quad \lambda_2 = 0.2735, \quad \lambda_3 = 0.$$

The figure shows the function r on two different scales.

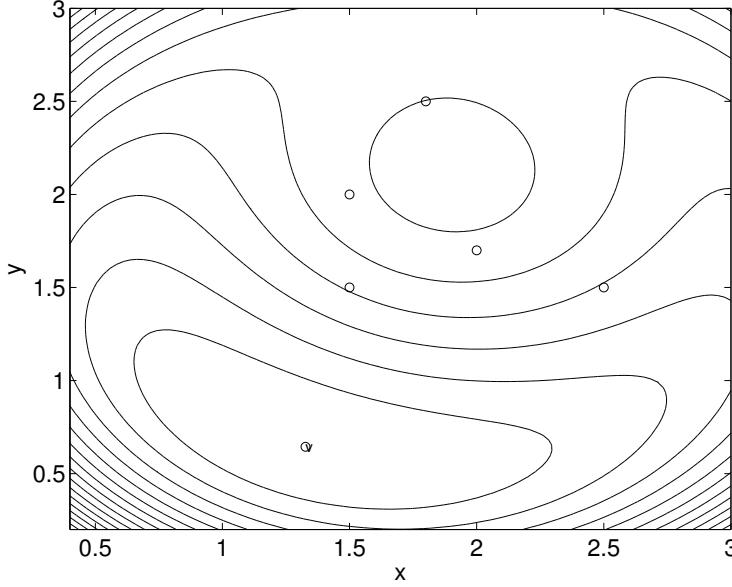


The correct solution of $r(\nu) = 0$ is the one that satisfies $1 + \nu \lambda_k \geq 0$ for $k = 1, 2, 3$, i.e., the solution to the right of the two singularities. This solution can be determined using Newton's method by repeating the iteration

$$\nu := \nu - \frac{r(\nu)}{r'(\nu)}$$

a few times, starting at a value close to the solution. This gives $\nu^* = 0.5896$. From ν^* , we determine x^* as in the first method.

The last figure shows the contour lines and the optimal x^* .



4.5 Projection on the ℓ_1 ball. Consider the problem of projecting a point $a \in \mathbf{R}^n$ on the unit ball in ℓ_1 -norm:

$$\begin{aligned} & \text{minimize} && (1/2)\|x - a\|_2^2 \\ & \text{subject to} && \|x\|_1 \leq 1. \end{aligned}$$

Derive the dual problem and describe an efficient method for solving it. Explain how you can obtain the optimal x from the solution of the dual problem.

Solution. The Lagrangian is

$$L(x, \lambda) = \frac{1}{2}\|x - a\|_2^2 + \lambda\|x\|_1 - \lambda = \sum_{k=1}^n \frac{(x_k - a_k)^2}{2} + \lambda|x_k| - \lambda.$$

This is easy to minimize over x because it is separable:

$$g_k(\lambda) = \inf_{x_k} \left(\frac{(x_k - a_k)^2}{2} + \lambda|x_k| \right) = \begin{cases} -\lambda^2/2 + \lambda|a_k| & \lambda \leq |a_k| \\ a_k^2/2 & \lambda > |a_k|. \end{cases}$$

This is a differential concave function with derivative

$$g'_k(\lambda) = \max\{|a_k| - \lambda, 0\}.$$

The dual problem is

$$\begin{aligned} & \text{maximize} && g(\lambda) = \sum_k g_k(\lambda) - \lambda \\ & \text{subject to} && \lambda \geq 0. \end{aligned}$$

g is differentiable and concave, with derivative

$$g'(\lambda) = \sum_{k=1}^n \max\{|a_k| - \lambda, 0\} - 1.$$

The derivative varies from $\|a\|_1 - 1$ at the origin, to -1 as $\lambda \geq \max |a_k|$. If $\|a\|_1 \leq 1$, then g is decreasing on \mathbf{R}_+ , the optimal λ is zero, and the optimal x is $x = a$.

If $\|a\|_1 > 1$, we can find the optimal λ by solving the piecewise-linear equation

$$\sum_{k=1}^n \max\{|a_k| - \lambda, 0\} = 1.$$

From the optimal λ , we obtain the optimal x as follows:

$$x_k = \begin{cases} 0 & \lambda \geq |a_k| \\ a_k - \lambda & \lambda < |a_k|, a_k > 0 \\ a_k + \lambda & \lambda < |a_k|, a_k < 0. \end{cases}$$

4.6 A nonconvex problem with strong duality. On page 229 of *Convex Optimization*, we consider the problem

$$\begin{aligned} & \text{minimize} && f(x) = x^T Ax + 2b^T x \\ & \text{subject to} && x^T x \leq 1 \end{aligned} \tag{10}$$

with variable $x \in \mathbf{R}^n$, and data $A \in \mathbf{S}^n$, $b \in \mathbf{R}^n$. We do not assume that A is positive semidefinite, and therefore the problem is not necessarily convex. In this exercise we show that x is (globally) optimal if and only if there exists a λ such that

$$\|x\|_2 \leq 1, \quad \lambda \geq 0, \quad A + \lambda I \succeq 0, \quad (A + \lambda I)x = -b, \quad \lambda(1 - \|x\|_2^2) = 0. \tag{11}$$

From this we will develop an efficient method for finding the global solution. The conditions (11) are the KKT conditions for (10) with the inequality $A + \lambda I \succeq 0$ added.

- (a) Show that if x and λ satisfy (11), then $f(x) = \inf_{\tilde{x}} L(\tilde{x}, \lambda) = g(\lambda)$, where L is the Lagrangian of the problem and g is the dual function. Therefore strong duality holds, and x is globally optimal.
- (b) Next we show that the conditions (11) are also necessary. Assume that x is globally optimal for (10). We distinguish two cases.
 - (i) $\|x\|_2 < 1$. Show that (11) holds with $\lambda = 0$.
 - (ii) $\|x\|_2 = 1$. First prove that $(A + \lambda I)x = -b$ for some $\lambda \geq 0$. (In other words, the negative gradient $-(Ax + b)$ of the objective function is normal to the unit sphere at x , and point away from the origin.) You can show this by contradiction: if the condition does not hold, then there exists a direction v with $v^T x < 0$ and $v^T(Ax + b) < 0$. Show that $f(x + tv) < f(x)$ for small positive t . It remains to show that $A + \lambda I \succeq 0$. If not, there exists a w with $w^T(A + \lambda I)w < 0$, and without loss of generality we can assume that $w^T x \neq 0$. Show that the point $y = x + tw$ with $t = -2w^T x / w^T w$ satisfies $\|y\|_2 = 1$ and $f(y) < f(x)$.
- (c) The optimality conditions (11) can be used to derive a simple algorithm for (10). Using the eigenvalue decomposition $A = \sum_{i=1}^n \alpha_i q_i q_i^T$, of A , we make a change of variables $y_i = q_i^T x$, and write (10) as

$$\begin{aligned} & \text{minimize} && \sum_{i=1}^n \alpha_i y_i^2 + 2 \sum_{i=1}^n \beta_i y_i \\ & \text{subject to} && y^T y \leq 1 \end{aligned}$$

where $\beta_i = q_i^T b$. The transformed optimality conditions (11) are

$$\|y\|_2 \leq 1, \quad \lambda \geq -\alpha_n, \quad (\alpha_i + \lambda)y_i = -\beta_i, \quad i = 1, \dots, n, \quad \lambda(1 - \|y\|_2) = 0,$$

if we assume that $\alpha_1 \geq \alpha_2 \geq \dots \geq \alpha_n$. Give an algorithm for computing the solution y and λ .

Solution.

(a) Suppose x and λ satisfy the conditions. Then

$$\begin{aligned} f(x) &= f(x) + \lambda(\|x\|_2^2 - 1) \\ &= x^T(A + \lambda I)x + 2b^T x - \lambda \\ &= \inf_{\tilde{x}} (\tilde{x}^T(A + \lambda I)\tilde{x} + 2b^T \tilde{x} - \lambda) \\ &= g(\lambda). \end{aligned}$$

The first line follows from complementary slackness ($\lambda(1 - \|x\|_2^2) = 0$). Line 3 follows from the fact that the Lagrangian $L(\tilde{x}, \lambda)$ is convex in \tilde{x} if $A + \lambda I \succeq 0$ and therefore x minimizes the Lagrangian if $\nabla_x L(x, \lambda) = (A - \lambda I)x + b = 0$.

- (b) (i) If $\|x\|_2 < 1$ and x is globally optimal, then x is the unconstrained minimum of $x^T Ax + 2b^T x$. Therefore f must be convex ($A \succeq 0$) and the gradient $Ax + b$ must be zero.
- (ii) Suppose there exists a v with $(Ax + b)^T v < 0$ and $v^T x < 0$. For small positive t we have $\|x + tv\|_2 < 1$ and

$$\begin{aligned} f(x + tv) &= x^T Ax + 2b^T x + t^2 v^T Av + 2tv^T(Ax + b) \\ &< x^T Ax + 2b^T x. \end{aligned}$$

Therefore x is not even locally optimal.

For the second part, we first note that

$$y^T y = (x - 2 \frac{w^T x}{w^T w} w)^T (x - 2 \frac{w^T x}{w^T w} w) = x^T x = 1.$$

Also, using $b = -(A + \lambda I)x$,

$$\begin{aligned} y^T A y + 2b^T y &= y^T (A + \lambda I) y - 2x^T (A + \lambda I) y - \lambda \\ &= (y - x)^T (A + \lambda I) (y - x) - x^T (A + \lambda I) x - \lambda \\ &< -x^T (A + \lambda I) x - \lambda \\ &= x^T (A + \lambda I) x + 2b^T x - \lambda \\ &= x^T Ax + 2b^T x. \end{aligned}$$

The inequality on line 3 follows from $w^T (A + \lambda I) w < 0$.

- (c) We assume that the eigenvalues are sorted in nondecreasing order with

$$\alpha_1 \geq \alpha_2 \geq \dots \geq \alpha_{k-1} > \alpha_k = \alpha_{k+1} = \dots = \alpha_n.$$

In other words the smallest eigenvalue has multiplicity $n - k + 1$ where $k \geq 1$. For $\lambda > -\alpha_n$, define

$$\phi(\lambda) = \sum_{i=1}^n \frac{\beta_i^2}{(\alpha_i + \lambda)^2}.$$

We extend the function to $\lambda = -\alpha_n$ using the interpretation $a/0 = 0$ if $a = 0$ and $a/0 = \infty$ if $a > 0$. In other words, at $\lambda = -\alpha_n$ we define

$$\phi(\lambda) = \sum_{i=1}^{k-1} \frac{\beta_i^2}{(\alpha_i + \lambda)^2}$$

if $\beta_k = \dots = \beta_n = 0$ and as $\phi(\lambda) = +\infty$ otherwise. We also define $\bar{\lambda} = \max\{0, -\alpha_n\}$. Now distinguish three cases.

- $\phi(\bar{\lambda}) \geq 1$ (In other words $\phi(\bar{\lambda}) = +\infty$ or $\phi(\bar{\lambda}) \in [1, \infty)$.) Since $\phi(\lambda)$ decreases monotonically to zero as $\lambda \rightarrow \infty$ there is a unique $\lambda \geq \bar{\lambda}$ that satisfies the nonlinear equation $\phi(\lambda) = 1$. This equation is sometimes called the *secular* equation and is easily solved for λ (using Newton's method or the bisection method). From the solution λ , determine y as

$$y_i = -\frac{\beta_i}{\alpha_i + \lambda}, \quad i = 1, \dots, n$$

(with the interpretation $0/0 = 0$ if λ happens to be equal to α_n). It can be verified that y and λ satisfy the optimality conditions and $\|y\|_2 = 1$.

- $\phi(\bar{\lambda}) < 1$ and $\alpha_n \leq 0$. This implies $\beta_k = \dots = \beta_n = 0$ and $\bar{\lambda} = -\alpha_n$. Choose $\lambda = -\alpha_n = \bar{\lambda}$ and

$$y_i = \begin{cases} -\beta_i/(\alpha_i + \lambda) & i = 1, \dots, k-1 \\ (1 - \phi(\lambda))^{1/2} & i = k \\ 0 & i > k. \end{cases}$$

With this choice the optimality conditions are satisfied with $\|y\|_2 = 1$.

- $\phi(\bar{\lambda}) < 1$ and $\alpha_n > 0$. This implies $\bar{\lambda} = 0$. We take $\lambda = 0$ and

$$y_i = -\frac{\beta_i}{\alpha_i}, \quad i = 1, \dots, n.$$

With this choice the optimality conditions are satisfied with $\|y\|_2 < 1$.

4.7 Connection between perturbed optimal cost and Lagrange dual functions. In this exercise we explore the connection between the optimal cost, as a function of perturbations to the righthand sides of the constraints,

$$p^*(u) = \inf\{f_0(x) \mid \exists x \in \mathcal{D}, f_i(x) \leq u_i, i = 1, \dots, m\},$$

(as in §5.6), and the Lagrange dual function

$$g(\lambda) = \inf_x (f_0(x) + \lambda_1 f_1(x) + \dots + \lambda_m f_m(x)),$$

with domain restricted to $\lambda \succeq 0$. We assume the problem is convex. We consider a problem with inequality constraints only, for simplicity.

We have seen several connections between p^* and g :

- *Slater's condition and strong duality.* Slater's condition is: there exists $u \prec 0$ for which $p^*(u) < \infty$. Strong duality (which follows) is: $p^*(0) = \sup_\lambda g(\lambda)$. (Note that we include the condition $\lambda \succeq 0$ in the domain of g .)

- *A global inequality.* We have $p^*(u) \geq p^*(0) - \lambda^{*T} u$, for any u , where λ^* maximizes g .
- *Local sensitivity analysis.* If p^* is differentiable at 0, then we have $\nabla p^*(0) = -\lambda^*$, where λ^* maximizes g .

In fact the two functions are closely related by conjugation. Show that

$$p^*(u) = (-g)^*(-u).$$

Here $(-g)^*$ is the conjugate of the function $-g$. You can show this for $u \in \text{int dom } p^*$.

Hint. Consider the problem

$$\begin{aligned} & \text{minimize} && f_0(x) \\ & \text{subject to} && \tilde{f}_i(x) = f_i(x) - u_i \leq 0, \quad i = 1, \dots, m. \end{aligned}$$

Verify that Slater's condition holds for this problem, for $u \in \text{int dom } p^*$.

Solution. Suppose $u \in \text{int dom } p^*$. This means that there exists $\tilde{u} \prec u$ with $\tilde{u} \in \text{dom } p^*$, i.e., which $p^*(\tilde{u}) < \infty$. This in turn means that there exists $\tilde{x} \in \text{dom } f_0$ with $f_i(\tilde{x}) \leq \tilde{u}_i$.

Following the hint, consider the problem

$$\begin{aligned} & \text{minimize} && f_0(x) \\ & \text{subject to} && \tilde{f}_i(x) = f_i(x) - u_i \leq 0, \quad i = 1, \dots, m. \end{aligned}$$

Its optimal value is $p^*(u)$, and its dual function is

$$\tilde{g}(\lambda) = \inf_x (f_0(x) + \lambda_1(f_1(x) - u_1) + \dots + \lambda_m(f_m(x) - u_m)) = g(\lambda) - \lambda^T u.$$

Since \tilde{x} satisfies $f_i(\tilde{x}) \leq \tilde{u}_i < u_i$, it is strictly feasible for the problem above, so Slater's condition holds. Therefore we have

$$p^*(u) = \sup_{\lambda} (g(\lambda) - \lambda^T u).$$

We rewrite this as

$$p^*(u) = \sup_{\lambda} (\lambda^T(-u) - (-g(\lambda))) = (-g)^*(-u).$$

4.8 Exact penalty method for SDP.

Consider the pair of primal and dual SDPs

$$\begin{array}{ll} (\text{P}) & \text{minimize} & c^T x \\ & \text{subject to} & F(x) \preceq 0 \end{array} \quad \begin{array}{ll} (\text{D}) & \text{maximize} & \mathbf{tr}(F_0 Z) \\ & \text{subject to} & \mathbf{tr}(F_i Z) + c_i = 0, \quad i = 1, \dots, m \\ & & Z \succeq 0, \end{array}$$

where $F(x) = F_0 + x_1 F_1 + \dots + x_n F_n$ and $F_i \in \mathbf{S}^p$ for $i = 0, \dots, n$. Let Z^* be a solution of (D). Show that every solution x^* of the unconstrained problem

$$\text{minimize} \quad c^T x + M \max\{0, \lambda_{\max}(F(x))\},$$

where $M > \mathbf{tr} Z^*$, is a solution of (P).

Solution. The unconstrained problem can be written as an SDP

$$\begin{aligned} & \text{minimize} && c^T x + t \\ & \text{subject to} && F(x) \preceq tI \\ & && t \geq 0. \end{aligned}$$

The dual of this problem is

$$\begin{aligned} & \text{maximize} && \mathbf{tr}(F_0 Z) \\ & \text{subject to} && \mathbf{tr}(F_i Z) + c_i = 0, \quad i = 1, \dots, m \\ & && \mathbf{tr} Z + s = 1 \\ & && Z \succeq 0, \quad s \geq 0. \end{aligned}$$

We see that Z^* is feasible in this problem, with $s > 0$. By complementary slackness this means that $t = 0$ at the optimum of the primal problem.

4.9 Quadratic penalty. Consider the problem

$$\begin{aligned} & \text{minimize} && f_0(x) \\ & \text{subject to} && f_i(x) \leq 0, \quad i = 1, \dots, m, \end{aligned} \tag{12}$$

where the functions $f_i : \mathbf{R}^n \rightarrow \mathbf{R}$ are differentiable and convex.

Show that

$$\phi(x) = f_0(x) + \alpha \sum_{i=1}^m \max\{0, f_i(x)\}^2,$$

where $\alpha > 0$, is convex. Suppose \tilde{x} minimizes ϕ . Show how to find from \tilde{x} a feasible point for the dual of (12). Find the corresponding lower bound on the optimal value of (12).

Solution.

- (a) The function $\max\{0, f_i(x)\}^2$ is convex because it is the composition of a convex nondecreasing function $h(t) = \max\{0, t\}^2$ with a convex function f_i .
- (b) The function $h(t) = \max\{0, t\}^2$ is differentiable with $h'(t) = 2 \max\{0, t\}$. Therefore \tilde{x} satisfies

$$\nabla \phi(\tilde{x}) = \nabla f_0(\tilde{x}) + \sum_{i=1}^m \tilde{\lambda}_i \nabla f_i(\tilde{x}) = 0$$

where

$$\tilde{\lambda}_i = 2\alpha \max\{0, f_i(\tilde{x})\} = \begin{cases} 0 & f_i(\tilde{x}) \leq 0 \\ 2\alpha f_i(\tilde{x}) & f_i(\tilde{x}) > 0. \end{cases}$$

- (c) Let $g(\lambda) = \inf_x (f_0(x) + \sum_{i=1}^m \lambda_i f_i(x))$ be the dual function of (12). The lower bound corresponding to $\tilde{\lambda}$ is

$$\begin{aligned} g(\tilde{\lambda}) &= \inf_x (f_0(x) + \sum_{i=1}^m \tilde{\lambda}_i f_i(x)) \\ &= f_0(\tilde{x}) + \sum_{i=1}^m \tilde{\lambda}_i f_i(\tilde{x}) \end{aligned}$$

$$\begin{aligned}
&= f_0(\tilde{x}) + 2\alpha \sum_{i=1}^m \max\{0, f_i(\tilde{x})\} f_i(\tilde{x}) \\
&= f_0(\tilde{x}) + 2\alpha \sum_{i=1}^m \max\{0, f_i(\tilde{x})\}^2.
\end{aligned}$$

4.10 Binary least-squares. We consider the non-convex least-squares approximation problem with binary constraints

$$\begin{aligned}
&\text{minimize} && \|Ax - b\|_2^2 \\
&\text{subject to} && x_k^2 = 1, \quad k = 1, \dots, n,
\end{aligned} \tag{13}$$

where $A \in \mathbf{R}^{m \times n}$ and $b \in \mathbf{R}^m$. We assume that $\text{rank}(A) = n$, i.e., $A^T A$ is nonsingular.

One possible application of this problem is as follows. A signal $\hat{x} \in \{-1, 1\}^n$ is sent over a noisy channel, and received as $b = A\hat{x} + v$ where $v \sim \mathcal{N}(0, \sigma^2 I)$ is Gaussian noise. The solution of (13) is the maximum likelihood estimate of the input signal \hat{x} , based on the received signal b .

- (a) Derive the Lagrange dual of (13) and express it as an SDP.
- (b) Derive the dual of the SDP in part (a) and show that it is equivalent to

$$\begin{aligned}
&\text{minimize} && \text{tr}(A^T A Z) - 2b^T A z + b^T b \\
&\text{subject to} && \text{diag}(Z) = \mathbf{1} \\
&&& \begin{bmatrix} Z & z \\ z^T & 1 \end{bmatrix} \succeq 0.
\end{aligned} \tag{14}$$

Interpret this problem as a relaxation of (13). Show that if

$$\text{rank}\left(\begin{bmatrix} Z & z \\ z^T & 1 \end{bmatrix}\right) = 1 \tag{15}$$

at the optimum of (14), then the relaxation is exact, i.e., the optimal values of problems (13) and (14) are equal, and the optimal solution z of (14) is optimal for (13). This suggests a heuristic for rounding the solution of the SDP (14) to a feasible solution of (13), if (15) does not hold. We compute the eigenvalue decomposition

$$\begin{bmatrix} Z & z \\ z^T & 1 \end{bmatrix} = \sum_{i=1}^{n+1} \lambda_i \begin{bmatrix} v_i \\ t_i \end{bmatrix} \begin{bmatrix} v_i \\ t_i \end{bmatrix}^T,$$

where $v_i \in \mathbf{R}^n$ and $t_i \in \mathbf{R}$, and approximate the matrix by a rank-one matrix

$$\begin{bmatrix} Z & z \\ z^T & 1 \end{bmatrix} \approx \lambda_1 \begin{bmatrix} v_1 \\ t_1 \end{bmatrix} \begin{bmatrix} v_1 \\ t_1 \end{bmatrix}^T.$$

(Here we assume the eigenvalues are sorted in decreasing order). Then we take $x = \text{sign}(v_1)$ as our guess of good solution of (13).

- (c) We can also give a probabilistic interpretation of the relaxation (14). Suppose we interpret z and Z as the first and second moments of a random vector $v \in \mathbf{R}^n$ (i.e., $z = \mathbf{E} v$, $Z = \mathbf{E} v v^T$). Show that (14) is equivalent to the problem

$$\begin{aligned}
&\text{minimize} && \mathbf{E} \|Av - b\|_2^2 \\
&\text{subject to} && \mathbf{E} v_k^2 = 1, \quad k = 1, \dots, n,
\end{aligned}$$

where we minimize over all possible probability distributions of v .

This interpretation suggests another heuristic method for computing suboptimal solutions of (13) based on the result of (14). We choose a distribution with first and second moments $\mathbf{E} v = z$, $\mathbf{E} vv^T = Z$ (for example, the Gaussian distribution $\mathcal{N}(z, Z - zz^T)$). We generate a number of samples \tilde{v} from the distribution and round them to feasible solutions $x = \mathbf{sign}(\tilde{v})$. We keep the solution with the lowest objective value as our guess of the optimal solution of (13).

- (d) Solve the dual problem (14) using CVX. Generate problem instances using the Matlab code

```
randn('state',0)
m = 50;
n = 40;
A = randn(m,n);
xhat = sign(randn(n,1));
b = A*xhat + s*randn(m,1);
```

for four values of the noise level s : $s = 0.5$, $s = 1$, $s = 2$, $s = 3$. For each problem instance, compute suboptimal feasible solutions x using the the following heuristics and compare the results.

- (i) $x^{(a)} = \mathbf{sign}(x_{ls})$ where x_{ls} is the solution of the least-squares problem

$$\text{minimize } \|Ax - b\|_2^2.$$

- (ii) $x^{(b)} = \mathbf{sign}(z)$ where z is the optimal value of the variable z in the SDP (14).
 (iii) $x^{(c)}$ is computed from a rank-one approximation of the optimal solution of (14), as explained in part (b) above.
 (iv) $x^{(d)}$ is computed by rounding 100 samples of $\mathcal{N}(z, Z - zz^T)$, as explained in part (c) above.

Solution.

- (a) The Lagrangian is

$$L(x, \nu) = x^T(A^T A + \mathbf{diag}(\nu))x - 2b^T Ax + b^T b - \mathbf{1}^T \nu.$$

The dual function is

$$g(\nu) = \begin{cases} -\mathbf{1}^T \nu - b^T A(A^T A + \mathbf{diag}(\nu))^{\dagger} A^T b + b^T b & A^T A + \mathbf{diag}(\nu) \succeq 0, \\ & A^T b \in \mathcal{R}(A^T A + \mathbf{diag}(\nu)) \\ -\infty & \text{otherwise.} \end{cases}$$

Using Schur complements we can express the dual as an SDP

$$\begin{aligned} & \text{maximize} && -\mathbf{1}^T \nu - t + b^T b \\ & \text{subject to} && \begin{bmatrix} A^T A + \mathbf{diag}(\nu) & -A^T b \\ -b^T A & t \end{bmatrix} \succeq 0. \end{aligned}$$

(b) We first write the problem as a minimization problem

$$\begin{aligned} & \text{minimize} && \mathbf{1}^T \nu + t - b^T b \\ & \text{subject to} && \begin{bmatrix} A^T A + \text{diag}(\nu) & -A^T b \\ -b^T A & t \end{bmatrix} \succeq 0. \end{aligned}$$

We introduce a Lagrange multiplier

$$\begin{bmatrix} Z & z \\ z^T & \lambda \end{bmatrix}$$

for the constraint and form the Lagrangian

$$\begin{aligned} L(\nu, t, Z, z, \lambda) &= \mathbf{1}^T \nu + t - b^T b - \text{tr}(Z(A^T A + \text{diag}(\nu))) + 2z^T A^T b - t\lambda \\ &= (\mathbf{1} - \text{diag}(Z))^T \nu + t(1 - \lambda) - b^T b - \text{tr}(ZA^T A) + 2z^T A^T b. \end{aligned}$$

This is unbounded below unless $\text{diag}(Z) = \mathbf{1}$ and $\lambda = 1$. The dual problem of the SDP in part 1 is therefore

$$\begin{aligned} & \text{minimize} && \text{tr}(A^T AZ) - 2b^T Az + b^T b \\ & \text{subject to} && \text{diag}(Z) = \mathbf{1} \\ & && \begin{bmatrix} Z & z \\ z^T & 1 \end{bmatrix} \succeq 0. \end{aligned}$$

To see that this is a relaxation of the original problem we note that the binary LS problem is equivalent to

$$\begin{aligned} & \text{minimize} && \text{tr}(A^T AZ) - 2b^T Az + b^T b \\ & \text{subject to} && \text{diag}(Z) = \mathbf{1} \\ & && Z = zz^T. \end{aligned}$$

In the relaxation we replace $Z = zz^T$ by the weaker constraint $Z \succeq zz^T$, which is equivalent to

$$\begin{bmatrix} Z & z \\ z^T & 1 \end{bmatrix} \succeq 0.$$

(c) We have

$$\begin{aligned} \mathbf{E} \|Av - b\|_2^2 &= \mathbf{E}(v^T A^T Av - 2b^T A^T v + b^T b) \\ &= \text{tr}(\mathbf{E}(vv^T)A^T A) - 2b^T A^T \mathbf{E} v + b^T b \\ &= \text{tr}(ZA^T A) - 2b^T A^T z + b^T b \\ \mathbf{E} v_k^2 &= Z_{kk}. \end{aligned}$$

(d) The Matlab code for solving the SDP is as follows.

```
cvx_begin sdp
    variable z(n);
    variable Z(n,n) symmetric;
    minimize( trace(A'*A*Z) - 2*b'*A*z + b'*b )
    subject to
        [Z, z; z' 1] >= 0
        diag(Z)==1;
cvx_end
```

The table lists, for each s , the values of

$$f(x) = \|Ax - b\|_2$$

for $x = \hat{x}$ and the four approximations, and also the lower bound on the optimal value of

$$\begin{aligned} & \text{minimize} && \|Ax - b\|_2 \\ & \text{subject to} && x_k^2 = 1, \quad k = 1, \dots, n, \end{aligned}$$

obtained by the SDP relaxation.

s	$f(\hat{x})$	$f(x^{(a)})$	$f(x^{(b)})$	$f(x^{(c)})$	$f(x^{(d)})$	lower bound
0.5	4.1623	4.1623	4.1623	4.1623	4.1623	4.0524
1.0	8.3245	12.7299	8.3245	8.3245	8.3245	7.8678
2.0	16.6490	30.1419	16.6490	16.6490	16.6490	15.1804
3.0	24.9735	33.9339	25.9555	25.9555	24.9735	22.1139

- For $s = 0.5$, all heuristics return \hat{x} . This is likely to be the global optimum, but that is not necessarily true. However, from the lower bound we know that the global optimum is in the interval $[4.0524, 4.1623]$, so even if 4.1623 is not the global optimum, it is quite close.
- For higher values of s , the result from the first heuristic ($x^{(a)}$) is substantially worse than the SDP heuristics.
- All three SDP heuristics return \hat{x} for $s = 1$ and $s = 2$.
- For $s = 3$, the randomized rounding method returns \hat{x} . The other SDP heuristics give slightly higher values.

4.11 Monotone transformation of the objective. Consider the optimization problem

$$\begin{aligned} & \text{minimize} && f_0(x) \\ & \text{subject to} && f_i(x) \leq 0, \quad i = 1, \dots, m. \end{aligned} \tag{16}$$

where $f_i : \mathbf{R}^n \rightarrow \mathbf{R}$ for $i = 0, 1, \dots, m$ are convex. Suppose $\phi : \mathbf{R} \rightarrow \mathbf{R}$ is increasing and convex. Then the problem

$$\begin{aligned} & \text{minimize} && \tilde{f}_0(x) = \phi(f_0(x)) \\ & \text{subject to} && f_i(x) \leq 0, \quad i = 1, \dots, m \end{aligned} \tag{17}$$

is convex and equivalent to it; in fact, it has the same optimal set as (16).

In this problem we explore the connections between the duals of the two problems (16) and (17). We assume f_i are differentiable, and to make things specific, we take $\phi(a) = \exp a$.

- (a) Suppose λ is feasible for the dual of (16), and \bar{x} minimizes

$$f_0(x) + \sum_{i=1}^m \lambda_i f_i(x).$$

Show that \bar{x} also minimizes

$$\exp f_0(x) + \sum_{i=1}^m \tilde{\lambda}_i f_i(x)$$

for appropriate choice of $\tilde{\lambda}$. Thus, $\tilde{\lambda}$ is dual feasible for (17).

- (b) Let p^* denote the optimal value of (16) (so the optimal value of (17) is $\exp p^*$). From λ we obtain the bound

$$p^* \geq g(\lambda),$$

where g is the dual function for (16). From $\tilde{\lambda}$ we obtain the bound $\exp p^* \geq \tilde{g}(\tilde{\lambda})$, where \tilde{g} is the dual function for (17). This can be expressed as

$$p^* \geq \log \tilde{g}(\tilde{\lambda}).$$

How do these bounds compare? Are they the same, or is one better than the other?

Solution.

- (a) Since \bar{x} minimizes $f_0(x) + \sum_{i=1}^m \lambda_i f_i(x)$ we have

$$\nabla f_0(\bar{x}) + \sum_{i=1}^m \lambda_i \nabla f_i(\bar{x}) = 0. \quad (18)$$

But

$$\begin{aligned} \frac{\partial}{\partial x} \left[\exp f_0(x) + \sum_{i=1}^m \tilde{\lambda}_i \nabla f_i(\bar{x}) \right]_{x=\bar{x}} &= \exp f_0(\bar{x}) \nabla f_0(\bar{x}) + \sum_{i=1}^m \tilde{\lambda}_i \nabla f_i(\bar{x}) \\ &= \exp f_0(\bar{x}) \left[\nabla f_0(\bar{x}) + \sum_{i=1}^m \tilde{\lambda}_i e^{-f_0(\bar{x})} \nabla f_i(\bar{x}) \right]. \end{aligned}$$

Clearly, by comparing this equation to (18), if we take $\tilde{\lambda}_i = \exp f_0(\bar{x}) \lambda_i \geq 0$

$$\frac{\partial}{\partial x} \left[\exp f_0(x) + \sum_{i=1}^m \tilde{\lambda}_i \nabla f_i(\bar{x}) \right]_{x=\bar{x}} = 0$$

or $\tilde{\lambda}_i = \exp f_0(\bar{x}) \lambda_i$ is dual feasible for the modified problem.

- (b) We have $\tilde{g}(\tilde{\lambda}) = e^{f_0(\bar{x})} + \sum_{i=1}^m e^{f_0(\bar{x})} \lambda_i f_i(\bar{x})$ and therefore

$$\begin{aligned} \log \tilde{g}(\tilde{\lambda}) &= \log \left(e^{f_0(\bar{x})} + \sum_{i=1}^m e^{f_0(\bar{x})} \lambda_i f_i(\bar{x}) \right) \\ &= \log \left[e^{f_0(\bar{x})} \left(1 + \sum_{i=1}^m \lambda_i f_i(\bar{x}) \right) \right] \\ &= f_0(\bar{x}) + \log \left(1 + \sum_{i=1}^m \lambda_i f_i(\bar{x}) \right). \end{aligned}$$

Now we show that the bound we get from the modified problem is always worse, *i.e.*, $\log \tilde{g}(\tilde{\lambda}) \leq g(\lambda)$. This follows immediately from the identity $\log(1+y) - y \leq 0$, since $g(\lambda) = f_0(\bar{x}) + \sum_{i=1}^m \lambda_i f_i(\bar{x})$ and therefore

$$\log \tilde{g}(\tilde{\lambda}) - g(\lambda) = \log \left(1 + \sum_{i=1}^m \lambda_i f_i(\bar{x}) \right) - \sum_{i=1}^m \lambda_i f_i(\bar{x})$$

which is true by taking $y = \sum_{i=1}^m \lambda_i f_i(\bar{x})$.

4.12 Variable bounds and dual feasibility. In many problems the constraints include *variable bounds*, as in

$$\begin{aligned} & \text{minimize} && f_0(x) \\ & \text{subject to} && f_i(x) \leq 0, \quad i = 1, \dots, m \\ & && l_i \leq x_i \leq u_i, \quad i = 1, \dots, n. \end{aligned} \tag{19}$$

Let $\mu \in \mathbf{R}_+^n$ be the Lagrange multipliers associated with the constraints $x_i \leq u_i$, and let $\nu \in \mathbf{R}_+^n$ be the Lagrange multipliers associated with the constraints $l_i \geq x_i$. Thus the Lagrangian is

$$L(x, \lambda, \mu, \nu) = f_0(x) + \sum_{i=1}^m \lambda_i f_i(x) + \mu^T(x - u) + \nu^T(l - x).$$

- (a) Show that for any $x \in \mathbf{R}^n$ and any λ , we can choose $\mu \succeq 0$ and $\nu \succeq 0$ so that x minimizes $L(x, \lambda, \mu, \nu)$. In particular, it is very easy to find dual feasible points.
- (b) Construct a dual feasible point (λ, μ, ν) by applying the method you found in part (a) with $x = (l + u)/2$ and $\lambda = 0$. From this dual feasible point you get a lower bound on f^* . Show that this lower bound can be expressed as

$$f^* \geq f_0((l + u)/2) - ((u - l)/2)^T |\nabla f_0((l + u)/2)|$$

where $|\cdot|$ means componentwise. Can you prove this bound directly?

Solution.

- (a) Suppose $\lambda \in \mathbf{R}^m$ ($\lambda \succeq 0$) is given. We show that we can always find $\mu \succeq 0$ and $\nu \succeq 0$ such that (λ, μ, ν) is dual feasible. As a matter of fact, given any $x \in \mathbf{R}^n$ and $\lambda \in \mathbf{R}^m$ we can find $\mu \succeq 0$ and $\nu \succeq 0$ so that x minimizes L . We have

$$\nabla_x L(x, \lambda, \mu, \nu) = \nabla f_0(x) + \sum_{i=1}^m \lambda_i \nabla f_i(x) + (\mu - \nu).$$

If x minimizes L , we have $\nabla_x L = 0$ and therefore

$$\nu - \mu = \nabla f_0(x) + \sum_{i=1}^m \lambda_i \nabla f_i(x). \tag{20}$$

If we can choose $\mu \succeq 0$ and $\nu \succeq 0$ such that (20) holds for any x and λ we are done. But this is easy, since any vector can be written as the difference of two componentwise positive vectors. Simply take ν and μ as the positive and negative parts of $\nabla f_0(x) + \sum_{i=1}^m \lambda_i \nabla f_i(x)$ respectively, *i.e.*,

$$\begin{aligned} \nu &= \left[\nabla f_0(x) + \sum_{i=1}^m \lambda_i \nabla f_i(x) \right]^+ \\ &= \frac{1}{2} \left(|\nabla f_0(x) + \sum_{i=1}^m \lambda_i \nabla f_i(x)| + \nabla f_0(x) + \sum_{i=1}^m \lambda_i \nabla f_i(x) \right) \end{aligned}$$

and

$$\begin{aligned}\mu &= \left[\nabla f_0(x) + \sum_{i=1}^m \lambda_i \nabla f_i(x) \right]^- \\ &= \frac{1}{2} \left(|\nabla f_0(x) + \sum_{i=1}^m \lambda_i \nabla f_i(x)| - \nabla f_0(x) - \sum_{i=1}^m \lambda_i \nabla f_i(x) \right)\end{aligned}$$

where $|\cdot|$ is componentwise. Thus, x minimizes L for any given λ and some $\mu \succeq 0, \nu \succeq 0$. Therefore, if $\lambda \succeq 0$ then (λ, μ, ν) is dual feasible.

(b) In this case

$$\begin{aligned}\mu &= \frac{1}{2} (\nabla f_0((l+u)/2) - |\nabla f_0((l+u)/2)|) \\ \nu &= \frac{1}{2} (\nabla f_0((l+u)/2) + |\nabla f_0((l+u)/2)|)\end{aligned}$$

and therefore the lower bound becomes

$$\begin{aligned}L(x, 0, \mu, \nu) &= f_0(\frac{l+u}{2}) + \frac{1}{2} \left(\nabla f_0(\frac{l+u}{2}) - |\nabla f_0(\frac{l+u}{2})| \right)^T \left(\frac{l+u}{2} - u \right) \\ &\quad + \frac{1}{2} \left(\nabla f_0(\frac{l+u}{2}) + |\nabla f_0(\frac{l+u}{2})| \right)^T \left(\frac{l+u}{2} - l \right) \\ &= f_0(\frac{l+u}{2}) + (\frac{u-l}{2})^T |\nabla f_0(\frac{l+u}{2})|.\end{aligned}$$

This bound can also be derived directly. Since f_0 is convex

$$\begin{aligned}f^* &\geq f_0(\frac{u+l}{2}) + \nabla f_0(\frac{u+l}{2})^T (x^* - \frac{u+l}{2}) \\ &\geq f_0(\frac{u+l}{2}) + \inf_{l \leq x \leq u} \nabla f_0(\frac{u+l}{2})^T (x - \frac{u+l}{2}).\end{aligned}$$

But $\inf_{l \leq x \leq u} \nabla f_0((u+l)/2)^T (x - (l+u)/2)$ can be simply found by setting $(x_i - (l_i + u_i)/2)$ to its maximum value (*i.e.*, $(u_i - l_i)/2$) if $\nabla f_0((l+u)/2)_i \leq 0$, and by setting $(x_i - (l_i + u_i)/2)$ to its minimum value (*i.e.*, $(l_i - u_i)/2$) if $\nabla f_0((l+u)/2)_i > 0$. Therefore, $\inf_{l \leq x \leq u} \nabla f_0((u+l)/2)^T (x - (u+l)/2) = -|\nabla f_0((u+l)/2)|^T (u-l)/2$ and we get

$$f^* \geq f_0((u+l)/2) - |\nabla f_0((u+l)/2)|^T (u-l)/2.$$

4.13 Deducing costs from samples of optimal decision. A system (such as a firm or an organism) chooses a vector of values x as a solution of the LP

$$\begin{aligned}&\text{minimize} && c^T x \\ &\text{subject to} && Ax \succeq b,\end{aligned}$$

with variable $x \in \mathbf{R}^n$. You can think of $x \in \mathbf{R}^n$ as a vector of activity levels, $b \in \mathbf{R}^m$ as a vector of requirements, and $c \in \mathbf{R}^n$ as a vector of costs or prices for the activities. With this interpretation, the LP above finds the cheapest set of activity levels that meet all requirements. (This interpretation is not needed to solve the problem.)

We suppose that A is known, along with a set of data

$$(b^{(1)}, x^{(1)}), \dots, (b^{(r)}, x^{(r)}),$$

where $x^{(j)}$ is an optimal point for the LP, with $b = b^{(j)}$. (The solution of an LP need not be unique; all we say here is that $x^{(j)}$ is *an* optimal solution.) Roughly speaking, we have samples of optimal decisions, for different values of requirements.

You *do not* know the cost vector c . Your job is to compute the tightest possible bounds on the costs c_i from the given data. More specifically, you are to find c_i^{\max} and c_i^{\min} , the maximum and minimum possible values for c_i , consistent with the given data.

Note that if x is optimal for the LP for a given c , then it is also optimal if c is scaled by any positive factor. To normalize c , then, we will assume that $c_1 = 1$. Thus, we can interpret c_i as the relative cost of activity i , compared to activity 1.

- (a) Explain how to find c_i^{\max} and c_i^{\min} . Your method can involve the solution of a reasonable number (not exponential in n, m or r) of convex or quasiconvex optimization problems.
- (b) Carry out your method using the data found in `deducing_costs_data.m`. You may need to determine whether individual inequality constraints are tight; to do so, use a tolerance threshold of $\epsilon = 10^{-3}$. (In other words: if $a_k^T x - b_k \leq 10^{-3}$, you can consider this inequality as tight.)

Give the values of c_i^{\max} and c_i^{\min} , and make a very brief comment on the results.

Solution. A feasible point x is optimal for the LP if and only if there exists a $\lambda \succeq 0$ with $c = A^T \lambda$ and $\lambda_k(a_k^T x - b_k) = 0$, $k = 1, \dots, m$, where a_k^T is the k th row of A . So all we know about $x^{(j)}$ is that there exists a $\lambda^{(j)} \succeq 0$ with $c = A^T \lambda^{(j)}$ and $\lambda_k^{(j)}(a_k^T x^{(j)} - b_k^{(j)}) = 0$, $k = 1, \dots, m$. But we don't know the vectors $\lambda^{(j)}$.

We can express this as follows: c is a possible cost vector if and only if there exists $\lambda^{(j)}$, $j = 1, \dots, r$, such that

$$\begin{aligned} \lambda^{(j)} &\succeq 0, \quad j = 1, \dots, r \\ c &= A^T \lambda^{(j)}, \quad j = 1, \dots, r \\ \lambda_k^{(j)}(a_k^T x^{(j)} - b_k^{(j)}) &= 0, \quad j = 1, \dots, r, \quad k = 1, \dots, m. \end{aligned}$$

Here we know a_k , $x^{(j)}$, $b^{(j)}$; we don't know c or $\lambda^{(j)}$. But careful examination of these equations and inequalities shows that they are in fact a set of linear equalities and inequalities on the unknowns, c and $\lambda^{(j)}$. So we can minimize (or maximize) over c_i by solving the optimization problem:

$$\begin{aligned} &\text{minimize} && c_i \\ &\text{subject to} && \lambda^{(j)} \succeq 0, \quad j = 1, \dots, r \\ &&& c = A^T \lambda^{(j)}, \quad j = 1, \dots, r \\ &&& \lambda_k^{(j)}(a_k^T x^{(j)} - b_k^{(j)}) = 0, \quad j = 1, \dots, r, \quad k = 1, \dots, m \\ &&& c_1 = 1, \end{aligned}$$

with variables c and $\lambda^{(1)}, \dots, \lambda^{(r)}$. The solution gives us c_i^{\min} . If you look at this carefully, you'll see that it is an LP. $\lambda_k^{(j)}(a_k^T x^{(j)} - b_k^{(j)}) = 0$ reduces to $\lambda_k^{(j)} \geq 0$ when $a_k^T x^{(j)} = b_k^{(j)}$, and $\lambda_k^{(j)} = 0$ when $a_k^T x^{(j)} > b_k^{(j)}$. If we maximize instead of minimize, we get c_i^{\max} . Therefore, we solve a set of $2(n-1)$ LPs: one to maximize each c_i and one to minimize each c_i , for $i = 2, \dots, n$.

In the problem instance, $m = 10$, $n = 5$, and $r = 10$. Our method requires the solution of 8 LPs. The following Matlab code solves the problem.

```
%this script solves the deducing costs problem
deducing_costs_data;

cmin = ones(n,1);
cmax = ones(n,1);
for i = 2:n
% calculate lower bound cmin(i)
cvx_begin
variables c(n) lambdas(m,r)
    minimize(c(i))
    c(1) == 1;
    lambdas >= 0;
    lambdas(abs(A*X-B)>=1e-3) == 0;
    c*ones(1,r) == A'*lambdas;
cvx_end
cmin(i) = cvx_optval;

% calculate upper bound cmax(i)
cvx_begin
variables c(n) lambdas(m,r)
    maximize(c(i))
    c(1) == 1;
    lambdas >= 0;
    lambdas(abs(A*X-B)>=1e-3) == 0;
    c*ones(1,r) == A'*lambdas;
cvx_end
cmax(i) = cvx_optval;
end

disp('cmin c_true cmax:')
[cmin c_true cmax]
```

The bounds, displayed as [cmin c_true cmax], are

```
ans =
```

1.0000	1.0000	1.0000
0.6477	1.3183	1.7335
0.0609	0.3077	0.4044
0.0161	1.0190	1.5127
0.7378	0.8985	1.0723

The tightest bounds are on c_3 and c_5 and the weakest bound is on c_4 .

4.14 Kantorovich inequality.

- (a) Suppose $a \in \mathbf{R}^n$ with $a_1 \geq a_2 \geq \dots \geq a_n > 0$, and $b \in \mathbf{R}^n$ with $b_k = 1/a_k$.

Derive the KKT conditions for the convex optimization problem

$$\begin{aligned} &\text{minimize} && -\log(a^T x) - \log(b^T x) \\ &\text{subject to} && x \succeq 0, \quad \mathbf{1}^T x = 1. \end{aligned}$$

Show that $x = (1/2, 0, \dots, 0, 1/2)$ is optimal.

- (b) Suppose $A \in \mathbf{S}_{++}^n$ with eigenvalues λ_k sorted in decreasing order. Apply the result of part (a), with $a_k = \lambda_k$, to prove the *Kantorovich inequality*:

$$2 \left(u^T A u \right)^{1/2} \left(u^T A^{-1} u \right)^{1/2} \leq \sqrt{\frac{\lambda_1}{\lambda_n}} + \sqrt{\frac{\lambda_n}{\lambda_1}}$$

for all u with $\|u\|_2 = 1$.

Solution.

- (a) The KKT conditions are

$$\frac{1}{a^T x} a + \frac{1}{b^T x} b \preceq \nu \mathbf{1} \quad x \succeq 0, \quad \mathbf{1}^T x = 1,$$

plus the complementary slackness conditions

$$x_k \left(\nu - \frac{1}{a^T x} a_k - \frac{1}{b^T x} b_k \right) = 0, \quad k = 1, \dots, n.$$

We show that $x = (1/2, 0, \dots, 0, 1/2)$, $\nu = 2$ solve these equations, and hence are primal and dual optimal.

The feasibility conditions $x \succeq 0$, $\mathbf{1}^T x = 1$ obviously hold, and the complementary slackness conditions are trivially satisfied for $k = 2, \dots, n-2$. It remains to verify the inequality

$$\frac{a_k}{a^T x} + \frac{b_k}{b^T x} \leq \nu, \quad k = 1, \dots, n, \tag{21}$$

and the complementary slackness condition

$$x_k \left(\nu - \frac{1}{a^T x} a_k - \frac{1}{b^T x} b_k \right) = 0, \quad k = 1, n. \tag{22}$$

For $x = (1/2, 0, \dots, 0, 1/2)$, $\nu = 2$ the inequality (21) holds with equality for $k = 1$ and $k = n$, since

$$\frac{a_1}{a^T x} + \frac{b_1}{b^T x} = \frac{2a_1}{a_1 + a_n} + \frac{2/a_1}{1/a_1 + 1/a_n} = 2,$$

and

$$\frac{a_n}{a^T x} + \frac{b_n}{b^T x} = \frac{2a_n}{a_1 + a_n} + \frac{2/a_n}{1/a_1 + 1/a_n} = 2.$$

Therefore also (22) is satisfied. The remaining inequalities in (21) reduce to

$$\frac{a_k}{a^T x} + \frac{b_k}{b^T x} = 2 \frac{a_k + a_1 a_n / a_k}{a_1 + a_n} \leq 2, \quad k = 2, \dots, n-1.$$

This is valid, since it holds with equality for $k=1$ and $k=n$, and the function $t + a_1 a_n / t$ is convex in t , so

$$\frac{t + a_1 a_n / t}{a_1 + a_n} \leq 2$$

for all $t \in [a_n, a_1]$.

- (b) Diagonalize A using its eigenvalue decomposition $A = Q\Lambda Q^T$, and define $a_k = \lambda_k$, $b_k = 1/\lambda_k$, $x_k = (Q^T u)_k^2$. From part (a), $Q^T u = (1/\sqrt{2}, 0, \dots, 1/\sqrt{2})$ is optimal. Therefore,

$$\begin{aligned} (u^T A u)(u^T A^{-1} u) &\leq \frac{1}{4}(\lambda_1 + \lambda_n)(\lambda_1^{-1} + \lambda_n^{-1}) \\ &= \frac{1}{4} \left(\sqrt{\frac{\lambda_1}{\lambda_n}} + \sqrt{\frac{\lambda_n}{\lambda_1}} \right)^2. \end{aligned}$$

4.15 State and solve the optimality conditions for the problem

$$\begin{aligned} \text{minimize} \quad & \log \det \left(\begin{bmatrix} X_1 & X_2 \\ X_2^T & X_3 \end{bmatrix}^{-1} \right) \\ \text{subject to} \quad & \mathbf{tr} X_1 = \alpha \\ & \mathbf{tr} X_2 = \beta \\ & \mathbf{tr} X_3 = \gamma. \end{aligned}$$

The optimization variable is

$$X = \begin{bmatrix} X_1 & X_2 \\ X_2^T & X_3 \end{bmatrix},$$

with $X_1 \in \mathbf{S}^n$, $X_2 \in \mathbf{R}^{n \times n}$, $X_3 \in \mathbf{S}^n$. The domain of the objective function is \mathbf{S}_{++}^{2n} . We assume $\alpha > 0$, and $\alpha\gamma > \beta^2$.

Solution. This is a convex problem with three equality constraints

$$\begin{aligned} \text{minimize} \quad & f_0(X) \\ \text{subject to} \quad & h_1(X) = \alpha \\ & h_2(X) = \beta \\ & h_3(X) = \gamma, \end{aligned}$$

where $f_0(X) = -\log \det X$ and

$$h_1(X) = \mathbf{tr} \left(\begin{bmatrix} I & 0 \\ 0 & 0 \end{bmatrix} X \right), \quad h_2(X) = (1/2) \mathbf{tr} \left(\begin{bmatrix} 0 & I \\ I & 0 \end{bmatrix} X \right), \quad h_3(X) = \mathbf{tr} \left(\begin{bmatrix} 0 & 0 \\ 0 & I \end{bmatrix} X \right).$$

The general optimality condition for an equality constrained problem,

$$\nabla f_0(X) + \sum_{i=1}^3 \nu_i \nabla h_i(X) = 0,$$

reduces to

$$-X^{-1} + \nu_1 \begin{bmatrix} I & 0 \\ 0 & 0 \end{bmatrix} + (\nu_2/2) \begin{bmatrix} 0 & I \\ I & 0 \end{bmatrix} + \nu_3 \begin{bmatrix} 0 & 0 \\ 0 & I \end{bmatrix} = 0,$$

along with the feasibility conditions $\mathbf{tr} X_1 = \alpha$, $\mathbf{tr} X_2 = \beta$, $\mathbf{tr} X_3 = \gamma$.

From the first condition

$$X = \begin{bmatrix} \nu_1 I & (\nu_2/2)I \\ (\nu_2/2)I & \nu_3 I \end{bmatrix}^{-1} = \begin{bmatrix} \lambda_1 I & \lambda_2 I \\ \lambda_2 I & \lambda_3 I \end{bmatrix}$$

where

$$\begin{bmatrix} \lambda_1 & \lambda_2 \\ \lambda_2 & \lambda_3 \end{bmatrix} = \begin{bmatrix} \nu_1 & (\nu_2/2) \\ (\nu_2/2) & \nu_3 \end{bmatrix}^{-1}.$$

From the feasibility conditions we see that we have to choose λ_i (and hence ν_i), such that

$$\lambda_1 = \alpha/n, \quad \lambda_2 = \beta/n, \quad \lambda_3 = \gamma/n.$$

We conclude that the optimal solution is

$$X = (1/n) \begin{bmatrix} \alpha I & \beta I \\ \beta I & \gamma I \end{bmatrix}.$$

4.16 Consider the optimization problem

$$\begin{aligned} & \text{minimize} && -\log \det X + \mathbf{tr}(SX) \\ & \text{subject to} && X \text{ is tridiagonal} \end{aligned}$$

with domain \mathbf{S}_{++}^n and variable $X \in \mathbf{S}^n$. The matrix $S \in \mathbf{S}^n$ is given. Show that the optimal X_{opt} satisfies

$$(X_{\text{opt}}^{-1})_{ij} = S_{ij}, \quad |i - j| \leq 1.$$

Solution.

$$\begin{aligned} & \text{minimize} && -\log \det X + \mathbf{tr}(SX) \\ & \text{subject to} && \mathbf{tr}((e_i e_j^T + e_j e_i^T)X) = 0, \quad i - j > 1. \end{aligned}$$

The optimality conditions are

$$X \succ 0, \quad X \text{ tridiagonal}, \quad X^{-1} = S + \sum_{i-j>1} \nu_{ij} (e_i e_j^T + e_j e_i^T).$$

From the last condition we see that $(X^{-1})_{ij} = S_{ij}$ if $|i - j| \leq 1$.

4.17 We denote by $f(A)$ the sum of the largest r eigenvalues of a symmetric matrix $A \in \mathbf{S}^n$ (with $1 \leq r \leq n$), i.e.,

$$f(A) = \sum_{k=1}^r \lambda_k(A),$$

where $\lambda_1(A), \dots, \lambda_n(A)$ are the eigenvalues of A sorted in decreasing order.

- (a) Show that the optimal value of the SDP

$$\begin{aligned} & \text{maximize} && \mathbf{tr}(AX) \\ & \text{subject to} && \mathbf{tr} X = r \\ & && 0 \preceq X \preceq I, \end{aligned}$$

with variable $X \in \mathbf{S}^n$, is equal to $f(A)$.

- (b) Show that f is a convex function.

- (c) Assume $A(x) = A_0 + x_1 A_1 + \cdots + x_m A_m$, with $A_k \in \mathbf{S}^n$. Use the observation in part (a) to formulate the optimization problem

$$\text{minimize } f(A(x)),$$

with variable $x \in \mathbf{R}^m$, as an SDP.

Solution.

- (a) Let $A = V\Lambda V^T = \sum_{k=1}^n \lambda_k v_k v_k^T$ be the eigenvalue decomposition of A . If we make a change of variables $Y = V^T X V$ the problem becomes

$$\begin{aligned} & \text{maximize} && \mathbf{tr}(\Lambda Y) \\ & \text{subject to} && \mathbf{tr} Y = r \\ & && 0 \preceq Y \preceq I. \end{aligned}$$

We can assume Y is diagonal at the optimum. To see this, note that the objective function and the first constraint only involve the diagonal elements of Y . Moreover, if Y satisfies $0 \preceq Y \preceq I$, then its diagonal elements satisfy $0 \leq Y_{ii} \leq 1$. Therefore if a non-diagonal Y is optimal, then setting its off-diagonal elements to zero yields another feasible matrix with the same objective value.

To find the optimal value we can therefore solve

$$\begin{aligned} & \text{maximize} && \sum_{i=1}^n \lambda_i Y_{ii} \\ & \text{subject to} && \sum_{i=1}^n Y_{ii} = r \\ & && 0 \leq Y_{ii} \leq 1, \quad i = 1, \dots, n. \end{aligned}$$

Since the eigenvalues λ_i are sorted in nonincreasing order, an optimal solution is $Y_{11} = \dots = Y_{rr} = 1$, $Y_{r+1,r+1} = \dots = Y_{nn} = 0$. Converting back to the original variables gives

$$X = \sum_{k=1}^r v_k v_k^T.$$

- (b) Any function of the form $\sup_{X \in C} \mathbf{tr}(AX)$ is convex in A .
(c) To derive the dual of the problem in part (a), we first write it as a minimization

$$\begin{aligned} & \text{minimize} && -\mathbf{tr}(AX) \\ & \text{subject to} && \mathbf{tr} X = r \\ & && 0 \preceq X \preceq I. \end{aligned}$$

The Lagrangian is

$$\begin{aligned} L(X, \nu, U, V) &= -\mathbf{tr}(AX) + \nu(\mathbf{tr} X - r) - \mathbf{tr}(UX) + \mathbf{tr}(V(X - I)) \\ &= \mathbf{tr}((-A + \nu I - U + V)X) - r\nu - \mathbf{tr} V. \end{aligned}$$

By minimizing over X we obtain the dual function

$$g(\nu, U, V) = \begin{cases} -r\nu - \mathbf{tr} V & -A + \nu I - U + V = 0 \\ -\infty & \text{otherwise.} \end{cases}$$

The dual problem is

$$\begin{aligned} &\text{maximize} && -r\nu - \mathbf{tr} V \\ &\text{subject to} && A - \nu I = V - U \\ & && U \succeq 0, \quad V \succeq 0. \end{aligned}$$

If we change the dual problem to a minimization and eliminate the variable U , we obtain a dual problem for the SDP in part (a) of the assignment:

$$\begin{aligned} &\text{minimize} && r\nu + \mathbf{tr} V \\ &\text{subject to} && A - \nu I \preceq V \\ & && V \succeq 0. \end{aligned}$$

By strong duality, the optimal value of this problem is equal to $f(A)$. We can therefore minimize $f(A(x))$ over x by solving the

$$\begin{aligned} &\text{minimize} && r\nu + \mathbf{tr} V \\ &\text{subject to} && A(x) - \nu I \preceq V \\ & && V \succeq 0, \end{aligned}$$

which is an SDP in the variables $\nu \in \mathbf{R}$, $V \in \mathbf{S}^n$, $x \in \mathbf{R}^m$.

4.18 An exact penalty function. Suppose we are given a convex problem

$$\begin{aligned} &\text{minimize} && f_0(x) \\ &\text{subject to} && f_i(x) \leq 0, \quad i = 1, \dots, m \end{aligned} \tag{23}$$

with dual

$$\begin{aligned} &\text{maximize} && g(\lambda) \\ &\text{subject to} && \lambda \succeq 0. \end{aligned} \tag{24}$$

We assume that Slater's condition holds, so we have strong duality and the dual optimum is attained. For simplicity we will assume that there is a unique dual optimal solution λ^* .

For fixed $t > 0$, consider the unconstrained minimization problem

$$\text{minimize } f_0(x) + t \max_{i=1,\dots,m} f_i(x)^+, \tag{25}$$

where $f_i(x)^+ = \max\{f_i(x), 0\}$.

- (a) Show that the objective function in (25) is convex.

(b) We can express (25) as

$$\begin{aligned} & \text{minimize} && f_0(x) + ty \\ & \text{subject to} && f_i(x) \leq y, \quad i = 1, \dots, m \\ & && 0 \leq y \end{aligned} \tag{26}$$

where the variables are x and $y \in \mathbf{R}$.

Find the Lagrange dual problem of (26) and express it in terms of the Lagrange dual function g for problem (23).

- (c) Use the result in (b) to prove the following property. If $t > \mathbf{1}^T \lambda^*$, then any minimizer of (25) is also an optimal solution of (23).

(The second term in (25) is called a *penalty function* for the constraints in (23). It is zero if x is feasible, and adds a penalty to the cost function when x is infeasible. The penalty function is called *exact* because for t large enough, the solution of the unconstrained problem (25) is also a solution of (23).)

Solution.

- (a) The first term is convex. The second term is convex since it can be expressed as

$$\max_{i=1, \dots, m} f_i(x)^+ = \max\{f_1(x), \dots, f_m(x), 0\},$$

i.e., the pointwise maximum of a number of convex functions.

- (b) The Lagrangian is

$$L(x, y, \lambda, \mu) = f_0(x) + ty + \sum_{i=1}^m \lambda_i(f_i(x) - y) - \mu y.$$

The dual function is

$$\begin{aligned} \inf_{x,y} L(x, y, \lambda, \mu) &= \inf_{x,y} f_0 + ty + \sum_{i=1}^m \lambda_i(f_i(x) - y) - \mu y \\ &= \begin{cases} g(\lambda) & \text{if } \mathbf{1}^T \lambda + \mu = t \\ -\infty & \text{otherwise.} \end{cases} \end{aligned}$$

Therefore the dual of problem (5) is

$$\begin{aligned} & \text{maximize} && g(\lambda) \\ & \text{subject to} && \mathbf{1}^T \lambda + \mu = t \\ & && \lambda \succeq 0, \quad \mu \geq 0, \end{aligned}$$

or, equivalently,

$$\begin{aligned} & \text{maximize} && g(\lambda) \\ & \text{subject to} && \mathbf{1}^T \lambda \leq t \\ & && \lambda \succeq 0. \end{aligned}$$

- (c) If $\mathbf{1}^T \lambda^* < t$, then λ^* is also optimal for the dual problem derived in part (b). By complementary slackness $y = 0$ in the optimal solution of the primal problem (5), and the optimal x satisfies $f_i(x) \leq 0$, $i = 1, \dots, m$, i.e., it is feasible in the original problem (2), and therefore also optimal.

4.19 Infimal convolution. Let f_1, \dots, f_m be convex functions on \mathbf{R}^n . Their *infimal convolution*, denoted $g = f_1 \diamond \dots \diamond f_m$ (several other notations are also used), is defined as

$$g(x) = \inf\{f_1(x_1) + \dots + f_m(x_m) \mid x_1 + \dots + x_m = x\},$$

with the natural domain (i.e., defined by $g(x) < \infty$). In one simple interpretation, $f_i(x_i)$ is the cost for the i th firm to produce a mix of products given by x_i ; $g(x)$ is then the optimal cost obtained if the firms can freely exchange products to produce, all together, the mix given by x . (The name ‘convolution’ presumably comes from the observation that if we replace the sum above with the product, and the infimum above with integration, then we obtain the normal convolution.)

- (a) Show that g is convex.
- (b) Show that $g^* = f_1^* + \dots + f_m^*$. In other words, the conjugate of the infimal convolution is the sum of the conjugates.
- (c) Verify the identity in part (b) for the specific case of two strictly convex quadratic functions, $f_i(x) = (1/2)x^T P_i x$, with $P_i \in \mathbf{S}_{++}^n$, $i = 1, 2$.

Hint: Depending on how you work out the conjugates, you might find the matrix identity $(X + Y)^{-1}Y = X^{-1}(X^{-1} + Y^{-1})^{-1}$ useful.

Solution.

- (a) We can express g as

$$g(x) = \inf_{x_1, \dots, x_m} (f_1(x_1) + \dots + f_m(x_m) + \phi(x_1, \dots, x_m, x)),$$

where $\phi(x_1, \dots, x_m, x)$ is 0 when $x_1 + \dots + x_m = x$, and ∞ otherwise. The function on the righthand side above is convex in x_1, \dots, x_m, x , so by the partial minimization rule, so is g .

- (b) We have

$$\begin{aligned} g^*(y) &= \sup_x (y^T x - f(x)) \\ &= \sup_x \left(y^T x - \inf_{x_1 + \dots + x_m = x} f_1(x_1) + \dots + f_m(x_m) \right) \\ &= \sup_{x=x_1+\dots+x_m} \left(y^T x_1 - f_1(x_1) + \dots + y^T x_m - f_m(x_m) \right), \end{aligned}$$

where we use the fact that $(-\inf S)$ is the same as $(\sup -S)$. The last line means we are to take the supremum over all x and all x_1, \dots, x_m that sum to x . But this is the same as just taking the supremum over all x_1, \dots, x_m , so we get

$$\begin{aligned} g^*(y) &= \sup_{x_1, \dots, x_m} \left(y^T x_1 - f_1(x_1) + \dots + y^T x_m - f_m(x_m) \right) \\ &= \sup_{x_1} (y^T x_1 - f_1(x_1)) + \dots + \sup_{x_m} (y^T x_m - f_m(x_m)) \\ &= f_1^*(y) + \dots + f_m^*(y). \end{aligned}$$

(c) For $f_i(x) = (1/2)x^T P_i x$, we have $f_i^*(y) = (1/2)y^T P_i^{-1} y$ (see Example 3.22 in the textbook). Therefore, we have $f_1^* + f_2^* = (1/2)y^T(P_1^{-1} + P_2^{-1})y$.

On the other hand, to compute $g = f_1 \diamond f_2$, we need to evaluate

$$g(x) = \inf\{(1/2)x_1^T P_1 x_1 + (1/2)x_2^T P_2 x_2 \mid x_1 + x_2 = x\}.$$

This requires solving a linearly constrained convex quadratic problem. The optimality condition is

$$\begin{bmatrix} P_1 & 0 & I \\ 0 & P_2 & I \\ I & I & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \nu \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ x \end{bmatrix}.$$

Applying block elimination, we obtain the equation $-(P_1^{-1} + P_2^{-1})\nu = x$, and thus we have:

$$x_1^* = P_1^{-1}(P_1^{-1} + P_2^{-1})^{-1}x, \quad x_2^* = P_2^{-1}(P_1^{-1} + P_2^{-1})^{-1}x.$$

Plugging in these values into the expression of $g(x)$ yields:

$$g(x) = (1/2)x^T(P_1^{-1} + P_2^{-1})^{-1}x,$$

and thus (since it is a quadratic function), we have indeed:

$$g^*(y) = (1/2)y^T(P_1^{-1} + P_2^{-1})y = f_1^*(y) + f_2^*(y).$$

As an aside, notice that the quadratic form $g(x)$ corresponds to the (matrix) *harmonic mean* of the matrices P_i .

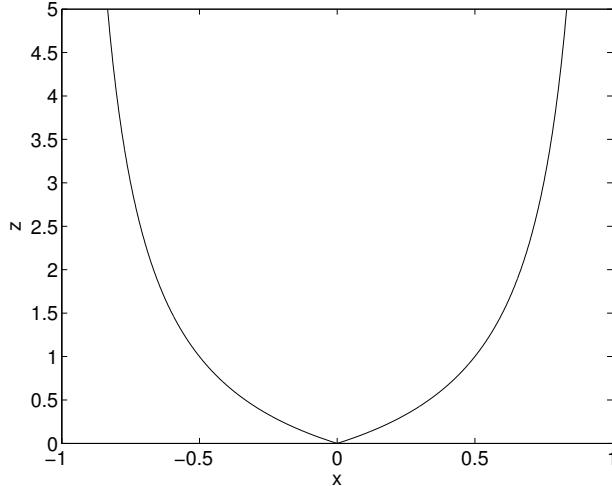
4.20 Derive the Lagrange dual of the optimization problem

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^n \phi(x_i) \\ \text{subject to} & Ax = b \end{array}$$

with variable $x \in \mathbf{R}^n$, where

$$\phi(u) = \frac{|u|}{c - |u|} = -1 + \frac{c}{c - |u|}, \quad \mathbf{dom} \phi = (-c, c).$$

c is a positive parameter. The figure shows ϕ for $c = 1$.



Solution. The Lagrangian is

$$L(x, z) = \sum_{i=1}^n (\phi(x_i) - x_i(a_i^T z)) + b^T z$$

where a_i is the i th column of A . The dual function is

$$\begin{aligned} g(z) &= b^T z + \sum_{i=1}^n \inf_{x_i} (\phi(x_i) - x_i(a_i^T z)) \\ &= b^T z + \sum_i h(a_i^T z) \end{aligned}$$

where $h(y) = \inf_u (\phi(u) - yu)$. If $|y| \leq 1/c$, the minimizer in the definition of h is $u = 0$ and $h(y) = 0$. Otherwise, we find the minimum by setting the derivative equal to zero. If $y > 1/c$, we solve

$$\phi'(u) = \frac{c}{(c-u)^2} = y.$$

The solution is $u = c - (c/y)^{1/2}$ and $h(y) = -(1 - \sqrt{cy})^2$. If $y < -1/c$, we solve

$$\phi'(u) = -\frac{c}{(c+u)^2} = y.$$

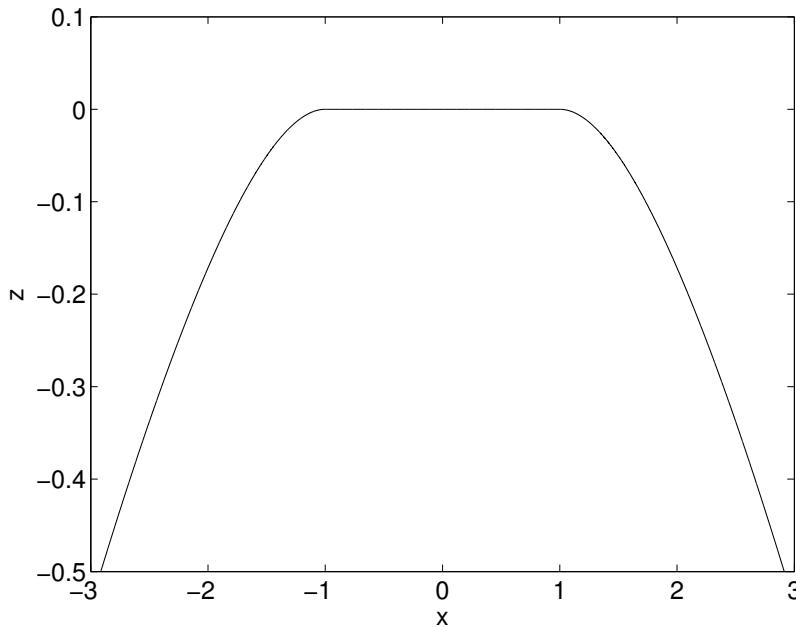
The solution is $u = -c + (-c/y)^{1/2}$ and $h(y) = -(1 - \sqrt{-cy})^2$. Combining the different cases, we can write

$$h(u) = \begin{cases} -\left(1 - \sqrt{c|y|}\right)^2 & |y| > 1/c \\ 0 & \text{otherwise.} \end{cases}$$

The dual problem is

$$\text{maximize } b^T z + \sum_i h(a_i^T z).$$

The figure shows the function h for $c = 1$.



4.21 Robust LP with polyhedral cost uncertainty. We consider a robust linear programming problem, with polyhedral uncertainty in the cost:

$$\begin{aligned} & \text{minimize} && \sup_{c \in \mathcal{C}} c^T x \\ & \text{subject to} && Ax \succeq b, \end{aligned}$$

with variable $x \in \mathbf{R}^n$, where $\mathcal{C} = \{c \mid Fc \preceq g\}$. You can think of x as the quantities of n products to buy (or sell, when $x_i < 0$), $Ax \succeq b$ as constraints, requirements, or limits on the available quantities, and \mathcal{C} as giving our knowledge or assumptions about the product prices at the time we place the order. The objective is then the worst possible (*i.e.*, largest) possible cost, given the quantities x , consistent with our knowledge of the prices.

In this exercise, you will work out a tractable method for solving this problem. You can assume that $\mathcal{C} \neq \emptyset$, and the inequalities $Ax \succeq b$ are feasible.

- (a) Let $f(x) = \sup_{c \in \mathcal{C}} c^T x$ be the objective in the problem above. Explain why f is convex.
- (b) Find the dual of the problem

$$\begin{aligned} & \text{maximize} && c^T x \\ & \text{subject to} && Fc \preceq g, \end{aligned}$$

with variable c . (The problem data are x , F , and g .) Explain why the optimal value of the dual is $f(x)$.

- (c) Use the expression for $f(x)$ found in part (b) in the original problem, to obtain a single LP equivalent to the original robust LP.
- (d) Carry out the method found in part (c) to solve a robust LP with the data below. In MATLAB:

```
rand('seed',0);
A = rand(30,10);
b = rand(30,1);
c_nom = 1+rand(10,1); % nominal c values
```

In Python:

```
import numpy as np
np.random.seed(10)
(m, n) = (30, 10)
A = np.random.rand(m, n); A = np.asmatrix(A)
b = np.random.rand(m, 1); b = np.asmatrix(b)
c_nom = np.ones((n, 1)) + np.random.rand(n, 1); c_nom = np.asmatrix(c_nom)
```

In Julia:

```
srand(10);
n = 10;
m = 30;
A = rand(m, n);
b = rand(m, 1);
c_nom = 1 + rand(n, 1);
```

Then, use \mathcal{C} described as follows. Each c_i deviates no more than 25% from its nominal value, *i.e.*, $0.75c_{\text{nom}} \leq c \leq 1.25c_{\text{nom}}$, and the average of c does not deviate more than 10% from the average of the nominal values, *i.e.*, $0.9(\mathbf{1}^T c_{\text{nom}})/n \leq \mathbf{1}^T c/n \leq 1.1(\mathbf{1}^T c_{\text{nom}})/n$.

Compare the worst-case cost $f(x)$ and the nominal cost $c_{\text{nom}}^T x$ for x optimal for the robust problem, and for x optimal for the nominal problem (*i.e.*, the case where $\mathcal{C} = \{c_{\text{nom}}\}$). Compare the values and make a brief comment.

Solution.

- (a) For each $c \in \mathcal{C}$, $c^T x$ is linear, therefore convex. $f(x)$ is the supremum of convex functions, and therefore convex.

- (b) The dual is

$$\begin{aligned} & \text{minimize} && \lambda^T g \\ & \text{subject to} && F^T \lambda = x, \quad \lambda \succeq 0, \end{aligned}$$

with variable λ . Since the primal problem is feasible (we know this since we assume $\mathcal{C} \neq \emptyset$), we are guaranteed there is zero duality gap, so $f(x)$, the optimal value of the primal problem, is also the optimal value of the dual above.

- (c) Substituting our expression for $f(x)$ into the original problem, we arrive at

$$\begin{aligned} & \text{minimize} && \inf\{\lambda^T g \mid F^T \lambda = x, \lambda \succeq 0\} \\ & \text{subject to} && Ax \succeq b. \end{aligned}$$

We can just as well minimize over x and λ at the same time, which gives the problem

$$\begin{aligned} & \text{minimize} && \lambda^T g \\ & \text{subject to} && Ax \succeq b, \quad F^T \lambda = x, \quad \lambda \succeq 0, \end{aligned}$$

which is an LP in the variables x and λ . Solving this single LP gives us the optimal x for the original robust LP.

- (d) We define x_{nom} and x_{rob} to be the nominal and robust solutions respectively. The numerical results are given in the table below.

MATLAB version:	x_{nom}	x_{rob}
$c_{\text{nom}}^T x$	1.50	1.94
$f(x)$	4.07	2.31

Python version:	x_{nom}	x_{rob}
$c_{\text{nom}}^T x$	2.11	2.52
$f(x)$	7.22	3.17

Julia version:	x_{nom}	x_{rob}
$c_{\text{nom}}^T x$	1.53	2.39
$f(x)$	4.32	2.99

The MATLAB code below formulates and solve the robust diet problem.

```

rand('seed',0);
n=10;m=30;
A = rand(m,n);
b = rand(m,1);
c_nom = 1+1*rand(n,1); % nominal c values

F = [eye(n);-eye(n);ones(1,n)/n;-ones(1,n)/n];
g = [1.25*c_nom;-0.75*c_nom;1.1*sum(c_nom)/n;-0.9*sum(c_nom)/n];
k = length(g);

% robust LP
cvx_begin
variables x_rob(n) lambda(k)
minimize(lambda'*g)
A*x_rob>=b
lambda>=0
F'*lambda==x_rob
cvx_end

% nominal cost of x_rob
c_nom'*x_rob

% nominal LP
cvx_begin
variables x_nom(n)
minimize(c_nom'*x_nom)
A*x_nom>=b
cvx_end

% worst case cost of x_nom
cvx_begin
variables c_wc(n)
maximize(c_wc'*x_nom)
F*c_wc<=g
cvx_end

```

The Python code below formulates and solve the robust diet problem.

```

import cvxpy as cvx
import numpy as np

np.random.seed(10)
(m, n) = (30, 10)
A = np.random.rand(m, n); A = np.asmatrix(A)
b = np.random.rand(m, 1); b = np.asmatrix(b)
c_nom = np.ones((n, 1)) + np.random.rand(n, 1); c_nom = np.asmatrix(c_nom)

```

```

#Solution begins
#-----
F = np.r_[np.eye(n), -np.eye(n), np.ones((1, n))/n, -np.ones((1, n))/n]
g = np.r_[1.25*c_nom, -0.75*c_nom, 1.1*sum(c_nom)/n, -0.9*sum(c_nom)/n]

lam = cvx.Variable(g.size)
x = cvx.Variable(n)

const = [A*x>=b, F.T*lam==x, lam>=0]
cvx.Problem(cvx.Minimize(lam.T*g), const).solve()
x_rob = x.value

const = [A*x>=b]
cvx.Problem(cvx.Minimize(c_nom.T*x), const).solve()
x_nom = x.value

print 'For x_nom and x_rob'
print 'nominal costs are %.2f, and %.2f' % (c_nom.T*x_nom, c_nom.T*x_rob)

c = cvx.Variable(n)
f_x_nom = cvx.Problem(cvx.Maximize(c.T*x_nom), [F*c<=g]).solve()
f_x_rob = cvx.Problem(cvx.Maximize(c.T*x_rob), [F*c<=g]).solve()

print 'worst-case costs are %.2f, and %.2f' % (f_x_nom, f_x_rob)

```

The Julia code below formulates and solve the robust diet problem.

```

srand(10);
n = 10;
m = 30;
A = rand(m, n);
b = rand(m, 1);
c_nom = 1 + rand(n, 1);

F = [eye(n); -eye(n); ones(1, n)/n; -ones(1, n)/n];
g = [1.25*c_nom; -0.75*c_nom; 1.1*sum(c_nom)/n; -0.9*sum(c_nom)/n];
k = length(g);

using Convex, SCS

#Robust LP
x = Variable(n);
lambda = Variable(k);
problem1 = minimize(lambda'*g, A*x >= b, lambda >= 0, F'*lambda - x ==0)
solve!(problem1, SCSSolver(verbose=0));
x_rob = x.value;

```

```

    println("Nominal cost of x_rob")
    println(c_nom'*x_rob)

    #Nominal LP
    x = Variable(n);
    problem2 = minimize(c_nom'*x, A*x >= b);
    solve!(problem2, SCSSolver(verbose=0));
    x_nom = x.value;
    println("Nominal cost of x_nom")
    println(c_nom'*x_nom)

    #Worst-case cost for x_rob
    c = Variable(n);
    constraint = F*c <= g;
    problem3 = maximize(x_rob'*c, F*c <= g);
    solve!(problem3, SCSSolver(verbose=0));
    f_c_rob = problem3.optval;
    println("Worst-case cost of x_rob:")
    println(f_c_rob)

    #Worst-case cost for x_nom
    problem4 = maximize(x_nom'*c, F*c <= g);
    solve!(problem4, SCSSolver(verbose=0));
    f_c_nom = problem4.optval;
    println("Worst-case cost of x_nom:")
    println(f_c_nom)

```

4.22 *Diagonal scaling with prescribed column and row sums.* Let A be an $n \times n$ matrix with positive entries, and let c and d be positive n -vectors that satisfy $\mathbf{1}^T c = \mathbf{1}^T d = 1$. Consider the geometric program

$$\begin{aligned} & \text{minimize} && x^T A y \\ & \text{subject to} && \prod_{i=1}^n x_i^{c_i} = 1 \\ & && \prod_{j=1}^n y_j^{d_j} = 1, \end{aligned}$$

with variables $x, y \in \mathbf{R}^n$ (and implicit constraints $x \succ 0, y \succ 0$). Write this geometric program in convex form and derive the optimality conditions. Show that if x and y are optimal, then the matrix

$$B = \frac{1}{x^T A y} \operatorname{diag}(x) A \operatorname{diag}(y)$$

satisfies $B\mathbf{1} = c$ and $B^T \mathbf{1} = d$.

Solution. We make a change of variables $u_i = \log x_i$, $v_j = \log y_j$ and define $\alpha_{ij} = \log A_{ij}$. The

GP in convex form is

$$\begin{aligned} & \text{minimize} && \log \left(\sum_{i=1}^n \sum_{j=1}^n e^{\alpha_{ij} + u_i + v_j} \right) \\ & \text{subject to} && c^T u = 0 \\ & && d^T v = 0. \end{aligned}$$

The optimality conditions are $c^T u = d^T v = 0$ and

$$\frac{e^{u_i} \sum_{j=1}^n e^{\alpha_{ij}} e^{v_j}}{\sum_{k=1}^n \sum_{l=1}^n e^{\alpha_{kl} + u_k + v_l}} \lambda c_i, \quad i = 1, \dots, n, \quad \frac{e^{v_j} \sum_{i=1}^n e^{\alpha_{ij}} e^{u_i}}{\sum_{k=1}^n \sum_{l=1}^n e^{\alpha_{kl} + u_k + v_l}} = \gamma d_j, \quad j = 1, \dots, n.$$

In the original variables $x_i = e^{u_i}$, $y_i = e^{v_i}$, this is

$$\frac{1}{x^T A y} \mathbf{diag}(x) A y = \lambda c, \quad \frac{1}{x^T A y} \mathbf{diag}(y) A^T x = \gamma d.$$

Taking the inner product with $\mathbf{1}$ shows $\lambda = \gamma = 1$.

This result can be found in the following paper: A. W. Marshall and I. Olkin, Scaling of matrices to achieve specified row and column sums. *Numerische Mathematik* 12, 83-90 (1968).

4.23 *A theorem due to Schoenberg.* Suppose m balls in \mathbf{R}^n , with centers a_i and radii r_i , have a nonempty intersection. We define y to be a point in the intersection, so

$$\|y - a_i\|_2 \leq r_i, \quad i = 1, \dots, m. \quad (27)$$

Suppose we move the centers to new positions b_i in such a way that the distances between the centers do not increase:

$$\|b_i - b_j\|_2 \leq \|a_i - a_j\|_2, \quad i, j = 1, \dots, m. \quad (28)$$

We will prove that the intersection of the translated balls is nonempty, *i.e.*, there exists a point x with $\|x - b_i\|_2 \leq r_i$, $i = 1, \dots, m$. To show this we prove that the optimal value of

$$\begin{aligned} & \text{minimize} && t \\ & \text{subject to} && \|x - b_i\|_2^2 \leq r_i^2 + t, \quad i = 1, \dots, m, \end{aligned} \quad (29)$$

with variables $x \in \mathbf{R}^n$ and $t \in \mathbf{R}$, is less than or equal to zero.

(a) Show that (28) implies that

$$t - (x - b_i)^T (x - b_j) \leq -(y - a_i)^T (y - a_j) \quad \text{for } i, j \in I,$$

if (x, t) is feasible in (29), and $I \subseteq \{1, \dots, m\}$ is the set of active constraints at x, t .

(b) Suppose x, t are optimal in (29) and that $\lambda_1, \dots, \lambda_m$ are optimal dual variables. Use the optimality conditions for (29) and the inequality in part a to show that

$$t = t - \left\| \sum_{i=1}^m \lambda_i (x - b_i) \right\|_2^2 \leq - \left\| \sum_{i=1}^m \lambda_i (y - a_i) \right\|_2^2.$$

Solution.

(a) Follows from

$$\begin{aligned} 2t + r_i^2 + r_j^2 - 2(x - b_i)^T(x - b_j) &= \|x - b_i\|_2^2 + \|x - b_j\|_2^2 - 2(x - b_i)^T(x - b_j) \\ &\leq \|y - a_i\|_2^2 + \|y - a_j\|_2^2 - 2(y - a_i)^T(y - a_j) \\ &\leq r_i^2 + r_j^2 - 2(y - a_i)^T(y - a_j). \end{aligned}$$

The equality follows from the fact that constraints i and j are active. The first inequality is (28), and the second inequality is (27).

(b) Let I be the set of active constraints at the optimal x, t . The optimality conditions are: primal feasibility and

$$\sum_{i=1}^m \lambda_i = 1, \quad x = \sum_{i=1}^m \lambda_i b_i, \quad \lambda_i \geq 0, \quad \lambda_i = 0 \quad \text{for } i \notin I.$$

Therefore

$$\begin{aligned} t &= t - \left\| \sum_{i=1}^m \lambda_i (x - b_i) \right\|_2^2 \\ &= \sum_{i=1}^m \sum_{j=1}^m \lambda_i \lambda_j \left(t - (x - b_i)^T (x - b_j) \right) \\ &= \sum_{i \in I} \sum_{j \in I} \lambda_i \lambda_j \left(t - (x - b_i)^T (x - b_j) \right) \\ &\leq - \sum_{i \in I} \sum_{j \in I} \lambda_i \lambda_j (y - a_i)^T (y - a_j) \\ &= - \sum_{i=1}^m \sum_{j=1}^m \lambda_i \lambda_j (y - a_i)^T (y - a_j) \\ &= - \left\| \sum_{i=1}^m \lambda_i (y - a_i) \right\|_2^2. \end{aligned}$$

Line 1 follows from $x = \sum_i \lambda_i b_i$. Line 2 follows from $\sum_i \lambda_i = 1$. Lines 3 and 5 follow from $\lambda_i = 0$ for $i \notin I$. Line 4 follows from part 1 and $\lambda_i \geq 0$.

The result appears in the following paper: I.J. Schoenberg, On a theorem of Kirzbraun and Valentine, *The American Mathematical Monthly* 60, 620-622 (1953).

5 Approximation and fitting

- 5.1 Three measures of the spread of a group of numbers.** For $x \in \mathbf{R}^n$, we define three functions that measure the spread or width of the set of its elements (or coefficients). The first function is the *spread*, defined as

$$\phi_{\text{sprd}}(x) = \max_{i=1,\dots,n} x_i - \min_{i=1,\dots,n} x_i.$$

This is the width of the smallest interval that contains all the elements of x .

The second function is the *standard deviation*, defined as

$$\phi_{\text{stdev}}(x) = \left(\frac{1}{n} \sum_{i=1}^n x_i^2 - \left(\frac{1}{n} \sum_{i=1}^n x_i \right)^2 \right)^{1/2}.$$

This is the statistical standard deviation of a random variable that takes the values x_1, \dots, x_n , each with probability $1/n$.

The third function is the average absolute deviation from the median of the values:

$$\phi_{\text{aamd}}(x) = (1/n) \sum_{i=1}^n |x_i - \text{med}(x)|,$$

where $\text{med}(x)$ denotes the median of the components of x , defined as follows. If $n = 2k - 1$ is odd, then the median is defined as the value of middle entry when the components are sorted, *i.e.*, $\text{med}(x) = x_{[k]}$, the k th largest element among the values x_1, \dots, x_n . If $n = 2k$ is even, we define the median as the average of the two middle values, *i.e.*, $\text{med}(x) = (x_{[k]} + x_{[k+1]})/2$.

Each of these functions measures the spread of the values of the entries of x ; for example, each function is zero if and only if all components of x are equal, and each function is unaffected if a constant is added to each component of x .

Which of these three functions is convex? For each one, either show that it is convex, or give a counterexample showing it is not convex. By a counterexample, we mean a specific x and y such that Jensen's inequality fails, *i.e.*, $\phi((x + y)/2) > (\phi(x) + \phi(y))/2$.

Solution. The first one is straightforward. The maximum of x_i is convex, and the minimum is concave. The difference, which is $\phi_{\text{sprd}}(x)$, is therefore convex.

The second one is also convex. The standard deviation is the Euclidean norm of a linear function of x ,

$$\phi_{\text{stdev}}(x) = (1/\sqrt{n}) \|(I - (1/n)\mathbf{1}\mathbf{1}^T)x\|_2,$$

and so is convex.

The third function is also convex. The snappiest proof we know goes like this. Consider the function defined as

$$\phi(x) = \inf_t \|x - t\mathbf{1}\|_1.$$

Since $\|x - t\mathbf{1}\|_1$ is convex in x, t , it follows that ϕ (which is obtained by minimizing over t) is convex in x . The median $t = \text{med}(x)$ minimizes $\|x - t\mathbf{1}\|_1$. (When n is even, any number between the two middle ones is also a minimizer.) Using this value of t , we see that

$$\phi_{\text{aamd}}(x) = (1/n)\phi(x),$$

and so is convex.

By the way, the same proof works for the other two functions. For $p = \infty$, the t that minimizes $\|x - t\mathbf{1}\|_\infty$ is $t = (\max_i x_i + \min_i x_i)/2$, and we have

$$\phi_{\text{sprd}}(x) = 2\phi(x).$$

For $p = 2$, the t that minimizes $\|x - t\mathbf{1}\|_2$ is the mean, $t = (1/n) \sum_i x_i$. Then we have

$$\phi_{\text{stdev}}(x) = (1/\sqrt{n})\phi(x).$$

We mention another nice proof of convexity of $\phi_{\text{aamd}}(x)$. Suppose that $n = 2k - 1$ is odd. We use $x_{[1]}$ to denote the largest element of x , $x_{[2]}$ the second largest, and so on. The median of x is given by $x_{[k]}$. Our function can be expressed as

$$\phi_{\text{sprd}}(x) = \sum_{i=1}^{k-1} (x_{[i]} - x_{[k]}) + \sum_{i=k+1}^n (x_{[k]} - x_{[i]}) = \sum_{i=1}^{k-1} x_{[i]} - \sum_{i=k+1}^n x_{[i]}.$$

But we know that for any r , the function

$$\sum_{i=1}^r x_{[i]},$$

the sum of the r largest entries of x , is convex. The sum of the r smallest elements can be expressed as

$$\sum_{i=n-r+1}^n x_{[i]} = \sum_{i=1}^n x_{[i]} - \sum_{i=1}^{n-r} x_{[i]},$$

and so is concave. It follows that $\phi_{\text{sprd}}(x)$ is the difference of a convex and concave function, and so is convex. The same type of argument works when n is even.

And, here is yet another proof of convexity of ϕ_{sprd} . Let's take n even for simplicity. Consider each vector c with all entries in c either 1 or -1 , with an equal number of 1s and -1 s. (There are $\binom{n}{n/2}$ such vectors.) The linear function $c^T x$ gives the difference between the sum of some half of the entries of x and the sum of the other half. Our function $\phi_{\text{sprd}}(x)$ is the maximum over all such linear functions, and therefore is convex.

5.2 Minimax rational fit to the exponential. (See exercise 6.9 of *Convex Optimization*.) We consider the specific problem instance with data

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

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

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

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

Find optimal values of a_0 , a_1 , a_2 , b_1 , b_2 , and give the optimal objective value, computed to an accuracy of 0.001. Plot the data and the optimal rational function fit on the same plot. On a different plot, give the fitting error, *i.e.*, $f(t_i) - y_i$.

Hint. To check if a feasibility problem is feasible, in Matlab, you can use `strcmp(cvx_status, 'Solved')` after `cvx_end`. In Python, use `problem.status == 'optimal'`. In Julia, use `problem.status == :Optimal`. In Julia, make sure to use the ECOS solver.

Solution. The objective function (and therefore also the problem) is not convex, but it is quasi-convex. We have $\max_{i=1,\dots,k} |f(t_i) - y_i| \leq \gamma$ if and only if

$$\left| \frac{a_0 + a_1 t_i + a_2 t_i^2}{1 + b_1 t_i + b_2 t_i^2} - y_i \right| \leq \gamma, \quad i = 1, \dots, k.$$

This is equivalent to (since the denominator is positive)

$$|a_0 + a_1 t_i + a_2 t_i^2 - y_i(1 + b_1 t_i + b_2 t_i^2)| \leq \gamma(1 + b_1 t_i + b_2 t_i^2), \quad i = 1, \dots, k,$$

which is a set of $2k$ linear inequalities in the variables a and b (for fixed γ). In particular, this shows the objective is quasiconvex. (In fact, it is a generalized linear fractional function.)

To solve the problem we can use a bisection method, solving an LP feasibility problem at each step. At each step we select some value of γ and solve the feasibility problem

$$\begin{aligned} & \text{find } a, b \\ & \text{subject to } |a_0 + a_1 t_i + a_2 t_i^2 - y_i(1 + b_1 t_i + b_2 t_i^2)| \leq \gamma(1 + b_1 t_i + b_2 t_i^2), \quad i = 1, \dots, k, \end{aligned}$$

with variables a and b . (Note that as long as $\gamma > 0$, the condition that the denominator is positive is enforced automatically.) This can be turned into the LP feasibility problem

$$\begin{aligned} & \text{find } a, b \\ & \text{subject to } \begin{aligned} a_0 + a_1 t_i + a_2 t_i^2 - y_i(1 + b_1 t_i + b_2 t_i^2) &\leq \gamma(1 + b_1 t_i + b_2 t_i^2), \quad i = 1, \dots, k \\ a_0 + a_1 t_i + a_2 t_i^2 - y_i(1 + b_1 t_i + b_2 t_i^2) &\geq -\gamma(1 + b_1 t_i + b_2 t_i^2), \quad i = 1, \dots, k. \end{aligned} \end{aligned}$$

The following Matlab code solves the problem for the particular problem instance.

```

k=201;
t=(-3:6/(k-1):3)';
y=exp(t);

Tpowers=[ones(k,1) t t.^2];

u=exp(3); l=0; % initial upper and lower bounds
bisection_tol=1e-3; % bisection tolerance

while u-l>= bisection_tol
    gamma=(l+u)/2;
    cvx_begin % solve the feasibility problem
        cvx_quiet(true);
    
```

```

variable a(3);
variable b(2);
subject to
    abs(Tpowers*a-y.*Tpowers*[1;b])) <= gamma*Tpowers*[1;b];
cvx_end

if strcmp(cvx_status,'Solved')
    u=gamma;
    a_opt=a;
    b_opt=b;
    objval_opt=gamma;
else
    l=gamma;
end
end

y_fit=Tpowers*a_opt./(Tpowers*[1;b_opt]);

figure(1);
plot(t,y,'b', t,y_fit,'r+');
xlabel('t');
ylabel('y');

figure(2);
plot(t, y_fit-y);
xlabel('t');
ylabel('err');

```

The following Python code solves the problem for the particular problem instance.

```

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

k = 201
t = -3 + 6 * np.arange(k) / (k - 1)
y = np.exp(t)

Tpowers = np.vstack((np.ones(k), t, t**2)).T

a = cvx.Variable(3)
b = cvx.Variable(2)
gamma = cvx.Parameter(sign='positive')
lhs = cvx.abs(Tpowers * a - (y[:, np.newaxis] * Tpowers) * cvx.vstack(1, b))
rhs = gamma * Tpowers * cvx.vstack(1, b)
problem = cvx.Problem(cvx.Minimize(0), [lhs <= rhs])

```

```

l, u = 0, np.exp(3) # initial upper and lower bounds
bisection_tol = 1e-3 # bisection tolerance
while u - l >= bisection_tol:
    gamma.value = (l + u) / 2
    # solve the feasibility problem
    problem.solve()
    if problem.status == 'optimal':
        u = gamma.value
        a_opt = a.value
        b_opt = b.value
        objval_opt = gamma.value
    else:
        l = gamma.value

y_fit = (Tpowers * a_opt / (Tpowers * np.vstack((1, b_opt)))).A1

plt.figure()
plt.plot(t, y, 'b', t, y_fit, 'r+')
plt.xlabel('t')
plt.ylabel('y')
plt.show()

plt.figure()
plt.plot(t, y_fit - y)
plt.xlabel('t')
plt.ylabel('err')
plt.show()

```

The following Julia code solves the problem for the particular problem instance.

```

using Convex, ECOS, PyPlot

set_default_solver(ECOSSolver(verbose=false))

k = 201
t = -3:6/(k-1):3
y = exp(t)

Tpowers = [ones(k) t t.^2];

a = Variable(3)
b = Variable(2)

l, u = 0, exp(3) # initial upper and lower bounds
bisection_tol = 1e-3 # bisection tolerance

```

```

a_opt, b_opt, objval_opt = zeros(3), zeros(2), 0
while u - l >= bisection_tol
    gamma = (l + u) / 2
    lhs = abs(Tpowers * a - (y .* Tpowers) * [1; b])
    rhs = gamma * Tpowers * [1; b]
    problem = minimize(0, [lhs <= rhs])
    # solve the feasibility problem
    solve!(problem)
    if problem.status == :Optimal
        u = gamma
        a_opt = evaluate(a)
        b_opt = evaluate(b)
        objval_opt = gamma
    else
        l = gamma
    end
end

y_fit = (Tpowers * a_opt ./ (Tpowers * [1; b_opt]))

figure()
plot(t, y, "b", t, y_fit, "r+")
xlabel("t")
ylabel("y")
show()

figure()
plot(t, y_fit - y)
xlabel("t")
ylabel("err")
show()

```

The optimal values are

$$a_0 = 1.0099, \quad a_1 = 0.6117, \quad a_2 = 0.1134, \quad b_1 = -0.4147, \quad b_2 = 0.0485,$$

and the optimal objective value is 0.0233. We plot the fit and the error in Figure 5 and 6.

5.3 Approximation with trigonometric polynomials. Suppose $y : \mathbf{R} \rightarrow \mathbf{R}$ is a 2π -periodic function. We will approximate y with the trigonometric polynomial

$$f(t) = \sum_{k=0}^K a_k \cos(kt) + \sum_{k=1}^K b_k \sin(kt).$$

We consider two approximations: one that minimizes the L_2 -norm of the error, defined as

$$\|f - y\|_2 = \left(\int_{-\pi}^{\pi} (f(t) - y(t))^2 dt \right)^{1/2},$$

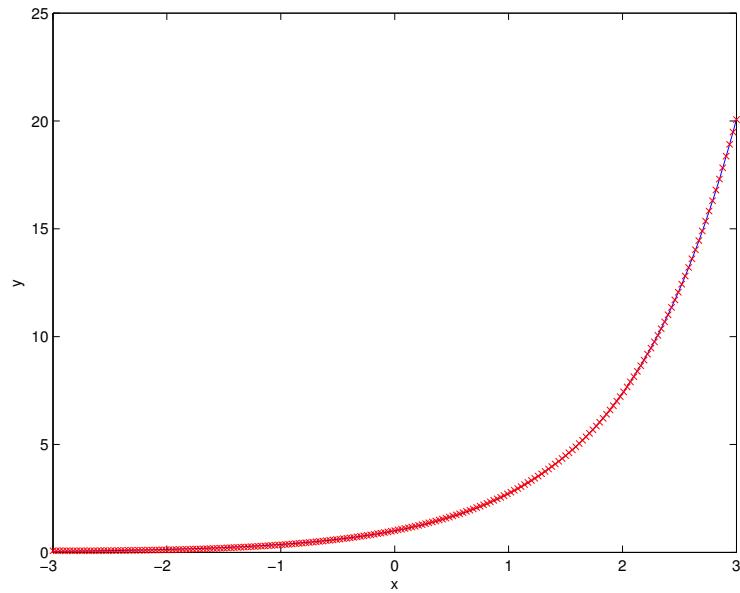


Figure 5: Chebyshev fit with rational function. The line represents the data and the crosses the fitted points.

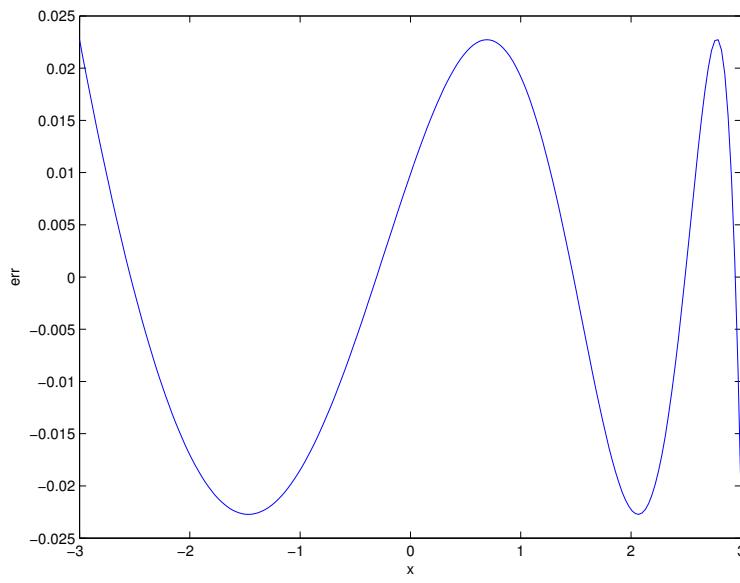


Figure 6: Fitting error for Chebyshev fit of exponential with rational function.

and one that minimizes the L_1 -norm of the error, defined as

$$\|f - y\|_1 = \int_{-\pi}^{\pi} |f(t) - y(t)| dt.$$

The L_2 approximation is of course given by the (truncated) Fourier expansion of y .

To find an L_1 approximation, we discretize t at $2N$ points,

$$t_i = -\pi + i\pi/N, \quad i = 1, \dots, 2N,$$

and approximate the L_1 norm as

$$\|f - y\|_1 \approx (\pi/N) \sum_{i=1}^{2N} |f(t_i) - y(t_i)|.$$

(A standard rule of thumb is to take N at least 10 times larger than K .) The L_1 approximation (or really, an approximation of the L_1 approximation) can now be found using linear programming.

We consider a specific case, where y is a 2π -periodic square-wave, defined for $-\pi \leq t \leq \pi$ as

$$y(t) = \begin{cases} 1 & |t| \leq \pi/2 \\ 0 & \text{otherwise.} \end{cases}$$

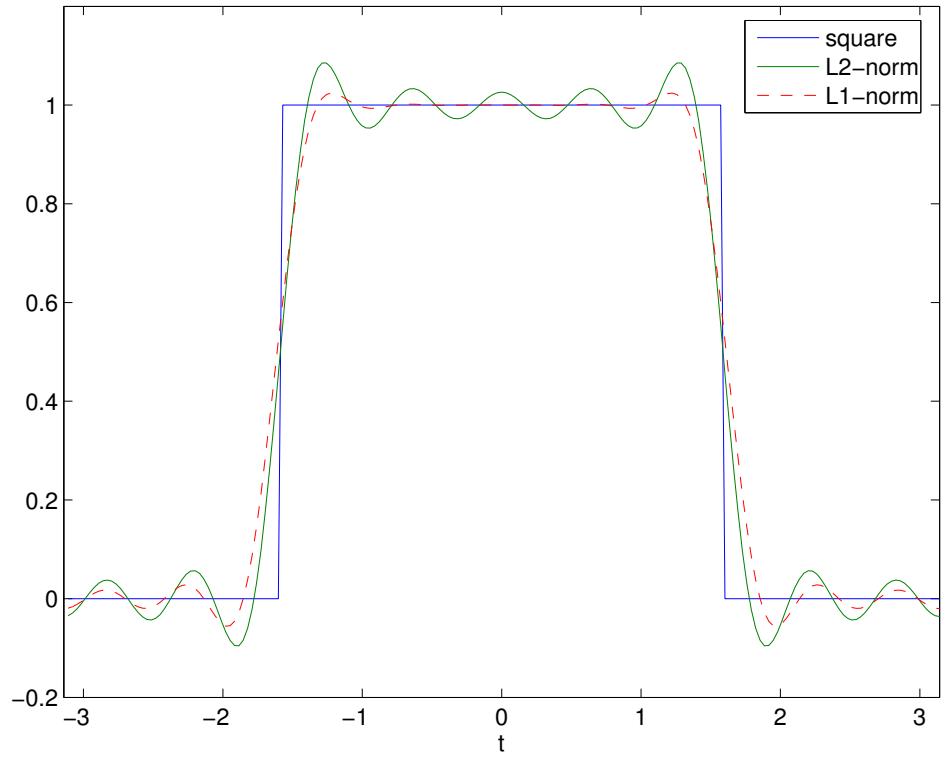
(The graph of y over a few cycles explains the name ‘square-wave’.)

Find the optimal L_2 approximation and (discretized) L_1 optimal approximation for $K = 10$. You can find the L_2 optimal approximation analytically, or by solving a least-squares problem associated with the discretized version of the problem. Since y is even, you can take the sine coefficients in your approximations to be zero. Show y and the two approximations on a single plot.

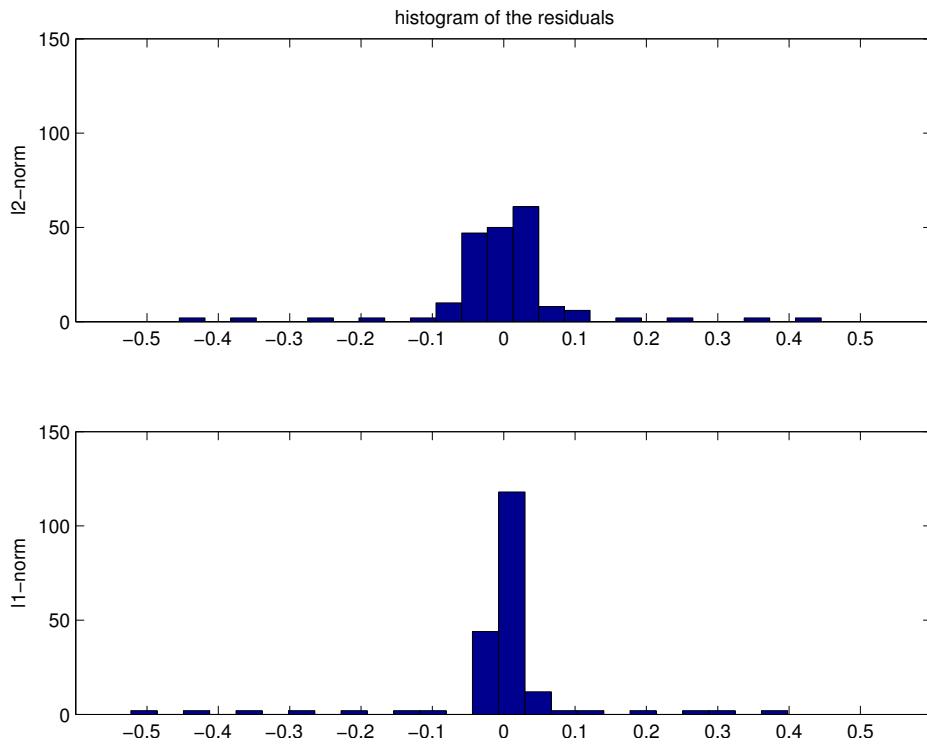
In addition, plot a histogram of the residuals (*i.e.*, the numbers $f(t_i) - y(t_i)$) for the two approximations. Use the same horizontal axis range, so the two residual distributions can easily be compared. (Matlab command `hist` might be helpful here.) Make some brief comments about what you see.

Solution.

The optimal approximations are shown below. The L_2 optimal approximation has the familiar Gibb’s ringing near the discontinuities in y , but the L_1 optimal approximation has much less pronounced oscillation near the discontinuity in y . The L_1 optimal approximation has very small error, except near the discontinuity.



The residual distributions of the two approximations are shown below. As expected, the L_2 approximation has more small residuals, and fewer large residuals, compared to the L_1 approximation. The L_1 approximation has a large number of zero and very small errors, and a few large ones.



The Matlab code is as follows.

```
% square wave approximations
close all; clear all;
K = 10;
N = 10*K;
t = -pi+pi/N:pi/N:pi;
y = (abs(t) <= pi/2)';

% A = [f0 f1 f2 f3 ... f9 f10];
A = ones(2*N,1);
for k = 1:K
A = [A cos(k*t)'];
end

% L2 approximation
c = A\y
y_2 = A*c;

% L1 approximation
cvx_begin
    variable x(K+1)
    minimize(norm(A*x-y,1))
cvx_end
```

```

y_1 = A*x;

% plot of the square-wave and optimal fits
figure;
plot(t,y,t,y_2,t,y_1,'--');
title('approximations')
legend('square','l2norm','l1norm');
axis([-pi pi -0.2 1.2])

% distribution of residual magnitudes
figure;
subplot(2,1,1), hist(y - y_2, 25), axis([- .6 .6 0 150]);
title('l2 and l1 residual distributions'), ylabel('l2-norm')
subplot(2,1,2), hist(y - y_1, 25), axis([- .6 .6 0 150]);

```

5.4 Penalty function approximation. We consider the approximation problem

$$\text{minimize } \phi(Ax - b)$$

where $A \in \mathbf{R}^{m \times n}$ and $b \in \mathbf{R}^m$, the variable is $x \in \mathbf{R}^n$, and $\phi : \mathbf{R}^m \rightarrow \mathbf{R}$ is a convex penalty function that measures the quality of the approximation $Ax \approx b$. We will consider the following choices of penalty function:

(a) *Euclidean norm.*

$$\phi(y) = \|y\|_2 = \left(\sum_{k=1}^m y_k^2 \right)^{1/2}.$$

(b) *ℓ_1 -norm.*

$$\phi(y) = \|y\|_1 = \sum_{k=1}^m |y_k|.$$

(c) *Sum of the largest $m/2$ absolute values.*

$$\phi(y) = \sum_{k=1}^{\lfloor m/2 \rfloor} |y|_{[k]}$$

where $|y|_{[1]}, |y|_{[2]}, |y|_{[3]}, \dots$, denote the absolute values of the components of y sorted in decreasing order.

(d) *A piecewise-linear penalty.*

$$\phi(y) = \sum_{k=1}^m h(y_k), \quad h(u) = \begin{cases} 0 & |u| \leq 0.2 \\ |u| - 0.2 & 0.2 \leq |u| \leq 0.3 \\ 2|u| - 0.5 & |u| \geq 0.3. \end{cases}$$

(e) *Huber penalty.*

$$\phi(y) = \sum_{k=1}^m h(y_k), \quad h(u) = \begin{cases} u^2 & |u| \leq M \\ M(2|u| - M) & |u| \geq M \end{cases}$$

with $M = 0.2$.

(f) *Log-barrier penalty.*

$$\phi(y) = \sum_{k=1}^m h(y_k), \quad h(u) = -\log(1 - u^2), \quad \text{dom } h = \{u \mid |u| < 1\}.$$

Here is the problem. Generate data A and b as follows:

```
m = 200;
n = 100;
A = randn(m,n);
b = randn(m,1);
b = b/(1.01*max(abs(b)));
```

(The normalization of b ensures that the domain of $\phi(Ax - b)$ is nonempty if we use the log-barrier penalty.) To compare the results, plot a histogram of the vector of residuals $y = Ax - b$, for each of the solutions x , using the Matlab command

```
hist(A*x-b,m/2);
```

Some additional hints and remarks for the individual problems:

- (a) This problem can be solved using least-squares (`x=A\b`).
- (b) Use the CVX function `norm(y,1)`.
- (c) Use the CVX function `norm_largest()`.
- (d) Use CVX, with the overloaded `max()`, `abs()`, and `sum()` functions.
- (e) Use the CVX function `huber()`.
- (f) The current version of CVX handles the logarithm using an iterative procedure, which is slow and not entirely reliable. However, you can reformulate this problem as

$$\text{maximize } (\prod_{k=1}^m ((1 - (Ax - b)_k)(1 + (Ax - b)_k)))^{1/2m},$$

and use the CVX function `geo_mean()`.

Solution.

We show that each of the functions ϕ is convex and, hence, $\phi(Ax - b)$ is convex.

- Parts 1,2. Any norm $\|\cdot\|$ is a convex function.
- Part 3.. See exercise 3.19 (b).
- Parts 4,5,6. Making a plot of each of these functions h clearly shows that they are convex.
For the piecewise-linear penalty, we can also note that

$$h(u) = \max\{0, |u| - 0.2, 2|u| - 0.5\},$$

so it is the pointwise maximum of convex functions. In part 6, h is the Euclidean norm of the vector $(\sqrt{\rho}, u)$.

- Part 7. We can express h as the sum of two convex functions:

$$h(u) = -\log(1+u) - \log(1-u).$$

The Matlab code is as follows.

```
m = 200; n = 100
A = randn(m,n);
b = randn(m,1);
b = b/(1.01*max(abs(b)));

% Part 1. L2
x1 = A\b;

% Part 2. L1
cvx_begin
    variable x2(n)
    minimize(norm(A*x2-b,1))
cvx_end

% Part 3. Sum of largest abs. values
cvx_begin
    variable x3(n)
    minimize(norm_largest(A*x3-b,floor(m/2)))
cvx_end

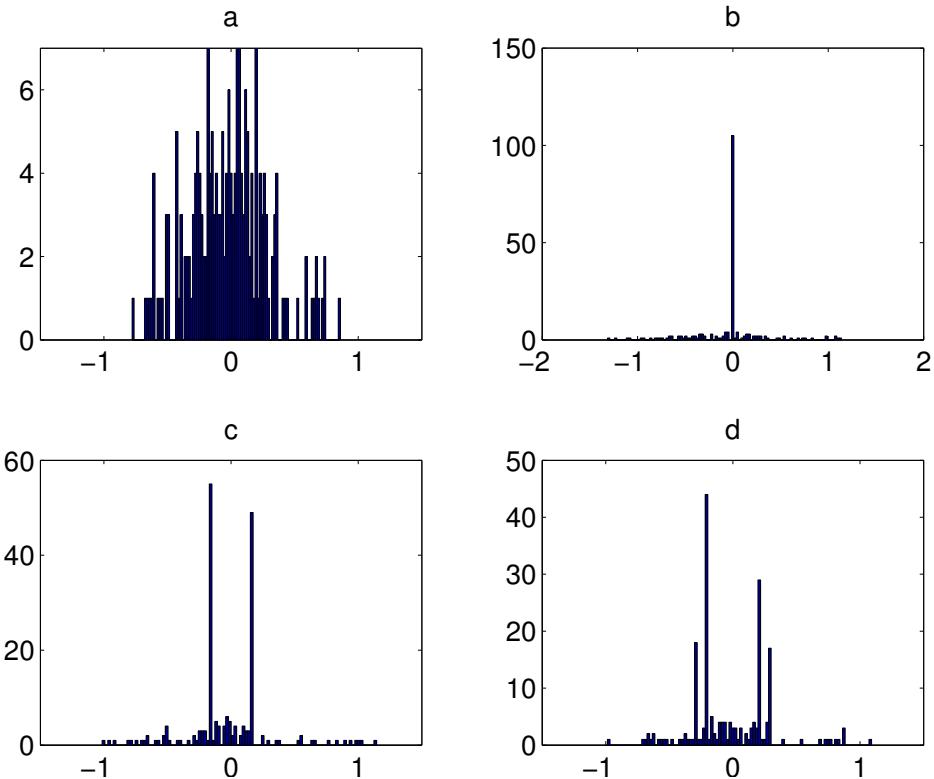
% Part 4. PWL
cvx_begin
    variable x4(n)
    minimize(sum(max([zeros(m,1), abs(A*x4-b)-0.2, 2*abs(A*x4-b)-0.5])))
cvx_end

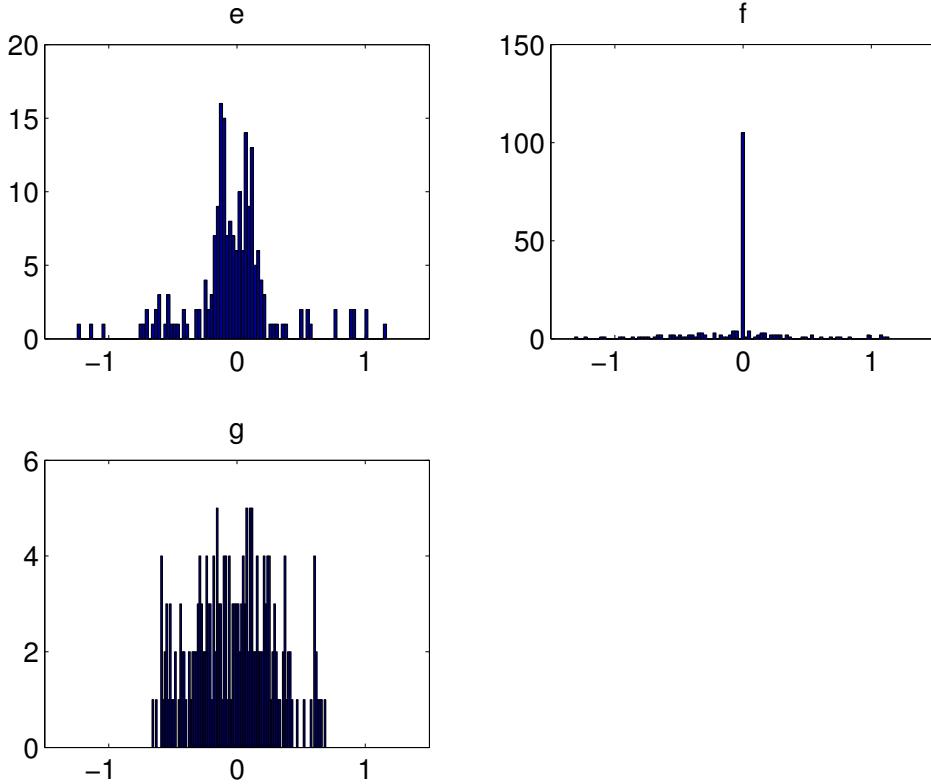
% Part 5. huber
disp('huber')
cvx_begin
    variable x5(n)
    minimize sum(huber(A*x5 - b, .2))
cvx_end

% Part 6. Smoothed l1
disp('sqrt')
cvx_begin
    variable x6(n)
    minimize(sl1(A*x6-b, 1e-8))
cvx_end
```

```
% Part 7. entropy
cvx_begin
    variable x7(n)
    maximize(geomean([1-(A*x7-b); 1+(A*x7-b)]))
cvx_end
```

The residual distributions of an example problem are shown in the figure.





5.5 $\ell_{1.5}$ 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 de-noising, 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:

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

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.

- (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

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

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 i th residual. We can, of course, drop the factor $3/2$.

- (b) We can write an equivalent problem

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

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

$$\begin{bmatrix} u & v \\ v & w \end{bmatrix} \succeq 0$$

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

$$\begin{bmatrix} \sqrt{s_i} & s_i \\ s_i & t \end{bmatrix} \succeq 0.$$

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 $0 \preceq y \preceq \sqrt{s}$:

$$\begin{bmatrix} y_i & s_i \\ s_i & t_i \end{bmatrix} \succeq 0, \quad \begin{bmatrix} s_i & y_i \\ y_i & 1 \end{bmatrix} \succeq 0.$$

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:

$$\begin{aligned} & \text{minimize} && \mathbf{1}^T t \\ & \text{subject to} && -s_i \preceq a_i^T x - b_i \preceq s_i, \quad i = 1, \dots, m \\ & && \begin{bmatrix} y_i & s_i \\ s_i & t_i \end{bmatrix} \succeq 0, \quad \begin{bmatrix} s_i & y_i \\ y_i & 1 \end{bmatrix} \succeq 0, \quad i = 1, \dots, m, \end{aligned}$$

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

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

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

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

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
x12=A\b;

%l1 solution
cvx_begin
    variables x11(n);
    minimize(norm(A*x11-b,1));
cvx_end

r=A*x-b; %residuals
r12=A*x12-b;
r11=A*x11-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(r12)
axis([-2.5 2.5 0 50])
xlabel('r2')

```

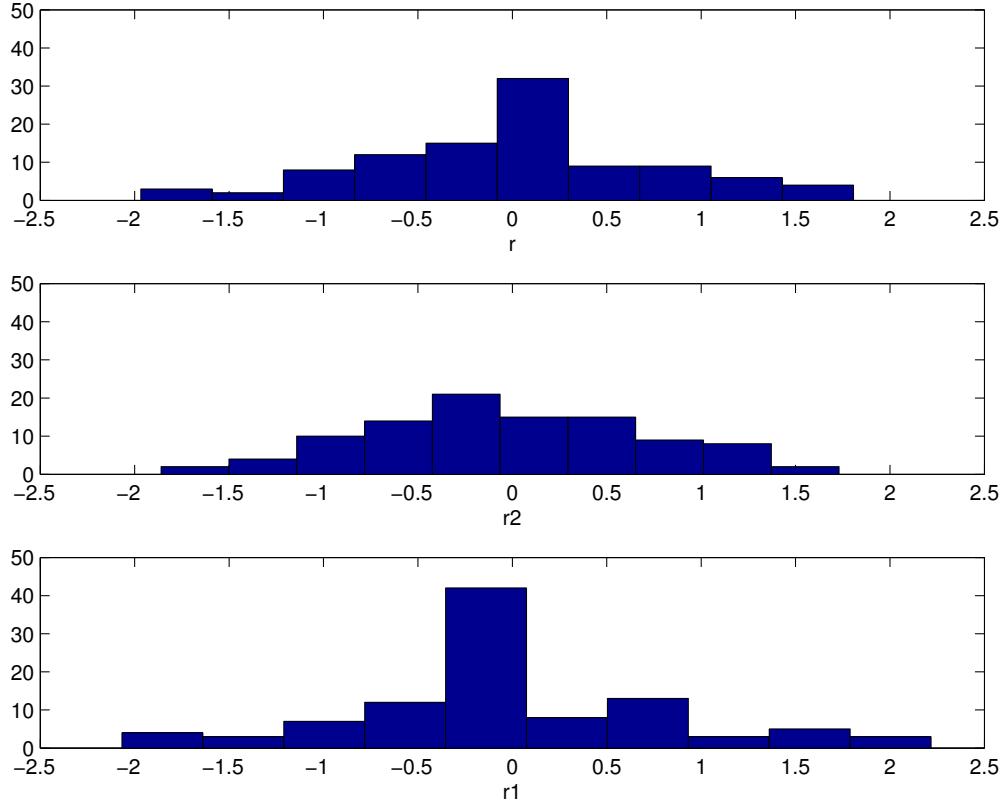


Figure 7: Histogram of the residuals for $\ell_{1.5}$ -norm, ℓ_2 -norm, and ℓ_1 -norm

```

subplot(3,1,3)
hist(r11)
axis([-2.5 2.5 0 50])
xlabel('r1')

%solution using SDP
cvx_begin sdp
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 7 shows the histograms of the residuals for the three norms.

5.6 Total variation image interpolation. A grayscale image is represented as an $m \times n$ matrix of

intensities U^{orig} . You are given the values U_{ij}^{orig} , for $(i, j) \in \mathcal{K}$, where $\mathcal{K} \subset \{1, \dots, m\} \times \{1, \dots, n\}$. Your job is to *interpolate* the image, by guessing the missing values. The reconstructed image will be represented by $U \in \mathbf{R}^{m \times n}$, where U satisfies the interpolation conditions $U_{ij} = U_{ij}^{\text{orig}}$ for $(i, j) \in \mathcal{K}$.

The reconstruction is found by minimizing a roughness measure subject to the interpolation conditions. One common roughness measure is the ℓ_2 variation (squared),

$$\sum_{i=2}^m \sum_{j=1}^n (U_{ij} - U_{i-1,j})^2 + \sum_{i=1}^m \sum_{j=2}^n (U_{ij} - U_{i,j-1})^2.$$

Another method minimizes instead the *total variation*,

$$\sum_{i=2}^m \sum_{j=1}^n |U_{ij} - U_{i-1,j}| + \sum_{i=1}^m \sum_{j=2}^n |U_{ij} - U_{i,j-1}|.$$

Evidently both methods lead to convex optimization problems.

Carry out ℓ_2 and total variation interpolation on the problem instance with data given in `tv_img_interp.m`. This will define `m`, `n`, and matrices `Uorig` and `Known`. The matrix `Known` is $m \times n$, with (i, j) entry one if $(i, j) \in \mathcal{K}$, and zero otherwise. The mfile also has skeleton plotting code. (We give you the entire original image so you can compare your reconstruction to the original; obviously your solution cannot access U_{ij}^{orig} for $(i, j) \notin \mathcal{K}$.)

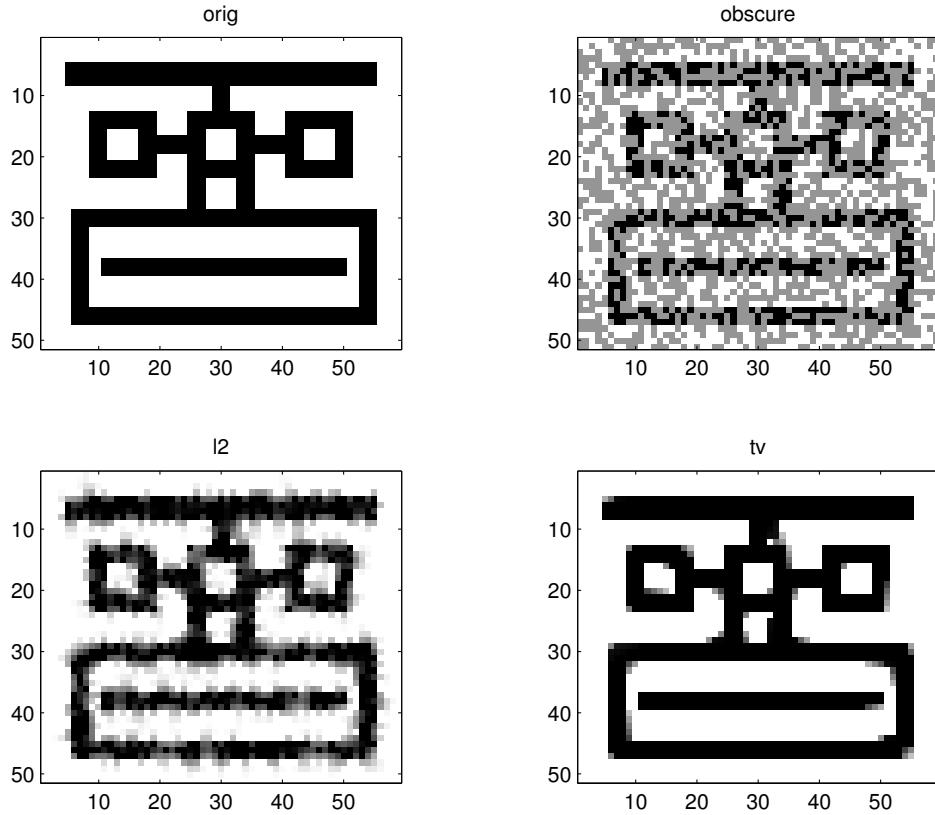
Solution. The code for the interpolation is very simple. For ℓ_2 interpolation, the code is the following.

```
cvx_begin
    variable Ul2(m, n);
    Ul2(Known) == Uorig(Known); % Fix known pixel values.
    Ux = Ul2(1:end,2:end) - Ul2(1:end,1:end-1); % x (horiz) differences
    Uy = Ul2(2:end,1:end) - Ul2(1:end-1,1:end); % y (vert) differences
    minimize(norm([Ux(:); Uy(:)], 2)); % l2 roughness measure
cvx_end
```

For total variation interpolation, we use the following code.

```
cvx_begin
    variable Utv(m, n);
    Utv(Known) == Uorig(Known); % Fix known pixel values.
    Ux = Utv(1:end,2:end) - Utv(1:end,1:end-1); % x (horiz) differences
    Uy = Utv(2:end,1:end) - Utv(1:end-1,1:end); % y (vert) differences
    minimize(norm([Ux(:); Uy(:)], 1)); % tv roughness measure
cvx_end
```

We get the following images



5.7 Piecewise-linear fitting. In many applications some function in the model is not given by a formula, but instead as tabulated data. The tabulated data could come from empirical measurements, historical data, numerically evaluating some complex expression or solving some problem, for a set of values of the argument. For use in a convex optimization model, we then have to fit these data with a convex function that is compatible with the solver or other system that we use. In this problem we explore a very simple problem of this general type.

Suppose we are given the data (x_i, y_i) , $i = 1, \dots, m$, with $x_i, y_i \in \mathbf{R}$. We will assume that x_i are sorted, i.e., $x_1 < x_2 < \dots < x_m$. Let $a_0 < a_1 < a_2 < \dots < a_K$ be a set of fixed knot points, with $a_0 \leq x_1$ and $a_K \geq x_m$. Explain how to find the convex piecewise linear function f , defined over $[a_0, a_K]$, with knot points a_i , that minimizes the least-squares fitting criterion

$$\sum_{i=1}^m (f(x_i) - y_i)^2.$$

You must explain what the variables are and how they parametrize f , and how you ensure convexity of f .

Hints. One method to solve this problem is based on the Lagrange basis, f_0, \dots, f_K , which are the piecewise linear functions that satisfy

$$f_j(a_i) = \delta_{ij}, \quad i, j = 0, \dots, K.$$

Another method is based on defining $f(x) = \alpha_i x + \beta_i$, for $x \in (a_{i-1}, a_i]$. You then have to add conditions on the parameters α_i and β_i to ensure that f is continuous and convex.

Apply your method to the data in the file `pwl_fit_data.m`, which contains data with $x_j \in [0, 1]$. Find the best affine fit (which corresponds to $a = (0, 1)$), and the best piecewise-linear convex function fit for 1, 2, and 3 internal knot points, evenly spaced in $[0, 1]$. (For example, for 3 internal knot points we have $a_0 = 0$, $a_1 = 0.25$, $a_2 = 0.50$, $a_3 = 0.75$, $a_4 = 1$.) Give the least-squares fitting cost for each one. Plot the data and the piecewise-linear fits found. Express each function in the form

$$f(x) = \max_{i=1,\dots,K} (\alpha_i x + \beta_i).$$

(In this form the function is easily incorporated into an optimization problem.)

Solution. Following the hint, we will use the Lagrange basis functions f_0, \dots, f_K . These can be expressed as

$$\begin{aligned} f_0(x) &= \left(\frac{a_1 - x}{a_1 - a_0} \right)_+, \\ f_i(x) &= \left(\min \left(\frac{x - a_{i-1}}{a_i - a_{i-1}}, \frac{a_{i+1} - x}{a_i - a_{i+1}} \right) \right)_+, \quad i = 1, \dots, K-1, \end{aligned}$$

and

$$f_K(x) = \left(\frac{x - a_{K-1}}{a_K - a_{K-1}} \right)_+.$$

The function f can be parametrized as

$$f(x) = \sum_{i=0}^K z_i f_i(x),$$

where $z_i = f(a_i)$, $i = 0, \dots, K$. We will use $z = (z_0, \dots, z_K)$ to parametrize f . The least-squares fitting criterion is then

$$J = \sum_{i=1}^m (f(x_i) - y_i)^2 = \|Fz - y\|_2^2,$$

where $F \in \mathbf{R}^{m \times (K+1)}$ is the matrix

$$F_{ij} = f_j(x_i), \quad i = 1, \dots, m, \quad j = 0, \dots, K.$$

(We index the columns of F from 0 to K here.)

We must add the constraint that f is convex. This is the same as the condition that the slopes of the segments are nondecreasing, *i.e.*,

$$\frac{z_{i+1} - z_i}{a_{i+1} - a_i} \geq \frac{z_i - z_{i-1}}{a_i - a_{i-1}}, \quad i = 1, \dots, K-1.$$

This is a set of linear inequalities in z . Thus, the best PWL convex fit can be found by solving the QP

$$\begin{array}{ll} \text{minimize} & \|Fz - y\|_2^2 \\ \text{subject to} & \frac{z_{i+1} - z_i}{a_{i+1} - a_i} \geq \frac{z_i - z_{i-1}}{a_i - a_{i-1}}, \quad i = 1, \dots, K-1. \end{array}$$

The following code solves this problem for the data in `pwl_fit_data`.

```

figure
plot(x,y,'k:', 'linewidth',2)
hold on

% Single line
p = [x ones(100,1)]\y;
alpha = p(1)
beta = p(2)
plot(x,alpha*x+beta,'b','linewidth',2)
mse = norm(alpha*x+beta-y)^2

for K = 2:4
    % Generate Lagrange basis
    a = (0:(1/K):1)';
    F = max((a(2)-x)/(a(2)-a(1)),0);
    for k = 2:K
        a_1 = a(k-1);
        a_2 = a(k);
        a_3 = a(k+1);
        f = max(0,min((x-a_1)/(a_2-a_1),(a_3-x)/(a_3-a_2)));
        F = [F f];
    end
    f = max(0,(x-a(K))/(a(K+1)-a(K)));
    F = [F f];

    % Solve problem
    cvx_begin
        variable z(K+1)
        minimize(norm(F*z-y))
        subject to
            (z(3:end)-z(2:end-1))./(a(3:end)-a(2:end-1)) >=...
            (z(2:end-1)-z(1:end-2))./(a(2:end-1)-a(1:end-2))
    cvx_end

    % Calculate alpha and beta
    alpha = (z(2:end)-z(1:end-1))./(a(2:end)-a(1:end-1))
    beta = z(2:end)-alpha(1:end).*a(2:end)

    % Plot solution
    y2 = F*z;
    mse = norm(y2-y)^2
    if K==2
        plot(x,y2,'r','linewidth',2)
    elseif K==3
        plot(x,y2,'g','linewidth',2)
    end
end

```

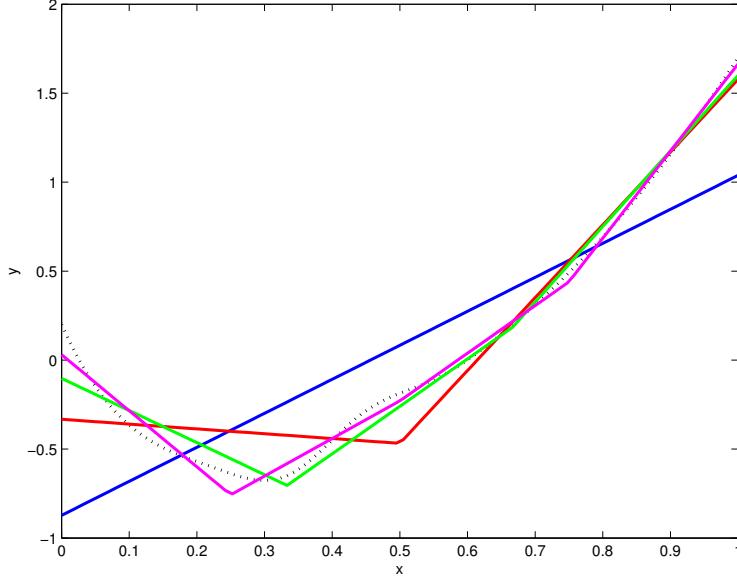


Figure 8: Piecewise-linear approximations for $K = 1, 2, 3, 4$

```

else
    plot(x,y2,'m','linewidth',2)
end

end
xlabel('x')
ylabel('y')

```

This generates figure 8. We can see that the approximation improves as K increases. The following table shows the result of this approximation.

K	$\alpha_1, \dots, \alpha_K$	β_1, \dots, β_K	J
1	1.91	-0.87	12.73
2	-0.27, 4.09	-0.33, -2.51	2.62
3	-1.80, 2.67, 4.25	-0.10, -1.59, -2.65	0.60
4	-3.15, 2.11, 2.68, 4.90	0.03, -1.29, -1.57, -3.23	0.22

There is another way to solve this problem. We are looking for a piecewise linear function. If we have at least one internal knot ($K \geq 2$), the function should satisfy the two following constraints:

- convexity: $\alpha_1 \leq \alpha_2 \leq \dots \leq \alpha_K$
- continuity: $\alpha_i a_i + \beta_i = \alpha_{i+1} a_i + \beta_{i+1}$, $i = 1, \dots, K - 1$.

Therefore, the optimization problem is

$$\begin{aligned} & \text{minimize} && (\sum_{i=1}^m f(x_i) - y_i)^2 \\ & \text{subject to} && \alpha_i \leq \alpha_{i+1}, \quad i = 1, \dots, K-1 \\ & && \alpha_i a_i + \beta_i = \alpha_{i+1} a_i + \beta_{i+1}, \quad i = 1, \dots, K-1 \end{aligned}$$

Reformulating the problem by representing $f(x_i)$ in matrix form, we get

$$\begin{aligned} & \text{minimize} && \|\text{diag}(x)F\alpha + F\beta - y\|^2 \\ & \text{subject to} && \alpha_i \leq \alpha_{i+1}, \quad i = 1, \dots, K-1 \\ & && \alpha_i a_i + \beta_i = \alpha_{i+1} a_i + \beta_{i+1}, \quad i = 1, \dots, K-1 \end{aligned}$$

where the variables are $\alpha \in \mathbf{R}^K$ and $\beta \in \mathbf{R}^K$, and problem data are $x \in \mathbf{R}^m$, $y \in \mathbf{R}^m$ and

$$F_{ij} = \begin{cases} 1 & \text{if } a_{j-1} = x_i, \quad j = 1 \\ 1 & \text{if } a_{j-1} < x_i \leq a_j \\ 0 & \text{otherwise.} \end{cases}$$

```
% another approach for PWL fitting problem
clear all;
pwl_fit_data;
m = length(x);
xp = 0:0.001:1; % for fine-grained pwl function plot
mp = length(xp);
yp = [];

for K = 1:4 % internal knot 1,2,3

    a = [0:1/K:1]'; % a_0,...,a_K
    % matrix for sum f(x_i)
    F = sparse(1:m,max(1,ceil(x*K)),1,m,K);

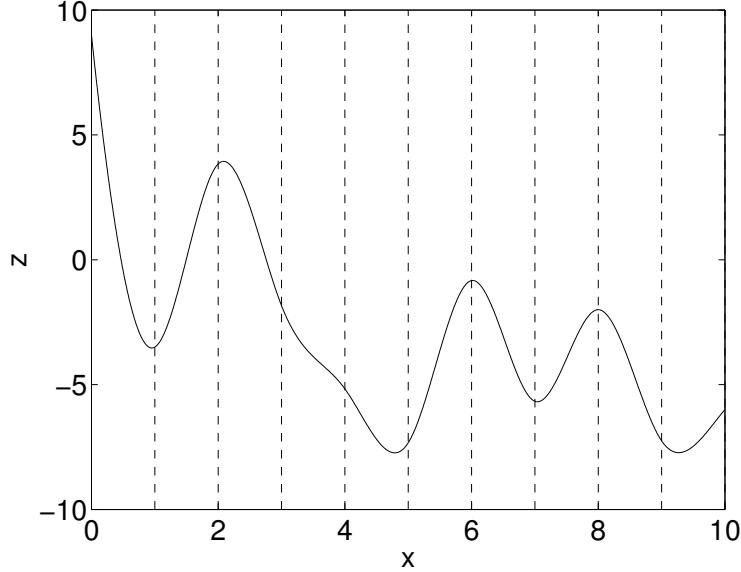
    % solve problem
    cvx_begin
        variables alpha(K) beta(K)
        minimize( norm(diag(x)*F*alpha+F*beta-y) )
        subject to
        if (K>=2)
            alpha(1:K-1).*a(2:K)+beta(1:K-1) == alpha(2:K).*a(2:K)+beta(2:K)
            a(1:K-1) <= a(2:K)
        end
        cvx_end

        fp = sparse(1:mp,max(1,ceil(xp*K)),1,mp,K);
        yp = [yp diag(xp)*fp*alpha+fp*beta];
    end
    plot(x,y,'b.',xp,yp);
end
```

5.8 Least-squares fitting with convex splines. A *cubic spline* (or *fourth-order spline*) with breakpoints $\alpha_0, \alpha_1, \dots, \alpha_M$ (that satisfy $\alpha_0 < \alpha_1 < \dots < \alpha_M$) is a piecewise-polynomial function with the following properties:

- the function is a cubic polynomial on each interval $[\alpha_i, \alpha_{i+1}]$
- the function values, and the first and second derivatives are continuous on the interval (α_0, α_M) .

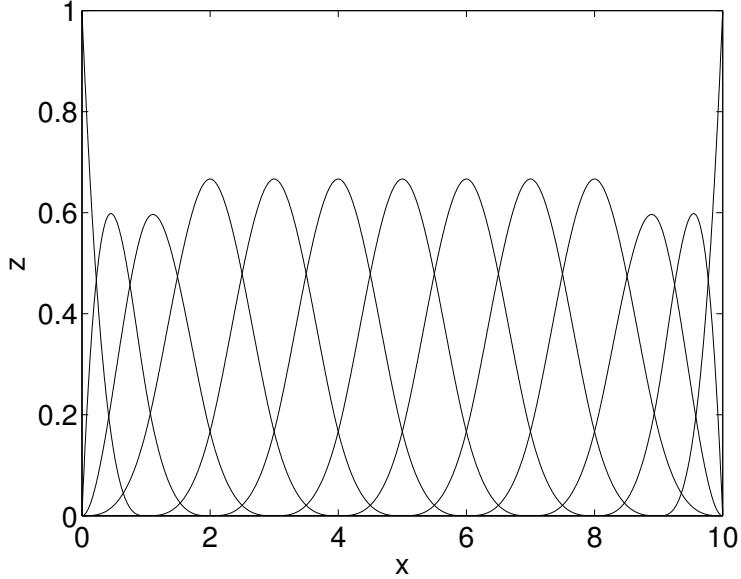
The figure shows an example of a cubic spline $f(t)$ with $M = 10$ segments and breakpoints $\alpha_0 = 0, \alpha_1 = 1, \dots, \alpha_{10} = 10$.



In approximation problems with splines it is convenient to parametrize a spline as a linear combination of basis functions, called *B-splines*. The precise definition of B-splines is not important for our purposes; it is sufficient to know that every cubic spline can be written as a linear combination of $M + 3$ cubic B-splines $g_k(t)$, *i.e.*, in the form

$$f(t) = x_1 g_1(t) + \dots + x_{M+3} g_{M+3}(t) = x^T g(t),$$

and that there exist efficient algorithms for computing $g(t) = (g_1(t), \dots, g_{M+3}(t))$. The next figure shows the 13 B-splines for the breakpoints $0, 1, \dots, 10$.



In this exercise we study the problem of fitting a cubic spline to a set of data points, subject to the constraint that the spline is a convex function. Specifically, the breakpoints $\alpha_0, \dots, \alpha_M$ are fixed, and we are given N data points (t_k, y_k) with $t_k \in [\alpha_0, \alpha_M]$. We are asked to find the convex cubic spline $f(t)$ that minimizes the least-squares criterion

$$\sum_{k=1}^N (f(t_k) - y_k)^2.$$

We will use B-splines to parametrize f , so the variables in the problem are the coefficients x in $f(t) = x^T g(t)$. The problem can then be written as

$$\begin{aligned} & \text{minimize} && \sum_{k=1}^N (x^T g(t_k) - y_k)^2 \\ & \text{subject to} && x^T g(t) \text{ is convex in } t \text{ on } [\alpha_0, \alpha_M]. \end{aligned} \tag{30}$$

- (a) Express problem (30) as a convex optimization problem of the form

$$\begin{aligned} & \text{minimize} && \|Ax - b\|_2^2 \\ & \text{subject to} && Gx \preceq h. \end{aligned}$$

- (b) Use CVX to solve a specific instance of the optimization problem in part (a). As in the figures above, we take $M = 10$ and $\alpha_0 = 0, \alpha_1 = 1, \dots, \alpha_{10} = 10$.

Download the Matlab files `spline_data.m` and `bsplines.m`. The first m-file is used to generate the problem data. The command `[t, y] = spline_data` will generate two vectors t, y of length $N = 51$, with the data points t_k, y_k .

The second function can be used to compute the B-splines, and their first and second derivatives, at any given point $u \in [0, 10]$. The command `[g, gp, gpp] = bsplines(u)` returns three vectors of length 13 with elements $g_k(u)$, $g'_k(u)$, and $g''_k(u)$. (The right derivatives are returned for $u = 0$, and the left derivatives for $u = 10$.)

Solve the convex spline fitting problem (30) for this example, and plot the optimal spline.

Solution.

(a) The objective function is

$$\sum_{k=1}^N (x^T g(t_k) - y_k)^2 = \|Ax - b\|_2^2$$

with

$$A = \begin{bmatrix} g(t_1)^T \\ g(t_2)^T \\ \vdots \\ g(t_N)^T \end{bmatrix}, \quad b = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}.$$

To handle the convexity constraint we note that f'' is piecewise linear in t . Therefore $f''(t) \geq 0$ for all $g \in (\alpha_0, \alpha_M)$ if and only if $f''(\alpha_k) = x^T g''(\alpha_k) \geq 0$ for $k = 0, \dots, M$. This gives a set of linear inequalities $Gx \leq 0$ with

$$G = - \begin{bmatrix} g''(\alpha_0)^T \\ g''(\alpha_1)^T \\ \vdots \\ g''(\alpha_M)^T \end{bmatrix}.$$

```
(b) [u, y] = spline_data;
N = length(u);

A = zeros(N, 13);
b = y;
for k = 1:N
    [g, gp, gpp] = bsplines(u(k));
    A(k,:) = g';
end;

% Solution without convexity constraint
xls = A\b;

% Solution with convexity constraint
G = zeros(11, 13);
for k = 1:11
    [g, gp, gpp] = bsplines(k-1);
    G(k,:)= gpp';
end;
cvx_begin
    variable x(13);
    minimize( norm(A*x - b) );
    subject to
        G*x >= 0;
cvx_end
```

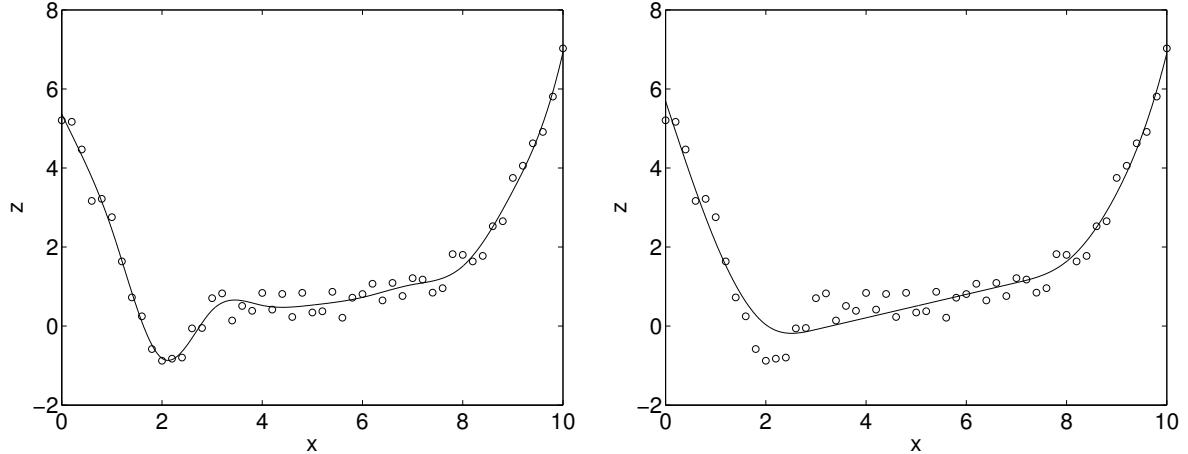
```

% plot solution without convexity constraints
figure(1)
npts = 1000;
t = linspace(0, 10, npts);
fls = zeros(1, npts);
for k = 1:npts
    [g, gp, gpp] = bsplines(t(k));
    fls(k) = xls' * g;
end;
plot(u, y,'o', t, fls, '-');

% plot solution with convexity constraints
figure(2)
f = zeros(1, npts);
for k = 1:npts
    [g, gp, gpp] = bsplines(t(k));
    f(k) = x' * g;
end;
plot(u, y,'o', t, f, '-');

```

The function on the right is the optimal convex spline. The function on the left is the the optimal spline without convexity constraint.



5.9 Robust least-squares with interval coefficient matrix. An *interval matrix* in $\mathbf{R}^{m \times n}$ is a matrix whose entries are intervals:

$$\mathcal{A} = \{A \in \mathbf{R}^{m \times n} \mid |A_{ij} - \bar{A}_{ij}| \leq R_{ij}, i = 1, \dots, m, j = 1, \dots, n\}.$$

The matrix $\bar{A} \in \mathbf{R}^{m \times n}$ is called the *nominal value* or *center value*, and $R \in \mathbf{R}^{m \times n}$, which is elementwise nonnegative, is called the *radius*.

The robust least-squares problem, with interval matrix, is

$$\text{minimize } \sup_{A \in \mathcal{A}} \|Ax - b\|_2,$$

with optimization variable $x \in \mathbf{R}^n$. The problem data are \mathcal{A} (*i.e.*, \bar{A} and R) and $b \in \mathbf{R}^m$. The objective, as a function of x , is called the *worst-case residual norm*. The robust least-squares problem is evidently a convex optimization problem.

- (a) Formulate the interval matrix robust least-squares problem as a standard optimization problem, *e.g.*, a QP, SOCP, or SDP. You can introduce new variables if needed. Your reformulation should have a number of variables and constraints that grows linearly with m and n , and not exponentially.
- (b) Consider the specific problem instance with $m = 4$, $n = 3$,

$$\mathcal{A} = \begin{bmatrix} 60 \pm 0.05 & 45 \pm 0.05 & -8 \pm 0.05 \\ 90 \pm 0.05 & 30 \pm 0.05 & -30 \pm 0.05 \\ 0 \pm 0.05 & -8 \pm 0.05 & -4 \pm 0.05 \\ 30 \pm 0.05 & 10 \pm 0.05 & -10 \pm 0.05 \end{bmatrix}, \quad b = \begin{bmatrix} -6 \\ -3 \\ 18 \\ -9 \end{bmatrix}.$$

(The first part of each entry in \mathcal{A} gives \bar{A}_{ij} ; the second gives R_{ij} , which are all 0.05 here.) Find the solution x_{ls} of the nominal problem (*i.e.*, minimize $\|\bar{A}x - b\|_2$), and robust least-squares solution x_{rls} . For each of these, find the nominal residual norm, and also the worst-case residual norm. Make sure the results make sense.

Solution.

- (a) The problem is equivalent to

$$\text{minimize } \sup_{A \in \mathcal{A}} \|Ax - b\|_2^2,$$

which can be reformulated as

$$\begin{aligned} & \text{minimize } y^T y \\ & \text{subject to } -y \preceq Ax - b \preceq y, \quad \text{for all } A \in \mathcal{A}. \end{aligned}$$

We have

$$\begin{aligned} \sup_{A \in \mathcal{A}} (Ax - b)_i &= \sum_{j=1}^n (\bar{A}_{ij} x_j + R_{ij} |x_j|) - b_i \\ \inf_{A \in \mathcal{A}} (Ax - b)_i &= \sum_{j=1}^n (\bar{A}_{ij} x_j - R_{ij} |x_j|) - b_i. \end{aligned}$$

We can therefore write the problem as

$$\begin{aligned} & \text{minimize } y^T y \\ & \text{subject to } \bar{A}x + R|x| - b \preceq y \\ & \quad \bar{A}x - R|x| - b \succeq -y \end{aligned}$$

where $|x| \in \mathbf{R}^n$ is the vector with elements $|x|_i = |x_i|$, or equivalently as the QP

$$\begin{aligned} & \text{minimize } y^T y \\ & \text{subject to } \bar{A}x + Rz - b \preceq y \\ & \quad \bar{A}x - Rz - b \succeq -y \\ & \quad -z \preceq x \preceq z. \end{aligned}$$

The variables are $x \in \mathbf{R}^n$, $y \in \mathbf{R}^m$, $z \in \mathbf{R}^n$.

The problem also has an alternative formulation: the trick is to find an alternative expression for the worst-case residual norm as a function of x :

$$\begin{aligned} f(x) &= \sup_{A \in \mathcal{A}} \|Ax - b\|_2^2 \\ &= \sum_{i=1}^m \sup_{\substack{|V_{ij}| \leq R_{ij} \\ j=1, \dots, n}} \left(\sum_{j=1}^n (\bar{A}_{ij} + V_{ij})x_j - b_i \right)^2. \end{aligned}$$

Assume x is fixed and define $r = \bar{A}x - b$. We work out the worst-case value of the uncertain parameters V , i.e., the values that maximize

$$\left(\sum_{j=1}^n (\bar{A}_{ij} + V_{ij})x_j - b_i \right)^2 = \left(r_i + \sum_{j=1}^n V_{ij}x_j \right)^2$$

for $i = 1, \dots, m$. If $r_i \geq 0$, the worst-case values for V_{ij} are the values that make $\sum_i V_{ij}x_j$ positive and as large as possible, i.e., $V_{ij} = R_{ij}$ if $x_j \geq 0$ and $V_{ij} = -R_{ij}$ otherwise. If $r_i < 0$, the worst-case values for V_{ij} are the values that make $\sum_i V_{ij}x_j$ negative and as large as possible, i.e., $V_{ij} = R_{ij}$ if $x_j \leq 0$ and $V_{ij} = -R_{ij}$ otherwise. Therefore

$$\begin{aligned} f(x) &= \sum_{i=1}^m \left(|r_i| + \sum_{j=1}^n R_{ij}|x_j| \right)^2 \\ &= \| |\bar{A}x - b| + R|x| \|_2^2 \end{aligned}$$

if we interpret the absolute values of the vectors component-wise.

We can therefore write the problem as

$$\begin{array}{ll} \text{minimize} & y^T y \\ \text{subject to} & |\bar{A}x - b| + R|x| \preceq y. \end{array}$$

This can be further written as a QP

$$\begin{array}{ll} \text{minimize} & y^T y \\ \text{subject to} & \bar{A}x + Rz - b \preceq y \\ & \bar{A}x - Rz - b \succeq -y \\ & -z \preceq x \preceq z. \end{array}$$

The variables are $x \in \mathbf{R}^n$, $y \in \mathbf{R}^m$, $z \in \mathbf{R}^n$.

- (b) The results are given by the following:

Residual norms for the nominal problem when using LS solution:

7.5895

Residual norms for the nominal problem when using robust solution:

17.7106

```
Residual norms for the worst-case problem when using LS solution:  
26.7012
```

```
Residual norms for the worst-case problem when using robust solution:  
17.7940
```

The MATLAB script below computes the least-squares and the robust solutions and also computes, for each one, the nominal and the worst-case residual norms.

```
% input data  
A_bar = [ 60      45      -8; ...  
          90      30     -30; ...  
          0      -8      -4; ...  
         30      10     -10];  
d = .05;  
R = d*ones(4,3);  
b = [ -6; -3; 18; -9];  
  
% least-squares solution  
x_ls = A_bar\b;  
  
% robust least-squares solution  
cvx_begin  
    variables x(3) y(4) z(3)  
    minimize ( norm( y ) )  
    A_bar*x + R*z - b <= y  
    A_bar*x - R*z - b >= -y  
    x <= z  
    x + z >= 0  
cvx_end  
  
% computing nominal residual norms  
nom_res_ls = norm(A_bar*x_ls - b);  
nom_res_rls = norm(A_bar*x - b);  
  
% computing worst-case nominal norms  
r = A_bar*x_ls - b;  
Delta = zeros(4,3);  
for i=1:length(r)  
    if r(i) < 0  
        Delta(i,:) = -d*sign(x_ls');  
    else  
        Delta(i,:) = d*sign(x_ls');  
    end  
end  
wc_res_ls = norm(r + Delta*x_ls);
```

```

wc_res_rls = cvx_optval;

% display
disp('Residual norms for the nominal problem when using LS solution: ');
disp(nom_res_ls);
disp('Residual norms for the nominal problem when using robust solution: ');
disp(nom_res_rls);
disp('Residual norms for the worst-case problem when using LS solution: ');
disp(wc_res_ls);
disp('Residual norms for the worst-case problem when using robust solution: ');
disp(wc_res_rls);

```

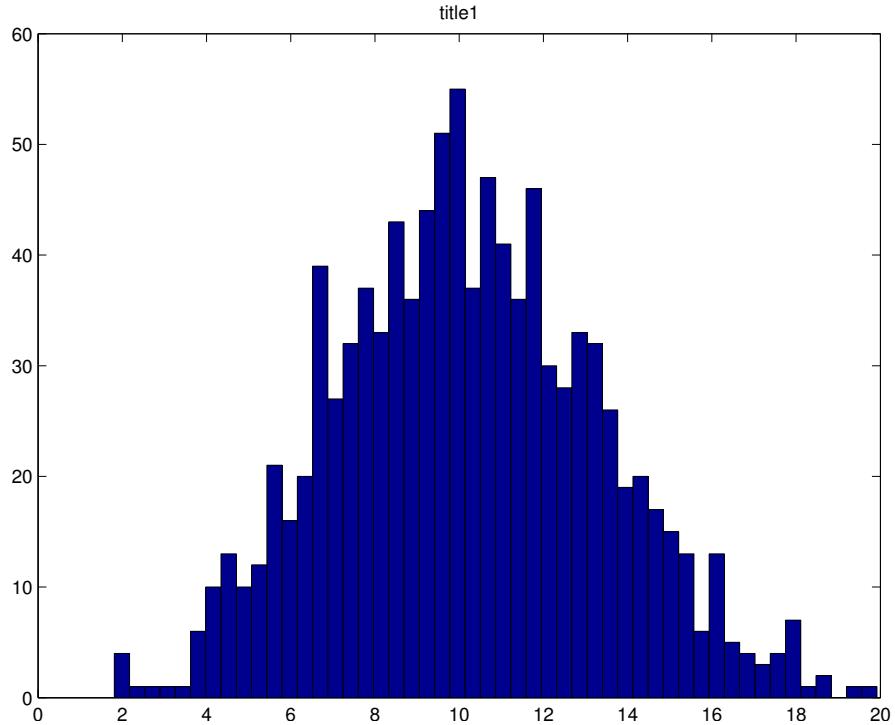
The robust least-square solution can also be found using the following script:

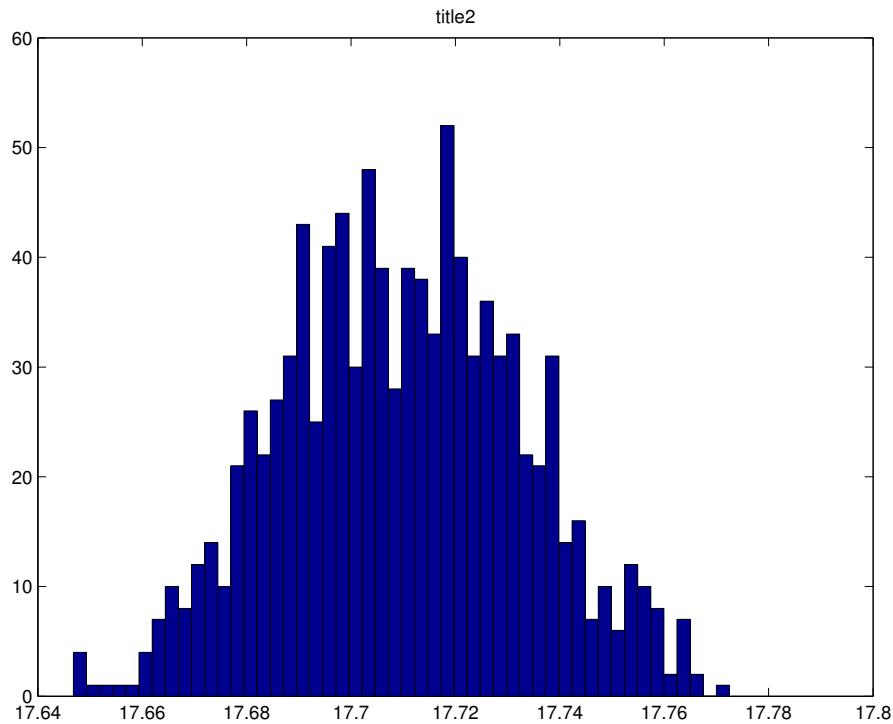
```

cvx_begin
    variables x(3) t(4)
    minimize ( norm ( t ) )
    abs(A_bar*x - b) + R*abs(x) <= t
cvx_end

```

We also generated, for fun, the following histograms showing the distribution of the residual norms for the case where $x = x_{\text{ls}}$ and $x = x_{\text{rls}}$. Those were obtained by creating 1000 instances of A by sampling A_{ij} uniformly between $\bar{A}_{ij} - R_{ij}$ and $\bar{A}_{ij} + R_{ij}$, and then evaluating the residual norm for each A and each of the 2 solutions.





The following MATLAB script generates these histograms:

```
% Monte-Carlo simulation
N = 1000;
res_ls = zeros(N,1);
res_rls = zeros(N,1);
for k=1:N
    Delta = d*(2*rand(4,3)-1);
    A = A_bar + Delta;
    res_ls(k) = norm(A*x_ls - b);
    res_rls(k) = norm(A*x - b);
end
figure;
hist(res_ls,50);
figure;
hist(res_rls,50);
```

In Python, the least-squares and the robust solutions can be found using the following code.

```
# generating matrices
import numpy as np
import cvxpy as cvx

A_bar = np.array(np.mat(
'60 45 -8;\ \
90 30 -30;\ \
```

```

0 -8 -4;\n
30 10 -10'))\n
d = .05;\n
R = d*np.ones((4,3))\n
b = np.array([[[-6],\n
                [-3],\n
                [18],\n
                [-9]]])\n\n
# least-squares solution\n
x_ls = np.linalg.lstsq(A_bar, b)[0]\n\n
# robust least-squares solution\n
x = cvx.Variable(3)\n
y = cvx.Variable(4)\n
z = cvx.Variable(3)\n
objective = cvx.Minimize(cvx.norm(y))\n
constraints = [A_bar*x + R*z - b <= y,\n
                A_bar*x - R*z - b >= -y,\n
                x <= z,\n
                x + z >= 0]\n
prob = cvx.Problem(objective, constraints)\n
result = prob.solve()\n\n
x_rls = x.value\n
# computing nominal residual norms\n
nom_res_ls = np.linalg.norm(np.dot(A_bar, x_ls) - b)\n
nom_res_rls = np.linalg.norm(np.dot(A_bar, x_rls) - b)\n
# computing worst-case nominal norms\n
r = np.dot(A_bar, x_ls) - b;\n
Delta = np.zeros((4,3));\n
for i in range(r.shape[0]):\n
    if r[i] < 0:\n
        Delta[i, :] = -d*np.sign(x_ls.T)\n
    else:\n
        Delta[i,:] = d*np.sign(x_ls.T)\n\n
wc_res_ls = np.linalg.norm(r + np.dot(Delta, x_ls))\n
wc_res_rls = result\n
# display\n
print "Residual norms for the nominal problem when using LS solution: "\n
print nom_res_ls\n
print "Residual norms for the nominal problem when using robust solution: "\n
print nom_res_rls\n
print "Residual norms for the worst-case problem when using LS solution: "\n

```

```

print wc_res_ls
print "Residual norms for the worst-case problem when using robust solution: "
print wc_res_rls

In Julia, the least-squares and the robust solutions can be found using the following code.

# generating matrices
close
A_bar = [ 60 45 -8;
          90 30 -30;
          0 -8 -4;
          30 10 -10];
d = .05;
R = d*ones(4,3);
b = [ -6; -3; 18; -9];

using Convex, SCS

# least-squares solution
x_ls = \(\mathbf{A}_{\text{bar}}, \mathbf{b}\);

# robust least-squares solution

x = Variable(3);
y = Variable(4);
z = Variable(3);
constraint = [\mathbf{A}_{\text{bar}}\mathbf{x} + R\mathbf{z} - \mathbf{b} \leq \mathbf{y}
              \mathbf{A}_{\text{bar}}\mathbf{x} - R\mathbf{z} - \mathbf{b} \geq -\mathbf{y}
              \mathbf{x} \leq \mathbf{z}
              \mathbf{x} + \mathbf{z} \geq 0];
problem = minimize(norm(y), constraint);
solve!(problem)

# computing nominal residual norms
x_rls = x.value;
nom_res_ls = norm(A_bar*x_ls - b);
nom_res_rls = norm(A_bar*x_rls - b);
# computing worst-case nominal norms
r = A_bar*x_ls - b;
Delta = zeros(4,3);
for i=1:length(r)
  if r[i] < 0
    Delta[i,:] = -d*sign(x_ls');
  else
    Delta[i,:] = d*sign(x_ls');
  end
end

```

```

end
end
wc_res_ls = norm(r + Delta*x_ls);
wc_res_rls = problem.optval;

# display
@printf("Nominal problem,      using LS solution: %e\n", nom_res_ls);
@printf("Nominal problem,      using robust solution: %e\n", nom_res_rls);
@printf("Worst-case problem, using LS solution: %e\n", wc_res_ls);
@printf("Worst-case problem, using robust solution: %e\n", wc_res_rls);

```

5.10 Identifying a sparse linear dynamical system. A linear dynamical system has the form

$$x(t+1) = Ax(t) + Bu(t) + w(t), \quad t = 1, \dots, T-1,$$

where $x(t) \in \mathbf{R}^n$ is the state, $u(t) \in \mathbf{R}^m$ is the input signal, and $w(t) \in \mathbf{R}^n$ is the process noise, at time t . We assume the process noises are IID $\mathcal{N}(0, W)$, where $W \succ 0$ is the covariance matrix. The matrix $A \in \mathbf{R}^{n \times n}$ is called the dynamics matrix or the state transition matrix, and the matrix $B \in \mathbf{R}^{n \times m}$ is called the input matrix.

You are given accurate measurements of the state and input signal, *i.e.*, $x(1), \dots, x(T)$, $u(1), \dots, u(T-1)$, and W is known. Your job is to find a state transition matrix \hat{A} and input matrix \hat{B} from these data, that are plausible, and in addition are sparse, *i.e.*, have many zero entries. (The sparser the better.)

By doing this, you are effectively estimating the structure of the dynamical system, *i.e.*, you are determining which components of $x(t)$ and $u(t)$ affect which components of $x(t+1)$. In some applications, this structure might be more interesting than the actual values of the (nonzero) coefficients in \hat{A} and \hat{B} .

By plausible, we mean that

$$\sum_{t=1}^{T-1} \left\| W^{-1/2} (x(t+1) - \hat{A}x(t) - \hat{B}u(t)) \right\|_2^2 \leq n(T-1) + 2\sqrt{2n(T-1)}.$$

(You can just take this as our definition of plausible. But to explain this choice, we note that when $\hat{A} = A$ and $\hat{B} = B$, the left-hand side is χ^2 , with $n(T-1)$ degrees of freedom, and so has mean $n(T-1)$ and standard deviation $\sqrt{2n(T-1)}$. Thus, the constraint above states that the LHS does not exceed the mean by more than 2 standard deviations.)

- (a) Describe a method for finding \hat{A} and \hat{B} , based on convex optimization.

We are looking for a *very simple* method, that involves solving *one* convex optimization problem. (There are many extensions of this basic method, that would improve the simple method, *i.e.*, yield sparser \hat{A} and \hat{B} that are still plausible. We're not asking you to describe or implement any of these.)

- (b) Carry out your method on the data found in `sparse_lds_data.m`. Give the values of \hat{A} and \hat{B} that you find, and verify that they are plausible.

In the data file, we give you the true values of A and B , so you can evaluate the performance of your method. (Needless to say, you are not allowed to use these values when forming \hat{A} and

$\hat{B}.$) Using these true values, give the number of false positives and false negatives in both \hat{A} and \hat{B} . A false positive in \hat{A} , for example, is an entry that is nonzero, while the corresponding entry in A is zero. A false negative is an entry of \hat{A} that is zero, while the corresponding entry of A is nonzero. To judge whether an entry of \hat{A} (or \hat{B}) is nonzero, you can use the test $|\hat{A}_{ij}| \geq 0.01$ (or $|\hat{B}_{ij}| \geq 0.01$).

Solution. The problem can be expressed as

$$\begin{aligned} & \text{minimize} && \mathbf{card}(\hat{A}) + \mathbf{card}(\hat{B}) \\ & \text{subject to} && \sum_{t=1}^{T-1} \left\| W^{-1/2} \left(x(t+1) - \hat{A}x(t) - \hat{B}u(t) \right) \right\|_2^2 \in n(T-1) \pm 2\sqrt{2n(T-1)}, \end{aligned}$$

where $\mathbf{card}(X)$ is the cardinality (the number of nonzero entries) of matrix X .

However, there are two problems with this: the objective is non-convex, and the lower bound $\sum_{t=1}^{T-1} \left\| W^{-1/2} \left(x(t+1) - \hat{A}x(t) - \hat{B}u(t) \right) \right\|_2^2 \geq n(T-1) - 2\sqrt{2n(T-1)}$ is not a convex constraint. The second problem is dealt with by noting that we can always increase the magnitudes of the implied errors by (for example) multiplying candidate \hat{A} and \hat{B} by a constant. Hence the lower bound can be neglected, without loss of generality.

Unfortunately the **card** function is less comprehensively dealt with. However, we can use the (standard) heuristic of instead minimizing the ℓ_1 norm of the entries of \hat{A} and \hat{B} . This gives the convex optimization problem

$$\begin{aligned} & \text{minimize} && \|\mathbf{vec}(\hat{A})\|_1 + \|\mathbf{vec}(\hat{B})\|_1 \\ & \text{subject to} && \sum_{t=1}^{T-1} \left\| W^{-1/2} \left(x(t+1) - \hat{A}x(t) - \hat{B}u(t) \right) \right\|_2^2 \leq n(T-1) + 2\sqrt{2n(T-1)}, \end{aligned}$$

where $\mathbf{vec}(X)$ represents the columns of X concatenated to make a single column vector. Roughly speaking, note that the constraint will always be tight: relaxing the requirement on the implied errors allows more freedom to reduce the ℓ_1 norm of \hat{A} and \hat{B} . Thus, in reality, we never need to worry about the lower bound from above as it will be always satisfied.

This problem is easily solved in CVX using the following code

```
% Load problem data.
sparse_lds_data;

fit_tol = sqrt(n*(T-1) + 2*sqrt(2*n*(T-1))); % fit tolerance.
cvx_begin
    variables Ahat(n,n) Bhat(n,m);
    minimize(sum(norms(Ahat, 1)) + sum(norms(Bhat, 1)))
    norm(inv(Whalf)*(xs(:,2:T) - Ahat*xs(:,1:T-1) ...
        - Bhat*u), 'fro') <= fit_tol;
cvx_end
disp(cvx_status)

% Check lower bound.
fit_tol_m = sqrt(n*(T-1) - 2*sqrt(2*n*(T-1)));
disp(norm(inv(Whalf)*(xs(:,2:T) ...
```

```

- Ahat*xs(:,1:T-1) - Bhat*us), 'fro') >= fit_tol_m);

% Round near-zero elements to zero.
Ahat = Ahat .* (abs(Ahat) >= 0.01)
Bhat = Bhat .* (abs(Bhat) >= 0.01)

% Display results.
disp(['false positives, Ahat: ' num2str(nnz((Ahat ~= 0) & (A == 0)))] )
disp(['false negatives, Ahat: ' num2str(nnz((Ahat == 0) & (A ~= 0)))] )
disp(['false positives, Bhat: ' num2str(nnz((Bhat ~= 0) & (B == 0)))] )
disp(['false negatives, Bhat: ' num2str(nnz((Bhat == 0) & (B ~= 0)))] )

```

With the given problem data, we get 1 false positive and 2 false negatives for \hat{A} , and no false positives and 1 false negative for \hat{B} . The matrix estimates are

$$\hat{A} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1.1483 & -0.0899 & 0.1375 & -0.0108 & 0 & 0 \\ 0 & 0 & 0 & 0.9329 & 0 & 0 & 0.2868 & 0 \\ 0 & 0.2055 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -0.0190 & 0.9461 & 0 & 0.8697 \\ 0 & 0 & 0 & 0 & 0.2066 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

and

$$\hat{B} = \begin{bmatrix} -1.4717 & 0 & 0 & 0 \\ 0 & 0 & 0 & -0.2832 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1.6363 & -0.0456 & 0 & 0 \\ 0 & 1.4117 & 0 & 0 \\ -0.0936 & 0 & 0 & -0.7755 \\ 0 & -0.5705 & 0 & 0 \end{bmatrix}.$$

Finally, there are lots of methods that will do better than this, usually by taking this as a starting point and ‘polishing’ the result after that. Several of these have been shown to give fairly reliable, if modest, improvements. You were not required to implement any of these methods.

- 5.11 Measurement with bounded errors.** A series of K measurements $y_1, \dots, y_K \in \mathbf{R}^p$, are taken in order to estimate an unknown vector $x \in \mathbf{R}^q$. The measurements are related to the unknown vector x by $y_i = Ax + v_i$, where v_i is a measurement noise that satisfies $\|v_i\|_\infty \leq \alpha$ but is otherwise unknown. (In other words, the entries of v_1, \dots, v_K are no larger than α .) The matrix A and the measurement noise norm bound α are known. Let X denote the set of vectors x that are consistent with the observations y_1, \dots, y_K , *i.e.*, the set of x that could have resulted in the measurements made. Is X convex?

Now we will examine what happens when the measurements are occasionally in error, *i.e.*, for a few i we have no relation between x and y_i . More precisely suppose that I_{fault} is a subset of $\{1, \dots, K\}$,

and that $y_i = Ax + v_i$ with $\|v_i\|_\infty \leq \alpha$ (as above) for $i \notin I_{\text{fault}}$, but for $i \in I_{\text{fault}}$, there is no relation between x and y_i . The set I_{fault} is the set of times of the faulty measurements.

Suppose you know that I_{fault} has at most J elements, *i.e.*, out of K measurements, at most J are faulty. You do not know I_{fault} ; you know only a bound on its cardinality (size). For what values of J is X , the set of x consistent with the measurements, convex?

Solution. The set X of vectors x consistent with the observations y_1, \dots, y_K is given by:

$$X = \{x \mid Ax \leq y_i + \alpha e, \quad Ax \geq y_i - \alpha e, \quad i = 1, \dots, K\}$$

where e is the p -dimensional vector of ones. This is a polyhedron and thus a convex set.

Now assume that we know an *upper* bound, J , on the cardinality of I_{fault} , the set of faulty measurements. If $J > 0$, for each possible set I_{fault} , the set of vectors x consistent with the valid measurements $(y_i, i \in \{1, \dots, K\} \setminus I_{\text{fault}})$ is again a polyhedron. The set X will be the union of these polyhedra, which is in general not convex.

For example, consider the case where $p = q = 1$ and $A = \alpha = 1$. Assume that we have three measurements where $y_1 = 2$, $y_2 = -0.5$, $y_3 = 0.5$. In the case that $J = 1$ we have four different alternatives, for the cases where measurements 1, 2, 3 or none of them are faulty. The possible x for the four cases are $[-0.5, 0.5]$, $[1, 1.5]$, \emptyset , \emptyset respectively. Clearly all of these sets are convex, whereas their union, which is equal to $[-0.5, 0.5] \cup [1, 1.5]$, is nonconvex.

Clearly, if $J = 0$ we are back to the case with no faulty measurements analyzed above. Thus, the set X is convex for $J = 0$.

5.12 Least-squares with some permuted measurements. We want to estimate a vector $x \in \mathbf{R}^n$, given some linear measurements of x corrupted with Gaussian noise. Here's the catch: some of the measurements have been *permuted*.

More precisely, our measurement vector $y \in \mathbf{R}^m$ has the form

$$y = P(Ax + v),$$

where v_i are IID $\mathcal{N}(0, 1)$ measurement noises, $x \in \mathbf{R}^n$ is the vector of parameters we wish to estimate, and $P \in \mathbf{R}^{m \times m}$ is a permutation matrix. (This means that each row and column of P has exactly one entry equal to one, and the remaining $m - 1$ entries zero.) We assume that $m > n$ and that at most k of the measurements are permuted; *i.e.*, $Pe_i \neq e_i$ for no more than k indices i . We are interested in the case when $k < m$ (*e.g.* $k = 0.4m$); that is, only *some* of the measurements have been permuted. We want to estimate x and P .

Once we make a guess \hat{P} for P , we can get the maximum likelihood estimate of x by minimizing $\|Ax - \hat{P}^T y\|_2$. The residual $A\hat{x} - \hat{P}^T y$ is then our guess of what v is, and should be consistent with being a sample of a $\mathcal{N}(0, I)$ vector.

In principle, we can find the maximum likelihood estimate of x and P by solving a set of $\binom{m}{k}(k! - 1)$ least-squares problems, and choosing one that has minimum residual. But this is not practical unless m and k are both very small.

Describe a *heuristic* method for approximately solving this problem, using convex optimization. (There are many different approaches which work quite well.)

You might find the following fact useful. The solution to

$$\text{minimize } \|Ax - P^T y\|$$

over $P \in \mathbf{R}^{m \times m}$ a permutation matrix, is the permutation that matches the smallest entry in y with the smallest entry in Ax , does the same for the second smallest entries and so forth.

Carry out your method on the data in `ls_perm_meas_data.*`. Give your estimate of the permuted indices. The data file includes the true permutation matrix and value of x (which of course you cannot use in forming your estimate). Compare the estimate of x you get after your guessed permutation with the estimate obtained assuming $P = I$.

Remark. This problem comes up in several applications. In target tracking, we get multiple noisy measurements of a set of targets, and then guess which targets are the same in the different sets of measurements. If some of our guesses are wrong (*i.e.*, our target association is wrong) we have the present problem. In vision systems the problem arises when we have multiple camera views of a scene, which give us noisy measurements of a set of features. A feature correspondence algorithm guesses which features in one view correspond to features in other views. If we make some feature correspondence errors, we have the present problem.

Note. If you are using Julia, you might have to set the solver to run more than the default number of iterations, using `solve!(problem, SCSSolver(max_iters=10000))`.

Solution.

The basic idea is to treat the permuted measurements as measurements with high noise, *i.e.*, as outliers. So we first use some robust estimator, like ℓ_1 or Huber, to estimate x , with no permutations.

We then look for measurements with big residuals; these are (probably) the ones that were permuted. If a small enough number of candidates come up, we can try all permutations of these measurements, to see which has the smallest residuals.

Otherwise we can remove the k largest outliers, and solve the least squares problem without those measurements. We can then use this estimate x to permute the suspected measurements, using the method described in the problem statement.

Finally, the method can be applied recursively. That is, once we guess \hat{P} , we permute the measurements to get $\hat{y} = \hat{P}^T y$, and apply the method again. If with this new data we guess that $\hat{P} = I$, we're done; if not, we permute again. (Our final estimate of the permutation is then the product of the estimated permutations from each step.)

The figure shows the residuals after solving the robust estimation problem. There are some clear outliers, the k largest of which we take to be our candidate indices. We use one iteration of the method outlined above to get x_{final} . Simply using $\hat{P} = I$ gives the naive estimator, x_{naive} . The errors were

$$\begin{aligned}\|x_{\text{true}} - x_{\text{final}}\| &= 0.061 \\ \|x_{\text{true}} - x_{\text{naive}}\| &= 3.4363\end{aligned}$$

in Matlab,

$$\begin{aligned}\|x_{\text{true}} - x_{\text{final}}\| &= 0.084 \\ \|x_{\text{true}} - x_{\text{naive}}\| &= 2.2684\end{aligned}$$

in Python, and

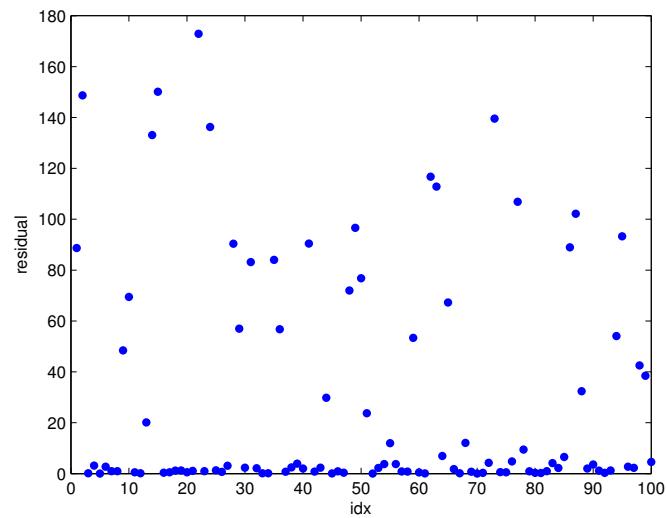


Figure 9: Robust estimator residuals

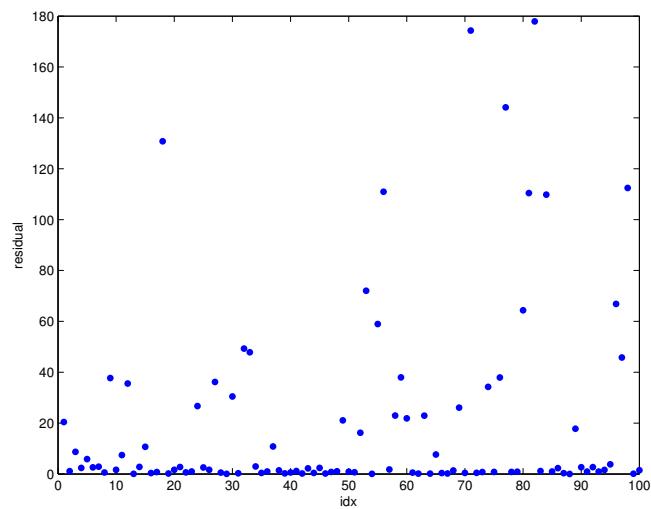


Figure 10: Robust estimator residuals (Python)

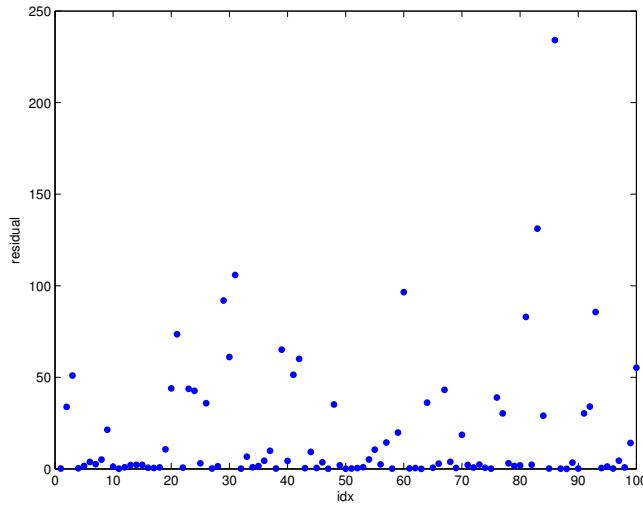


Figure 11: Robust estimator residuals (Julia)

$$\begin{aligned}\|x_{\text{true}} - x_{\text{final}}\| &= 0.068 \\ \|x_{\text{true}} - x_{\text{naive}}\| &= 2.6768\end{aligned}$$

in Julia. We identified all but one of the (37) permuted indices for the Matlab case, all but four of the (40) permuted indices for the Python case, and all but two of the (39) permuted indices for the Julia case.

The following Matlab code solves the problem.

```
ls_perm_meas_data

% naive estimator (P = I)
x_naive = A\y;

% robust estimator
cvx_begin
variable x_hub(n)
minimize(sum(huber(A*x_hub-y,1)))
cvx_end

plot(abs(A*x_hub-y),'.','MarkerSize', 15);
ylabel('residual'); xlabel('idx');

% remove k largest residuals
[vals,cand_idxs]=sort(abs(A*x_hub-y),'descend');
```

```

cand_idxs=sort(cand_idxs(1:k));
A_hat=A;y_hat=y;
A_hat(cand_idxs,:)=[];y_hat(cand_idxs)=[] ;
% ls estimate with candidate idxs removed
x_ls=A_hat\y_hat;

% match predicted outputs with measurements
[a,b]=sort(A(cand_idxs,:)*x_ls);
[a,c]=sort(y(cand_idxs));

% reorder A matrix
cand_perms=cand_idxs;
cand_perms(b,:)=cand_perms(c,:);
A(cand_perms,:)=A(cand_idxs,:);
x_final=A\y;

% final estimate of permuted indices
cand_idxs(find(cand_perms==cand_idxs))=[];

% residuals
norm(x_naive-x_true)
norm(x_final-x_true)

```

The following Python code solves the problem.

```

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

np.random.seed(0)
m=100
k=40 # max # permuted measurements
n=20
A=10*np.random.randn(m,n)
x_true=np.random.randn(n,1) # true x value
y_true = A.dot(x_true) + np.random.randn(m,1)
# build permuted indices
perm_idxs=np.random.permutation(m)
perm_idxs=np.sort(perm_idxs[:k])
temp_perm=np.random.permutation(k)
new_pos=np.zeros(k)
for i in range(k):
    new_pos[i] = perm_idxs[temp_perm[i]]
new_pos = new_pos.astype(int)
# true permutation matrix
P=np.identity(m)

```

```

P[perm_idxs]=P[new_pos,:]
true_perm=[]
for i in range(k):
    if perm_idxs[i] != new_pos[i]:
        true_perm = np.append(true_perm, perm_idxs[i])
y=P.dot(y_true)
new_pos = None

# naive estimator (P=I)
x_naive = np.linalg.lstsq(A,y)[0]

# robust estimator
x_hub = cvx.Variable(n)
obj = cvx.sum_entries(cvx.huber(A*x_hub-y))
cvx.Problem(cvx.Minimize(obj)).solve()

plt.figure(1)
plt.plot(np.arange(m), np.abs(A.dot(x_hub.value)-y), '.')
plt.ylabel('residual')
plt.xlabel('idx')

# remove k largest residuals
cand_ids = np.zeros(m);
cand_ids[:] = np.fliplr(np.argsort(np.abs(A.dot(x_hub.value)-y).T))
cand_ids = np.sort(cand_ids[:k])
cand_ids = cand_ids.astype(int)
keep_ids = np.zeros(m);
keep_ids[:] = np.argsort(np.abs(A.dot(x_hub.value)-y).T)
keep_ids = np.sort(keep_ids[:-(m-k)])
keep_ids = keep_ids.astype(int)
A_hat = A[keep_ids,:]
y_hat = y[keep_ids,:]
# ls estimate with candidate idxs removed
x_ls = np.linalg.lstsq(A_hat,y_hat)[0]

# match predicted outputs with measurements
b = np.zeros(k)
c = np.zeros(k)
b[:] = np.argsort(A[cand_ids,:].dot(x_ls).T)
b = b.astype(int)
c[:] = np.argsort(y[cand_ids].T)
c = c.astype(int)

# reorder A matrix
cand_perms = np.zeros(len(cand_ids));

```

```

cand_perms[:] = cand_idxs[:]
cand_perms[b] = cand_perms[c]
cand_perms = cand_perms.astype(int)
A[cand_perms, :] = A[cand_idxs, :]
x_final = np.linalg.lstsq(A, y)[0]

# final estimate of permuted indices
perm_estimate = []
for i in range(k):
    if cand_perms[i] != cand_idxs[i]:
        perm_estimate.append(cand_idxs[i])

naive_error = np.linalg.norm(x_naive - x_true)
final_error = np.linalg.norm(x_final - x_true)

```

The following Julia code solves the problem.

```

include("ls_perm_meas_data.jl")

using Convex, SCS, Gadfly

# naive estimator (P = I)
x_naive = A\y;

# robust estimator
x_hub = Variable(n);
p = minimize(sum(huber(A*x_hub-y, 1)));
solve!(p, SCSSolver(max_iters=10000));
residuals = abs(vec(evaluate(A*x_hub - y)));

pl = plot(
    x=1:length(residuals),
    y=residuals,
    Geom.point,
    Guide.xlabel("idx"),
    Guide.ylabel("residual")
);
display(p);
draw(PS("ls_perm_meas_jl.eps", 6inch, 5inch), pl);

# remove k largest residuals
cand_idxs = sortperm(residuals, rev=true);
not_cand_idxs = sort(cand_idxs[k+1:end]);
cand_idxs = sort(cand_idxs[1:k]);
A_hat = A[not_cand_idxs, :];
y_hat = y[not_cand_idxs];

```

```

# ls estimate with candidate idxs removed
x_ls = A_hat\y_hat;

# match predicted outputs with measurements
b = sortperm(A[cand_idxs, :]*x_ls);
c = sortperm(y[cand_idxs]); 

# reorder A matrix
cand_perms = copy(cand_idxs);
cand_perms[b] = cand_perms[c];
A[cand_perms,:] = A[cand_idxs, :];
x_final = A\y;

# final estimate of permuted indices
cand_idxs = cand_idxs[cand_perms.!=cand_idxs];

# residuals
println(norm(x_naive-x_true));
println(norm(x_final-x_true));

```

5.13 Fitting with censored data. In some experiments there are two kinds of measurements or data available: The usual ones, in which you get a number (say), and *censored data*, in which you don't get the specific number, but are told something about it, such as a lower bound. A classic example is a study of lifetimes of a set of subjects (say, laboratory mice). For those who have died by the end of data collection, we get the lifetime. For those who have not died by the end of data collection, we do not have the lifetime, but we do have a lower bound, *i.e.*, the length of the study. These are the censored data values.

We wish to fit a set of data points,

$$(x^{(1)}, y^{(1)}), \dots, (x^{(K)}, y^{(K)}),$$

with $x^{(k)} \in \mathbf{R}^n$ and $y^{(k)} \in \mathbf{R}$, with a linear model of the form $y \approx c^T x$. The vector $c \in \mathbf{R}^n$ is the model parameter, which we want to choose. We will use a least-squares criterion, *i.e.*, choose c to minimize

$$J = \sum_{k=1}^K (y^{(k)} - c^T x^{(k)})^2.$$

Here is the tricky part: some of the values of $y^{(k)}$ are censored; for these entries, we have only a (given) lower bound. We will re-order the data so that $y^{(1)}, \dots, y^{(M)}$ are given (*i.e.*, uncensored), while $y^{(M+1)}, \dots, y^{(K)}$ are all censored, *i.e.*, unknown, but larger than D , a given number. All the values of $x^{(k)}$ are known.

- (a) Explain how to find c (the model parameter) and $y^{(M+1)}, \dots, y^{(K)}$ (the censored data values) that minimize J .
- (b) Carry out the method of part (a) on the data values in `cens_fit_data.*`. Report \hat{c} , the value of c found using this method.

Also find \hat{c}_{ls} , the least-squares estimate of c obtained by simply ignoring the censored data samples, *i.e.*, the least-squares estimate based on the data

$$(x^{(1)}, y^{(1)}), \dots, (x^{(M)}, y^{(M)}).$$

The data file contains c_{true} , the true value of c , in the vector `c_true`. Use this to give the two relative errors

$$\frac{\|c_{true} - \hat{c}\|_2}{\|c_{true}\|_2}, \quad \frac{\|c_{true} - \hat{c}_{ls}\|_2}{\|c_{true}\|_2}.$$

Solution.

- (a) The trick is to introduce dummy variables to serve as placeholders for the measurements which are censored, *i.e.*, $y^{(k)}$, $k = M + 1, \dots, K$. By introducing the dummy variables $(z^{(1)}, \dots, z^{(K-M)})$, we get the QP

$$\begin{aligned} & \text{minimize} && \sum_{k=1}^M (y^{(k)} - c^T x^{(k)})^2 + \sum_{k=M+1}^K (z^{(k-M)} - c^T x^{(k)})^2 \\ & \text{subject to} && z^{(k)} \geq D, \quad k = 1, \dots, K - M, \end{aligned}$$

where the variables are c and $z^{(k)}$, $k = 1, \dots, K - M$.

- (b) The following MATLAB code solves the problem.

```

cens_fit_data;

% Using censored data method
cvx_begin
    variables c(n) z(K-M)
    minimize(sum_square(y-X(:,1:M)'*c)+sum_square(z-X(:,M+1:K)'*c))
    subject to
        z >= D
cvx_end
c_cens = c;

% Comparison to least squares method, ignoring all censored data
cvx_begin
    variable c(n)
    minimize(sum_square(y-X(:,1:M)'*c))
cvx_end
c_ls = c;

[c_true c_cens c_ls]
cens_rellerr = norm(c_cens-c_true)/norm(c_true)
ls_rellerr = norm(c_ls-c_true)/norm(c_true)

```

We get the following estimates of our parameter vector:

```
[c_true c_cens c_ls] =
-0.4326   -0.2946   -0.3476
```

-1.6656	-1.7541	-1.7955
0.1253	0.2589	0.2000
0.2877	0.2241	0.1672
-1.1465	-0.9917	-0.8357
1.1909	1.3018	1.3005
1.1892	1.4262	1.8276
-0.0376	-0.1554	-0.5612
0.3273	0.3785	0.3686
0.1746	0.2261	-0.0454
-0.1867	-0.0826	-0.1096
0.7258	1.0427	1.5265
-0.5883	-0.4648	-0.4980
2.1832	2.1942	2.4164
-0.1364	-0.3586	-0.5563
0.1139	-0.1973	-0.3701
1.0668	1.0194	0.9900
0.0593	-0.1186	-0.2539
-0.0956	-0.1211	-0.1762
-0.8323	-0.7523	-0.4349

This gives a relative error of 0.1784 for \hat{c} , and a relative error of 0.3907 for \hat{c}_{ls} .
The following Python code solves the problem.

```

import numpy as np
import cvxpy as cvx
# data for censored fitting problem.
np.random.seed(15)

n = 20; # dimension of x's
M = 25; # number of non-censored data points
K = 100; # total number of points
c_true = np.random.randn(n,1)
X = np.random.randn(n,K)
y = np.dot(np.transpose(X),c_true) + 0.1*(np.sqrt(n))*np.random.randn(K,1)

# Reorder measurements, then censor
sort_ind = np.argsort(y.T)
y = np.sort(y.T)
y = y.T
X = X[:, sort_ind.T]
D = (y[M-1]+y[M])/2.0
y = y[range(M)]

# Using censored data method
z = cvx.Variable(K-M)
c = cvx.Variable(n)
constraints = [D <= z]

```

```

objective = cvx.Minimize(cvx.sum_squares(y-X[:,range(M)].T*c) + cvx.sum_squares(z-X[:,M:]).T*c)
prob = cvx.Problem(objective, constraints)
prob.solve()
c_cens = c.value

# Comparison to least squares method, ignoring all censored data
objective = cvx.Minimize(cvx.sum_squares(y-X[:,range(M)].T*c))
prob = cvx.Problem(objective, [])
prob.solve()
c_ls = c.value;
cens_rellerr = np.linalg.norm(c_cens-c_true)/np.linalg.norm(c_true)
ls_rellerr = np.linalg.norm(c_ls-c_true)/np.linalg.norm(c_true)

print "c_true is:"
print c_true.T
print "c_cens is:"
print np.squeeze(np.asarray(c_cens.T))
print "c_ls is:"
print np.squeeze(np.asarray(c_ls.T))
print "The relative error when we use the censored data is:"
print cens_rellerr
print "The relative error when we do not use the censored data is:"
print ls_rellerr

```

We get the following estimates of our parameter vector:

[c_true c_cens c_ls] =		
-0.3123	-0.4063	-0.8695
0.3393	0.4083	0.3878
-0.1559	-0.3226	-0.0792
-0.5018	-0.6489	-0.5269
0.2356	0.3401	0.4485
-1.7636	-1.8654	-2.1460
-1.0959	-0.9181	-0.7917
-1.0878	-1.1706	-0.8663
-0.3052	-0.3446	-0.1832
-0.4737	-0.4661	-0.2506
-0.2006	-0.2165	-0.1583
0.3552	0.2531	0.5553
0.6895	0.5241	0.4282
0.4106	0.3151	0.0690
-0.5650	-0.5048	-0.4336
0.5994	0.5844	0.3568
-0.1629	-0.1862	-0.2024
1.6002	1.6390	2.0071
0.6816	0.6823	0.8107
0.0149	0.1073	0.0936

This gives a relative error of 0.1306 for \hat{c} , and a relative error of 0.3332 for \hat{c}_{ls} .
The following Julia code solves the problem.

```
include("cens_fit_data.jl");

using Convex, SCS

z = Variable(K-M);
c = Variable(n);
constraints = [D <= z];
prob = minimize(sumsquares(y - X[:,1:M]'*c) + sumsquares(z - X[:, M+1:end]'*c), constraint);
solve!(prob, SCSSolver(verbose=0));
c_cens = c.value;

# Comparison to least squares method, ignoring all censored data
prob = minimize(sumsquares(y - X[:,1:M]'*c), constraints);
solve!(prob, SCSSolver(verbose=0));
c_ls = c.value;

cens_rellerr = norm(c_cens - c_true)/norm(c_true);
ls_rellerr = norm(c_ls - c_true)/norm(c_true);

println("c_true, c_cens, c_ls: ")
println([c_true c_cens c_ls])

@printf("Relative error when we use the censored data is: %.4f\n", cens_rellerr)
@printf("Relative error when we do not use the censored data is: %.4f\n", ls_rellerr)
```

We get the following estimates of our parameter vector:

[c_true	c_cens	c_ls]	=
-0.1555	0.0593	-0.0546	
0.7980	1.0252	0.7304	
-0.6821	-0.7113	-0.9468	
0.1050	0.2874	0.5334	
0.1488	0.1385	0.5203	
0.2274	0.2207	0.1647	
-0.3334	-0.2384	-0.1678	
1.9171	1.5830	1.7391	
-0.6174	-0.4677	-0.2668	
-0.3075	-0.6946	-0.8639	
1.2054	1.2582	1.3485	
-0.3394	-0.1406	0.1117	
-1.9851	-1.9871	-2.1897	
0.8647	0.8050	0.9064	

1.1234	1.2350	1.4486
-1.0329	-1.0031	-1.2435
0.1215	0.1263	0.2917
0.5975	0.5997	0.4654
-0.5449	-0.6555	-0.8844
-1.3857	-1.6498	-1.8234

This gives a relative error of 0.1844 for \hat{c} , and a relative error of 0.3170 for \hat{c}_{ls} .

- 5.14 Spectrum analysis with quantized measurements.** A sample is made up of n compounds, in quantities $q_i \geq 0$, for $i = 1, \dots, n$. Each compound has a (nonnegative) spectrum, which we represent as a vector $s^{(i)} \in \mathbf{R}_+^m$, for $i = 1, \dots, n$. (Precisely what $s^{(i)}$ means won't matter to us.) The spectrum of the sample is given by $s = \sum_{i=1}^n q_i s^{(i)}$. We can write this more compactly as $s = Sq$, where $S \in \mathbf{R}^{m \times n}$ is a matrix whose columns are $s^{(1)}, \dots, s^{(n)}$.

Measurement of the spectrum of the sample gives us an interval for each spectrum value, *i.e.*, $l, u \in \mathbf{R}_+^m$ for which

$$l_i \leq s_i \leq u_i, \quad i = 1, \dots, m.$$

(We don't directly get s .) This occurs, for example, if our measurements are quantized.

Given l and u (and S), we cannot in general deduce q exactly. Instead, we ask you to do the following. For each compound i , find the range of possible values for q_i consistent with the spectrum measurements. We will denote these ranges as $q_i \in [q_i^{\min}, q_i^{\max}]$. Your job is to find q_i^{\min} and q_i^{\max} .

Note that if q_i^{\min} is large, we can confidently conclude that there is a significant amount of compound i in the sample. If q_i^{\max} is small, we can confidently conclude that there is not much of compound i in the sample.

- (a) Explain how to find q_i^{\min} and q_i^{\max} , given S , l , and u .
- (b) Carry out the method of part (a) for the problem instance given in `spectrum_data.m`. (Executing this file defines the problem data, and plots the compound spectra and measurement bounds.) Plot the minimum and maximum values versus i , using the commented out code in the data file. Report your values for q_4^{\min} and q_4^{\max} .

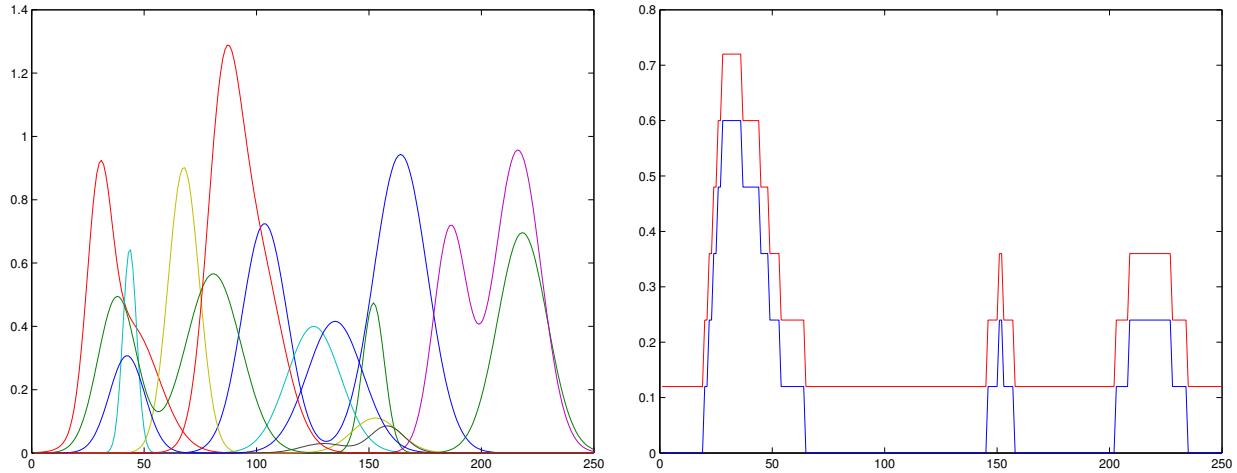
Solution. To find q_i^{\min} , we solve the convex optimization problem

$$\begin{aligned} & \text{minimize} && q_i \\ & \text{subject to} && l \preceq Sq \preceq u, \quad q \succeq 0, \end{aligned}$$

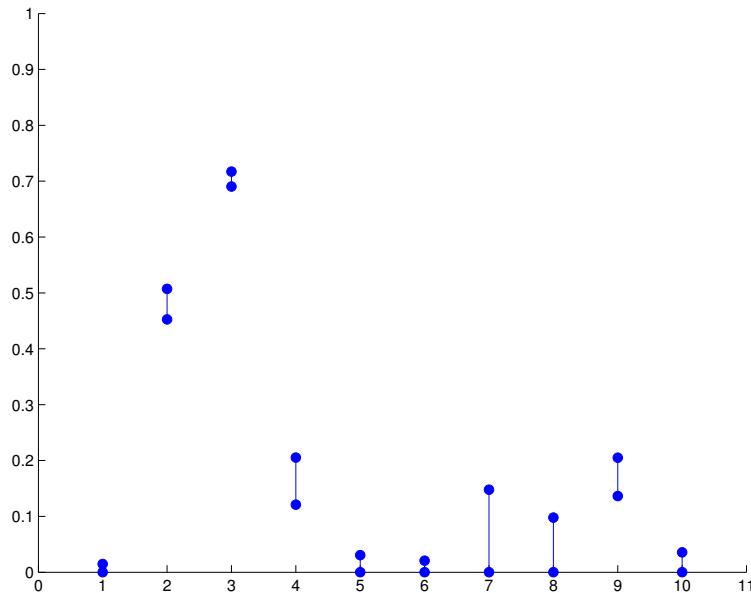
with variable $q \in \mathbf{R}^n$. Then we set $q_i^{\min} = q_i^*$. Similarly to find q_i^{\max} , we solve the convex optimization problem

$$\begin{aligned} & \text{maximize} && q_i \\ & \text{subject to} && l \preceq Sq \preceq u, \quad q \succeq 0, \end{aligned}$$

and we set $q_i^{\max} = q_i^*$. Here is a plot of the spectra of the compounds $s^{(1)}, \dots, s^{(n)}$, alongside the lower and upper bounds u and l .



Here is a plot of q_i^{\min} and q_i^{\max} , for $i = 1, \dots, n$.



For this particular instance, $q_4^{\min} = 0.121$, $q_4^{\max} = 0.205$.

The matlab code to solve this problem is given below.

```
spectrum_data;

qmin = zeros(n,1); qmax = zeros(n,1);
for i = 1:n
    cvx_begin
        variable q(n)
        l <= S*q; u >= S*q;
        q >= 0;
        minimize(q(i))
```

```

cvx_end
qmin(i) = q(i);
cvx_begin
    variable q(n)
    l <= S*q; u >= S*q;
    q >= 0;
    maximize(q(i))
cvx_end
qmax(i) = q(i);
end

% print quantity bounds
[qmin qmax]

figure; hold on;
for i = 1:n
    plot([i,i],[qmin(i),qmax(i)],'o-','MarkerFaceColor','b');
end
axis([0,11,0,1]);
%print('-depsc','spect_minmax.eps');

```

5.15 Learning a quadratic pseudo-metric from distance measurements. We are given a set of N pairs of points in \mathbf{R}^n , x_1, \dots, x_N , and y_1, \dots, y_N , together with a set of distances $d_1, \dots, d_N > 0$.

The goal is to find (or estimate or learn) a quadratic pseudo-metric d ,

$$d(x, y) = \left((x - y)^T P (x - y) \right)^{1/2},$$

with $P \in \mathbf{S}_+^n$, which approximates the given distances, *i.e.*, $d(x_i, y_i) \approx d_i$. (The pseudo-metric d is a metric only when $P \succ 0$; when $P \succeq 0$ is singular, it is a pseudo-metric.)

To do this, we will choose $P \in \mathbf{S}_+^n$ that minimizes the mean squared error objective

$$\frac{1}{N} \sum_{i=1}^N (d_i - d(x_i, y_i))^2.$$

- (a) Explain how to find P using convex or quasiconvex optimization. If you cannot find an exact formulation (*i.e.*, one that is guaranteed to minimize the total squared error objective), give a formulation that approximately minimizes the given objective, subject to the constraints.
- (b) Carry out the method of part (a) with the data given in `quad_metric_data.m`. The columns of the matrices \mathbf{X} and \mathbf{Y} are the points x_i and y_i ; the row vector \mathbf{d} gives the distances d_i . Give the optimal mean squared distance error.

We also provide a test set, with data $\mathbf{X_test}$, $\mathbf{Y_test}$, and $\mathbf{d_test}$. Report the mean squared distance error on the test set (using the metric found using the data set above).

Solution.

(a) The problem is

$$\text{minimize } \frac{1}{N} \sum_{i=1}^N (d_i - d(x_i, y_i))^2$$

with variable $P \in \mathbf{S}_+^n$. This problem can be rewritten as

$$\text{minimize } \frac{1}{N} \sum_{i=1}^N (d_i^2 - 2d_i d(x_i, y_i) + d(x_i, y_i)^2),$$

with variable P (which enters through $d(x_i, y_i)$). The objective is convex because each term of the objective can be written as (ignoring the $1/N$ factor)

$$d_i^2 - 2d_i \left((x_i - y_i)^T P (x_i - y_i) \right)^{1/2} + (x_i - y_i)^T P (x_i - y_i),$$

which is convex in P . To see this, note that the first term is constant and the third term is linear in P . The middle term is convex because it is the negation of the composition of a concave function (square root) with a linear function of P .

- (b) The following code solves the problem for the given instance. We find that the optimal mean squared error on the training set is 0.887; on the test set, it is 0.827. This tells us that we probably haven't overfit. In fact, the optimal P is singular; it has one zero eigenvalue. This is correct; the positive semidefinite constraint is active.

Here is the solution in Matlab.

```
% learning a quadratic metric

quad_metric_data;

Z = X-Y;
cvx_begin
    variable P(n,n) symmetric
    % objective
    f = 0;
    for i = 1:N
        f = f + d(i)^2 - 2*d(i)*sqrt(Z(:,i)'*P*Z(:,i)) + Z(:,i)'*P*Z(:,i);
    end
    minimize (f/N)
    subject to
        P == semidefinite(n);
cvx_end

Z_test = X_test-Y_test;
d_hat = norms(sqrtm(P)*Z_test);
obj_test = sum_square(d_test - d_hat)/N_test
```

Here it is in Python.

```
from quad_metric_data import *
import numpy as np
from scipy import linalg as la
import cvxpy as cvx
```

```

Z = X - Y
P = cvx.Variable(n,n)
f = 0
for i in range(N):
    f += d[i]**2
    f += -2*d[i]*cvx.sqrt(cvx.quad_form(Z[:,i],P))
    f += cvx.quad_form(Z[:,i],P)

prob = cvx.Problem(cvx.Minimize(f/N),[P == cvx.Semidef(n)])
train_error = prob.solve()
print(train_error)

Z_test = X_test-Y_test
d_hat = np.linalg.norm(la.sqrtm(P.value).dot(Z_test),axis=0)
obj_test = (np.linalg.norm(d_test - d_hat)**2)/N_test
print(obj_test)

```

And finally, Julia.

```

# learning a quadratic metric
include("quad_metric_data.jl");
using Convex, SCS

Z = X-Y;
P = Semidefinite(n, n);

f = 0;
for i = 1:N
    f += d[i]^2 - 2*d[i]*sqrt(Z[:,i]'*P*Z[:,i]) + Z[:,i]'*P*Z[:,i];
end
p = minimize(f/N);
solve!(p, SCSSolver(max_iters=100000));

Z_test = X_test-Y_test;
d_hat = sqrt(sum((real(sqrtm(P.value))*Z_test).^2, 1));
obj_test = vecnorm(d_test - d_hat)^2/N_test;
println("train error: ", p.optval)
println("test error: ", obj_test)
eig(P.value)

```

5.16 Polynomial approximation of inverse using eigenvalue information. We seek a polynomial of degree k , $p(a) = c_0 + c_1a + c_2a^2 + \cdots + c_ka^k$, for which

$$p(A) = c_0I + c_1A + c_2A^2 + \cdots + c_kA^k$$

is an approximate inverse of the nonsingular matrix A , for all $A \in \mathcal{A} \subset \mathbf{R}^{n \times n}$. When $\hat{x} = p(A)b$ is used as an approximate solution of the linear equation $Ax = b$, the associated residual norm is

$\|A(p(A)b) - b\|_2$. We will judge our polynomial (*i.e.*, the coefficients c_0, \dots, c_k) by the worst case residual over $A \in \mathcal{A}$ and b in the unit ball:

$$R^{\text{wc}} = \sup_{A \in \mathcal{A}, \|b\|_2 \leq 1} \|A(p(A)b) - b\|_2.$$

The set of matrices we take is $\mathcal{A} = \{A \in \mathbf{S}^n \mid \sigma(A) \subseteq \Omega\}$, where $\sigma(A)$ is the set of eigenvalues of A (*i.e.*, its spectrum), and $\Omega \subset \mathbf{R}$ is a union of a set of intervals (that do not contain 0).

- (a) Explain how to find coefficients c_0^*, \dots, c_k^* that minimize R^{wc} . Your solution can involve expressions that involve the supremum of a polynomial (with scalar argument) over an interval.
- (b) Carry out your method for $k = 4$ and $\Omega = [-0.6, -0.3] \cup [0.7, 1.8]$. You can replace the supremum of a polynomial over Ω by a maximum over uniformly spaced (within each interval) points in Ω , with spacing 0.01. Give the optimal value $R^{\text{wc}*}$ and the optimal coefficients $c^* = (c_0^*, \dots, c_k^*)$.

Remarks. (Not needed to solve the problem.)

- The approximate inverse $p(A)b$ would be computed by recursively, requiring the multiplication of A with a vector k times.
- This approximate inverse could be used as a preconditioner for an iterative method.
- The Cayley-Hamilton theorem tells us that the inverse of any (invertible) matrix is a polynomial of degree $n - 1$ of the matrix. Our hope here, however, is to get a single polynomial, of relatively low degree, that serves as an approximate inverse for many different matrices.

Solution.

- (a) We can rewrite

$$R^{\text{wc}} = \sup_{A \in \mathcal{A}} \sup_{\|b\|_2 \leq 1} \|A(p(A)b) - b\|_2,$$

and recognize the inner supremum as the definition of the spectral norm of $Ap(A) - I$. If A is symmetric, then $Ap(A) - I$ is also symmetric, and its spectral norm is the largest absolute value of its eigenvalues.

Let QDQ^T be an eigenvalue decomposition of A , with Q orthogonal and D diagonal. Then

$$\begin{aligned} Ap(A) - I &= c_0 A + c_1 A^2 + \cdots + c_k A^{k+1} - I \\ &= c_0 Q D Q^T + c_1 Q D^2 Q^T + \cdots + c_k Q D^{k+1} Q^T - I \\ &= Q(c_0 D + c_1 D^2 + \cdots + c_k D^{k+1} - I) Q^T \\ &= Q(Dp(D) - I) Q^T, \end{aligned}$$

which shows that $\lambda p(\lambda) - 1 \in \sigma(Ap(A) - I)$ if $\lambda \in \sigma(A)$.

We can then rewrite

$$\begin{aligned} R^{\text{wc}} &= \sup_{A \in \mathcal{A}} \|Ap(A) - I\|_2 \\ &= \sup_{A \in \mathcal{A}} \sup_{\lambda \in \sigma(A)} |\lambda p(\lambda) - 1| \\ &= \sup_{\lambda \in \Omega} |\lambda p(\lambda) - 1| \\ &= \sup_{\lambda \in \Omega} |c_0 \lambda + c_1 \lambda^2 + \cdots + c_k \lambda^{k+1} - 1|, \end{aligned}$$

and note that R^{wc} is a convex function of c since, for any λ , $c_0\lambda + c_1\lambda^2 + \cdots + c_k\lambda^{k+1} - 1$ is an affine function of c . We can write the optimization problem as

$$\text{minimize } \sup_{\lambda \in \Omega} |c_0\lambda + c_1\lambda^2 + \cdots + c_k\lambda^{k+1} - 1|.$$

This problem can be converted exactly into an SDP (since the supremum of a polynomial has an LMI representation), but for any practical purpose simple sampling of Ω is fine.

- (b) Let $\lambda_1, \dots, \lambda_N$ be uniformly spaced (within each interval contained in Ω) sample points, with spacing 0.01. We approximate the objective by

$$\max_{i=1,\dots,N} |c_0\lambda_i + c_1\lambda_i^2 + \cdots + c_k\lambda_i^{k+1} - 1|.$$

Define $\Lambda \in \mathbf{R}^{N \times (k+1)}$ as $\Lambda_{ij} = \lambda_i^j$. This allows us to write the (approximate) optimization problem as

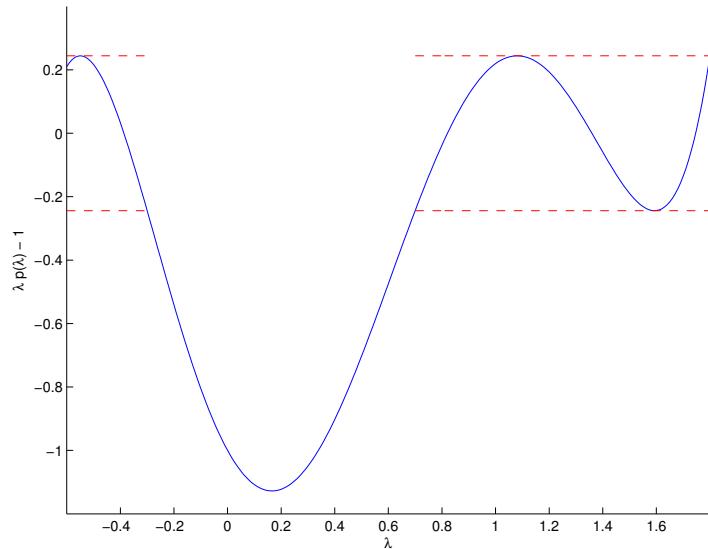
$$\text{minimize } \|\Lambda c - \mathbf{1}\|_\infty.$$

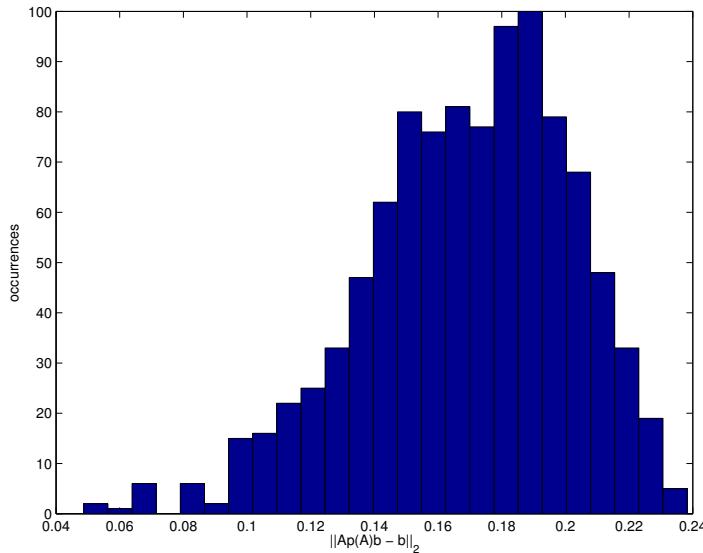
We find that $R^{\text{wc}*} = 0.2441$, with coefficients

$$c^* = (-1.54, 4.43, 1.98, -5.61, 1.96).$$

That's impressive: A single polynomial of degree 4 serves as a crude inverse for a whole family of matrices! We plot $\lambda p(\lambda) - 1$ with a blue line over the interval $[-0.6, 1.8]$, and indicate Ω and the bounds $\pm R^{\text{wc}*}$ with a red dashed line.

We didn't ask you to do this, but we also compute $\|Ap(A)b - b\|_2$ for 1000 random instances of $A \in \mathcal{A}$ with $n = 10$, and b with $\|b\|_2 = 1$. We generate A by sampling eigenvalues uniformly from Ω , placing them in a diagonal matrix D , and forming $A = QDQ^T$, for a random orthogonal matrix Q (which we generate by taking the QR decomposition of a matrix with Gaussian entries). We generate b from a Gaussian distribution and normalize (which gives a uniform distribution on the unit sphere). Of course, the distributions don't matter. A histogram and the code are given below. Note that the worst case error across the samples is a bit under 0.24, quite consistent with our value of $R^{\text{wc}*}$.





```
% Polynomial approximation of inverse using eigenvalue information.
k = 4; % k degree polynomials
l1 = -.6;
u1 = -.3;
l2 = .7;
u2 = 1.8;
eta = 0.01;

lambda = lambdas';
for i = 1:k
    Lambda(:,end+1) = Lambda(:,end).*lambdas;
end

cvx_begin
    variable c(k+1)
    minimize( norm(Lambda*c - 1,inf) )
cvx_end
R_wc = cvx_optval;

lambdas_full = [l1:eta:u2]';
Lambda_full = lambdas_full;
for i = 1:k
    Lambda_full(:,end+1) = Lambda_full(:,end).*lambdas_full;
end

y2 = Lambda_full*c - 1;
figure
```

```

hold on
plot(lambdas_full,y2)
plot(l1:eta:u1,0*(l1:eta:u1) + R_wc,'--r',...
     l1:eta:u1,0*(l1:eta:u1) - R_wc,'--r',...
     l2:eta:u2,0*(l2:eta:u2) + R_wc,'--r',...
     l2:eta:u2,0*(l2:eta:u2) - R_wc,'--r')

ylabel('lambda p(\lambda) - 1')
xlabel('lambda')
axis([l1 u2 -1.2 .4])
print '-depsc' 'poly_approx_inv.eps'

%testing code
randn('state',0)
rand('state',0)
n = 10;
runs = 1000;
r = (u1-l1)/(u2-l2 + u1-l1);

errors = zeros(runs,1);
for i = 1:runs
    %create random eigenvalues in [l1,u1] \cup [l2,u2]
    a = rand(n,1);
    a = l1 + (l2-u1)*(sign(a-r)+1)/2 + (u2-l2 + u1-l1)*a;
    [Q R] = qr(rand(n,n));
    A = Q*diag(a)*Q';

    b = randn(n,1);
    b = b/norm(b);

    q = -b;
    for j = 1:k+1
        q = q + c(j)*A^j*b;
    end
    errors(i) = norm(q);
end

figure
hist(errors,25)
xlabel('||Ap(A)b - b||_2')
ylabel('occurrences')
print -depsc poly_approx_inv_hist.eps

```

5.17 Fitting a generalized additive regression model. A *generalized additive model* has the form

$$f(x) = \alpha + \sum_{j=1}^n f_j(x_j),$$

for $x \in \mathbf{R}^n$, where $\alpha \in \mathbf{R}$ is the offset, and $f_j : \mathbf{R} \rightarrow \mathbf{R}$, with $f_j(0) = 0$. The functions f_j are called the *regressor functions*. When each f_j is linear, *i.e.*, has the form $w_j x_j$, the generalized additive model is the same as the standard (linear) regression model. Roughly speaking, a generalized additive model takes into account nonlinearities in each regressor x_j , but not nonlinear interactions among the regressors. To visualize a generalized additive model, it is common to plot each regressor function (when n is not too large).

We will restrict the functions f_j to be piecewise-affine, with given knot points $p_1 < \dots < p_K$. This means that f_j is affine on the intervals $(-\infty, p_1]$, $[p_1, p_2]$, \dots , $[p_{K-1}, p_K]$, $[p_K, \infty)$, and continuous at p_1, \dots, p_K . Let C denote the total (absolute value of) change in slope across all regressor functions and all knot points. The value C is a measure of nonlinearity of the regressor functions; when $C = 0$, the generalized additive model reduces to a linear regression model.

Now suppose we observe samples or data $(x^{(1)}, y^{(1)}), \dots, (x^{(N)}, y^{(N)}) \in \mathbf{R}^n \times \mathbf{R}$, and wish to fit a generalized additive model to the data. We choose the offset and the regressor functions to minimize

$$\frac{1}{N} \sum_{i=1}^N (y^{(i)} - f(x^{(i)}))^2 + \lambda C,$$

where $\lambda > 0$ is a regularization parameter. (The first term is the mean-square error.)

- (a) Explain how to solve this problem using convex optimization.
- (b) Carry out the method of part (a) using the data in the file `gen_add_reg_data.m`. This file contains the data, given as an $N \times n$ matrix \mathbf{X} (whose rows are $(x^{(i)})^T$), a column vector \mathbf{y} (which give $y^{(i)}$), a vector \mathbf{p} that gives the knot points, and the scalar `lambda`.

Give the mean-square error achieved by your generalized additive regression model. Compare the estimated and true regressor functions in a 3×3 array of plots (using the plotting code in the data file as a template), over the range $-10 \leq x_i \leq 10$. The true regressor functions (to be used only for plotting, of course) are given in the cell array \mathbf{f} .

Hints.

- You can represent each regressor function f_j as a linear combination of the basis functions $b_0(u) = u$ and $b_i(u) = (u - p_k)_+ - (-p_k)_+$ for $k = 1, 2, \dots, K$, where $(a)_+ = \max\{a, 0\}$.
- You might find the matrix $\mathbf{XX} = [b_0(\mathbf{X}) \ b_1(\mathbf{X}) \ \dots \ b_K(\mathbf{X})]$ useful.

Solution. There is not much more to say beyond showing the code and the plot.

```
% Fitting a generalized additive regression model.
gen_add_reg_data;
```

```
%build an augmented data matrix XX
XX=X;
```

```

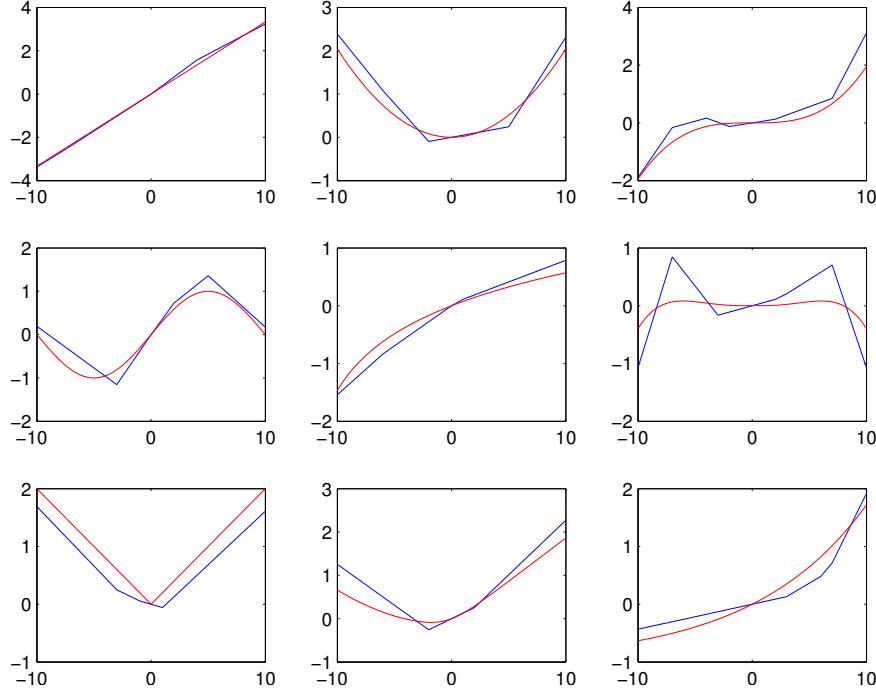
for ii=1:K
    XX=[XX,max(0,X-p(ii))+min(p(ii),0)];
end

%Perform regression
cvx_begin
    variables alpha c(9*(K+1))
    minimize(1/N*sum_square(y-alpha-XX*c)+lambda*norm(c,1))
cvx_end

%Plot functions.
xx=linspace(-10,10,1024);
yy=zeros(9,1024);
figure
for jj=1:9
    yy(jj,:)=c(jj)*xx;
    for ii=1:K
        yy(jj,:)=yy(jj,:)+c(ii*9+jj)*(pos(xx-p(ii))-pos(-p(ii)));
    end
    subplot(3,3,jj);
    plot(xx,yy(jj,:));
    hold on;
    plot(xx,f{jj}(xx),'r')
end

print -depsc gen_add_reg.eps

```



The blue and red lines correspond to the estimated and true regressors, respectively.

5.18 Multi-label support vector machine. The basic SVM described in the book is used for classification of data with two labels. In this problem we explore an extension of SVM that can be used to carry out classification of data with more than two labels. Our data consists of pairs $(x_i, y_i) \in \mathbf{R}^n \times \{1, \dots, K\}$, $i = 1, \dots, m$, where x_i is the feature vector and y_i is the label of the i th data point. (So the labels can take the values $1, \dots, K$.) Our classifier will use K affine functions, $f_k(x) = a_k^T x + b_k$, $k = 1, \dots, K$, which we also collect into affine function from \mathbf{R}^n into \mathbf{R}^K as $f(x) = Ax + b$. (The rows of A are a_k^T .) Given feature vector x , we guess the label $\hat{y} = \text{argmax}_k f_k(x)$. We assume that exact ties never occur, or if they do, an arbitrary choice can be made. Note that if a multiple of $\mathbf{1}$ is added to b , the classifier does not change. Thus, without loss of generality, we can assume that $\mathbf{1}^T b = 0$.

To correctly classify the data examples, we need $f_{y_i}(x_i) > \max_{k \neq y_i} f_k(x_i)$ for all i . This is a set of homogeneous strict inequalities in a_k and b_k , which are feasible if and only if the set of nonstrict inequalities $f_{y_i}(x_i) \geq 1 + \max_{k \neq y_i} f_k(x_i)$ are feasible. This motivates the loss function

$$L(A, b) = \sum_{i=1}^m \left(1 + \max_{k \neq y_i} f_k(x_i) - f_{y_i}(x_i) \right)_+,$$

where $(u)_+ = \max\{u, 0\}$. The multi-label SVM chooses A and b to minimize

$$L(A, b) + \mu \|A\|_F^2,$$

subject to $\mathbf{1}^T b = 0$, where $\mu > 0$ is a regularization parameter. (Several variations on this are possible, such as regularizing b as well, or replacing the Frobenius norm squared with the sum of norms of the columns of A .)

- (a) Show how to find A and b using convex optimization. Be sure to justify any changes of variables or reformulation (if needed), and convexity of the objective and constraints in your formulation.
- (b) Carry out multi-label SVM on the data given in `multi_label_svm_data.m`. Use the data given in X and y to fit the SVM model, for a range of values of μ . This data set includes an additional set of data, X_{test} and y_{test} , that you can use to test the SVM models. Plot the test set classification error rate (*i.e.*, the fraction of data examples in the test set for which $\hat{y} \neq y$) versus μ .

You don't need to try more than 10 or 20 values of μ , and we suggest choosing them uniformly on a log scale, from (say) 10^{-2} to 10^2 .

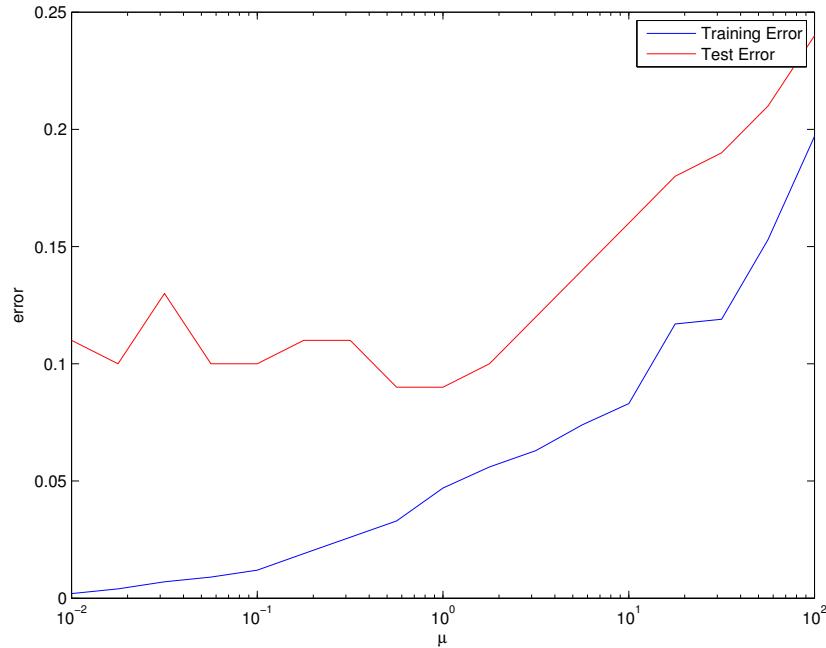
Solution.

- (a) The multi-label SVM problem is convex as stated. The variables are A and b . The only constraint, $\mathbf{1}^T b = 0$, is linear. The regularization term in the objective, $\mu \|A\|_F^2$, is convex quadratic. Let us justify that the loss function term $L(A, b)$ is convex, which we can do by showing each term,

$$\left(1 + \max_{k \neq y_i} f_k(x_i) - f_{y_i}(x_i) \right)_+,$$

is convex. Since $f_k(x_i)$ is linear in the variables, $\max_{k \neq y_i} f_k(x_i)$ is convex. The argument of $(\cdot)_+$ is a sum of a constant, a convex, and a linear function, and so is convex. The function $(\cdot)_+$ is convex and nondecreasing, so by the composition rule, the term above is convex.

- (b) The plot below shows the training and test error obtained for different values of μ . A reasonable value to choose would be around $\mu = 1$, although smaller values seem to work well too.



The following code solves the problem.

```

% multi-label support vector machine
multi_label_svm_data;

mus = 10.^(-2:0.25:2);
errorTrain = [];
errorTest = [];

for mu = mus
    cvx_begin quiet
        variables A(K, n) b(K)
        expressions L f(K, mTrain)
        % f(k, i) stores f_k(x_i)
        f = A*x + b*ones([1 mTrain]);
        L = 0;
        for k = 1:K
            % process all examples with y_i = k simultaneously
            ind = [1:k-1 k+1:K];
            L = L + sum(pos(1 + max(f(ind, y==k), [], 1) ...
                - f(k, y==k)));
        end
        minimize (L + mu*sum(sum(A.^2)))
        subject to
            sum(b) == 0;
    cvx_end

    [~, indTrain] = max(A*x + b*ones([1 mTrain]), [], 1);
    errorTrain(end+1) = sum(indTrain~=y)/mTrain;
    [~, indTest] = max(A*xtest + b*ones([1 mTest]), [], 1);
    errorTest(end+1) = sum(indTest~=ytest)/mTest;
end

% plot
clf;
semilogx(mus, errorTrain); hold on;
semilogx(mus, errorTest, 'r'); hold off;
xlabel('\mu'); ylabel('error');
legend('Training Error', 'Test Error');
print -depsc multi_label_svm.eps;

```

- 5.19 Colorization with total variation regularization.** A $m \times n$ color image is represented as three matrices of intensities $R, G, B \in \mathbf{R}^{m \times n}$, with entries in $[0, 1]$, representing the red, green, and blue pixel intensities, respectively. A color image is converted to a monochrome image, represented as one matrix $M \in \mathbf{R}^{m \times n}$, using

$$M = 0.299R + 0.587G + 0.114B.$$

(These weights come from different perceived brightness of the three primary colors.)

In *colorization*, we are given M , the monochrome version of an image, and the color values of *some* of the pixels; we are to guess its color version, *i.e.*, the matrices R, G, B . Of course that's a very underdetermined problem. A very simple technique is to minimize the total variation of (R, G, B) , defined as

$$\mathbf{tv}(R, G, B) = \sum_{i=1}^{m-1} \sum_{j=1}^{n-1} \left\| \begin{bmatrix} R_{ij} - R_{i,j+1} \\ G_{ij} - G_{i,j+1} \\ B_{ij} - B_{i,j+1} \\ R_{ij} - R_{i+1,j} \\ G_{ij} - G_{i+1,j} \\ B_{ij} - B_{i+1,j} \end{bmatrix} \right\|_2,$$

subject to consistency with the given monochrome image, the known ranges of the entries of (R, G, B) (*i.e.*, in $[0, 1]$), and the given color entries. Note that the sum above is of the norm of 6-vectors, and not the norm-squared. (The 6-vector is an approximation of the spatial gradient of (R, G, B) .)

Carry out this method on the data given in `image_colorization_data.*`. The file loads `flower.png` and provides the monochrome version of the image, M , along with vectors of known color intensities, `R_known`, `G_known`, and `B_known`, and `known_ind`, the indices of the pixels with known values. If `R` denotes the red channel of an image, then `R(known_ind)` returns the known red color intensities in Matlab, and `R[known_ind]` returns the same in Python and Julia. The file also creates an image, `flower_given.png`, that is monochrome, with the known pixels colored.

The `tv` function, invoked as `tv(R, G, B)`, gives the total variation. CVXPY has the `tv` function built-in, but CVX and CVX.jl do not, so we have provided the files `tv.m` and `tv.jl` which contain implementations for you to use.

In Python and Julia we have also provided the function `save_img(filename, R, G, B)` which writes the image defined by the matrices R, G, B , to the file `filename`. To view an image in Matlab use the `imshow` function.

The problem instance is a small image, 75×75 , so the solve time is reasonable, say, under ten seconds or so in CVX or CVXPY, and around 60 seconds in Julia.

Report your optimal objective value and, if you have access to a color printer, attach your reconstructed image. If you don't have access to a color printer, it's OK to just give the optimal objective value.

Solution.

Let R, G, B denote the matrices for the image, $R^{\text{known}}, G^{\text{known}}, B^{\text{known}}$ the matrices of known color values (only defined for some entries), and \mathcal{K} denote the indices of color values. Image colorization is then the following optimization problem:

$$\begin{aligned} & \text{minimize} && \mathbf{tv}(R, G, B) \\ & \text{subject to} && 0.299R + 0.587G + 0.114B = M \\ & && R_{ij} = R_{ij}^{\text{known}}, \quad (i, j) \in \mathcal{K} \\ & && G_{ij} = G_{ij}^{\text{known}}, \quad (i, j) \in \mathcal{K} \\ & && B_{ij} = B_{ij}^{\text{known}}, \quad (i, j) \in \mathcal{K} \\ & && 0 \leq R_{ij}, G_{ij}, B_{ij} \leq 1. \end{aligned}$$

The following Matlab code solves this problem.

```

image_colorization_data;
cvx_begin quiet
    variables R(m,n) G(m,n) B(m,n)
    minimize tv(R,G,B)
    subject to
        % grayscale reconstruction matches given grayscale
        0.299*R + 0.587*G + 0.114*B == M
        % colors match given colors
        R(known_ind) == R_known
        G(known_ind) == G_known
        B(known_ind) == B_known
        % colors in range
        R >= 0; G >= 0; B >= 0
        R <= 1; G <= 1; B <= 1
cvx_end
outim = cat(3,R,G,B);
imshow(outim);
imwrite(outim,'flower_reconstructed.png');

```

Here is the solution in Python.

```

import cvxpy as cvx
from image_colorization_data import *

R = cvx.Variable(m,n)
G = cvx.Variable(m,n)
B = cvx.Variable(m,n)
constraints = [
    0.299*R + 0.587*G + 0.114*B == M,
    R[known_ind] == R_known,
    G[known_ind] == G_known,
    B[known_ind] == B_known,
    0 <= R, 0 <= G, 0 <= B,
    1 >= R, 1 >= G, 1 >= B,
]
optval = cvx.Problem(cvx.Minimize(cvx.tv(R,G,B)), constraints).solve()
print(optval)
save_img('flower_reconstructed.png', R.value, G.value, B.value)

```

Here is the solution in Julia.

```

using Convex
include("image_colorization_data.jl");
include("tv.jl");

```

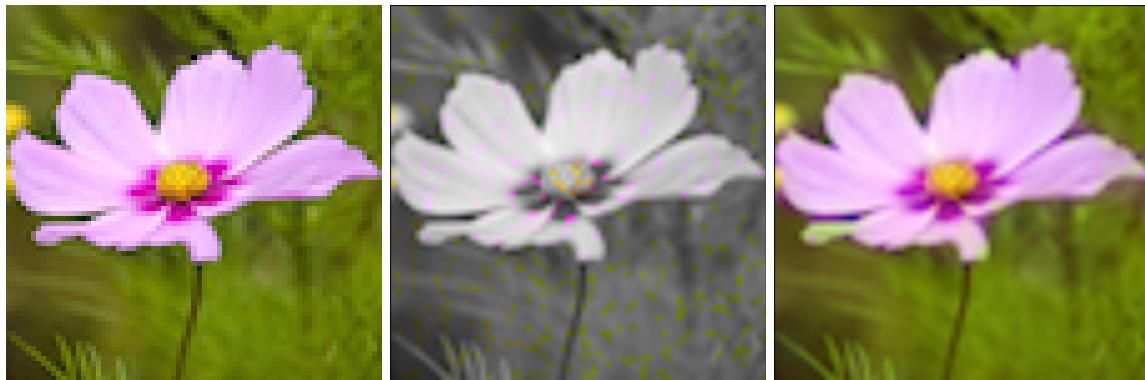
```

R = Variable(m,n);
G = Variable(m,n);
B = Variable(m,n);

constraints = [
    0.299*R + 0.587*G + 0.114*B == M;
    R[known_ind] == R_known;
    G[known_ind] == G_known;
    B[known_ind] == B_known;
    0 <= R; 0 <= G; 0 <= B;
    1 >= R; 1 >= G; 1 >= B;
];
problem = minimize(tv(R,G,B), constraints)
solve!(problem)
save_img("flower_reconstructed.png", R.value, G.value, B.value)

```

The results are shown below.



Original image on the left, image in monochrome with colored pixels shown in the middle, and reconstructed image on the right.

In Matlab our optimum objective value was 609.03. In Python it was 620.78. In Julia it was 615.00. (These values should be the same, of course.)

6 Statistical estimation

6.1 Maximum likelihood estimation of x and noise mean and covariance. Consider the maximum likelihood estimation problem with the linear measurement model

$$y_i = a_i^T x + v_i, \quad i = 1, \dots, m.$$

The vector $x \in \mathbf{R}^n$ is a vector of unknown parameters, y_i are the measurement values, and v_i are independent and identically distributed measurement errors.

In this problem we make the assumption that the *normalized* probability density function of the errors is given (normalized to have zero mean and unit variance), but not their mean and variance. In other words, the density of the measurement errors v_i is

$$p(z) = \frac{1}{\sigma} f\left(\frac{z - \mu}{\sigma}\right),$$

where f is a given, normalized density. The parameters μ and σ are the mean and standard deviation of the distribution p , and are not known.

The maximum likelihood estimates of x, μ, σ are the maximizers of the log-likelihood function

$$\sum_{i=1}^m \log p(y_i - a_i^T x) = -m \log \sigma + \sum_{i=1}^m \log f\left(\frac{y_i - a_i^T x - \mu}{\sigma}\right),$$

where y is the observed value. Show that if f is log-concave, then the maximum likelihood estimates of x, μ, σ can be determined by solving a convex optimization problem.

Solution. With a change of variables

$$z = (1/\sigma)x, \quad u = \mu/\sigma, \quad t = 1/\sigma,$$

the problem reduces to

$$\text{maximize } m \log t + \sum_{i=1}^m \log f(y_i t - a_i^T z - u),$$

which is a convex optimization problem since the objective is a concave function of (z, u, t) . We recover optimal values of x, μ , and σ using

$$\sigma = 1/t, \quad \mu = u/t, \quad x = (1/t)z.$$

6.2 Mean and covariance estimation with conditional independence constraints. Let $X \in \mathbf{R}^n$ be a Gaussian random variable with density

$$p(x) = \frac{1}{(2\pi)^{n/2} (\det S)^{1/2}} \exp(-(x - a)^T S^{-1}(x - a)/2).$$

The conditional density of a subvector $(X_i, X_j) \in \mathbf{R}^2$ of X , given the remaining variables, is also Gaussian, and its covariance matrix R_{ij} is equal to the Schur complement of the 2×2 submatrix

$$\begin{bmatrix} S_{ii} & S_{ij} \\ S_{ij} & S_{jj} \end{bmatrix}$$

in the covariance matrix S . The variables X_i, X_j are called *conditionally independent* if the covariance matrix R_{ij} of their conditional distribution is diagonal.

Formulate the following problem as a convex optimization problem. We are given N independent samples $y_1, \dots, y_N \in \mathbf{R}^n$ of X . We are also given a list $\mathcal{N} \subseteq \{1, \dots, n\} \times \{1, \dots, n\}$ of pairs of conditionally independent variables: $(i, j) \in \mathcal{N}$ means X_i and X_j are conditionally independent. The problem is to compute the maximum likelihood estimate of the mean a and the covariance matrix S , subject to the constraint that X_i and X_j are conditionally independent for $(i, j) \in \mathcal{N}$.

Solution. The log-likelihood function is

$$\begin{aligned} l(S, a) &= -(Nn/2) \log(2\pi) - (N/2) \log \det S - (1/2) \sum_{k=1}^N (y_k - a)^T S^{-1} (y_k - a) \\ &= \frac{N}{2} \left(-n \log(2\pi) - \log \det S - \text{tr}(S^{-1}Y) - (a - \mu)^T S^{-1} (a - \mu) \right) \end{aligned}$$

where μ and Y are the sample mean and covariance:

$$\mu = \frac{1}{N} \sum_{k=1}^N y_k, \quad Y = \frac{1}{N} \sum_{k=1}^N (y_k - \mu)(y_k - \mu)^T.$$

Note that the inverse of the conditional covariance matrix R_{ij}^{-1} is the 2×2 submatrix of S^{-1} formed by rows and columns i and j . The variables i and j are conditionally independent of $(S^{-1})_{ij} = 0$.

The ML problem is therefore

$$\begin{aligned} &\text{maximize} && l(S, a) \\ &\text{subject to} && (S^{-1})_{ij} = 0, \quad (i, j) \in \mathcal{N}. \end{aligned}$$

The optimal a is clearly $a = \mu$, so it remains to solve

$$\begin{aligned} &\text{maximize} && \log \det S^{-1} - \text{tr}(S^{-1}Y) \\ &\text{subject to} && (S^{-1})_{ij} = 0, \quad (i, j) \in \mathcal{N} \end{aligned}$$

which is convex in S^{-1} .

6.3 Maximum likelihood estimation for exponential family. A probability distribution or density on a set \mathcal{D} , parametrized by $\theta \in \mathbf{R}^n$, is called an *exponential family* if it has the form

$$p_\theta(x) = a(\theta) \exp(\theta^T c(x)),$$

for $x \in \mathcal{D}$, where $c : \mathcal{D} \rightarrow \mathbf{R}^n$, and $a(\theta)$ is a normalizing function. Here we interpret $p_\theta(x)$ as a density function when \mathcal{D} is a continuous set, and a probability distribution when \mathcal{D} is discrete. Thus we have

$$a(\theta) = \left(\int_{\mathcal{D}} \exp(\theta^T c(x)) dx \right)^{-1}$$

when p_θ is a density, and

$$a(\theta) = \left(\sum_{x \in \mathcal{D}} \exp(\theta^T c(x)) \right)^{-1}$$

when p_θ represents a distribution. We consider only values of θ for which the integral or sum above is finite. Many families of distributions have this form, for appropriate choice of the parameter θ and function c .

- (a) When $c(x) = x$ and $\mathcal{D} = \mathbf{R}_+^n$, what is the associated family of densities? What is the set of valid values of θ ?
- (b) Consider the case with $\mathcal{D} = \{0, 1\}$, with $c(0) = 0$, $c(1) = 1$. What is the associated exponential family of distributions? What are the valid values of the parameter $\theta \in \mathbf{R}$?
- (c) Explain how to represent the normal family $\mathcal{N}(\mu, \Sigma)$ as an exponential family. *Hint.* Use parameter $(z, Y) = (\Sigma^{-1}\mu, \Sigma^{-1})$. With this parameter, $\theta^T c(x)$ has the form $z^T c_1(x) + \mathbf{tr} Y C_2(x)$, where $C_2(x) \in \mathbf{S}^n$.
- (d) *Log-likelihood function.* Show that for any $x \in \mathcal{D}$, the log-likelihood function $\log p_\theta(x)$ is concave in θ . This means that maximum-likelihood estimation for an exponential family leads to a convex optimization problem. You don't have to give a formal proof of concavity of $\log p_\theta(x)$ in the general case: You can just consider the case when \mathcal{D} is finite, and state that the other cases (discrete but infinite \mathcal{D} , continuous \mathcal{D}) can be handled by taking limits of finite sums.
- (e) *Optimality condition for ML estimation.* Let $\ell_\theta(x_1, \dots, x_K)$ be the log-likelihood function for K IID samples, x_1, \dots, x_k , from the distribution or density p_θ . Assuming $\log p_\theta$ is differentiable in θ , show that

$$(1/K) \nabla_\theta \ell_\theta(x_1, \dots, x_K) = \frac{1}{K} \sum_{i=1}^K c(x_i) - \mathbf{E}_\theta c(x).$$

(The subscript under \mathbf{E} means the expectation under the distribution or density p_θ .)

Interpretation. The ML estimate of θ is characterized by the empirical mean of $c(x)$ being equal to the expected value of $c(x)$, under the density or distribution p_θ . (We assume here that the maximizer of ℓ is characterized by the gradient vanishing.)

Solution.

- (a) We need $\theta \prec 0$ for the integral to converge, in which case we have

$$\int_{\mathbf{R}_+^n} \exp(\theta^T x) dx = \frac{1}{\prod_{i=1}^n (-\theta_i)},$$

so $a(\theta) = \prod_{i=1}^n (-\theta_i)$. In this case the distribution is that of n independent exponentially distributed variables, with means $-1/\theta_i$.

- (b) We have

$$p_\theta(0) = \frac{1}{\exp \theta + 1}, p_\theta(1) = \frac{\exp \theta}{\exp \theta + 1},$$

which we recognize as the Bernoulli distribution. Any value of $\theta \in \mathbf{R}$ is valid.

- (c) We write the $\mathcal{N}(\mu, \Sigma)$ density as

$$p(x) = a \exp \left(-(x - \mu)^T \Sigma^{-1} (x - \mu)/2 \right),$$

where a is the normalizing constant,

$$a = (2\pi)^{-n/2} \det \Sigma^{-1/2}.$$

We write this in the form

$$p(x) = \tilde{a} \exp \left(z^T c_1(x) + \mathbf{tr} Y C_2(x) \right),$$

where $Y = \Sigma^{-1}$ and $z = \Sigma^{-1}\mu$,

$$c_1(x) = x, \quad C_2(x) = -(1/2)xx^T,$$

and \tilde{a} is the normalizing constant

$$\tilde{a} = a \exp(-\mu^T \Sigma^{-1} \mu / 2) = (2\pi)^{-n/2} \det Y^{1/2} \exp(-z^T Y^{-1} z / 2).$$

The set of valid values of z and Y is simply $\mathbf{R}^n \times \mathbf{S}_{++}^n$. (Note that the mapping between (μ, Σ) and (z, Y) is a bijection.)

We didn't ask you to find it, but the log-likelihood function is

$$\log p_\theta(x) = -(n/2) \log(2\pi) + (1/2) \log \det Y - z^T Y^{-1} z / 2 + z^T x - x^T Y^{-1} x / 2,$$

which is indeed concave (as part (d) tells us it must be).

- (d) We'll consider the case when \mathcal{D} is finite. From the definition of $p_\theta(x)$ we have

$$\begin{aligned} \log p_\theta(x) &= \log a(\theta) + \theta^T c(x) \\ &= -\log \left(\sum_{x \in \mathcal{D}} \exp(\theta^T c(x)) \right) + \theta^T c(x). \end{aligned}$$

The second term is affine in θ ; the first is concave, since it is the negative log-sum-exp function composed with the affine function of theta $\theta^T c(x)$.

We can show that $f(\theta) = \log p_\theta(x)$ is concave, in the more general case of a density, by computing the Hessian. Defining $e(\theta, x) = \exp(\theta^T c(x))$, the second partial derivatives are

$$\frac{\partial^2 f}{\partial \theta_i \partial \theta_j} = \frac{(\int_{\mathcal{D}} c_i(x)e(\theta, x) dx)(\int_{\mathcal{D}} c_j(x)e(\theta, x) dx) - (\int_{\mathcal{D}} c_i(x)c_j(x)e(\theta, x) dx)(\int_{\mathcal{D}} e(\theta, x) dx)}{(\int_{\mathcal{D}} e(\theta, x) dx)^2},$$

where $c(x) = (c_1(x), \dots, c_m(x))$. For any vector $v \in \mathbf{R}^m$, this gives

$$v^T (\nabla^2 f(\theta)) v = \frac{(\int_{\mathcal{D}} (\sum_i v_i c_i(x))e(\theta, x) dx)^2 - (\int_{\mathcal{D}} (\sum_i v_i c_i(x))^2 e(\theta, x) dx)(\int_{\mathcal{D}} e(\theta, x) dx)}{(\int_{\mathcal{D}} e(\theta, x) dx)^2}.$$

From the Cauchy-Schwarz inequality for functions,

$$\left(\int g(x)h(x) dx \right)^2 \leq \left(\int (g(x))^2 dx \right) \left(\int (h(x))^2 dx \right),$$

with $g(x) = (\sum_i v_i c_i(x)) \sqrt{e(\theta, x)}$ and $h(x) = \sqrt{e(\theta, x)}$, we get $v^T (\nabla^2 f(\theta)) v \leq 0$ for all v . The Hessian is negative semidefinite, so $\log p_\theta(x)$ is concave.

- (e) It's a simple calculation to get the gradient of $\log p_\theta$ (especially when you ignore issues such as whether or not it exists, which we plan to do). The log-likelihood function is

$$\ell_\theta(x_1, \dots, x_K) = \sum_{i=1}^K \log p_\theta(x_i),$$

so we have

$$\begin{aligned}\nabla_\theta \ell_\theta(x_1, \dots, x_K) &= \sum_{i=1}^K \nabla_\theta \left(c(x_i)^T \theta - \log \left(\sum_{x \in \mathcal{D}} \exp c(x)^T \theta \right) \right) \\ &= \sum_{i=1}^K c(x_i) - K \frac{\sum_{x \in \mathcal{D}} (\exp c(x)^T \theta) c(x)}{\sum_{x \in \mathcal{D}} \exp c(x)^T \theta} \\ &= \sum_{i=1}^K c(x_i) - K \mathbf{E}_\theta c(x).\end{aligned}$$

6.4 Maximum likelihood prediction of team ability. A set of n teams compete in a tournament. We model each team's ability by a number $a_j \in [0, 1]$, $j = 1, \dots, n$. When teams j and k play each other, the probability that team j wins is equal to $\text{prob}(a_j - a_k + v > 0)$, where $v \sim \mathcal{N}(0, \sigma^2)$.

You are given the outcome of m past games. These are organized as

$$(j^{(i)}, k^{(i)}, y^{(i)}), \quad i = 1, \dots, m,$$

meaning that game i was played between teams $j^{(i)}$ and $k^{(i)}$; $y^{(i)} = 1$ means that team $j^{(i)}$ won, while $y^{(i)} = -1$ means that team $k^{(i)}$ won. (We assume there are no ties.)

- (a) Formulate the problem of finding the maximum likelihood estimate of team abilities, $\hat{a} \in \mathbf{R}^n$, given the outcomes, as a convex optimization problem. You will find the *game incidence matrix* $A \in \mathbf{R}^{m \times n}$, defined as

$$A_{il} = \begin{cases} y^{(i)} & l = j^{(i)} \\ -y^{(i)} & l = k^{(i)} \\ 0 & \text{otherwise,} \end{cases}$$

useful.

The prior constraints $\hat{a}_i \in [0, 1]$ should be included in the problem formulation. Also, we note that if a constant is added to all team abilities, there is no change in the probabilities of game outcomes. This means that \hat{a} is determined only up to a constant, like a potential. But this doesn't affect the ML estimation problem, or any subsequent predictions made using the estimated parameters.

- (b) Find \hat{a} for the team data given in `team_data.m`, in the matrix `train`. (This matrix gives the outcomes for a tournament in which each team plays each other team once.) You may find the CVX function `log_normcdf` helpful for this problem.

You can form A using the commands

```
A = sparse(1:m,train(:,1),train(:,3),m,n) + ...
sparse(1:m,train(:,2),-train(:,3),m,n);
```

- (c) Use the maximum likelihood estimate \hat{a} found in part (b) to predict the outcomes of next year's tournament games, given in the matrix `test`, using $\hat{y}^{(i)} = \text{sign}(\hat{a}_{j^{(i)}} - \hat{a}_{k^{(i)}})$. Compare these predictions with the actual outcomes, given in the third column of `test`. Give the fraction of correctly predicted outcomes.

The games played in `train` and `test` are the same, so another, simpler method for predicting the outcomes in `test` it to just assume the team that won last year's match will also win this year's match. Give the percentage of correctly predicted outcomes using this simple method.

Solution.

- (a) The likelihood of the outcomes y given a is

$$p(y|a) = \prod_{i=1,\dots,n} \Phi\left(\frac{1}{\sigma} y^{(i)}(a_{j^{(i)}} - a_{k^{(i)}})\right),$$

where Φ is the cumulative distribution of the standard normal. The log-likelihood function is therefore

$$l(a) = \log p(y|a) = \sum_i \log \Phi((1/\sigma)(Aa)_i).$$

This is a concave function.

The maximum likelihood estimate \hat{a} is any solution of

$$\begin{aligned} & \text{maximize} && l(a) \\ & \text{subject to} && 0 \preceq a \preceq 1. \end{aligned}$$

This is a convex optimization problem since the objective, which is maximized, is concave, and the constraints are $2n$ linear inequalities.

- (b) The following code solves the problem

```
% Form adjacency matrix
team_data
A1 = sparse(1:m,train(:,1),train(:,3),m,n);
A2 = sparse(1:m,train(:,2),-train(:,3),m,n);
A = A1+A2;

% Estimate abilities
cvx_begin
    variable a_hat(n)
    minimize(-sum(log_normcdf(A*a_hat/sigma)))
    subject to
        a_hat >= 0
        a_hat <= 1
cvx_end
```

Using this code we get that $\hat{a} = (1.0, 0.0, 0.68, 0.37, 0.79, 0.58, 0.38, 0.09, 0.67, 0.58)$.

- (c) The following code is used to predict the outcomes in the test set

```

% Estimate errors in test set
A1 = sparse(1:m_test,test(:,1),1,m_test,n);
A2 = sparse(1:m_test,test(:,2),-1,m_test,n);
A_test = A1+A2;
res = sign(A_test*a_hat);
Pml = 1-length(find(res-test(:,3)))/m_test
Ply = 1-length(find(train(:,3)-test(:,3)))/m_test

```

The maximum likelihood estimate gives a correct prediction of 86.7% of the games in `test`. On the other hand, 75.6% of the games in `test` have the same outcome as the games in `train`.

6.5 *Estimating a vector with unknown measurement nonlinearity.* (A specific instance of exercise 7.9 in *Convex Optimization*.) We want to estimate a vector $x \in \mathbf{R}^n$, given some measurements

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

Here $a_i \in \mathbf{R}^n$ are known, v_i are IID $\mathcal{N}(0, \sigma^2)$ random noises, and $\phi : \mathbf{R} \rightarrow \mathbf{R}$ 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 $z_i = \phi^{-1}(y_i)$, $i = 1, \dots, m$. So by estimating ϕ we really mean estimating the m numbers z_1, \dots, 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, \dots, 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, \dots, a_m^T . Give \hat{x}_{ml} , the maximum likelihood estimate of x . Plot your estimated function $\hat{\phi}_{\text{ml}}$. (You can do this by plotting $(\hat{z}_{\text{ml}})_i$ versus y_i , with y_i on the vertical axis and $(\hat{z}_{\text{ml}})_i$ on the horizontal axis.)

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

Solution.

- (a) We can write the measurement model as

$$\phi^{-1}(y_i) = a_i^T x + v_i, \quad i = 1, \dots, m.$$

The function ϕ^{-1} is unknown, but it has derivatives between $1/\beta$ and $1/\alpha$. Therefore $z_i = \phi^{-1}(y_i)$ and y_i must satisfy the inequalities

$$\frac{y_{i+1} - y_i}{\beta} \leq z_{i+1} - z_i \leq \frac{y_{i+1} - y_i}{\alpha}, \quad i = 1, \dots, m-1,$$

if we assume that the points are sorted with y_i in increasing order. Conversely, if z and y satisfy these inequalities, then there exists a nonlinear function ϕ with $y_i = \phi(z_i)$, $i = 1, \dots, m$, and with derivatives between α and β (for example, a piecewise-linear function that interpolates the points). Therefore, as suggested in the problem statement, we can use z_1, \dots, z_m as parameters instead of ϕ .

The log-likelihood function is

$$l(z, x) = -\frac{1}{2\sigma^2} \sum_{i=1}^m (z_i - a_i^T x)^2 - m \log(\sigma\sqrt{2\pi}).$$

Thus to find a maximum likelihood estimate of x and z one solves the problem

$$\begin{aligned} & \text{minimize} && \sum_{i=1}^m (z_i - a_i^T x)^2 \\ & \text{subject to} && (y_{i+1} - y_i)/\beta \leq z_{i+1} - z_i \leq (y_{i+1} - y_i)/\alpha, \quad i = 1, \dots, m-1. \end{aligned}$$

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

- (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);

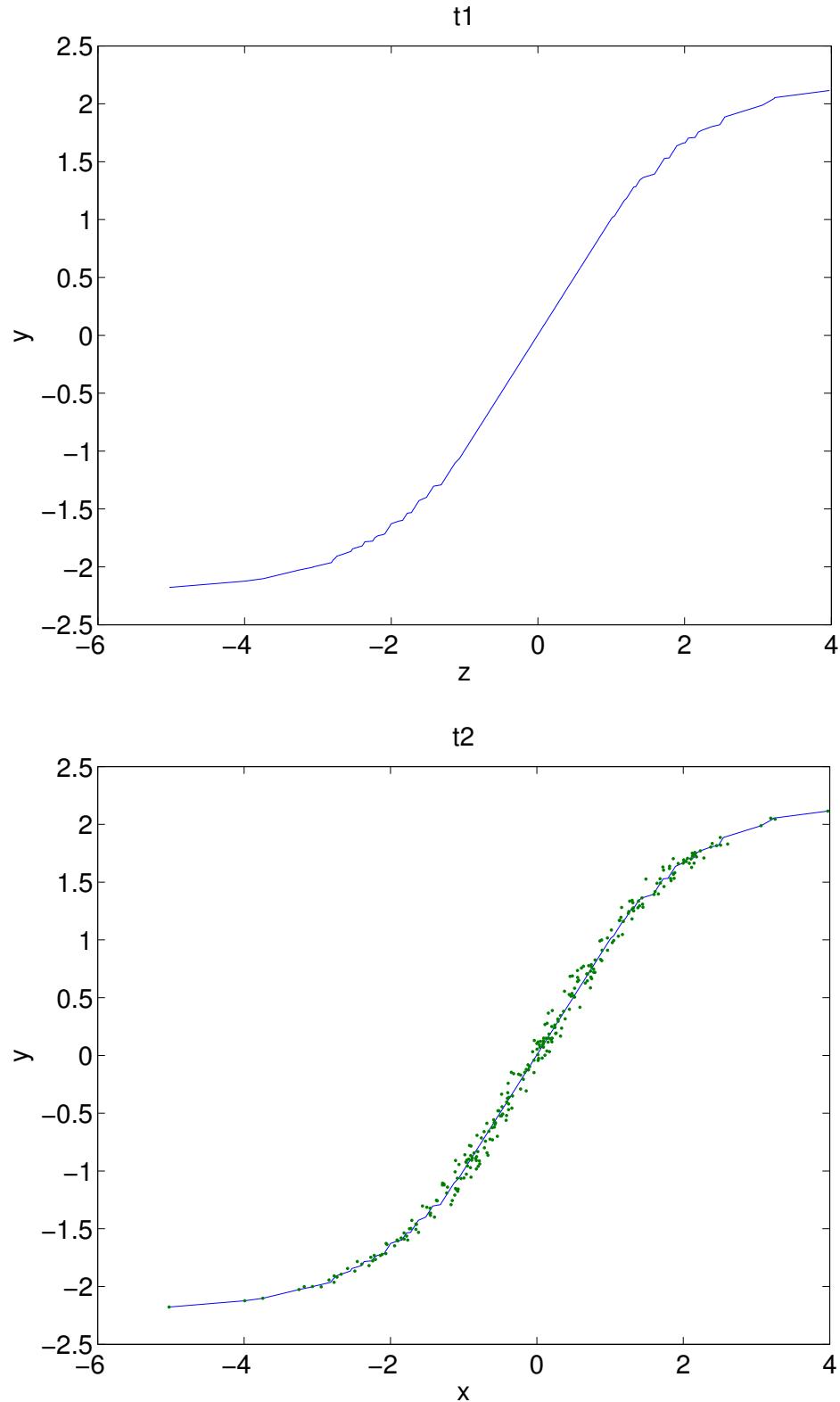
cvx_begin
    variable x(n);
    variable z(m);
    minimize(norm(z-A*x));
    subject to
        1/beta*B*y<=B*z;
        B*z<=1/alpha*B*y;
cvx_end

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

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

The estimated x is $x = (0.4819, -0.4657, 0.9364, 0.9297)$.

The first figure shows the estimate function ϕ . The second figure shows ϕ and the data points $a_i^T x, y_i$.



6.6 Maximum likelihood estimation of an increasing nonnegative signal. We wish to estimate a scalar

signal $x(t)$, for $t = 1, 2, \dots, N$, which is known to be nonnegative and monotonically nondecreasing:

$$0 \leq x(1) \leq x(2) \leq \dots \leq x(N).$$

This occurs in many practical problems. For example, $x(t)$ might be a measure of wear or deterioration, that can only get worse, or stay the same, as time t increases. We are also given that $x(t) = 0$ for $t \leq 0$.

We are given a noise-corrupted moving average of x , given by

$$y(t) = \sum_{\tau=1}^k h(\tau)x(t-\tau) + v(t), \quad t = 2, \dots, N+1,$$

where $v(t)$ are independent $\mathcal{N}(0, 1)$ random variables.

- (a) Show how to formulate the problem of finding the maximum likelihood estimate of x , given y , taking into account the prior assumption that x is nonnegative and monotonically nondecreasing, as a convex optimization problem. Be sure to indicate what the problem variables are, and what the problem data are.
- (b) We now consider a specific instance of the problem, with problem data (*i.e.*, N , k , h , and y) given in the file `ml_estim_incr_signal_data.*`. (This file contains the true signal `xtrue`, which of course you cannot use in creating your estimate.) Find the maximum likelihood estimate \hat{x}_{ml} , and plot it, along with the true signal. Also find and plot the maximum likelihood estimate $\hat{x}_{\text{ml,free}}$ *not taking into account the signal nonnegativity and monotonicity*.

Hints.

- Matlab: The function `conv` (convolution) is overloaded to work with CVX.
- Python: Numpy has a function `convolve` which performs convolution. CVXPY has `conv` which does the same thing for variables.
- Julia: The function `conv` is overloaded to work with Convex.jl.

Solution.

- (a) To simplify our notation, we let the signal \hat{y}_x be the noiseless moving average of the signal x . That is,

$$\hat{y}_x(t) = \sum_{\tau=1}^k h(\tau)x(t-\tau), \quad t = 2, \dots, N+1.$$

Note that \hat{y}_x is a linear function of x .

The nonnegativity and monotonicity constraint on x can be expressed as a set of linear inequalities,

$$x(1) \geq 0, \quad x(1) \leq x(2), \quad \dots \quad x(N-1) \leq x(N).$$

Now we turn to the maximum likelihood problem. The likelihood function is

$$\prod_{t=2}^{N+1} p(y(t) - \hat{y}_x(t)),$$

where p is the density function of a $\mathcal{N}(0, 1)$ random variable. The negative log-likelihood function has the form

$$\alpha + \beta \|\hat{y}_x - y\|_2^2,$$

where α is a constant, and β is a positive constant. Thus, the ML estimate is found by minimizing the quadratic objective $\|\hat{y}_x - y\|_2^2$. The ML estimate when we *do not* take into account signal nonnegativity and monotonicity can be found by solving a least-squares problem,

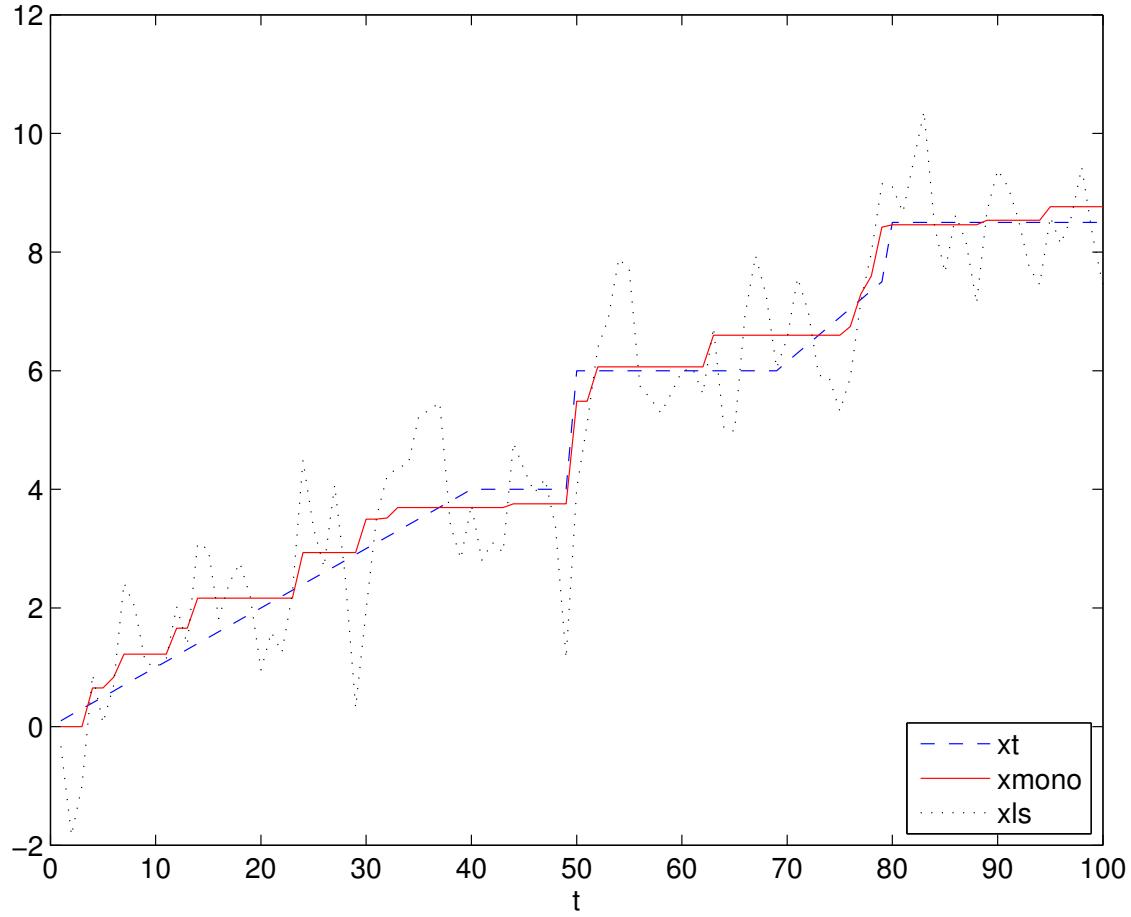
$$\hat{x}_{\text{ml,free}} = \operatorname{argmin}_x \|\hat{y}_x - y\|_2^2.$$

Since convolution is an invertible linear operation, to get the ML estimate without the monotonicity constraint, we simply apply deconvolution. Since the number of measurements and variables to estimate are the same (N), there is no smoothing effect to reduce noise, and we can expect the deconvolved estimate to be poor.

With the prior assumption that the signal x is nonnegative and monotonically nondecreasing we can find the ML estimate \hat{x}_{ml} by solving the QP

$$\begin{aligned} & \text{minimize} && \|\hat{y}_x - y\|_2^2 \\ & \text{subject to} && x(1) \geq 0 \\ & && x(t) \leq x(t+1) \quad t = 1, \dots, N-1. \end{aligned}$$

- (b) The ML estimate for the given problem instance, with and without the assumption of nonnegativity and monotonicity, is plotted below. We've also plotted the true signal x . We observe that the ML estimate \hat{x}_{ml} , which takes into consideration nonnegativity and monotonicity, is a much better estimate of signal x than the simple unconstrained solution $\hat{x}_{\text{ml,free}}$ obtained via deconvolution.



Plots for the other languages are shown below.

The following Matlab code computes the ML solutions for the constrained and unconstrained estimation problems.

```
% ML estimation of increasing nonnegative signal
% problem data
ml_estim_incr_signal_data;

% maximum likelihood estimation with no monotonicity taken in to account
% can be solved analytically
cvx_begin
    variable xls(N)
    yhat = conv(h,xls);      % estimated output
    % yhat is truncated to match problem description
    minimize (sum_square(yhat(1:end-3) - y))
cvx_end

% monotonic and non-negative signal estimation
cvx_begin
```

```

variable xmono(N)
yhat = conv(h,xmono); % estimated output
minimize (sum_square(yhat(1:end-3) - y))
subject to
xmono(1) >= 0;
xmono(1:N-1) <= xmono(2:N);
cvx_end

t = 1:N;
figure; set(gca, 'FontSize',12);
plot(t,xtrue,'--',t,xmono,'r',t,xls,'k:');
xlabel('t'); legend('xt','xmono','xls','Location','SouthEast');
%print -depsc ml_estim_incr_signal_plot

```

The following code implements the same solution in Python

```

import cvxpy as cvx
import matplotlib.pyplot as plt

# Load data file, gives us N and Y
from ml_estim_incr_signal_data import *

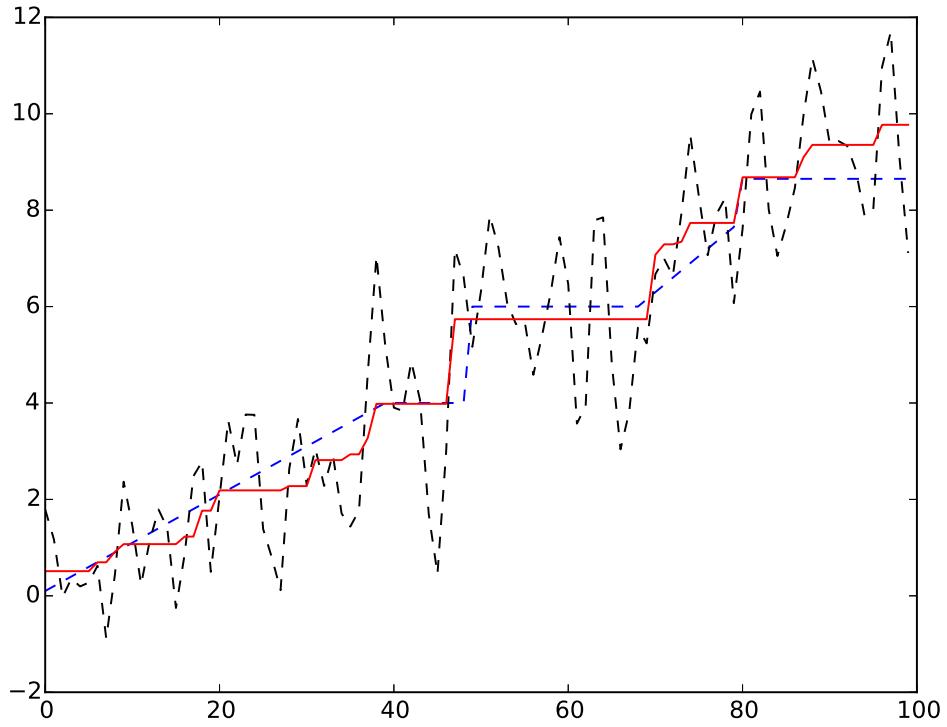
# Estimate signal without constraints
xls = cvx.Variable(N)
yhat = cvx.conv(h,xls)
error = cvx.sum_squares(yhat[0:-3] - y)
prob = cvx.Problem(cvx.Minimize(error))
prob.solve()

xmono = cvx.Variable(N)
yhat = cvx.conv(h,xmono)
error = cvx.sum_squares(yhat[0:-3] - y)
constraints = [xmono[0] >= 0]
constraints += [xmono[0:-1] <= xmono[1:]]
prob2 = cvx.Problem(cvx.Minimize(error),constraints)
prob2.solve()

fig = plt.figure()
t = np.arange(N)
plt.plot(t,xtrue,'b--',t,xls.value,'k--',t,xmono.value,'r')
plt.show()
plt.savefig('ml_estim_incr_signal_plot.eps',format='eps')

```

The plot below shows the results from the Python implementation.



In Julia, the code is written as follows.

```
# ML estimation of increasing nonnegative signal
# problem data
include("ml_estim_incr_signal_data.jl");

using Convex

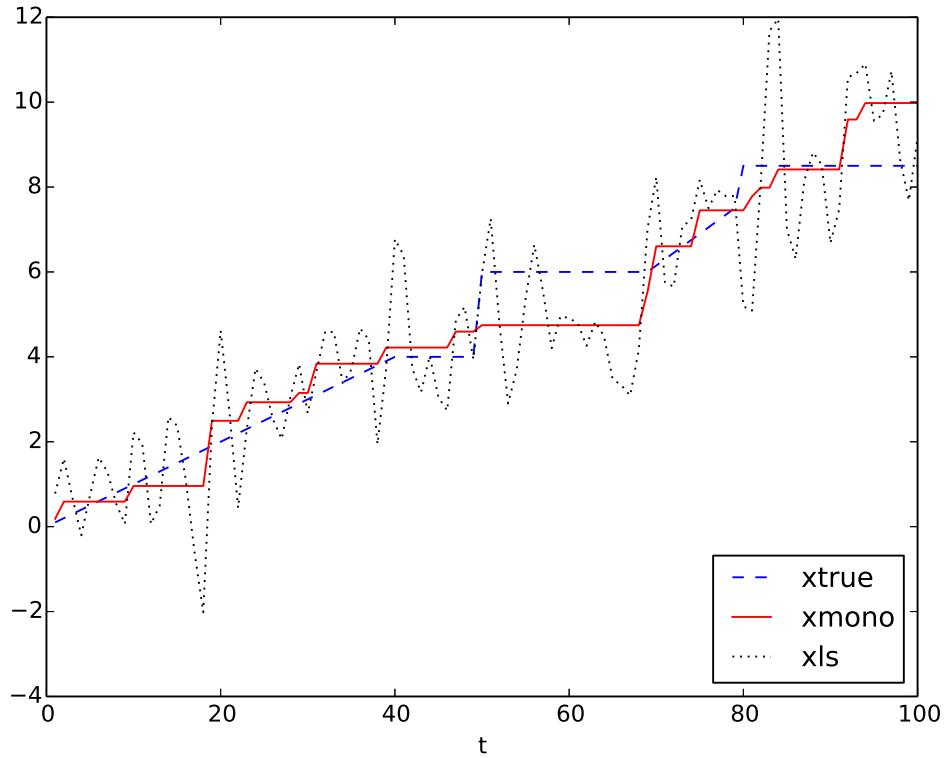
# maximum likelihood estimation with no monotonicity taken in to account
# can be solved analytically
xls = Variable(N);
yhat = conv(h, xls); # estimated output
# yhat is truncated to match problem description
p = minimize (sum_squares(yhat[1:end-3] - y));
solve!(p);

# monotonic and non-negative signal estimation
xmono = Variable(N);
yhat = conv(h, xmono); # estimated output
constraints = [xmono[1] >= 0, xmono[1:N-1] <= xmono[2:N]]
p = minimize (sum_squares(yhat[1:end-3] - y), constraints);
solve!(p);
```

```

using PyPlot
t = 1:N;
figure();
hold(true);
plot(t, xtrue, "b--", label="xtrue");
plot(t, xmono.value, "r", label="xmono");
plot(t, xls.value, "k:", label="xls");
xlabel("t");
legend(loc=4);
savefig("ml_estim_incr_signal.eps");

```



6.7 Relaxed and discrete A-optimal experiment design. This problem concerns the A -optimal experiment design problem, described on page 387, with data generated as follows.

```

n = 5; % dimension of parameters to be estimated
p = 20; % number of available types of measurements
m = 30; % total number of measurements to be carried out
randn('state', 0);
V=randn(n,p); % columns are vi, the possible measurement vectors

```

Solve the relaxed A -optimal experiment design problem,

$$\begin{aligned} & \text{minimize} && (1/m) \operatorname{tr} \left(\sum_{i=1}^p \lambda_i v_i v_i^T \right)^{-1} \\ & \text{subject to} && \mathbf{1}^T \lambda = 1, \quad \lambda \succeq 0, \end{aligned}$$

with variable $\lambda \in \mathbf{R}^p$. Find the optimal point λ^* and the associated optimal value of the relaxed problem. This optimal value is a lower bound on the optimal value of the discrete A -optimal experiment design problem,

$$\begin{aligned} & \text{minimize} && \operatorname{tr} \left(\sum_{i=1}^p m_i v_i v_i^T \right)^{-1} \\ & \text{subject to} && m_1 + \cdots + m_p = m, \quad m_i \in \{0, \dots, m\}, \quad i = 1, \dots, p, \end{aligned}$$

with variables m_1, \dots, m_p . To get a suboptimal point for this discrete problem, round the entries in $m\lambda^*$ to obtain integers \hat{m}_i . If needed, adjust these by hand or some other method to ensure that they sum to m , and compute the objective value obtained. This is, of course, an upper bound on the optimal value of the discrete problem. Give the gap between this upper bound and the lower bound obtained from the relaxed problem. Note that the two objective values can be interpreted as mean-square estimation error $\mathbf{E} \|\hat{x} - x\|_2^2$.

Solution. The objective of the relaxed problem is convex, so it is a convex problem. Expressing it in CVX requires a little work. We'd like to write the objective as

```
minimize ((1/m)*trace(inv(V*diag(lambda)*V')))
```

but this won't work, because CVX doesn't know about matrix convex functions. Instead, we can express the objective as a sum of matrix fractional functions,

$$\begin{aligned} & \text{minimize} && (1/m) \sum_{k=1}^n e_k^T \left(\sum_{i=1}^p \lambda_i v_i v_i^T \right)^{-1} e_k \\ & \text{subject to} && \mathbf{1}^T \lambda = 1, \quad \lambda \succeq 0. \end{aligned}$$

where $e_k \in \mathbf{R}^n$ is the k th unit vector. (Note that e is defined in exercise 6.9 as the estimation error vector, so e_k could also mean the k th entry of the error vector. But here, clearly, e_k is k th unit vector.)

We can express this in CVX using the function `matrix_frac`. The following code solves the problem.

```
n = 5; % dimension
p = 20; % number of available types of measurements
m = 30; % total number of measurements to be carried out
randn('state', 0);
V=randn(n,p); % columns are vi, the possible measurement vectors

cvx_begin
    variable lambda(p)
    obj = 0;
    for k=1:n
        ek = zeros(n,1);
        for i=1:p
            ek = ek + lambda(i)*V(:,i);
        end
        obj = obj + ek'*inv(ek*V*lambda*V'*ek);
    end
    obj = obj/m;
    subject to
        1' * lambda == 1;
        lambda >= 0;
    end
cvx_end
```

```

ek(k)=1;
obj = obj + (1/m)*matrix_frac(ek,V*diag(lambda)*V');
end
minimize( obj )
subject to
    sum(lambda) == 1
    lambda >= 0
cvx_end

lower_bound = cvx_optval

t = -0.00; % small offset chosen by hand to make rounding work out.
% for this problem data, none is needed!
m_rnd = pos(round(m*lambda+t));
sum(m_rnd) % should be == m

% now find objective value of rounded experiment design
upper_bound = trace(inv(V*diag(m_rnd/m)*V'))/m
gap = upper_bound-lower_bound
rel_gap = gap/lower_bound

```

For this problem instance, simple rounding yielded \hat{m}_i that summed to $m = 30$, so no adjustment of the rounded values is needed. The lower bound is 0.2481; the upper bound is 0.2483. The gap is 0.00023, which is around 0.1%.

What this means is this: We have found a choice of 30 measurements, each one from the set of 20 possible measurements, that yields a mean-square estimation error $\mathbf{E} \|\hat{x} - x\|_2^2 = 0.2483$. We do not know whether this is the optimal choice of 30 measurements. But we do know that this choice is no more than 0.1% suboptimal; the optimal choice can achieve a mean-square error that is no smaller than 0.2481. Our experiment design is, if not optimal, very nearly optimal. (In fact, it is very likely to be optimal.)

6.8 Optimal detector design. We adopt here the notation of §7.3 of the book. Explain how to design a (possibly randomized) detector that minimizes the worst-case probability of our estimate being off by more than one,

$$P_{\text{wc}} = \max_{\theta} \mathbf{prob}(|\hat{\theta} - \theta| \geq 2).$$

(The probability above is under the distribution associated with θ .)

Carry out your method for the problem instance with data in `off_by_one_det_data.m`. Give the optimal detection probability matrix D . Compare the optimal worst-case probability P_{wc}^* with the worst-case probability $P_{\text{wc}}^{\text{ml}}$ obtained using a maximum-likelihood detector.

Solution. For $\theta = j$ we have

$$\mathbf{prob}(|\hat{\theta} - \theta| \geq 2) = \sum_{|i-j| \geq 2} D_{ij} = 1 - D_{j+1,j} - D_{jj} - D_{j-1,j},$$

where we interpret D_{ij} as zero for $i = 0$ and $i = m + 1$. Therefore we find our detector by solving the problem

$$\begin{aligned} & \text{minimize} \quad \max_j \left(\sum_{|i-j| \geq 2} D_{ij} \right) \\ & \text{subject to} \quad D = [t_1 \cdots t_n]P \\ & \quad t_k \succeq 0, \quad \mathbf{1}^T t_k = 1, \quad k = 1, \dots, n, \end{aligned}$$

with variables t_1, \dots, t_n . This problem is evidently convex, since the constraints are linear equalities and inequalities, and the objective is a piecewise linear convex function.

The optimal detection probability matrix for the given problem instance is

$$D = \begin{bmatrix} 0.2466 & 0.2816 & 0.1616 & 0.0911 & 0.1102 \\ 0.4816 & 0.3855 & 0.4761 & 0.1807 & 0.1601 \\ 0.0046 & 0.0611 & 0.1691 & 0.0662 & 0.0014 \\ 0.1477 & 0.1687 & 0.1218 & 0.3664 & 0.3940 \\ 0.1195 & 0.1031 & 0.0715 & 0.2956 & 0.3343 \end{bmatrix}.$$

The worst-case probability that we are off by more than one is $P_{\text{wc}}^* = 0.27$. The worst-case probability of being off by more than one using a maximum-likelihood detector is $P_{\text{wc}}^{\text{ml}} = 0.74$, nearly a factor of three worse than the optimal detector.

The following matlab code finds the optimal probability detection matrix D .

```
off_by_one_det_data;

% off by one randomized detector
cvx_begin
    variables D(m,m) T(m,n)
    % d(i) is prob we're off by zero or one, with hypothesis i
    d = cvx(zeros(m,1));
    d(1) = D(1,1)+D(2,1);
    for i = 2:m-1
        d(i) = D(i-1,i)+D(i,i)+D(i+1,i);
    end
    d(m) = D(m-1,m)+D(m,m);
    T >= 0; sum(T) == 1;
    D == T*P; % detection probability matrix
    minimize (max(1-d))
cvx_end
Popt = cvx_optval; % max prob we are off by more than one

% maximum likelihood (ML) detector
% first we form the detector matrix, T_ml
Tml = zeros(m,n);
for i=1:n
    j = find(P(i,:)==max(P(i,:)));
    Tml(j,i) = 1;
end
```

```

% and now the ML probability detector matrix, D_ml
Dml = Tml*P;

% worst case probability using maximum-likelihood detector
% dml(i) is prob we're off by zero or one, with hypothesis i
dml(1) = Dml(1,1)+Dml(2,1);
for i=2:m-1
    dml(i) = Dml(i-1,i)+Dml(i,i)+Dml(i+1,i);
end
dml(m) = Dml(m-1,m)+Dml(m,m);
Pml = max(1-dml); % ML max prob we are off by more than one

disp('the optimal detection prob matrix');
disp(D)
disp('max prob using optimal detector');
disp(Popt)
disp('max prob using ML detector');
disp(Pml)

```

- 6.9** *Experiment design with condition number objective.* Explain how to solve the experiment design problem (§7.5) with the condition number $\text{cond}(E)$ of E (the error covariance matrix) as the objective to be minimized.

Solution. The problem is a quasiconvex minimization problem. We can write it as

$$\begin{aligned} & \text{minimize} && t_1/t_2 \\ & \text{subject to} && t_2 I \preceq \sum_{i=1}^m \gamma_i v_i v_i^T \preceq t_1 I \\ & && \gamma \succeq 0, \quad \mathbf{1}^T \gamma = 1 \end{aligned}$$

(where the domain of t_1/t_2 is $\mathbf{R} \times \mathbf{R}_{++}$), with variables γ , t_1 , t_2 . We can solve this using bisection, solving an SDP feasibility problem in each step.

We can also solve the problem with one SDP, as follows. We divide the LMI by t_2 , and define $s = t_1/t_2$, $\lambda = \gamma/t_2$ to get

$$\begin{aligned} & \text{minimize} && s \\ & \text{subject to} && I \preceq \sum_{i=1}^m \lambda_i v_i v_i^T \preceq sI \\ & && \lambda \succeq 0, \quad \mathbf{1}^T \lambda = 1/t_2, \end{aligned}$$

where the variable t_2 must be positive. The last constraint is the same as $\mathbf{1}^T \lambda > 0$, which is redundant. So we end up with the single SDP

$$\begin{aligned} & \text{minimize} && s \\ & \text{subject to} && I \preceq \sum_{i=1}^m \lambda_i v_i v_i^T \preceq tI \\ & && \lambda \succeq 0. \end{aligned}$$

We reconstruct the optimal γ as $\gamma^* = \lambda^*/\mathbf{1}^T \lambda^*$.

6.10 Worst-case probability of loss. Two investments are made, with random returns R_1 and R_2 . The total return for the two investments is $R_1 + R_2$, and the probability of a loss (including breaking even, *i.e.*, $R_1 + R_2 = 0$) is $p^{\text{loss}} = \mathbf{prob}(R_1 + R_2 \leq 0)$. The goal is to find the worst-case (*i.e.*, maximum possible) value of p^{loss} , consistent with the following information. Both R_1 and R_2 have Gaussian marginal distributions, with known means μ_1 and μ_2 and known standard deviations σ_1 and σ_2 . In addition, it is known that R_1 and R_2 are correlated with correlation coefficient ρ , *i.e.*,

$$\mathbf{E}(R_1 - \mu_1)(R_2 - \mu_2) = \rho\sigma_1\sigma_2.$$

Your job is to find the worst-case p^{loss} over any joint distribution of R_1 and R_2 consistent with the given marginals and correlation coefficient.

We will consider the specific case with data

$$\mu_1 = 8, \quad \mu_2 = 20, \quad \sigma_1 = 6, \quad \sigma_2 = 17.5, \quad \rho = -0.25.$$

We can compare the results to the case when R_1 and R_2 are jointly Gaussian. In this case we have

$$R_1 + R_2 \sim \mathcal{N}(\mu_1 + \mu_2, \sigma_1^2 + \sigma_2^2 + 2\rho\sigma_1\sigma_2),$$

which for the data given above gives $p^{\text{loss}} = 0.050$. Your job is to see how much larger p^{loss} can possibly be.

This is an infinite-dimensional optimization problem, since you must maximize p^{loss} over an infinite-dimensional set of joint distributions. To (approximately) solve it, we discretize the values that R_1 and R_2 can take on, to $n = 100$ values r_1, \dots, r_n , uniformly spaced from $r_1 = -30$ to $r_n = +70$. We use the discretized marginals $p^{(1)}$ and $p^{(2)}$ for R_1 and R_2 , given by

$$p_i^{(k)} = \mathbf{prob}(R_k = r_i) = \frac{\exp(-(r_i - \mu_k)^2/(2\sigma_k^2))}{\sum_{j=1}^n \exp(-(r_j - \mu_k)^2/(2\sigma_k^2))},$$

for $k = 1, 2$, $i = 1, \dots, n$.

Formulate the (discretized) problem as a convex optimization problem, and solve it. Report the maximum value of p^{loss} you find. Plot the joint distribution that yields the maximum value of p^{loss} using the Matlab commands `mesh` and `contour`.

Remark. You might be surprised at both the maximum value of p^{loss} , and the joint distribution that achieves it.

Solution. Let $P \in \mathbf{R}_+^{n \times n}$ be the matrix of joint probabilities, with $P_{ij} = \mathbf{prob}(R_1 = r_i, R_2 = r_j)$. The condition that the marginals are the given ones is

$$P\mathbf{1} = p^{(1)}, \quad P^T\mathbf{1} = p^{(2)}.$$

The correlation constraint can be expressed as

$$(r - \mu_1\mathbf{1})^T P(r - \mu_2\mathbf{1}) = \rho\sigma_1\sigma_2.$$

The probability of a loss $R_1 + R_2 \leq 0$ is given by

$$\mathbf{prob}(R_1 + R_2 \leq 0) = \sum_{r_i + r_j \leq 0} P_{ij}.$$

So the problem is an LP,

$$\begin{aligned} & \text{maximize} && \sum_{r_i+r_j \leq 0} P_{ij} \\ & \text{subject to} && P_{ij} \geq 0, \quad i, j = 1, \dots, n \\ & && P\mathbf{1} = p^{(1)} \\ & && P^T\mathbf{1} = p^{(2)} \\ & && (r - \mu_1\mathbf{1})^T P (r - \mu_2\mathbf{1}) = \rho\sigma_1\sigma_2, \end{aligned}$$

with variable $P \in \mathbf{R}_+^{n \times n}$.

The code to find the worst-case joint distribution is below.

```
% loss bounds solution
clear all; close all;

mu1 = 8; mu2 = 20;
sigma1 = 6; sigma2 = 17.5;
rho = -0.25;

% assuming jointly gaussian distribution
mu = mu1 + mu2;
sigma = sqrt(sigma1^2 + sigma2^2 + 2*rho*sigma1*sigma2);
ploss = normcdf(0, mu, sigma); % gaussian probability of loss

n = 100;
rmin = -30; rmax = 70;
% discretize outcomes of R1 and R2
r = linspace(rmin,rmax,n)';

% marginal distributions
p1 = exp(-(r-mu1).^2/(2*sigma1^2)); p1 = p1/sum(p1);
p2 = exp(-(r-mu2).^2/(2*sigma2^2)); p2 = p2/sum(p2);

% form mask of region where R1 + R2 <= 0
r1p = r*ones(1,n); r2p = ones(n,1)*r';
loss_mask = (r1p + r2p <= 0)';

cvx_begin
    variable P(n,n)
    maximize (sum(sum(P(loss_mask))))
    subject to
        P >= 0;
        sum(P ,2) == p1;
        sum(P' ,2) == p2;
        (r - mu1)'*P*(r - mu2) == rho*sigma1*sigma2;
cvx_end
```

```

pmax = cvx_optval; % worst case probability of loss
pmax
ploss

P = full(P);
figure(1); mesh(r1p, r2p, P');
xlabel('R1'); ylabel('R2'); zlabel('density');
xlim([rmin rmax]); ylim([rmin rmax]);
print -depsc 'loss_bounds_mesh.eps'

figure(2); contour(r1p, r2p, P');
xlabel('R1'); ylabel('R2'); grid on;
xlim([rmin rmax]); ylim([rmin rmax]);
print -depsc 'loss_bounds_cont.eps'

```

This yields a probability of loss of approximately 0.192, almost four times larger than the loss probability when R_1 and R_2 are jointly Gaussian! The resulting (worst-case) distribution is plotted below. It has mass on the line where $R_1 + R_2 = 0$ (*i.e.*, break even, which counts as a loss for us). Then it has extra mass on another region in the plane, which is needed to make the marginals the given Gaussians, as well as to meet the constraint on the correlation coefficient.

By the way, we were not picky about the numerics when grading, and gave generous partial credit so long as you grasped the main idea.

Remark. We didn't ask you to show this, but here's the analysis when R_1 and R_2 are jointly Gaussian. Since we are given the means, marginal variances, and correlation, we have

$$(R_1, R_2) \sim \mathcal{N}((\mu_1, \mu_2), \Sigma), \quad \Sigma = \begin{bmatrix} \sigma_1^2 & \rho\sigma_1\sigma_2 \\ \rho\sigma_1\sigma_2 & \sigma_2^2 \end{bmatrix}.$$

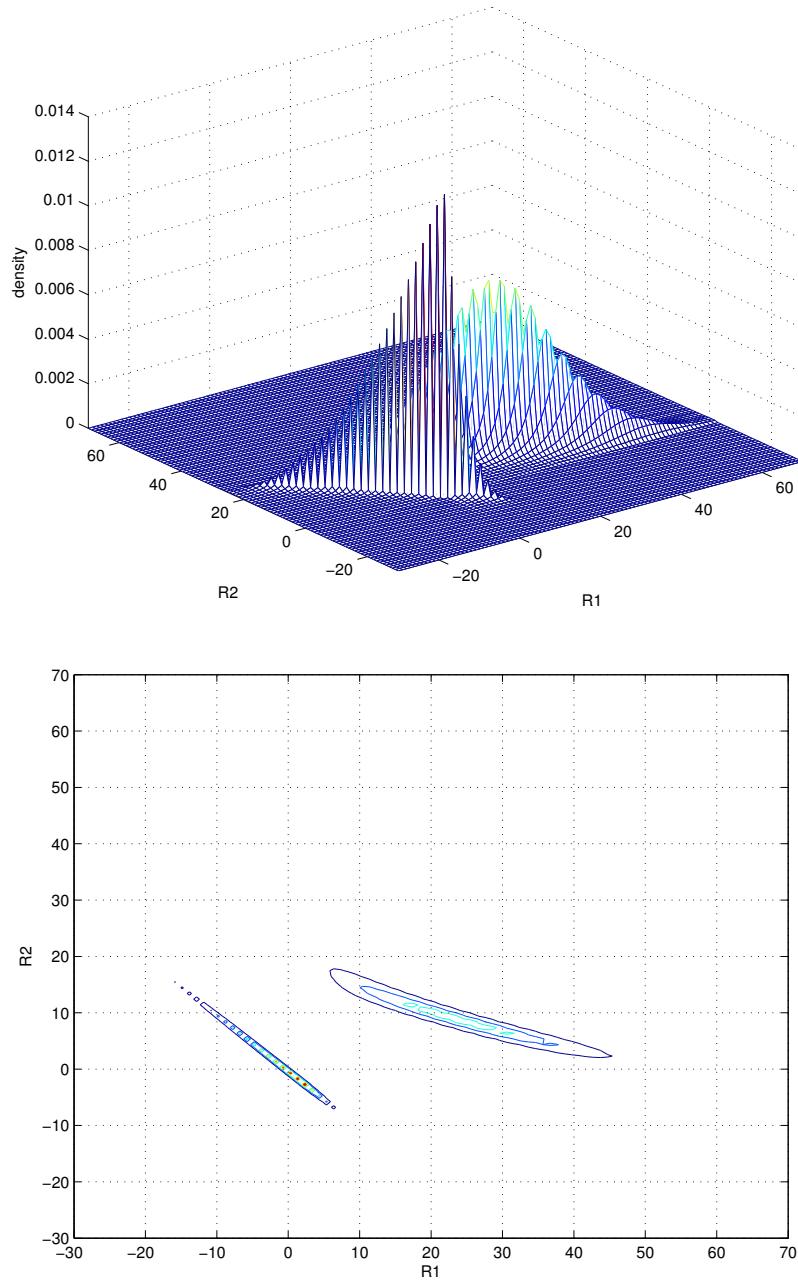
The total return is $R_1 + R_2$ is also Gaussian, with $\mathbf{E}(R_1 + R_2) = \mu_1 + \mu_2$ and variance

$$\text{var}(R_1 + R_2) = \mathbf{1}^T \Sigma \mathbf{1} = \sigma_1^2 + \sigma_2^2 + 2\rho\sigma_1\sigma_2.$$

The probability of loss for the jointly Gaussian case is therefore

$$p^{\text{loss}} = \Phi \left(-\frac{\mu_1 + \mu_2}{\sqrt{\sigma_1^2 + \sigma_2^2 + 2\rho\sigma_1\sigma_2}} \right),$$

where $\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-t^2/2} dt$ is the cumulative distribution of a standard Gaussian.



6.11 Minimax linear fitting. Consider a linear measurement model $y = Ax + v$, where $x \in \mathbf{R}^n$ is a vector of parameters to be estimated, $y \in \mathbf{R}^m$ is a vector of measurements, $v \in \mathbf{R}^m$ is a set of measurement errors, and $A \in \mathbf{R}^{m \times n}$ with rank n , with $m \geq n$. We know y and A , but we don't know v ; our goal is to estimate x . We make only one assumption about the measurement error v : $\|v\|_\infty \leq \epsilon$.

We will estimate x using a linear estimator $\hat{x} = By$; we must choose the estimation matrix $B \in \mathbf{R}^{n \times m}$. The estimation error is $e = \hat{x} - x$. We will choose B to minimize the maximum possible value of $\|e\|_\infty$, where the maximum is over all values of x and all values of v satisfying $\|v\|_\infty \leq \epsilon$.

- (a) Show how to find B via convex optimization.
- (b) *Numerical example.* Solve the problem instance given in `minimax_fit_data.m`. Display the \hat{x} you obtain and report $\|\hat{x} - x^{\text{true}}\|_\infty$. Here x^{true} is the value of x used to generate the measurement y ; it is given in the data file.

Solution.

- (a) The problem is to minimize the objective $f(B)$, where

$$f(B) = \max_{x, \|v\|_\infty \leq \epsilon} \|\hat{x} - x\|_\infty = \max_{x, \|v\|_\infty \leq \epsilon} \|(BA - I)x + Bv\|_\infty.$$

The maximum over x is $+\infty$, unless we have $BA = I$, so this will be a constraint for us. (By the way, $BA = I$ means that B is a left inverse of A . The associated estimator is called an unbiased linear estimator, since without noise there is no estimation error.) Assuming $BA = I$, we have

$$f(B) = \max_{\|v\|_\infty \leq \epsilon} \|Bv\|_\infty = \max_{\|v\|_\infty \leq \epsilon} \max_{i=1,\dots,m} |b_i^T v| = \epsilon \max_{i=1,\dots,m} \|b_i\|_1,$$

where b_i^T are the rows of B . (In fact, $f(B)$ is a norm of B called, for obvious reasons, the max-row-sum norm, and denoted $\|B\|_\infty$.) The problem is thus

$$\begin{aligned} & \text{minimize} && \max_{i=1,\dots,m} \|b_i\|_1 \\ & \text{subject to} && BA = I. \end{aligned}$$

This is evidently a convex problem, with variable B . In fact, it is separable in the rows; we can solve for each row separately, by solving

$$\begin{aligned} & \text{minimize} && \|b_i\|_1 \\ & \text{subject to} && b_i^T A = e_i^T, \end{aligned}$$

for $i = 1, \dots, n$.

- (b) The solution is given on the following page.

```

%% minimax linear fitting

minimax_fit_data;

cvx_begin
    variable B(n,m)
    minimize (norm(B,inf))
    subject to
        B*A == eye(n);
cvx_end

x = B*y;
fprintf(1, 'estimation error = %f\n', norm(x - x_true, inf));

```

6.12 Cox proportional hazards model. Let T be a continuous random variable taking on values in \mathbf{R}_+ . We can think of T as modeling an event that takes place at some unknown future time, such as the death of a living person or a machine failure.

The *survival function* is $S(t) = \text{prob}(T \geq t)$, which satisfies $S(0) = 1$, $S'(t) \leq 0$, and $\lim_{t \rightarrow \infty} S(t) = 0$. The *hazard rate* is given by $\lambda(t) = -S'(t)/S(t) \in \mathbf{R}_+$, and has the following interpretation: For small $\delta > 0$, $\lambda(t)\delta$ is approximately the probability of the event occurring in $[t, t + \delta]$, given that it has not occurred up to time t . The survival function can be expressed in terms of the hazard rate:

$$S(t) = \exp\left(-\int_0^t \lambda(\tau) d\tau\right).$$

(The hazard rate must have infinite integral over $[0, \infty)$.)

The *Cox proportional hazards model* gives the hazard rate as a function of some features or explanatory variables (assumed constant in time) $x \in \mathbf{R}^n$. In particular, λ is given by

$$\lambda(t) = \lambda_0(t) \exp(w^T x),$$

where λ_0 (which is nonnegative, with infinite integral) is called the *baseline hazard rate*, and $w \in \mathbf{R}^n$ is a vector of model parameters. (The name derives from the fact that $\lambda(t)$ is proportional to $\exp(w_i x_i)$, for each i .)

Now suppose that we have observed a set of independent samples, with event times t^j and feature values x^j , for $j = 1, \dots, N$. In other words, we observe that the event with features x^j occurred at time t^j . You can assume that the baseline hazard rate λ_0 is known. Show that maximum likelihood estimation of the parameter w is a convex optimization problem.

Remarks. Regularization is typically included in Cox proportional hazards fitting; for example, adding ℓ_1 regularization yields a sparse model, which selects the features to be used. The basic Cox proportional hazards model described here is readily extended to include discrete times of the event, censored measurements (which means that we only observe T to be in an interval), and the effects of features that can vary with time.

Solution. Since the cumulative distribution function of T is $1 - S(t)$, the density is given by

$$-S'(t) = \lambda(t)S(t) = \lambda_0(t) \exp(w^T x) \exp\left(-\int_0^t \lambda_0(\tau) \exp(w^T x) d\tau\right),$$

where we use the definition $\lambda(t) = -S'(t)/S(t)$ and substitute the model form for $\lambda(t)$. Thus the log density is

$$w^T x - S_0(t) \exp(w^T x)$$

plus a constant (*i.e.*, a term that does not depend on w), where

$$S_0(t) = \int_0^t \lambda_0(\tau) d\tau$$

is the baseline survival function. Since λ_0 is nonnegative, we have that $S_0(t) \geq 0$, so the log density is a concave function of w .

The log-likelihood function for the observed data is then

$$\ell(w) = \sum_{j=1}^N \left(w^T x^j - S_0(t^j) \exp(w^T x^j) \right),$$

which is a concave function of w . Maximizing w is a convex optimization problem.

- 6.13 Maximum likelihood estimation for an affinely transformed distribution.** Let z be a random variable on \mathbf{R}^n with density $p_z(u) = \exp -\phi(\|u\|_2)$, where $\phi : \mathbf{R} \rightarrow \mathbf{R}$ is convex and increasing. Examples of such distributions include the standard normal $\mathcal{N}(0, \sigma^2 I)$, with $\phi(u) = (u)_+^2 + \alpha$, and the multivariate Laplacian distribution, with $\phi(u) = (u)_+ + \beta$, where α and β are normalizing constants, and $(a)_+ = \max\{a, 0\}$. Now let x be the random variable $x = Az + b$, where $A \in \mathbf{R}^{n \times n}$ is nonsingular. The distribution of x is parametrized by A and b .

Suppose x_1, \dots, x_N are independent samples from the distribution of x . Explain how to find a maximum likelihood estimate of A and b using convex optimization. If you make any further assumptions about A and b (beyond invertibility of A), you must justify it.

Hint. The density of $x = Az + b$ is given by

$$p_x(v) = \frac{1}{|\det A|} p_z(A^{-1}(v - b)).$$

Solution. The density of $x = Az + b$ is given by

$$p_x(v) = \frac{1}{|\det A|} \exp -\phi(\|A^{-1}(v - b)\|_2).$$

We first observe that the density with parameters (A, b) is the same as the density with parameters (AQ, b) , for any orthogonal matrix Q , since

$$|\det(AQ)| = |\det A||\det Q| = |\det A|,$$

and

$$\|(AQ)^{-1}(v - b)\|_2 = \|Q^T A^{-1}(v - b)\|_2 = \|A^{-1}(v - b)\|_2.$$

Let A have SVD $A = U\Sigma V^T$. Choosing $Q = VU^T$, we see that $AQ = U\Sigma U^T \in \mathbf{S}_{++}^n$. So we can always assume that $A \in \mathbf{S}_{++}^n$.

The log-likelihood function for a single sample x is

$$\ell(A, b) = -\log \det A - \phi(\|A^{-1}(x - b)\|_2),$$

so for N independent samples, we have log-likelihood function

$$\ell(A, b) = -N \log \det A - \sum_{i=1}^N \phi(\|A^{-1}(x_i - b)\|_2).$$

We must maximize ℓ over $A \in \mathbf{S}_{++}^n$ and $b \in \mathbf{R}^n$. It's not concave in these parameters, but we will use instead the parameters

$$B = A^{-1} \in \mathbf{S}_{++}^n, \quad c = A^{-1}b \in \mathbf{R}^n.$$

From B and c we can recover A and b as

$$A = B^{-1}, \quad b = B^{-1}c.$$

In terms of B and c , the log-likelihood function is

$$\tilde{\ell}(B, c) = N \log \det B - \sum_{i=1}^N \phi(\|Bx_i - c\|_2),$$

which is concave. To see this, we note that the first term is concave in B , and the second term is concave since $\|Bx_i - c\|_2$ is convex in (B, c) , and by the composition rule, $\phi(\|Bx_i - c\|_2)$ is convex. So we just maximize $\tilde{\ell}(B, c)$ over $B \in \mathbf{S}_{++}^n$, $c \in \mathbf{R}^n$, and then get A and b as described above.

6.14 A simple MAP problem. We seek to estimate a point $x \in \mathbf{R}_+^2$, with exponential prior density $p(x) = \exp(-(x_1 + x_2))$, based on the measurements

$$y_1 = x_1 + v_1, \quad y_2 = x_2 + v_2, \quad y_3 = x_1 - x_2 + v_3,$$

where v_1, v_2, v_3 are IID $\mathcal{N}(0, 1)$ random variables (also independent of x). A naïve estimate of x is given by $\hat{x}_{\text{naive}} = (y_1, y_2)$.

- (a) Explain how to find the MAP estimate of x , given the observations y_1, y_2, y_3 .
- (b) Generate 100 random instances of x and y , from the given distributions. For each instance, find the MAP estimate \hat{x}_{map} and the naïve estimate x_{naive} . Give a scatter plot of the MAP estimation error, *i.e.*, $\hat{x}_{\text{map}} - x$, and another scatter plot of the naïve estimation error, $\hat{x}_{\text{naive}} - x$.

XXX missing (March 2016) XXX

6.15 Minimum possible maximum correlation. Let Z be a random variable taking values in \mathbf{R}^n , and let $\Sigma \in \mathbf{S}_{++}^n$ be its covariance matrix. We do not know Σ , but we do know the variance of m linear functions of Z . Specifically, we are given nonzero vectors $a_1, \dots, a_m \in \mathbf{R}^n$ and $\sigma_1, \dots, \sigma_m > 0$ for which

$$\text{var}(a_i^T Z) = \sigma_i^2, \quad i = 1, \dots, m.$$

For $i \neq j$ the correlation of Z_i and Z_j is defined to be

$$\rho_{ij} = \frac{\Sigma_{ij}}{\sqrt{\Sigma_{ii}\Sigma_{jj}}}.$$

Let $\rho^{\max} = \max_{i \neq j} |\rho_{ij}|$ be the maximum (absolute value) of the correlation among entries of Z . If ρ^{\max} is large, then at least two components of Z are highly correlated (or anticorrelated).

- (a) Explain how to find the smallest value of ρ^{\max} that is consistent with the given information, using convex or quasiconvex optimization. If your formulation involves a change of variables or other transformation, justify it.
- (b) The file `correlation_bounds_data.*` contains $\sigma_1, \dots, \sigma_m$ and the matrix A with columns a_1, \dots, a_m . Find the minimum value of ρ^{\max} that is consistent with this data. Report your minimum value of ρ^{\max} , and give a corresponding covariance matrix Σ that achieves this value. You can report the minimum value of ρ^{\max} to an accuracy of 0.01.

Solution.

- (a) Using the formula for the variance of a linear function of a random variable, we have that

$$\text{var}(a_i^T Z) = a_i^T \text{var}(Z) a_i = a_i^T \Sigma a_i,$$

which is a linear function of Σ . We can find the minimum value of the maximum correlation among components of Z that is consistent with the data by solving the following optimization problem.

$$\begin{aligned} & \text{minimize} && \max_{i \neq j} |\rho_{ij}| \\ & \text{subject to} && a_i^T \Sigma a_i = \sigma_i^2, \quad i = 1, \dots, m \\ & && \Sigma \succeq 0. \end{aligned}$$

Observe that

$$|\rho_{ij}| = \frac{|\Sigma_{ij}|}{\sqrt{\Sigma_{ii}\Sigma_{jj}}}$$

is a quasiconvex function of Σ : the numerator is a nonnegative convex function of Σ , and the denominator is a positive concave function of Σ (it is the composition of the geometric mean and a linear function of Σ). Since the maximum of quasiconvex functions is quasiconvex, the objective in the optimization problem above is quasiconvex. Thus, we can find the smallest value of ρ^{\max} by solving a quasiconvex optimization problem. In particular, we have that $\rho^{\max} \leq t$ is consistent with the data if and only if the following convex feasibility problem is feasible:

$$\begin{aligned} |\Sigma_{ij}| &\leq t\sqrt{\Sigma_{ii}\Sigma_{jj}}, \quad i \neq j \\ a_i^T \Sigma a_i &= \sigma_i^2, \quad i = 1, \dots, m \\ \Sigma &\succeq 0 \end{aligned}$$

We can find the minimum value of t for which this problem is feasible using bisection search starting with $t = 0$ and $t = 1$.

- (b) The following Matlab code solves the problem.

```
clear all; close all; clc
correlation_bounds_data;

% find the minimum possible maximum correlation
lb = 0;
ub = 1;
Sigma_opt = nan(n,n);
while ub-lb > 1e-3
    t = (lb+ub)/2;
```

```

cvx_begin sdp czz
    variable Sigma(n,n) symmetric
    for i = 1:(n-1)
        for j = (i+1):n
            abs(Sigma(i,j)) <= t * geo_mean([Sigma(i,i), Sigma(j,j)])
        end
    end
    for i = 1:m
        A(:,i)' * Sigma * A(:,i) == sigma(i)^2
    end
    Sigma >= 0
cvx_end
if strcmp(cvx_status, 'Solved')
    ub = t;
    Sigma_opt = Sigma;
else
    lb = t;
end
end

% print the results and check the correlation matrix
t
Sigma = Sigma_opt
C = diag(1./sqrt(diag(Sigma)));
R = C * Sigma * C;
rho_max = max(max(R - diag(diag(R))))

```

The following Python code solves the problem.

```

import cvxpy as cvx
from math import sqrt

from correlation_bounds_data import *

Sigma = cvx.Semidef(n)
t = cvx.Parameter(sign='positive')
rho_cons = []
for i in range(n - 1):
    for j in range(i + 1, n):
        Sij = cvx.vstack(Sigma[i, i], Sigma[j, j])
        rho_cons += [cvx.abs(Sigma[i, j]) <= t * cvx.geo_mean(Sij)]
var_cons = [A[:, i].T * Sigma * A[:, i] == sigma[i]**2
            for i in range(m)]
problem = cvx.Problem(cvx.Minimize(0), rho_cons + var_cons)

lb, ub = 0.0, 1.0

```

```

Sigma_opt = None
while ub - lb > 1e-3:
    t.value = (lb + ub) / 2
    problem.solve()
    if problem.status == cvx.OPTIMAL:
        ub = t.value
        Sigma_opt = Sigma.value
    else:
        lb = t.value

print('rho_max =', t.value)
print('Sigma =', Sigma_opt)

# compute the correlation matrix
C = np.diag([1 / sqrt(Sigma_opt[i, i]) for i in range(n)])
R = C * Sigma_opt * C
print('R =', R)

```

The following Julia code solves the problem.

```

using Convex, SCS
set_default_solver(SCSSolver(verbose=false))

include("correlation_bounds_data.jl")

lb = 0; ub = 1
t = (lb+ub)/2
Sigma_opt = []
while ub-lb > 1e-3
    t = (lb+ub)/2
    Sigma = Semidefinite(n)
    problem = satisfy()
    for i = 1:(n-1)
        for j = (i+1):n
            problem.constraints +=
                (abs(Sigma[i,j]) <= t*geomean(Sigma[i,i],Sigma[j,j]))
        end
    end
    for i = 1:m
        problem.constraints += (A[:,i]' * Sigma * A[:,i] == sigma[i]^2)
    end

    solve!(problem)
    if problem.status == :Optimal
        ub = t
        Sigma_opt = Sigma.value
    else

```

```

    lb = t
end
endl
println("t = $(round(t,4))")
println("Sigma = $(round(Sigma_opt,4))")

# compute the correlation matrix
C = diagm(Float64[1/sqrt(Sigma_opt[i,i]) for i in 1:n])
R = C * Sigma_opt * C
println("R = $(round(R,4))")

```

We find that the minimum value of the maximum correlation that is consistent with the data is $\rho^{\max} = 0.62$. A corresponding covariance matrix is

$$\Sigma = \begin{bmatrix} 3.78 & 0.30 & 1.46 & 1.47 & 0.24 \\ 0.30 & 2.91 & -1.28 & 1.29 & 0.86 \\ 1.46 & -1.28 & 1.46 & 0.09 & 0.27 \\ 1.47 & 1.29 & 0.09 & 1.48 & 0.49 \\ 0.24 & 0.09 & 0.27 & 0.49 & 0.42 \end{bmatrix}.$$

Although we did not ask you to do so, it is a good idea to compute the correlation matrix to this value of Σ , and check that the maximum correlation is equal to ρ^{\max} :

$$R = \begin{bmatrix} 1.00 & 0.09 & 0.62 & 0.62 & 0.19 \\ 0.09 & 1.00 & -0.62 & 0.62 & 0.08 \\ 0.62 & -0.62 & 1.00 & 0.06 & 0.34 \\ 0.62 & 0.62 & 0.06 & 1.00 & 0.62 \\ 0.19 & 0.08 & 0.34 & 0.62 & 1.00 \end{bmatrix}.$$

7 Geometry

7.1 Efficiency of maximum volume inscribed ellipsoid. In this problem we prove the following geometrical result. Suppose C is a polyhedron in \mathbf{R}^n , symmetric about the origin, and described as

$$C = \{x \mid -1 \leq a_i^T x \leq 1, i = 1, \dots, p\}.$$

Let

$$\mathcal{E} = \{x \mid x^T Q^{-1} x \leq 1\},$$

with $Q \in \mathbf{S}_{++}^n$, be the maximum volume ellipsoid with center at the origin, inscribed in C . Then the ellipsoid

$$\sqrt{n}\mathcal{E} = \{x \mid x^T Q^{-1} x \leq n\}$$

(i.e., the ellipsoid \mathcal{E} , scaled by a factor \sqrt{n} about the origin) *contains* C .

- (a) Show that the condition $\mathcal{E} \subseteq C$ is equivalent to $a_i^T Q a_i \leq 1$ for $i = 1, \dots, p$.
- (b) The volume of \mathcal{E} is proportional to $(\det Q)^{1/2}$, so we can find the maximum volume ellipsoid \mathcal{E} inside C by solving the convex problem

$$\begin{aligned} & \text{minimize} && \log \det Q^{-1} \\ & \text{subject to} && a_i^T Q a_i \leq 1, \quad i = 1, \dots, p. \end{aligned} \tag{31}$$

The variable is the matrix $Q \in \mathbf{S}^n$ and the domain of the objective function is \mathbf{S}_{++}^n .

Derive the Lagrange dual of problem (31).

- (c) Note that Slater's condition for (31) holds ($a_i^T Q a_i < 1$ for $Q = \epsilon I$ and $\epsilon > 0$ small enough), so we have strong duality, and the KKT conditions are necessary and sufficient for optimality. What are the KKT conditions for (31)?

Suppose Q is optimal. Use the KKT conditions to show that

$$x \in C \implies x^T Q^{-1} x \leq n.$$

In other words $C \subseteq \sqrt{n}\mathcal{E}$, which is the desired result.

Solution.

- (a) The ellipsoid $\mathcal{E} = \{Q^{1/2}y \mid \|y\|_2 \leq 1\}$ is contained in C if and only if

$$\|Q^{1/2}a_i\|_2 = \sup_{\|y\|_2 \leq 1} |a_i^T Q^{1/2}y| \leq 1, \quad i = 1, \dots, p.$$

- (b) The dual function is

$$\begin{aligned} g(\lambda) &= \inf_{Q\succ 0} L(Q, \lambda) \\ &= \inf_{Q\succ 0} \left(\log \det Q^{-1} + \sum_{i=1}^p \lambda_i (a_i^T Q a_i - 1) \right) \\ &= \inf_{Q\succ 0} \left(\log \det Q^{-1} + \mathbf{tr} \left(\left(\sum_{i=1}^p \lambda_i a_i a_i^T \right) Q \right) - \sum_{i=1}^p \lambda_i \right). \end{aligned}$$

We now use the following fact:

$$\inf_{X \succ 0} (\log \det X^{-1} + \mathbf{tr}(XY)) = \begin{cases} \log \det Y + n & Y \succ 0 \\ -\infty & \text{otherwise.} \end{cases}$$

The value for $Y \succ 0$ follows by setting the gradient of $\log \det X^{-1} + \mathbf{tr}(XY)$ to zero. This gives $-X^{-1} + Y = 0$, so the minimizer is $X = Y^{-1}$ if $Y \succ 0$. If $Y \not\succ 0$, there exists a nonzero a with $a^T Y a \leq 0$. Choosing $X = I + taa^T$ gives $\det X = 1 + t\|a\|_2^2$ and

$$\log \det X^{-1} + \mathbf{tr}(XY) = -\log(1 + ta^T a) + \mathbf{tr} Y + ta^T Y a.$$

If $a^T Y a \leq 0$ this goes to $-\infty$ as $t \rightarrow \infty$.

We conclude that the dual function is

$$g(\lambda) = \begin{cases} \log \det \sum_{i=1}^p (\lambda_i a_i a_i^T) - \sum_{i=1}^p \lambda_i + n & \text{if } \sum_{i=1}^p (\lambda_i a_i a_i^T) \succ 0 \\ -\infty & \text{otherwise.} \end{cases}$$

The resulting dual problem is

$$\begin{aligned} & \text{maximize} && \log \det \sum_{i=1}^p (\lambda_i a_i a_i^T) - \sum_{i=1}^p \lambda_i + n \\ & \text{subject to} && \lambda \succeq 0. \end{aligned}$$

(c) The KKT conditions are:

- *Primal feasibility:* $Q \succ 0$ and $a_i^T Q a_i \leq 1$ for $i = 1, \dots, p$.
- *Nonnegativity of dual multipliers:* $\lambda \succeq 0$.
- *Complementary slackness:* $\lambda_i(1 - a_i^T Q a_i) = 0$ for $i = 1, \dots, p$.
- *Gradient of Lagrangian is zero:*

$$Q^{-1} = \sum_{i=1}^p \lambda_i a_i a_i^T. \quad (32)$$

The complementary slackness condition implies that $a_i^T Q a_i = 1$ if $\lambda_i > 0$.

Now suppose Q and λ are primal and dual optimal. If we take the inner product of the two sides of the equation (32) with Q , we get

$$n = \sum_{i=1}^p \lambda_i \mathbf{tr}(Q a_i a_i^T) = \sum_{i=1}^p \lambda_i a_i^T Q a_i = \sum_{i=1}^p \lambda_i.$$

The last step follows from the complementary slackness conditions. Finally, we note, again using (32), that

$$x^T Q^{-1} x = \sum_{i=1}^p \lambda_i (a_i^T x)^2 \leq \sum_{i=1}^p \lambda_i = n$$

if $x \in C$, i.e., if $|a_i^T x| \leq 1$ for $i = 1, \dots, p$.

7.2 Euclidean distance matrices. A matrix $X \in \mathbf{S}^n$ is a *Euclidean distance matrix* if its elements x_{ij} can be expressed as

$$x_{ij} = \|p_i - p_j\|_2^2, \quad i, j = 1, \dots, n,$$

for some vectors p_1, \dots, p_n (of arbitrary dimension). In this exercise we prove several classical characterizations of Euclidean distance matrices, derived by I. Schoenberg in the 1930s.

- (a) Show that X is a Euclidean distance matrix if and only if

$$X = \text{diag}(Y)\mathbf{1}^T + \mathbf{1}\text{diag}(Y)^T - 2Y \quad (33)$$

for some matrix $Y \in \mathbf{S}_+^n$ (the symmetric positive semidefinite matrices of order n). Here, $\text{diag}(Y)$ is the n -vector formed from the diagonal elements of Y , and $\mathbf{1}$ is the n -vector with all its elements equal to one. The equality (33) is therefore equivalent to

$$x_{ij} = y_{ii} + y_{jj} - 2y_{ij}, \quad i, j = 1, \dots, n.$$

Hint. Y is the Gram matrix associated with the vectors p_1, \dots, p_n , *i.e.*, the matrix with elements $y_{ij} = p_i^T p_j$.

- (b) Show that the set of Euclidean distance matrices is a convex cone.
(c) Show that X is a Euclidean distance matrix if and only if

$$\text{diag}(X) = 0, \quad X_{22} - X_{21}\mathbf{1}^T - \mathbf{1}X_{21}^T \preceq 0. \quad (34)$$

The subscripts refer to the partitioning

$$X = \begin{bmatrix} x_{11} & X_{21}^T \\ X_{21} & X_{22} \end{bmatrix}$$

with $X_{21} \in \mathbf{R}^{n-1}$, and $X_{22} \in \mathbf{S}^{n-1}$.

Hint. The definition of Euclidean distance matrix involves only the distances $\|p_i - p_j\|_2$, so the origin can be chosen arbitrarily. For example, it can be assumed without loss of generality that $p_1 = 0$. With this assumption there is a unique Gram matrix Y for a given Euclidean distance matrix X . Find Y from (33), and relate it to the lefthand side of the inequality (34).

- (d) Show that X is a Euclidean distance matrix if and only if

$$\text{diag}(X) = 0, \quad (I - \frac{1}{n}\mathbf{1}\mathbf{1}^T)X(I - \frac{1}{n}\mathbf{1}\mathbf{1}^T) \preceq 0. \quad (35)$$

Hint. Use the same argument as in part (c), but take the mean of the vectors p_k at the origin, *i.e.*, impose the condition that $p_1 + p_2 + \dots + p_n = 0$.

- (e) Suppose X is a Euclidean distance matrix. Show that the matrix $W \in \mathbf{S}^n$ with elements

$$w_{ij} = e^{-x_{ij}}, \quad i, j = 1, \dots, n,$$

is positive semidefinite.

Hint. Use the following identity from probability theory. Define $z \sim \mathcal{N}(0, I)$. Then

$$\mathbf{E} e^{iz^T x} = e^{-\frac{1}{2}\|x\|_2^2}$$

for all x , where $i = \sqrt{-1}$ and \mathbf{E} denotes expectation with respect to z . (This is the characteristic function of a multivariate normal distribution.)

Solution.

- (a) Suppose X is a Euclidean distance matrix with $x_{ij} = \|p_i - p_j\|_2^2$, and let Y be the corresponding Gram matrix. Then

$$\begin{aligned} x_{ij} &= \|p_i - p_j\|_2^2 \\ &= \|p_i\|_2^2 + \|p_j\|_2^2 - 2p_i^T p_j \\ &= y_{ii} + y_{jj} - 2y_{ij}. \end{aligned}$$

Moreover $Y \succeq 0$, because all Gram matrices are positive semidefinite. (To see this, write Y as $Y = P^T P$, where P is the matrix with the vectors p_i as its columns. Then for all u ,

$$u^T Y u = u^T P^T P u = \|Pu\|_2^2 \geq 0.$$

Hence, $Y \succeq 0$.)

Conversely, suppose $x_{ij} = y_{ii} + y_{jj} - 2y_{ij}$ for some $Y \succeq 0$. By factoring Y as $Y = P^T P$ (for example, using the eigenvalue decomposition), we obtain a set of points p_i (the columns of P) that satisfy $x_{ij} = \|p_i - p_j\|_2^2$. This proves that X is a Euclidean distance matrix.

- (b) The result in part (a) characterizes the set of Euclidean distance matrices as the image of the positive semidefinite cone under a linear mapping

$$Y \mapsto \mathbf{diag}(Y)\mathbf{1}^T + \mathbf{1}\mathbf{diag}(Y)^T - 2Y.$$

Since the set of positive semidefinite matrices is a convex cone, its image under a linear transformation is also a convex cone.

- (c) Suppose X is a Euclidean distance matrix. Then clearly, $\mathbf{diag}(X) = 0$. To prove that

$$X_{22} - X_{21}\mathbf{1}^T - \mathbf{1}X_{21}^T \preceq 0$$

we follow the hint and derive the Gram matrix Y corresponding to a configuration p_1, \dots, p_n with $p_1 = 0$. Since $p_1 = 0$, the first column and the first row of Y are zero, *i.e.*,

$$Y = \begin{bmatrix} 0 & 0 \\ 0 & Y_{22} \end{bmatrix},$$

if we use the same partitioning as for X . (Y_{22} is a matrix of order $n - 1$.) The equation

$$X = \mathbf{diag}(Y)\mathbf{1}^T + \mathbf{1}\mathbf{diag}(Y)^T - 2Y$$

then gives the conditions

$$X_{21} = \mathbf{diag}(Y_{22}), \quad X_{22} = \mathbf{diag}(Y_{22})\mathbf{1}^T + \mathbf{1}\mathbf{diag}(Y_{22})^T - 2Y_{22}.$$

Solving for Y gives

$$Y_{22} = \frac{1}{2} (X_{21}\mathbf{1}^T + \mathbf{1}X_{21}^T - X_{22}),$$

and this matrix is positive semidefinite because the Gram matrix Y is positive semidefinite. This proves that $X_{22} - X_{21}\mathbf{1}^T - \mathbf{1}X_{21}^T \preceq 0$ if X is a Euclidean distance matrix.

Conversely, suppose X satisfies

$$\mathbf{diag}(X) = 0, \quad X_{22} - X_{21}\mathbf{1}^T - \mathbf{1}X_{21}^T \preceq 0.$$

Define

$$Y = \begin{bmatrix} 0 & 0 \\ 0 & (1/2)(X_{21}\mathbf{1}^T + \mathbf{1}X_{21}^T - X_{22}) \end{bmatrix}.$$

Then $Y \succeq 0$ and

$$\begin{aligned} \mathbf{diag}(Y) &= \begin{bmatrix} 0 \\ X_{21} \end{bmatrix} \\ \mathbf{diag}(Y)\mathbf{1}^T + \mathbf{1} \mathbf{diag}(Y)^T - 2Y &= \begin{bmatrix} 0 & X_{21}^T \\ X_{21} & X_{21}\mathbf{1}^T + \mathbf{1}X_{21}^T - 2Y_{22} \end{bmatrix} \\ &= \begin{bmatrix} 0 & X_{21}^T \\ X_{21} & X_{22} \end{bmatrix} \\ &= X. \end{aligned}$$

From the result in part (a) this implies that X is a Euclidean distance matrix.

- (d) Suppose X is a Euclidean distance matrix. Setting $\sum_k p_k = 0$ is equivalent to imposing $Y\mathbf{1} = 0$. Under this condition we can solve

$$X = \mathbf{diag}(Y)\mathbf{1}^T + \mathbf{1} \mathbf{diag}(Y)^T - 2Y$$

for Y . Define $y = \mathbf{diag}(Y)$. Then

$$X\mathbf{1} = (y\mathbf{1}^T + \mathbf{1}y^T - 2Y)\mathbf{1} = ny + (\mathbf{1}^T y)\mathbf{1}$$

and $\mathbf{1}^T X \mathbf{1} = 2n\mathbf{1}^T y$. Solving for $\mathbf{1}^T y$ and y gives $\mathbf{1}^T y = (\mathbf{1}^T X \mathbf{1})/(2n)$ and

$$y = \frac{1}{n} (X\mathbf{1} - (\mathbf{1}^T y)\mathbf{1}) = \frac{1}{n} X\mathbf{1} - \frac{1}{2n^2} (\mathbf{1}^T X \mathbf{1})\mathbf{1}.$$

This gives

$$\begin{aligned} Y &= -\frac{1}{2} (X - y\mathbf{1}^T - \mathbf{1}y^T) \\ &= -\frac{1}{2} \left(X - \frac{1}{n} X \mathbf{1} \mathbf{1}^T - \frac{1}{n} \mathbf{1} \mathbf{1}^T X + \frac{1}{n^2} (\mathbf{1}^T X \mathbf{1}) \mathbf{1} \mathbf{1}^T \right) \\ &= -\frac{1}{2} \left(I - \frac{1}{n} \mathbf{1} \mathbf{1}^T \right) X \left(I - \frac{1}{n} \mathbf{1} \mathbf{1}^T \right). \end{aligned}$$

The result now follows from $Y \succeq 0$.

Conversely, suppose X satisfies

$$\mathbf{diag}(X) = 0, \quad \left(I - \frac{1}{n} \mathbf{1} \mathbf{1}^T \right) X \left(I - \frac{1}{n} \mathbf{1} \mathbf{1}^T \right) \preceq 0.$$

Define

$$\begin{aligned} Y &= -\frac{1}{2}(I - \frac{1}{n}\mathbf{1}\mathbf{1}^T)X(I - \frac{1}{n}\mathbf{1}\mathbf{1}^T) \\ &= -\frac{1}{2}\left(X - \frac{1}{n}X\mathbf{1}\mathbf{1}^T - \frac{1}{n}\mathbf{1}\mathbf{1}^T X + \frac{1}{n^2}(\mathbf{1}^T X \mathbf{1})\mathbf{1}\mathbf{1}^T\right). \end{aligned}$$

Then $Y \succeq 0$, and

$$\begin{aligned} \text{diag}(Y) &= \frac{1}{n}\left(X\mathbf{1} - \frac{1}{2n}(\mathbf{1}^T X \mathbf{1})\mathbf{1}\right) \\ \text{diag}(Y)\mathbf{1}^T + \mathbf{1} \text{diag}(Y)^T - 2Y &= \frac{1}{n}(X\mathbf{1})\mathbf{1}^T + \frac{1}{n}\mathbf{1}(X\mathbf{1})^T - \frac{1}{n^2}(\mathbf{1}^T X \mathbf{1})\mathbf{1}\mathbf{1}^T \\ &\quad + X - \frac{1}{n}(X\mathbf{1})\mathbf{1}^T - \frac{1}{n}\mathbf{1}(X\mathbf{1})^T + \frac{1}{n^2}(\mathbf{1}^T X \mathbf{1})\mathbf{1}\mathbf{1}^T \\ &= X. \end{aligned}$$

Therefore X is a Euclidean distance matrix.

- (e) To simplify the notation, we show that the matrix W with elements $w_{ik} = \exp(-x_{ik}/2)$ is positive semidefinite. Since the Euclidean distance matrices form a cone, this is equivalent to the result in the problem statement.

Following the hint we can write

$$w_{ik} = e^{-\frac{1}{2}\|p_i - p_k\|_2^2} = \mathbf{E} e^{jz^T(p_i - p_k)},$$

or, in matrix notation,

$$W = \mathbf{E} \begin{bmatrix} e^{jz^T p_1} \\ e^{jz^T p_2} \\ \vdots \\ e^{jz^T p_n} \end{bmatrix} \begin{bmatrix} e^{jz^T p_1} \\ e^{jz^T p_2} \\ \vdots \\ e^{jz^T p_n} \end{bmatrix}^H$$

where the subscript H denotes complex conjugate transpose. This shows that the matrix W is positive semidefinite, since for all u ,

$$\begin{aligned} u^T W u &= \mathbf{E} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix}^T \begin{bmatrix} e^{jz^T p_1} \\ e^{jz^T p_2} \\ \vdots \\ e^{jz^T p_n} \end{bmatrix} \begin{bmatrix} e^{jz^T p_1} \\ e^{jz^T p_2} \\ \vdots \\ e^{jz^T p_n} \end{bmatrix}^H \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix} \\ &= \mathbf{E} \left| \sum_k u_k e^{jz^T p_k} \right|^2 \\ &\geq 0. \end{aligned}$$

- 7.3 Minimum total covering ball volume.** We consider a collection of n points with locations $x_1, \dots, x_n \in \mathbf{R}^k$. We are also given a set of m groups or subsets of these points, $G_1, \dots, G_m \subseteq \{1, \dots, n\}$. For each group, let V_i be the volume of the smallest Euclidean ball that contains the points in group G_i . (The volume of a Euclidean ball of radius r in \mathbf{R}^k is $a_k r^k$, where a_k is known constant that

is positive but otherwise irrelevant here.) We let $V = V_1 + \dots + V_m$ be the total volume of these minimal covering balls.

The points x_{k+1}, \dots, x_n are fixed (*i.e.*, they are problem data). The variables to be chosen are x_1, \dots, x_k . Formulate the problem of choosing x_1, \dots, x_k , in order to minimize the total minimal covering ball volume V , as a convex optimization problem. Be sure to explain any new variables you introduce, and to justify the convexity of your objective and inequality constraint functions.

Solution. We introduce two new variables for each group, $z_i \in \mathbf{R}^k$ (the center of the covering ball), and $r_i \in \mathbf{R}$ (the radius). The Euclidean ball with center z_i and radius r_i covers the group G_i of points if and only if

$$\|z_i - x_j\|_2 \leq r_i \quad \text{for all } j \in G_i.$$

Therefore our problem can be stated as

$$\begin{aligned} & \text{minimize} && a_k \sum_{i=1}^n r_i^k \\ & \text{subject to} && \|z_i - x_j\|_2 \leq r_i \quad \text{for } j \in G_i, \quad i = 1, \dots, m. \end{aligned}$$

The variables here are $z_1, \dots, z_m, r_1, \dots, r_m$, and x_1, \dots, x_k . The problem data are x_{k+1}, \dots, x_n , and the groups G_1, \dots, G_m . The constraints are clearly convex, since $\|z_i - x_j\|_2$ is convex in the variables, and the righthand side, r_i , is linear. The objective is convex, since $r_i \geq 0$. (For k odd, the objective is *not* convex over all of \mathbf{R}^m , but it is convex on \mathbf{R}_+^m .)

7.4 Maximum-margin multiclass classification. In an m -category pattern classification problem, we are given m sets $C_i \subseteq \mathbf{R}^n$. Set C_i contains N_i examples of feature vectors in class i . The learning problem is to find a decision function $f : \mathbf{R}^n \rightarrow \{1, 2, \dots, m\}$ that maps each training example to its class, and also generalizes reliably to feature vectors that are not included in the training sets C_i .

(a) A common type of decision function for two-way classification is

$$f(x) = \begin{cases} 1 & \text{if } a^T x + b > 0 \\ 2 & \text{if } a^T x + b < 0. \end{cases}$$

In the simplest form, finding f is equivalent to solving a feasibility problem: find a and b such that

$$\begin{aligned} a^T x + b &> 0 && \text{if } x \in C_1 \\ a^T x + b &< 0 && \text{if } x \in C_2. \end{aligned}$$

Since these strict inequalities are homogeneous in a and b , they are feasible if and only if the nonstrict inequalities

$$\begin{aligned} a^T x + b &\geq 1 && \text{if } x \in C_1 \\ a^T x + b &\leq -1 && \text{if } x \in C_2 \end{aligned}$$

are feasible. This is a feasibility problem with $N_1 + N_2$ linear inequalities in $n + 1$ variables a , b .

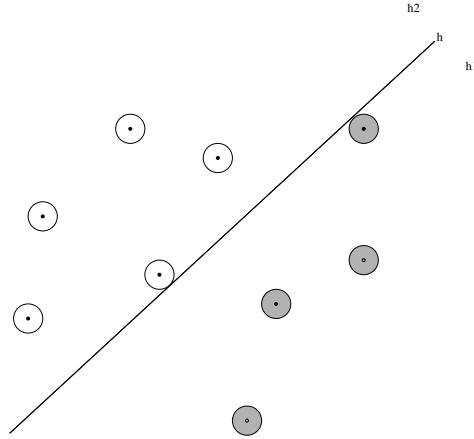
As an extension that improves the robustness (*i.e.*, generalization capability) of the classifier, we can impose the condition that the decision function f classifies all points in a neighborhood

of C_1 and C_2 correctly, and we can maximize the size of the neighborhood. This problem can be expressed as

$$\begin{aligned} & \text{maximize} && t \\ & \text{subject to} && a^T x + b > 0 \text{ if } \mathbf{dist}(x, C_1) \leq t, \\ & && a^T x + b < 0 \text{ if } \mathbf{dist}(x, C_2) \leq t, \end{aligned}$$

where $\mathbf{dist}(x, C) = \min_{y \in C} \|x - y\|_2$.

This is illustrated in the figure. The centers of the shaded disks form the set C_1 . The centers of the other disks form the set C_2 . The set of points at a distance less than t from C_i is the union of disks with radius t and center in C_i . The hyperplane in the figure separates the two expanded sets. We are interested in expanding the circles as much as possible, until the two expanded sets are no longer separable by a hyperplane.



Since the constraints are homogeneous in a , b , we can again replace them with nonstrict inequalities

$$\begin{aligned} & \text{maximize} && t \\ & \text{subject to} && a^T x + b \geq 1 \text{ if } \mathbf{dist}(x, C_1) \leq t, \\ & && a^T x + b \leq -1 \text{ if } \mathbf{dist}(x, C_2) \leq t. \end{aligned} \tag{36}$$

The variables are a , b , and t .

- (b) Next we consider an extension to more than two classes. If $m > 2$ we can use a decision function

$$f(x) = \operatorname{argmax}_{i=1,\dots,m} (a_i^T x + b_i),$$

parameterized by m vectors $a_i \in \mathbf{R}^n$ and m scalars b_i . To find f , we can solve a feasibility problem: find a_i , b_i , such that

$$a_i^T x + b_i > \max_{j \neq i} (a_j^T x + b_j) \quad \text{if } x \in C_i, \quad i = 1, \dots, m,$$

or, equivalently,

$$a_i^T x + b_i \geq 1 + \max_{j \neq i} (a_j^T x + b_j) \quad \text{if } x \in C_i, \quad i = 1, \dots, m.$$

Similarly as in part (a), we consider a robust version of this problem:

$$\begin{aligned} & \text{maximize} \quad t \\ & \text{subject to} \quad a_i^T x + b_i \geq 1 + \max_{j \neq i} (a_j^T x + b_j) \text{ if } \mathbf{dist}(x, C_i) \leq t, \\ & \quad \quad \quad i = 1, \dots, m. \end{aligned} \quad (37)$$

The variables in the problem are $a_i \in \mathbf{R}^n$, $b_i \in \mathbf{R}$, $i = 1, \dots, m$, and t .

Formulate the optimization problems (36) and (37) as SOCPs (if possible), or as quasiconvex optimization problems involving SOCP feasibility problems (otherwise).

Solution.

(a) The constraint

$$a^T x + b \geq 1 \text{ if } \mathbf{dist}(x, C_1) \leq t$$

can be written as

$$\inf_{\|u\|_2 \leq t} a^T(x + u) + b \geq 1 \text{ if } x \in C_1.$$

This is equivalent to

$$a^T x - t\|a\|_2 + b \geq 1 \text{ if } x \in C_1.$$

Similarly, the constraint

$$a^T x + b \leq -1 \text{ if } \mathbf{dist}(x, C_2) \leq t$$

is equivalent to

$$a^T x + t\|a\|_2 + b \leq -1 \text{ if } x \in C_2.$$

The problem is therefore equivalent to

$$\begin{aligned} & \text{maximize} \quad t \\ & \text{subject to} \quad a^T x + b \geq 1 + t\|a\|_2 \quad \text{if } x \in C_1 \\ & \quad \quad \quad a^T x + b \leq -(1 + t\|a\|_2) \quad \text{if } x \in C_2. \end{aligned}$$

This is a quasiconvex optimization problem. It can be solved by bisection on t , via a sequence of SOCP feasibility problems.

A better method is to make the problem convex by a change of variables

$$\hat{a} = \frac{1}{1+t\|a\|_2}a, \quad \hat{b} = \frac{1}{1+t\|a\|_2}b.$$

We have to add the constraint $\|\hat{a}\|_2 \leq 1/t$ to ensure that the change of variables is invertible. This gives

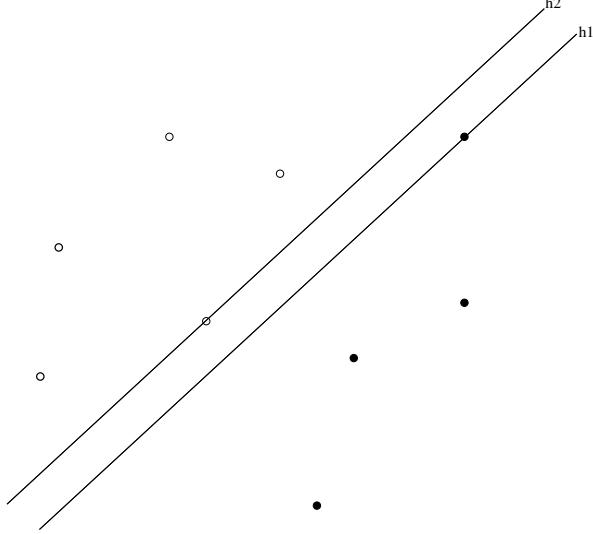
$$\begin{aligned} & \text{maximize} \quad t \\ & \text{subject to} \quad \hat{a}^T x + \hat{b} \geq 1 \quad \forall x \in C_1, \\ & \quad \quad \quad \hat{a}^T x + \hat{b} \leq -1 \quad \forall x \in C_2 \\ & \quad \quad \quad \|\hat{a}\|_2 \leq 1/t, \end{aligned}$$

which is still not convex. However it is equivalent to

$$\begin{aligned} & \text{minimize} \quad \|\hat{a}\|_2 \\ & \text{subject to} \quad \hat{a}^T x + \hat{b} \geq 1 \quad \forall x \in C_1, \\ & \quad \quad \quad \hat{a}^T x + \hat{b} \leq -1 \quad \forall x \in C_2. \end{aligned}$$

The optimal t is $1/\|\hat{a}\|_2$. If we square the objective of this problem it is a QP.

This last problem can be interpreted as follows. By minimizing $\|\hat{a}\|_2$ we maximize the distance between the hyperplanes $\hat{a}^T z + \hat{b} = 1$ and $\hat{a}^T z + \hat{b} = -1$. (Recall from homework 1 that the distance is $2/\|\hat{a}\|_2$.)



(b) We first write the problem as

$$\begin{aligned} & \text{minimize } t \\ & \text{subject to } a_i^T x + b_i \geq 1 + a_j^T x + b_j \text{ if } \mathbf{dist}(x, C_i) \leq t, i, j = 1, \dots, m, j \neq i. \end{aligned}$$

Using similar arguments as in part (a), we can write this as a quasiconvex optimization problem

$$\begin{aligned} & \text{maximize } t \\ & \text{subject to } (a_i - a_j)^T x + b_i - b_j \geq 1 + t \|a_i - a_j\|_2 \\ & \quad \text{if } x \in C_i, i, j = 1, \dots, m, j \neq i. \end{aligned}$$

This problem can be solved by bisection in t . Each bisection step reduces to an SOCP feasibility problem with variables $a_i, b_i, i = 1, \dots, m$.

7.5 Three-way linear classification. We are given data

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

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:

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

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,

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

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. Solve the specific instance of the 3-way separation problem given in `sep3way_data.m`, with the columns of the matrices X , Y and Z giving the $x^{(j)}$, $j = 1, \dots, N$, $y^{(j)}$, $j = 1, \dots, M$ and $z^{(j)}$, $j = 1, \dots, P$. To save you the trouble of plotting data points and separation boundaries, we have included the plotting code in `sep3way_data.m`. (Note that $a1$, $a2$, $a3$, $b1$ and $b2$ contain arbitrary numbers; you should compute the correct values using CVX.)

Solution. The inequalities

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

are homogeneous in a_i and b_i so we can express them as

$$\begin{aligned} f_1(x^{(j)}) &\geq \max\{f_2(x^{(j)}), f_3(x^{(j)})\} + 1, \quad j = 1, \dots, N, \\ f_2(y^{(j)}) &\geq \max\{f_1(y^{(j)}), f_3(y^{(j)})\} + 1, \quad j = 1, \dots, M, \\ f_3(z^{(j)}) &\geq \max\{f_1(z^{(j)}), f_2(z^{(j)})\} + 1, \quad j = 1, \dots, P. \end{aligned}$$

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 $a_1 + a_2 + a_3 = 0$ and $b_1 + b_2 + b_3 = 0$.

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

```
clear all; close all;
% data for problem instance
M = 20;
N = 20;
P = 20;

X = [
    3.5674    4.1253    2.8535    5.1892    4.3273    3.8133    3.4117 ...
    1.2345    2.5678    3.9812    4.7654    5.0123    6.1234    7.2345 ...
    8.3456    9.0123    10.5678   11.2345   12.3456   13.4567   14.5678 ...
    15.6789   16.3456   17.0123   18.5678   19.2345   20.3456   21.4567 ...
    22.5678   23.2345   24.0123   25.5678   26.2345   27.3456   28.4567 ...
    29.5678   30.2345   31.0123   32.5678   33.2345   34.3456   35.4567 ...
    36.5678   37.2345   38.0123   39.5678   40.2345   41.3456   42.4567 ...
    43.5678   44.2345   45.0123   46.5678   47.2345   48.3456   49.4567 ...
    50.5678   51.2345   52.0123   53.5678   54.2345   55.3456   56.4567 ...
    57.5678   58.2345   59.0123   60.5678   61.2345   62.3456   63.4567 ...
    64.5678   65.2345   66.0123   67.5678   68.2345   69.3456   70.4567 ...
    71.5678   72.2345   73.0123   74.5678   75.2345   76.3456   77.4567 ...
    78.5678   79.2345   80.0123   81.5678   82.2345   83.3456   84.4567 ...
    85.5678   86.2345   87.0123   88.5678   89.2345   90.3456   91.4567 ...
    92.5678   93.2345   94.0123   95.5678   96.2345   97.3456   98.4567 ...
    99.5678   100.2345  101.0123  102.5678  103.2345  104.3456  105.4567 ...
    106.5678  107.2345  108.0123  109.5678  110.2345  111.3456  112.4567 ...
    113.5678  114.2345  115.0123  116.5678  117.2345  118.3456  119.4567 ...
    120.5678  121.2345  122.0123  123.5678  124.2345  125.3456  126.4567 ...
    127.5678  128.2345  129.0123  130.5678  131.2345  132.3456  133.4567 ...
    134.5678  135.2345  136.0123  137.5678  138.2345  139.3456  140.4567 ...
    141.5678  142.2345  143.0123  144.5678  145.2345  146.3456  147.4567 ...
    148.5678  149.2345  150.0123  151.5678  152.2345  153.3456  154.4567 ...
    155.5678  156.2345  157.0123  158.5678  159.2345  160.3456  161.4567 ...
    162.5678  163.2345  164.0123  165.5678  166.2345  167.3456  168.4567 ...
    169.5678  170.2345  171.0123  172.5678  173.2345  174.3456  175.4567 ...
    176.5678  177.2345  178.0123  179.5678  180.2345  181.3456  182.4567 ...
    183.5678  184.2345  185.0123  186.5678  187.2345  188.3456  189.4567 ...
    190.5678  191.2345  192.0123  193.5678  194.2345  195.3456  196.4567 ...
    197.5678  198.2345  199.0123  200.5678  201.2345  202.3456  203.4567 ...
    204.5678  205.2345  206.0123  207.5678  208.2345  209.3456  210.4567 ...
    211.5678  212.2345  213.0123  214.5678  215.2345  216.3456  217.4567 ...
    218.5678  219.2345  220.0123  221.5678  222.2345  223.3456  224.4567 ...
    225.5678  226.2345  227.0123  228.5678  229.2345  230.3456  231.4567 ...
    232.5678  233.2345  234.0123  235.5678  236.2345  237.3456  238.4567 ...
    239.5678  240.2345  241.0123  242.5678  243.2345  244.3456  245.4567 ...
    246.5678  247.2345  248.0123  249.5678  250.2345  251.3456  252.4567 ...
    253.5678  254.2345  255.0123  256.5678  257.2345  258.3456  259.4567 ...
    260.5678  261.2345  262.0123  263.5678  264.2345  265.3456  266.4567 ...
    267.5678  268.2345  269.0123  270.5678  271.2345  272.3456  273.4567 ...
    274.5678  275.2345  276.0123  277.5678  278.2345  279.3456  280.4567 ...
    281.5678  282.2345  283.0123  284.5678  285.2345  286.3456  287.4567 ...
    288.5678  289.2345  290.0123  291.5678  292.2345  293.3456  294.4567 ...
    295.5678  296.2345  297.0123  298.5678  299.2345  300.3456  301.4567 ...
    302.5678  303.2345  304.0123  305.5678  306.2345  307.3456  308.4567 ...
    309.5678  310.2345  311.0123  312.5678  313.2345  314.3456  315.4567 ...
    316.5678  317.2345  318.0123  319.5678  320.2345  321.3456  322.4567 ...
    323.5678  324.2345  325.0123  326.5678  327.2345  328.3456  329.4567 ...
    330.5678  331.2345  332.0123  333.5678  334.2345  335.3456  336.4567 ...
    337.5678  338.2345  339.0123  340.5678  341.2345  342.3456  343.4567 ...
    344.5678  345.2345  346.0123  347.5678  348.2345  349.3456  350.4567 ...
    351.5678  352.2345  353.0123  354.5678  355.2345  356.3456  357.4567 ...
    358.5678  359.2345  360.0123  361.5678  362.2345  363.3456  364.4567 ...
    365.5678  366.2345  367.0123  368.5678  369.2345  370.3456  371.4567 ...
    372.5678  373.2345  374.0123  375.5678  376.2345  377.3456  378.4567 ...
    379.5678  380.2345  381.0123  382.5678  383.2345  384.3456  385.4567 ...
    386.5678  387.2345  388.0123  389.5678  390.2345  391.3456  392.4567 ...
    393.5678  394.2345  395.0123  396.5678  397.2345  398.3456  399.4567 ...
    400.5678  401.2345  402.0123  403.5678  404.2345  405.3456  406.4567 ...
    407.5678  408.2345  409.0123  410.5678  411.2345  412.3456  413.4567 ...
    414.5678  415.2345  416.0123  417.5678  418.2345  419.3456  420.4567 ...
    421.5678  422.2345  423.0123  424.5678  425.2345  426.3456  427.4567 ...
    428.5678  429.2345  430.0123  431.5678  432.2345  433.3456  434.4567 ...
    435.5678  436.2345  437.0123  438.5678  439.2345  440.3456  441.4567 ...
    442.5678  443.2345  444.0123  445.5678  446.2345  447.3456  448.4567 ...
    449.5678  450.2345  451.0123  452.5678  453.2345  454.3456  455.4567 ...
    456.5678  457.2345  458.0123  459.5678  460.2345  461.3456  462.4567 ...
    463.5678  464.2345  465.0123  466.5678  467.2345  468.3456  469.4567 ...
    470.5678  471.2345  472.0123  473.5678  474.2345  475.3456  476.4567 ...
    477.5678  478.2345  479.0123  480.5678  481.2345  482.3456  483.4567 ...
    484.5678  485.2345  486.0123  487.5678  488.2345  489.3456  490.4567 ...
    491.5678  492.2345  493.0123  494.5678  495.2345  496.3456  497.4567 ...
    498.5678  499.2345  500.0123  501.5678  502.2345  503.3456  504.4567 ...
    505.5678  506.2345  507.0123  508.5678  509.2345  510.3456  511.4567 ...
    512.5678  513.2345  514.0123  515.5678  516.2345  517.3456  518.4567 ...
    519.5678  520.2345  521.0123  522.5678  523.2345  524.3456  525.4567 ...
    526.5678  527.2345  528.0123  529.5678  530.2345  531.3456  532.4567 ...
    533.5678  534.2345  535.0123  536.5678  537.2345  538.3456  539.4567 ...
    540.5678  541.2345  542.0123  543.5678  544.2345  545.3456  546.4567 ...
    547.5678  548.2345  549.0123  550.5678  551.2345  552.3456  553.4567 ...
    554.5678  555.2345  556.0123  557.5678  558.2345  559.3456  560.4567 ...
    561.5678  562.2345  563.0123  564.5678  565.2345  566.3456  567.4567 ...
    568.5678  569.2345  570.0123  571.5678  572.2345  573.3456  574.4567 ...
    575.5678  576.2345  577.0123  578.5678  579.2345  580.3456  581.4567 ...
    582.5678  583.2345  584.0123  585.5678  586.2345  587.3456  588.4567 ...
    589.5678  590.2345  591.0123  592.5678  593.2345  594.3456  595.4567 ...
    596.5678  597.2345  598.0123  599.5678  600.2345  601.3456  602.4567 ...
    603.5678  604.2345  605.0123  606.5678  607.2345  608.3456  609.4567 ...
    610.5678  611.2345  612.0123  613.5678  614.2345  615.3456  616.4567 ...
    617.5678  618.2345  619.0123  620.5678  621.2345  622.3456  623.4567 ...
    624.5678  625.2345  626.0123  627.5678  628.2345  629.3456  630.4567 ...
    631.5678  632.2345  633.0123  634.5678  635.2345  636.3456  637.4567 ...
    638.5678  639.2345  640.0123  641.5678  642.2345  643.3456  644.4567 ...
    645.5678  646.2345  647.0123  648.5678  649.2345  650.3456  651.4567 ...
    652.5678  653.2345  654.0123  655.5678  656.2345  657.3456  658.4567 ...
    659.5678  660.2345  661.0123  662.5678  663.2345  664.3456  665.4567 ...
    666.5678  667.2345  668.0123  669.5678  670.2345  671.3456  672.4567 ...
    673.5678  674.2345  675.0123  676.5678  677.2345  678.3456  679.4567 ...
    680.5678  681.2345  682.0123  683.5678  684.2345  685.3456  686.4567 ...
    687.5678  688.2345  689.0123  690.5678  691.2345  692.3456  693.4567 ...
    694.5678  695.2345  696.0123  697.5678  698.2345  699.3456  700.4567 ...
    701.5678  702.2345  703.0123  704.5678  705.2345  706.3456  707.4567 ...
    708.5678  709.2345  710.0123  711.5678  712.2345  713.3456  714.4567 ...
    715.5678  716.2345  717.0123  718.5678  719.2345  720.3456  721.4567 ...
    722.5678  723.2345  724.0123  725.5678  726.2345  727.3456  728.4567 ...
    729.5678  730.2345  731.0123  732.5678  733.2345  734.3456  735.4567 ...
    736.5678  737.2345  738.0123  739.5678  740.2345  741.3456  742.4567 ...
    743.5678  744.2345  745.0123  746.5678  747.2345  748.3456  749.4567 ...
    750.5678  751.2345  752.0123  753.5678  754.2345  755.3456  756.4567 ...
    757.5678  758.2345  759.0123  760.5678  761.2345  762.3456  763.4567 ...
    764.5678  765.2345  766.0123  767.5678  768.2345  769.3456  770.4567 ...
    771.5678  772.2345  773.0123  774.5678  775.2345  776.3456  777.4567 ...
    778.5678  779.2345  780.0123  781.5678  782.2345  783.3456  784.4567 ...
    785.5678  786.2345  787.0123  788.5678  789.2345  790.3456  791.4567 ...
    792.5678  793.2345  794.0123  795.5678  796.2345  797.3456  798.4567 ...
    799.5678  800.2345  801.0123  802.5678  803.2345  804.3456  805.4567 ...
    806.5678  807.2345  808.0123  809.5678  810.2345  811.3456  812.4567 ...
    813.5678  814.2345  815.0123  816.5678  817.2345  818.3456  819.4567 ...
    820.5678  821.2345  822.0123  823.5678  824.2345  825.3456  826.4567 ...
    827.5678  828.2345  829.0123  830.5678  831.2345  832.3456  833.4567 ...
    834.5678  835.2345  836.0123  837.5678  838.2345  839.3456  840.4567 ...
    841.5678  842.2345  843.0123  844.5678  845.2345  846.3456  847.4567 ...
    848.5678  849.2345  850.0123  851.5678  852.2345  853.3456  854.4567 ...
    855.5678  856.2345  857.0123  858.5678  859.2345  860.3456  861.4567 ...
    862.5678  863.2345  864.0123  865.5678  866.2345  867.3456  868.4567 ...
    869.5678  870.2345  871.0123  872.5678  873.2345  874.3456  875.4567 ...
    876.5678  877.2345  878.0123  879.5678  880.2345  881.3456  882.4567 ...
    883.5678  884.2345  885.0123  886.5678  887.2345  888.3456  889.4567 ...
    890.5678  891.2345  892.0123  893.5678  894.2345  895.3456  896.4567 ...
    897.5678  898.2345  899.0123  900.5678  901.2345  902.3456  903.4567 ...
    904.5678  905.2345  906.0123  907.5678  908.2345  909.3456  910.4567 ...
    911.5678  912.2345  913.0123  914.5678  915.2345  916.3456  917.4567 ...
    918.5678  919.2345  920.0123  921.5678  922.2345  923.3456  924.4567 ...
    925.5678  926.2345  927.0123  928.5678  929.2345  930.3456  931.4567 ...
    932.5678  933.2345  934.0123  935.5678  936.2345  937.3456  938.4567 ...
    939.5678  940.2345  941.0123  942.5678  943.2345  944.3456  945.4567 ...
    946.5678  947.2345  948.0123  949.5678  950.2345  951.3456  952.4567 ...
    953.5678  954.2345  955.0123  956.5678  957.2345  958.3456  959.4567 ...
    960.5678  961.2345  962.0123  963.5678  964.2345  965.3456  966.4567 ...
    967.5678  968.2345  969.0123  970.5678  971.2345  972.3456  973.4567 ...
    974.5678  975.2345  976.0123  977.5678  978.2345  979.3456  980.4567 ...
    981.5678  982.2345
```

```

3.8636    5.0668    3.9044    4.2944    4.7143    3.3082    5.2540 ...
2.5590    3.6001    4.8156    5.2902    5.1908    3.9802 ;...
-2.9981    0.5178    2.1436   -0.0677    0.3144    1.3064    3.9297 ...
0.2051    0.1067   -1.4982   -2.4051    2.9224    1.5444   -2.8687 ...
1.0281    1.2420    1.2814    1.2035   -2.1644   -0.2821];

Y = [
-4.5665   -3.6904   -3.2881   -1.6491   -5.4731   -3.6170   -1.1876 ...
-1.0539   -1.3915   -2.0312   -1.9999   -0.2480   -1.3149   -0.8305 ...
-1.9355   -1.0898   -2.6040   -4.3602   -1.8105   0.3096; ...
2.4117    4.2642    2.8460    0.5250    1.9053    2.9831    4.7079 ...
0.9702    0.3854    1.9228    1.4914   -0.9984    3.4330    2.9246 ...
3.0833    1.5910    1.5266    1.6256    2.5037    1.4384];

Z = [
1.7451    2.6345    0.5937   -2.8217    3.0304    1.0917   -1.7793 ...
1.2422    2.1873   -2.3008   -3.3258    2.7617    0.9166    0.0601 ...
-2.6520   -3.3205    4.1229   -3.4085   -3.1594   -0.7311; ...
-3.2010   -4.9921   -3.7621   -4.7420   -4.1315   -3.9120   -4.5596 ...
-4.9499   -3.4310   -4.2656   -6.2023   -4.5186   -3.7659   -5.0039 ...
-4.3744   -5.0559   -3.9443   -4.0412   -5.3493   -3.0465];

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

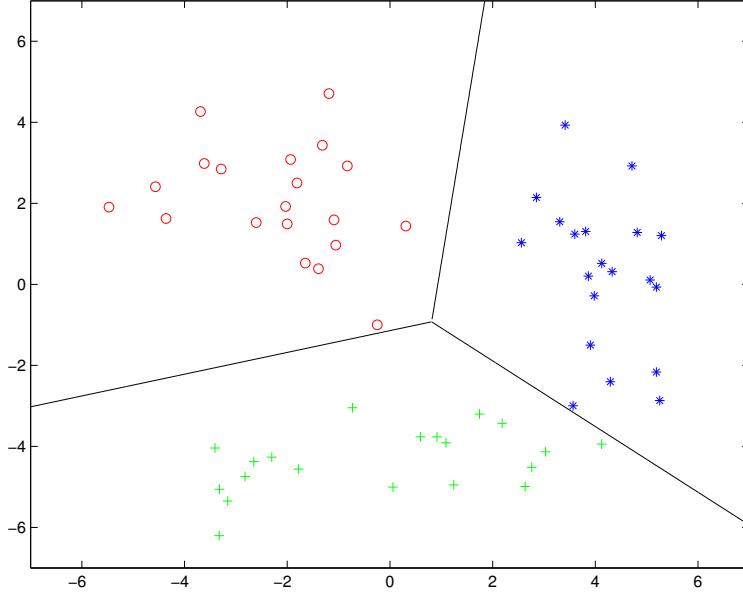
% now let's plot the three-way separation induced by
% a1,a2,a3,b1,b2,b3
% find maximally confusing point
p = [(a1-a2)';(a1-a3)']\[(b1-b2);(b1-b3)];;

% plot
t = [-7:0.01:7];
u1 = a1-a2; u2 = a2-a3; u3 = a3-a1;
v1 = b1-b2; v2 = b2-b3; v3 = b3-b1;
line1 = (-t*u1(1)+v1)/u1(2); idx1 = find(u2'*[t;line1]-v2>0);
line2 = (-t*u2(1)+v2)/u2(2); idx2 = find(u3'*[t;line2]-v3>0);
line3 = (-t*u3(1)+v3)/u3(2); idx3 = find(u1'*[t;line3]-v1>0);
plot(X(1,:),X(2,:),'*',Y(1,:),Y(2,:),'ro',Z(1,:),Z(2,:),'g+',...
t(idx1),line1(idx1),'k',t(idx2),line2(idx2),'k',t(idx3),line3(idx3),'k');

```

```
axis([-7 7 -7 7]);
```

The following figure is generated.



7.6 Feature selection and sparse linear separation. Suppose $x^{(1)}, \dots, x^{(N)}$ and $y^{(1)}, \dots, y^{(M)}$ 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

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

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

$$\begin{aligned} & \text{minimize} && \|a\|_2 \\ & \text{subject to} && a^T x^{(i)} - b \geq 1, \quad i = 1, \dots, N \\ & && a^T y^{(i)} - b \leq -1, \quad i = 1, \dots, M, \end{aligned}$$

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 j th feature is not used to carry out classification. So a sparse a corresponds to a classification function that is parsimonious; 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

$$\begin{aligned} & \text{minimize} && \|a\|_2 + \lambda \|a\|_1 \\ & \text{subject to} && a^T x^{(i)} - b \geq 1, \quad i = 1, \dots, N \\ & && a^T y^{(i)} - b \leq -1, \quad i = 1, \dots, M, \end{aligned}$$

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 `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 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
```

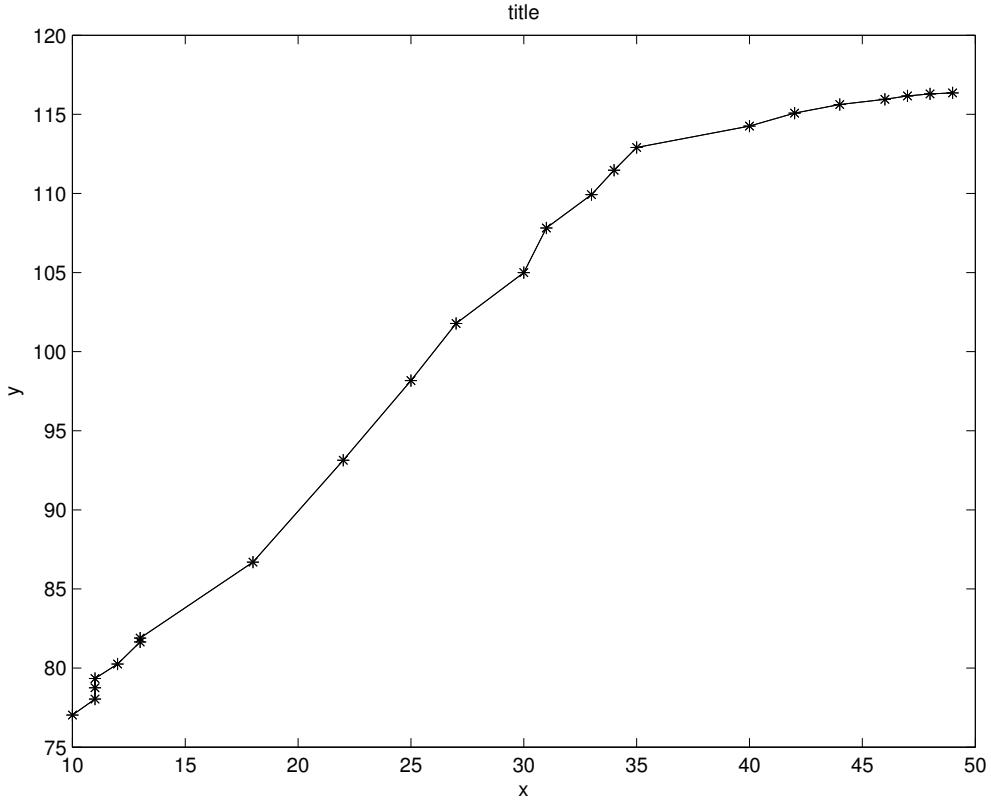
```

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.

7.7 Thickest slab separating two sets. We are given two sets in \mathbf{R}^n : a polyhedron

$$C_1 = \{x \mid Cx \preceq d\},$$

defined by a matrix $C \in \mathbf{R}^{m \times n}$ and a vector $d \in \mathbf{R}^m$, and an ellipsoid

$$C_2 = \{Pu + q \mid \|u\|_2 \leq 1\},$$

defined by a matrix $P \in \mathbf{R}^{n \times n}$ and a vector $q \in \mathbf{R}^n$. We assume that the sets are nonempty and that they do not intersect. We are interested in the optimization problem

$$\begin{aligned} & \text{maximize} && \inf_{x \in C_1} a^T x - \sup_{x \in C_2} a^T x \\ & \text{subject to} && \|a\|_2 = 1. \end{aligned}$$

with variable $a \in \mathbf{R}^n$.

Explain how you would solve this problem. You can answer the question by reducing the problem to a standard problem class (LP, QP, SOCP, SDP, ...), or by describing an algorithm to solve it.

Remark. The geometrical interpretation is as follows. If we choose

$$b = \frac{1}{2} \left(\inf_{x \in C_1} a^T x + \sup_{x \in C_2} a^T x \right),$$

then the hyperplane $H = \{x \mid a^T x = b\}$ is the maximum margin separating hyperplane separating C_1 and C_2 . Alternatively, a gives us the thickest slab that separates the two sets.

Solution.

$$\begin{aligned} & \text{maximize} && -d^T z - \|P^T a\|_2 - q^T a \\ & \text{subject to} && \|a\|_2 \leq 1 \\ & && C^T z + a = 0 \\ & && z \succeq 0. \end{aligned}$$

An SOCP.

7.8 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 the 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 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 $A_i, b_i, c_i, d_i, \hat{v}_i, \rho_i$, 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 to 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

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

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

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

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

To get l_k we solve the LP

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

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

- (b) Here is a 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 script returns the following results:

```

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

```

7.9 Triangulation from multiple camera views. A projective camera can be described by a linear-fractional function $f : \mathbf{R}^3 \rightarrow \mathbf{R}^2$,

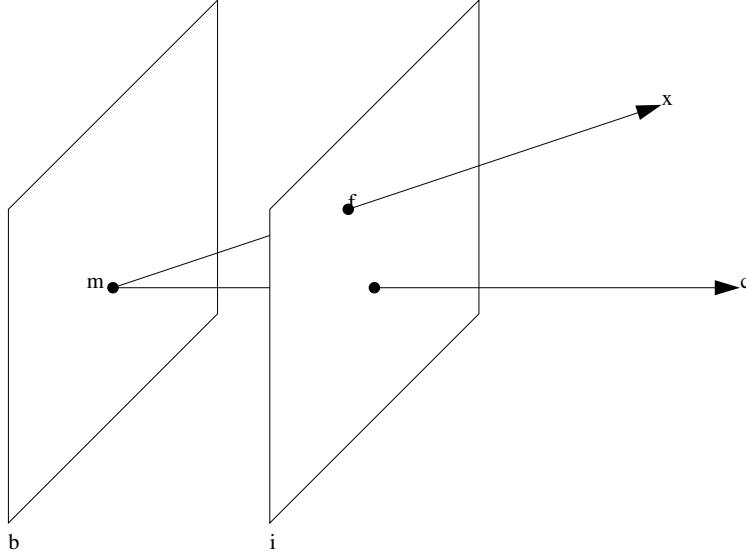
$$f(x) = \frac{1}{c^T x + d}(Ax + b), \quad \text{dom } f = \{x \mid c^T x + d > 0\},$$

with

$$\text{rank}\left(\begin{bmatrix} A \\ c^T \end{bmatrix}\right) = 3.$$

The domain of f consists of the points in front of the camera.

Before stating the problem, we give some background and interpretation, most of which will not be needed for the actual problem.



The 3×4 -matrix

$$P = \begin{bmatrix} A & b \\ c^T & d \end{bmatrix}$$

is called the *camera matrix* and has rank 3. Since f is invariant with respect to a scaling of P , we can normalize the parameters and assume, for example, that $\|c\|_2 = 1$. The numerator $c^T x + d$ is then the distance of x to the plane $\{z \mid c^T z + d = 0\}$. This plane is called the *principal plane*. The point

$$x_c = - \begin{bmatrix} A \\ c^T \end{bmatrix}^{-1} \begin{bmatrix} b \\ d \end{bmatrix}$$

lies in the principal plane and is called the *camera center*. The ray $\{x_c + \theta c \mid \theta \geq 0\}$, which is perpendicular to the principal plane, is the *principal axis*. We will define the *image plane* as the plane parallel to the principal plane, at a unit distance from it along the principal axis.

The point x' in the figure is the intersection of the image plane and the line through the camera center and x , and is given by

$$x' = x_c + \frac{1}{c^T(x - x_c)}(x - x_c).$$

Using the definition of x_c we can write $f(x)$ as

$$f(x) = \frac{1}{c^T(x - x_c)} A(x - x_c) = A(x' - x_c) = Ax' + b.$$

This shows that the mapping $f(x)$ can be interpreted as a projection of x on the image plane to get x' , followed by an affine transformation of x' . We can interpret $f(x)$ as the point x' expressed in some two-dimensional coordinate system attached to the image plane.

In this exercise we consider the problem of determining the position of a point $x \in \mathbf{R}^3$ from its image in N cameras. Each of the cameras is characterized by a known linear-fractional mapping f_k and camera matrix P_k :

$$f_k(x) = \frac{1}{c_k^T x + d_k} (A_k x + b_k), \quad P_k = \begin{bmatrix} A_k & b_k \\ c_k^T & d_k \end{bmatrix}, \quad k = 1, \dots, N.$$

The image of the point x in camera k is denoted $y^{(k)} \in \mathbf{R}^2$. Due to camera imperfections and calibration errors, we do not expect the equations $f_k(x) = y^{(k)}$, $k = 1, \dots, N$, to be exactly solvable. To estimate the point x we therefore minimize the maximum error in the N equations by solving

$$\text{minimize } g(x) = \max_{k=1, \dots, N} \|f_k(x) - y^{(k)}\|_2. \quad (38)$$

- (a) Show that (38) is a quasiconvex optimization problem. The variable in the problem is $x \in \mathbf{R}^3$. The functions f_k (*i.e.*, the parameters A_k , b_k , c_k , d_k) and the vectors $y^{(k)}$ are given.
- (b) Solve the following instance of (38) using CVX (and bisection): $N = 4$,

$$P_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \quad P_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 10 \end{bmatrix},$$

$$P_3 = \begin{bmatrix} 1 & 1 & 1 & -10 \\ -1 & 1 & 1 & 0 \\ -1 & -1 & 1 & 10 \end{bmatrix}, \quad P_4 = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & -1 & 1 & 0 \\ -1 & 0 & 0 & 10 \end{bmatrix},$$

$$y^{(1)} = \begin{bmatrix} 0.98 \\ 0.93 \end{bmatrix}, \quad y^{(2)} = \begin{bmatrix} 1.01 \\ 1.01 \end{bmatrix}, \quad y^{(3)} = \begin{bmatrix} 0.95 \\ 1.05 \end{bmatrix}, \quad y^{(4)} = \begin{bmatrix} 2.04 \\ 0.00 \end{bmatrix}.$$

You can terminate the bisection when a point is found with accuracy $g(x) - p^* \leq 10^{-4}$, where p^* is the optimal value of (38).

Solution.

- (a) The constraint $g(x) \leq \alpha$ is equivalent to

$$\|A_k x + b_k + \alpha(c_k^T x + d_k)\|_2 \leq \alpha(c_k^T x + d_k), \quad k = 1, \dots, N.$$

This is a set N convex second-order cone constraints in x .

- (b) The CVX code printed below returns $x = (4.9, 5.0, 5.2)$ and $t = 0.0495$.

```
P1 = [1, 0, 0, 0; 0, 1, 0, 0; 0, 0, 1, 0];
P2 = [1, 0, 0, 0; 0, 0, 1, 0; 0, -1, 0, 10];
P3 = [1, 1, 1, -10; -1, 1, 1, 0; -1, -1, 1, 10];
P4 = [0, 1, 1, 0; 0, -1, 1, 0; -1, 0, 0, 10];

u1 = [0.98; 0.93];
u2 = [1.01; 1.01];
```

```

u3 = [0.95; 1.05];
u4 = [2.04; 0.00];

cvx_quiet(true);
l = 0; u = 1;
tol = 1e-5;
while u-l > tol
    t = (l+u)/2;
    cvx_begin
        variable x(3);
        y1 = P1*[x;1];
        norm( y1(1:2) - y1(3)*u1 ) <= t * y1(3);

        y2 = P2*[x;1];
        norm( y2(1:2) - y2(3)*u2 ) <= t * y2(3);

        y3 = P3*[x;1];
        norm( y3(1:2) - y3(3)*u3 ) <= t * y3(3);

        y4 = P4*[x;1];
        norm( y4(1:2) - y4(3)*u4 ) <= t * y4(3);
    cvx_end
    disp(cvx_status)
    if cvx_optval == Inf,
        l = t;
    else
        lastx = x;
        u = t;
    end;
end;
lastx

```

7.10 *Projection onto the probability simplex.* In this problem you will work out a simple method for finding the Euclidean projection y of $x \in \mathbf{R}^n$ onto the probability simplex $\mathcal{P} = \{z \mid z \succeq 0, \mathbf{1}^T z = 1\}$.

Hints. Consider the problem of minimizing $(1/2)\|y - x\|_2^2$ subject to $y \succeq 0, \mathbf{1}^T y = 1$. Form the partial Lagrangian

$$L(y, \nu) = (1/2)\|y - x\|_2^2 + \nu(\mathbf{1}^T y - 1),$$

leaving the constraint $y \succeq 0$ implicit. Show that $y = (x - \nu\mathbf{1})_+$ minimizes $L(y, \nu)$ over $y \succeq 0$.

Solution. To find the Euclidean projection of y onto \mathcal{P} , we solve the problem,

$$\begin{aligned} &\text{minimize} && (1/2)\|y - x\|_2^2 \\ &\text{subject to} && y \succeq 0, \quad \mathbf{1}^T y = 1, \end{aligned}$$

with variable y . The partial Lagrangian formed by ‘dualizing’ the constraint $\mathbf{1}^T y = 1$, is

$$L(y, \nu) = (1/2)\|y - x\|_2^2 + \nu(\mathbf{1}^T y - 1),$$

with the implicit constraint $y \succeq 0$. This can also be written as

$$L(y, \nu) = (1/2)\|y - (x - \nu\mathbf{1})\|_2^2 + \nu(\mathbf{1}^T x - 1) - n\nu^2.$$

To minimize $L(y, \nu)$ over $y \succeq 0$ we solve the problem

$$\begin{aligned} & \text{minimize} && (1/2)\|y - (x - \nu\mathbf{1})\|_2^2 \\ & \text{subject to} && y \succeq 0, \end{aligned}$$

with variable y . This is simply the Euclidean projection of $x - \nu\mathbf{1}$ onto \mathbf{R}_+^n , with the solution $\tilde{y} = (x - \nu\mathbf{1})_+$. Substituting \tilde{y} into $L(y, \nu)$, we obtain the dual function

$$\begin{aligned} g(\nu) &= (1/2)\|(x - \nu\mathbf{1})_+ - (x - \nu\mathbf{1})\|_2^2 + \nu(\mathbf{1}^T x - 1) - n\nu^2 \\ &= (1/2)\|(x - \nu\mathbf{1})_-\|_2^2 + \nu(\mathbf{1}^T x - 1) - n\nu^2. \end{aligned}$$

Since ν is a scalar, we can find ν^* by using a bisection method to maximize $g(\nu)$. The projection of x onto \mathcal{P} is given by $y^* = (x - \nu^*\mathbf{1})_+$.

7.11 Conformal mapping via convex optimization. Suppose that Ω is a closed bounded region in \mathbf{C} with no holes (*i.e.*, it is simply connected). The Riemann mapping theorem states that there exists a conformal mapping φ from Ω onto $D = \{z \in \mathbf{C} \mid |z| \leq 1\}$, the unit disk in the complex plane. (This means that φ is an analytic function, and maps Ω one-to-one onto D .)

One proof of the Riemann mapping theorem is based on an infinite dimensional optimization problem. We choose a point $a \in \text{int } \Omega$ (the interior of Ω). Among all analytic functions that map $\partial\Omega$ (the boundary of Ω) into D , we choose one that maximizes the magnitude of the derivative at a . Amazingly, it can be shown that this function is a conformal mapping of Ω onto D .

We can use this theorem to construct an approximate conformal mapping, by sampling the boundary of Ω , and by restricting the optimization to a finite-dimensional subspace of analytic functions. Let b_1, \dots, b_N be a set of points in $\partial\Omega$ (meant to be a sampling of the boundary). We will search only over polynomials of degree up to n ,

$$\hat{\varphi}(z) = \alpha_1 z^n + \alpha_2 z^{n-1} + \dots + \alpha_n z + \alpha_{n+1},$$

where $\alpha_1, \dots, \alpha_{n+1} \in \mathbf{C}$. With these approximations, we obtain the problem

$$\begin{aligned} & \text{maximize} && |\hat{\varphi}'(a)| \\ & \text{subject to} && |\hat{\varphi}(b_i)| \leq 1, \quad i = 1, \dots, N, \end{aligned}$$

with variables $\alpha_1, \dots, \alpha_{n+1} \in \mathbf{C}$. The problem data are $b_1, \dots, b_N \in \partial\Omega$ and $a \in \text{int } \Omega$.

- (a) Explain how to solve the problem above via convex or quasiconvex optimization.
- (b) Carry out your method on the problem instance given in `conf_map_data.m`. This file defines the boundary points b_i and plots them. It also contains code that will plot $\hat{\varphi}(b_i)$, the boundary of the mapped region, once you provide the values of α_j ; these points should be very close to the boundary of the unit disk. (Please turn in this plot, and give us the values of α_j that you find.) The function `polyval` may be helpful.

Remarks.

- We've been a little informal in our mathematics here, but it won't matter.
- You do not need to know any complex analysis to solve this problem; we've told you everything you need to know.
- A basic result from complex analysis tells us that $\hat{\varphi}$ is one-to-one if and only if the image of the boundary does not 'loop over' itself. (We mention this just for fun; we're not asking you to verify that the $\hat{\varphi}$ you find is one-to-one.)

Solution. The constraint functions can be written as

$$|\hat{\varphi}(b_i)| = \left| \alpha_1 b_i^n + \alpha_2 b_i^{n-1} + \cdots + \alpha_n b_i + \alpha_{n+1} \right|,$$

which are evidently convex in α , since $\hat{\varphi}(b_i)$ is a (complex) affine function of α . The objective as stated is

$$|\hat{\varphi}'(a)| = \left| \alpha_1 n a^{n-1} + \alpha_2 (n-1) a^{n-2} + \cdots + \alpha_n \right|,$$

which is not concave in α . But we observe that if α satisfies the constraints, then so does $\gamma\alpha$, where $|\gamma| = 1$. Therefore we can just as well maximize $\Re \hat{\varphi}'(a)$, or even insist that $\hat{\varphi}'(a)$ be real (either of these works). This yields

$$\begin{aligned} &\text{maximize} && \Re(\alpha_1 n a^{n-1} + \alpha_2 (n-1) a^{n-2} + \cdots + \alpha_n) \\ &\text{subject to} && \left| \alpha_1 b_i^n + \alpha_2 b_i^{n-1} + \cdots + \alpha_n b_i + \alpha_{n+1} \right| \leq 1, \quad i = 1, \dots, N, \end{aligned}$$

which is convex (in fact, an SOCP).

The code which solves this problem is given below:

```
% approximate conformal mapping via convex optimization
conf_map_data;

cvx_begin
variable alpha(n+1) complex; % polynomial coefficients
alpha_prime = alpha(1:n).*([n:-1:1]'); % coefficients of derivative
maximize (real(polyval(alpha_prime,a)));
norm(polyval(alpha,b),Inf) <= 1; % must map boundary points into unit disk
cvx_end
w = polyval(alpha,b); % (boundary of) mapped region

subplot(1,2,1)
plot(real(b),imag(b));
title('Boundary of region');
axis equal;
axis([-1.5 1.5 -1.5 1.5]);

subplot(1,2,2)
plot(real(w),imag(w), 'b',cos(theta),sin(theta), 'g');
title('Boundary of mapped region');
axis equal;
```

```

axis([-1.5 1.5 -1.5 1.5]);

%print('-depsc','conf_map.eps');

```

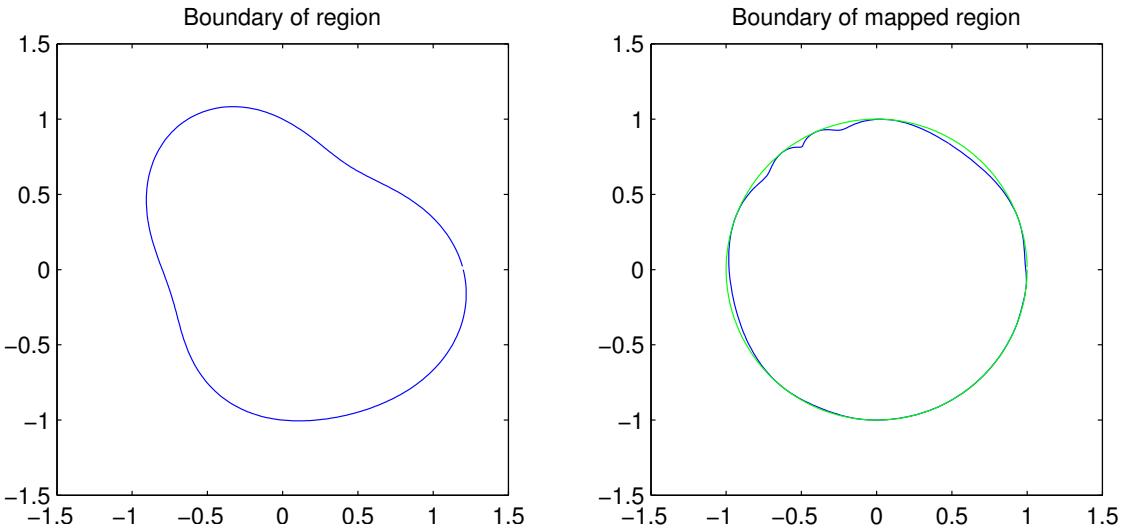
The coefficients α of the conformal mapping polynomial are

```

alpha =
-0.0015 + 0.0051i
-0.0074 - 0.0167i
0.0107 + 0.0091i
0.0206 - 0.0281i
0.0017 + 0.0740i
0.0031 - 0.0241i
-0.1263 + 0.0543i
0.0434 - 0.1671i
-0.1122 - 0.0056i
1.0430 - 0.0000i
0.0016 - 0.0026i

```

This gives us the following conformal mapping:



7.12 Fitting a vector field to given directions. This problem concerns a vector field on \mathbf{R}^n , i.e., a function $F : \mathbf{R}^n \rightarrow \mathbf{R}^n$. We are given the *direction* of the vector field at points $x^{(1)}, \dots, x^{(N)} \in \mathbf{R}^n$,

$$q^{(i)} = \frac{1}{\|F(x^{(i)})\|_2} F(x^{(i)}), \quad i = 1, \dots, N.$$

(These directions might be obtained, for example, from samples of trajectories of the differential equation $\dot{z} = F(z)$.) The goal is to fit these samples with a vector field of the form

$$\hat{F} = \alpha_1 F_1 + \cdots + \alpha_m F_m,$$

where $F_1, \dots, F_m : \mathbf{R}^n \rightarrow \mathbf{R}^n$ are given (basis) functions, and $\alpha \in \mathbf{R}^m$ is a set of coefficients that we will choose.

We will measure the fit using the maximum angle error,

$$J = \max_{i=1,\dots,N} |\angle(q^{(i)}, \hat{F}(x^{(i)}))|,$$

where $\angle(z, w) = \cos^{-1}((z^T w) / \|z\|_2 \|w\|_2)$ denotes the angle between nonzero vectors z and w . We are only interested in the case when J is smaller than $\pi/2$.

- (a) Explain how to choose α so as to minimize J using convex optimization. Your method can involve solving multiple convex problems. Be sure to explain how you handle the constraints $\hat{F}(x^{(i)}) \neq 0$.
- (b) Use your method to solve the problem instance with data given in `vfield_fit_data.m`, with an affine vector field fit, *i.e.*, $\hat{F}(z) = Az + b$. (The matrix A and vector b are the parameters α above.) Give your answer to the nearest degree, as in ' $20^\circ < J^* \leq 21^\circ$ '.

This file also contains code that plots the vector field directions, and also (but commented out) the directions of the vector field fit, $\hat{F}(x^{(i)}) / \|\hat{F}(x^{(i)})\|_2$. Create this plot, with your fitted vector field.

Solution. Let us work out the condition under which $J^* < \gamma$, where $\gamma \in (0, \pi/2)$. (And yes, we do mean to use strict inequality here.) Since \cos^{-1} is monotone decreasing on this region, we have $J^* < \gamma$ if and only if there exists α with

$$\frac{q^{(i)T} \hat{F}(x^{(i)})}{\|\hat{F}(x^{(i)})\|_2} > \cos(\gamma), \quad i = 1, \dots, N.$$

We write this as

$$q^{(i)T} \hat{F}(x^{(i)}) > \cos(\gamma) \|\hat{F}(x^{(i)})\|_2, \quad i = 1, \dots, N.$$

This is a set of *strict* SOC inequalities. Now we observe that the left and righthand sides are homogeneous in α . Therefore the strict inequalities above hold if and only if the nonstrict inequalities

$$q^{(i)T} \hat{F}(x^{(i)}) \geq 1 + \cos(\gamma) \|\hat{F}(x^{(i)})\|_2, \quad i = 1, \dots, N,$$

hold. (We can achieve this by scaling α ; this is the standard trick to handling strict inequalities when the problem is homogeneous.) Now we use bisection on γ to approximate J^* with any desired degree of accuracy. Note that any feasible α cannot be zero, since the righthand side of the inequality is positive. Our code (shown below) just checks whether a given number of degrees error is attainable; we find that $15^\circ < J^* \leq 16^\circ$. The code produces the plot shown below.

```
%solution of vector field direction fitting problem
vfield_fit_data;

%check manually for feasibility;
%we find 16 degrees works; 15 does not
deg = 16; % degrees of angle fit
```

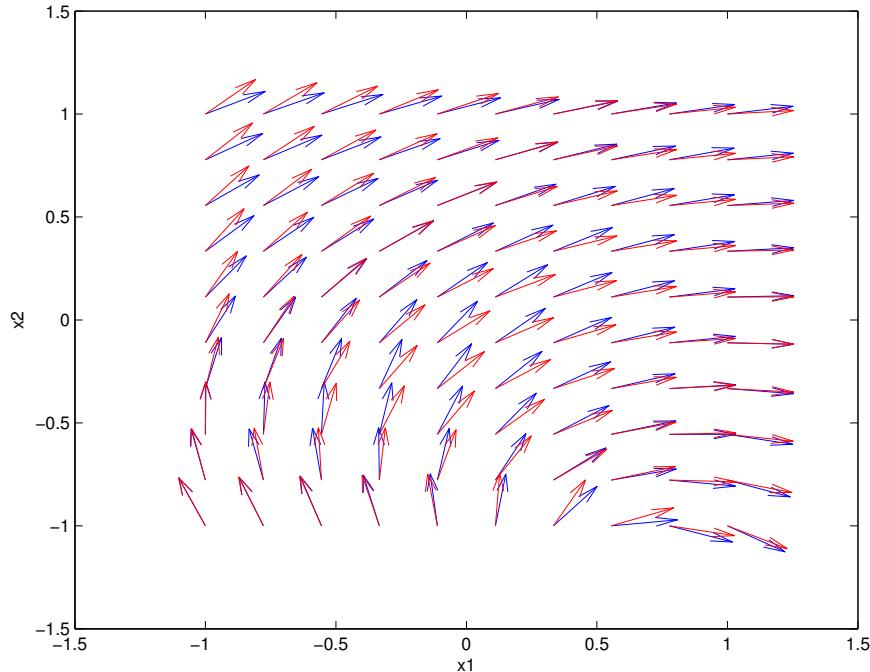
```

gamma=deg*(pi/180);

cvx_begin
variables A(2,2) b(2);
Fhat=A*X+b*ones(1,N);
sum(Q.*Fhat)>=1+cos(gamma)*norms(Fhat);
cvx_end

%Normalize F_hat
Fhatn=Fhat*diag(1./norms(Fhat));
%Let's plot the directions Q
figure
quiver(x1,x2,Q(1,:),Q(2,:))
hold on
quiver(x1,x2,Fhatn(1,:),Fhatn(2,:),'r')
xlabel('x1')
ylabel('x2')
print -depsc vfield_fit

```



- 7.13 Robust minimum volume covering ellipsoid.** Suppose z is a point in \mathbf{R}^n and \mathcal{E} is an ellipsoid in \mathbf{R}^n with center c . The *Mahalanobis distance* of the point to the ellipsoid center is defined as

$$M(z, \mathcal{E}) = \inf\{t \geq 0 \mid z \in c + t(\mathcal{E} - c)\},$$

which is the factor by which we need to scale the ellipsoid about its center so that z is on its boundary. We have $z \in \mathcal{E}$ if and only if $M(z, \mathcal{E}) \leq 1$. We can use $(M(z, \mathcal{E}) - 1)_+$ as a measure of the Mahalanobis distance of the point z to the ellipsoid \mathcal{E} .

Now we can describe the problem. We are given m points $x_1, \dots, x_m \in \mathbf{R}^n$. The goal is to find the optimal trade-off between the volume of the ellipsoid \mathcal{E} and the total Mahalanobis distance of the points to the ellipsoid, *i.e.*,

$$\sum_{i=1}^m (M(z, \mathcal{E}) - 1)_+.$$

Note that this can be considered a robust version of finding the smallest volume ellipsoid that covers a set of points, since here we allow one or more points to be outside the ellipsoid.

- (a) Explain how to solve this problem. You must say clearly what your variables are, what problem you solve, and why the problem is convex.
- (b) Carry out your method on the data given in `rob_min_vol_ellips_data.m`. Plot the optimal trade-off curve of ellipsoid volume versus total Mahalanobis distance. For some selected points on the trade-off curve, plot the ellipsoid and the points (which are in \mathbf{R}^2). We are only interested in the region of the curve where the ellipsoid volume is within a factor of ten (say) of the minimum volume ellipsoid that covers all the points.

Important. Depending on how you formulate the problem, you might encounter problems that are unbounded below, or where CVX encounters numerical difficulty. Just avoid these by appropriate choice of parameter.

Very important. If you use Matlab version 7.0 (which is filled with bugs) you might find that functions involving determinants don't work in CVX. If you use this version of Matlab, then you must download the file `blkdiag.m` on the course website and put it in your Matlab path before the default version (which has a bug).

Solution. We parametrize the ellipsoid \mathcal{E} as

$$\mathcal{E} = \{x \mid \|Ax - b\|_2 \leq 1\},$$

where $b \in \mathbf{R}^n$ and (without loss of generality) $A \in \mathbf{S}_{++}^n$. The volume of \mathcal{E} is proportional to $\det A^{-1}$. The Mahalanobis distance to the ellipsoid center is then $M(z, \mathcal{E}) = \|Az - b\|_2$, which is a convex function of A and b . It follows by the composition rules that

$$\sum_{i=1}^m (\|Ax_i - b\|_2 - 1)_+$$

is a convex function of A and b .

To find the optimal trade-off we simply minimize

$$\log \det A^{-1} + \lambda \sum_{i=1}^m (\|Ax_i - b\|_2 - 1)_+$$

over $A \in \mathbf{S}_{++}^n$ and $b \in \mathbf{R}^n$. Here λ is a positive weight that we use to trace out the optimal trade-off curve.

We can replace $\log \det A^{-1}$ with any other convex function that increases monotonically with $\det A^{-1}$. For example, we could minimize the convex function

$$-(\det A)^{1/n} + \lambda \sum_{i=1}^m (\|Ax_i - b\|_2 - 1)_+.$$

We get the same optimal trade-off curve (but with a different parametrization).

```

% robust min volume ellipsoid covering problem
rob_min_vol_ellips_data;
K = 10; % number of points on trade-off curve
MM = zeros(K,1);
vol = zeros(K,1);
lambda = logspace(-2,-1,K);
clf;

plot(x(1,:),x(2,:),'r') % plot the data points
hold on
t=linspace(0,2*pi,100);
u=[cos(t);sin(t)]; %unit circle

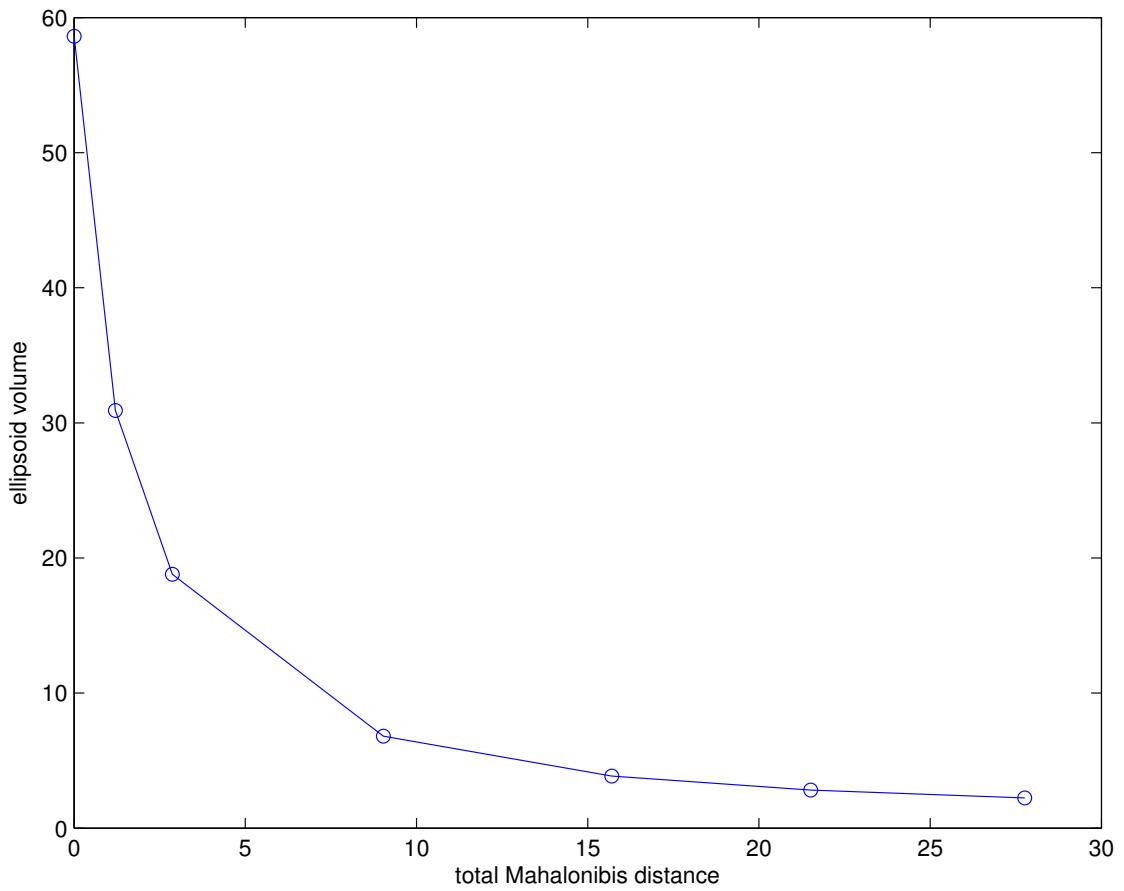
for i=1:K
cvx_begin
variable A(n,n) symmetric
variable b(n)
M = sum(pos(norms(A*x-b*ones(1,m))-1));
minimize (-det_rootn(A) + lambda(i)*M)
%minimize (-log_det(A) + lambda(i)*M)
cvx_end
vol(i) = 1/det(A);
MM(i)=M;
E=inv(A)*(u+b*ones(1,length(t)));
plot(E(1,:),E(2,:),'-') %covering ellipse
hold on
end

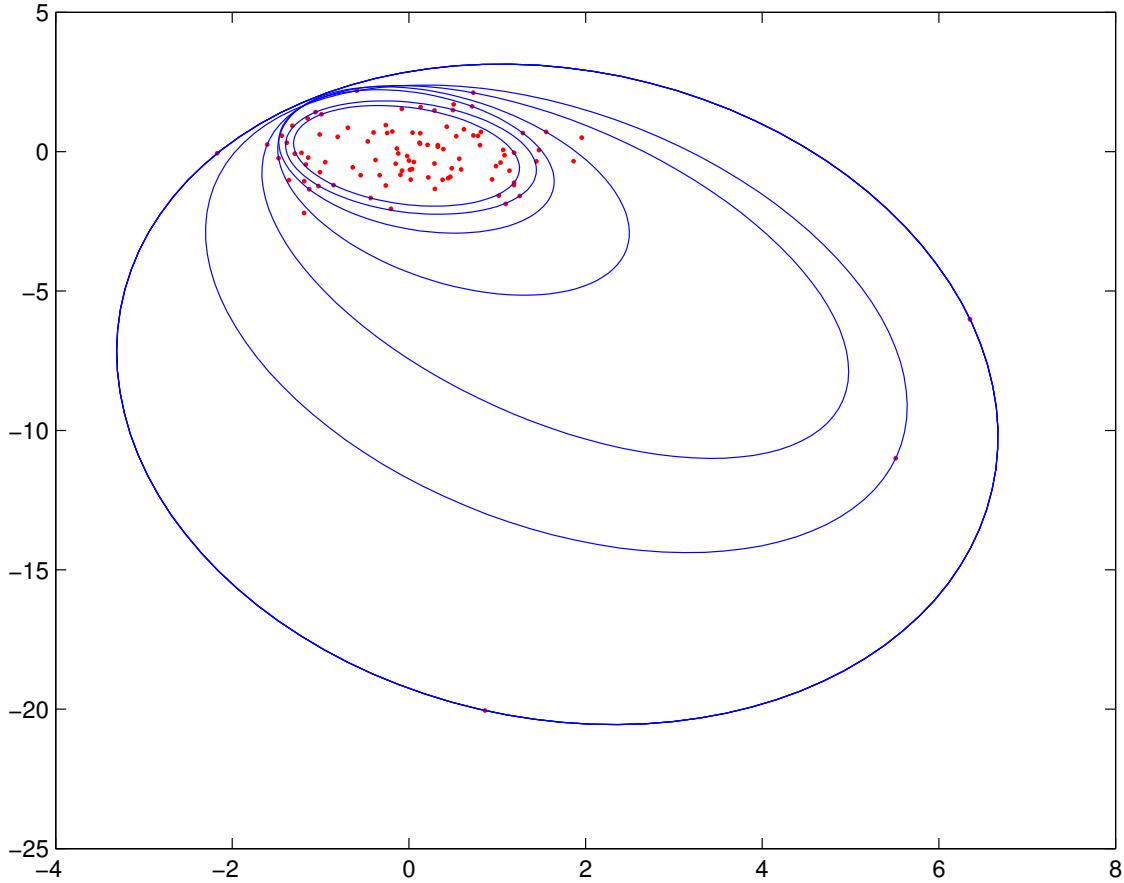
print -depsc rob_min_vol_ellips_pts

figure
plot(MM,vol,'o-')
xlabel('total Mahalonibis distance')
ylabel('ellipsoid volume')

print -depsc rob_min_vol_ellips_tradeoff

```





7.14 Isoperimetric problem. We consider the problem of choosing a curve in a two-dimensional plane that encloses as much area as possible between itself and the x -axis, subject to constraints. For simplicity we will consider only curves of the form

$$\mathcal{C} = \{(x, y) \mid y = f(x)\},$$

where $f : [0, a] \rightarrow \mathbf{R}$. This assumes that for each x -value, there can only be a single y -value, which need not be the case for general curves. We require that at the end points (which are given), the curve returns to the x -axis, so $f(0) = 0$, and $f(a) = 0$. In addition, the length of the curve cannot exceed a budget L , so we must have

$$\int_0^a \sqrt{1 + f'(x)^2} dx \leq L.$$

The objective is the area enclosed, which is given by

$$\int_0^a f(x) dx.$$

To pose this as a finite dimensional optimization problem, we discretize over the x -values. Specifically, we take $x_i = h(i - 1)$, $i = 1, \dots, N + 1$, where $h = a/N$ is the discretization step size, and

we let $y_i = f(x_i)$. Thus our objective becomes

$$h \sum_{i=1}^N y_i,$$

and our constraints can be written as

$$h \sum_{i=1}^N \sqrt{1 + ((y_{i+1} - y_i)/h)^2} \leq L, \quad y_1 = 0, \quad y_{N+1} = 0.$$

In addition to these constraints, we will also require that our curve passes through a set of pre-specified points. Let $\mathcal{F} \subseteq \{1, \dots, N+1\}$ be an index set. For $j \in \mathcal{F}$, we require $y_j = y_j^{\text{fixed}}$, where $y^{\text{fixed}} \in \mathbf{R}^{N+1}$ (the entries of y^{fixed} whose indices are not in \mathcal{F} can be ignored). Finally, we add a constraint on maximum curvature,

$$-C \leq (y_{i+2} - 2y_{i+1} + y_i)/h^2 \leq C, \quad i = 1, \dots, N-1.$$

Explain how to find the curve, *i.e.*, y_1, \dots, y_{N+1} , that maximizes the area enclosed subject to these constraints, using convex optimization. Carry out your method on the problem instance with data given in `iso_perim_data.m`. Report the optimal area enclosed, and use the commented out code in the data file to plot your curve.

Remark (for your amusement only). The isoperimetric problem is an ancient problem in mathematics with a history dating all the way back to the tragedy of queen Dido and the founding of Carthage. The story (which is mainly the account of the poet Virgil in his epic volume *Aeneid*), goes that Dido was a princess forced to flee her home after her brother murdered her husband. She travels across the mediterranean and arrives on the shores of what is today modern Tunisia. The natives weren't very happy about the newcomers, but Dido was able to negotiate with the local King: in return for her fortune, the King promised to cede her as much land as she could mark out with the skin of a bull.

The king thought he was getting a good deal, but Dido outmatched him in mathematical skill. She broke down the skin into thin pieces of leather and sewed them into a long piece of string. Then, taking the seashore as an edge, they laid the string in a semicircle, carving out a piece of land larger than anyone imagined; and on this land, the ancient city of Carthage was born. When the king saw what she had done, he was so impressed by Dido's talent that he asked her to marry him. Dido refused, so the king built a university in the hope that he could find another woman with similar talent.

Solution. Putting everything together, our problem is

$$\begin{aligned} & \text{minimize} && h \sum_{i=1}^N y_i \\ & \text{subject to} && h \sum_{i=1}^N \sqrt{1 + ((y_{i+1} - y_i)/h)^2} \leq L \\ & && -C \leq (y_{i+2} - 2y_{i+1} + y_i)/h^2 \leq C, \quad i = 1, \dots, N-1 \\ & && y_1 = 0, \quad y_{N+1} = 0, \quad y_j = y_j^{\text{fixed}}, \quad j \in \mathcal{F}, \end{aligned}$$

with variables y_1, \dots, y_{N+1} . This is convex since the objective is linear and we can write

$$\sqrt{1 + ((y_{i+1} - y_i)/h)^2} = \|(1, (y_{i+1} - y_i)/h)\|_2,$$

which is clearly a convex function of y_1, \dots, y_{N+1} .

The following matlab code solves this problem.

```

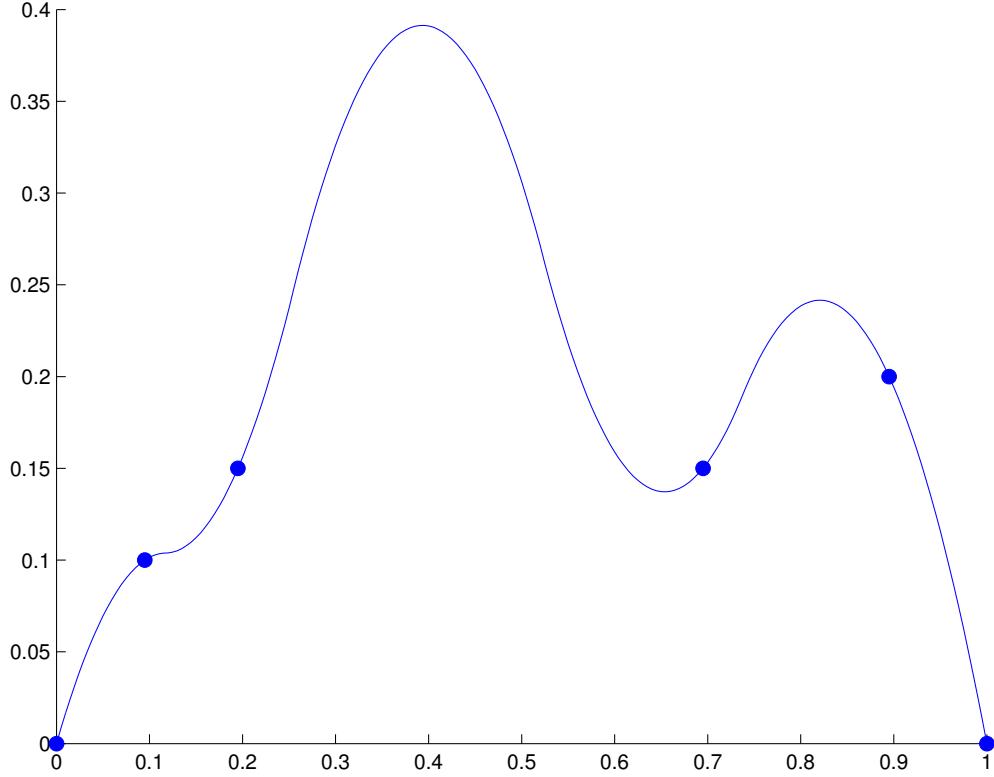
iso_perim_data;

cvx_begin
    variables y(N+1)
    for i = 1:N-1
        abs((y(i+2)-2*y(i+1)+y(i))/h^2) <= C;
    end
    len = 0;
    for i = 1:N
        len = len+h*norm([1,(y(i+1)-y(i))/h]);
    end
    len <= L;
    y(1) == 0; y(N+1) == 0;
    y(F) == yfixed(F);
    maximize(h*sum(y))
cvx_end
max_area = cvx_optval;

% plot the curve
x = [0:h:a];
figure; hold on;
plot(0,0,'bo','markerfacecolor','b','markersize',7);
plot(a,0,'bo','markerfacecolor','b','markersize',7);
for i = 1:length(F);
    plot(x(F(i)),yfixed(F(i)), 'bo','markerfacecolor','b','markersize',7);
end
axis([0,1,0,0.4]);
plot(x,y);
print('-depsc','iso_perim_curve.eps');

```

Running the code in matlab gives an optimal area of 0.206. The optimal curve is shown in the following figure (the blue dots show the fixed points).



7.15 Dual of maximum volume ellipsoid problem. Consider the problem of computing the maximum volume ellipsoid inscribed in a nonempty bounded polyhedron

$$C = \{x \mid a_i^T x \leq b_i, i = 1, \dots, m\}.$$

Parametrizing the ellipsoid as $\mathcal{E} = \{Bu + d \mid \|u\|_2 \leq 1\}$, with $B \in \mathbf{S}_{++}^n$ and $d \in \mathbf{R}^n$, the optimal ellipsoid can be found by solving the convex optimization problem

$$\begin{aligned} & \text{minimize} && -\log \det B \\ & \text{subject to} && \|Ba_i\|_2 + a_i^T d \leq b_i, \quad i = 1, \dots, m \end{aligned}$$

with variables $B \in \mathbf{S}^n$, $d \in \mathbf{R}^n$. Derive the Lagrange dual of the equivalent problem

$$\begin{aligned} & \text{minimize} && -\log \det B \\ & \text{subject to} && \|y_i\|_2 + a_i^T d \leq b_i, \quad i = 1, \dots, m \\ & && Ba_i = y_i, \quad i = 1, \dots, m \end{aligned}$$

with variables $B \in \mathbf{S}^n$, $d \in \mathbf{R}^n$, $y_i \in \mathbf{R}^n$, $i = 1, \dots, m$.

Solution. The Lagrangian is

$$\begin{aligned} & L(B, d, Y, \lambda, Z) \\ &= -\log \det B + \sum_{i=1}^m \lambda_i (\|y_i\|_2 + a_i^T d - b_i) + \sum_{i=1}^m (z_i^T Ba_i - z_i^T y_i) \\ &= -\log \det B + \sum_{i=1}^m z_i^T Ba_i + \sum_{i=1}^m (\lambda_i \|y_i\|_2 - z_i^T y_i) + \sum_{i=1}^m \lambda_i a_i^T d - b^T \lambda. \end{aligned}$$

Setting the gradient of the terms in B to zero gives $B^{-1} = (1/2) \sum_{i=1}^m (a_i z_i^T + z_i a_i^T)$, and

$$\inf_B \left(-\log \det B + \sum_i z_i^T B a_i \right) = \log \det \left(\sum_{i=1}^m (a_i z_i^T + z_i a_i^T) \right) + n - n \log 2.$$

The infimum of the terms in L that involve y_i is zero if $\|z_i\|_2 \leq \lambda_i$ and $-\infty$ otherwise. The infimum of the term that involves d is zero if $\sum_i \lambda_i a_i = 0$ and $-\infty$ otherwise. Combining everything gives the dual problem

$$\begin{aligned} & \text{maximize} && \log \det \left(\sum_{i=1}^m (a_i z_i^T + z_i a_i^T) \right) - b^T \lambda + n - n \log 2 \\ & \text{subject to} && \|z_i\|_2 \leq \lambda_i, \quad i = 1, \dots, m \\ & && \sum_{i=1}^m \lambda_i a_i = 0 \\ & && \lambda \succeq 0. \end{aligned}$$

7.16 Fitting a sphere to data. Consider the problem of fitting a sphere $\{x \in \mathbf{R}^n \mid \|x - x_c\|_2 = r\}$ to m points $u_1, \dots, u_m \in \mathbf{R}^n$, by minimizing the error function

$$\sum_{i=1}^m (\|u_i - x_c\|_2^2 - r^2)^2$$

over the variables $x_c \in \mathbf{R}^n$, $r \in \mathbf{R}$.

- (a) Explain how to solve this problem using convex or quasiconvex optimization. The simpler your formulation, the better. (For example: a convex formulation is simpler than a quasiconvex formulation; an LP is simpler than an SOCP, which is simpler than an SDP.) Be sure to explain what your variables are, and how your formulation minimizes the error function above.
- (b) Use your method to solve the problem instance with data given in the file `sphere_fit_data.m`, with $n = 2$. Plot the fitted circle and the data points.

Solution.

- (a) The problem can be formulated as a simple *least-squares problem*, the simplest nontrivial convex optimization problem!

We will formulate the problem as

$$\text{minimize } \|Ax - b\|_2^2.$$

Choose as variables $x = (x_c, t)$ with t defined as $t = r^2 - \|x_c\|_2^2$. Use the optimality conditions $A^T(Ax - b) = 0$ of the least-squares problem to show that $t + \|x_c\|_2^2 \geq 0$ at the optimum. This ensures that r can be computed from the optimal x_c , t using the formula $r = (t + \|x_c\|_2^2)^{1/2}$.

Take

$$A = \begin{bmatrix} 2u_1^T & 1 \\ 2u_2^T & 1 \\ \vdots & \vdots \\ 2u_m^T & 1 \end{bmatrix}, \quad x = \begin{bmatrix} x_c \\ t \end{bmatrix}, \quad b = \begin{bmatrix} \|u_1\|_2^2 \\ \|u_2\|_2^2 \\ \vdots \\ \|u_m\|_2^2 \end{bmatrix}.$$

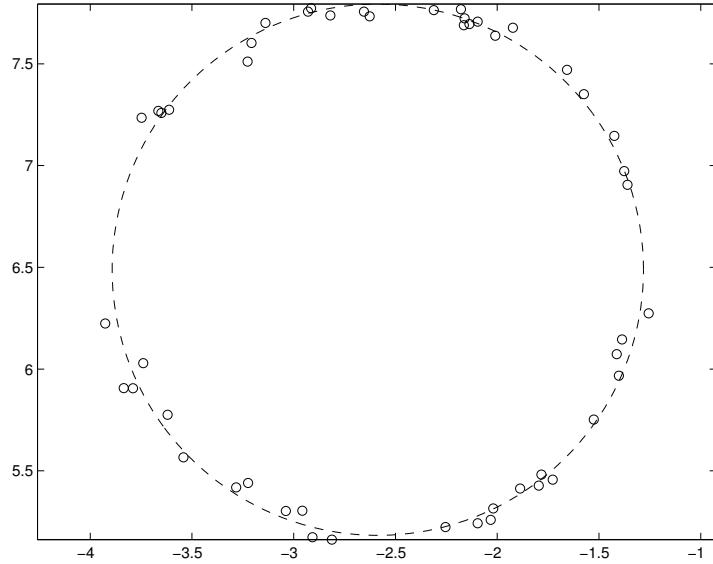
The last equation in $A^T(Ax - b) = 0$ gives

$$\sum_{i=1}^m \left(2u_i^T x_c + t - \|u_i\|_2^2 \right) = 0,$$

from which we obtain

$$t + \|x_c\|_2^2 = \frac{1}{m} \sum_{i=1}^m \|u_i - x_c\|_2^2.$$

(b) $x_c = (-2.5869, 6.4883)$, $R = 1.3052$.



7.17 The *polar* of a set $C \subseteq \mathbf{R}^n$ is defined as

$$C^\circ = \{x \mid u^T x \leq 1 \ \forall u \in C\}.$$

- (a) Show that C° is convex, regardless of the properties of C .
- (b) Let C_1 and C_2 be two nonempty polyhedra defined by sets of linear inequalities:

$$C_1 = \{u \in \mathbf{R}^n \mid A_1 u \preceq b_1\}, \quad C_2 = \{v \in \mathbf{R}^n \mid A_2 v \preceq b_2\}$$

with $A_1 \in \mathbf{R}^{m_1 \times n}$, $A_2 \in \mathbf{R}^{m_2 \times n}$, $b_1 \in \mathbf{R}^{m_1}$, $b_2 \in \mathbf{R}^{m_2}$. Formulate the problem of finding the Euclidean distance between C_1° and C_2° ,

$$\begin{aligned} & \text{minimize} && \|x_1 - x_2\|_2^2 \\ & \text{subject to} && x_1 \in C_1^\circ \\ & && x_2 \in C_2^\circ, \end{aligned}$$

as a QP. Your formulation should be efficient, *i.e.*, the dimensions of the QP (number of variables and constraints) should be linear in m_1 , m_2 , n . (In particular, formulations that require enumerating the extreme points of C_1 and C_2 are to be avoided.)

Solution.

- (a) C° is the intersection of halfspaces.
- (b) $x_i \in C_i^\circ$ if and only if the optimal value of the LP

$$\begin{aligned} & \text{maximize} && x_i^T u_i \\ & \text{subject to} && A_i u_i \preceq b_i \end{aligned}$$

is less than or equal to one. The dual of this problem is

$$\begin{aligned} & \text{minimize} && b_i^T z_i \\ & \text{subject to} && A_i^T z_i + x_i = 0 \\ & && z_i \succeq 0 \end{aligned}$$

with variable z_i . The optimal value is less than or equal to one if and only if there exists a feasible z_i with $b_i^T z_i \leq 1$. The problem in the statement is therefore equivalent to

$$\begin{aligned} & \text{minimize} && \|x_1 - x_2\|_2^2 \\ & \text{subject to} && b_1^T z_1 \leq 1, \quad b_2^T z_2 \leq 1 \\ & && A_1^T z_1 + x_1 = 0, \quad A_2^T z_2 + x_2 = 0 \\ & && z_1 \succeq 0, \quad z_2 \succeq 0. \end{aligned}$$

The variables are $x_1, x_2 \in \mathbf{R}^n$, $z_1 \in \mathbf{R}^{m_1}$, $z_2 \in \mathbf{R}^{m_2}$.

7.18 Polyhedral cone questions.

You are given matrices $A \in \mathbf{R}^{n \times k}$ and $B \in \mathbf{R}^{n \times p}$.

Explain how to solve the following two problems using convex optimization. Your solution can involve solving multiple convex problems, as long as the number of such problems is no more than linear in the dimensions n , k , p .

- (a) How would you determine whether $A\mathbf{R}_+^k \subseteq B\mathbf{R}_+^p$? This means that every nonnegative linear combination of the columns of A can be expressed as a nonnegative linear combination of the columns of B .
- (b) How would you determine whether $A\mathbf{R}_+^k = \mathbf{R}^n$? This means that every vector in \mathbf{R}^n can be expressed as a nonnegative linear combination of the columns of A .

Solution.

- (a) If $A\mathbf{R}_+^k \subseteq B\mathbf{R}_+^p$ holds, then we must have $a_i \in B\mathbf{R}_+^p$, for $i = 1, \dots, k$, where a_i is the i th column of A . Conversely, if $a_i \in B\mathbf{R}_+^p$ for $i = 1, \dots, k$, then $A\mathbf{R}_+^k \subseteq B\mathbf{R}_+^p$. To see this, suppose that $a_i = By_i$, with $y_i \succeq 0$, $i = 1, \dots, k$. For any $\alpha \succeq 0$, we have

$$A\alpha = \sum_{i=1}^k \alpha_i a_i = \sum_{i=1}^k \alpha_i By_i = B(\alpha_1 y_1 + \dots + \alpha_k y_k) \in B\mathbf{R}_+^p,$$

since $\alpha_1 y_1 + \dots + \alpha_k y_k \succeq 0$.

We can determine whether $a_i \in B\mathbf{R}_+^p$ by solving a feasibility LP: Find $y_i \succeq 0$ with $By_i = a_i$. So we can determine whether $A\mathbf{R}_+^k \subseteq B\mathbf{R}_+^p$ by solving k LPs. If any of these is infeasible, we know that $A\mathbf{R}_+^k \not\subseteq B\mathbf{R}_+^p$ (indeed, we have found a point a_i in $A\mathbf{R}_+^k$ that is not in $B\mathbf{R}_+^p$); if they are all feasible, then we know $A\mathbf{R}_+^k \subseteq B\mathbf{R}_+^p$.

We can lump these k LPs into one LP: Find $Y \in \mathbf{R}^{p \times k}$ with $Y_{ij} \geq 0$ and $A = BY$. (But this LP separates into the LPs given above.)

- (b) We can use a very similar argument here. We check whether or not there are $y_i \succeq 0$ and $z_i \succeq 0$ for which $Ay_i = e_i$ and $Az_i = -e_i$, where e_i is the i th standard unit vector. If not, then $A\mathbf{R}_+^k \neq \mathbf{R}^n$. But if we find such vectors, then we have, for any $x \in \mathbf{R}^n$,

$$x = A(Y((x)_+) + Z((x)_-)),$$

where $(x)_+$ is the nonnegative part of x and $(x)_-$ is the negative part. The vector $Y((x)_+) + Z((x)_-)$ on the right is nonnegative.

7.19 Projection on convex hull of union of ellipsoids. Let E_1, \dots, E_m be m ellipsoids in \mathbf{R}^n defined as

$$E_i = \{A_i u + b_i \mid \|u\|_2 \leq 1\}, \quad i = 1, \dots, m,$$

with $A_i \in \mathbf{R}^{n \times n}$ and $b_i \in \mathbf{R}^n$. Consider the problem of projecting a point $a \in \mathbf{R}^n$ on the convex hull of the union of the ellipsoids:

$$\begin{aligned} & \text{minimize} && \|x - a\|_2 \\ & \text{subject to} && x \in \mathbf{conv}(E_1 \cup \dots \cup E_m). \end{aligned}$$

Formulate this as a second order cone program.

Solution. First express the problem as

$$\begin{aligned} & \text{minimize} && \|x - a\|_2 \\ & \text{subject to} && x = \sum_{i=1}^m \theta_i (A_i u_i + b_i) \\ & && \theta_i \geq 0, \quad i = 1, \dots, m \\ & && \sum_{i=1}^m \theta_i = 1 \\ & && \|u_i\|_2 \leq 1, \quad i = 1, \dots, m. \end{aligned}$$

The variables are $x, \theta_1, \dots, \theta_m, u_1, \dots, u_m$. This problem is not convex because of the products $\theta_i u_i$. It becomes convex if we make a change of variables $\theta_i u_i = y_i$:

$$\begin{aligned} & \text{minimize} && \|x - a\|_2 \\ & \text{subject to} && x = \sum_{i=1}^m (A_i y_i + \theta_i b_i) \\ & && \theta_i \geq 0, \quad i = 1, \dots, m \\ & && \sum_{i=1}^m \theta_i = 1 \\ & && \|y_i\|_2 \leq \theta_i, \quad i = 1, \dots, m. \end{aligned}$$

To cast in the standard SOCP form, we also need to make the objective linear by introducing a variable t :

$$\begin{aligned} & \text{minimize} && t \\ & \text{subject to} && \|x - a\|_2 \leq t \\ & && x = \sum_{i=1}^m (A_i y_i + \theta_i b_i) \\ & && \theta_i \geq 0, \quad i = 1, \dots, m \\ & && \sum_{i=1}^m \theta_i = 1 \\ & && \|y_i\|_2 \leq \theta_i, \quad i = 1, \dots, m. \end{aligned}$$

7.20 Bregman divergences. Let $f : \mathbf{R}^n \rightarrow \mathbf{R}$ be strictly convex and differentiable. Then the *Bregman divergence* associated with f is the function $D_f : \mathbf{R}^n \times \mathbf{R}^n \rightarrow \mathbf{R}$ given by

$$D_f(x, y) = f(x) - f(y) - \nabla f(y)^T(x - y).$$

- (a) Show that $D_f(x, y) \geq 0$ for all $x, y \in \text{dom } f$.
- (b) Show that if $f = \|\cdot\|_2^2$, then $D_f(x, y) = \|x - y\|_2^2$.
- (c) Show that if $f(x) = \sum_{i=1}^n x_i \log x_i$ (negative entropy), with $\text{dom } f = \mathbf{R}_+^n$ (with $0 \log 0$ taken to be 0), then

$$D_f(x, y) = \sum_{i=1}^n (x_i \log(x_i/y_i) - x_i + y_i),$$

the *Kullback-Leibler divergence* between x and y .

- (d) *Bregman projection.* The previous parts suggest that Bregman divergences can be viewed as generalized ‘distances’, *i.e.*, functions that measure how similar two vectors are. This suggests solving geometric problems that measure distance between vectors using a Bregman divergence rather than Euclidean distance.

Explain whether

$$\begin{aligned} & \text{minimize} && D_f(x, y) \\ & \text{subject to} && x \in \mathcal{C}, \end{aligned}$$

with variable $x \in \mathbf{R}^n$, is a convex optimization problem (assuming \mathcal{C} is convex).

- (e) *Duality.* Show that $D_g(y^*, x^*) = D_f(x, y)$, where $g = f^*$ and $z^* = \nabla f(z)$. You can assume that $\nabla f^* = (\nabla f)^{-1}$ and that f is closed.

Solution.

- (a) Since f is convex and differentiable,

$$f(x) \geq f(y) + \nabla f(y)^T(x - y)$$

for all $x, y \in \text{dom } f$. Rearranging,

$$f(x) - f(y) - \nabla f(y)^T(x - y) \geq 0.$$

The lefthand side is $D_f(x, y)$.

- (b) If $f = \|\cdot\|_2^2$, then

$$\begin{aligned} D_f(x, y) &= \|x\|_2^2 - \|y\|_2^2 - 2y^T(x - y) \\ &= \|x\|_2^2 - 2y^T x + \|y\|_2^2 \\ &= \|x - y\|_2^2. \end{aligned}$$

- (c) If $f(x) = \sum_{i=1}^n x_i \log x_i$, then

$$\begin{aligned} D_f(x, y) &= \sum_{i=1}^n x_i \log x_i - \sum_{i=1}^n y_i \log y_i + \sum_{i=1}^n (x_i - y_i)(-\log y_i - 1) \\ &= \sum_{i=1}^n x_i \log x_i - \sum_{i=1}^n x_i \log y_i - \sum_{i=1}^n y_i \log y_i + \sum_{i=1}^n y_i \log y_i - \mathbf{1}^T x + \mathbf{1}^T y \\ &= \sum_{i=1}^n x_i \log(x_i/y_i) - \mathbf{1}^T x + \mathbf{1}^T y. \end{aligned}$$

- (d) Since $D_f(x, y)$ is affine in x for fixed y , Bregman projection (onto a convex set) is a convex optimization problem.

- (e) Recall (from page 95) that

$$f^*(\nabla f(z)) = z^T \nabla f(z) - f(z).$$

It follows that

$$\begin{aligned} D_f(x, y) &= f(x) - f(y) - \nabla f(y)^T (x - y) \\ &= f(x) + y^T \nabla f(y) - f(y) - x^T \nabla f(y) \\ &= f(x) + f^*(y^*) - x^T \nabla f(y). \end{aligned}$$

Applying this equality to D_g , we obtain

$$D_g(y^*, x^*) = f^*(y^*) + f^{**}(x^{**}) - y^{*T} x^{**},$$

where $x^{**} = \nabla f^*(x^*)$. Since $\nabla f^* = (\nabla f)^{-1}$, it follows that $x^{**} = \nabla f^{-1}(x^*) = x$. Similarly, because f is closed and convex, we have that $f^{**} = f$. Substituting above, we get

$$D_g(y^*, x^*) = f^*(y^*) + f(x) - x^T y^*,$$

which is precisely $D_f(x, y)$.

Alternatively, we could start from the definition of D_g :

$$D_g(y^*, x^*) = f^*(y^*) - f^*(x^*) - \nabla f^*(x^*)^T (y^* - x^*).$$

The result follows by plugging in $\nabla f(y)$ and $\nabla f(x)$ for y^* and x^* , observing that $\nabla f^*(x^*) = x$, using the expression above for $f^*(\nabla f(z))$, and rearranging.

- 7.21 Ellipsoidal peeling.** In this problem, you will implement an outlier identification technique using Löwner-John ellipsoids. Given a set of points $\mathcal{D} = \{x_1, \dots, x_N\}$ in \mathbf{R}^n , the goal is to identify a set $\mathcal{O} \subseteq \mathcal{D}$ that are anomalous in some sense. Roughly speaking, we think of an outlier as a point that is far away from most of the points, so we would like the points in $\mathcal{D} \setminus \mathcal{O}$ to be relatively close together, and to be relatively far apart from the points in \mathcal{O} .

We describe a heuristic technique for identifying \mathcal{O} . We start with $\mathcal{O} = \emptyset$ and find the minimum volume (Löwner-John) ellipsoid \mathcal{E} containing all $x_i \notin \mathcal{O}$ (which is all x_i in the first step). Each iteration, we flag (*i.e.*, add to \mathcal{O}) the point that corresponds to the largest dual variable for the constraint $x_i \in \mathcal{E}$; this point will be one of the points on the boundary of \mathcal{E} , and intuitively, it will be the one for whom the constraint is ‘most’ binding. We then plot **vol** \mathcal{E} (on a log scale) versus **card** \mathcal{O} and hope that we see a sharp drop in the curve. We use the value of \mathcal{O} after the drop.

The hope is that after removing a relatively small number of points, the volume of the minimum volume ellipsoid containing the remaining points will be much smaller than the minimum volume ellipsoid for \mathcal{D} , which means the removed points are far away from the others.

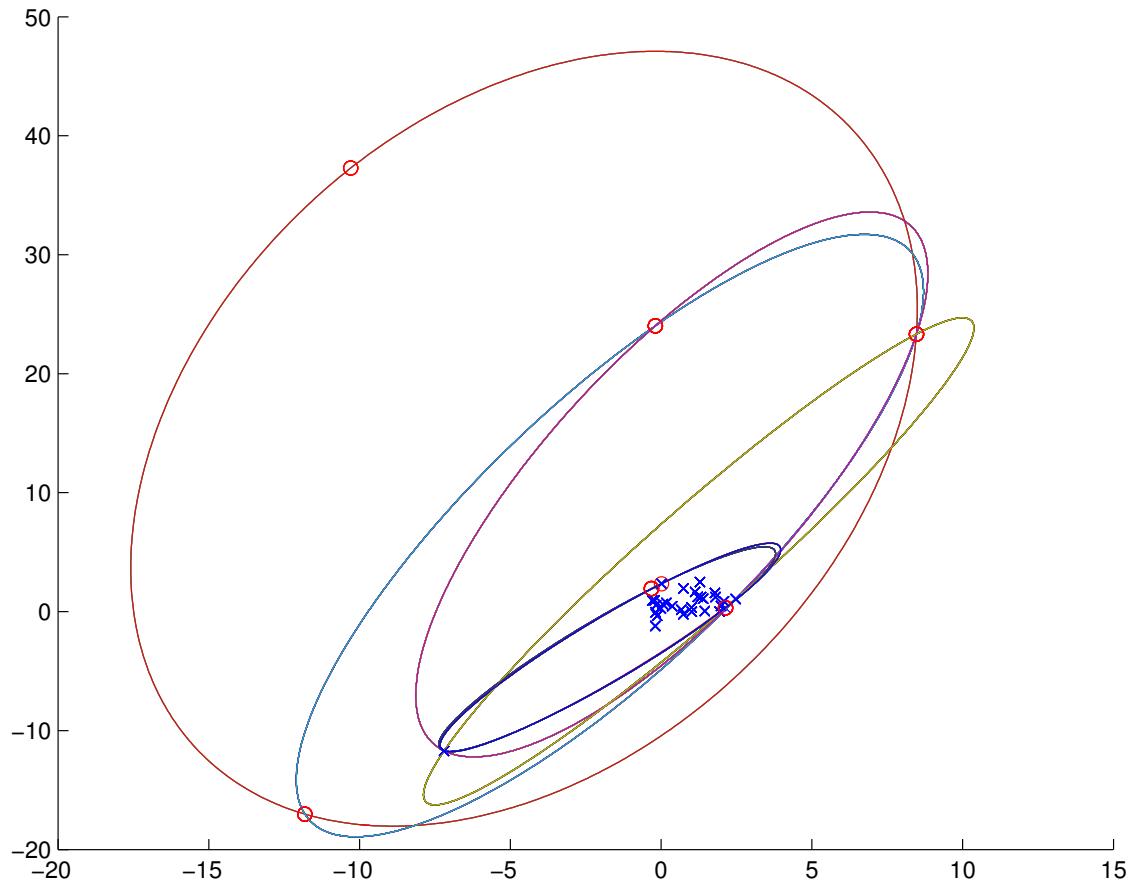
For example, suppose we have 100 points that lie in the unit ball and 3 points with (Euclidean) norm 1000. Intuitively, it is clear that it is reasonable to consider the three large points outliers. The minimum volume ellipsoid of all 103 points will have very large volume. The three points will be the first ones removed, and as soon as they are, the volume of the ellipsoid ellipsoid will drop dramatically and be on the order of the volume of the unit ball.

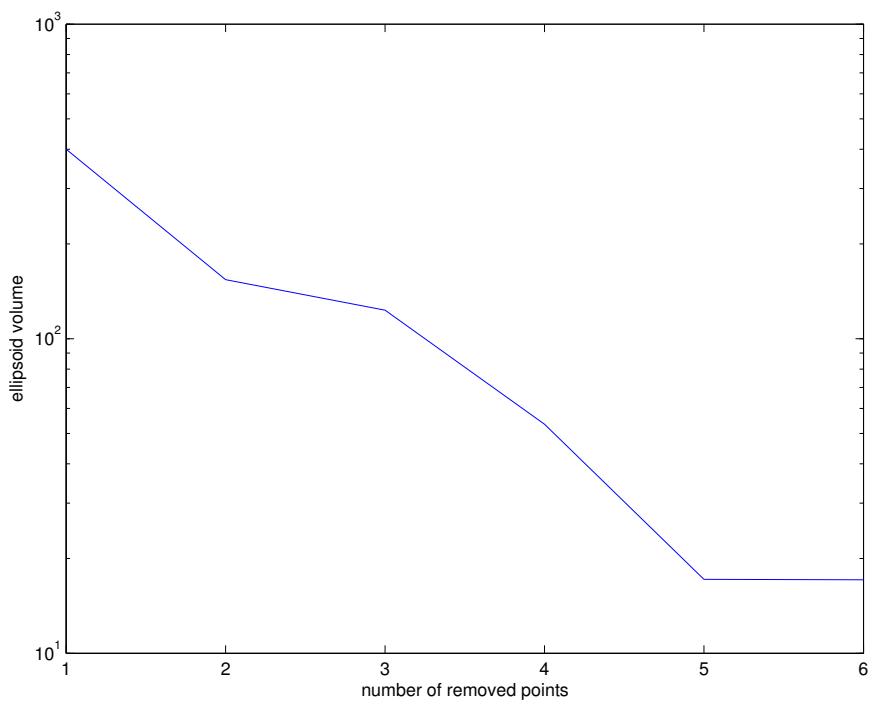
Run 6 iterations of the algorithm on the data given in `ellip_anomaly_data.m`. Plot $\text{vol } \mathcal{E}$ (on a log scale) versus `card` \mathcal{O} . In addition, on a single plot, plot all the ellipses found with the function `ellipse_draw(A,b)` along with the outliers (in red) and the remaining points (in blue).

Of course, we have chosen an example in \mathbf{R}^2 so the ellipses can be plotted, but one can detect outliers in \mathbf{R}^2 simply by inspection. In dimension much higher than 3, however, detecting outliers by plotting will become substantially more difficult, while the same algorithm can be used.

Note. In CVX, you should use `det_rootn` (which is SDP-representable and handled exactly) instead of `log_det` (which is handled using an inefficient iterative procedure).

Solution. The code for removing the outliers and generating the plots is given below. We remark that the algorithm described here is typically called *ellipsoidal peeling* in the literature.





```

%% ellipsoidal peeling

ellip_anomaly_data;

volumes = [];
removed = [];

figure(1); hold all;

for i = 1:6
    % fit ellipsoid
    cvx_begin
        variable A(2,2) symmetric
        variable b(2)
        dual variable v
        maximize (det_rootn(A))
        subject to
            v : norms(A*X + b*ones(1,size(X,2))) <= 1
    cvx_end

    ellipse_draw(A,b);

    % detect outliers
    [vm idx] = max(v);
    removed = [ removed X(:,idx) ];
    X(:,idx) = [];
    volumes = [ volumes 1/det(A) ];
end

plot(X(1,:), X(2,:), 'bx'); % normal points
plot(removed(1,:), removed(2,:), 'ro'); % outliers
print('-depsc','../figures/ellipsoids.eps');

figure(2);
semilogy(volumes)
xlabel('number of removed points');
ylabel('ellipsoid volume');
set(gca,'XTick', 1:length(volumes));
print('-depsc','../figures/volume_iters.eps');

```

8 Unconstrained and equality constrained minimization

8.1 Gradient descent and nondifferentiable functions.

- (a) Let $\gamma > 1$. Show that the function

$$f(x_1, x_2) = \begin{cases} \sqrt{x_1^2 + \gamma x_2^2} & |x_2| \leq x_1 \\ \frac{x_1 + \gamma|x_2|}{\sqrt{1+\gamma}} & \text{otherwise} \end{cases}$$

is convex. You can do this, for example, by verifying that

$$f(x_1, x_2) = \sup \left\{ x_1 y_1 + \sqrt{\gamma} x_2 y_2 \mid y_1^2 + y_2^2 \leq 1, y_1 \geq 1/\sqrt{1+\gamma} \right\}.$$

Note that f is unbounded below. (Take $x_2 = 0$ and let x_1 go to $-\infty$.)

- (b) Consider the gradient descent algorithm applied to f , with starting point $x^{(0)} = (\gamma, 1)$ and an exact line search. Show that the iterates are

$$x_1^{(k)} = \gamma \left(\frac{\gamma - 1}{\gamma + 1} \right)^k, \quad x_2^{(k)} = \left(-\frac{\gamma - 1}{\gamma + 1} \right)^k.$$

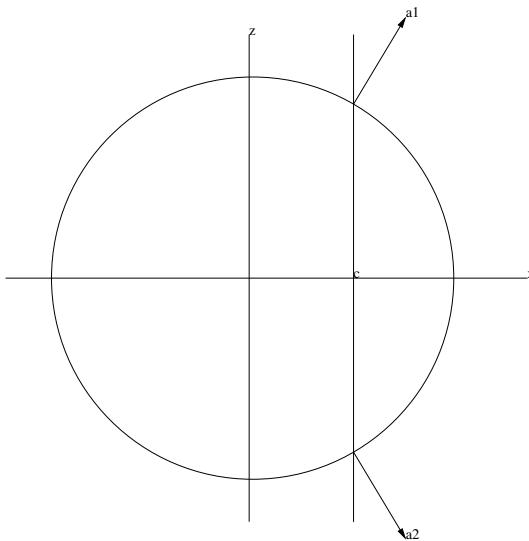
Therefore $x^{(k)}$ converges to $(0, 0)$. However, this is not the optimum, since f is unbounded below.

Solution.

- (a) We follow the hint and examine the optimization problem

$$\begin{array}{ll} \text{minimize} & x_1 y_1 + \sqrt{\gamma} x_2 y_2 \\ \text{subject to} & y_1^2 + y_2^2 \leq 1 \\ & y_1 \geq 1/\sqrt{1+\gamma} \end{array}$$

with variables y_1, y_2 . The feasible set is the part of the unit disk to the right of the vertical line through $y_1 = (1 + \gamma)^{-1/2}$.



We maximize the inner product of y with the coefficient vector $(x_1, \sqrt{\gamma}x_2)$. There are three cases to distinguish, depending on the orientation of the coefficient vector.

- If $x_1 > 0$ and $|x_2| \leq x_1$, the coefficient vector lies in the cone between the vectors $(1, -\sqrt{\gamma})$ and $(1, \sqrt{\gamma})$, and the optimum is

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \frac{1}{x_1^2 + \gamma x_2^2} \begin{bmatrix} x_1 \\ \sqrt{\gamma}x_2 \end{bmatrix}.$$

The optimal value is $(x_1^2 + \gamma x_2^2)^{1/2}$.

- If $x_2 \geq 0$, and $x_1 < x_2$, then the point

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \frac{1}{\sqrt{1+\gamma}} \begin{bmatrix} 1 \\ \sqrt{\gamma} \end{bmatrix}$$

is optimal, and the optimal value is $(x_1 + \gamma x_2)/(1 + \gamma)^{1/2}$.

- If $x_2 \leq 0$, and $x_1 < -x_2$, then the point

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \frac{1}{\sqrt{1+\gamma}} \begin{bmatrix} 1 \\ -\sqrt{\gamma} \end{bmatrix}$$

is optimal, and the optimal value is $(x_1 - \gamma x_2)/(1 + \gamma)^{1/2}$.

- (b) We first note that the iterates given in the problem satisfy $|x_2^{(k)}| < x_1^{(k)}$, so they are in the interior of the region where $f(x_1, x_2) = (x_1^2 + \gamma x_2^2)^{1/2}$. In this region the function is differentiable with gradient

$$\nabla f(x) = \frac{1}{\sqrt{x_1^2 + \gamma x_2^2}} \begin{bmatrix} x_1 \\ \gamma x_2 \end{bmatrix}.$$

Since we use an exact line search, only the direction of $\nabla f(x)$ matters.

We now verify the expressions

$$x_1^{(k)} = \gamma \left(\frac{\gamma - 1}{\gamma + 1} \right)^k, \quad x_2^{(k)} = \left(-\frac{\gamma - 1}{\gamma + 1} \right)^k.$$

in the assignment. For $k = 0$, we get the starting point $x^{(0)} = (\gamma, 1)$. The gradient at $x^{(k)}$ is proportional to $(x_1^{(k)}, \gamma x_2^{(k)})$, and therefore the exact line search minimizes f along the line

$$\begin{bmatrix} (1-t)x_1^{(k)} \\ (1-\gamma t)x_2^{(k)} \end{bmatrix} = \left(\frac{\gamma - 1}{\gamma + 1} \right)^k \begin{bmatrix} (1-t)\gamma \\ (1-\gamma t)(-1)^k \end{bmatrix}.$$

Along this line f is given by

$$f((1-t)x_1^{(k)}, (1-\gamma t)x_2^{(k)}) = \left(\gamma^2(1-t)^2 + \gamma(1-\gamma t)^2 \right)^{1/2} \left(\frac{\gamma - 1}{\gamma + 1} \right)^k.$$

This is minimized by $t = 2/(1 + \gamma)$, so we have

$$\begin{aligned} x^{(k+1)} &= \left(\frac{\gamma - 1}{\gamma + 1} \right)^k \begin{bmatrix} (1-t)\gamma \\ (1-\gamma t)(-1)^k \end{bmatrix} \\ &= \left(\frac{\gamma - 1}{\gamma + 1} \right)^{k+1} \begin{bmatrix} \gamma \\ (-1)^{k+1} \end{bmatrix}. \end{aligned}$$

8.2 A characterization of the Newton decrement. Let $f : \mathbf{R}^n \rightarrow \mathbf{R}$ be convex and twice differentiable, and let A be a $p \times n$ -matrix with rank p . Suppose \hat{x} is feasible for the equality constrained problem

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & Ax = b. \end{array}$$

Recall that the Newton step Δx at \hat{x} can be computed from the linear equations

$$\left[\begin{array}{cc} \nabla^2 f(\hat{x}) & A^T \\ A & 0 \end{array} \right] \left[\begin{array}{c} \Delta x \\ u \end{array} \right] = \left[\begin{array}{c} -\nabla f(\hat{x}) \\ 0 \end{array} \right],$$

and that the Newton decrement $\lambda(\hat{x})$ is defined as

$$\lambda(\hat{x}) = (-\nabla f(\hat{x})^T \Delta x)^{1/2} = (\Delta x^T \nabla^2 f(\hat{x}) \Delta x)^{1/2}.$$

Assume the coefficient matrix in the linear equations above is nonsingular and that $\lambda(\hat{x})$ is positive. Express the solution y of the optimization problem

$$\begin{array}{ll} \text{minimize} & \nabla f(\hat{x})^T y \\ \text{subject to} & Ay = 0 \\ & y^T \nabla^2 f(\hat{x}) y \leq 1 \end{array}$$

in terms of Newton step Δx and the Newton decrement $\lambda(\hat{x})$.

Solution. Strong duality holds ($y = 0$ is strictly feasible). The optimality conditions for y are

- $Ay = 0$ and $y^T \nabla^2 f(\hat{x}) y \leq 1$.
- $\mu \geq 0$.
- $\mu(1 - y^T \nabla^2 f(\hat{x}) y) = 0$.
- $\nabla f(\hat{x}) + A^T \nu + \mu \nabla^2 f(\hat{x}) y = 0$

where μ is the multiplier for the inequality $y^T \nabla^2 f(\hat{x}) y \leq 1$ and ν is the multiplier for $Ay = 0$. From the last condition and $Ay = 0$, we see that $\mu y = \Delta x$. The value of the multiplier μ follows from

$$\mu^2 = \mu^2(y^T \nabla^2 f(\hat{x}) y) = \Delta x^T \nabla^2 f(\hat{x}) \Delta x = \lambda(\hat{x})^2.$$

Hence $\mu = \lambda(\hat{x})$ and

$$y = \frac{1}{\lambda(\hat{x})} \Delta x.$$

8.3 Suggestions for exercises 9.30 in Convex Optimization. We recommend the following to generate a problem instance:

```
n = 100;
m = 200;
randn('state',1);
A=randn(m,n);
```

Of course, you should try out your code with different dimensions, and different data as well.

In all cases, be sure that your line search *first* finds a step length for which the tentative point is in $\text{dom } f$; if you attempt to evaluate f outside its domain, you'll get complex numbers, and you'll never recover.

To find expressions for $\nabla f(x)$ and $\nabla^2 f(x)$, use the chain rule (see Appendix A.4); if you attempt to compute $\partial^2 f(x)/\partial x_i \partial x_j$, you will be sorry.

To compute the Newton step, you can use `vnt=-H\g`.

8.4 *Suggestions for exercise 9.31 in Convex Optimization.* For 9.31a, you should try out $N = 1$, $N = 15$, and $N = 30$. You might as well compute and store the Cholesky factorization of the Hessian, and then back solve to get the search directions, even though you won't really see any speedup in Matlab for such a small problem. After you evaluate the Hessian, you can find the Cholesky factorization as `L=chol(H, 'lower')`. You can then compute a search step as `-L'\(L\g)`, where \mathbf{g} is the gradient at the current point. Matlab will do the right thing, *i.e.*, it will first solve $\mathbf{L}\mathbf{g}$ using forward substitution, and then it will solve $-\mathbf{L}'\backslash(\mathbf{L}\mathbf{g})$ using backward substitution. Each substitution is order n^2 .

To fairly compare the convergence of the three methods (*i.e.*, $N = 1$, $N = 15$, $N = 30$), the horizontal axis should show the approximate total number of flops required, and not the number of iterations. You can compute the approximate number of flops using $n^3/3$ for each factorization, and $2n^2$ for each solve (where each ‘solve’ involves a forward substitution step and a backward substitution step).

8.5 *Efficient numerical method for a regularized least-squares problem.* We consider a regularized least squares problem with smoothing,

$$\text{minimize}_{\mathbf{x}} \quad \sum_{i=1}^k (a_i^T \mathbf{x} - b_i)^2 + \delta \sum_{i=1}^{n-1} (x_i - x_{i+1})^2 + \eta \sum_{i=1}^n x_i^2,$$

where $\mathbf{x} \in \mathbf{R}^n$ is the variable, and $\delta, \eta > 0$ are parameters.

- (a) Express the optimality conditions for this problem as a set of linear equations involving \mathbf{x} . (These are called the normal equations.)
- (b) Now assume that $k \ll n$. Describe an efficient method to solve the normal equations found in part (a). Give an approximate flop count for a general method that does not exploit structure, and also for your efficient method.
- (c) *A numerical instance.* In this part you will try out your efficient method. We'll choose $k = 100$ and $n = 4000$, and $\delta = \eta = 1$. First, randomly generate \mathbf{A} and \mathbf{b} with these dimensions. Form the normal equations as in part (a), and solve them using a generic method. Next, write (short) code implementing your efficient method, and run it on your problem instance. Verify that the solutions found by the two methods are nearly the same, and also that your efficient method is much faster than the generic one.

Note: You'll need to know some things about Matlab to be sure you get the speedup from the efficient method. Your method should involve solving linear equations with tridiagonal coefficient matrix. In this case, both the factorization and the back substitution can be carried out very

efficiently. The Matlab documentation says that banded matrices are recognized and exploited, when solving equations, but we found this wasn't always the case. To be sure Matlab knows your matrix is tridiagonal, you can declare the matrix as sparse, using `spdiags`, which can be used to create a tridiagonal matrix. You could also create the tridiagonal matrix conventionally, and then convert the resulting matrix to a sparse one using `sparse`.

One other thing you need to know. Suppose you need to solve a group of linear equations with the same coefficient matrix, *i.e.*, you need to compute $F^{-1}a_1, \dots, F^{-1}a_m$, where F is invertible and a_i are column vectors. By concatenating columns, this can be expressed as a single matrix

$$[F^{-1}a_1 \ \dots \ F^{-1}a_m] = F^{-1}[a_1 \ \dots \ a_m].$$

To compute this matrix using Matlab, you should collect the righthand sides into one matrix (as above) and use Matlab's backslash operator: `F\A`. This will do the right thing: factor the matrix F once, and carry out multiple back substitutions for the righthand sides.

Solution.

- (a) The objective function is

$$x^T(A^T A + \delta\Delta + \eta I)x - 2b^T A x + b^T b,$$

where $A \in \mathbf{R}^{k \times n}$ is the matrix with rows a_i , and $\Delta \in \mathbf{R}^{n \times n}$ is the tridiagonal matrix

$$\Delta = \begin{bmatrix} 1 & -1 & 0 & \cdots & 0 & 0 & 0 \\ -1 & 2 & -1 & \cdots & 0 & 0 & 0 \\ 0 & -1 & 2 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 2 & -1 & 0 \\ 0 & 0 & 0 & \cdots & -1 & 2 & -1 \\ 0 & 0 & 0 & \cdots & 0 & -1 & 1 \end{bmatrix}.$$

Since the problem is unconstrained, the optimality conditions are

$$(A^T A + \delta\Delta + \eta I)x^* = A^T b. \quad (39)$$

- (b) If no structure is exploited, then solving (39) costs approximately $(1/3)n^3$ flops. If $k \ll n$, we need to solve a system $Fx = g$ where F is the sum of a tridiagonal and a (relatively) low-rank matrix. We can use the Sherman-Morrison-Woodbury formula

$$x^* = (\delta\Delta + \eta I)^{-1}g - (\delta\Delta + \eta I)^{-1}A^T(I + A(\delta\Delta + \eta I)^{-1}A^T)^{-1}A(\delta\Delta + \eta I)^{-1}g$$

to efficiently solve (39) as follows:

- (i) Solve $(\delta\Delta + \eta I)z_1 = g$ and $(\delta\Delta + \eta I)Z_2 = A^T$ for z_1 and Z_2 . Since $\delta\Delta + \eta I$ is tridiagonal, the total cost for this is approximately $6nk + 10n$ flops (4n for factorization and $6n(k+1)$ for the solves).
- (ii) Form Az_1 and AZ_2 ($2nk + 2nk^2$ flops).
- (iii) Solve $(I + AZ_2)z_3 = Az_1$ for z_3 ($(1/3)k^3$ flops).

(iv) Form $x^* = z_1 - Z_2 z_3$ ($2nk$ flops).

The total flop count, keeping only leading terms, is $2nk^2$ flops, which is much smaller than $(1/3)n^3$ when $k \ll n$.

(c) Here's the Matlab code:

```
clear all; close all;

n = 4000;
k = 100;
delta = 1;
eta = 1;

A = rand(k,n);
b = rand(k,1);

e = ones(n,1);
D = spdiags([-e 2*e -e], [-1 0 1], n,n);
D(1,1) = 1; D(n,n) = 1;
I = sparse(1:n,1:n,1);

F = A'*A + eta*I + delta*D;
P = eta*I + delta*D; %P is cheap to invert since it's tridiagonal
g = A'*b;

%Directly computing optimal solution
fprintf('\nComputing solution directly\n');
s1 = cputime;
x_gen = F\g;
s2 = cputime;
fprintf('Done (in %g sec)\n',s2-s1);

fprintf('\nComputing solution using efficient method\n');
%x_eff = P^{-1}g - P^{-1}A'(I + AP^{-1}A')^{-1}AP^{-1}g.

t1= cputime;
Z_0 = P\[g A'];
z_1 = Z_0(:,1);
%z_2 = A*z_1;
Z_2 = Z_0(:,2:k+1);
z_3 = (sparse(1:k,1:k,1) +A*Z_2)\(A*z_1);
x_eff = z_1 - Z_2*z_3;
t2 = cputime;
fprintf('Done (in %g sec)\n',t2-t1);

fprintf('\nrelative error = %e\n',norm(x_eff-x_gen)/norm(x_gen) );
```

8.6 Newton method for approximate total variation de-noising. Total variation de-noising is based on the bi-criterion problem with the two objectives

$$\|x - x^{\text{cor}}\|_2, \quad \phi_{\text{tv}}(x) = \sum_{i=1}^{n-1} |x_{i+1} - x_i|.$$

Here $x^{\text{cor}} \in \mathbf{R}^n$ is the (given) corrupted signal, $x \in \mathbf{R}^n$ is the de-noised signal to be computed, and ϕ_{tv} is the total variation function. This bi-criterion problem can be formulated as an SOCP, or, by squaring the first objective, as a QP. In this problem we consider a method used to approximately formulate the total variation de-noising problem as an unconstrained problem with twice differentiable objective, for which Newton's method can be used.

We first observe that the Pareto optimal points for the bi-criterion total variation de-noising problem can found as the minimizers of the function

$$\|x - x^{\text{cor}}\|_2^2 + \mu \phi_{\text{tv}}(x),$$

where $\mu \geq 0$ is parameter. (Note that the Euclidean norm term has been squared here, and so is twice differentiable.) In *approximate total variation de-noising*, we substitute a twice differentiable approximation of the total variation function,

$$\phi_{\text{atv}}(x) = \sum_{i=1}^{n-1} \left(\sqrt{\epsilon^2 + (x_{i+1} - x_i)^2} - \epsilon \right),$$

for the total variation function ϕ_{tv} . Here $\epsilon > 0$ is parameter that controls the level of approximation. In approximate total variation de-noising, we use Newton's method to minimize

$$\psi(x) = \|x - x^{\text{cor}}\|_2^2 + \mu \phi_{\text{atv}}(x).$$

(The parameters $\mu > 0$ and $\epsilon > 0$ are given.)

- (a) Find expressions for the gradient and Hessian of ψ .
- (b) Explain how you would exploit the structure of the Hessian to compute the Newton direction for ψ efficiently. (Your explanation can be brief.) Compare the approximate flop count for your method with the flop count for a generic method that does not exploit any structure in the Hessian of ψ .
- (c) Implement Newton's method for approximate total variation de-noising. Get the corrupted signal x^{cor} from the file `approx_tv_denoising_data.m`, and compute the de-noised signal x^* , using parameters $\epsilon = 0.001$, $\mu = 50$ (which are also in the file). Use line search parameters $\alpha = 0.01$, $\beta = 0.5$, initial point $x^{(0)} = 0$, and stopping criterion $\lambda^2/2 \leq 10^{-8}$. Plot the Newton decrement versus iteration, to verify asymptotic quadratic convergence. Plot the final smoothed signal x^* , along with the corrupted one x^{cor} .

Solution.

- (a) The gradient and Hessian of ψ are most easily derived using the chain rule. We first start with the expressions

$$\nabla \psi(x) = 2(x - x^{\text{cor}}) + \mu \nabla \phi_{\text{atv}}(x), \quad \nabla^2 \psi(x) = 2I + \mu \nabla^2 \phi_{\text{atv}}(x),$$

so the challenge is to find the gradient the Hessian of ϕ_{atv} . Let $f : \mathbf{R} \rightarrow \mathbf{R}$ denote the function

$$f(u) = \sqrt{\epsilon^2 + u^2} - \epsilon,$$

which is our approximation of the absolute value function. Its first and second derivatives are

$$f'(u) = u(\epsilon^2 + u^2)^{-1/2}, \quad f''(u) = \epsilon^2(\epsilon^2 + u^2)^{-3/2}.$$

We define $F : \mathbf{R}^{(n-1)} \rightarrow \mathbf{R}$ as

$$F(u_1, \dots, u_{n-1}) = \sum_{i=1}^{n-1} f(u_i),$$

i.e., $F(u)$ is the sum of the approximate absolute values of the components of u . Its gradient and Hessian are

$$\nabla F(u) = (f'(u_1), \dots, f'(u_{n-1})), \quad \nabla^2 F(u) = \text{diag}(f''(u_1), \dots, f''(u_{n-1})).$$

We have $\phi_{\text{atv}}(x) = F(Dx)$, where D is the forward difference matrix

$$D = \begin{bmatrix} -1 & 1 & & & \\ & -1 & 1 & & \\ & & \ddots & \ddots & \\ & & & -1 & 1 \end{bmatrix} \in \mathbf{R}^{(n-1) \times n}.$$

Using the chain rule, we obtain

$$\nabla \phi_{\text{atv}}(x) = D^T \nabla F(Dx), \quad \nabla^2 \phi_{\text{atv}}(x) = D^T \nabla^2 F(Dx) D.$$

Putting it all together, we get

$$\begin{aligned} \nabla \psi(x) &= 2(x - x^{\text{cor}}) + \mu D^T \nabla F(Dx) \\ \nabla^2 \psi(x) &= 2I + \mu D^T \nabla^2 F(Dx) D. \end{aligned}$$

Note that the Hessian is tridiagonal.

We can also write out the gradient and Hessian explicitly in terms of x , by components:

$$\frac{\partial \psi}{\partial x_i} = \begin{cases} 2(x_1 - x_1^{\text{cor}}) - \mu(x_2 - x_1)(\epsilon^2 + (x_2 - x_1)^2)^{-1/2} & i = 1 \\ 2(x_i - x_i^{\text{cor}}) - \mu(x_{i+1} - x_i)(\epsilon^2 + (x_{i+1} - x_i)^2)^{-1/2} + \mu(x_i - x_{i-1})(\epsilon^2 + (x_i - x_{i-1})^2)^{-1/2} & i = 2, \dots, n-1 \\ 2(x_n - x_n^{\text{cor}}) + \mu(x_n - x_{n-1})(\epsilon^2 + (x_n - x_{n-1})^2)^{-1/2} & i = n. \end{cases}$$

The Hessian components are:

$$\frac{\partial^2 \psi}{\partial x_i \partial x_j} = \begin{cases} 2 + \mu\epsilon^2(\epsilon^2 + (x_2 - x_1)^2)^{-3/2} & i = j = 1 \\ 2 + \mu\epsilon^2(\epsilon^2 + (x_i - x_{i-1})^2)^{-3/2} + \mu\epsilon^2(\epsilon^2 + (x_{i+1} - x_i)^2)^{-3/2} & i = j = 2, \dots, n-1 \\ 2 + \mu\epsilon^2(\epsilon^2 + (x_n - x_{n-1})^2)^{-3/2} & i = j = n \\ -\mu\epsilon^2(\epsilon^2 + (x_i - x_{i-1})^2)^{-3/2} & i - j = 1 \\ -\mu\epsilon^2(\epsilon^2 + (x_j - x_{j-1})^2)^{-3/2} & j - i = 1 \\ 0 & \text{otherwise.} \end{cases}$$

- (b) Let $H = \nabla^2\psi(x)$ and $g = \nabla\psi(x)$. The Newton step is found by solving the linear equations $H\Delta x_{nt} = -g$. A generic method to solve the equations, that does not exploit structure in H , costs $O(n^3)$ flops. Since H is tridiagonal, it is banded with bandwidth $k = 1$. We can form its Cholesky factor (which is lower bidiagonal), and carry out the forward and back substitutions, in $O(nk^2)$ flops, which is the same as $O(n)$ flops. Thus, by exploiting the fact that H is tridiagonal, we can compute Δx_{nt} in $O(n)$ flops. Of course, that's much faster than $O(n^3)$ flops. (The memory requirement drops from $O(n^2)$ to $O(n)$.)
- (c) The only trick to implementing Newton's method is to be sure that Matlab exploits the fact that H is tridiagonal. This can be done by defining the Hessian H to be a sparse matrix. Once this is done, the Newton step can be computed extremely efficiently.

The following Matlab code implements Newton's method.

```
% Newton method for approximate total variation de-noising
%
% problem data
approx_tv_denoising_data;
D = spdiags([-1*ones(n,1) ones(n,1)], 0:1, n-1, n);

% Newton's method
ALPHA = 0.01;
BETA = 0.5;
MAXITERS = 100;
NTTOL = 1e-10;

x = zeros(n,1);
newt_dec = [];

for iter = 1:MAXITERS
    d = (D*x);
    val = (x-xcor)'*(x-xcor) + ...
        MU*sum(sqrt(EPSILON^2+d.^2)-EPSILON*ones(n-1,1));
    grad = 2*(x - xcor) + ...
        MU*D'*(d./sqrt(EPSILON^2+d.^2));
    hess = 2*speye(n) + ...
        MU*D'*spdiags(EPSILON^2*(EPSILON^2+d.^2).^( -3/2 ),0,n-1,n-1)*D;

    v = -hess\grad;
    lambdasqr = -grad'*v; newt_dec = [newt_dec sqrt(lambdasqr)];

    if (lambdasqr/2) < NTTOL, break; end;

    t = 1;
    while ((x+t*v-xcor)'*(x+t*v-xcor) + ...
        MU*sum(sqrt(EPSILON^2+(D*(x+t*v)).^2)-EPSILON*ones(n-1,1)) > ...
        val - ALPHA*t*lambdasqr )
        t = BETA*t;
    end;
end;
```

```

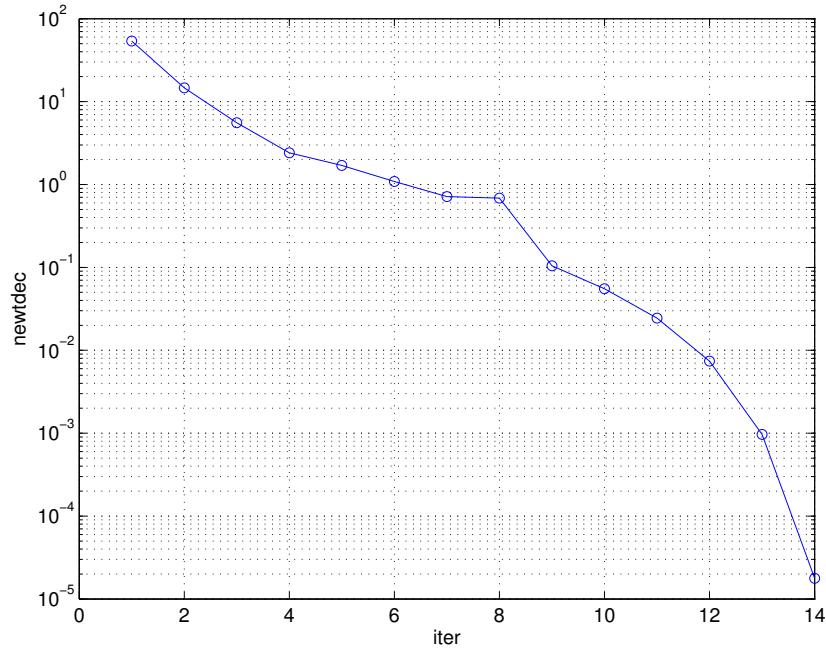
    end;
    x = x+t*v;
end;

% plotting results
figure;
semilogy([1:iter],newt_dec,'o-');
xlabel('iter'); ylabel('newtdec'); grid on;

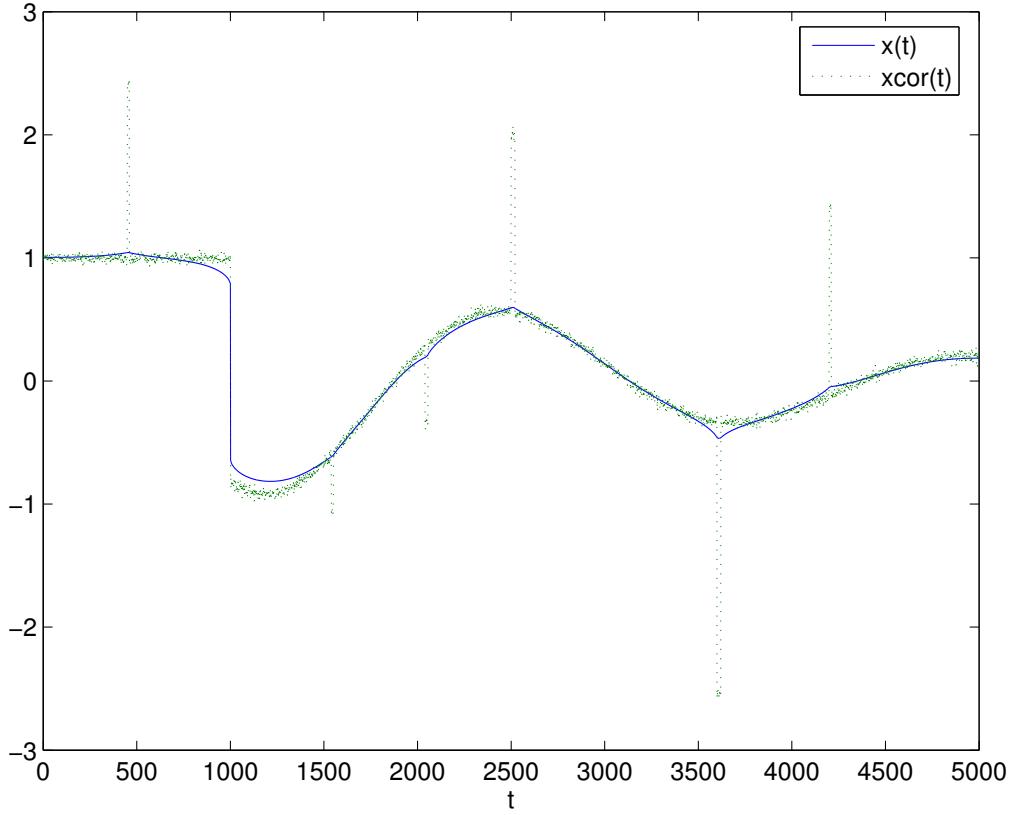
figure;
time = 1:5000;
plot(time,x,time,xcor,:);
xlabel('t'); legend('x(t)', 'xcor(t)');

```

We start our algorithm from $x = 0$. The resulting Newton decrement versus iteration is plotted below. High accuracy is obtained in 14 steps, with quadratic convergence observed in steps 10 through 14.



The final smoothed signal (shown in solid line type, in blue), along with the corrupted one (shown in dotted line type, green), is shown below.

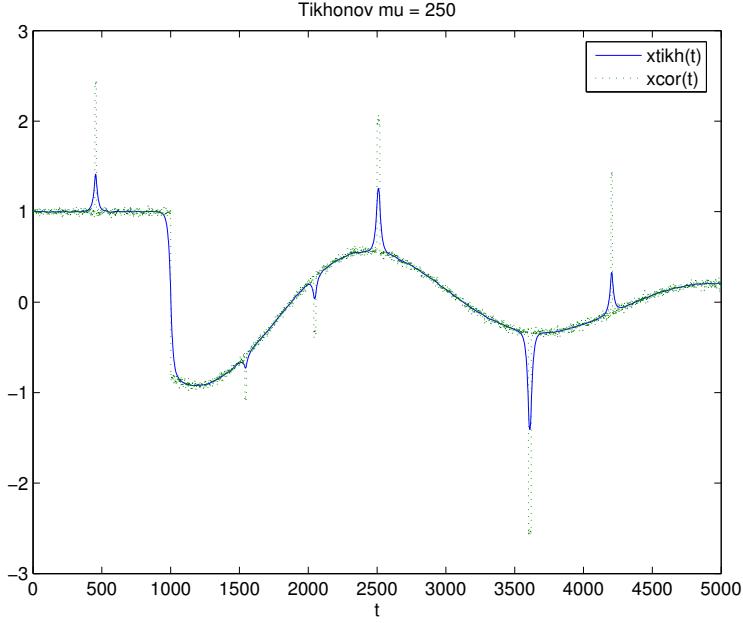


The approximate total variation de-noising has performed well, more or less removing the noise spikes as well as the faster smaller noise, while preserving the jump discontinuity in the signal.

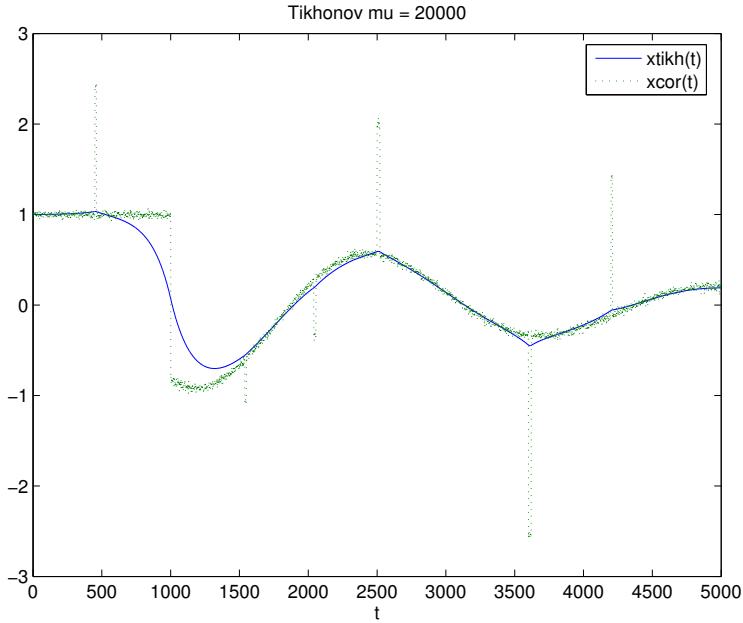
Although we didn't ask you to do it, it's interesting to compare approximate total variation de-noising with Tikhonov smoothing, with objective

$$\|x - x_{\text{cor}}\|_2^2 + \mu \|Dx\|_2^2.$$

Here we have a trade-off between preserving the jump discontinuity and attenuating noise spikes in the signal. We got the best jump preservation and some spike attenuation for $\mu = 250$ which gives the signal shown below.



To achieve a similar level of spike attenuation as the one obtained by approximate total variation de-noising, we choose $\mu = 20000$, which gives the signal shown below. The spikes are more or less removed, but we have significantly distorted the jump discontinuity.



8.7 Derive the Newton equation for the unconstrained minimization problem

$$\text{minimize} \quad (1/2)x^T x + \log \sum_{i=1}^m \exp(a_i^T x + b_i).$$

Give an efficient method for solving the Newton system, assuming the matrix $A \in \mathbf{R}^{m \times n}$ (with rows a_i^T) is dense with $m \ll n$. Give an approximate flop count of your method.

Solution. The Hessian is

$$H = I + A^T \left(\mathbf{diag}(z) - zz^T \right) A,$$

where

$$z_i = \frac{\exp(a_i^T x + b_i)}{\sum_i \exp(a_i^T x + b_i)},$$

so H is diagonal plus a low rank term, and we can more or less follow the method of page 10-30 of the lecture notes. However $\mathbf{diag}(z) - zz^T$ is singular, since $(\mathbf{diag}(z) - zz^T)\mathbf{1} = 0$, so we cannot directly factor it using the Cholesky factorization. Note that

$$\mathbf{diag}(z) - zz^T = L \mathbf{diag}(z)^{-1} L^T$$

where

$$L = \mathbf{diag}(z) - zz^T.$$

The Newton system

$$\left(I + A^T \left(\mathbf{diag}(z) - zz^T \right) A \right) \Delta x = g$$

is therefore equivalent to

$$\begin{bmatrix} I & A^T L \\ L^T A & -\mathbf{diag}(z) \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta u \end{bmatrix} = \begin{bmatrix} g \\ 0 \end{bmatrix}.$$

Eliminating Δx gives an equation

$$(\mathbf{diag}(z) + L^T A A^T L) \Delta u = L^T A g.$$

with $m + 1$ variables.

The cost is roughly $(1/3)m^3 + m^2n$ flops.

8.8 We consider the equality constrained problem

$$\begin{aligned} &\text{minimize} && \mathbf{tr}(CX) - \log \det X \\ &\text{subject to} && \mathbf{diag}(X) = \mathbf{1}. \end{aligned}$$

The variable is the matrix $X \in \mathbf{S}^n$. The domain of the objective function is \mathbf{S}_{++}^n . The matrix $C \in \mathbf{S}^n$ is a problem parameter. This problem is similar to the analytic centering problem discussed in lecture 11 (p.18–19) and pages 553–555 of the textbook. The differences are the extra linear term $\mathbf{tr}(CX)$ in the objective, and the special form of the equality constraints. (Note that the equality constraints can be written as $\mathbf{tr}(A_i X) = 1$ with $A_i = e_i e_i^T$, a matrix of zeros except for the i, i element, which is equal to one.)

- (a) Show that X is optimal if and only if

$$X \succ 0, \quad X^{-1} - C \text{ is diagonal}, \quad \mathbf{diag}(X) = \mathbf{1}.$$

- (b) The Newton step ΔX at a feasible X is defined as the solution of the Newton equations

$$X^{-1} \Delta X X^{-1} + \mathbf{diag}(w) = -C + X^{-1}, \quad \mathbf{diag}(\Delta X) = 0,$$

with variables $\Delta X \in \mathbf{S}^n$, $w \in \mathbf{R}^n$. (Note the two meanings of the **diag** function: **diag**(w) is the diagonal matrix with the vector w on its diagonal; **diag**(ΔX) is the vector of the diagonal elements of ΔX .) Eliminating ΔX from the first equation gives an equation

$$\mathbf{diag}(X \mathbf{diag}(w) X) = \mathbf{1} - \mathbf{diag}(X C X).$$

This is a set of n linear equations in n variables, so it can be written as $Hw = g$. Give a simple expression for the coefficients of the matrix H .

- (c) Implement the feasible Newton method in Matlab. You can use $X = I$ as starting point. The code should terminate when $\lambda(X)^2/2 \leq 10^{-6}$, where $\lambda(X)$ is the Newton decrement.

You can use the Cholesky factorization to evaluate the cost function: if $X = LL^T$ where L is triangular with positive diagonal then $\log \det X = 2 \sum_i \log L_{ii}$.

To ensure that the iterates remain feasible, the line search has to consist of two phases. Starting at $t = 1$, you first need to backtrack until $X + t\Delta X \succ 0$. Then you continue the backtracking until the condition of sufficient decrease

$$f_0(X + t\Delta X) \leq f_0(X) + \alpha t \mathbf{tr}(\nabla f_0(X)\Delta X)$$

is satisfied. To check that a matrix $X + t\Delta X$ is positive definite, you can use the Cholesky factorization with two output arguments (`[R, p] = chol(A)` returns $p > 0$ if A is not positive definite).

Test your code on randomly generated problems of sizes $n = 10, \dots, 100$ (for example, using `n = 100; C = randn(n); C = C + C'`).

Solution.

- (a) The Lagrangian is

$$\begin{aligned} L(X, w) &= \mathbf{tr}(CX) - \log \det X + w^T \mathbf{diag}(X) - \mathbf{1}^T w \\ &= \mathbf{tr}(CX) - \log \det X + \mathbf{tr}(\mathbf{diag}(w)X) - \mathbf{1}^T w, \end{aligned}$$

and its gradient with respect to X is

$$\nabla_X L(X, w) = C - X^{-1} + \mathbf{diag}(w).$$

Therefore the optimality conditions are:

$$X \in \mathbf{dom} f_0, \quad X^{-1} - C - \mathbf{diag}(w) = 0, \quad \mathbf{diag}(X) = \mathbf{1}.$$

- (b) Eliminating ΔX from the first equation gives

$$\Delta X = -X C X + X - X \mathbf{diag}(w) X.$$

Substituting this in the second equation gives

$$\mathbf{diag}(\Delta X) = -\mathbf{diag}(X C X) + \mathbf{diag}(X) - \mathbf{diag}(X \mathbf{diag}(w) X) = 0.$$

Since X is assumed to be feasible, we can simplify this as

$$\mathbf{diag}(X \mathbf{diag}(w) X) = \mathbf{1} - \mathbf{diag}(X C X).$$

To write this as a linear equation $Hw = g$, we note that

$$(\text{diag}(X \text{ diag}(w)X))_i = \sum_{j=1}^n X_{ij}^2 w_j, \quad i = 1, \dots, n.$$

Therefore $H_{ij} = X_{ij}^2$, i.e., H is obtained from X by squaring its components.

(c) `maxiters = 50;`

`alpha = 0.01;`

`beta = 0.5;`

`tol = 1e-6;`

```

X = eye(n);
for iter = 1:maxiters

R = chol(X); % X = R' * R.
val = C(:)'*X(:) - 2*sum(log(diag(R)));

% Solve
%
%      inv(X) * dX * inv(X) + diag(y) = inv(X) - C
%      diag(dX) = 0
%
% using block elimination:
%
%      (X.^2) * y = 1 - diag(X*C*X)
%      dX = X - X * C * X - X*diag(y)*X.

y = X.^2 \ ( 1 - diag(X*C*X) );
dX = X - X * (C + diag(y)) * X;

% fprime = -trace( dX * X^-1 * dX * X^-1 )
%           = -|| R^{-T} * dX * R^{-1} ||_F^2

fprime = -norm(R' \ dX / R, 'fro')^2;
if -fprime/2 <= tol; break; end;
t = 1;
[R, p] = chol(X + t*dX);
while p
    t = beta*t;
    [R, p] = chol(X + t*dX);
end;
while ( -2*sum(log(diag(R))) + C(:)'*(X(:)+t*dX(:)) > ...
    val + alpha*t*fprime )
    t = beta*t;
    R = chol(X + t*dX);
end;

```

```

X = X + t * dX;
end;

```

- 8.9 Estimation of a vector from one-bit measurements.** A system of m sensors is used to estimate an unknown parameter $x \in \mathbf{R}^n$. Each sensor makes a noisy measurement of some linear combination of the unknown parameters, and quantizes the measured value to one bit: it returns $+1$ if the measured value exceeds a certain threshold, and -1 otherwise. In other words, the output of sensor i is given by

$$y_i = \mathbf{sign}(a_i^T x + v_i - b_i) = \begin{cases} 1 & a_i^T x + v_i \geq b_i \\ -1 & a_i^T x + v_i < b_i, \end{cases}$$

where a_i and b_i are known, and v_i is measurement error. We assume that the measurement errors v_i are independent random variables with a zero-mean unit-variance Gaussian distribution (*i.e.*, with a probability density $\phi(v) = (1/\sqrt{2\pi})e^{-v^2/2}$). As a consequence, the sensor outputs y_i are random variables with possible values ± 1 . We will denote $\mathbf{prob}(y_i = 1)$ as $P_i(x)$ to emphasize that it is a function of the unknown parameter x :

$$\begin{aligned} P_i(x) &= \mathbf{prob}(y_i = 1) = \mathbf{prob}(a_i^T x + v_i \geq b_i) = \frac{1}{\sqrt{2\pi}} \int_{b_i - a_i^T x}^{\infty} e^{-t^2/2} dt \\ 1 - P_i(x) &= \mathbf{prob}(y_i = -1) = \mathbf{prob}(a_i^T x + v_i < b_i) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{b_i - a_i^T x} e^{-t^2/2} dt. \end{aligned}$$

The problem is to estimate x , based on observed values $\bar{y}_1, \bar{y}_2, \dots, \bar{y}_m$ of the m sensor outputs.

We will apply the maximum likelihood (ML) principle to determine an estimate \hat{x} . In maximum likelihood estimation, we calculate \hat{x} by maximizing the *log-likelihood function*

$$l(x) = \log \left(\prod_{\bar{y}_i=1} P_i(x) \prod_{\bar{y}_i=-1} (1 - P_i(x)) \right) = \sum_{\bar{y}_i=1} \log P_i(x) + \sum_{\bar{y}_i=-1} \log(1 - P_i(x)).$$

- (a) Show that the maximum likelihood estimation problem

$$\text{maximize } l(x)$$

is a convex optimization problem. The variable is x . The measured vector \bar{y} , and the parameters a_i and b_i are given.

- (b) Solve the ML estimation problem with data defined in `one_bit_meas_data.m`, using Newton's method with backtracking line search. This file will define a matrix A (with rows a_i^T), a vector b , and a vector \bar{y} with elements ± 1 .

Remark. The Matlab functions `erfc` and `erfcx` are useful to evaluate the following functions:

$$\begin{aligned} \frac{1}{\sqrt{2\pi}} \int_{-\infty}^u e^{-t^2/2} dt &= \frac{1}{2} \mathbf{erfc}\left(-\frac{u}{\sqrt{2}}\right), & \frac{1}{\sqrt{2\pi}} \int_u^{\infty} e^{-t^2/2} dt &= \frac{1}{2} \mathbf{erfc}\left(\frac{u}{\sqrt{2}}\right) \\ \frac{1}{\sqrt{2\pi}} e^{u^2/2} \int_{-\infty}^u e^{-t^2/2} dt &= \frac{1}{2} \mathbf{erfcx}\left(-\frac{u}{\sqrt{2}}\right), & \frac{1}{\sqrt{2\pi}} e^{u^2/2} \int_u^{\infty} e^{-t^2/2} dt &= \frac{1}{2} \mathbf{erfcx}\left(\frac{u}{\sqrt{2}}\right). \end{aligned}$$

Solution.

(a) The problem is

$$\text{minimize } - \sum_{y_i=1}^m \log \Phi(-b_i + a_i^T x) - \sum_{y_i=-1}^m \log \Phi(b_i - a_i^T x).$$

where

$$\Phi(u) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^u e^{-t^2/2} dt.$$

$\Phi(u)$ is log-concave (it is the cumulative distribution function of a log-concave density; see exercise T3.55). Therefore $\Phi(a_i^T x - b_i)$ and $\Phi(b_i - a_i^T x)$ are log-concave.

(b) To simplify notation we redefine A and b as

$$A := \text{diag}(y)A, \quad b := \text{diag}(y)b.$$

This allows us to express the problem as

$$\text{minimize } h(Ax - b),$$

where $h : \mathbf{R}^m \rightarrow \mathbf{R}$ is defined as

$$h(w) = - \sum_{i=1}^m \log \Phi(w_i).$$

The gradient and Hessian of $f(x) = h(Ax - b)$ are given by

$$\nabla f(x) = A^T \nabla h(Ax - b), \quad \nabla^2 f(x) = A^T \nabla^2 h(Ax - b) A.$$

The first derivatives of h are

$$\begin{aligned} \frac{\partial h(w)}{\partial w_i} &= -\frac{\Phi'(w_i)}{\Phi(w_i)} \\ &= \frac{-1/\sqrt{2\pi}}{\exp(w_i^2/2)\Phi(w_i)}. \end{aligned}$$

The Hessian $\nabla^2 h(w)$ is diagonal with diagonal elements

$$\begin{aligned} \frac{\partial^2 h(w)}{\partial w_i^2} &= -\frac{\Phi''(w_i)}{\Phi(w_i)} + \frac{\Phi'(w_i)^2}{\Phi(w_i)^2} \\ &= \frac{w_i/\sqrt{2\pi}}{\exp(w_i^2/2)\Phi(w_i)} + \left(\frac{1/\sqrt{2\pi}}{\exp(w_i^2/2)\Phi(w_i)} \right)^2. \end{aligned}$$

In the following Matlab implementation we take the least-squares solution as starting point.

```
one_bit_meas_data;
[m,n] = size(A);
A = diag(y)*A;
b = y.*b;
x = A\b;
```

```

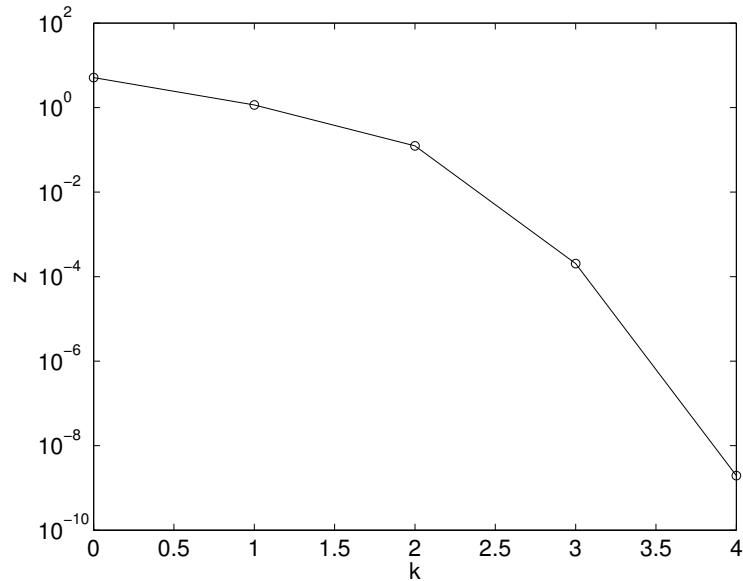
for k=1:50
    w = A*x-b;
    Phi = 0.5*erfc( -w/sqrt(2) );
    Phix = 0.5*sqrt(2*pi) * erfcx(-w/sqrt(2));
    val = -sum(log(Phi));
    grad = -A'* (1./Phix);
    hess = A'* diag((w + 1./Phix)./Phix) * A;
    v = -hess\grad;
    fprime = grad'*v
    if (-fprime/2 < 1e-8), break; end;
    t = 1;
    while ( -sum(log(0.5*erfc(-(A*(x+t*v)-b)/sqrt(2)))) > ...
        val + 0.01*t*fprime )
        t = t/2;
    end;
    x = x + t*v;
end;

```

This converges in a few iterations to

$$x = (-0.27, 9.15, 7.98, 6.70, 6.02, 5.0, 4.30, 2.68, 2.02, 0.68)$$

as shown in the plot.



8.10 Functions with bounded Newton decrement. Let $f : \mathbf{R}^n \rightarrow \mathbf{R}$ be a convex function with $\nabla^2 f(x) \succ 0$ for all $x \in \text{dom } f$ and Newton decrement bounded by a positive constant c :

$$\lambda(x)^2 \leq c \quad \forall x \in \text{dom } f.$$

Show that the function $g(x) = \exp(-f(x)/c)$ is concave.

Solution. The gradient and Hessian of g are

$$\begin{aligned}\nabla g(x) &= -\frac{e^{-f(x)/c}}{c} \nabla f(x) \\ \nabla^2 g(x) &= -\frac{e^{-f(x)/c}}{c} \nabla^2 f(x) + \frac{e^{-f(x)/c}}{c^2} \nabla f(x) \nabla f(x)^T.\end{aligned}$$

The Hessian is negative semidefinite if

$$\nabla^2 f(x) - \frac{1}{c} \nabla f(x) \nabla f(x)^T \succeq 0.$$

Using Schur complements this can be written in the equivalent form

$$\begin{bmatrix} \nabla^2 f(x) & \nabla f(x) \\ \nabla f(x)^T & c \end{bmatrix} \succeq 0.$$

By another application of the Schur complement theorem, this is equivalent to

$$c - \nabla f(x)^T \nabla^2 f(x)^{-1} \nabla f(x) \geq 0.$$

- 8.11 Monotone convergence of Newton's method.** Suppose $f : \mathbf{R} \rightarrow \mathbf{R}$ is strongly convex and smooth, and in addition, $f''' \leq 0$. Let x^* minimize f , and suppose Newton's method is initialized with $x^{(0)} < x^*$. Show that the iterates $x^{(k)}$ converge to x^* monotonically, and that a backtracking line search always takes a step size of one, i.e., $t^{(k)} = 1$.

9 Interior point methods

9.1 Dual feasible point from analytic center. We consider the problem

$$\begin{aligned} & \text{minimize} && f_0(x) \\ & \text{subject to} && f_i(x) \leq 0, \quad i = 1, \dots, m, \end{aligned} \tag{40}$$

where the functions f_i are convex and differentiable. For $u > p^*$, define $x_{\text{ac}}(u)$ as the analytic center of the inequalities

$$f_0(x) \leq u, \quad f_i(x) \leq 0, \quad i = 1, \dots, m,$$

i.e.,

$$x_{\text{ac}}(u) = \operatorname{argmin} \left(-\log(u - f_0(x)) - \sum_{i=1}^m \log(-f_i(x)) \right).$$

Show that $\lambda \in \mathbf{R}^m$, defined by

$$\lambda_i = \frac{u - f_0(x_{\text{ac}}(u))}{-f_i(x_{\text{ac}}(u))}, \quad i = 1, \dots, m$$

is dual feasible for the problem above. Express the corresponding dual objective value in terms of u , $x_{\text{ac}}(u)$ and the problem parameters.

Solution. Setting the gradient of the barrier function equal to zero we get

$$\frac{1}{u - f_0(x_{\text{ac}}(u))} \nabla f_0(x_{\text{ac}}(u)) + \sum_{i=1}^m \frac{1}{-f_i(x_{\text{ac}}(u))} \nabla f_i(x_{\text{ac}}(u)) = 0.$$

This shows that $x_{\text{ac}}(u)$ minimizes the Lagrangian

$$L(x, \lambda) = f_0(x) + \sum_{i=1}^m \lambda_i f_i(x)$$

for

$$\lambda_i = \frac{u - f_0(x_{\text{ac}}(u))}{-f_i(x_{\text{ac}}(u))}.$$

The corresponding dual objective value is

$$L(x_{\text{ac}}(u), \lambda) = f_0(x_{\text{ac}}(u)) - m(u - f_0(x_{\text{ac}}(u))).$$

9.2 Efficient solution of Newton equations. Explain how you would solve the Newton equations in the barrier method applied to the quadratic program

$$\begin{aligned} & \text{minimize} && (1/2)x^T x + c^T x \\ & \text{subject to} && Ax \preceq b \end{aligned}$$

where $A \in \mathbf{R}^{m \times n}$ is dense. Distinguish two cases, $m \gg n$ and $n \gg m$, and give the most efficient method in each case.

Solution. The Newton equations for

$$\text{minimize } t((1/2)x^T x + c^T x) - \sum_i \log(b_i - a_i^T x)$$

are

$$(tI + A^T \mathbf{diag}(d)^2 A) \Delta x = -tc - A^T d$$

where $d_k = 1/(b_k - a_k^T x)$.

If $m \gg n$ we can form this Newton system and solve it using the Cholesky factorization, at a cost of roughly $mn^2 + (1/3)n^3$ operations.

If $m \ll n$, it is more efficient to first write the equations as

$$\begin{bmatrix} tI & A^T \\ A & -\mathbf{diag}(d)^{-2} \end{bmatrix} \begin{bmatrix} \Delta x \\ v \end{bmatrix} = \begin{bmatrix} -tc - A^T d \\ 0 \end{bmatrix}$$

and then eliminate Δx :

$$(\mathbf{diag}(d)^{-2} + \frac{1}{t} A A^T) v = -Ac - \frac{1}{t} A A^T d.$$

This can be solved at a cost of roughly $nm^2 + (1/3)m^3$ operations.

9.3 Efficient solution of Newton equations. Describe an efficient method for solving the Newton equation in the barrier method for the quadratic program

$$\begin{aligned} \text{minimize} \quad & (1/2)(x - a)^T P^{-1}(x - a) \\ \text{subject to} \quad & 0 \preceq x \preceq \mathbf{1}, \end{aligned}$$

with variable $x \in \mathbf{R}^n$. The matrix $P \in \mathbf{S}^n$ and the vector $a \in \mathbf{R}^n$ are given.

Assume that the matrix P is large, positive definite, and sparse, and that P^{-1} is dense. ‘Efficient’ means that the complexity of the method should be much less than $O(n^3)$.

Solution. The barrier function is

$$\phi(x) = -\sum_{i=1}^n \log x_i - \sum_{i=1}^n \log(1 - x_i),$$

and its derivatives are

$$\begin{aligned} \nabla \phi(x) &= -\mathbf{diag}(x)^{-1} \mathbf{1} + \mathbf{diag}(1-x)^{-1} \mathbf{1} \\ \nabla^2 \phi(x) &= \mathbf{diag}(x)^{-2} + \mathbf{diag}(1-x)^{-2}. \end{aligned}$$

The Newton equation is

$$(tP^{-1} + D)\Delta x = g$$

where $D = \nabla^2 \phi(x)$, a positive diagonal matrix, and $g = -tP^{-1}(x - a) - \nabla \phi(x)$. To take advantage of the structure in P^{-1} , we rewrite the equation as

$$\begin{bmatrix} tP^{-1} & I \\ I & -D \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} g \\ 0 \end{bmatrix}.$$

Eliminating Δx from the first equation gives

$$(tD + P)\Delta y = Pg, \quad \Delta x = \frac{1}{t}P(g - \Delta y).$$

If P is sparse, then $D + (1/t)P$ is sparse, with the same sparsity pattern as P . The equation in Δy can therefore be solved efficiently via a sparse Cholesky factorization. The computation of Δx requires a sparse matrix-vector multiplication.

9.4 Dual feasible point from incomplete centering.

Consider the SDP

$$\begin{aligned} & \text{minimize} && \mathbf{1}^T x \\ & \text{subject to} && W + \mathbf{diag}(x) \succeq 0, \end{aligned}$$

with variable $x \in \mathbf{R}^n$, and its dual

$$\begin{aligned} & \text{maximize} && -\mathbf{tr} WZ \\ & \text{subject to} && Z_{ii} = 1, \quad i = 1, \dots, n \\ & && Z \succeq 0, \end{aligned}$$

with variable $X \in \mathbf{S}^n$. (These problems arise in a relaxation of the two-way partitioning problem, described on page 219; see also exercises 5.39 and 11.23.)

Standard results for the barrier method tell us that when x is on the central path, *i.e.*, minimizes the function

$$\phi(x) = t\mathbf{1}^T x + \log \det(W + \mathbf{diag}(x))^{-1}$$

for some parameter $t > 0$, the matrix

$$Z = \frac{1}{t}(W + \mathbf{diag}(x))^{-1}$$

is dual feasible, with objective value $-\mathbf{tr} WZ = \mathbf{1}^T x - n/t$.

Now suppose that x is strictly feasible, but not necessarily on the central path. (For example, x might be the result of using Newton's method to minimize ϕ , but with early termination.) Then the matrix Z defined above will not be dual feasible. In this problem we will show how to construct a dual feasible \hat{Z} (which agrees with Z as given above when x is on the central path), from any point x that is *near* the central path. Define $X = W + \mathbf{diag}(x)$, and let $v = -\nabla^2\phi(x)^{-1}\nabla\phi(x)$ be the Newton step for the function ϕ defined above. Define

$$\hat{Z} = \frac{1}{t} \left(X^{-1} - X^{-1} \mathbf{diag}(v) X^{-1} \right).$$

- (a) Verify that when x is on the central path, we have $\hat{Z} = Z$.
- (b) Show that $\hat{Z}_{ii} = 1$, for $i = 1, \dots, n$.
- (c) Let $\lambda(x) = \nabla\phi(x)^T \nabla^2\phi(x)^{-1} \nabla\phi(x)$ be the Newton decrement at x . Show that

$$\lambda(x) = \mathbf{tr}(X^{-1} \mathbf{diag}(v) X^{-1} \mathbf{diag}(v)) = \mathbf{tr}(X^{-1/2} \mathbf{diag}(v) X^{-1/2})^2.$$

- (d) Show that $\lambda(x) < 1$ implies that $\hat{Z} \succ 0$. Thus, when x is near the central path (meaning, $\lambda(x) < 1$), Z is dual feasible.

Solution.

(a) When x is on the central path, we have $v = 0$, so $\hat{Z} = Z$.

(b) $\nabla^2\phi(x)v = -t\mathbf{1} + \mathbf{diag} X^{-1}$, hence

$$\begin{aligned} Z_{ii} &= \frac{1}{t} \left((X^{-1})_{ii} - \sum_{k=1}^n (X^{-1})_{ik} v_k (X^{-1})_{ik} \right) \\ &= \frac{1}{t} \left((X^{-1})_{ii} + \sum_{k=1}^n (X^{-1})_{ik}^2 v_k \right) \\ &= \frac{1}{t} \left((X^{-1})_{ii} + \nabla^2\phi(\hat{x})v \right) \\ &= 1. \end{aligned}$$

(c) $\lambda^2 = v^T \nabla^2\phi(\hat{x})v = \sum_{ij} v_i v_j (X^{-1})_{ij}^2 = \text{tr } X^{-1} \mathbf{diag}(v) X^{-1} \mathbf{diag}(v)$.

(d) From 2, all eigenvalues of $(X^{-1/2} \mathbf{diag}(v) X^{-1/2})^2$ are less than one in absolute value. Therefore the eigenvalues of $(X^{-1/2} \mathbf{diag}(v) X^{-1/2})$ are less than one, *i.e.*,

$$I - X^{-1/2} \mathbf{diag}(v) X^{-1/2} \succ 0.$$

9.5 Standard form LP barrier method. In the following three parts of this exercise, you will implement a barrier method for solving the standard form LP

$$\begin{aligned} &\text{minimize} && c^T x \\ &\text{subject to} && Ax = b, \quad x \succeq 0, \end{aligned}$$

with variable $x \in \mathbf{R}^n$, where $A \in \mathbf{R}^{m \times n}$, with $m < n$. Throughout these exercises we will assume that A is full rank, and the sublevel sets $\{x \mid Ax = b, x \succeq 0, c^T x \leq \gamma\}$ are all bounded. (If this is not the case, the centering problem is unbounded below.)

(a) *Centering step.* Implement Newton's method for solving the centering problem

$$\begin{aligned} &\text{minimize} && c^T x - \sum_{i=1}^n \log x_i \\ &\text{subject to} && Ax = b, \end{aligned}$$

with variable x , given a strictly feasible starting point x_0 .

Your code should accept A , b , c , and x_0 , and return x^* , the primal optimal point, ν^* , a dual optimal point, and the number of Newton steps executed.

Use the block elimination method to compute the Newton step. (You can also compute the Newton step via the KKT system, and compare the result to the Newton step computed via block elimination. The two steps should be close, but if any x_i is very small, you might get a warning about the condition number of the KKT matrix.)

Plot $\lambda^2/2$ versus iteration k , for various problem data and initial points, to verify that your implementation gives asymptotic quadratic convergence. As stopping criterion, you can use $\lambda^2/2 \leq 10^{-6}$. Experiment with varying the algorithm parameters α and β , observing the effect on the total number of Newton steps required, for a fixed problem instance. Check that your computed x^* and ν^* (nearly) satisfy the KKT conditions.

To generate some random problem data (*i.e.*, A , b , c , x_0), we recommend the following approach. First, generate A randomly. (You might want to check that it has full rank.) Then generate a random positive vector x_0 , and take $b = Ax_0$. (This ensures that x_0 is strictly feasible.) The parameter c can be chosen randomly. To be sure the sublevel sets are bounded, you can add a row to A with all positive elements. If you want to be able to repeat a run with the same problem data, be sure to set the state for the uniform and normal random number generators.

Here are some hints that may be useful.

- We recommend computing λ^2 using the formula $\lambda^2 = -\Delta x_{\text{nt}}^T \nabla f(x)$. You don't really need λ for anything; you can work with λ^2 instead. (This is important for reasons described below.)
- There can be small numerical errors in the Newton step Δx_{nt} that you compute. When x is nearly optimal, the computed value of λ^2 , *i.e.*, $\lambda^2 = -\Delta x_{\text{nt}}^T \nabla f(x)$, can actually be (slightly) negative. If you take the squareroot to get λ , you'll get a complex number, and you'll never recover. Moreover, your line search will never exit. However, this only happens when x is nearly optimal. So if you exit on the condition $\lambda^2/2 \leq 10^{-6}$, everything will be fine, even when the computed value of λ^2 is negative.
- For the line search, you must first multiply the step size t by β until $x + t\Delta x_{\text{nt}}$ is feasible (*i.e.*, strictly positive). If you don't, when you evaluate f you'll be taking the logarithm of negative numbers, and you'll never recover.

- (b) *LP solver with strictly feasible starting point.* Using the centering code from part (a), implement a barrier method to solve the standard form LP

$$\begin{aligned} &\text{minimize} && c^T x \\ &\text{subject to} && Ax = b, \quad x \succeq 0, \end{aligned}$$

with variable $x \in \mathbf{R}^n$, given a strictly feasible starting point x_0 . Your LP solver should take as argument A , b , c , and x_0 , and return x^* .

You can terminate your barrier method when the duality gap, as measured by n/t , is smaller than 10^{-3} . (If you make the tolerance much smaller, you might run into some numerical trouble.) Check your LP solver against the solution found by CVX*, for several problem instances.

The comments in part (a) on how to generate random data hold here too.

Experiment with the parameter μ to see the effect on the number of Newton steps per centering step, and the total number of Newton steps required to solve the problem.

Plot the progress of the algorithm, for a problem instance with $n = 500$ and $m = 100$, showing duality gap (on a log scale) on the vertical axis, versus the cumulative total number of Newton steps (on a linear scale) on the horizontal axis.

Your algorithm should return a $2 \times k$ matrix **history**, (where k is the total number of centering steps), whose first row contains the number of Newton steps required for each centering step, and whose second row shows the duality gap at the end of each centering step. In order to get a plot that looks like the ones in the book (*e.g.*, figure 11.4, page 572), you should use the following code:

```
[xx, yy] = stairs(cumsum(history(1,:)), history(2,:));
semilogy(xx,yy);
```

- (c) *LP solver.* Using the code from part (b), implement a general standard form LP solver, that takes arguments A , b , c , determines (strict) feasibility, and returns an optimal point if the problem is (strictly) feasible.

You will need to implement a phase I method, that determines whether the problem is strictly feasible, and if so, finds a strictly feasible point, which can then be fed to the code from part (b). In fact, you can use the code from part (b) to implement the phase I method.

To find a strictly feasible initial point x_0 , we solve the phase I problem

$$\begin{aligned} & \text{minimize} && t \\ & \text{subject to} && Ax = b \\ & && x \succeq (1-t)\mathbf{1}, \quad t \geq 0, \end{aligned}$$

with variables x and t . If we can find a feasible (x, t) , with $t < 1$, then x is strictly feasible for the original problem. The converse is also true, so the original LP is strictly feasible if and only if $t^* < 1$, where t^* is the optimal value of the phase I problem.

We can initialize x and t for the phase I problem with any x^0 satisfying $Ax^0 = b$, and $t^0 = 2 - \min_i x_i^0$. (Here we can assume that $\min_i x_i^0 \leq 0$; otherwise x^0 is already a strictly feasible point, and we are done.) You can use a change of variable $z = x + (t-1)\mathbf{1}$ to transform the phase I problem into the form in part (b).

Check your LP solver against CVX* on several numerical examples, including both feasible and infeasible instances.

Solution.

- (a) The Newton step Δx_{nt} is defined by the KKT system:

$$\begin{bmatrix} H & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta x_{\text{nt}} \\ w \end{bmatrix} = \begin{bmatrix} -g \\ 0 \end{bmatrix},$$

where $H = \text{diag}(1/x_1^2, \dots, 1/x_n^2)$, and $g = c - (1/x_1, \dots, 1/x_n)$. The KKT system can be efficiently solved by block elimination, *i.e.*, by solving

$$AH^{-1}A^T w = -AH^{-1}g,$$

and setting $\Delta x_{\text{nt}} = -H^{-1}(A^T w + g)$. The KKT optimality condition is

$$A^T \nu^* + c - (1/x_1^*, \dots, 1/x_n^*) = 0.$$

When the Newton method converges, *i.e.*, $\Delta x_{\text{nt}} \approx 0$, w is the dual optimal point ν^* .

The following functions compute the analytic center using Newton's method. Here is the function in MATLAB.

```
function [x_star, nu_star, lambda_hist] = lp_acent(A,b,c,x_0)
% solves problem
% minimize c'*x - sum(log(x))
% subject to A*x = b
% using Newton's method, given strictly feasible starting point x0
```

```

% input (A, b, c, x_0)
% returns primal and dual optimal points
% lambda_hist is a vector showing lambda^2/2 for each newton step
% returns [], [] if MAXITERS reached, or x_0 not feasible

% algorithm parameters
ALPHA = 0.01;
BETA = 0.5;
EPSILON = 1e-3;
MAXITERS = 100;

if (min(x_0) <= 0) || (norm(A*x_0 - b) > 1e-3) % x0 not feasible
    fprintf('FAILED');
    nu_star = []; x_star = []; lambda_hist=[];
    return;
end

m = length(b);
n = length(x_0);

x = x_0; lambda_hist = [];
for iter = 1:MAXITERS
    H = diag(x.^(-2));
    g = c - x.^(-1);
    % lines below compute newton step via whole KKT system
    % M = [ H A'; A zeros(m,m)];
    % d = M\[-g; zeros(m,1)];
    % dx = d(1:n);
    % w = d(n+1:end);

    % newton step by elimination method
    w = (A*diag(x.^2)*A')\(-A*diag(x.^2)*g);
    dx = -diag(x.^2)*(A'*w + g);

    lambdasqr = -g'*dx; % dx'*H*dx;
    lambda_hist = [lambda_hist lambdasqr/2];
    if lambdasqr/2 <= EPSILON break; end

    % backtracking line search
    % first bring the point inside the domain
    t = 1; while min(x+t*dx) <= 0 t = BETA*t; end
    % now do backtracking line search
    while c'*(t*dx)-sum(log(x+t*dx))+sum(log(x))-ALPHA*t*g'*dx> 0
        t = BETA*t;
    end

```

```

x = x + t*dx;
end

if iter == MAXITERS % MAXITERS reached
    fprintf('ERROR: MAXITERS reached.\n');
    x_star = [] ; nu_star = [];
else
    x_star = x;
    nu_star = w;
end

```

Here is the function in Python.

```

# solves the problem
#   minimize c *x - sum(log(x))
#   subject to A*x = b
# using Newton's method, given strictly feasible starting point x0
# returns
#   x_star: primal optimal point
#   nu_star: dual optimal point
#   lambda_hist: array of lambda^2/2 for each newton step
# x_star and nu_star will be empty arrays
#   if max iterations reached or x_0 is not feasible
def lp_agent(A, b, c, x_0):
    # parameters
    x_0 = x_0.reshape(len(x_0), 1)
    b = b.reshape(len(b), 1)
    c = c.reshape(len(c), 1)
    ALPHA = 0.01
    BETA = 0.5
    EPSILON = 1e-3
    MAXITERS = 100

    lambda_hist = np.array([])
    A = np.matrix(A)

    if min(x_0) <= 0 or np.linalg.norm(np.dot(A, x_0) - b) > EPSILON:
        print "ERROR: x_0 not feasible"
        return np.array([]), np.array([]), lambda_hist

    m = b.size
    n = x_0.size
    x = x_0
    for iterNum in xrange(MAXITERS):
        # .reshape(n) makes x into a 1-D array instead of a 2-D column
        H = np.diag(x.reshape(n) ** (-2))
        g = c - x ** (-1)

```

```

# newton step by elimination method
X = np.diag(x.reshape(n) ** 2)
w = np.linalg.lstsq(A * X * A.T, -A * X * g)[0]
dx = -X * (A.T * w + g)

lambdasqr = - np.dot(g.reshape(n), dx.reshape(n).T);
lambda_hist = np.append(lambda_hist, lambdasqr/2)

if lambdasqr / 2 <= EPSILON:
    return x, w, lambda_hist

# backtracking line search
t = 1
# first bring the point inside the domain
while min(x + t*dx) <= 0: t *= BETA
# backtracking line search
while t*np.dot(c.reshape(n), dx.reshape(n).T) - np.sum(np.log(x.reshape(n))
    t *= BETA
x += t*dx
print "ERROR: MAXITERS reached"
return np.array([]), np.array([]), lambda_hist

```

Here is the function in Julia.

```

# solves the problem
#   minimize c *x - sum(log(x))
#   subject to A*x = b
# using Newton s method, given strictly feasible starting point x0
# returns
#   x_star: primal optimal point
#   nu_star: dual optimal point
#   lambda_hist: array of lambda^2/2 for each newton step
# x_star and nu_star will be empty arrays
#   if max iterations reached or x_0 is not feasible
function lp_acent(A, b, c, x_0)
    # parameters
    ALPHA = 0.01
    BETA = 0.5
    EPSILON = 1e-3
    MAXITERS = 100

    lambda_hist = Float64 []

    # check feasibility of x0
    if minimum(x_0) <= 0 || norm(A*x_0 - b) > EPSILON
        println("ERROR: x_0 not feasible")
    end

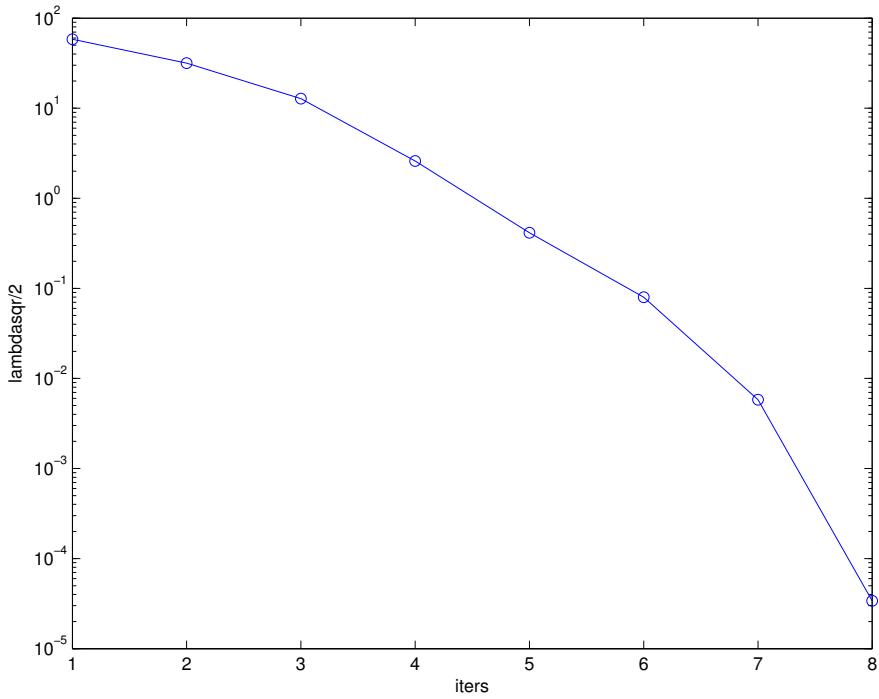
```

```

        return Float64 [] , Float64 [] , lambda_hist
    else
        m = length(b)
        n = length(x_0)
        x = x_0
        for iter = 1:MAXITERS
            H = diagm(x.^(-2))
            g = c - x.^(-1)
            # lines below compute newton step via whole KKT system
            # M = [ H A ; A zeros(m,m) ];
            # d = M\[-g; zeros(m,1)];
            # dx = d(1:n);
            # w = d(n+1:end);
            # newton step by elimination method
            X = diagm(x.^2)
            w = (A*X*A')\(-A*X*g)
            dx = -X*(A'*w + g)
            lambdasqr = -dot(g, dx) # d x *H*dx;
            push!(lambda_hist, lambdasqr/2)
            if lambdasqr/2 <= EPSILON
                return x_star, nu_star, lambda_hist
            end
            # backtracking line search
            # first bring the point inside the domain
            t = 1
            while minimum(x + t*dx) <= 0
                t *= BETA
            end
            # now do backtracking line search
            while t*dot(c, dx) - sum(log(x+t*dx)) + sum(log(x)) - ALPHA*t*dot(g, dx)
                t *= BETA
            end
            x += t*dx
            x_star = x
            nu_star = w
        end
        println("ERROR: MAXITERS reached")
        return Float64 [] , Float64 [] , lambda_hist
    end
end

```

The random data is generated as given in the problem statement, with $A \in \mathbf{R}^{100 \times 500}$. The Newton decrement versus number of Newton steps is plotted below. Quadratic convergence is clear. The Newton direction computed by the two methods are very close. The KKT optimality conditions are verified for the points returned by the function.



- (b) The following functions solve the LP using the barrier method. Here is the function in MATLAB.

```

function [x_star, history, gap] = lp_barrier(A,b,c,x_0)
% solves standard form LP
% minimize      c^T x
% subject to    Ax = b, x >=0;
% using barrier method, given strictly feasible x0
% uses function std_form_LP_acent() to carry out centering steps
% returns:
% - primal optimal point x_star
% - history, a 2xk matrix that returns number of newton steps
% in each centering step (top row) and duality gap (bottom row)
% (k is total number of centering steps)
% - gap, optimal duality gap

% barrier method parameters
T_0 = 1;
MU = 20;
EPSILON = 1e-3; % duality gap stopping criterion

n = length(x_0);
t = T_0;
x = x_0;
history = [];

```

```

while(1)
    [x_star, nu_star, lambda_hist] = lp_acent(A,b,t*c,x);
    x = x_star;
    gap = n/t;
    history = [history [length(lambda_hist); gap]];
    if gap < EPSILON break; end
    t = MU*t;
end

```

Here is the function in Python.

```

# solves standard form LP
# minimize      c^T x
# subject to    Ax = b, x >=0;
# using barrier method, given strictly feasible x0
# uses function lp_acent() to carry out centering steps
# returns
#   x_star: primal optimal point x_star
#   gap: optimal duality gap
#   num_newton_steps: array of number of newton steps per iteration
#   duality_gaps: array of duality gap each iteration
# x_star will be empty arry if the centering steps are infeasible or
# reach maximum iterations
def lp_barrier(A, b, c, x_0):
    # parameters
    T_0 = 1
    MU = 20
    EPSILON = 1e-3 # duality gap stopping criterion
    n = len(x_0)
    t = T_0
    x = x_0
    num_newton_steps = list()
    duality_gaps = list()
    gap = float(n) / t
    while True:
        x_star, nu_star, lambda_hist = lp_acent(A, b, t*c, x)
        # invalid centring step
        if len(x_star) == 0:
            return np.array([]), gap, num_newton_steps, duality_gaps
        x = x_star
        gap = float(n) / t
        num_newton_steps.append(len(lambda_hist))
        duality_gaps.append(gap)
        if gap < EPSILON:
            return x_star, gap, num_newton_steps, duality_gaps
        t *= MU

```

Here is the function in Julia.

```
# solves standard form LP
# minimize      c^T x
# subject to    Ax = b, x >=0;
# using barrier method, given strictly feasible x0
# uses function lp_acent() to carry out centering steps
# returns
#   x_star: primal optimal point x_star
#   gap: optimal duality gap
#   num_newton_steps: array of number of newton steps per iteration
#   duality_gaps: array of duality gap each iteration
# x_star will be empty arry if the centering steps are infeasible or
# reach maximum iterations
function lp_barrier(A, b, c, x_0)
    # barrier method parameters
    T_0 = 1
    MU = 20
    EPSILON = 1e-3 # duality gap stopping criterion
    n = length(x_0)
    t = T_0
    x = x_0
    num_newton_steps = Int64 []
    duality_gaps = Float64 []
    while (true)
        x_star, nu_star, lambda_hist = lp_acent(A, b, t*c, x)
        # invalid centering step
        if length(x_star) == 0
            return Float64 [], gap, num_newton_steps, duality_gaps
        end
        x = x_star
        gap = n/t
        push!(num_newton_steps, length(lambda_hist))
        push!(duality_gaps, gap)
        if gap < EPSILON
            return x_star, gap, num_newton_steps, duality_gaps
        end
        t *= MU
    end
end
```

The following scripts generate test data and plots the progress of the barrier method. The scripts also check the computed solution against CVX*. Here is the script in MATLAB.

```
% script that generates data and tests the functions
% std_form_LP_acent
% std_form_LP_barrier
```

```

clear all;
m = 10;
n = 200;

rand('seed',0);
randn('seed',0);
A = [randn(m-1,n); ones(1,n)];
x_0 = rand(n,1) + 0.1;
b = A*x_0;
c = randn(n,1);

% analytic centering
figure
[x_star, nu_star, lambda_hist] = lp_acent(A,b,c,x_0);
semilogy(lambda_hist,'bo-')
xlabel('iters')
ylabel('lambdasqr/2')
print -depsc lp_acent_newtondec.eps

% solve the LP with barrier
figure
[x_star, history, gap] = lp_barrier(A,b,c,x_0);
[xx, yy] = stairs(cumsum(history(1,:)),history(2,:));
semilogy(xx,yy,'bo-');
xlabel('iters')
ylabel('gap')
print -depsc lp_barrier_iters.eps

p_star = c'*x_star;

% solve LP using cvx for comparison
cvx_begin
    variable x(n)
    minimize(c'*x)
    subject to
        A*x == b
        x >= 0
cvx_end

fprintf('\n\nOptimal value found by barrier method:\n');
p_star
fprintf('Optimal value found by CVX:\n');
cvx_optval
fprintf('Duality gap from barrier method:\n');

```

gap

Here is the script in Python.

```
## tests lp_acent and lp_barrier
m = 10
n = 200
np.random.seed(2)
A = np.vstack((np.random.randn(m-1, n), np.ones((1, n))))
A = np.matrix(A)
x_0 = np.random.rand(n, 1) + 0.1;
b = A*x_0;
c = np.random.randn(n, 1);

# test lp_acent
x_star, nu_star, lambda_hist = lp_acent(A, b, c, x_0);
plt.semilogy(range(1, len(lambda_hist)+1), lambda_hist)
plt.show()

# test lp_barrier
x_star, gap, num_newton_steps, duality_gaps = lp_barrier(A, b, c, x_0);
plt.figure()
plt.step(np.cumsum(num_newton_steps), duality_gaps, where='post')
plt.yscale('log')
plt.show()

# compare to CVXPY
x = Variable(n)
obj = Minimize(sum_entries(mul_elemwise(c, x)))
prob = Problem(obj, [A * x == b, x >= 0])
prob.solve()
print "optimal value from barrier method:", np.dot(c.reshape(n), x_star.reshape(n))
print "optimal value from CVXPY:", prob.value;
print "duality gap from barrier method:", gap;
```

Here is the script in Julia.

```
# tests lp_acent and lp_barrier
using Gadfly, Convex, SCS
m = 10;
n = 200;
srand(1);
A = [randn(m-1,n); ones(1,n)];
x_0 = rand(n) + 0.1;
b = A*x_0;
c = randn(n);

# analytic centering
```

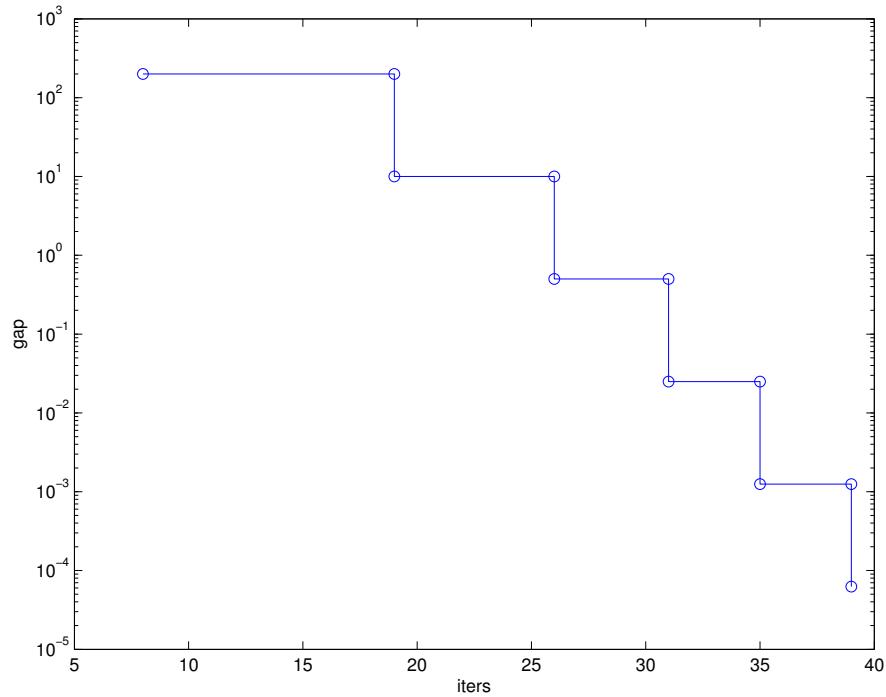
```

x_star, nu_star, lambda_hist = lp_acent(A, b, c, x_0);
p1 = plot(
    x=1:length(lambda_hist), y=lambda_hist, Geom.path, Geom.point,
    Guide.xlabel("iters"), Guide.ylabel("lambda^2/2"),
    Scale.y.log10
);
display(p1);

# solve LP with barrier
x_star, gap, num_newton_steps, duality_gaps = lp_barrier(A, b, c, x_0);
p2 = plot(
    x=cumsum(num_newton_steps), y=duality_gaps, Geom.step, Geom.point,
    Guide.xlabel("iters"), Guide.ylabel("gap"),
    Scale.y.log10
);
display(p2);
p_star = dot(c, x_star);

# solve LP with Convex.jl
x = Variable(n);
p = minimize(dot(c, x), A*x == b, x >= 0);
solve!(p, SCSSolver(max_iters=100000));
println("optimal value from barrier method: ", p_star);
println("optimal value from Convex.jl: ", p.optval);
println("duality gap from barrier method: ", gap);

```



- (c) The following functions implement the full LP solver (phase I and phase II). Here is the function in MATLAB.

```

function [x_star,p_star,gap,status,nsteps] = lp_solve(A,b,c);
% solves the LP
% minimize      c^T x
% subject to    Ax = b, x >= 0;
% using a barrier method
% computes a strictly feasible point by carrying out
% a phase I method
% returns:
% - a primal optimal point x_star
% - the primal optimal value p_star
% - status: either 'Infeasible' or 'Solved'
% - nsteps(1): number of newton steps for phase I
% - nsteps(2): number of newton steps for phase I

[m,n] = size(A);
nsteps = zeros(2,1);

% phase I
x0 = A\b; t0 = 2+max(0,-min(x0));
A1 = [A,-A*ones(n,1)];
b1 = b-A*ones(n,1);
z0 = x0+t0*ones(n,1)-ones(n,1);
c1 = [zeros(n,1);1];
[z_star, history, gap] = lp_barrier(A1,b1,c1,[z0;t0]);
if (z_star(n+1) >= 1)
    fprintf('\nProblem is infeasible\n');
    x_star = []; p_star = Inf; status = 'Infeasible';
    nsteps(1) = sum(history(1,:)); gap = [];
    return;
end
fprintf('\nFeasible point found\n');
nsteps(1) = sum(history(1,:));
x_0 = z_star(1:n)-z_star(n+1)*ones(n,1)+ones(n,1);

% phase II
[x_star, history, gap] = lp_barrier(A,b,c,x_0);
status = 'Solved'; p_star = c'*x_star;
nsteps(2) = sum(history(1,:));

```

Here is the function in Python.

```

# solves the LP
# minimize      c^T x
# subject to    Ax = b, x >= 0

```

```

# using a barrier method
# computes a strictly feasible point by carrying out a phase I method
# returns:
#   x_star: primal optimal point
#   p_star: primal optimal value
#   gap: optimal duality gap
#   status: either :Infeasible or :Optimal
#   nsteps: array of number of newton steps for phase I and phase II
def lp_solve(A, b, c):
    m, n = A.shape
    b = b.reshape(m,1)
    nsteps = np.zeros(2)
    # phase I
    x0 = np.linalg.lstsq(A, b)[0];
    t0 = 2 + max(0, -min(x0))

    A1 = np.hstack((A, -np.dot(A, np.ones(n)).reshape(m, 1)))
    b1 = b - np.dot(A, np.ones(n)).reshape(m, 1)
    z0 = x0.reshape(n,1) + t0 * np.ones((n,1)) - np.ones((n,1))
    c1 = np.vstack((np.zeros((n,1)), np.ones((1,1))))
    x_0 = np.vstack((z0, t0)).reshape(n+1, 1)
    z_star, gap, num_newton_steps, duality_gaps = lp_barrier(A1, b1, c1, x_0)
    # z_star = mat['z_star']
    nsteps[0] = sum(num_newton_steps)
    if len(z_star) == 0:
        print "Phase I: problem is infeasible"
        return np.array([]), np.inf, np.inf, "Infeasible", nsteps

    print "Phase I: feasible point found"
    x_0 = z_star[:n] - z_star[n][0] * np.ones((n,1)) + np.ones((n,1))

    # phase II
    x_star, gap, num_newton_steps, duality_gaps = lp_barrier(A, b, c, x_0)
    nsteps[1] = sum(num_newton_steps)
    if len(x_star) == 0:
        return np.array([]), np.inf, np.inf, "Infeasible", nsteps

    p_star = np.dot(c.reshape(len(c)), x_star.reshape(len(c)))
    return x_star, p_star, gap, "Optimal", nsteps

```

Here is the function in Julia.

```

# solves the LP
#   minimize      c^T x
#   subject to    Ax = b, x >= 0
# using a barrier method
# computes a strictly feasible point by carrying out a phase I method

```

```

# returns:
#   x_star: primal optimal point
#   p_star: primal optimal value
#   gap: optimal duality gap
#   status: either :Infeasible or :Optimal
#   nsteps: array of number of newton steps for phase I and phase II
function lp_solve(A, b, c)
    m, n = size(A)
    nsteps = zeros(2)
    # phase I
    x0 = A\b
    t0 = 2 + max(0, -minimum(x0))
    A1 = [A -A*ones(n)]
    b1 = b - A*ones(n)
    z0 = x0 + t0*ones(n)-ones(n)
    c1 = [zeros(n); 1]
    z_star, gap, num_newton_steps, duality_gaps = lp_barrier(A1, b1, c1, [z0; t0])
    nsteps[1] = sum(num_newton_steps)
    if z_star[n+1] >= 1
        println("Phase I: problem is infeasible")
        return Float64[], Inf, Inf, :Infeasible, nsteps
    end
    println("Phase I: feasible point found")
    x_0 = z_star[1:n] - z_star[n+1]*ones(n) + ones(n)
    # phase II
    x_star, gap, num_newton_steps, duality_gaps = lp_barrier(A, b, c, x_0)
    nsteps[2] = sum(num_newton_steps)
    if length(x_star) == 0
        return Float64[], Inf, Inf, :Infeasible, nsteps
    end
    p_star = dot(c, x_star)
    return x_star, p_star, gap, :Optimal, nsteps
end

```

We test our LP solver on two problem instances, one infeasible, and one feasible. We check our results against the output of CVX*. Here is the script in MATLAB.

```
% solves standard form LP for two problem instances
```

```

clear all;
m = 100;
n = 500;

% infeasible problem instance
rand('state',0);
randn('state',0);
A = [randn(m-1,n); ones(1,n)];

```

```

b = randn(m,1);
c = randn(n,1);

[x_star,p_star,gap,status,nsteps] = lp_solve(A,b,c);

% solve LP using cvx for comparison
cvx_begin
    variable x(n)
    minimize(c'*x)
    subject to
        A*x == b
        x >= 0
cvx_end

% feasible problem instance
A = [randn(m-1,n); ones(1,n)];
v = rand(n,1) + 0.1;
b = A*v;
c = randn(n,1);

[x_star,p_star,gap,status,nsteps] = lp_solve(A,b,c);

% solve LP using cvx for comparison
cvx_begin
    variable x(n)
    minimize(c'*x)
    subject to
        A*x == b
        x >= 0
cvx_end

fprintf('\n\nOptimal value found by barrier method:\n');
p_star
fprintf('Optimal value found by CVX:\n');
cvx_optval
fprintf('Duality gap from barrier method:\n');
gap

```

Here is the script in Python.

```

## tests lp_solve for infeasible and feasible problem
m = 100
n = 500

# infeasible problem
A = np.vstack((np.random.randn(m-1, n), np.ones((1, n))))
b = np.random.randn(m);

```

```

c = np.random.randn(n, 1);
x_star, p_star, gap, status, nsteps = lp_solve(A, b, c);

# compare to CVXPY
x = Variable(n)
obj = Minimize(sum_entries(mul_elemwise(c, x)))
prob = Problem(obj, [A * x == b, x >= 0])
prob.solve()

print "status from lp solver:", status
print "status from CVXPY:", prob.status
print

# feasible problem
A = np.vstack((np.random.randn(m-1, n), np.ones((1, n))))
v = np.random.rand(n) + 0.1
b = np.dot(A, v)
c = np.random.randn(n)
x_star, p_star, gap, status, nsteps = lp_solve(A, b, c);

# compare to CVXPY
x = Variable(n)
obj = Minimize(sum_entries(mul_elemwise(c, x)))
prob = Problem(obj, [A * x == b, x >= 0])
prob.solve()
print "optimal value from barrier method:", np.dot(c.reshape(n), x_star.reshape(n))
print "optimal value from CVXPY:", prob.value;
print "duality gap from barrier method:", gap;

```

Here is the script in Julia.

```

# tests lp_solve for infeasible and feasible problem
m = 100;
n = 500;

# infeasible problem
srand(1);
A = [randn(m-1,n); ones(1,n)];
b = randn(m);
c = randn(n);
x_star, p_star, gap, status, nsteps = lp_solve(A, b, c);
# comapare to Convex.jl
x = Variable(n)
p = minimize(dot(c, x), A*x == b, x >= 0)
solve!(p, SCSSolver(verbose=false));

println("status from lp solver: ", status)

```

```

println(" status from Convex.jl: ", p.status)

# feasible problem
A = [randn(m-1,n); ones(1,n)];
v = rand(n) + 0.1;
b = A*v;
c = randn(n);
x_star, p_star, gap, status, nsteps = lp_solve(A, b, c);
# compare to Convex.jl
x = Variable(n)
p = minimize(dot(c, x), A*x == b, x >= 0)
solve!(p, SCSSolver(max_iters=100000, verbose=false));
println(" optimal value from barrier method: ", p_star);
println(" optimal value from Convex.jl: ", p.optval);
println(" duality gap from barrier method: ", gap);

```

9.6 Primal and dual feasible points in the barrier method for LP. Consider a standard form LP and its dual

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Ax = b \\ & x \succeq 0 \end{array} \quad \begin{array}{ll} \text{maximize} & b^T y \\ \text{subject to} & A^T y \preceq c, \end{array}$$

with $A \in \mathbf{R}^{m \times n}$ and $\text{rank}(A) = m$. In the barrier method the (feasible) Newton method is applied to the equality constrained problem

$$\begin{array}{ll} \text{minimize} & tc^T x + \phi(x) \\ \text{subject to} & Ax = b, \end{array}$$

where $t > 0$ and $\phi(x) = -\sum_{i=1}^n \log x_i$. The Newton equation at a strictly feasible \hat{x} is given by

$$\begin{bmatrix} \nabla^2 \phi(\hat{x}) & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ w \end{bmatrix} = \begin{bmatrix} -tc - \nabla \phi(\hat{x}) \\ 0 \end{bmatrix}.$$

Suppose $\lambda(\hat{x}) \leq 1$ where $\lambda(\hat{x})$ is the Newton decrement at \hat{x} .

- (a) Show that $\hat{x} + \Delta x$ is primal feasible.
- (b) Show that $y = -(1/t)w$ is dual feasible.
- (c) Let p^* be the optimal value of the LP. Show that

$$c^T \hat{x} - p^* \leq \frac{n + \lambda(\hat{x})\sqrt{n}}{t}.$$

Solution.

- (a) $\hat{x} + \Delta x$ satisfies the equality constraint $A(\hat{x} + \Delta x) = A\hat{x} + A\Delta x = b$. It is nonnegative because, from the first equation

$$\lambda^2 = \Delta x^T \nabla^2 \phi(\hat{x}) \Delta x = \sum_{i=1}^n \left(\frac{\Delta x_i}{\hat{x}_i} \right)^2.$$

Therefore $\lambda \leq 1$ implies $|\Delta x_i| \leq \hat{x}_i$.

- (b) From the first part of the Newton equation

$$c + \frac{1}{t} A^T w = -\frac{1}{t} (\nabla \phi(\hat{x}) + \nabla^2 \phi(\hat{x}) \Delta x).$$

The i th component is

$$(c - A^T y)_i = \frac{1}{t \hat{x}_i} \left(1 - \frac{\Delta x_i}{\hat{x}_i} \right).$$

This is nonnegative because $|\Delta x_i| \leq \hat{x}_i$.

- (c) The gap between primal and dual objective is

$$c^T \hat{x} - b^T y = \hat{x}^T (c - A^T y) = \frac{1}{t} \sum_{i=1}^n \left(1 - \frac{\Delta x_i}{\hat{x}_i} \right) = \frac{1}{t} \left(n - \sum_{i=1}^n \frac{\Delta x_i}{\hat{x}_i} \right).$$

To bound the last term we use the Cauchy-Schwarz inequality $|\mathbf{1}^T v| \leq \sqrt{n} \|v\|_2$ with $v_i = \Delta x_i / \hat{x}_i$:

$$\left| \sum_{i=1}^n \frac{\Delta x_i}{\hat{x}_i} \right| \leq \sqrt{n} \lambda.$$

10 Mathematical background

10.1 Some famous inequalities. The Cauchy-Schwarz inequality states that

$$|a^T b| \leq \|a\|_2 \|b\|_2$$

for all vectors $a, b \in \mathbf{R}^n$ (see page 633 of the textbook).

- (a) Prove the Cauchy-Schwarz inequality.

Hint. A simple proof is as follows. With a and b fixed, consider the function $g(t) = \|a + tb\|_2^2$ of the scalar variable t . This function is nonnegative for all t . Find an expression for $\inf_t g(t)$ (the minimum value of g), and show that the Cauchy-Schwarz inequality follows from the fact that $\inf_t g(t) \geq 0$.

- (b) The 1-norm of a vector x is defined as $\|x\|_1 = \sum_{k=1}^n |x_k|$. Use the Cauchy-Schwarz inequality to show that

$$\|x\|_1 \leq \sqrt{n} \|x\|_2$$

for all x .

- (c) The *harmonic mean* of a positive vector $x \in \mathbf{R}_{++}^n$ is defined as

$$\left(\frac{1}{n} \sum_{k=1}^n \frac{1}{x_k} \right)^{-1}.$$

Use the Cauchy-Schwarz inequality to show that the arithmetic mean $(\sum_k x_k)/n$ of a positive n -vector is greater than or equal to its harmonic mean.

Solution.

- (a) First note that the inequality is trivially satisfied if $b = 0$. Assume $b \neq 0$. We write g as

$$g(t) = (a - tb)^T (a - tb) = \|a\|_2^2 + 2ta^T b + t^2 \|b\|_2^2.$$

Setting the derivative equal to zero gives $t = -(a^T b)/\|b\|^2$. Therefore

$$\inf_t g(t) = \|a\|_2^2 - \frac{(a^T b)^2}{\|b\|_2^2}.$$

The Cauchy-Schwarz inequality now follows from $\inf g(t) \geq 0$.

- (b) Define $a_k = |x_k|$, $b_k = 1$. Then

$$\|x\|_1 = a^T b \leq \|a\|_2 \|b\|_2 = \sqrt{n} \|x\|_2.$$

- (c) Define $a_k = \sqrt{x_k/n}$ and $b_k = 1/\sqrt{nx_k}$. Then

$$1 = (a^T b)^2 \leq \|a\|_2^2 \|b\|_2^2 = \left(\frac{1}{n} \sum_{k=1}^n x_k \right) \left(\sum_{k=1}^n \frac{1}{nx_k} \right).$$

Hence,

$$\frac{1}{n} \sum_k x_k \geq \frac{n}{\sum_k 1/x_k}.$$

10.2 Schur complements. Consider a matrix $X = X^T \in \mathbf{R}^{n \times n}$ partitioned as

$$X = \begin{bmatrix} A & B \\ B^T & C \end{bmatrix},$$

where $A \in \mathbf{R}^{k \times k}$. If $\det A \neq 0$, the matrix $S = C - B^T A^{-1} B$ is called the *Schur complement* of A in X . Schur complements arise in many situations and appear in many important formulas and theorems. For example, we have $\det X = \det A \det S$. (You don't have to prove this.)

- (a) The Schur complement arises when you minimize a quadratic form over some of the variables. Let $f(u, v) = (u, v)^T X(u, v)$, where $u \in \mathbf{R}^k$. Let $g(v)$ be the minimum value of f over u , i.e., $g(v) = \inf_u f(u, v)$. Of course $g(v)$ can be $-\infty$. Show that if $A \succ 0$, we have $g(v) = v^T S v$.
- (b) The Schur complement arises in several characterizations of positive definiteness or semidefiniteness of a block matrix. As examples we have the following three theorems:
 - $X \succ 0$ if and only if $A \succ 0$ and $S \succ 0$.
 - If $A \succ 0$, then $X \succeq 0$ if and only if $S \succeq 0$.
 - $X \succeq 0$ if and only if $A \succeq 0$, $B^T(I - AA^\dagger) = 0$ and $C - B^T A^\dagger B \succeq 0$, where A^\dagger is the pseudo-inverse of A . ($C - B^T A^\dagger B$ serves as a generalization of the Schur complement in the case where A is positive semidefinite but singular.)

Prove *one* of these theorems. (You can choose which one.)

Solution.

- (a) *If $A \succ 0$, then $g(v) = v^T S v$.*

We have $f(u, v) = u^T A u + 2v^T B u + v^T C v$. If $A \succ 0$, we can minimize f over u by setting the gradient with respect to u equal to zero. We obtain $u^*(v) = -A^{-1} B v$, and

$$g(v) = f(u^*(v), v) = v^T (C - B^T A^{-1} B) v = v^T S v.$$

- (b) *Positive definite and semidefinite block matrices.*

- $X \succ 0$ if and only if $A \succ 0$ and $S \succ 0$.

Suppose $X \succ 0$. Then $f(u, v) > 0$ for all non-zero (u, v) , and in particular, $f(u, 0) = u^T A u > 0$ for all non-zero u (hence, $A \succ 0$), and $f(-A^{-1} B v, v) = v^T (C - B^T A^{-1} B) v > 0$ (hence, $S = C - B^T A^{-1} B \succ 0$). This proves the ‘only if’ part.

To prove the ‘if’ part, we have to show that if $A \succ 0$ and $S \succ 0$, then $f(u, v) > 0$ for all nonzero (u, v) (that is, for all u, v that are not both zero). If $v \neq 0$, then it follows from (a) that

$$f(u, v) \geq \inf_u f(u, v) = v^T S v > 0.$$

If $v = 0$ and $u \neq 0$, $f(u, 0) = u^T A u > 0$.

- *If $A \succ 0$, then $X \succeq 0$ if and only if $S \succeq 0$.*

From part (a) we know that if $A \succ 0$, then $\inf_u f(u, v) = v^T S v$. If $S \succeq 0$, then

$$f(u, v) \geq \inf_u f(u, v) = v^T S v \geq 0$$

for all u, v , and hence $X \succeq 0$. This proves the ‘if’-part.

To prove the ‘only if’-part we note that $f(u, v) \geq 0$ for all (u, v) implies that $\inf_u f(u, v) \geq 0$ for all v , i.e., $S \succeq 0$.

- $X \succeq 0$ if and only if $A \succeq 0$, $B^T(I - AA^\dagger) = 0$, $C - B^TA^\dagger B \succeq 0$.

Suppose $A \in \mathbf{R}^{k \times k}$ with $\text{rank}(A) = r$. Then there exist matrices $Q_1 \in \mathbf{R}^{k \times r}$, $Q_2 \in \mathbf{R}^{k \times (k-r)}$ and an invertible diagonal matrix $\Lambda \in \mathbf{R}^{r \times r}$ such that

$$A = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} \Lambda & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} Q_1 & Q_2 \end{bmatrix}^T,$$

and $[Q_1 \ Q_2]^T [Q_1 \ Q_2] = I$. The matrix

$$\begin{bmatrix} Q_1 & Q_2 & 0 \\ 0 & 0 & I \end{bmatrix} \in \mathbf{R}^{n \times n}$$

is nonsingular, and therefore

$$\begin{aligned} \begin{bmatrix} A & B \\ B^T & C \end{bmatrix} \succeq 0 &\iff \begin{bmatrix} Q_1 & Q_2 & 0 \\ 0 & 0 & I \end{bmatrix}^T \begin{bmatrix} A & B \\ B^T & C \end{bmatrix} \begin{bmatrix} Q_1 & Q_2 & 0 \\ 0 & 0 & I \end{bmatrix} \succeq 0 \\ &\iff \begin{bmatrix} \Lambda & 0 & Q_1^T B \\ 0 & 0 & Q_2^T B \\ B^T Q_1 & B^T Q_2 & C \end{bmatrix} \succeq 0 \\ &\iff Q_2^T B = 0, \begin{bmatrix} \Lambda & Q_1^T B \\ B^T Q_1 & C \end{bmatrix} \succeq 0. \end{aligned}$$

We have $\Lambda \succ 0$ if and only if $A \succeq 0$. It can be verified that

$$A^\dagger = Q_1 \Lambda^{-1} Q_1^T, \quad I - AA^\dagger = Q_2 Q_2^T.$$

Therefore

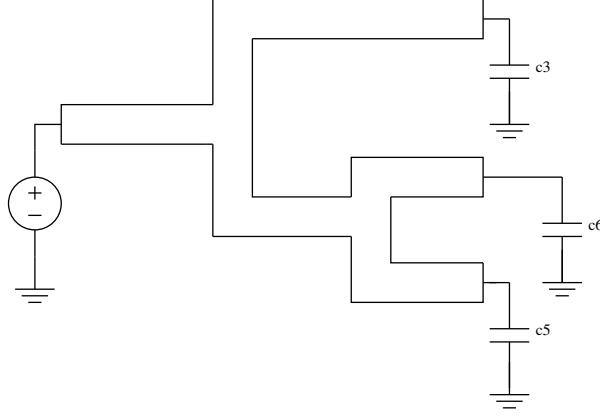
$$Q_2^T B = 0 \iff Q_2 Q_2^T B = (I - A^\dagger A)B = 0.$$

Moreover, since Λ is invertible,

$$\begin{bmatrix} \Lambda & Q_1^T B \\ B^T Q_1 & C \end{bmatrix} \succeq 0 \iff \Lambda \succ 0, \quad C - B^T Q_1 \Lambda^{-1} Q_1^T B = C - B^T A^\dagger B \succeq 0.$$

11 Circuit design

- 11.1 Interconnect sizing.** In this problem we will size the interconnecting wires of the simple circuit shown below, with one voltage source driving three different capacitive loads C_{load1} , C_{load2} , and C_{load3} .



We divide the wires into 6 segments of fixed length l_i ; our variables will be the widths w_i of the segments. (The height of the wires is related to the particular IC technology process, and is fixed.) The total area used by the wires is, of course,

$$A = \sum_i w_i l_i.$$

We'll take the lengths to be one, for simplicity. The wire widths must be between a minimum and maximum allowable value:

$$W_{\min} \leq w_i \leq W_{\max}.$$

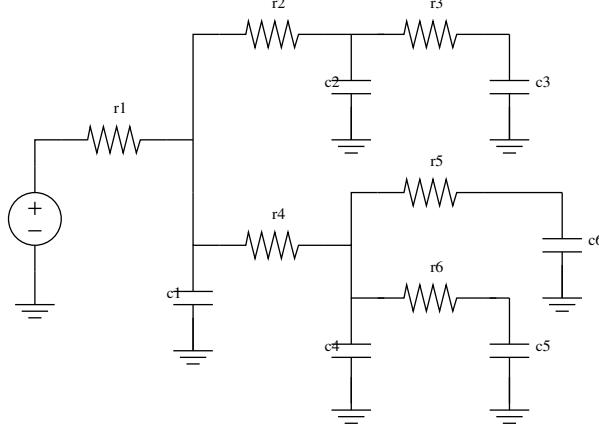
For our specific problem, we'll take $W_{\min} = 0.1$ and $W_{\max} = 10$.

Each of the wire segments will be modeled by a simple simple RC circuit, with the resistance inversely proportional to the width of the wire and the capacitance proportional to the width. (A far better model uses an extra constant term in the capacitance, but this complicates the equations.) The capacitance and resistance of the i th segment is thus

$$C_i = k_0 w_i, \quad R_i = \rho / w_i,$$

where k_0 and ρ are positive constants, which we take to be one for simplicity. We also have $C_{load1} = 1.5$, $C_{load2} = 1$, and $C_{load3} = 5$.

Using the RC model for the wire segments yields the circuit shown below.



We will use the Elmore delay to model the delay from the source to each of the loads. The Elmore delay to loads 1, 2, and 3 are given by

$$\begin{aligned}
 T_1 &= (C_3 + C_{load1})(R_1 + R_2 + R_3) + C_2(R_1 + R_2) + \\
 &\quad + (C_1 + C_4 + C_5 + C_6 + C_{load2} + C_{load3})R_1 \\
 T_2 &= (C_5 + C_{load2})(R_1 + R_4 + R_5) + C_4(R_1 + R_4) + \\
 &\quad + (C_6 + C_{load3})(R_1 + R_4) + (C_1 + C_2 + C_3 + C_{load1})R_1 \\
 T_3 &= (C_6 + C_{load3})(R_1 + R_4 + R_6) + C_4(R_1 + R_4) + \\
 &\quad + (C_1 + C_2 + C_3 + C_{load1})R_1 + (C_5 + C_{load2})(R_1 + R_4).
 \end{aligned}$$

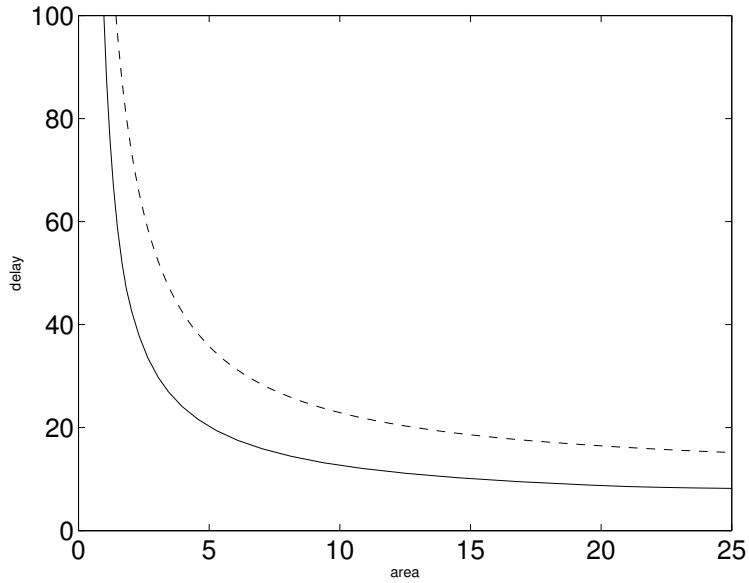
Our main interest is in the maximum of these delays,

$$T = \max\{T_1, T_2, T_3\}.$$

- (a) Explain how to find the optimal trade-off curve between area A and delay T .
- (b) *Optimal area-delay sizing.* For the specific problem parameters given, plot the area-delay trade-off curve, together with the individual Elmore delays. Comment on the results you obtain.
- (c) *The simple method.* Plot the area-delay trade-off obtained when you assign all wire widths to be the same width (which varies between W_{min} and W_{max}). Compare this curve to the optimal one, obtained in part (b). How much better does the optimal method do than the simple method? *Note:* for a large circuit, say with 1000 wires to size, the difference is *far larger*.

For this problem you can use the CVX in GP mode. We've also made available the function `elm_del_example.m`, which evaluates the three delays, given the widths of the wires.

Solution. The dashed line in the figure is the answer for part 1. The solid line is the optimal trade-off curve for part 2.



The figure was generated using the following matlab code.

```

Cload = [ 1.5; 1; 5];
maxw = 10;
minw = 0.1;

% Optimal trade-off.
N = 50;
mus = logspace(-3,3,N);
areas = zeros(1,N);
delays = zeros(1,N);
for i=1:N
    cvx_begin gp
        variables T w(6)
        C = w;
        R = 1./w;
        minimize( sum(w) + mus(i)*T )
        subject to
            w / maxw <= 1.0;
            minw ./ w <= 1;
            (C(3) + Cload(1)) * sum(R([1,2,3])) ...
                + C(2) * sum(R([1,2])) ...
                + (sum(C([1,4,5,6])) + sum(Cload([2,3]))) * R(1) <= T;
            (C(5) + Cload(2)) * sum(R([1,4,5])) ...
                + C(4) * sum(R([1,4])) ...
                + (C(6) + Cload(3)) * sum(R([1,4])) ...
                + (sum(C([1,2,3])) + Cload(1)) * R(1) <= T;
            (C(6) + Cload(3)) * sum(R([1,4,6])) ...

```

```

+ C(4) * sum(R([1,4])) ...
+ sum(C([1,2,3]) + Cload(1)) * R(1) ...
+ (C(5) + Cload(2)) * sum(R([1,4])) <= T;
cvx_end
delays(i) = T;
areas(i) = sum(w);
end;

% Equal wire widths
N=100;
ws = logspace(-1,1,N);
areas2 = zeros(1,N);
delays2 = zeros(1,N);
for i=1:length(ws)
    w = ws(i)*ones(6,1);
    areas2(i) = sum(w);
    C = w;
    R = 1./w;
    T1 = (C(3) + Cload(1)) * sum(R([1,2,3])) ...
        + C(2) * sum(R([1,2])) ...
        + (sum(C([1,4,5,6])) + sum(Cload([2,3]))) * R(1);
    T2 = (C(5) + Cload(2)) * sum(R([1,4,5])) ...
        + C(4) * sum(R([1,4])) ...
        + (C(6) + Cload(3)) * sum(R([1,4])) ...
        + (sum(C([1,2,3])) + Cload(1)) * R(1);
    T3 = (C(6) + Cload(3)) * sum(R([1,4,6])) ...
        + C(4) * sum(R([1,4])) ...
        + sum(C([1,2,3]) + Cload(1)) * R(1) ...
        + (C(5) + Cload(2)) * sum(R([1,4]));
    delays2(i) = max([T1, T2, T3]);
end;

plot(areas,delays,'-', areas2,delays2,'--');

```

XXX old solution XXX. To find the optimal trade-off curve between area and delay, we can minimize over a weighted average of area and delay. In particular the problem becomes:

$$\begin{aligned} & \text{minimize} && \max_{i=1,2,3}(T_i) + \mu A \\ & \text{subject to} && W_{\min} \leq w_i \leq W_{\max}, \quad i = 1, \dots, 6 \end{aligned}$$

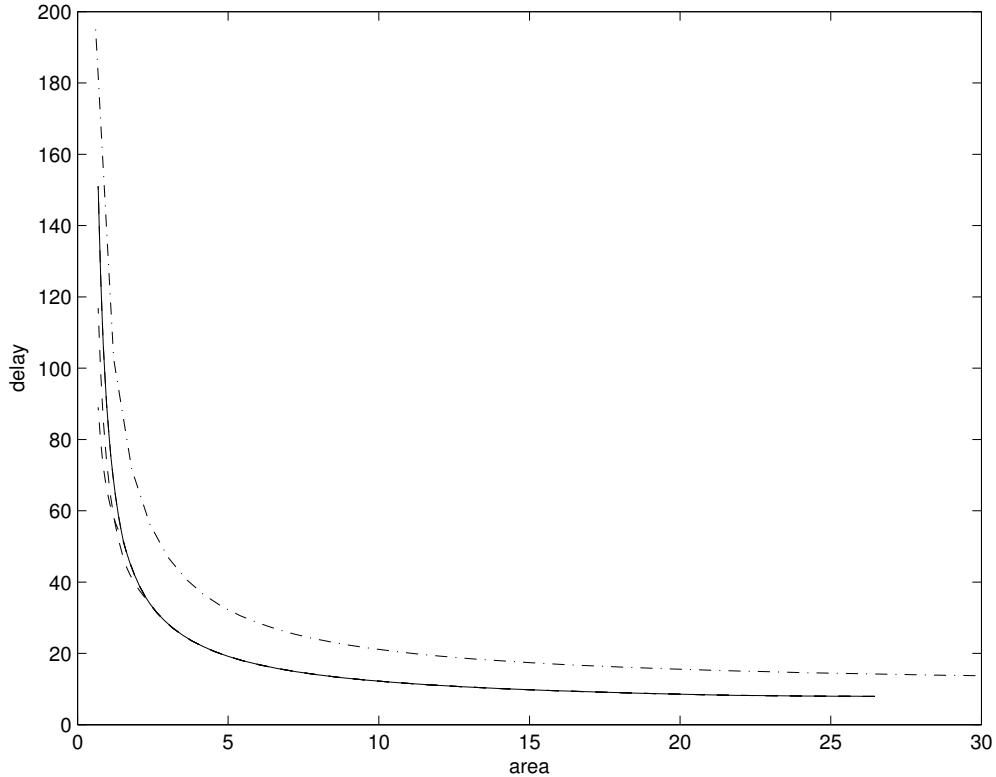
For each value of $\mu \geq 0$, we find a new point on the optimal trade-off curve and μ represents the slope of the curve at that point. We can introduce a new variable t which maximizes the three delays and by doing so we obtain an equivalent problem:

$$\begin{aligned} & \text{minimize} && t + \mu A \\ & \text{subject to} && T_i \leq t, \quad i = 1, 2, 3 \\ & && W_{\min} \leq w_i \leq W_{\max}, \quad i = 1, \dots, 6 \end{aligned}$$

Since the expressions of T_i and the area are a posynomial in the variables w_i we can formulate this minimization problem as a geometric programming with variables w_i and t .

$$\begin{aligned} & \text{minimize} && t + \mu A \\ & \text{subject to} && t^{-1} T_i \leq 1, \quad i = 1, 2, 3 \\ & && W_{\min} w_i^{-1} \leq 1, \quad i = 1, \dots, 6 \\ & && W_{\max} w_i \leq 1, \quad i = 1, \dots, 6 \end{aligned}$$

We then use the geometric programming solver provided on the web to solve this minimization problem . The matlab code used is reported at the end of the solution.



In the figure above, we can see the optimal trade-off curve (solid line) together with the individual delays (dashed lines). Moreover the dash-dot line is the simple design area delay trade-off. As we expected the simple design is always worst than the optimal one. In particular the difference between the two designs is larger when the area is smaller.

We can also see that the three delays are equal for a large part of the trade-off curve. This is what we expect, because if one of the delays is less than the others, it means that we could reallocate the area to have a higher delay on that path but lower delays on the other two. When the delays are not equal it means that the constraints on the width dimensions are active and are forcing the design to have different delays.

```
clear all
close all
```

```

%We specify the constants of this problem
ro=1;
k=1;
L=1;
cload1=1.5;
cload2=1;
cload3=5;
maxw=10;
minw=0.1;
%A1 and B1 describe T1/t
A1=[ -1 -1 0 0 0 0 0
      -1 0 -1 0 0 0 0
      -1 0 0 -1 0 0 0
      -1 0 0 0 0 0 0
      -1 -1 1 0 0 0 0
      -1 -1 0 1 0 0 0
      -1 -1 0 0 1 0 0
      -1 -1 0 0 0 1 0
      -1 -1 0 0 0 0 1
      -1 0 -1 1 0 0 0
];
B1=[log((cload1+cload2+cload3)*ro)
    log(cload1*ro)
    log(cload1*ro)
    log(+3*k*ro)
    log(ro*k)
    log(ro*k)
    log(ro*k)
    log(ro*k)
    log(ro*k)];
%A2 and B2 describe T2/t
A2=[ -1 -1 0 0 0 0 0
      -1 0 0 0 -1 0 0
      -1 0 0 0 0 -1 0
      -1 0 0 0 0 0 0
      -1 -1 1 0 0 0 0
      -1 -1 0 1 0 0 0
      -1 -1 0 0 1 0 0
      -1 -1 0 0 0 1 0
      -1 -1 0 0 0 0 1
      -1 0 0 0 -1 1 0
      -1 0 0 0 -1 0 1
];
B2=[log((cload1+cload2+cload3)*ro)

```

```

log((cload2+cload3)*ro)
log(cload2*ro)
log(+3*k*ro)
log(ro*k)
log(ro*k)
log(ro*k)
log(ro*k)
log(ro*k)
log(ro*k)
log(ro*k);
%A2 and B2 describe T2/t
A3= [-1 -1 0 0 0 0 0
      -1 0 0 0 -1 0 0
      -1 0 0 0 0 0 -1
      -1 0 0 0 0 0 0
      -1 -1 1 0 0 0 0
      -1 -1 0 1 0 0 0
      -1 -1 0 0 1 0 0
      -1 -1 0 0 0 1 0
      -1 -1 0 0 0 0 1
      -1 0 0 0 -1 1 0
      -1 0 0 0 -1 0 1
];
B3=[log((cload1+cload2+cload3)*ro)
    log((cload2+cload3)*ro)
    log(cload3*ro)
    log(+3*k*ro)
    log(ro*k)
    log(ro*k)
    log(ro*k)
    log(ro*k)
    log(ro*k)
    log(ro*k)];
%A4 and B4 are the size constraints on w_i
A4= [zeros(6,1) -eye(6)
      zeros(6,1) eye(6)];
B4= [ones(6,1)*log(minw)
      -ones(6,1)*log(maxw)];

%A0 and B0 describre the wheigted average of area and maximum delay
% For each value of mu we find a new point on the optimal trade-off curve
A0=eye(7);
mu=0.02;
for i=1:55,

```

```

B0=[0
log(mu*L)*ones(6,1)] ;

A=[A0' A1' A2' A3' A4']';
B=[B0' B1' B2' B3' B4']';
szs=[7 10 11 11 1 1 1 1 1 1 1 1 1 1]';

%initial feasible point
x0=[100 1 1 1 1 1 1]';

[x,nu,lambda] = gp(A,B,szs,x0);
o=exp(x);
tmax(i)=o(1);
W=o(2:7);
[t1(i),t2(i),t3(i)]=elm_del_example(W);
Area(i)=ones(1,6)*W*L;
mu=mu*1.2;
end

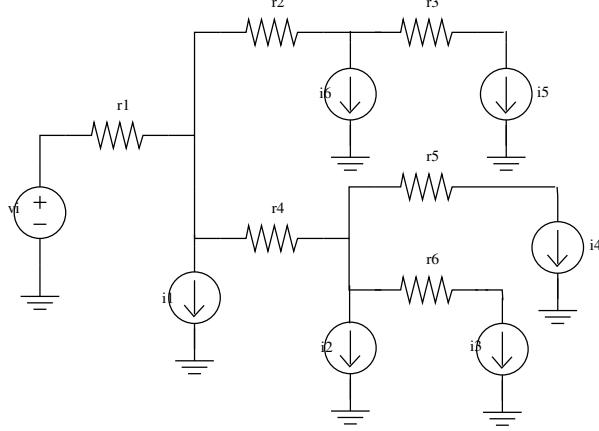
%we calculate the area versus delay for the suboptimal design
i=1;
for ws=0.1:0.1:5
    [subt1(i),subt2(i),subt3(i)]=elm_del_example(ones(6,1)*ws);
    subt(i)=max([subt1(i),subt2(i),subt3(i)]);
    Areasub(i)=L*ws*6;
    i=i+1;
end

%we plot the results together
plot(Area,tmax);
hold on
plot(Area,t1,'--');
plot(Area,t2,'--');
plot(Area,t3,'--');
plot(Areasub,subt,'-.');
xlabel('area')
ylabel('delay')

```

11.2 Optimal sizing of power and ground trees. We consider a system or VLSI device with many subsystems or subcircuits, each of which needs one or more power supply voltages. In this problem we consider the case where the power supply network has a tree topology with the power supply (or external pin connection) at the root. Each node of the tree is connected to some subcircuit that draws power.

We model the power supply as a constant voltage source with value V . The m subcircuits are modeled as current sources that draw currents $i_1(t), \dots, i_m(t)$ from the node (to ground) (see the figure below).



The subcircuit current draws have two components:

$$i_k(t) = i_k^{\text{dc}} + i_k^{\text{ac}}(t)$$

where i_k^{dc} is the DC current draw (which is a positive constant), and $i_k^{\text{ac}}(t)$ is the AC draw (which has zero average value). We characterize the AC current draw by its RMS value, defined as

$$\text{RMS}(i_k^{\text{ac}}) = \left(\lim_{T \rightarrow \infty} \frac{1}{T} \int_0^T i_k^{\text{ac}}(t)^2 dt \right)^{1/2}.$$

For each subcircuit we are given maximum values for the DC and RMS AC currents draws, *i.e.*, constants I_k^{dc} and I_k^{ac} such that

$$0 \leq i_k^{\text{dc}} \leq I_k^{\text{dc}}, \quad \text{RMS}(i_k^{\text{ac}}) \leq I_k^{\text{ac}}. \quad (41)$$

The n wires that form the distribution network are modeled as resistors R_k (which, presumably, have small value). (Since the circuit has a tree topology, we can use the following labeling convention: node k and the current source $i_k(t)$ are immediately following resistor R_k .) The resistance of the wires is given by

$$R_i = \alpha l_i / w_i,$$

where α is a constant and l_i are the lengths of the wires, which are known and fixed. The variables in the problem are the width of the wires, w_1, \dots, w_n . Obviously by making the wires very wide, the resistances become very low, and we have a nearly ideal power network. The purpose of this problem is to optimally select wire widths, to minimize area while meeting certain specifications. Note that in this problem we ignore dynamics, *i.e.*, we do not model the capacitance or inductance of the wires.

As a result of the current draws and the nonzero resistance of the wires, the voltage at node k (which supplies subcircuit k) has a DC value less than the supply voltage, and also an AC voltage (which is called power supply ripple or noise). By superposition these two effects can be analyzed separately.

- The DC voltage drop $V - v_k^{\text{dc}}$ at node k is equal to the sum of the voltage drops across wires on the (unique) path from node k to the root. It can be expressed as

$$V - v_k^{\text{dc}} = \sum_{j=1}^m i_j^{\text{dc}} \sum_{i \in \mathcal{N}(j,k)} R_i, \quad (42)$$

where $\mathcal{N}(j,k)$ consists of the indices of the branches upstream from nodes j and k , *i.e.*, $i \in \mathcal{N}(j,k)$ if and only if R_i is in the path from node j to the root and in the path from node k to the root.

- The power supply noise at a node can be found as follows. The AC voltage at node k is equal to

$$v_k^{\text{ac}}(t) = - \sum_{j=1}^m i_j^{\text{ac}}(t) \sum_{i \in \mathcal{N}(j,k)} R_i.$$

We assume the AC current draws are independent, so the RMS value of $v_k^{\text{ac}}(t)$ is given by the squareroot of the sum of the squares of the RMS value of the ripple due to each other node, *i.e.*,

$$\text{RMS}(v_k^{\text{ac}}) = \left(\sum_{j=1}^m \left(\text{RMS}(i_j^{\text{ac}}) \sum_{i \in \mathcal{N}(j,k)} R_i \right)^2 \right)^{1/2}. \quad (43)$$

The problem is to choose wire widths w_i that minimize the total wire area $\sum_{i=k}^n w_k l_k$ subject to the following specifications:

- maximum allowable DC voltage drop at each node:

$$V - v_k^{\text{dc}} \leq V_{\max}^{\text{dc}}, \quad k = 1, \dots, m, \quad (44)$$

where $V - v_k^{\text{dc}}$ is given by (42), and V_{\max}^{dc} is a given constant.

- maximum allowable power supply noise at each node:

$$\text{RMS}(v_k^{\text{ac}}) \leq V_{\max}^{\text{ac}}, \quad k = 1, \dots, m, \quad (45)$$

where $\text{RMS}(v_k^{\text{ac}})$ is given by (43), and V_{\max}^{ac} is a given constant.

- upper and lower bounds on wire widths:

$$w_{\min} \leq w_i \leq w_{\max}, \quad i = 1, \dots, n, \quad (46)$$

where w_{\min} and w_{\max} are given constants.

- maximum allowable DC current density in a wire:

$$\left(\sum_{j \in \mathcal{M}(k)} i_j^{\text{dc}} \right) / w_k \leq \rho_{\max}, \quad k = 1, \dots, n, \quad (47)$$

where $\mathcal{M}(k)$ is the set of all indices of nodes downstream from resistor k , *i.e.*, $j \in \mathcal{M}(k)$ if and only if R_k is in the path from node j to the root, and ρ_{\max} is a given constant.

- maximum allowable total DC power dissipation in supply network:

$$\sum_{k=1}^n R_k \left(\sum_{j \in \mathcal{M}(k)} i_j^{\text{dc}} \right)^2 \leq P_{\max}, \quad (48)$$

where P_{\max} is a given constant.

These specifications must be satisfied for all possible $i_k(t)$ that satisfy (41).

Formulate this as a convex optimization problem in the standard form

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

You may introduce new variables, or use a change of variables, but you must say very clearly

- what the optimization variable x is, and how it corresponds to the problem variables w (*i.e.*, is x equal to w , does it include auxiliary variables, ...?)
- what the objective f_0 and the constraint functions f_i are, and how they relate to the objectives and specifications of the problem description
- why the objective and constraint functions are convex
- what A and b are (if applicable).

Solution. There are at least three correct solutions.

- (a) Use the inverse widths as variables, *i.e.*, define $y_i = 1/w_i$ and solve the problem

$$\begin{aligned} & \text{minimize} && \sum_{i=1}^n l_i/y_i \\ & \text{subject to} && \sum_{j=1}^m I_j^{\text{dc}} \sum_{i \in \mathcal{N}(j,k)} \alpha l_i y_i \leq V_{\max}^{\text{dc}}, \quad k = 1, \dots, m \\ & && \left(\sum_{j=1}^m \left(I_j^{\text{ac}} \sum_{k \in \mathcal{N}(i,j)} \alpha l_k y_k \right)^2 \right)^{1/2} \leq V_{\max}^{\text{ac}}, \quad k = 1, \dots, m \\ & && 1/w_{\max} \leq y_i \leq 1/w_{\min}, \quad i = 1, \dots, n \\ & && \left(\sum_{j \in \mathcal{M}(k)} I_j^{\text{dc}} \right) y_k \leq \rho_{\max}, \quad k = 1, \dots, n \\ & && \sum_{k=1}^n \alpha l_i y_i \left(\sum_{j \in \mathcal{M}(k)} I_j^{\text{dc}} \right)^2 \leq P_{\max}. \end{aligned}$$

The objective is a convex function of y_i since the lengths l_i are positive constants. All the constraints except the second are linear inequalities in y . The second constraint is a second-order cone constraint.

(b) Express the problem as a geometric program in w :

$$\begin{aligned}
& \text{minimize} && \sum_{i=1}^n l_i w_i \\
& \text{subject to} && \sum_{j=1}^m \sum_{i \in \mathcal{N}(j,k)} (V_{\max}^{\text{dc}})^{-1} I_j^{\text{dc}} \alpha l_i w_i^{-1} \leq 1, \quad k = 1, \dots, m \\
& && \left(\sum_{j=1}^m \sum_{k \in \mathcal{N}(i,j)} (V_{\max}^{\text{ac}})^{-1} I_j^{\text{ac}} \alpha l_k w_k^{-1} \right)^2 \leq 1, \quad k = 1, \dots, m \\
& && w_i w_{\max}^{-1} \leq 1, \quad i = 1, \dots, n \\
& && w_{\min} w_i^{-1} \leq 1, \quad i = 1, \dots, n \\
& && \rho_{\max}^{-1} \left(\sum_{j \in \mathcal{M}(k)} I_j^{\text{dc}} \right) w_k^{-1} \leq 1, \quad k = 1, \dots, n \\
& && \sum_{k=1}^n P_{\max}^{-1} \alpha l_i w_i^{-1} \left(\sum_{j \in \mathcal{M}(k)} I_j^{\text{dc}} \right)^2 \leq 1.
\end{aligned}$$

(Note that we have squared both sides of the second constraint.) The objective function and all constraint functions are posynomial in w , so we have a geometric program. Writing the geometric program in the exponential form yields a convex optimization problem.

(c) In fact the problem is also convex in the original variables w :

$$\begin{aligned}
& \text{minimize} && \sum_{i=1}^n l_i w_i \\
& \text{subject to} && \sum_{j=1}^m I_j^{\text{dc}} \sum_{i \in \mathcal{N}(j,k)} \alpha l_i / w_i \leq V_{\max}^{\text{dc}}, \quad k = 1, \dots, m \\
& && \left(\sum_{j=1}^m \left(I_j^{\text{ac}} \sum_{k \in \mathcal{N}(i,j)} \alpha l_k / w_k \right)^2 \right)^{1/2} \leq V_{\max}^{\text{ac}}, \quad k = 1, \dots, m \\
& && w_{\min} \leq w_i \leq w_{\max}, \quad i = 1, \dots, n \\
& && \left(\sum_{j \in \mathcal{M}(k)} I_j^{\text{dc}} \right) / w_k \leq \rho_{\max}, \quad k = 1, \dots, n \\
& && \sum_{k=1}^n \alpha l_i / w_i \left(\sum_{j \in \mathcal{M}(k)} I_j^{\text{dc}} \right)^2 \leq P_{\max}.
\end{aligned}$$

The objective function is linear in w . The left hand side of the first constraint is a convex function of w , since $w \succeq 0$, and all the constants in the expression are positive. To show that the third constraint is convex, we can use one of the composition theorems. We know that the function

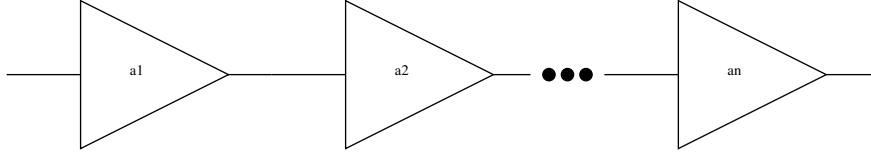
$$f(z_1, \dots, z_m) = \left(\sum_{i=1}^m z_i^2 \right)^{1/2}$$

is convex. When restricted to the set $\{z \mid z \succeq 0\}$, it is also nondecreasing in each argument. The function on the left hand side of the constraint is the composition of f with a nonnegative convex function

$$z_i = g_i(w) = I_j^{\text{ac}} \sum_{k \in \mathcal{N}(i,j)} \alpha l_k / w_k.$$

The third set of constraints are linear inequalities. The last two constraints are convex because $1/w_i$ is convex on $w_i > 0$, and all coefficients in the expressions are positive.

11.3 Optimal amplifier gains. We consider a system of n amplifiers connected (for simplicity) in a chain, as shown below. The variables that we will optimize over are the gains $a_1, \dots, a_n > 0$ of the amplifiers. The first specification is that the overall gain of the system, *i.e.*, the product $a_1 \cdots a_n$, is equal to A^{tot} , which is given.



We are concerned about two effects: noise generated by the amplifiers, and amplifier overload. These effects are modeled as follows.

We first describe how the noise depends on the amplifier gains. Let N_i denote the noise level (RMS, or root-mean-square) at the output of the i th amplifier. These are given recursively as

$$N_0 = 0, \quad N_i = a_i \left(N_{i-1}^2 + \alpha_i^2 \right)^{1/2}, \quad i = 1, \dots, n$$

where $\alpha_i > 0$ (which is given) is the ('input-referred') RMS noise level of the i th amplifier. The *output noise level* N_{out} of the system is given by $N_{\text{out}} = N_n$, *i.e.*, the noise level of the last amplifier. Evidently N_{out} depends on the gains a_1, \dots, a_n .

Now we describe the amplifier overload limits. S_i will denote the signal level at the output of the i th amplifier. These signal levels are related by

$$S_0 = S_{\text{in}}, \quad S_i = a_i S_{i-1}, \quad i = 1, \dots, n,$$

where $S_{\text{in}} > 0$ is the *input signal level*. Each amplifier has a maximum allowable output level $M_i > 0$ (which is given). (If this level is exceeded the amplifier will distort the signal.) Thus we have the constraints $S_i \leq M_i$, for $i = 1, \dots, n$. (We can ignore the noise in the overload condition, since the signal levels are much larger than the noise levels.)

The *maximum output signal level* S_{max} is defined as the maximum value of S_n , over all input signal levels S_{in} that respect the the overload constraints $S_i \leq M_i$. Of course $S_{\text{max}} \leq M_n$, but it can be smaller, depending on the gains a_1, \dots, a_n .

The *dynamic range* D of the system is defined as $D = S_{\text{max}}/N_{\text{out}}$. Evidently it is a (rather complicated) function of the amplifier gains a_1, \dots, a_n .

The goal is to choose the gains a_i to maximize the dynamic range D , subject to the constraint $\prod_i a_i = A^{\text{tot}}$, and upper bounds on the amplifier gains, $a_i \leq A_i^{\text{max}}$ (which are given).

Explain how to solve this problem as a convex (or quasiconvex) optimization problem. If you introduce new variables, or transform the variables, explain. Clearly give the objective and inequality constraint functions, explaining why they are convex if it is not obvious. If your problem involves equality constraints, give them explicitly.

Carry out your method on the specific instance with $n = 4$, and data

$$\begin{aligned} A^{\text{tot}} &= 10000, \\ \alpha &= (10^{-5}, 10^{-2}, 10^{-2}, 10^{-2}), \\ M &= (0.1, 5, 10, 10), \\ A^{\max} &= (40, 40, 40, 20). \end{aligned}$$

Give the optimal gains, and the optimal dynamic range.

Solution. We have the constraints

$$a_1 \cdots a_n = A, \quad a_i \leq A_i, \quad i = 1, \dots, n.$$

You should already be getting that GP (geometric programming) feeling, given the form of these two constraints. Indeed, that is what we are going to end up with.

To get an expression for the output noise level, we first write the recursion as

$$N_0^2 = 0, \quad N_i^2 = a_i^2 (N_{i-1}^2 + \alpha_i^2), \quad i = 1, \dots, n.$$

Then we can express the square of the output noise level (*i.e.*, the output noise power) as

$$N_{\text{out}}^2 = \sum_{i=1}^n \prod_{j=i}^n a_j^2 \alpha_j^2.$$

The amplifier overload constraints can be expressed as

$$S_i = S_{\text{in}} \prod_{j=1}^i a_j \leq M_i, \quad i = 1, \dots, n.$$

Hence the maximum possible input signal level is

$$S_{\text{in,max}} = \min_i \frac{M_i}{\prod_{j=1}^i a_j}$$

Since $S_n = AS_{\text{in}}$, we can express the maximum output signal level as

$$S_{\text{max}} = AS_{\text{in,max}} = \min_i M_i \prod_{j=i+1}^n a_j$$

(we interpret \prod_{n+1}^n as 1).

Therefore the dynamic range is given by

$$D = \frac{\min_i M_i \prod_{j=i+1}^n a_j}{\left(\sum_{i=1}^n \prod_{j=i}^n a_j^2 \alpha_j^2 \right)^{1/2}}$$

(not exactly what you'd call a simple function of a_i).

To maximize D , we can minimize $1/D^2$. To do this, we introduce a new variable t , and minimize t subject to

$$\sum_{i=1}^n \prod_{j=i}^n a_j^2 \alpha_i^2 \leq t M_i^2 \prod_{j=i+1}^n a_j^2, \quad i = 1, \dots, n$$

and the constraints

$$a_1 \cdots a_n = A, \quad a_i \leq A_i, \quad i = 1, \dots, n.$$

This is a GP.

```
% optimal amplifier gains
n = 4; % 4 amplifier stages
Atot = 10000; % 80dB total gain
alpha = [1e-5 1e-2 1e-2 1e-2]'; % amplifier input-referred noise levels
M = [0.1 5 10 10]'; % amplifier overload limits
Amax = [40 40 40 20]'; % amplifier max gains

cvx_begin gp
variables a(n) S(n) Sin
prod(a) == Atot; % overall gain is A
a <= Amax; % max amplifier stage gains
% noise levels
Nsquare(1) = a(1)^2*alpha(1)^2;
for i=2:n
Nsquare(i) = a(i)^2*(Nsquare(i-1)+alpha(i)^2);
end
% signal levels
S(1) == a(1)*Sin;
for i=2:n
S(i) == a(i)*S(i-1);
end
S <= M; % signal level limits
maximize (S(n)/sqrt(Nsquare(n)))
cvx_end

opt_dyn_range = cvx_optval

opt_gains = a

signal_levels_and_limits = [S M]
```

11.4 Blending existing circuit designs. In circuit design, we must select the widths of a set of n components, given by the vector $w = (w_1, \dots, w_n)$, which must satisfy width limits

$$W^{\min} \leq w_i \leq W^{\max}, \quad i = 1, \dots, n,$$

where W^{\min} and W^{\max} are given (positive) values. (You can assume there are no other constraints on w .) The design is judged by three objectives, each of which we would like to be small: the circuit power $P(w)$, the circuit delay $D(w)$, and the total circuit area $A(w)$. These three objectives are (complicated) posynomial functions of w .

You *do not know* the functions P , D , or A . (That is, you do not know the coefficients or exponents in the posynomial expressions.) You *do know* a set of k designs, given by $w^{(1)}, \dots, w^{(k)} \in \mathbf{R}^n$, and their associated objective values

$$P(w^{(j)}), \quad D(w^{(j)}), \quad A(w^{(j)}), \quad j = 1, \dots, k.$$

You can assume that these designs satisfy the width limits. The goal is to find a design w that satisfies the width limits, and the design specifications

$$P(w) \leq P_{\text{spec}}, \quad D(w) \leq D_{\text{spec}}, \quad A(w) \leq A_{\text{spec}},$$

where P_{spec} , D_{spec} , and A_{spec} are given.

Now consider the specific data given in `blend_design_data.m`. Give the following.

- A feasible design (*i.e.*, w) that satisfies the specifications.
- A clear argument as to how you know that your design satisfies the specifications, even though you do not know the formulas for P , D , and A .
- Your method for finding w , including any code that you write.

Hints/comments.

- You do not need to know *anything* about circuit design to solve this problem.
- See the title of this problem.

Solution. We're given very little to go on in this problem! Clearly, the solution must rely on some basic properties of posynomials, since we are not even told what the posynomials P , D , and A are, other than their values at the designs $w^{(1)}, \dots, w^{(k)}$.

First ideas and attempt. The title of this course, and the title of the problem, suggests that you should consider blended designs, which have the form

$$w = \theta_1 w^{(1)} + \cdots + \theta_k w^{(k)},$$

where $\theta \succeq 0$, $\mathbf{1}^T \theta = 1$. When the function f is convex, we can say that

$$f(w) \leq \theta_1 f(w^{(1)}) + \cdots + \theta_k f(w^{(k)}),$$

by Jensen's inequality. We're getting close, since we can now make a statement that our design has an f -value less than some other number that depends on the f -values at our given designs, and our blending parameters. But no cigar, since P , D , and A are posynomials, and therefore not known to be convex.

An attempt that works. A little reflection leads us in the right direction. When f is a posynomial function, the function g defined by

$$g(x) = \log f(\exp x)$$

is convex, where $\exp x$ is meant elementwise. Therefore the functions $\log P$, $\log D$, and $\log A$ are convex functions of the variables $x = \log w$. And this means, for example, that

$$\log P(\exp x) \leq \theta_1 \log P(\exp x^{(1)}) + \cdots + \theta_k \log P(\exp x^{(k)}),$$

whenever $\theta \succeq 0$, $\mathbf{1}^T \theta = 1$, and $x^{(j)} = \log w^{(j)}$. Letting $P^{(j)}$ denote $P(w^{(j)})$, and defining the blended design w as

$$\log w = \theta_1 \log w^{(1)} + \cdots + \theta_k \log w^{(k)},$$

we write the inequality above as

$$\log P(w) \leq \theta_1 \log P^{(1)} + \cdots + \theta_k \log P^{(k)}.$$

We've now got a method to form a blended design, for which we can predict an upper bound on how large the power can be. It's just linear interpolation on a log scale; equivalently, the geometric interpolation

$$w_i = (w_i^{(1)})^{\theta_1} \cdots (w_i^{(k)})^{\theta_k}.$$

Note that w will satisfy the width limits, since the original designs do.

Of course similar inequalities hold for the delay and area. We can try to find a blend that meets the specifications by solving the LP feasibility problem

$$\begin{array}{ll} \text{find} & \theta \\ \text{subject to} & \sum_{j=1}^k \theta_j \log P^{(j)} \leq \log P_{\text{spec}} \\ & \sum_{j=1}^k \theta_j \log D^{(j)} \leq \log D_{\text{spec}} \\ & \sum_{j=1}^k \theta_j \log A^{(j)} \leq \log A_{\text{spec}} \\ & \mathbf{1}^T \theta = 1, \quad \theta \succeq 0, \end{array}$$

with variable θ (the blending parameter).

It's important to understand exactly what we have done here. If the LP above is feasible, then the blend obtained from the feasible θ (and done geometrically, as described above) *must satisfy the design specifications*. We can make this statement *without knowing the specific posynomial expressions for P, D, and A*. It's nothing more than Jensen's inequality, but even so, it's pretty interesting.

On the other hand, if the LP above is infeasible, then we can say nothing. The design specifications could be infeasible, or feasible; we do not know.

The following code attempts to find a feasible blending parameter in Matlab.

```
% solution for circuit blending problem
blend_design_data;
% find feasible blend factors
cvx_begin
```

```

variable theta(k)
log(P)*theta <= log(P_spec);
log(D)*theta <= log(D_spec);
log(A)*theta <= log(A_spec);
sum(theta) == 1;
theta >= 0;
cvx_end
% now create design
w = exp(log(W)*theta)

```

Here is a Python version.

```

import numpy as np
import cvxpy as cvx
from blend_design_data import *

theta = cvx.Variable(k)
constraints = [np.log(P)*theta <= np.log(P_spec)]
constraints += [np.log(D)*theta <= np.log(D_spec)]
constraints += [np.log(A)*theta <= np.log(A_spec)]
constraints += [cvx.sum_entries(theta)==1, theta>=0]

cvx.Problem(cvx.Minimize(0),constraints).solve()

# now create design
w = np.exp(np.log(W)*theta.value)
print(w)
print(theta.value)

```

And finally, Julia.

```

# solution for circuit blending problem
include("blend_design_data.jl");
# find feasible blend factors

using Convex
theta = Variable(k);
constraints = [
    log(P)*theta <= log(P_spec);
    log(D)*theta <= log(D_spec);
    log(A)*theta <= log(A_spec);
    sum(theta) == 1;
    theta >= 0;
];
p = satisfy(constraints);
solve!(p);

```

```
# now create design
w = exp(log(W)*theta.value);
```

It turns out it's feasible, so we've solved the problem! So indeed there exists a design which satisfies the specifications. Our blending parameter (from Matlab) is

$$\theta = (0.0128, 0.5096, 0.0050, 0.4626, 0.0075, 0.0026),$$

resulting in widths

$$w = (2.6203, 3.3033, 2.9562, 3.2743, 2.3037, 3.6820, 2.9177, 3.7012, 3.9140, 3.4115).$$

This solution is not unique, so you might be interested to know how we graded the numerical answers. (Of course, the main point was not to get the right answer, but to realize that you must do the blending in log space, and to clearly explain why you could be certain that your design meets the specifications.)

To grade the numerical answers, we found the bounding box (*i.e.*, smallest and largest possible value of each width) over the inequalities above. The bounds were:

$$\begin{aligned}\underline{w} &= (2.5064, 3.1177, 2.8817, 3.2293, 2.2109, 3.5823, 2.8526, 3.5293, 3.7406, 3.3164) \\ \overline{w} &= (2.8151, 3.4431, 3.0924, 3.3212, 2.4757, 3.7381, 3.0056, 3.8082, 4.0065, 3.4695).\end{aligned}$$

Any number outside these ranges is wrong. (However, numbers inside the ranges need not be correct.)

An approach that almost works. We mention another approach which is correct, but fails to solve this particular problem instance. The functions $P(\exp x)$, $D(\exp x)$, and $A(\exp x)$ are convex, where $x = \log w$. (This follows immediately from the fact that their logs are convex, and the exponential of a convex function is convex.) Using the notation above, we have that the inequality

$$P(w) \leq \theta_1 P^{(1)} + \cdots + \theta_k P^{(k)}$$

holds. It follows that if the feasibility LP

$$\begin{array}{ll}\text{find} & \theta \\ \text{subject to} & \sum_{j=1}^k \theta_j P^{(j)} \leq P_{\text{spec}} \\ & \sum_{j=1}^k \theta_j D^{(j)} \leq D_{\text{spec}} \\ & \sum_{j=1}^k \theta_j A^{(j)} \leq A_{\text{spec}} \\ & \mathbf{1}^T \theta = 1, \quad \theta \succeq 0,\end{array}$$

is feasible, then we've found a set of blending parameters that must achieve the given specs. The logic here is absolutely correct. Note that the blending parameters here are the same as above—they correspond to geometric or logarithmic blending, and not arithmetic blending. This LP above is more stringent than the first one given above; that is, it has a smaller feasible set, which means it will work in fewer cases than the one given above.

For the given example, this LP is not feasible, so for this particular problem instance, this approach won't work. (You might ask: Did the teaching staff intentionally arrange for the given problem instance to fail using this approach? Would they really do something like that?)

11.5 Solving nonlinear circuit equations using convex optimization. An electrical circuit consists of b two-terminal devices (or branches) connected to n nodes, plus a so-called ground node. The goal is to compute several sets of physical quantities that characterize the circuit operation. The vector of *branch voltages* is $v \in \mathbf{R}^b$, where v_j is the voltage appearing across device j . The vector of *branch currents* is $i \in \mathbf{R}^b$, where i_j is the current flowing through device j . (The symbol i , which is often used to denote an index, is unfortunately the standard symbol used to denote current.) The vector of *node potentials* is $e \in \mathbf{R}^n$, where e_k is the potential of node k with respect to the ground node. (The ground node has potential zero by definition.)

The circuit variables v , i , and e satisfy several physical laws. Kirchhoff's current law (KCL) can be expressed as $Ai = 0$, and Kirchhoff's voltage law (KVL) can be expressed as $v = A^T e$, where $A \in \mathbf{R}^{n \times b}$ is the reduced incidence matrix, which describes the circuit topology:

$$A_{kj} = \begin{cases} -1 & \text{branch } j \text{ enters node } k \\ +1 & \text{branch } j \text{ leaves node } k \\ 0 & \text{otherwise,} \end{cases}$$

for $k = 1, \dots, n$, $j = 1, \dots, b$. (KCL states that current is conserved at each node, and KVL states that the voltage across each branch is the difference of the potentials of the nodes it is connected to.)

The branch voltages and currents are related by

$$v_j = \phi_j(i_j), \quad j = 1, \dots, b,$$

where ϕ_j is a given function that depends on the *type* of device j . We will assume that these functions are continuous and nondecreasing. We give a few examples. If device j is a resistor with resistance $R_j > 0$, we have $\phi_j(i_j) = R_j i_j$ (which is called Ohm's law). If device j is a voltage source with voltage V_j and internal resistance $r_j > 0$, we have $\phi_j(i_j) = V_j + r_j i_j$. And for a more interesting example, if device j is a diode, we have $\phi_j(i_j) = V_T \log(1 + i_j/I_S)$, where I_S and V_T are known positive constants.

- (a) Find a method to solve the circuit equations, *i.e.*, find v , i , and e that satisfy KCL, KVL, and the branch equations, that relies on convex optimization. State the optimization problem clearly, indicating what the variables are. Be sure to explain how solving the convex optimization problem you propose leads to choices of the circuit variables that satisfy all of the circuit equations. You can assume that no pathologies occur in the problem that you propose, for example, it is feasible, a suitable constraint qualification holds, and so on.

Hint. You might find the function $\psi : \mathbf{R}^b \rightarrow \mathbf{R}$,

$$\psi(i_1, \dots, i_b) = \sum_{j=1}^b \int_0^{i_j} \phi_j(u_j) du_j,$$

useful.

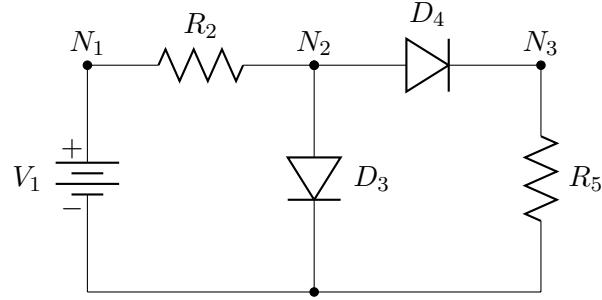
- (b) Consider the circuit shown in the diagram below. Device 1 is a voltage source with parameters $V_1 = 1000$, $r_1 = 1$. Devices 2 and 5 are resistors with resistance $R_2 = 1000$, and $R_5 = 100$ respectively. Devices 3 and 4 are identical diodes with parameters $V_T = 26$, $I_S = 1$. (The units are mV, mA, and Ω .)

The nodes are labeled N_1, N_2 , and N_3 ; the ground node is at the bottom. The incidence matrix A is

$$A = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 1 & 1 & 0 \\ 0 & 0 & 0 & -1 & 1 \end{bmatrix}.$$

(The reference direction for each edge is down or to the right.)

Use the method in part (a) to compute v, i , and e . Verify that all the circuit equations hold.



Solution.

(a) We first observe that the function ψ given in the hint is convex: since ϕ_j is nondecreasing,

$$\int_0^{i_j} \phi_j(u_j) du_j$$

is convex in i_j , and ψ is the sum of these functions. We note that

$$\nabla\psi(i) = (\phi_1(i_1), \dots, \phi_b(i_b)).$$

The optimization problem to solve is

$$\begin{aligned} &\text{minimize} && \psi(i) \\ &\text{subject to} && Ai = 0, \end{aligned}$$

with variable $i \in \mathbf{R}^b$. The optimality conditions are $Ai = 0$ (which is KCL), and $\nabla\psi(i) + A^T\nu = 0$, where ν is a dual variable associated with the constraint $Ai = 0$. Defining $v = \nabla\psi(i)$ and $e = -\nu$, the optimality conditions can be expressed as

$$Ai = 0, \quad A^T e = v, \quad v_j = \phi_j(i_j), \quad j = 1, \dots, b.$$

These are *exactly* the circuit equations.

By the way, this characterization of a circuit in terms of an optimization problem was known to J. C. Maxwell. The function ψ is called the *content function* for the circuit.

(b) Let us first compute the content function for each component.

For a resistor

$$\int_0^i ru du = (1/2)ri^2.$$

For a voltage source

$$\int_0^i V + ru \, du = Vi + (1/2)ri^2.$$

For a diode

$$V_T \int_0^i \log(1 + u/I_S) \, du = V_T I_S ((1 + i/I_S) \log(1 + i/I_S) - i/I_S).$$

We add the individual constraint functions for the circuit, yielding in the optimization problem

$$\begin{aligned} \text{minimize} \quad & V_1 i_1 + (1/2) \sum_{j \in \{1,2,5\}} R_j i_j^2 + \sum_{j \in \{3,4\}} V_T I_S ((1 + i_j/I_S) \log(1 + i_j/I_S) - i_j/I_S) \\ \text{subject to} \quad & A_i = 0. \end{aligned}$$

Solving this problem in CVX* results in

$$A^T e = \begin{bmatrix} 999.017 \\ 982.966 \\ 16.051 \\ 3.154 \\ 12.897 \end{bmatrix}, \quad i = \begin{bmatrix} -0.983 \\ 0.983 \\ 0.854 \\ 0.129 \\ 0.129 \end{bmatrix}, \quad e = \begin{bmatrix} 999.017 \\ 16.051 \\ 12.897 \end{bmatrix}, \quad \phi(i) = \begin{bmatrix} 999.017 \\ 982.967 \\ 16.051 \\ 3.154 \\ 12.897 \end{bmatrix}$$

and

$$\frac{\|\phi(i) - A^T e\|_2}{\|\phi(i)\|_2} = 3.996 \cdot 10^{-7}.$$

We conclude that the solutions match.

In CVX (and POGS)

```
% Setup
A = [ 1  1  0  0  0
      0 -1  1  1  0
      0  0  0 -1  1
      -1  0 -1  0 -1];

% Remove redundant ground constraint (ie force ground potential = 0)
A = A(1:end-1, :);

R1 = 1;
R2 = 1e3;
R5 = 1e2;

VT = 26;
IS = 1;
VS = 1e3;

cvx_begin
  variable ii(5)
  dual variable e
```

```

OBJ1 = VS*ii(1) + (1/2)*R1*ii(1)^2;
OBJ2 = (1/2)*R2*ii(2)^2;
OBJ3 = VT*IS*(-entr(1 + ii(3)/IS) - ii(3)/IS);
OBJ4 = VT*IS*(-entr(1 + ii(4)/IS) - ii(4)/IS);
OBJ5 = (1/2)*R5*ii(5)^2;
minimize(OBJ1 + OBJ2 + OBJ3 + OBJ4 + OBJ5)
subject to
    e : A * ii == 0;
cvx_end

% Check Constraints
v = [VS + R1*ii(1)
      R2*ii(2)
      VT*log(1 + ii(3)/IS)
      VT*log(1 + ii(4)/IS)
      R5*ii(5)];

v_err = v - A' * e;

fprintf('Relative error in voltage: %e\n', norm(v_err) / norm(v))

%% POGS

f.h = kIndEq0;
g.h = [kSquare; kSquare; kNegEntr; kNegEntr; kSquare];
g.a = [ 1; 1; 1/IS; 1/IS; 1];
g.b = [ 0; 0; -1; -1; 0];
g.c = [R1; R2; VT*IS; VT*IS; R5];
g.d = [VS; 0; -VT; -VT; 0];

[ii, ~, l] = pogs(A, f, g);
e = -l;

% Check Constraints
v = [VS + R1*ii(1)
      R2*ii(2)
      VT*log(1 + ii(3)/IS)
      VT*log(1 + ii(4)/IS)
      R5*ii(5)];

v_err = v - A' * e;

fprintf('Relative error in voltage: %e\n', norm(v_err) / norm(v))

```

In Julia

```

# Pkg.update()
# Pkg.add("Convex")
# Pkg.add("SCS")

using Convex

# Setup
A = [ 1  1  0  0  0
      0 -1  1  1  0
      0  0  0 -1  1
     -1  0 -1  0 -1];

# Remove redundant ground constraint
A = A[1:end-1, :];

R1 = 1
R2 = 1e3
R5 = 1e2

VT = 26
IS = 1
VS = 1e3

ii = Variable(5)

OBJ1 = VS*ii[1] + (1./2)*R1*ii[1]^2
OBJ2 = (1./2)*R2*ii[2]^2
OBJ3 = VT*IS*(-entropy(1. + ii[3]/IS) - ii[3]/IS)
OBJ4 = VT*IS*(-entropy(1. + ii[4]/IS) - ii[4]/IS)
OBJ5 = (1./2)*R5*ii[5]^2

problem = minimize(OBJ1 + OBJ2 + OBJ3 + OBJ4 + OBJ5, [A * ii == 0])
solve!(problem)
ee = -problem.constraints[1].dual

v = [VS + R1*ii.value[1]
      R2*ii.value[2]
      VT*log(1 + ii.value[3]/IS)
      VT*log(1 + ii.value[4]/IS)
      R5*ii.value[5]]

v_err = v - A' * ee

@printf("Relative error in voltage: %e\n", norm(v_err) / norm(v))

```

```

    println(v)
    println(ii)

In CVXPY

from cvxpy import *
import numpy as np
import math

A = np.array([[ 1,  1,  0,  0,  0],
              [ 0, -1,  1,  1,  0],
              [ 0,  0,  0, -1,  1],
              [-1,  0, -1,  0, -1]], dtype=np.float64)
A = A[0:-1,:]

R1 = 1.
R2 = 1e3
R5 = 1e2

VT = 26
IS = 1
VS = 1e3

ii = Variable(5)

OBJ1 = VS*ii[0] + (1./2)*R1*square(ii[0])
OBJ2 = (1./2)*R2*square(ii[1])
OBJ3 = VT*IS*(-entr(1. + ii[2]/IS) - ii[2]/IS)
OBJ4 = VT*IS*(-entr(1. + ii[3]/IS) - ii[3]/IS)
OBJ5 = (1./2)*R5*square(ii[4])

obj = Minimize(OBJ1 + OBJ2 + OBJ3 + OBJ4 + OBJ5)
constr = [A * ii == 0.]
problem = Problem(obj, constr)

#problem.solve(verbose=True, solver=SCS, eps=1e-4)
problem.solve(verbose=True)

e = -problem.constraints[0].dual_value

v = np.array([[VS + R1*float(ii.value[0])],
              [R2*float(ii.value[1])],
              [VT*math.log(1. + float(ii.value[2])/IS)],
              [VT*math.log(1. + float(ii.value[3])/IS)],
              [R5*float(ii.value[4])]])

v_err = v - np.transpose(A) * e

```

```
rel_err = np.linalg.norm(v_err) / np.linalg.norm(v)

print "Relative error in voltage: %e\n" % rel_err

print v
print ii.value
```

12 Signal processing and communications

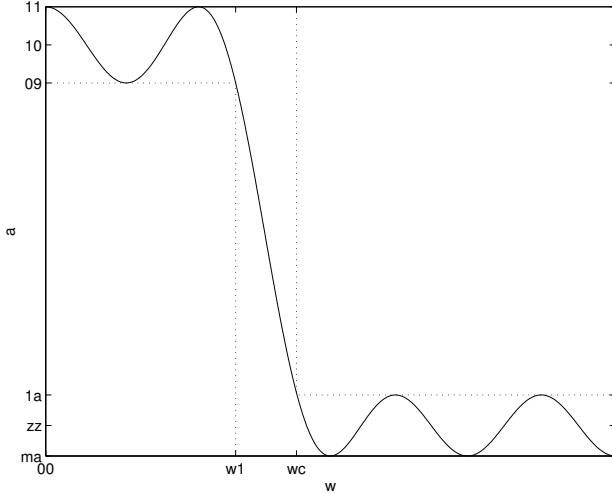
12.1 FIR low-pass filter design. Consider the (symmetric, linear phase) finite impulse response (FIR) filter described by its frequency response

$$H(\omega) = a_0 + \sum_{k=1}^N a_k \cos k\omega,$$

where $\omega \in [0, \pi]$ is the frequency. The design variables in our problems are the real coefficients $a = (a_0, \dots, a_N) \in \mathbf{R}^{N+1}$, where N is called the order or length of the FIR filter. In this problem we will explore the design of a low-pass filter, with specifications:

- For $0 \leq \omega \leq \pi/3$, $0.89 \leq H(\omega) \leq 1.12$, i.e., the filter has about $\pm 1\text{dB}$ ripple in the ‘passband’ $[0, \pi/3]$.
- For $\omega_c \leq \omega \leq \pi$, $|H(\omega)| \leq \alpha$. In other words, the filter achieves an attenuation given by α in the ‘stopband’ $[\omega_c, \pi]$. Here ω_c is called the filter ‘cutoff frequency’.

(It is called a low-pass filter since low frequencies are allowed to pass, but frequencies above the cutoff frequency are attenuated.) These specifications are depicted graphically in the figure below.



For parts (a)–(c), explain how to formulate the given problem as a convex or quasiconvex optimization problem.

- Maximum stopband attenuation.* We fix ω_c and N , and wish to maximize the stopband attenuation, i.e., minimize α .
- Minimum transition band.* We fix N and α , and want to minimize ω_c , i.e., we set the stopband attenuation and filter length, and wish to minimize the ‘transition’ band (between $\pi/3$ and ω_c).
- Shortest length filter.* We fix ω_c and α , and wish to find the smallest N that can meet the specifications, i.e., we seek the shortest length FIR filter that can meet the specifications.

- (d) *Numerical filter design.* Use CVX to find the shortest length filter that satisfies the filter specifications with

$$\omega_c = 0.4\pi, \quad \alpha = 0.0316.$$

(The attenuation corresponds to -30 dB.) For this subproblem, you may sample the constraints in frequency, which means the following. Choose K large (say, 500; an old rule of thumb is that K should be at least $15N$), and set $\omega_k = k\pi/K$, $k = 0, \dots, K$. Then replace the specifications with

- For k with $0 \leq \omega_k \leq \pi/3$, $0.89 \leq H(\omega_k) \leq 1.12$.
- For k with $\omega_c \leq \omega_k \leq \pi$, $|H(\omega_k)| \leq \alpha$.

Plot $H(\omega)$ versus ω for your design.

Solution.

- (a) The first problem can be expressed as

$$\begin{aligned} & \text{minimize} && \alpha \\ & \text{subject to} && f_1(a) \leq 1.12 \\ & && f_2(a) \geq 0.89 \\ & && f_3(a) \leq \alpha \\ & && f_4(a) \geq -\alpha, \end{aligned}$$

where

$$\begin{aligned} f_1(a) &= \sup_{0 \leq \omega \leq \pi/3} H(\omega), & f_2(a) &= \inf_{0 \leq \omega \leq \pi/3} H(\omega), \\ f_3(a) &= \sup_{\omega_c \leq \omega \leq \pi} H(\omega), & f_4(a) &= \inf_{\omega_c \leq \omega \leq \pi} H(\omega). \end{aligned}$$

For each ω , $H(\omega)$ is a linear function of a . hence the functions f_1 and f_3 are convex, and f_2 and f_4 are concave. It follows that the problem is convex.

- (b) This problem can be expressed

$$\begin{aligned} & \text{minimize} && f_5(a) \\ & \text{subject to} && f_1(a) \leq 1.12 \\ & && f_2(a) \geq 0.89 \end{aligned}$$

where f_1 and f_2 are the same functions as above, and

$$f_5(a) = \inf\{\Omega \mid -\alpha \leq H(\omega) \leq \alpha \text{ for } \Omega \leq \omega \leq \pi\}.$$

This is a quasiconvex optimization problem in the variables a because f_1 is convex, f_2 is concave, and f_5 is quasiconvex: its sublevel sets are

$$\{a \mid f_5(a) \leq \Omega\} = \{a \mid -\alpha \leq H(\omega) \leq \alpha \text{ for } \Omega \leq \omega \leq \pi\},$$

i.e., the intersection of an infinite number of halfspaces.

(c) This problem can be expressed as

$$\begin{aligned} & \text{minimize} && f_6(a) \\ & \text{subject to} && f_1(a) \leq 1.12 \\ & && f_2(a) \geq 0.89 \\ & && f_3(a) \leq \alpha \\ & && f_4(a) \geq -\alpha \end{aligned}$$

where f_1 , f_2 , f_3 , and f_4 are defined above and

$$f_6(a) = \min\{k \mid a_{k+1} = \dots = a_N = 0\}.$$

The sublevel sets of f_6 are affine sets:

$$\{a \mid f_6(a) \leq k\} = \{a \mid a_{k+1} = \dots = a_N = 0\}.$$

This means f_6 is a quasiconvex function, and again we have a quasiconvex optimization problem.

- (d) After discretizing we can express the problem in part (c) for a given length N as the feasibility LP

$$\begin{aligned} 0.89 \leq H(\omega_k) &\leq 1.12 \text{ for } 0 \leq \omega_k \leq \pi/3, \\ -\alpha \leq H(\omega_k) &\leq \alpha \text{ for } \omega_c \leq \omega_k \leq \pi \end{aligned}$$

with variable a . (For fixed ω_k , $H(\omega_k)$ is an affine function of a , hence all constraints in this problem are linear inequalities in a .)

The following code carries out the quasi-convex optimization. We find that the shortest filter has length $N = 16$.

```
clear all;
K = 500;
wp = pi/3; wc = .4*pi; alpha = 0.0316;
w = linspace(0, pi, K);
wi = max(find(w<=wp)); wo = min(find(w>=wc));
for N = 1:50 %we could have done bisection, but we were lazy this time
    k = [0:1:N]';
    C = cos(k*w)';
    cvx_begin
        variable a(N+1)
        %passband constraints
        C(1:wi,:)*a <= 1.12;
        C(1:wi,:)*a >= 0.89;
        cos(wp*linspace(0,N,N+1))*a >= 0.89
        %stopband constraints
        C(wo:K,:)*a <= alpha;
        C(wo:K,:)*a >= -alpha;
        cos(wc*linspace(0,N,N+1))*a <= alpha
    cvx_end
    if (strcmp(cvx_status,'Solved') == 1)
```

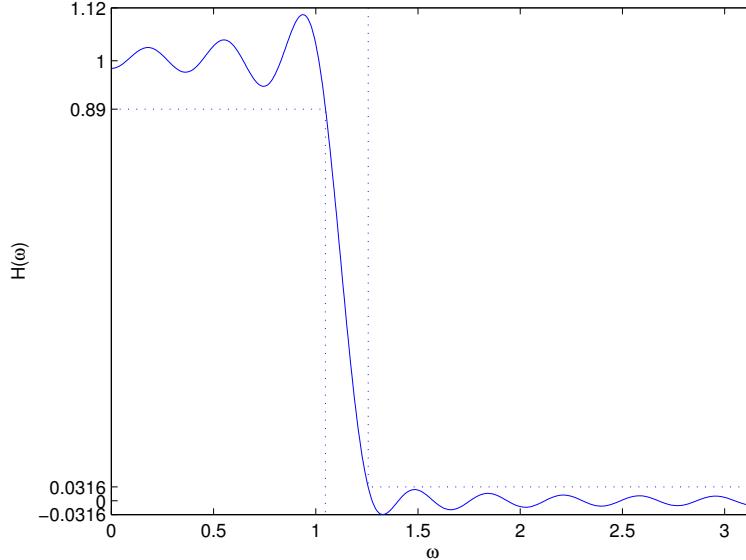
```

        break;
    end
end

H = a'*cos(k*w);
plot(w,H)
set(gca,'YTick',[ -alpha 0 alpha .89 1 1.12])
axis([0 pi -alpha 1.12])
hold on
plot([0 wp wp],[.89 .89 -alpha],':')
plot([wc wc pi],[1.12 alpha alpha],':')
xlabel('omega')
ylabel('H(omega)')
hold off

```

It yields the following filter.



12.2 SINR maximization. Solve the following instance of problem 4.20: We have $n = 5$ transmitters, grouped into two groups: $\{1, 2\}$ and $\{3, 4, 5\}$. The maximum power for each transmitter is 3, the total power limit for the first group is 4, and the total power limit for the second group is 6. The noise σ is equal to 0.5 and the limit on total received power is 5 for each receiver. Finally, the path gain matrix is given by

$$G = \begin{bmatrix} 1.0 & 0.1 & 0.2 & 0.1 & 0.0 \\ 0.1 & 1.0 & 0.1 & 0.1 & 0.0 \\ 0.2 & 0.1 & 2.0 & 0.2 & 0.2 \\ 0.1 & 0.1 & 0.2 & 1.0 & 0.1 \\ 0.0 & 0.0 & 0.2 & 0.1 & 1.0 \end{bmatrix}.$$

Find the transmitter powers p_1, \dots, p_5 that maximize the minimum SINR ratio over all receivers.

Also report the maximum SINR value. Solving the problem to an accuracy of 0.05 (in SINR) is fine.

Hint. When implementing a bisection method in CVX, you will need to check feasibility of a convex problem. You can do this using `strcmpi(cvx_status, 'Solved')`.

Solution. The Matlab code used to solve the problem is reported at the end of this paragraph. The optimal values of the transmitted powers are: $p_1 = 2.1188$, $p_2 = 1.8812$, $p_3 = 1.6444$, $p_4 = 2.3789$, $p_5 = 1.8011$. The maximum SINR is 1.6884.

```
% SINR maximization (an instance of exercise 4.20).
%
n = 5;
G =[1    0.1   0.2   0.1   0
     0.1   1     0.1   0.1   0
     0.2   0.1   2     0.2   0.2
     0.1   0.1   0.2   1     0.1
     0     0     0.2   0.1   1];
sigma = 0.5;
Pmax = 3;

% set up lower and upper bounds
l = 0;
u = 100;
tol = 1e-4;
Gtilde = G - diag(diag(G));

% use bisection to solve linear-fractional problem
while u-l > tol
    t = (l+u)/2;

    % solve feasibility problem for this value of t
    cvx_begin
        cvx_quiet(true);
        variable p(n);
        Gtilde*p + sigma*ones(n,1) <= t * diag(G).*p;
        p >= 0;
        p <= Pmax;
        p(1)+p(2) <= 4;
        p(3)+p(4)+p(5) <= 6;
        G*p <= 5;
    cvx_end

    if strcmpi(cvx_status, 'Solved')
        u = t;
        % save best values
        pstar = p;
```

```

sstar = 1/t;
else
    t = t;
end
end

% output results
pstar
sstar

```

12.3 *Power control for sum rate maximization in interference channel.* We consider the optimization problem

$$\begin{aligned} & \text{maximize} \quad \sum_{i=1}^n \log \left(1 + \frac{p_i}{\sum_{j \neq i} A_{ij} p_j + v_i} \right) \\ & \text{subject to} \quad \sum_{i=1}^n p_i = 1 \\ & \quad p_i \geq 0, \quad i = 1, \dots, n \end{aligned}$$

with variables $p \in \mathbf{R}^n$. The problem data are the matrix $A \in \mathbf{R}^{n \times n}$ and the vector $v \in \mathbf{R}^n$. We assume A and v are componentwise nonnegative ($A_{ij} \geq 0$ and $v_i \geq 0$), and that the diagonal elements of A are equal to one. If the off-diagonal elements of A are zero ($A = I$), the problem has a simple solution, given by the waterfilling method. We are interested in the case where the off-diagonal elements are nonzero.

We can give the following interpretation of the problem, which is not needed below. The variables in the problem are the transmission powers in a communications system. We limit the total power to one (for simplicity; we could have used any other number). The i th term in the objective is the Shannon capacity of the i th channel; the fraction in the argument of the log is the signal to interference plus noise ratio.

We can express the problem as

$$\begin{aligned} & \text{maximize} \quad \sum_{i=1}^n \log \left(\frac{\sum_{j=1}^n B_{ij} p_j}{\sum_{j=1}^n B_{ij} p_j - p_i} \right) \\ & \text{subject to} \quad \sum_{i=1}^n p_i = 1 \\ & \quad p_i \geq 0, \quad i = 1, \dots, n, \end{aligned} \tag{49}$$

where $B \in \mathbf{R}^{n \times n}$ is defined as $B = A + v\mathbf{1}^T$, i.e., $B_{ij} = A_{ij} + v_i$, $i, j = 1, \dots, n$. Suppose B is nonsingular and

$$B^{-1} = I - C$$

with $C_{ij} \geq 0$. Express the problem above as a convex optimization problem. *Hint.* Use $y = Bp$ as variables.

Solution. We make a change of variables

$$y = Bp, \quad p = B^{-1}y = y - Cy.$$

If we substitute this in (49) the cost function reduces to

$$\sum_{i=1}^n \log(y_i/(Cy)_i) = \log \prod_i (y_i/(Cy)_i),$$

and the last constraint to

$$y_i \geq (Cy)_i, \quad i = 1, \dots, n.$$

Since this constraint and the objective are homogeneous, we can replace the constraint $\mathbf{1}^T p = 1$ by another normalization, *e.g.*,

$$\prod_i y_i = 1.$$

This results in the GP

$$\begin{aligned} & \text{minimize} && \prod_i (Cy)_i \\ & \text{subject to} && y_i \geq (Cy)_i, \quad i = 1, \dots, n \\ & && \prod_i y_i = 1. \end{aligned}$$

12.4 Radio-relay station placement and power allocation. Radio relay stations are to be located at positions $x_1, \dots, x_n \in \mathbf{R}^2$, and transmit at power $p_1, \dots, p_n \geq 0$. In this problem we will consider the problem of simultaneously deciding on good locations *and* operating powers for the relay stations.

The received signal power S_{ij} at relay station i from relay station j is proportional to the transmit power and inversely proportional to the distance, *i.e.*,

$$S_{ij} = \frac{\alpha p_j}{\|x_i - x_j\|^2},$$

where $\alpha > 0$ is a known constant.

Relay station j must transmit a signal to relay station i at the rate (or bandwidth) $R_{ij} \geq 0$ bits per second; $R_{ij} = 0$ means that relay station j does not need to transmit any message (directly) to relay station i . The matrix of bit rates R_{ij} is given. Although it doesn't affect the problem, R would likely be sparse, *i.e.*, each relay station needs to communicate with only a few others.

To guarantee accurate reception of the signal from relay station j to i , we must have

$$S_{ij} \geq \beta R_{ij},$$

where $\beta > 0$ is a known constant. (In other words, the minimum allowable received signal power is proportional to the signal bit rate or bandwidth.)

The relay station positions x_{r+1}, \dots, x_n are fixed, *i.e.*, problem parameters. The problem variables are x_1, \dots, x_r and p_1, \dots, p_n . The goal is to choose the variables to minimize the total transmit power, *i.e.*, $p_1 + \dots + p_n$.

Explain how to solve this problem as a convex or quasiconvex optimization problem. If you introduce new variables, or transform the variables, explain. Clearly give the objective and inequality constraint functions, explaining why they are convex. If your problem involves equality constraints, express them using an affine function.

Solution. XXX lost? it should exist somewhere XXX

12.5 Power allocation with coherent combining receivers. In this problem we consider a variation on the power allocation problem described on pages 4-13 and 4-14 of the notes. In that problem we have m transmitters, each of which transmits (broadcasts) to n receivers, so the total number of receivers is mn . In this problem we have the converse: multiple transmitters send a signal to each receiver.

More specifically we have m receivers labeled $1, \dots, m$, and mn transmitters labeled (j, k) , $j = 1, \dots, m$, $k = 1, \dots, n$. The transmitters $(i, 1), \dots, (i, n)$ all transmit the same message to the receiver i , for $i = 1, \dots, m$.

Transmitter (j, k) operates at power p_{jk} , which must satisfy $0 \leq p_{jk} \leq P_{\max}$, where P_{\max} is a given maximum allowable transmitter power.

The path gain from transmitter (j, k) to receiver i is $A_{ijk} > 0$ (which are given and known). Thus the power received at receiver i from transmitter (j, k) is given by $A_{ijk}p_{jk}$.

For $i \neq j$, the received power $A_{ijk}p_{jk}$ represents an interference signal. The total interference-plus-noise power at receiver i is given by

$$I_i = \sum_{j \neq i, k=1, \dots, n} A_{ijk}p_{jk} + \sigma$$

where $\sigma > 0$ is the known, given (self) noise power of the receivers. Note that the *powers* of the interference and noise signals add to give the total interference-plus-noise power.

The receivers use *coherent detection and combining* of the desired message signals, which means the effective received signal power at receiver i is given by

$$S_i = \left(\sum_{k=1, \dots, n} (A_{iik}p_{ik})^{1/2} \right)^2.$$

(Thus, the *amplitudes* of the desired signals add to give the effective signal amplitude.)

The total signal to interference-plus-noise ratio (SINR) for receiver i is given by $\gamma_i = S_i/I_i$.

The problem is to choose transmitter powers p_{jk} that maximize the minimum SINR $\min_i \gamma_i$, subject to the power limits.

Explain in detail how to solve this problem using convex or quasiconvex optimization. If you transform the problem by using a different set of variables, explain completely. Identify the objective function, and all constraint functions, indicating if they are convex or quasiconvex, etc.

Solution. The objective is just $\mathbf{1}^T p$. The requirement for accurate reception is

$$\frac{\alpha p_j}{\|x_i - x_j\|^2} \geq \beta R_{ij}, \quad i, j = 1, \dots, n$$

which can be written as

$$\|x_i - x_j\|^2 \beta R_{ij} - \alpha p_j \leq 0, \quad i, j = 1, \dots, n.$$

This constraint is convex in (x, p) since $\|x_i - x_j\|$ is convex and $-\alpha p_j$ is linear in (x, p) (hence convex). In fact this constraint is a second-order cone constraint. Note that this constraint includes the nonnegativity constraint on p_i .

Therefore we can formulate this problem as the quadratically constrained linear program

$$\begin{aligned} & \text{minimize} && \mathbf{1}^T p \\ & \text{subject to} && \|x_i - x_j\|^2 \beta R_{ij} - \alpha p_j \leq 0, \quad i, j = 1, \dots, n \end{aligned}$$

where the variables are x_1, \dots, x_r and p_1, \dots, p_n . The problem parameters or data are x_{r+1}, \dots, x_n , α, β , and R_{ij} , $i, j = 1, \dots, n$.

The power limit constraints, $0 \leq p_{jk} \leq P_{\max}$, are evidently convex in p .

The S/I ratio for receiver i is given by the (scary) expression

$$\gamma_i = \frac{\left(\sum_{k=1, \dots, n} (A_{iik} p_{ik})^{1/2} \right)^2}{\sum_{j \neq i, k=1, \dots, n} A_{ijk} p_{jk} + \sigma}.$$

To say that our objective exceeds t , *i.e.*, $\min_i \gamma_i \geq t$, is the same as

$$\left(\sum_{k=1, \dots, n} (A_{iik} p_{ik})^{1/2} \right)^2 \geq t \left(\sum_{j \neq i, k=1, \dots, n} A_{ijk} p_{jk} + \sigma \right). \quad (50)$$

In fact, we will see that this describes a convex constraint on p , which means that the minimum S/I ratio is a *quasiconcave* function of p . So our problem is, directly, a quasiconvex problem in the variables p_{ij} .

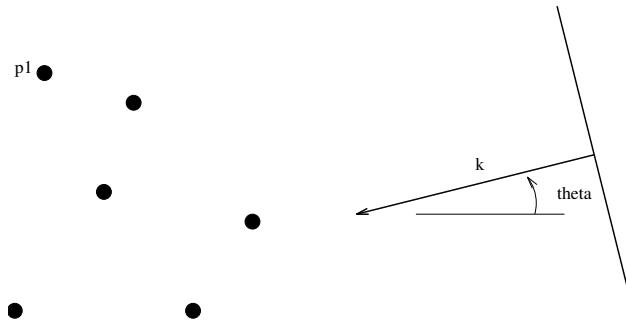
To see this, we start by examining the function

$$f(x) = \left(\sum_{k=1}^n x_k^{1/2} \right)^2$$

for $x \succeq 0$. It looks reminiscent of a p -norm, except that $p = 1/2$, so we know the function isn't convex. Well, in fact, the function is *concave* on $x \succeq 0$ as given in the notes.

Therefore, it follows that the constraint (50) is convex, since it has the form $g(p) \geq h(p)$ where g is concave and h is affine.

12.6 Antenna array weight design. We consider an array of n omnidirectional antennas in a plane, at positions (x_k, y_k) , $k = 1, \dots, n$.



A unit plane wave with frequency ω is incident from an angle θ . This incident wave induces in the k th antenna element a (complex) signal $\exp(i(x_k \cos \theta + y_k \sin \theta - \omega t))$, where $i = \sqrt{-1}$. (For simplicity we assume that the spatial units are normalized so that the wave number is one, *i.e.*, the wavelength is $\lambda = 2\pi$.) This signal is demodulated, *i.e.*, multiplied by $e^{i\omega t}$, to obtain the baseband signal (complex number) $\exp(i(x_k \cos \theta + y_k \sin \theta))$. The baseband signals of the n antennas are combined linearly to form the output of the antenna array

$$\begin{aligned} G(\theta) &= \sum_{k=1}^n w_k e^{i(x_k \cos \theta + y_k \sin \theta)} \\ &= \sum_{k=1}^n (w_{\text{re},k} \cos \gamma_k(\theta) - w_{\text{im},k} \sin \gamma_k(\theta)) + i(w_{\text{re},k} \sin \gamma_k(\theta) + w_{\text{im},k} \cos \gamma_k(\theta)), \end{aligned}$$

if we define $\gamma_k(\theta) = x_k \cos \theta + y_k \sin \theta$. The complex weights in the linear combination,

$$w_k = w_{\text{re},k} + i w_{\text{im},k}, \quad k = 1, \dots, n,$$

are called the *antenna array coefficients* or *shading coefficients*, and will be the design variables in the problem. For a given set of weights, the combined output $G(\theta)$ is a function of the angle of arrival θ of the plane wave. The design problem is to select weights w_i that achieve a desired directional pattern $G(\theta)$.

We now describe a basic weight design problem. We require unit gain in a target direction θ^{tar} , *i.e.*, $G(\theta^{\text{tar}}) = 1$. We want $|G(\theta)|$ small for $|\theta - \theta^{\text{tar}}| \geq \Delta$, where 2Δ is our beamwidth. To do this, we can minimize

$$\max_{|\theta - \theta^{\text{tar}}| \geq \Delta} |G(\theta)|,$$

where the maximum is over all $\theta \in [-\pi, \pi]$ with $|\theta - \theta^{\text{tar}}| \geq \Delta$. This number is called the sidelobe level for the array; our goal is to minimize the sidelobe level. If we achieve a small sidelobe level, then the array is relatively insensitive to signals arriving from directions more than Δ away from the target direction. This results in the optimization problem

$$\begin{aligned} &\text{minimize} && \max_{|\theta - \theta^{\text{tar}}| \geq \Delta} |G(\theta)| \\ &\text{subject to} && G(\theta^{\text{tar}}) = 1, \end{aligned}$$

with $w \in \mathbf{C}^n$ as variables.

The objective function can be approximated by discretizing the angle of arrival with (say) N values (say, uniformly spaced) $\theta_1, \dots, \theta_N$ over the interval $[-\pi, \pi]$, and replacing the objective with

$$\max\{|G(\theta_k)| \mid |\theta_k - \theta^{\text{tar}}| \geq \Delta\}$$

- (a) Formulate the antenna array weight design problem as an SOCP.
- (b) Solve an instance using CVX, with $n = 40$, $\theta^{\text{tar}} = 15^\circ$, $\Delta = 15^\circ$, $N = 400$, and antenna positions generated using

```
rand('state',0);
n = 40;
x = 30 * rand(n,1);
y = 30 * rand(n,1);
```

Compute the optimal weights and make a plot of $|G(\theta)|$ (on a logarithmic scale) versus θ .
Hint. CVX can directly handle complex variables, and recognizes the modulus `abs(x)` of a complex number as a convex function of its real and imaginary parts, so you do not need to explicitly form the SOCP from part (a). Even more compactly, you can use `norm(x, Inf)` with complex argument.

Solution.

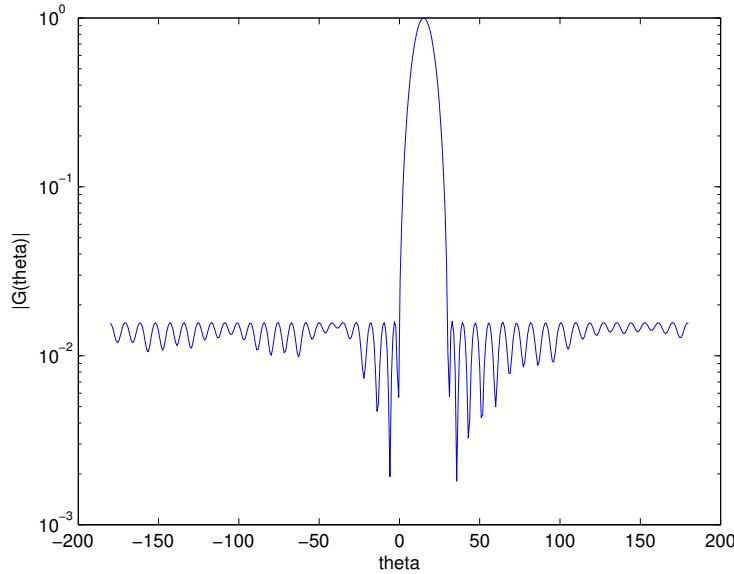
(a) The problem can be expressed as the SOCP

$$\begin{aligned} & \text{minimize} && t \\ & \text{subject to} && \|A_k x\|_2 \leq t, \quad k \in \mathcal{I} \\ & && Bx = d \end{aligned}$$

with variables x, t , where $\mathcal{I} = \{k \mid |\theta_k - \theta^{\text{tar}}| \geq \Delta\}$ and

$$\begin{aligned} x &= \begin{bmatrix} w_{\text{re}} \\ w_{\text{im}} \end{bmatrix} \in \mathbf{R}^{2n} \\ A_k &= \begin{bmatrix} \cos \gamma_1(\theta_k) & \dots & \cos \gamma_n(\theta_k) & -\sin \gamma_1(\theta_k) & \dots & -\sin \gamma_n(\theta_k) \\ \sin \gamma_1(\theta_k) & \dots & \sin \gamma_n(\theta_k) & \cos \gamma_1(\theta_k) & \dots & \cos \gamma_n(\theta_k) \end{bmatrix} \\ B &= \begin{bmatrix} \cos \gamma_1(\theta^{\text{tar}}) & \dots & \cos \gamma_n(\theta^{\text{tar}}) & -\sin \gamma_1(\theta^{\text{tar}}) & \dots & -\sin \gamma_n(\theta^{\text{tar}}) \\ \sin \gamma_1(\theta^{\text{tar}}) & \dots & \sin \gamma_n(\theta^{\text{tar}}) & \cos \gamma_1(\theta^{\text{tar}}) & \dots & \cos \gamma_n(\theta^{\text{tar}}) \end{bmatrix} \\ d &= \begin{bmatrix} 1 \\ 0 \end{bmatrix}. \end{aligned}$$

(b) The figure below shows the output of the antenna array for different values of θ .



The following code solves the problem.

```

rand('state',0);
n = 40;
X = 30*[rand(1,n); rand(1,n)];
N=400;
beamwidth = 15*pi/180;
theta_tar=15*pi/180;
theta = linspace(theta_tar+beamwidth, 2*pi+theta_tar-beamwidth, N)';
A = exp(i * [cos(theta), sin(theta)] * X);
Atar = exp(i * [cos(theta_tar), sin(theta_tar)] * X)
cvx_begin
variable w(n) complex
minimize(max(abs(A*w)))
subject to
Atar*w == 1;
cvx_end

```

12.7 Power allocation problem with analytic solution. Consider a system of n transmitters and n receivers. The i th transmitter transmits with power x_i , $i = 1, \dots, n$. The vector x will be the variable in this problem. The path gain from each transmitter j to each receiver i will be denoted A_{ij} and is assumed to be known (obviously, $A_{ij} \geq 0$, so the matrix A is elementwise nonnegative, and $A_{ii} > 0$). The signal received by each receiver i consists of three parts: the desired signal, arriving from transmitter i with power $A_{ii}x_i$, the interfering signal, arriving from the other receivers with power $\sum_{j \neq i} A_{ij}x_j$, and noise β_i (which are positive and known). We are interested in allocating the powers x_i in such a way that the signal to noise plus interference ratio at each of the receivers exceeds a level α . (Thus α is the minimum acceptable SNIR for the receivers; a typical value might be around $\alpha = 3$, *i.e.*, around 10dB). In other words, we want to find $x \succeq 0$ such that for $i = 1, \dots, n$

$$A_{ii}x_i \geq \alpha \left(\sum_{j \neq i} A_{ij}x_j + \beta_i \right).$$

Equivalently, the vector x has to satisfy

$$x \succeq 0, \quad Bx \succeq \alpha\beta \tag{51}$$

where $B \in \mathbf{R}^{n \times n}$ is defined as

$$B_{ii} = A_{ii}, \quad B_{ij} = -\alpha A_{ij}, \quad j \neq i.$$

- (a) Show that (51) is feasible if and only if B is invertible and $z = B^{-1}\mathbf{1} \succeq 0$ ($\mathbf{1}$ is the vector with all components 1). Show how to construct a feasible power allocation x from z .
- (b) Show how to find the largest possible SNIR, *i.e.*, how to maximize α subject to the existence of a feasible power allocation.

To solve this problem you may need the following:

Hint. Let $T \in \mathbf{R}^{n \times n}$ be a matrix with nonnegative elements, and $s \in \mathbf{R}$. Then the following are equivalent:

- (a) $s > \rho(T)$, where $\rho(T) = \max_i |\lambda_i(T)|$ is the spectral radius of T .
- (b) $sI - T$ is nonsingular and the matrix $(sI - T)^{-1}$ has nonnegative elements.
- (c) there exists an $x \succeq 0$ with $(sI - T)x \succ 0$.

(For such s , the matrix $sI - T$ is called a *nonsingular M-matrix*.)

Remark. This problem gives an analytic solution to a very special form of transmitter power allocation problem. Specifically, there are exactly as many transmitters as receivers, and no power limits on the transmitters. One consequence is that the receiver noises β_i play no role at all in the solution — just crank up all the transmitters to overpower the noises!

Solution. First note that the existence of a vector x satisfying

$$x \succeq 0, \quad Bx \succeq \alpha\beta \quad (52)$$

is equivalent to the existence of a vector x satisfying

$$x \succeq 0, \quad Bx \succ 0. \quad (53)$$

The reason is simply that we can always multiply any x satisfying (53) by a large enough positive number γ so that

$$\gamma x \succeq 0, \quad B(\gamma x) \succeq \alpha\beta$$

and hence γx would satisfy (52). Obviously, since $\alpha\beta \succ 0$, any solution of (52) is also a solution of (53).

We can assume without loss of generality that the diagonal elements of A are equal to one (by dividing each row of A by A_{ii}). The matrix B can then be written as

$$B = (1 + \alpha)I - \alpha A,$$

which has the form $B = sI - T$ where T has nonnegative elements. Hence the following properties are equivalent:

- (a) $1 + \alpha > \alpha\rho(A)$
- (b) B is nonsingular and B^{-1} has nonnegative elements
- (c) there exists an $x \succeq 0$ with $Bx \succ 0$.

The two parts of the problem now follow immediately.

- (a) The ‘if’-part is immediate since z solves (53) if it is nonnegative. For the ‘only if’ part, we combine the 2nd and 3rd properties listed above, and conclude that there exists an x satisfying (53) if and only if B^{-1} exists and has nonnegative elements. This implies that $z = B^{-1}\mathbf{1} \succeq 0$.
- (b) B is a nonsingular *M-matrix* if $1 + \alpha > \alpha\rho(A)$, i.e., if

$$\alpha < \frac{1}{\rho(A) - 1}.$$

12.8 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, \dots, 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, \dots, t_n , that must satisfy $t_1 + \dots + t_n = T$, $t_i \geq 0$. 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

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

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,

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

dual feasibility, $\lambda \geq 0$,

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

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

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

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

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.

12.9 Optimal jamming power allocation. A set of n jammers transmit with (nonnegative) powers p_1, \dots, p_n , which are to be chosen subject to the constraints

$$p \succeq 0, \quad Fp \preceq g.$$

The jammers produce interference power at m receivers, given by

$$d_i = \sum_{j=1}^n G_{ij} p_j, \quad i = 1, \dots, m,$$

where G_{ij} is the (nonnegative) channel gain from jammer j to receiver i .

Receiver i has capacity (in bits/s) given by

$$C_i = \alpha \log(1 + \beta_i / (\sigma_i^2 + d_i)), \quad i = 1, \dots, m,$$

where α , β_i , and σ_i are positive constants. (Here β_i is proportional to the signal power at receiver i and σ_i^2 is the receiver i self-noise, but you won't need to know this to solve the problem.)

Explain how to choose p to minimize the sum channel capacity, $C = C_1 + \dots + C_m$, using convex optimization. (This corresponds to the most effective jamming, given the power constraints.) The problem data are F , g , G , α , β_i , σ_i .

If you change variables, or transform your problem in any way that is not obvious (for example, you form a relaxation), you must explain fully how your method works, and why it gives the solution. If your method relies on any convex functions that we have not encountered before, you must show that the functions are convex.

Disclaimer. The teaching staff does not endorse jamming, optimal or otherwise.

Solution. This is almost a trick question, because it is so easy: there is no need to change variables, or introduce any relaxation. We'll show that the following problem is convex:

$$\begin{aligned} & \text{minimize} && C \\ & \text{subject to} && p \succeq 0, \quad Fp \preceq g. \end{aligned}$$

We observe that the function $f(x) = \log(1 + 1/x)$ is convex for $x > 0$: we have

$$f'(x) = -\frac{1}{x^2 + x},$$

which evidently is increasing in x , so $f''(x) > 0$.

Here's another proof that $f(x) = \log(1 + 1/x)$ is convex: it is the composition of $h(u_1, u_2) = \log(e^{u_1} + e^{u_2})$, which is increasing and convex, with $u_1 = 0$ and $u_2 = -\log x$, which are convex. Thus,

$$h(g(x)) = \log(e^0 + e^{-\log x}) = \log(1 + 1/x)$$

is convex.

And, here's yet another, just for fun. We write

$$\log(1 + 1/x) = -\log(x/(x+1)) = -\log(1 - 1/(x+1)),$$

the composition of $h(u) = -\log(1 - u)$, which is convex and increasing, with $u = 1/(x+1)$, which is convex.

For rows g_i^T of G , we write

$$C_i = \alpha f((g_i^T p + \sigma_i^2)/\beta_i),$$

which is convex in p , since the argument to f is affine in p . It follows that C is a convex function of p .

12.10 2D filter design. A symmetric convolution kernel with support $\{-(N-1), \dots, N-1\}^2$ is characterized by N^2 coefficients

$$h_{kl}, \quad k, l = 1, \dots, N.$$

These coefficients will be our variables. The corresponding 2D frequency response (Fourier transform) $H : \mathbf{R}^2 \rightarrow \mathbf{R}$ is given by

$$H(\omega_1, \omega_2) = \sum_{k,l=1,\dots,N} h_{kl} \cos((k-1)\omega_1) \cos((l-1)\omega_2),$$

where ω_1 and ω_2 are the frequency variables. Evidently we only need to specify H over the region $[0, \pi]^2$, although it is often plotted over the region $[-\pi, \pi]^2$. (It won't matter in this problem, but we should mention that the coefficients h_{kl} above are not exactly the same as the impulse response coefficients of the filter.)

We will design a 2D filter (*i.e.*, find the coefficients h_{kl}) to satisfy $H(0, 0) = 1$ and to minimize the maximum response R in the rejection region $\Omega_{\text{rej}} \subset [0, \pi]^2$,

$$R = \sup_{(\omega_1, \omega_2) \in \Omega_{\text{rej}}} |H(\omega_1, \omega_2)|.$$

- (a) Explain why this 2D filter design problem is convex.
- (b) Find the optimal filter for the specific case with $N = 5$ and

$$\Omega_{\text{rej}} = \{(\omega_1, \omega_2) \in [0, \pi]^2 \mid \omega_1^2 + \omega_2^2 \geq W^2\},$$

with $W = \pi/4$.

You can approximate R by sampling on a grid of frequency values. Define

$$\omega^{(p)} = \pi(p-1)/M, \quad p = 1, \dots, M.$$

(You can use $M = 25$.) We then replace the exact expression for R above with

$$\hat{R} = \max\{|H(\omega^{(p)}, \omega^{(q)})| \mid p, q = 1, \dots, M, (\omega^{(p)}, \omega^{(q)}) \in \Omega_{\text{rej}}\}.$$

Give the optimal value of \hat{R} . Plot the optimal frequency response using `plot_2D_filt(h)`, available on the course web site, where `h` is the matrix containing the coefficients h_{kl} .

Solution.

```
% 2D filter design problem via convex optimization

% Number of taps of the FIR filter
N=5;

% Frequency discretization for design
M=25;

% Specifications
W = pi/4;
```

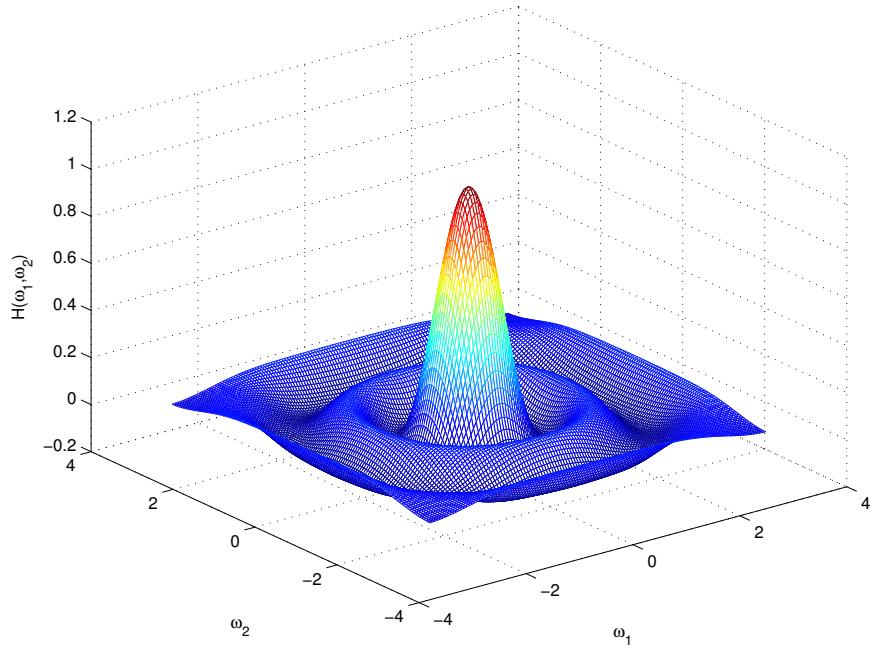
```

cvx_begin
variable h(N,N);
variable t
sum(sum(h)) == 1; % H(0,0)==1
minimize (t);
% loop over frequency
for p=1:M
    for q=1:M
        w1 = pi*((p-1)/M); w2=pi*((q-1)/M);
        cosw1k = cos(w1*[0:(N-1)]);
        cosw2l = cos(w2*[0:(N-1)])';
        H(p,q) = cosw1k*h*cosw2l;
        if (w1^2+w2^2 >= W^2)
            abs(H(p,q)) <= t
        end
    end
end
cvx_end

HH=plot_2D_filt(h);
print -depsc opt_2D_filt_resp

```

This yields the following optimal frequency response.



12.11 Maximizing log utility in a wireless system with interference. Consider a wireless network consisting of n data links, labeled $1, \dots, n$. Link i transmits with power $P_i > 0$, and supports a data rate

$R_i = \log(1 + \gamma_i)$, where γ_i is the signal-to-interference-plus-noise ratio (SINR). These SINR ratios depend on the transmit powers, as described below.

The system is characterized by the link gain matrix $G \in \mathbf{R}_{++}^{n \times n}$, where G_{ij} is the gain from the transmitter on link j to the receiver for link i . The received signal power for link i is $G_{ii}P_i$; the noise plus interference power for link i is given by

$$\sigma_i^2 + \sum_{j \neq i} G_{ij}P_j,$$

where $\sigma_i^2 > 0$ is the receiver noise power for link i . The SINR is the ratio

$$\gamma_i = \frac{G_{ii}P_i}{\sigma_i^2 + \sum_{j \neq i} G_{ij}P_j}.$$

The problem is to choose the transmit powers P_1, \dots, P_n , subject to $0 < P_i \leq P_i^{\max}$, in order to maximize the log utility function

$$U(P) = \sum_{i=1}^n \log R_i.$$

(This utility function can be argued to yield a fair distribution of rates.) The data are G , σ_i^2 , and P_i^{\max} .

Formulate this problem as a convex or quasiconvex optimization problem. If you make any transformations or use any steps that are not obvious, explain.

Hints.

- The function $\log \log(1 + e^x)$ is concave. (If you use this fact, you must show it.)
- You might find the new variables defined by $z_i = \log P_i$ useful.

Solution. First we show that $f(x) = \log \log(1 + e^x)$ is concave. Its first and second derivatives are

$$\begin{aligned} f'(x) &= \frac{1}{(\log(1 + e^x))(1 + e^{-x})}, \\ f''(x) &= \frac{-1}{(\log(1 + e^x))^2(1 + e^{-x})^2} + \frac{e^{-x}}{\log(1 + e^x)(1 + e^{-x})^2} \\ &= \frac{1}{\log(1 + e^x)(1 + e^{-x})^2} \left(\frac{-1}{\log(1 + e^x)} + \frac{1}{e^x} \right). \end{aligned}$$

The first term is positive. The second is negative since $e^x \geq \log(1 + e^x)$, which follows from $\log(1 + u) \leq u$. (This in turn follows since $\log(1 + u)$ is a concave function, and u is its first order Taylor approximation at 0.)

Now let's solve the problem. We can write $\log R_i$ as

$$\log R_i = \log \log(1 + \gamma_i) = \log \log(1 + e^{\log \gamma_i}).$$

Defining new variables $z_i = \log P_i$, we write $\log \gamma_i$ as

$$\begin{aligned} \log \gamma_i &= \log \left(\frac{G_{ii}e^{z_i}}{N_i + \sum_{j \neq i} G_{ij}e^{z_j}} \right) \\ &= \log G_{ii} + z_i - \log \left(N_i + \sum_{j \neq i} G_{ij}e^{z_j} \right). \end{aligned}$$

The term $\log \left(N_i + \sum_{j \neq i} G_{ij} e^{z_j} \right)$ can be rewritten (with some effort) into log-sum-exp of an affine transformation of z , so we see that $\log \gamma_i$ is a concave function of z . Using the fact that $\log \log(1+e^u)$ is concave and increasing, we conclude that the function

$$\sum_{i=1}^n \log R_i = \sum_{i=1}^n \log \log(1 + e^{\gamma_i})$$

is a concave function of z . (It is not a simple function, to be sure—but it is concave!)

To solve the problem, then, we maximize this concave function, subject to the constraints $z_i \leq \log P_i^{\max}$. We recover the optimal P_i via $P_i^* = e^{z_i^*}$.

One common error was to argue that $\log(1 + \gamma_i)$ is quasiconcave in the powers (which is true), so the problem is quasiconcave (which is false, since sum of quasiconcave is not quasiconcave).

12.12 Spectral factorization via semidefinite programming. A Toeplitz matrix is a matrix that has constant values on its diagonals. We use the notation

$$T_m(x_1, \dots, x_m) = \begin{bmatrix} x_1 & x_2 & x_3 & \cdots & x_{m-1} & x_m \\ x_2 & x_1 & x_2 & \cdots & x_{m-2} & x_{m-1} \\ x_3 & x_2 & x_1 & \cdots & x_{m-3} & x_{m-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{m-1} & x_{m-2} & x_{m-2} & \cdots & x_1 & x_2 \\ x_m & x_{m-1} & x_{m-2} & \cdots & x_2 & x_1 \end{bmatrix}$$

to denote the symmetric Toeplitz matrix in $\mathbf{S}^{m \times m}$ constructed from x_1, \dots, x_m . Consider the semidefinite program

$$\begin{aligned} & \text{minimize} && c^T x \\ & \text{subject to} && T_n(x_1, \dots, x_n) \succeq e_1 e_1^T, \end{aligned}$$

with variable $x = (x_1, \dots, x_n)$, where $e_1 = (1, 0, \dots, 0)$.

- (a) Derive the dual of the SDP above. Denote the dual variable as Z . (Hence $Z \in \mathbf{S}^n$ and the dual constraints include an inequality $Z \succeq 0$.)
- (b) Show that $T_n(x_1, \dots, x_n) \succ 0$ for every feasible x in the SDP above. You can do this by induction on n .
 - For $n = 1$, the constraint is $x_1 \geq 1$ which obviously implies $x_1 > 0$.
 - In the induction step, assume $n \geq 2$ and that $T_{n-1}(x_1, \dots, x_{n-1}) \succ 0$. Use a Schur complement argument and the Toeplitz structure of T_n to show that $T_n(x_1, \dots, x_n) \succeq e_1 e_1^T$ implies $T_n(x_1, \dots, x_n) \succ 0$.
- (c) Suppose the optimal value of the SDP above is finite and attained, and that Z is dual optimal. Use the result of part (b) to show that the rank of Z is at most one, *i.e.*, Z can be expressed as $Z = yy^T$ for some n -vector y . Show that y satisfies

$$\begin{aligned} y_1^2 + y_2^2 + \cdots + y_n^2 &= c_1 \\ y_1 y_2 + y_2 y_3 + \cdots + y_{n-1} y_n &= c_2 / 2 \\ &\vdots \\ y_1 y_{n-1} + y_2 y_n &= c_{n-1} / 2 \\ y_1 y_n &= c_n / 2. \end{aligned}$$

This can be expressed as an identity $|Y(\omega)|^2 = R(\omega)$ between two functions

$$\begin{aligned} Y(\omega) &= y_1 + y_2 e^{-i\omega} + y_3 e^{-3i\omega} + \cdots + y_n e^{-i(n-1)\omega} \\ R(\omega) &= c_1 + c_2 \cos \omega + c_3 \cos(2\omega) + \cdots + c_n \cos((n-1)\omega) \end{aligned}$$

(with $i = \sqrt{-1}$). The function $Y(\omega)$ is called a *spectral factor* of the trigonometric polynomial $R(\omega)$.

Solution.

(a) The Lagrangian is

$$\begin{aligned} L(x, Z) &= c^T x + \mathbf{tr}(Z(e_1 e_1^T - T_n(x_1, \dots, x_n))) \\ &= c^T x + Z_{11} - x_1(Z_{11} + \cdots + Z_{nn}) - 2x_2(Z_{21} + \cdots + Z_{n,n-1}) \\ &\quad - 2x_3(Z_{31} + \cdots + Z_{n,n-2}) - \cdots - 2x_n Z_{n1}. \end{aligned}$$

In the dual SDP we maximize $g(Z) = \inf_x L(x, Z)$ subject to $Z \succeq 0$:

$$\begin{aligned} &\text{maximize} && Z_{11} \\ &\text{subject to} && Z_{11} + Z_{22} + \cdots + Z_{nn} = c_1 \\ & && 2(Z_{21} + Z_{32} + \cdots + Z_{n,n-1}) = c_2 \\ & && 2(Z_{31} + Z_{42} + \cdots + Z_{n,n-2}) = c_3 \\ & && \vdots \\ & && 2(Z_{n-1,1} + Z_{n2}) = c_{n-1} \\ & && 2Z_{n1} = c_n \\ & && Z \succeq 0. \end{aligned}$$

(b) The constraint $T_n(x_1, \dots, x_n) \succeq e_1 e_1^T$ can be written as

$$\begin{bmatrix} x_1 - 1 & \bar{x}^T \\ \bar{x} & A \end{bmatrix} \succeq 0.$$

where $\bar{x} = (x_2, \dots, x_n)$ and $A = T_{n-1}(x_1, \dots, x_{n-1})$. By assumption the 2,2 block A is positive definite, so by the Schur complement theorem the inequality is equivalent to $x_1 - 1 - \bar{x}^T A^{-1} \bar{x} \geq 0$. Hence $x_1 - \bar{x}^T A^{-1} x \geq 1 > 0$ and therefore

$$T_n(x_1, \dots, x_n) = \begin{bmatrix} x_1 & \bar{x}^T \\ \bar{x} & A \end{bmatrix} \succ 0.$$

(c) By strong duality, the primal and dual optimal solutions satisfy $\mathbf{tr}(XZ) = 0$ where

$$X = T_n(x_1, \dots, x_n) - e_1 e_1^T.$$

From part (b), $T_n(x_1, \dots, x_n)$ is strictly positive definite and therefore the rank of X is at least $n-1$, i.e., its nullspace has dimension at most one. Then $\mathbf{tr}(ZX) = 0$ and $Z \succeq 0$ imply $Z = yy^T$ with y in the nullspace of X .

Plugging in $Z_{ij} = y_i y_j$ in the equality constraints in the dual SDP gives

$$y_1^2 + \cdots + y_n^2 = c_1, \quad y_1 y_k + \cdots + y_{n-k} y_n = c_k/2, \quad k = 2, \dots, n.$$

12.13 Bandlimited signal recovery from zero-crossings. Let $y \in \mathbf{R}^n$ denote a *bandlimited* signal, which means that it can be expressed as a linear combination of sinusoids with frequencies in a band:

$$y_t = \sum_{j=1}^B a_j \cos\left(\frac{2\pi}{n}(f_{\min} + j - 1)t\right) + b_j \sin\left(\frac{2\pi}{n}(f_{\min} + j - 1)t\right), \quad t = 1, \dots, n,$$

where f_{\min} is lowest frequency in the band, B is the bandwidth, and $a, b \in \mathbf{R}^B$ are the cosine and sine coefficients, respectively. We are given f_{\min} and B , but not the coefficients a, b or the signal y .

We do not know y , but we are given its sign $s = \mathbf{sign}(y)$, where $s_t = 1$ if $y_t \geq 0$ and $s_t = -1$ if $y_t < 0$. (Up to a change of overall sign, this is the same as knowing the ‘zero-crossings’ of the signal, *i.e.*, when it changes sign. Hence the name of this problem.)

We seek an estimate \hat{y} of y that is consistent with the bandlimited assumption and the given signs. Of course we cannot distinguish y and αy , where $\alpha > 0$, since both of these signals have the same sign pattern. Thus, we can only estimate y up to a positive scale factor. To normalize \hat{y} , we will require that $\|\hat{y}\|_1 = n$, *i.e.*, the average value of $|y_i|$ is one. Among all \hat{y} that are consistent with the bandlimited assumption, the given signs, and the normalization, we choose the one that minimizes $\|\hat{y}\|_2$.

- (a) Show how to find \hat{y} using convex or quasiconvex optimization.
- (b) Apply your method to the problem instance with data in `zero_crossings_data.*`. The data files also include the true signal y (which of course you cannot use to find \hat{y}). Plot \hat{y} and y , and report the relative recovery error, $\|y - \hat{y}\|_2/\|y\|_2$. Give one short sentence commenting on the quality of the recovery.

Solution.

- (a) We can express our estimate as $\hat{y} = Ax$, where $x = (a, b) \in \mathbf{R}^{2B}$ is the vector of cosine and sinusoid coefficients, and we define the matrix

$$A = [C \ S] \in \mathbf{R}^{n \times 2B},$$

where $C, S \in \mathbf{R}^{n \times B}$ have entries

$$C_{tj} = \cos(2\pi(f_{\min} + j - 1)t/n), \quad S_{tj} = \sin(2\pi(f_{\min} + j - 1)t/n),$$

respectively.

To ensure that the signs of \hat{y} are consistent with s , we need the constraints $s_t a_t^T x \geq 0$ for $t = 1, \dots, n$, where a_1^T, \dots, a_n^T are the rows of A . To achieve the proper normalization, we also need the linear equality constraint $\|\hat{y}\|_1 = s^T A x = n$. (Note that an ℓ_1 -norm equality constraint is not convex in general, but here it is, since the signs are given.)

We have a convex objective and linear inequality and equality constraints, so our optimization problem is convex:

$$\begin{aligned} &\text{minimize} && \|Ax\|_2 \\ &\text{subject to} && s_t a_t^T x \geq 0, \quad t = 1, \dots, n \\ & && s^T A x = n. \end{aligned}$$

We get our estimate as $\hat{y} = Ax^*$, where x^* is a solution of this problem.

One common mistake was to formulate the problem above without the normalization constraint. The (incorrect) argument was that you'd solve the problem, which is homogeneous, and then scale what you get so its ℓ_1 norm is one. This doesn't work, since the (unique) solution to the homogeneous problem is $x = 0$ (since $x = 0$ is feasible). However, this method did give numerical results far better than $x = 0$. The reason is that the solvers returned a very small x , for which Ax had the right sign. And no, that does not mean the error wasn't a bad one.

- (b) The recovery error is 0.1208. This is very impressive considering how little information we were given.

The following matlab code solves the problem:

```

zero_crossings_data;

% Construct matrix A whose columns are bandlimited sinusoids
C = zeros(n,B);
S = zeros(n,B);
for j = 1:B
    C(:,j) = cos(2*pi * (f_min+j-1) * (1:n) / n);
    S(:,j) = sin(2*pi * (f_min+j-1) * (1:n) / n);
end
A = [C S];

% Minimize norm subject to L1 normalization and sign constraints
cvx_begin quiet
    variable x(2*B)
    minimize norm(A*x)
    subject to
        s .* (A*x) >= 0
        s' * (A*x) == n
cvx_end

y_hat = A*x;
fprintf('Recovery error: %f\n', norm(y - y_hat) / norm(y));
figure
plot(y)
hold all
plot(y_hat)
xlim([0,n])
legend('original', 'recovered', 'Location', 'SouthEast');
title('original and recovered bandlimited signals');

```

The following Python code solves the problem:

```

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

```

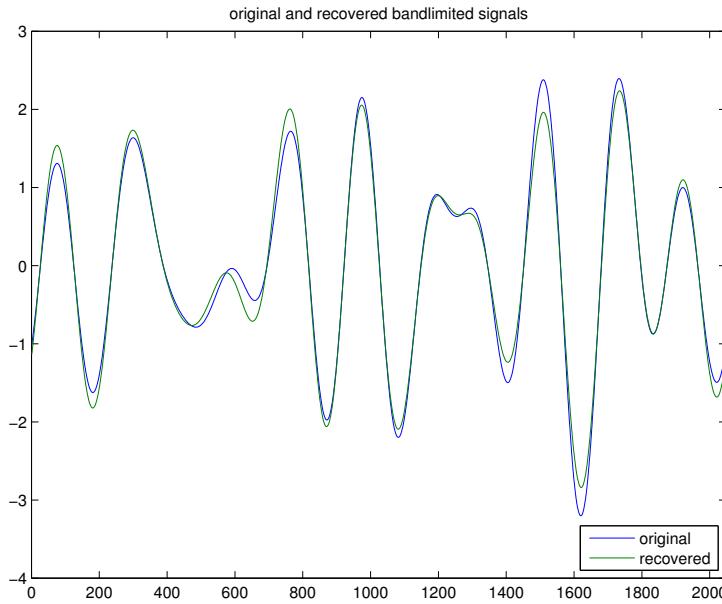


Figure 12: The original bandlimited signal y and the estimate \hat{y} recovered from zero crossings.

```

from zero_crossings_data import *

# Construct matrix A whose columns are bandlimited sinusoids
C = np.zeros((n, B))
S = np.zeros((n, B))
for j in range(B):
    C[:, j] = np.cos(2 * np.pi * (f_min + j) * np.arange(1, n + 1) / n)
    S[:, j] = np.sin(2 * np.pi * (f_min + j) * np.arange(1, n + 1) / n)
A = np.hstack((C, S))

# Minimize norm subject to L1 normalization and sign constraints
x = cvx.Variable(2 * B)
obj = cvx.norm(A * x)
constraints = [cvx.mul_elemwise(s, A * x) >= 0,
               s.T * (A * x) == n]
problem = cvx.Problem(cvx.Minimize(obj), constraints)
problem.solve()

y_hat = np.dot(A, x.value.A1)
print('Recovery error: {}'
      .format(np.linalg.norm(y - y_hat) / np.linalg.norm(y)))
plt.figure()

```

```

plt.plot(np.arange(0, n), y, label='original');
plt.plot(np.arange(0, n), y_hat, label='recovered');
plt.xlim([0, n])
plt.legend(loc='lower left')
plt.show()

```

The following Julia code solves the problem:

```

using Convex, SCS, Gadfly
set_default_solver(SCSSolver(verbose=false))

include("zero_crossings_data.jl")

# Construct matrix A whose columns are bandlimited sinusoids
C = zeros(n, B);
S = zeros(n, B);
for j in 1:B
    C[:, j] = cos(2 * pi * (f_min + j - 1) * (1:n) / n);
    S[:, j] = sin(2 * pi * (f_min + j - 1) * (1:n) / n);
end
A = [C S];

# Minimize norm subject to L1 normalization and sign constraints
x = Variable(2 * B)
obj = norm(A * x, 2)
constraints = [s .* (A * x) >= 0,
               s' * (A * x) == n]
problem = minimize(obj, constraints)
solve!(problem)

y_hat = A * x.value
println("Recovery error: $(norm(y - y_hat) / norm(y))")
pl = plot(
    layer(x=1:n, y=y, Geom.line, Theme(default_color = colorant"blue")),
    layer(x=1:n, y=y_hat, Geom.line, Theme(default_color = colorant"green")),
);
display(pl);

```

13 Finance

13.1 Transaction cost. Consider a market for some asset or commodity, which we assume is infinitely divisible, *i.e.*, can be bought or sold in quantities of shares that are real numbers (as opposed to integers). The *order book* at some time consists of a set of offers to sell or buy the asset, at a given price, up to a given quantity of shares. The N offers to sell the asset have positive prices per share $p_1^{\text{sell}}, \dots, p_N^{\text{sell}}$, sorted in increasing order, in positive share quantities $q_1^{\text{sell}}, \dots, q_N^{\text{sell}}$. The M offers to buy the asset have positive prices $p_1^{\text{buy}}, \dots, p_M^{\text{buy}}$, sorted in decreasing order, and positive quantities $q_1^{\text{buy}}, \dots, q_M^{\text{buy}}$. The price p_1^{sell} is called the (current) *ask price* for the asset; p_1^{buy} is the *bid price* for the asset. The ask price is larger than the bid price; the difference is called the *spread*. The average of the ask and bid prices is called the *mid-price*, denoted p^{mid} .

Now suppose that you want to purchase $q > 0$ shares of the asset, where $q \leq q_1^{\text{sell}} + \dots + q_N^{\text{sell}}$, *i.e.*, your purchase quantity does not exceed the total amount of the asset currently offered for sale. Your purchase proceeds as follows. Suppose that

$$q_1^{\text{sell}} + \dots + q_k^{\text{sell}} < q \leq q_1^{\text{sell}} + \dots + q_{k+1}^{\text{sell}}.$$

Then you pay an amount

$$A = p_1^{\text{sell}} q_1^{\text{sell}} + \dots + p_k^{\text{sell}} q_k^{\text{sell}} + p_{k+1}^{\text{sell}} (q - q_1^{\text{sell}} - \dots - q_k^{\text{sell}}).$$

Roughly speaking, you work your way through the offers in the order book, from the least (ask) price, and working your way up the order book until you fill the order. We define the *transaction cost* as

$$T(q) = A - p^{\text{mid}} q.$$

This is the difference between what you pay, and what you would have paid had you been able to purchase the shares at the mid-price. It is always positive.

We handle the case of selling the asset in a similar way. Here we take $q < 0$ to mean that we sell $-q$ shares of the asset. Here you sell shares at the bid price, up to the quantity q^{buy} (*or* $-q$, whichever is smaller); if needed, you sell shares at the price p_2^{buy} , and so on, until all $-q$ shares are sold. Here we assume that $-q \leq q_1^{\text{buy}} + \dots + q_M^{\text{buy}}$, *i.e.*, you are not selling more shares than the total quantity of offers to buy. Let A denote the amount you receive from the sale. Here we define the transaction cost as

$$T(q) = -p^{\text{mid}} q - A,$$

the difference between the amount you would have received had you sold the shares at the mid-price, and the amount you received. It is always positive. We set $T(0) = 0$.

- (a) Show that T is a convex piecewise linear function.
- (b) Show that $T(q) \geq (s/2)|q|$, where s is the spread. When would we have $T(q) = (s/2)|q|$ for all q (in the range between the total shares offered to purchase or sell)?
- (c) Give an interpretation of the conjugate function $T^*(y) = \sup_q (yq - T(q))$. *Hint.* Suppose you can purchase or sell the asset in another market, at the price p^{other} .

Solution.

(a)

(b)

- (c) Suppose you can purchase (or sell) elsewhere, in another market, any amount of the asset at the price p^{other} . If you buy q shares in the market, and then sell them in the other market, your profit is $qp^{\text{other}} - qp^{\text{mid}} - T(q)$. In fact, this formula works for $q < 0$ as well, which corresponds to buying q shares in the other market, and selling them in the given market. Naturally, you would choose q to maximize your profit. Your optimal profit is $T^*(p^{\text{other}} - p^{\text{mid}})$. So, the conjugate function maps a price differential with respect to another market into the optimal profit you can attain.

13.2 Risk-return trade-off in portfolio optimization. We consider the portfolio risk-return trade-off problem of page 185, with the following data:

$$\bar{p} = \begin{bmatrix} 0.12 \\ 0.10 \\ 0.07 \\ 0.03 \end{bmatrix}, \quad \Sigma = \begin{bmatrix} 0.0064 & 0.0008 & -0.0011 & 0 \\ 0.0008 & 0.0025 & 0 & 0 \\ -0.0011 & 0 & 0.0004 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

- (a) Solve the quadratic program

$$\begin{aligned} & \text{minimize} && -\bar{p}^T x + \mu x^T \Sigma x \\ & \text{subject to} && \mathbf{1}^T x = 1, \quad x \succeq 0 \end{aligned}$$

for a large number of positive values of μ (for example, 100 values logarithmically spaced between 1 and 10^7). Plot the optimal values of the expected return $\bar{p}^T x$ versus the standard deviation $(x^T \Sigma x)^{1/2}$. Also make an area plot of the optimal portfolios x versus the standard deviation (as in figure 4.12).

- (b) Assume the price change vector p is a Gaussian random variable, with mean \bar{p} and covariance Σ . Formulate the problem

$$\begin{aligned} & \text{maximize} && \bar{p}^T x \\ & \text{subject to} && \mathbf{prob}(p^T x \leq 0) \leq \eta \\ & && \mathbf{1}^T x = 1, \quad x \succeq 0, \end{aligned}$$

as a convex optimization problem, where $\eta < 1/2$ is a parameter. In this problem we maximize the expected return subject to a constraint on the probability of a negative return. Solve the problem for a large number of values of η between 10^{-4} and 10^{-1} , and plot the optimal values of $\bar{p}^T x$ versus η . Also make an area plot of the optimal portfolios x versus η .

Hint: The Matlab functions `erfc` and `erfcinv` can be used to evaluate

$$\Phi(x) = (1/\sqrt{2\pi}) \int_{-\infty}^x e^{-t^2/2} dt$$

and its inverse:

$$\Phi(u) = \frac{1}{2} \mathbf{erfc}(-u/\sqrt{2}).$$

Since you will have to solve this problem for a large number of values of η , you may find the command `cvx_quiet(true)` helpful.

- (c) *Monte Carlo simulation.* Let x be the optimal portfolio found in part (b), with $\eta = 0.05$. This portfolio maximizes the expected return, subject to the probability of a loss being no more than 5%. Generate 10000 samples of p , and plot a histogram of the returns. Find the empirical mean of the return samples, and calculate the percentage of samples for which a loss occurs.

Hint: You can generate samples of the price change vector using

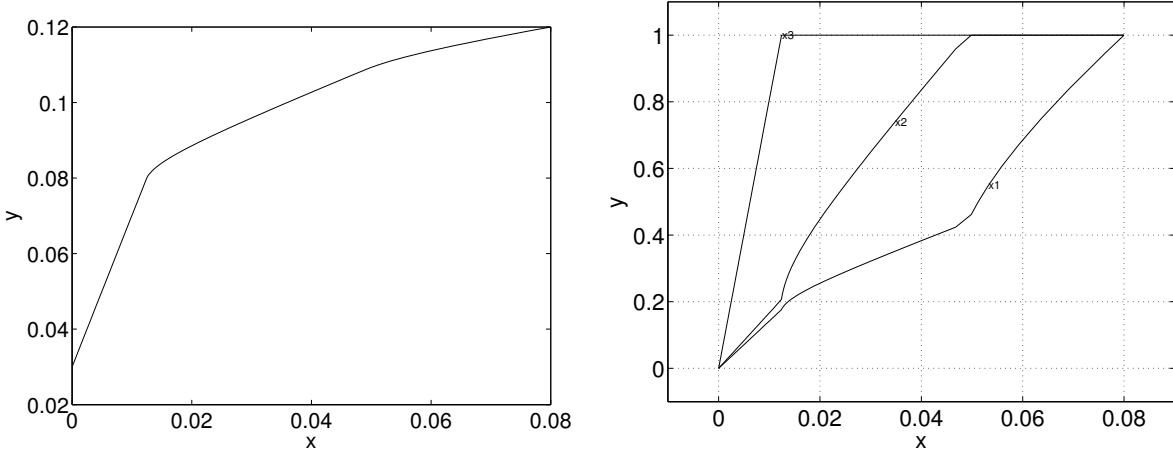
```
p=pbar+sqrtm(Sigma)*randn(4,1);
```

Solution.

- (a) *Risk-return tradeoff.*

```
n=4;
pbar = [.12 .10 .07 .03]';
S = [0.0064 0.0008 -0.0011 0;
      0.0008 0.0025 0 0;
     -0.0011 0 0.0004 0;
      0 0 0 0];
novals = 100;
returns = zeros(1,novals);
stdevs = zeros(1,novals);
pfs = zeros(n,novals);
muvals = logspace(0,7,novals);
for i=1:novals
    mu = muvals(i);
    cvx_begin
        variable x(n);
        minimize (-pbar'*x + mu*quad_form(x,S));
        subject to
            sum(x) == 1;
            x >= 0;
    cvx_end;
    returns(i) = pbar'*x;
    stdevs(i) = sqrt(x'*S*x);
    pfs(n,i)= x;
end;

figure(1)
plot(stdevs,returns);
figure(2)
plot(stdevs, pfs(1,:)');
hold on
plot(stdevs, (pfs(1,:)+pfs(2,:))');
plot(stdevs, (pfs(1,:)+pfs(2,:)+pfs(3,:))');
hold off
axis([-0.01 .09 -0.1 1.1]);
```



(b) *Portfolio optimization with loss constraint.* The problem

$$\begin{aligned} & \text{maximize} && \bar{p}^T x \\ & \text{subject to} && \Phi((\beta - \bar{p}^T x)/\sqrt{x^T \Sigma^{1/2} x}) \leq \eta \\ & && \mathbf{1}^T x = 1, \quad x \succeq 0 \end{aligned}$$

is equivalent to the SOCP

$$\begin{aligned} & \text{maximize} && \bar{p}^T X \\ & \text{subject to} && \bar{p}^T x + \Phi^{-1}(\eta) \|\Sigma^{1/2}\|_2 \geq \beta. \end{aligned}$$

(Note that $\Phi^{-1}(\eta) < 0$ if $\eta > 1/2$.)

```
% compute S^{1/2}
[V,D] = eig(S);
sqrtS = V*diag(sqrt(diag(D)))*V';

beta = 0.0;
novals = 100;
etas = logspace(-4,-1,novals);

% maximize pbar'*x
% subject to Phi((beta-pbar'*x) / sqrt(x'*S*x)) <= eta
%           1'*x = 1, x >= 0
% gamma = Phi^{-1}(eta) where Phi(u) = .5 * erfc(-u/sqrt(2))

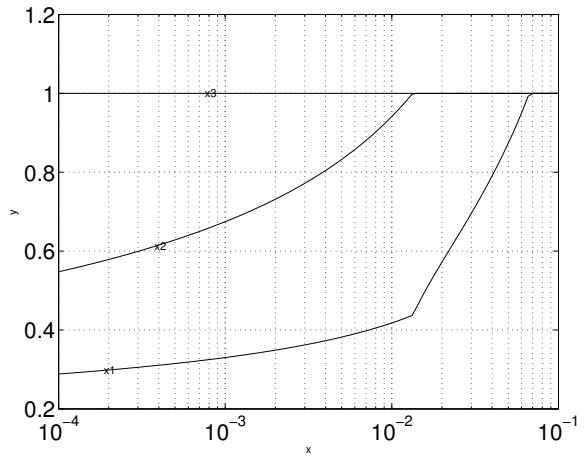
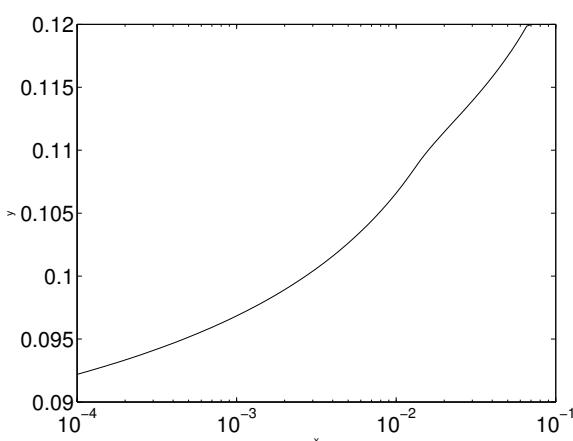
returns = zeros(1,novals);
pfs = zeros(n,novals);
for k = 1:novals
    eta = etas(k);
    gamma = -sqrt(2)*erfcinv(2*eta);
    cvx_begin
        variable x(n)
```

```

maximize(pbar'*x)
subject to
    sum(x) == 1;
    x >= 0;
    pbar'*x + gamma * norm(sqrtS*x,2) >= beta
cvx_end
returns(k) = pbar'*x;
pfs(:,k) = x;
end;

figure(1)
semilogx(etas,returns)
figure(2)
semilogx(etas, pfs(1,:)); hold on
semilogx(etas, (pfs(1,:)+pfs(2,:)));
semilogx(etas, (pfs(1,:)+pfs(2,:)+pfs(3,:)));
grid on
hold off

```



(c) The following code was used for this part of the problem:

```

randn('state',0);

% Number of Monte Carlo samples
N=10000;

eta = 0.05;
gamma = sqrt(2)*erfcinv(2*(1-eta));

% Get maximizing portfolio for eta = 10
cvx_begin
variable x(n)
maximize(pbar'*x)

```

```

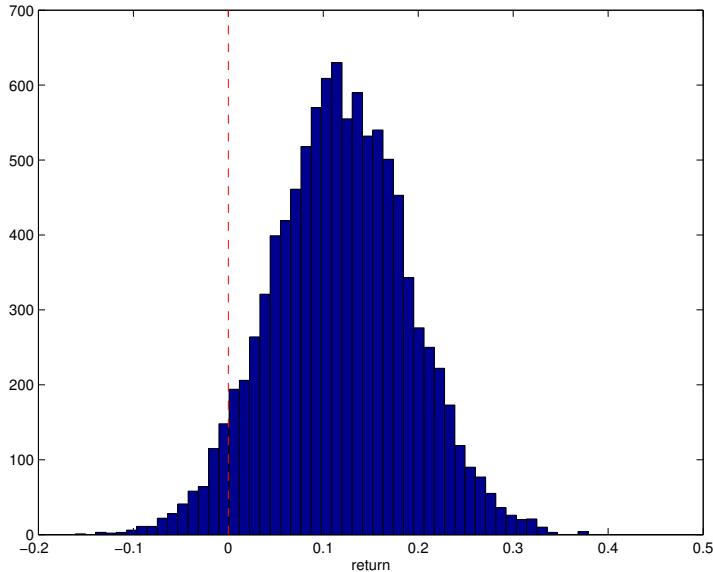
subject to
    sum(x) == 1
    x >= 0
    pbar'*x + gamma * norm(sqrtS*x,2) >= 0
cvx_end

% Monte Carlo simulation
returns = [];
for i = 1:N
    p = pbar+sqrtS*randn(n,1);
    returns = [returns p'*x];
end

% Empirical mean, histogram and percentage of losses
emp_mean = mean(returns)
perc_losses = sum(returns < 0)/N
hist(returns,50)
hold on
plot([0;0],[0;700],'r--')

```

The following figure shows the histogram of the returns:



In this case the empirical mean of the return is 0.1182 and approximately 4.97% of the samples generate a loss.

13.3 Simple portfolio optimization. We consider a portfolio optimization problem as described on pages 155 and 185–186 of *Convex Optimization*, with data that can be found in the file `simple_portfolio_data.*`.

- (a) Find minimum-risk portfolios with the same expected return as the uniform portfolio ($x = (1/n)\mathbf{1}$), with risk measured by portfolio return variance, and the following portfolio constraints (in addition to $\mathbf{1}^T x = 1$):

- No (additional) constraints.
- Long-only: $x \succeq 0$.
- Limit on total short position: $\mathbf{1}^T(x_-) \leq 0.5$, where $(x_-)_i = \max\{-x_i, 0\}$.

Compare the optimal risk in these portfolios with each other and the uniform portfolio.

- (b) Plot the optimal risk-return trade-off curves for the long-only portfolio, and for total short-position limited to 0.5, in the same figure. Follow the style of figure 4.12 (top), with horizontal axis showing standard deviation of portfolio return, and vertical axis showing mean return.

Solution.

- (a) We can express these as QPs:

- No (additional) constraints:

$$\begin{aligned} & \text{minimize} && x^T \Sigma x \\ & \text{subject to} && \mathbf{1}^T x = 1, \quad \bar{p}^T x = \bar{p}^T (1/n) \mathbf{1} \end{aligned}$$

- Long only

$$\begin{aligned} & \text{minimize} && x^T \Sigma x \\ & \text{subject to} && x \succeq 0, \quad \mathbf{1}^T x = 1 \\ & && \bar{p}^T x = \bar{p}^T (1/n) \mathbf{1} \end{aligned}$$

- Limit on total short position:

$$\begin{aligned} & \text{minimize} && x^T \Sigma x \\ & \text{subject to} && \mathbf{1}^T x = 1, \quad \bar{p}^T x = \bar{p}^T (1/n) \mathbf{1} \\ & && \mathbf{1}^T x_- \leq 0.5 \end{aligned}$$

Although the two portfolios have the same expected return, their risk profiles differ drastically. The standard deviations are:

- uniform: 8.7%
- long-only: 5.1%
- limit on total short position: 2.1%
- unconstrained: 1.9%

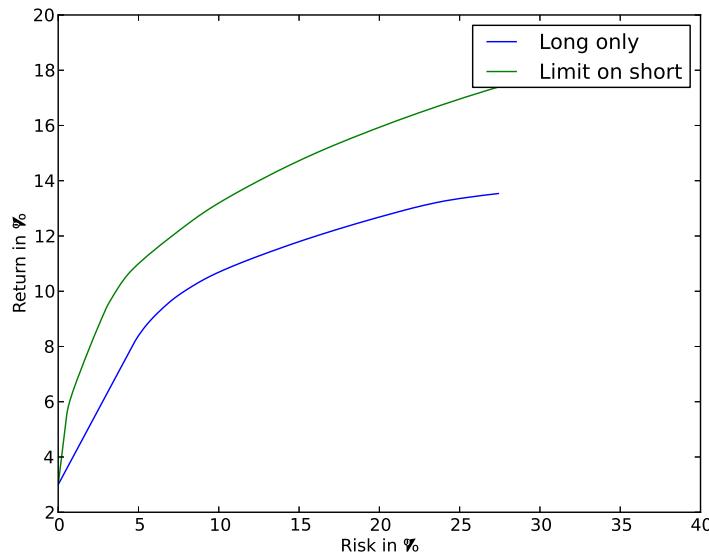
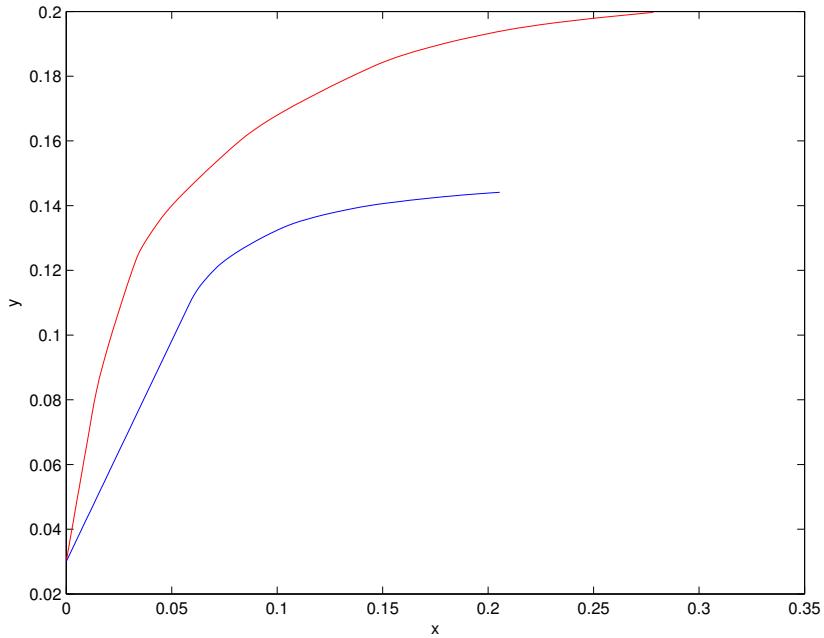
for the MATLAB data, and

- uniform: 27.00%
- long-only: 3.95%
- limit on total short position: 1.50%
- unconstrained: 0.9%

for the Python data. Notice that as the size of the feasible set increases, the objective value improves.

- (b) The optimal risk-return trade-off curves can be generated by scalarizing the bicriterion objective, $\bar{p}^T x - \gamma x^T \Sigma x$, over a range of values for γ . For example, the optimal curve for the long-only portfolio is generated by solving the following family of QPs:

$$\begin{aligned} & \text{maximize} && \bar{p}^T x - \gamma x^T \Sigma x \\ & \text{subject to} && x \succeq 0, \quad \mathbf{1}^T x = 1 \end{aligned}$$



For every level of return, there is a portfolio with limits on total short positions (red) that has greater expected return than the optimal long-only portfolio (blue).

The following CVX code solves this problem:

```
simple_portfolio_data;
%% part i
%minimum-risk unconstrained portfolio with same expected return as uniform
%allocation
cvx_begin
cvx_quiet(true)
```

```

variable x_unconstrained(n)
minimize(quad_form(x_unconstrained,S))
subject to
    sum(x_unconstrained)==1;
    pbar'*x_unconstrained==x_unif'*pbar;
cvx_end
%% part ii
%minimum-risk long-only portfolio with same expected return as uniform
%allocation
cvx_begin
cvx_quiet(true)
variable x_long(n)
minimize(quad_form(x_long,S))
subject to
    x_long>=0;
    sum(x_long)==1;
    pbar'*x_long==x_unif'*pbar;
cvx_end
%% part iii
%minimum-risk constrained short portfolio with same expected return as uniform
%allocation
cvx_begin
cvx_quiet(true)
variable x_shortconstr(n)
minimize(quad_form(x_shortconstr,S))
subject to
    sum(pos(-x_shortconstr))<=0.5;
    sum(x_shortconstr)==1;
    pbar'*x_shortconstr==x_unif'*pbar;
cvx_end
%% Generate risk-return trade-off curves
sprintf('unconstrained sd: %0.3g\n', sqrt(quad_form(x_unconstrained,S)))
sprintf('long only sd: %0.3g\n', sqrt(quad_form(x_long,S)))
sprintf('constrained short sd: %0.3g\n', sqrt(quad_form(x_shortconstr,S)))
sprintf('x_unif sd: %0.3g\n', sqrt(quad_form(x_unif,S)))
novals=100;
r_long = [];
r_shortconstr = [];
sd_long = [];
sd_shortconstr = [];
muvals = logspace(-1,4,novals);

for i=1:novals
    mu = muvals(i);
    %long only

```

```

cvx_begin
cvx_quiet(true)
variable x(n)
maximize(pbar'*x - mu*quad_form(x,S))
subject to
    x>=0;
    sum(x)==1;
cvx_end
r_long = [r_long, pbar'*x];
sd_long = [sd_long, sqrt(x'*S*x) ];
%constrained short
cvx_begin
cvx_quiet(true)
variables x(n)
maximize(pbar'*x - mu*quad_form(x,S))
subject to
    sum(x)==1;
    sum(pos(-x))<=.5;
cvx_end
r_shortconstr = [r_shortconstr, pbar'*x];
sd_shortconstr = [sd_shortconstr, sqrt(x'*S*x)];
end;

plot(sd_long, r_long);
hold; plot(sd_shortconstr, r_shortconstr, 'r');

```

The following CVXPY code implements this:

```

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

np.random.seed(1)
n = 20
pbar = np.ones((n,1))*0.03 + np.r_[np.random.rand(n-1,1), np.zeros((1,1))]*.12;
S = np.random.randn(n, n); S = np.asmatrix(S)
S = S.T*S
S = S/max(np.abs(np.diag(S)))*.2
S[:, -1] = np.zeros((n, 1))
S[-1, :] = np.zeros((n, 1)).T
x_unif = np.ones((n, 1))/n; x_unit = np.asmatrix(x_unif)

x = cvx.Variable(n)
risk = cvx.quad_form(x,S)
constraints = [cvx.sum_entries(x)==1, pbar.T*x==sum(pbar)/n]

```

```

#Uniform portfolio
print 'Risk for uniform: %.2f%%' % float(np.sqrt(np.sum(pbar)/n)*100)

#No additional constraints
cvx.Problem(cvx.Minimize(risk), constraints).solve()
print 'Risk for unconstrained: %.2f%%' % float(np.sqrt(risk.value)*100)

#Long only
cvx.Problem(cvx.Minimize(risk), constraints + [x>=0]).solve()
print 'Risk for long only: %.2f%%' % float(np.sqrt(risk.value)*100)

#Limit on total short position
cvx.Problem(cvx.Minimize(risk), constraints\
           + [cvx.sum_entries(cvx.neg(x))<=0.5]).solve()
print 'Risk for limit on short: %.2f%%' % float(np.sqrt(risk.value)*100)

gamma = cvx.Parameter(sign='positive')
expec_return = pbar.T*x
prob = cvx.Problem(cvx.Maximize(expec_return-gamma*risk), [])

N = 128

#Long only
gamma_vals = np.logspace(-1,5,num=N)
return_vec1 = np.zeros((N,1))
risk_vec1 = np.zeros((N,1))
prob.constraints = [cvx.sum_entries(x)==1, x>=0]
for i in range(N):
    gamma.value = gamma_vals[i]
    prob.solve()
    return_vec1[i] = expec_return.value
    risk_vec1[i] = risk.value

plt.figure()
plt.plot(np.sqrt(risk_vec1)*100, return_vec1*100, label='Long only')

#Limit on Short
return_vec2 = np.zeros((N,1))
risk_vec2 = np.zeros((N,1))
prob.constraints = [cvx.sum_entries(x)==1, cvx.sum_entries(cvx.neg(x))<=0.5]
for i in range(N):
    gamma.value = gamma_vals[i]
    prob.solve()
    return_vec2[i] = expec_return.value

```

```

risk_vec2[i] = risk.value

plt.plot(np.sqrt(risk_vec2)*100, return_vec2*100, label='Limit on short')
plt.legend()
plt.xlabel('Risk in %')
plt.ylabel('Return in %')
plt.savefig('simple_portfolio.eps')
plt.show()

```

The following Convex.jl code implements this:

```

using Convex, Gadfly
include("simple_portfolio_data.jl")

# uniform portfolio
println("uniform sd: ", sqrt(x_unif'*S*x_unif));

# part i
# minimum-risk unconstrained portfolio with same expected return as uniform
# allocation
x_unconstrained = Variable(n);
p = minimize(quadform(x_unconstrained, S));
p.constraints += sum(x_unconstrained) == 1;
p.constraints += pbar'*x_unconstrained == pbar'*x_unif;
solve!(p);
println("unconstrained sd: ", sqrt(p.optval));

# part ii
# minimum-risk long-only portfolio with same expected return as uniform
# allocation
x_long = Variable(n);
p = minimize(quadform(x_long, S));
p.constraints += x_long >= 0;
p.constraints += sum(x_long) == 1;
p.constraints += pbar'*x_long == pbar'*x_unif;
solve!(p);
println("long only sd: ", sqrt(p.optval));

# part iii
# minimum-risk constrained short portfolio with same expected return as uniform
# allocation
x_shortconstr = Variable(n);
p = minimize(quadform(x_shortconstr, S));
p.constraints += sum(pos(-x_shortconstr)) <= 0.5;
p.constraints += sum(x_shortconstr) == 1;
p.constraints += pbar'*x_shortconstr == pbar'*x_unif;
solve!(p);

```

```

    println("constrained short sd: ", sqrt(p.optval));

    # Generate risk-return trade-off curves
    novals=50;
    r_long = [];
    r_shortconstr = [];
    sd_long = [];
    sd_shortconstr = [];
    muvals = logspace(-1,3,novals);

    for i=1:novals
        mu = muvals[i];

        # long only
        x = Variable(n);
        ret = pbar'*x;
        risk = quadform(x, S);
        p = maximize(ret - mu*risk);
        p.constraints += x >= 0;
        p.constraints += sum(x) == 1;
        solve!(p);
        r_long = [r_long; evaluate(ret)];
        sd_long = [sd_long; sqrt(evaluate(risk))];

        # constrained short
        x = Variable(n);
        ret = pbar'*x;
        risk = quadform(x, S);
        p = maximize(ret - mu*risk);
        p.constraints += sum(pos(-x)) <= 0.5;
        p.constraints += sum(x) == 1;
        solve!(p);
        r_shortconstr = [r_shortconstr; evaluate(ret)];
        sd_shortconstr = [sd_shortconstr; sqrt(evaluate(risk))];
    end

    plot(
        layer(x=sd_long, y=r_long, Geom.line,
              Theme(default_color=color("red"))),
        layer(x=sd_shortconstr, y=r_shortconstr, Geom.line,
              Theme(default_color=color("blue")))
    )

```

13.4 Bounding portfolio risk with incomplete covariance information. Consider the following instance of the problem described in §4.6, on p171–173 of *Convex Optimization*. We suppose that Σ_{ii} , which

are the squares of the price volatilities of the assets, are known. For the off-diagonal entries of Σ , all we know is the sign (or, in some cases, nothing at all). For example, we might be given that $\Sigma_{12} \geq 0$, $\Sigma_{23} \leq 0$, etc. This means that we do not know the correlation between p_1 and p_2 , but we do know that they are nonnegatively correlated (*i.e.*, the prices of assets 1 and 2 tend to rise or fall together).

Compute σ_{wc}^2 , the worst-case variance of the portfolio return, for the specific case

$$x = \begin{bmatrix} 0.1 \\ 0.2 \\ -0.05 \\ 0.1 \end{bmatrix}, \quad \Sigma = \begin{bmatrix} 0.2 & + & + & \pm \\ + & 0.1 & - & - \\ + & - & 0.3 & + \\ \pm & - & + & 0.1 \end{bmatrix},$$

where a “+” entry means that the element is nonnegative, a “−” means the entry is nonpositive, and “ \pm ” means we don’t know anything about the entry. (The negative value in x represents a *short position*: you sold stocks that you didn’t have, but must produce at the end of the investment period.) In addition to σ_{wc}^2 , give the covariance matrix Σ_{wc} associated with the maximum risk. Compare the worst-case risk with the risk obtained when Σ is diagonal.

Solution. We can solve the problem with the following CVXCVXPY code:

```

x = [0.1 0.2 -0.05 0.1]';

cvx_begin
    variable Sigma(4,4) symmetric
    maximize(x'*Sigma*x)
    subject to
        Sigma(1,1) == 0.2; Sigma(2,2) == 0.1;
        Sigma(3,3) == 0.3; Sigma(4,4) == 0.1;
        Sigma(1,2) >= 0; Sigma(1,3) >= 0;
        Sigma(2,3) <= 0; Sigma(2,4) <= 0;
        Sigma(3,4) >= 0;
        Sigma == semidefinite(4);
cvx_end

s_wc = sqrt(cvx_optval)

import cvxpy as cvx
import numpy as np

x = np.matrix('0.1; 0.2; -0.05; 0.1');

Sigma = cvx.semidefinite(4)
constraints = [Sigma[0,0]==0.2, Sigma[1,1]==0.1]
constraints += [Sigma[2,2]==0.3, Sigma[3,3]==0.1]
constraints += [Sigma[0,1]>=0, Sigma[0,2]>=0]
constraints += [Sigma[1,2]<=0, Sigma[1,3]<=0, Sigma[2,3]>=0]
objective = cvx.Maximize(x.T*Sigma*x)

```

```

cvx.Problem(objective, constraints).solve()

sigma_wc = np.sqrt(objective.value)

```

Running this script we get the optimal value to be 0.0151662, that is, we get $\sigma_{wc}^2 = 0.1232$. The associated Σ matrix is given by

$$\Sigma = \begin{bmatrix} 0.2000 & 0.0790 & 0.0000 & 0.1118 \\ 0.0790 & 0.1000 & -0.1387 & 0.0000 \\ 0.0000 & -0.1387 & 0.3000 & 0.0752 \\ 0.1118 & 0.0000 & 0.0752 & 0.1000 \end{bmatrix}$$

in Matlab,

$$\Sigma = \begin{bmatrix} 0.2000 & 0.0847 & 0.0000 & 0.1003 \\ 0.0847 & 0.1000 & -0.1273 & 0.0000 \\ 0.0000 & -0.1273 & 0.3000 & 0.0522 \\ 0.1003 & 0.0000 & 0.0522 & 0.1000 \end{bmatrix}$$

in Python, and

$$\Sigma = \begin{bmatrix} 0.2000 & 0.0979 & 0.0000 & 0.0741 \\ 0.0978 & 0.1000 & -0.1010 & 0.0000 \\ 0.0000 & -0.1010 & 0.3000 & 0.0000 \\ 0.0741 & 0.0000 & 0.0000 & 0.1000 \end{bmatrix}$$

in Julia.

13.5 Log-optimal investment strategy. In this problem you will solve a specific instance of the log-optimal investment problem described in exercise 4.60, with $n = 5$ assets and $m = 10$ possible outcomes in each period. The problem data are defined in `log_opt_invest.*`, with the rows of the matrix P giving the asset return vectors p_j^T . The outcomes are equiprobable, *i.e.*, we have $\pi_j = 1/m$. Each column of the matrix P gives the return of the associated asset in the different possible outcomes. You can examine the columns to get an idea of the types of assets. For example, the last asset gives a fixed and certain return of 1%; the first asset is a very risky one, with occasional large return, and (more often) substantial loss.

Find the log-optimal investment strategy x^* , and its associated long term growth rate R_{lt}^* . Compare this to the long term growth rate obtained with a uniform allocation strategy, *i.e.*, $x = (1/n)\mathbf{1}$, and also with a pure investment in each asset.

For the optimal investment strategy, and also the uniform investment strategy, plot 10 sample trajectories of the accumulated wealth, *i.e.*, $W(T) = W(0) \prod_{t=1}^T \lambda(t)$, for $T = 0, \dots, 200$, with initial wealth $W(0) = 1$.

To save you the trouble of figuring out how to simulate the wealth trajectories or plot them nicely, we've included the simulation and plotting code in `log_opt_invest.*`; you just have to add the code needed to find x^* .

Hint (MATLAB users only): The current version of CVX handles the logarithm via an iterative method, which can be slow and unreliable. You're better off using `geo_mean()`, which is directly handled by CVX, to solve the problem.

Solution.

(a) The following MATLAB code was used to solve this problem:

```
clear all

P = [3.5000    1.1100    1.1100    1.0400    1.0100;
      0.5000    0.9700    0.9800    1.0500    1.0100;
      0.5000    0.9900    0.9900    0.9900    1.0100;
      0.5000    1.0500    1.0600    0.9900    1.0100;
      0.5000    1.1600    0.9900    1.0700    1.0100;
      0.5000    0.9900    0.9900    1.0600    1.0100;
      0.5000    0.9200    1.0800    0.9900    1.0100;
      0.5000    1.1300    1.1000    0.9900    1.0100;
      0.5000    0.9300    0.9500    1.0400    1.0100;
      3.5000    0.9900    0.9700    0.9800    1.0100];

[m,n] = size(P);

% Find log-optimal investment policy
cvx_begin
    variable x_opt(n)
    maximize(geomean(P*x_opt))
    sum(x_opt) == 1
    x_opt >= 0
cvx_end

x_opt
x_unif = ones(n,1)/n;
R_opt = sum(log(P*x_opt))/m
R_unif = sum(log(P*x_unif))/m
```

The following Python code solves the problem:

```
import numpy as np
import cvxpy as cvx
from scipy.stats import gmean
from cmath import log

P = np.array(np.mat(
    '3.5000    1.1100    1.1100    1.0400    1.0100;\\
     0.5000    0.9700    0.9800    1.0500    1.0100;\\
     0.5000    0.9900    0.9900    0.9900    1.0100;\\
     0.5000    1.0500    1.0600    0.9900    1.0100;\\
     0.5000    1.1600    0.9900    1.0700    1.0100;\\
     0.5000    0.9900    0.9900    1.0600    1.0100;\\
     0.5000    0.9200    1.0800    0.9900    1.0100;\\
     0.5000    1.1300    1.1000    0.9900    1.0100;\\
     0.5000    0.9300    0.9500    1.0400    1.0100;\\
     3.5000    0.9900    0.9700    0.9800    1.0100;\\
'))
```

```

3.5000    0.9900    0.9700    0.9800    1.0100')))

m=P.shape[0];
n=P.shape[1];
x_unif = np.ones(n)/n; # uniform resource allocation

x_opt = cvx.Variable(n)
objective = cvx.Maximize(cvx.sum_entries(cvx.log(P*x_opt)))
constraints = [ cvx.sum_entries(x_opt) == 1,
x_opt >= 0]
prob = cvx.Problem(objective, constraints)
result = prob.solve()

x_opt = x_opt.value
print "status:", prob.status
print "Log-optimal investment strategy:"
print x_opt
print "Optimal long term growth rate:" , prob.value/m
print "Long term growth associated with uniform allocation strategy:",
print np.log(gmean(P.dot(x_unif)))

```

The following Julia code solves the problem:

```

P = [3.5000    1.1100    1.1100    1.0400    1.0100;
      0.5000    0.9700    0.9800    1.0500    1.0100;
      0.5000    0.9900    0.9900    0.9900    1.0100;
      0.5000    1.0500    1.0600    0.9900    1.0100;
      0.5000    1.1600    0.9900    1.0700    1.0100;
      0.5000    0.9900    0.9900    1.0600    1.0100;
      0.5000    0.9200    1.0800    0.9900    1.0100;
      0.5000    1.1300    1.1000    0.9900    1.0100;
      0.5000    0.9300    0.9500    1.0400    1.0100;
      3.5000    0.9900    0.9700    0.9800    1.0100];

```

```

m,n = size(P);
x_unif = ones(n)/n; # uniform resource allocation

using Convex, SCS

# Find log-optimal investment policy
x_rob = Variable(n);
constraint = sum(x_rob) == 1;
constraint += x_rob >=0;
problem = maximize(sum(log(P*x_rob)), constraint);
solve!(problem);

# Nominal cost of x_rob

```

```

x_opt = x_rob.value;
println("Log-optimal investment strategy:")
println(x_opt)
R_opt = sum(log(P*x_opt))/m
println("Optimal long term growth rate:")
println(R_opt)
R_unif = sum(log(P*x_unif))/m
println("Long term growth associated with uniform allocation strategy:")
println(R_unif)

```

It was found that the log-optimal investment strategy is:

$$x_{\text{opt}} = (0.0580, 0.4000, 0.2923, 0.2497, 0.0000).$$

This strategy achieves a long term growth rate $R_{\text{lt}}^* = 0.0231$. In contrast, the uniform allocation strategy achieves a growth rate of $R_{\text{unif}} = 0.0114$.

Clearly asset 1 is a high-risk asset. The amount that we invest in this asset will grow by a factor of 3.50 with probability 20% and will be halved with probability 80%. On the other hand, asset 5 is an asset with a certain return of 1% per time period. Finally, assets 2, 3 and 4 are low-risk assets. It turns out that the log-optimal policy in this case is to invest very little wealth in the high-risk asset and no wealth on the sure asset and to invest most of the wealth in asset 2.

- (b) We show the scripts to generate the random event sequences and the trajectory plots. Since the trajectories depend on the specific random numbers generated, we show the plots for all languages, for completeness. In MATLAB:

```

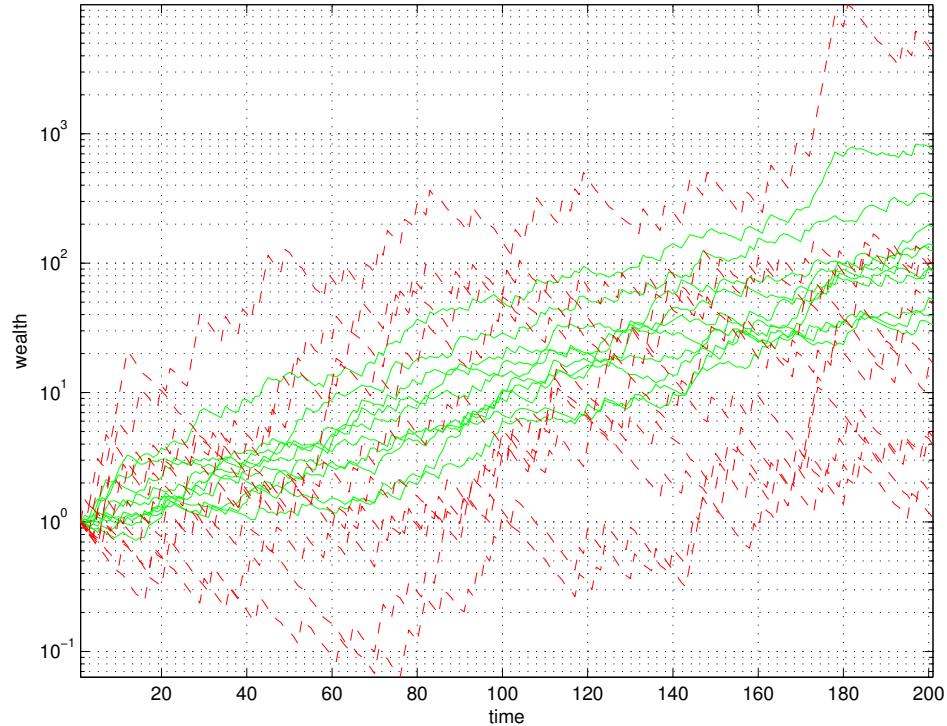
% Generate random event sequences
rand('state',10);
N = 10; % number of random trajectories
T = 200; % time horizon
w_opt = [] ; w_unif = [] ;
for i = 1:N
    events = ceil(rand(1,T)*m);
    P_event = P(events,:);
    w_opt = [w_opt [1; cumprod(P_event*x_opt)]] ;
    w_unif = [w_unif [1; cumprod(P_event*x_unif)]] ;
end

% Plot wealth versus time
figure
semilogy(w_opt,'g')
hold on
semilogy(w_unif,'r--')
grid
axis tight
xlabel('time')

```

```
ylabel('wealth')
```

This generates the following plot:



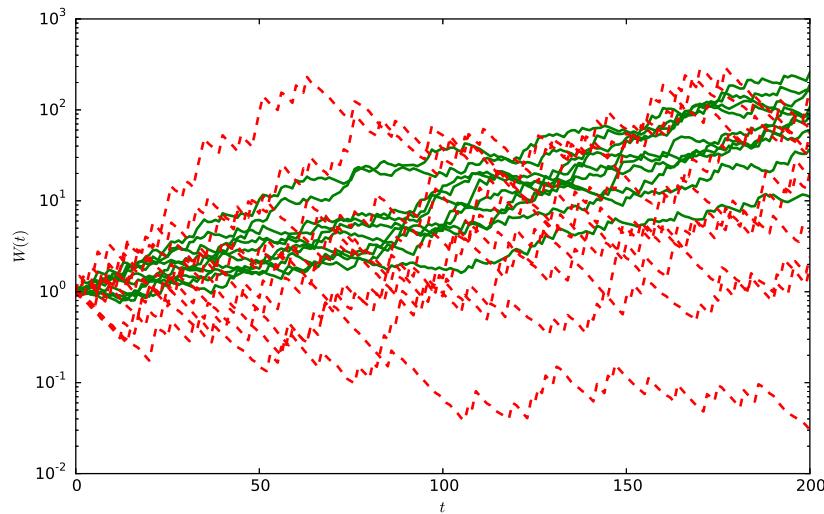
In Python:

```
import matplotlib.pyplot as plt
np.random.seed(10);
N = 10; # number of random trajectories
T = 200; # time horizon
w_opt = np.zeros((N,T+1))
w_unif = np.zeros((N,T+1))
for i in range(N):
    events = np.floor(np.random.rand(T)*m)
    events = events.astype(int)
    P_event = P[events,:]
    w_opt[i,:] = np.append(1, np.cumprod(P_event.dot(x_opt)))
    w_unif[i,:] = np.append(1, np.cumprod(P_event.dot(x_unif)))

plt.figure(figsize=(10, 6), dpi=80)
plt.xlabel('$t$')
plt.ylabel('$W(t)$')
plt.gca().set_yscale('log')
plt.plot(range(T+1), np.transpose(w_opt), color="green", linewidth=2.0)
plt.plot(range(T+1), np.transpose(w_unif), color="red", linewidth=2.0, linestyle='--')
plt.savefig('log_opt_invest.eps')
```

```
plt.show()
```

This generates the following plot:



In Julia:

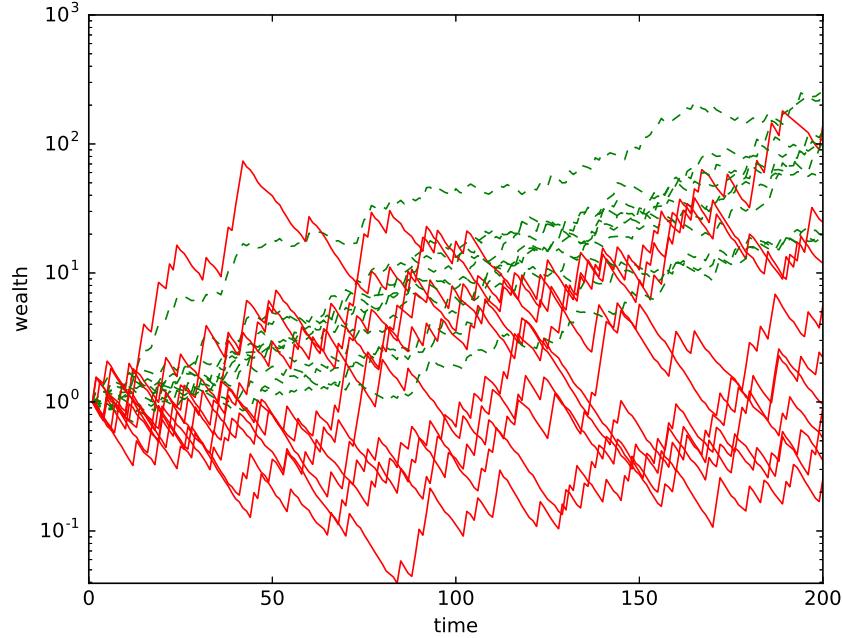
```
using PyPlot, Distributions

srand(10);
N = 10; # number of random trajectories
t = 200; # time horizon
w_opt = zeros(N,t+1);
w_unif = zeros(N,t+1);
for i = 1:N
events = vec(ceil(rand(1,t)*m))
events = round(Int64, events)
P_event = P[events,:];
w_opt[i, :] = [1.0 ; cumprod(P_event*x_opt)];
w_unif[i, :] = [1.0 ; cumprod(P_event*x_unif)];
end

# Plot wealth versus time
figure();
hold("true");
for i = 1:N
semilogy(1:t+1, vec(w_opt[i,:]), "g", linestyle="--");
semilogy(1:t+1, vec(w_unif[i,:]), "r");
end
axis([0, t, 0, 1000])
xlabel("time");
ylabel("wealth");
```

```
savefig("log_opt_invest.eps");
```

This generates the following plot:



The log-optimal investment policy consistently increases the wealth. On the other hand the uniform allocation policy generates quite random trajectories, a few with very large increases in wealth, and many with poor performance. This is due to the fact that with this policy 20% of the wealth is invested in the high-risk asset.

13.6 Optimality conditions and dual for log-optimal investment problem.

- (a) Show that the optimality conditions for the log-optimal investment problem described in exercise 4.60 can be expressed as: $\mathbf{1}^T x = 1$, $x \succeq 0$, and for each i ,

$$x_i > 0 \Rightarrow \sum_{j=1}^m \pi_j \frac{p_{ij}}{p_j^T x} = 1, \quad x_i = 0 \Rightarrow \sum_{j=1}^m \pi_j \frac{p_{ij}}{p_j^T x} \leq 1.$$

We can interpret this as follows. $p_{ij}/p_j^T x$ is a random variable, which gives the ratio of the investment gain with asset i only, to the investment gain with our mixed portfolio x . The optimality condition is that, for each asset we invest in, the expected value of this ratio is one, and for each asset we do not invest in, the expected value cannot exceed one. Very roughly speaking, this means our portfolio does as well as any of the assets that we choose to invest in, and cannot do worse than any assets that we do not invest in.

Hint. You can start from the simple criterion given in §4.2.3 or the KKT conditions.

- (b) In this part we will derive the dual of the log-optimal investment problem. We start by writing the problem as

$$\begin{aligned} & \text{minimize} && -\sum_{j=1}^m \pi_j \log y_j \\ & \text{subject to} && y = P^T x, \quad x \succeq 0, \quad \mathbf{1}^T x = 1. \end{aligned}$$

Here, P has columns p_1, \dots, p_m , and we have introduced new variables y_1, \dots, y_m , with the implicit constraint $y \succ 0$. We will associate dual variables ν, λ and ν_0 with the constraints $y = P^T x$, $x \succeq 0$, and $\mathbf{1}^T x = 1$, respectively. Defining $\tilde{\nu}_j = \nu_j/\nu_0$ for $j = 1, \dots, m$, show that the dual problem can be written as

$$\begin{aligned} & \text{maximize} && \sum_{j=1}^m \pi_j \log(\tilde{\nu}_j/\pi_j) \\ & \text{subject to} && P\tilde{\nu} \preceq \mathbf{1}, \end{aligned}$$

with variable $\tilde{\nu}$. The objective here is the (negative) Kullback-Leibler divergence between the given distribution π and the dual variable $\tilde{\nu}$.

Solution.

- (a) The problem is the same as minimizing $f(x) = -\sum_{j=1}^m \pi_j \log(p_j^T x)$ over the probability simplex. If x is feasible the optimality condition is that $\nabla f(x)^T(z - x) \geq 0$ for all z with $z \succeq 0$, $\mathbf{1}^T z = 1$. This holds if and only if for each k ,

$$x_k > 0 \Rightarrow \frac{\partial f}{\partial x_k} = \min_{i=1, \dots, n} \frac{\partial f}{\partial x_i} = \nabla f(x)^T x.$$

From $f(x) = -\sum_{j=1}^m \pi_j \log(p_j^T x)$, we get

$$\nabla f(x)_i = -\sum_{j=1}^m \pi_j (1/p_j^T x) p_{ij},$$

so

$$\nabla f(x)^T x = -\sum_{j=1}^m \pi_j (1/p_j^T x) p_j^T x = -1.$$

The optimality conditions are, $\mathbf{1}^T x = 1$, $x \succeq 0$, and for each i ,

$$x_i > 0 \Rightarrow \sum_{j=1}^m \pi_j \frac{p_{ij}}{p_j^T x} = 1, \quad x_i = 0 \Rightarrow \sum_{j=1}^m \pi_j \frac{p_{ij}}{p_j^T x} \leq 1.$$

- (b) The Lagrangian is

$$L(x, \nu, \lambda, \nu_0) = -\sum_{j=1}^m \pi_j \log y_j + \nu^T (y - P^T x) - \lambda^T x + \nu_0 (\mathbf{1}^T x - 1).$$

This is unbounded below unless $-\nu^T P^T - \lambda^T + \nu_0 \mathbf{1}^T = 0$. Since L is separable in y we can minimize it over y by minimizing over y_j . We find the minimum is obtained for $y_j = \pi_j/\nu_j$, so we have

$$g(\nu, \lambda, \nu_0) = 1 - \nu_0 + \sum_{j=1}^m \pi_j \log(\nu_j/\pi_j),$$

provided $P\nu \preceq \nu_0 \mathbf{1}$. Thus we can write the dual problem as

$$\begin{aligned} & \text{maximize} && 1 - \nu_0 + \sum_{j=1}^m \pi_j \log(\nu_j/\pi_j) \\ & \text{subject to} && P\nu \preceq \nu_0 \mathbf{1} \end{aligned}$$

with variables $\nu \in \mathbf{R}^m$, $\nu_0 \in \mathbf{R}$. This has implicit constraint $\nu \succ 0$.

We can further simplify this, by analytically optimizing over ν_0 . From the constraint inequality we see that $\nu_0 > 0$. Defining $\tilde{\nu} = \nu/\nu_0$, we get the problem in variables $\tilde{\nu}$, ν_0

$$\begin{aligned} &\text{maximize} && 1 - \nu_0 + \sum_{j=1}^m \pi_j \log(\tilde{\nu}_j \nu_0 / \pi_j) \\ &\text{subject to} && \nu_0 P \tilde{\nu} \preceq \nu_0 \mathbf{1}. \end{aligned}$$

Cancelling ν_0 from the constraint we get

$$\begin{aligned} &\text{maximize} && 1 - \nu_0 + \log \nu_0 + \sum_{j=1}^m \pi_j \log(\tilde{\nu}_j / \pi_j) \\ &\text{subject to} && P \tilde{\nu} \preceq \mathbf{1}. \end{aligned}$$

The optimal value of ν_0 is evidently $\nu_0 = 1$, so we end up with the dual problem

$$\begin{aligned} &\text{maximize} && \sum_{j=1}^m \pi_j \log(\tilde{\nu}_j / \pi_j) \\ &\text{subject to} && P \tilde{\nu} \preceq \mathbf{1}. \end{aligned}$$

13.7 Arbitrage and theorems of alternatives. Consider an event (for example, a sports game, political elections, the evolution of the stock market over a certain period) with m possible outcomes. Suppose that n wagers on the outcome are possible. If we bet an amount x_j on wager j , and the outcome of the event is i ($i = 1, \dots, m$), then our return will be equal to $r_{ij}x_j$. The return $r_{ij}x_j$ is the net gain: we pay x_j initially, and receive $(1 + r_{ij})x_j$ if the outcome of the event is i . We allow the bets x_j to be positive, negative, or zero. The interpretation of a negative bet is as follows. If $x_j < 0$, then initially we *receive* an amount of money $|x_j|$, with an obligation to *pay* $(1 + r_{ij})|x_j|$ if outcome i occurs. In that case, we lose $r_{ij}|x_j|$, *i.e.*, our net is gain $r_{ij}x_j$ (a negative number).

We call the matrix $R \in \mathbf{R}^{m \times n}$ with elements r_{ij} the *return matrix*. A *betting strategy* is a vector $x \in \mathbf{R}^n$, with as components x_j the amounts we bet on each wager. If we use a betting strategy x , our total return in the event of outcome i is equal to $\sum_{j=1}^n r_{ij}x_j$, *i.e.*, the i th component of the vector Rx .

- (a) *The arbitrage theorem.* Suppose you are given a return matrix R . Prove the following theorem: there is a betting strategy $x \in \mathbf{R}^n$ for which

$$Rx \succ 0$$

if and only if there exists no vector $p \in \mathbf{R}^m$ that satisfies

$$R^T p = 0, \quad p \succeq 0, \quad p \neq 0.$$

We can interpret this theorem as follows. If $Rx \succ 0$, then the betting strategy x guarantees a positive return for all possible outcomes, *i.e.*, it is a sure-win betting scheme. In economics, we say there is an *arbitrage opportunity*.

If we normalize the vector p in the second condition, so that $\mathbf{1}^T p = 1$, we can interpret it as a probability vector on the outcomes. The condition $R^T p = 0$ means that

$$\mathbf{E} Rx = p^T Rx = 0$$

for all x , *i.e.*, the expected return is zero for all betting strategies. In economics, p is called a risk neutral probability.

We can therefore rephrase the arbitrage theorem as follows: There is no sure-win betting strategy (or arbitrage opportunity) if and only if there is a probability vector on the outcomes that makes all bets fair (*i.e.*, the expected gain is zero).

Country	Odds
Holland	3.5
Italy	5.0
Spain	5.5
France	6.5
Germany	7.0
England	10.0
Belgium	14.0
Sweden	16.0

Country	Odds
Czech Republic	17.0
Romania	18.0
Yugoslavia	20.0
Portugal	20.0
Norway	20.0
Denmark	33.0
Turkey	50.0
Slovenia	80.0

Table 1: Odds for the 2000 European soccer championships.

- (b) *Betting.* In a simple application, we have exactly as many wagers as there are outcomes ($n = m$). Wager i is to bet that the outcome will be i . The returns are usually expressed as *odds*. For example, suppose that a bookmaker accepts bets on the result of the 2000 European soccer championships. If the odds against Belgium winning are 14 to one, and we bet \$100 on Belgium, then we win \$1400 if they win the tournament, and we lose \$100 otherwise. In general, if we have m possible outcomes, and the odds against outcome i are λ_i to one, then the return matrix $R \in \mathbf{R}^{m \times m}$ is given by

$$\begin{aligned} r_{ij} &= \lambda_i && \text{if } j = i \\ r_{ij} &= -1 && \text{otherwise.} \end{aligned}$$

Show that there is no sure-win betting scheme (or arbitrage opportunity) if

$$\sum_{i=1}^m \frac{1}{1 + \lambda_i} = 1.$$

In fact, you can verify that if this equality is not satisfied, then the betting strategy

$$x_i = \frac{1/(1 + \lambda_i)}{1 - \sum_{i=1}^m 1/(1 + \lambda_i)}$$

always results in a profit.

The common situation in real life is

$$\sum_{i=1}^m \frac{1}{1 + \lambda_i} > 1,$$

because the bookmakers take a cut on all bets.

Solution.

- (a) This follows directly from the theorem of alternatives for strict linear inequalities on page 8-26 of the lecture notes.
(b) XXX missing ?? XXX

- (c) Using the theorem proved in part (a), we know that if there is no arbitrage opportunity, then there exists a solution to the conditions (2), *i.e.*, there exists p_1, p_2 , satisfying

$$p_1 \geq 0, \quad p_2 \geq 0, \quad p_1 + p_2 \neq 0.$$

and $R^T p = 0$. We can assume that $p_1 + p_2 = 1$, so the condition $R^T p = 0$ reduces to

$$p_1 \frac{u - 1 - r}{1 + r} + (1 - p_1) \frac{d - 1 - r}{1 + r} = 0$$

and

$$p_1 \left(\frac{\max\{0, Su - K\}}{(1+r)C} - 1 \right) + (1 - p_1) \left(\frac{\max\{0, Sd - K\}}{(1+r)C} - 1 \right) = 0.$$

The first equality yields

$$p_1 = \frac{1 + r - d}{u - d}.$$

The second equality yields

$$C = \frac{1}{1+r} (p_1 \max\{0, Su - K\} + (1 - p_1) \max\{0, Sd - K\}).$$

- 13.8 Log-optimal investment.** We consider an instance of the log-optimal investment problem described in exercise 4.60 of *Convex Optimization*. In this exercise, however, we allow x , the allocation vector, to have negative components. Investing a negative amount $x_i W(t)$ in an asset is called *shorting* the asset. It means you borrow the asset, sell it for $|x_i W(t)|$, and have an obligation to purchase it back later and return it to the lender.

- (a) Let R be the $n \times m$ -matrix with columns r_j :

$$R = \begin{bmatrix} r_1 & r_2 & \cdots & r_m \end{bmatrix}.$$

We assume that the elements r_{ij} of R are all positive, which implies that the log-optimal investment problem is feasible. Show the following property: if there exists a $v \in \mathbf{R}^n$ with

$$\mathbf{1}^T v = 0, \quad R^T v \succeq 0, \quad R^T v \neq 0 \tag{54}$$

then the log-optimal investment problem is unbounded (assuming that the probabilities p_j are all positive).

- (b) Derive a Lagrange dual of the log-optimal investment problem (or an equivalent problem of your choice). Use the Lagrange dual to show that the condition in part a is also necessary for unboundedness. In other words, the log-optimal investment problem is bounded if and only if there does not exist a v satisfying (54).
- (c) Consider the following small example. We have four scenarios and three investment options. The return vectors for the four scenarios are

$$r_1 = \begin{bmatrix} 2 \\ 1.3 \\ 1 \end{bmatrix}, \quad r_2 = \begin{bmatrix} 2 \\ 0.5 \\ 1 \end{bmatrix}, \quad r_3 = \begin{bmatrix} 0.5 \\ 1.3 \\ 1 \end{bmatrix}, \quad r_4 = \begin{bmatrix} 0.5 \\ 0.5 \\ 1 \end{bmatrix}.$$

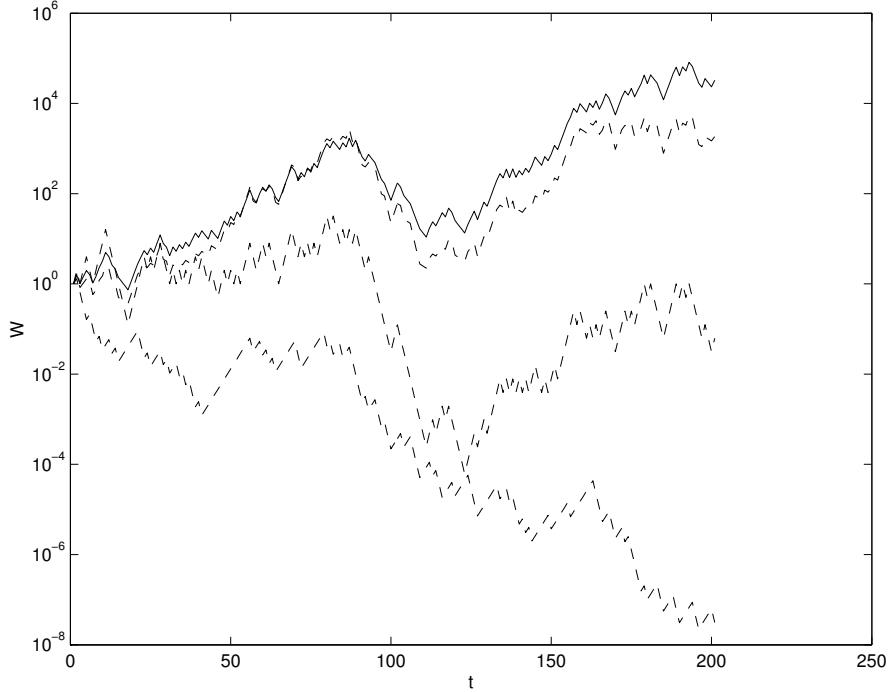
The probabilities of the three scenarios are

$$p_1 = 1/3, \quad p_2 = 1/6, \quad p_3 = 1/3, \quad p_4 = 1/6.$$

The interpretation is as follows. We can invest in two stocks. The first stock doubles in value in each period with a probability 1/2, or decreases by 50% with a probability 1/2. The second stock either increases by 30% with a probability 2/3, or decreases by 50% with a probability 1/3. The fluctuations in the two stocks are independent, so we have four scenarios: both stocks go up (probability 2/6), stock 1 goes up and stock 2 goes down (probability 1/6), stock 1 goes down and stock 2 goes up (probability 1/3), both stocks go down (probability 1/6). The fractions of our capital we invest in stocks 1 and 2 are denoted by x_1 and x_2 , respectively. The rest of our capital, $x_3 = 1 - x_1 - x_2$ is not invested.

What is the expected growth rate of the log-optimal strategy x ? Compare with the strategies $(x_1, x_2, x_3) = (1, 0, 0)$, $(x_1, x_2, x_3) = (0, 1, 0)$ and $(x_1, x_2, x_3) = (1/2, 1/2, 0)$. (Obviously the expected growth rate for $(x_1, x_2, x_3) = (0, 0, 1)$ is zero.)

Remark. The figure below shows a simulation that compares three investment strategies over 200 periods. The solid line shows the log-optimal investment strategy. The dashed lines show the growth for strategies $x = (1, 0, 0)$, $(0, 1, 0)$, and $(0, 0, 1)$.



Solution.

- (a) The objective function grows unboundedly along the line defined as $x + tv$, $t \geq 0$.
- (b) We introduce new variables y , and write the problem as

$$\begin{aligned} & \text{minimize} && -\sum_{j=1}^m p_j \log y_j \\ & \text{subject to} && y = R^T x \\ & && \mathbf{1}^T x = 1. \end{aligned}$$

The Lagrangian is

$$L(x, y, \nu, \lambda) = -\sum_{j=1}^m p_j \log y_j + \nu^T(y - R^T x) + \lambda(\mathbf{1}^T x - 1),$$

which is bounded below as a function of x only if $R\nu = \lambda\mathbf{1}$. As a function of y , L reaches its minimum if $y_j = p_j/\nu_j$. We obtain the dual function

$$g(\nu, \lambda) = -\sum_{i=1}^m p_i \log(p_i/\nu_i) + 1 - \lambda$$

if $\nu \succ 0$ and $R\nu = \lambda\mathbf{1}$. The dual problem is

$$\begin{aligned} & \text{minimize} && \sum_{i=1}^m p_i \log p_i/\nu_i - 1 + \lambda \\ & \text{subject to} && R\nu = \mathbf{1}\lambda. \end{aligned}$$

The variables are $\nu \in \mathbf{R}^m$ and $\lambda \in \mathbf{R}$.

To prove the condition for unboundedness we can use the following theorem of alternatives. There exists a solution $x \succ 0$ satisfying $Ax = b$ if and only if there does not exist a v with $b^T v \leq 0$, $A^T v \succeq 0$, $v \neq 0$. We can prove this as in lecture 8, or the reader on page 130. The dual function of the feasibility problem

$$x \succ 0, \quad Ax = b$$

is

$$\begin{aligned} g(\lambda, \nu) &= \inf_x -\lambda^T x + \nu^T(Ax - b) \\ &= \begin{cases} -b^T \nu & \text{if } A^T \nu = \lambda \\ -\infty & \text{otherwise.} \end{cases} \end{aligned}$$

Therefore the alternative system is

$$A^T \nu \succeq 0, \quad A^T \nu \neq 0, \quad b^T \nu \leq 0.$$

Now we apply this result to problem (4). If (4) is infeasible, then

$$\mathbf{1}^T v \leq 0, \quad R^T v \succeq 0, \quad R^T v \neq 0$$

is infeasible. (This follows from the fact that all elements of R are positive). By the theorem of alternatives , there exists a $y \succ 0$ such that

$$Ry = \mathbf{1},$$

which means $\nu = y$, $\lambda = 1$ is feasible in the dual problem. Therefore the primal problem is bounded above.

```

(c) R = [1      1      -1/2 -1/2;
          0.3 -0.5    0.3 -0.5];
p = [1/3;  1/6;  1/3; 1/6];
x = [0;0];
alpha = 0.2;
beta = 0.5;

for i=1:100
val = -p'*log(1+R'*x);
grad = -R*(p./(1+R'*x));
hess = R*diag(p./(1+R'*x).^2)*R';
dx = -hess\grad;
fprime = grad'*dx;
if (sqrt(-fprime) <= 1e-5), break; end;
t=1;
for k=1:50
newx = x+t*dx;
newval = -p'*log(1+R'*newx);
if ((min(1+R'*newx) > 0) & (-p'*log(1+R'*newx) <= val + t*alpha*fprime))
else t = beta*t; end;
end;
x = x + t*dx;
end;

```

The optimal solution is

$$x_1 = 0.4973, \quad x_2 = 0.1994, \quad x_3 = 0.3034.$$

The expected growth rate is 0.0623.

The expected growth rate for $x = (1, 0, 0)$ is 0. The expected growth rate for $x = (0, 1, 0)$ is -0.0561 . The expected growth rate for $x = (1/2, 1/2, 0)$ is 0.0535.

13.9 Maximizing house profit in a gamble and imputed probabilities. A set of n participants bet on which one of m outcomes, labeled $1, \dots, m$, will occur. Participant i offers to purchase up to $q_i > 0$ gambling contracts, at price $p_i > 0$, that the true outcome will be in the set $S_i \subset \{1, \dots, m\}$. The house then sells her x_i contracts, with $0 \leq x_i \leq q_i$. If the true outcome j is in S_i , then participant i receives \$1 per contract, *i.e.*, x_i . Otherwise, she loses, and receives nothing. The house collects a total of $x_1 p_1 + \dots + x_n p_n$, and pays out an amount that depends on the outcome j ,

$$\sum_{i: j \in S_i} x_i.$$

The difference is the house profit.

- (a) *Optimal house strategy.* How should the house decide on x so that its worst-case profit (over the possible outcomes) is maximized? (The house determines x after examining all the participant offers.)

- (b) *Imputed probabilities.* Suppose x^* maximizes the worst-case house profit. Show that there exists a probability distribution π on the possible outcomes (*i.e.*, $\pi \in \mathbf{R}_+^m$, $\mathbf{1}^T \pi = 1$) for which x^* also maximizes the expected house profit. Explain how to find π .

Hint. Formulate the problem in part (a) as an LP; you can construct π from optimal dual variables for this LP.

Remark. Given π , the ‘fair’ price for offer i is $p_i^{\text{fair}} = \sum_{j \in S_i} \pi_j$. All offers with $p_i > p_i^{\text{fair}}$ will be completely filled (*i.e.*, $x_i = q_i$); all offers with $p_i < p_i^{\text{fair}}$ will be rejected (*i.e.*, $x_i = 0$).

Remark. This exercise shows how the probabilities of outcomes (*e.g.*, elections) can be guessed from the offers of a set of gamblers.

- (c) *Numerical example.* Carry out your method on the simple example below with $n = 5$ participants, $m = 5$ possible outcomes, and participant offers

Participant i	p_i	q_i	S_i
1	0.50	10	{1,2}
2	0.60	5	{4}
3	0.60	5	{1,4,5}
4	0.60	20	{2,5}
5	0.20	10	{3}

Compare the optimal worst-case house profit with the worst-case house profit, if all offers were accepted (*i.e.*, $x_i = q_i$). Find the imputed probabilities.

Solution.

- (a) The worst-case house profit is

$$p^T x - \max_{j=1, \dots, m} \sum_{i: j \in S_i} x_i,$$

which is a piecewise-linear concave function of x . To find the x that maximizes the worst-case profit, we solve the problem,

$$\begin{aligned} & \text{maximize} && p^T x - \max_{j=1, \dots, m} a_j^T x \\ & \text{subject to} && 0 \preceq x \preceq q, \end{aligned}$$

with variable x . a_j^T are the rows of the subset matrix A , with

$$A_{ji} = \begin{cases} 1 & j \in S_i \\ 0 & \text{otherwise.} \end{cases}$$

- (b) The problem from part (a) can be expressed as

$$\begin{aligned} & \text{maximize} && p^T x - t \\ & \text{subject to} && t\mathbf{1} \succeq Ax \\ & && 0 \preceq x \preceq q, \end{aligned} \tag{55}$$

where t is a new scalar variable. The Lagrangian is

$$L(x, t, \lambda_1, \lambda_2, \lambda_3) = t - p^T x + \lambda_1^T (Ax - t\mathbf{1}) - \lambda_2^T x + \lambda_3^T (x - q).$$

This is bounded below if and only if $\mathbf{1}^T \lambda = 1$, and $A^T \lambda_1 - \lambda_2 + \lambda_3 = p$. The dual can be written as

$$\begin{aligned} & \text{maximize} && -q^T \lambda_3 \\ & \text{subject to} && \mathbf{1}^T \lambda_1 = 1 \\ & && A^T \lambda_1 - \lambda_2 + \lambda_3 = p \\ & && \lambda_1 \succeq 0, \quad \lambda_2 \succeq 0, \quad \lambda_3 \succeq 0, \end{aligned} \tag{56}$$

with variables λ_1 , λ_2 , and λ_3 . Notice that λ_1 must satisfy $\mathbf{1}^T \lambda_1 = 1$, and $\lambda_1 \succeq 0$, hence it is a probability distribution.

Suppose x_{wc}^* , t^* , λ_1^* , λ_2^* , and λ_3^* are primal and dual optimal for problem (55), and let us set $\pi = \lambda_1^*$. To maximize the expected house profit we solve the problem,

$$\begin{aligned} & \text{maximize} && p^T x - \pi^T Ax \\ & \text{subject to} && 0 \preceq x \preceq q. \end{aligned} \tag{57}$$

Let b_1^T, \dots, b_n^T be the rows of A^T . We know that a point x_e^* is optimal for problem (57) if and only if $x_{ei}^* = q_i$ when $p_i - b_i^T \pi > 0$, $x_{ei}^* = 0$ when $p_i - b_i^T \pi < 0$, and $0 \leq x_{ei}^* \leq q_i$ when $p_i - b_i^T \pi = 0$.

To see why $x_e^* = x_{\text{wc}}^*$, let us take a look at one of the KKT conditions for problem (55). This can be written as

$$p - A^T \pi = \lambda_3^* - \lambda_2^*,$$

with $\pi = \lambda_1^*$. If $p_i - b_i^T \pi > 0$, then we must have $\lambda_{3i}^* - \lambda_{2i}^* > 0$, which means that $\lambda_{2i}^* = 0$ and $\lambda_{3i}^* > 0$ (by complementary slackness), and so $x_{\text{wci}}^* = q_i$. Similarly, if $p_i - b_i^T \pi < 0$, then $\lambda_{3i}^* - \lambda_{2i}^* < 0$, which means that $\lambda_{2i}^* > 0$ and $\lambda_{3i}^* = 0$, and so $x_{\text{wci}}^* = 0$. Finally, when $p_i - b_i^T \pi = 0$, we must have $\lambda_{2i}^* = 0$ and $\lambda_{3i}^* = 0$, and so $0 \leq x_{\text{wci}}^* \leq q_i$.

In summary, in order to find a probability distribution on the possible outcomes for which the same x^* maximizes both the worst-case as well as the expected house profit, we solve the dual LP (56), and set $\pi = \lambda_1^*$.

(c) The following Matlab code solves the problem.

```
% solution for gambling problem
A = [1 0 1 0 0;
      1 0 0 1 0;
      0 0 0 0 1;
      0 1 1 0 0;
      0 0 1 1 0];

p = [0.5; 0.6; 0.6; 0.6; 0.2];
q = [10; 5; 5; 20; 10];

n = 5; m = 5;

cvx_begin
    variables x(n) t
    dual variable lambda1
    maximize (p'*x-t)
```

```

subject to
    lambda1: A*x <= t
    x >= 0
    x <= q
cvx_end

% optimal worst case house profit
pwc = cvx_optval
% optimal worst case profit if all offer are accepted
pwc_accept = p'*q-max(A*q)
% imputed probabilities
pi = lambda1
% fair prices
pfair = A'*pi
% optimal purchase quantities
xopt = x

```

The following Python code solves the problem.

```

# Solution for gambling problem
import cvxpy as cvx
import numpy as np

A = np.matrix('1, 0, 1, 0, 0; \
                1, 0, 0, 1, 0; \
                0, 0, 0, 0, 1; \
                0, 1, 1, 0, 0; \
                0, 0, 1, 1, 0')
p = np.matrix('0.5; 0.6; 0.6; 0.6; 0.2')
q = np.matrix('10; 5; 5; 20; 10')

n = 5
m = 5

x, t = cvx.Variable(n), cvx.Variable(1)
obj = p.T*x-t
cons = [A*x <= t]
cons += [x >= 0]
cons += [x <= q]

p_star = cvx.Problem(cvx.Maximize(obj), cons).solve()
lambda1 = cons[0].dual_value

# optimal worst case house profit
pwc = p_star
# optimal worst case profit if all offers are accepted
pwc_accept = p.T*q-max(A*q)

```

```

# imputed probabilities
pi = lambda1
# fair prices
pfair = A.T*pi
# optimal purchase quantities
xopt = x.value

```

The following Julia code solves the problem.

```

# solution for gambling problem
A = [1 0 1 0 0;
      1 0 0 1 0;
      0 0 0 0 1;
      0 1 1 0 0;
      0 0 1 1 0];

p = [0.5; 0.6; 0.6; 0.6; 0.2];
q = [10; 5; 5; 20; 10];

n = 5; m = 5;

x = Variable(n);
t = Variable();
constraints = [A*x <= t, x >= 0, x <= q];
problem = maximize(p'*x - t, constraints);
solve!(problem)

# optimal worst case house profit
pwc = problem.optval
# optimal worst case profit if all offer are accepted
pwc_accept = p'*q - maximum(A*q)
# imputed probabilities
pi = constraints[1].dual
# fair prices
pfair = A'*pi
# optimal purchase quantities
xopt = x.value

```

Our results are summarized in the following table. We find that the optimal worst case house profit is 3.5, and the worst case house profit, if all offers are accepted, is -5. The imputed probabilities are

$$\pi = (0.1145, 0.3855, 0.0945, 0.1910, 0.2145)$$

in Matlab,

$$\pi = (0.0970, 0.4040, 0.1316, 0.1715, 0.1970)$$

in Python, and

$$\pi = (0.0567, 0.4433, 0.0784, 0.2649, 0.1567)$$

in Julia. Note that although these are the imputed probabilities that we found, the probabilities are not unique. The associated fair prices and optimal contract numbers are shown below.

Participant i	p_i	p_i^{fair}	q_i	x_i	S_i
1	0.50	0.5000	10	5	{1,2}
2	0.60	0.1910	5	5	{4}
3	0.60	0.5200	5	5	{1,4,5}
4	0.60	0.6000	20	5	{2,5}
5	0.20	0.0945	10	10	{3}

- 13.10 Optimal investment to fund an expense stream.** An organization (such as a municipality) knows its operating expenses over the next T periods, denoted E_1, \dots, E_T . (Normally these are positive; but we can have negative E_t , which corresponds to income.) These expenses will be funded by a combination of investment income, from a mixture of bonds purchased at $t = 0$, and a cash account.

The bonds generate investment income, denoted I_1, \dots, I_T . The cash balance is denoted B_0, \dots, B_T , where $B_0 \geq 0$ is the amount of the initial deposit into the cash account. We can have $B_t < 0$ for $t = 1, \dots, T$, which represents borrowing.

After paying for the expenses using investment income and cash, in period t , we are left with $B_t - E_t + I_t$ in cash. If this amount is positive, it earns interest at the rate $r_+ > 0$; if it is negative, we must pay interest at rate r_- , where $r_- \geq r_+$. Thus the expenses, investment income, and cash balances are linked as follows:

$$B_{t+1} = \begin{cases} (1 + r_+)(B_t - E_t + I_t) & B_t - E_t + I_t \geq 0 \\ (1 + r_-)(B_t - E_t + I_t) & B_t - E_t + I_t < 0, \end{cases}$$

for $t = 1, \dots, T-1$. We take $B_1 = (1 + r_+)B_0$, and we require that $B_T - E_T + I_T = 0$, which means the final cash balance, plus income, exactly covers the final expense.

The initial investment will be a mixture of bonds, labeled $1, \dots, n$. Bond i has a price $P_i > 0$, a coupon payment $C_i > 0$, and a maturity $M_i \in \{1, \dots, T\}$. Bond i generates an income stream given by

$$a_t^{(i)} = \begin{cases} C_i & t < M_i \\ C_i + 1 & t = M_i \\ 0 & t > M_i, \end{cases}$$

for $t = 1, \dots, T$. If x_i is the number of units of bond i purchased (at $t = 0$), the total investment cash flow is

$$I_t = x_1 a_t^{(1)} + \dots + x_n a_t^{(n)}, \quad t = 1, \dots, T.$$

We will require $x_i \geq 0$. (The x_i can be fractional; they do not need to be integers.)

The total initial investment required to purchase the bonds, and fund the initial cash balance at $t = 0$, is $x_1 P_1 + \dots + x_n P_n + B_0$.

- (a) Explain how to choose x and B_0 to minimize the total initial investment required to fund the expense stream.

- (b) Solve the problem instance given in `opt_funding_data.m`. Give optimal values of x and B_0 . Give the optimal total initial investment, and compare it to the initial investment required if no bonds were purchased (which would mean that all the expenses were funded from the cash account). Plot the cash balance (versus period) with optimal bond investment, and with no bond investment.

Solution.

- (a) We will give several solutions (which are close, but not the same).

Method 1. The cash balance propagation equations,

$$B_{t+1} = \begin{cases} (1+r_+)(B_t - E_t + I_t) & B_t - E_t + I_t \geq 0 \\ (1+r_-)(B_t - E_t + I_t) & B_t - E_t + I_t < 0, \end{cases}$$

which can be expressed as

$$B_{t+1} = \min \{(1+r_+)(B_t - E_t + I_t), (1+r_-)(B_t - E_t + I_t)\},$$

are clearly not convex when B_t are considered as variables. We will describe two ways to handle them.

In the first method, we relax the propagation equations to the convex inequalities

$$B_{t+1} \leq \min \{(1+r_+)(B_t - E_t + I_t), (1+r_-)(B_t - E_t + I_t)\}.$$

With this relaxation, the problem can be expressed as:

$$\begin{aligned} \text{minimize} \quad & B_0 + \sum_{i=1}^n P_i x_i \\ \text{subject to} \quad & \sum_{i=1}^n a_t^{(i)} x_i = I_t, \quad t = 1, \dots, T \\ & x \succeq 0, \quad B_0 \geq 0 \\ & B_1 = (1+r_+)B_0 \\ & B_{t+1} \leq \min \{(1+r_+)(B_t - E_t + I_t), (1+r_-)(B_t - E_t + I_t)\}, \quad t = 1, \dots, T-1 \\ & B_T - E_T + I_T = 0, \end{aligned}$$

with variables $B_0, \dots, B_T, I_1, \dots, I_T, x_1, \dots, x_n$. (We could treat I_t as an affine expression, instead of a variable; it makes no difference.)

We will now argue that any solution of this problem must satisfy

$$B_{t+1} = \min \{(1+r_+)(B_t - E_t + I_t), (1+r_-)(B_t - E_t + I_t)\}, \quad t = 1, \dots, T-1,$$

which means that by solving the LP above, we are also solving the original problem. Suppose this is not the case; suppose, for example, that τ is the first period for which

$$B_{\tau+1} < \min \{(1+r_+)(B_\tau - E_\tau + I_\tau), (1+r_-)(B_\tau - E_\tau + I_\tau)\}.$$

Of course, this is kind of silly: it means we have elected to throw away some cash between periods τ and $\tau+1$. To give a formal argument that this cannot happen, we first note that B_{t+1} is monotonically nondecreasing in B_t for all t , since each term within the minimization is monotonically increasing in B_t . Thus, we can decrease B_τ until $B_{\tau+1} = \min \{(1+r_+)(B_\tau - E_\tau + I_\tau), (1+r_-)(B_\tau - E_\tau + I_\tau)\}$, and still maintain feasibility of the cash stream. By induction, we can decrease $B_{\tau-1}, B_{\tau-2}, \dots, B_0$ and still have a feasible solution. This new stream is feasible, and reduces B_0 , which shows that the original point was not optimal.

Method 2. The second method carries out a different relaxation. In this method we do not consider B_1, \dots, B_T as variables; instead we think of them as functions of x and B_0 , defined by a recursion. We will show that $B_T(B_0, x)$ is a concave function of B_0 and x . $B_1(B_0, x) = (1 + r_+)B_0$ is an concave (affine) function. If $B_t(B_0, x)$ is a concave function, then

$$B_{t+1}(B_0, x) = \min \{(1 + r_+)(B_t(B_0, x) - E_t + I_t(x)), \\ (1 + r_-)(B_t(B_0, x) - E_t + I_t(x))\}, \quad t = 1, \dots, T - 1$$

is also a concave function because it is the pointwise minimum of two concave functions. Therefore, we can conclude that $B_T(B_0, x)$ is a concave function.

The final cash balance constraint $B_T(B_0, x) - E_T + I_T(x) = 0$ is clearly not convex because $B_T(B_0, x)$ is a concave function. To handle this equation, we relax it to the convex inequality

$$B_T(B_0, x) - E_T + I_T(x) \geq 0.$$

With this relaxation, the problem can be expressed as:

$$\begin{aligned} & \text{minimize} && B_0 + \sum_{i=1}^n P_i x_i \\ & \text{subject to} && x \succeq 0, \quad B_0 \geq 0 \\ & && B_T(B_0, x) - E_T + I_T(x) \geq 0, \end{aligned}$$

with variables B_0 and x_1, \dots, x_n . Here $B_T(B_0, x)$ and $I_T(x)$ are defined by the following recursive definitions:

$$\begin{aligned} B_1(B_0, x) &= (1 + r_+)B_0 \\ B_{t+1}(B_0, x) &= \min \{(1 + r_+)(B_t(B_0, x) - E_t + I_t(x)), \\ &\quad (1 + r_-)(B_t(B_0, x) - E_t + I_t(x))\}, \quad t = 1, \dots, T - 1 \\ I_t(x) &= \sum_{i=1}^n a_t^{(i)} x_i, \quad t = 1, \dots, T. \end{aligned}$$

We can argue that any solution of this problem must satisfy

$$B_T(B_0, x) - E_T + I_T(x) = 0,$$

which is very similar to the argument given above (for the inequality relaxation).

Method 3. Here is yet another solution to this problem. The cash balance propagation equations,

$$B_{t+1} = \begin{cases} (1 + r_+)(B_t - E_t + I_t) & B_t - E_t + I_t \geq 0 \\ (1 + r_-)(B_t - E_t + I_t) & B_t - E_t + I_t < 0, \end{cases}$$

are equivalent to LP constraints

$$\begin{aligned} B_{t+1} &= (1 + r_+)S_t^+ - (1 + r_-)S_t^-, \\ B_t - E_t + I_t &= S_t^+ - S_t^-, \\ S_t^+ &\geq 0, \\ S_t^- &\geq 0. \end{aligned}$$

The equivalence can be shown easily: Suppose $B_t - E_t + I_t$ is positive, then $S_t^+ = B_t - E_t + I_t$ and $S_t^- = 0$, so, $B_{t+1} = (1 + r_+)(B_t - E_t + I_t)$. If $B_t - E_t + I_t$ is negative, then $S_t^+ = 0$ and $S_t^- = B_t - E_t + I_t$, so, $B_{t+1} = (1 + r_-)(B_t - E_t + I_t)$.

Therefore, the problem can be expressed as a LP:

$$\begin{aligned} \text{minimize} \quad & B_0 + \sum_{i=1}^n P_i x_i \\ \text{subject to} \quad & \sum_{i=1}^n a_t^{(i)} x_i = I_t, \quad t = 1, \dots, T \\ & x \succeq 0, \quad B_0 \geq 0 \\ & B_1 = (1 + r_+) B_0 \\ & B_T - E_T + I_T = 0, \\ & B_{t+1} = (1 + r_+) S_t^+ - (1 + r_-) S_t^-, \quad t = 1, \dots, T-1 \\ & B_t - E_t + I_t = S_t^+ - S_t^-, \quad t = 1, \dots, T-1 \\ & S_t^+ \geq 0, \quad t = 1, \dots, T-1 \\ & S_t^- \geq 0 \quad t = 1, \dots, T-1, \end{aligned}$$

with variables B_0, \dots, B_T , I_1, \dots, I_T , x_1, \dots, x_n , S_1^+, \dots, S_{T-1}^+ and S_1^-, \dots, S_{T-1}^- . With this approach, we have to argue that for any t , we will either have S_t^+ or S_t^- zero. (This is easily seen.)

```
(b) % Optimal investment to fund an expense stream
clear all; close all;
opt_funding_data;

cvx_begin
variables x(n) B0 B(T) I(T)
minimize( P'*x+B0 )
subject to
    I == A*x
    x >= 0 % use x == 0 for no bond purchase
    B0 >= 0
    B(1) <= (1+rp)*B0
    for t = 1:T-1
        B(t+1) <= min( (1+rp)*(B(t)-E(t)+I(t)), (1+rn)*(B(t)-E(t)+I(t)) )
    end
    B(T)-E(T)+I(T) == 0
cvx_end

% optimal total initial investment
cvx_optval

% associated optimal cash balance plot
bar(0:T,[B0;B]);
ylabel('cash balance'); xlabel('t');
```

The code (CVX part) with recursive definition of $B_t(B_0, x)$ is given below.

```
cvx_begin
variables x(n) B0
```

```

minimize( P'*x+B0 )
subject to
    x >= 0 % use x == 0 for no bond purchase
    B0 >= 0
    B(1) = (1+rp)*B0
    for t = 1:T-1
        B(t+1)=min((1+rp)*(B(t)-E(t)+A(t,:)*x),(1+rn)*(B(t)-E(t)+A(t,:)*x));
    end
    B(T)-E(T)+A(T,:)*x >= 0
cvx_end

```

The code (CVX part) with LP formulation is given below.

```

cvx_begin
variables x(n) B0 B(T) I(T) Sp(T-1) Sn(T-1)
minimize( P'*x+B0 )
subject to
    I == A*x
    x >= 0 % use x == 0 for no bond purchase
    B0 >= 0
    B(1) == (1+rp)*B0
    Sp >= 0
    Sn >= 0
    for t = 1:T-1
        B(t)-E(t)+I(t) == Sp(t) - Sn(t);
        B(t+1) == (1+rp)*Sp(t)-(1+rn)*Sn(t)
    end
    B(T)-E(T)+I(T) == 0
cvx_end

```

The optimal bond investment is

$$x = (0.0000, 18.9326, 0.0000, 0.0000, 13.8493, 8.9228),$$

and the optimal initial deposit to the cash account is $B_0 = 0$. The optimal total initial investment is 40.7495.

With no bond investment (*i.e.*, $x = 0$), an initial deposit of $B_0 = 41.7902$ is cash is required, around 2.5% more than when we invest in bonds.

13.11 Planning production with uncertain demand. You must order (nonnegative) amounts r_1, \dots, r_m of raw materials, which are needed to manufacture (nonnegative) quantities q_1, \dots, q_n of n different products. To manufacture one unit of product j requires at least A_{ij} units of raw material i , so we must have $r \succeq Aq$. (We will assume that A_{ij} are nonnegative.) The per-unit cost of the raw materials is given by $c \in \mathbf{R}_+^m$, so the total raw material cost is $c^T r$.

The (nonnegative) demand for product j is denoted d_j ; the number of units of product j sold is $s_j = \min\{q_j, d_j\}$. (When $q_j > d_j$, $q_j - d_j$ is the amount of product j produced, but not sold; when $d_j > q_j$, $d_j - q_j$ is the amount of unmet demand.) The revenue from selling the products is $p^T s$,

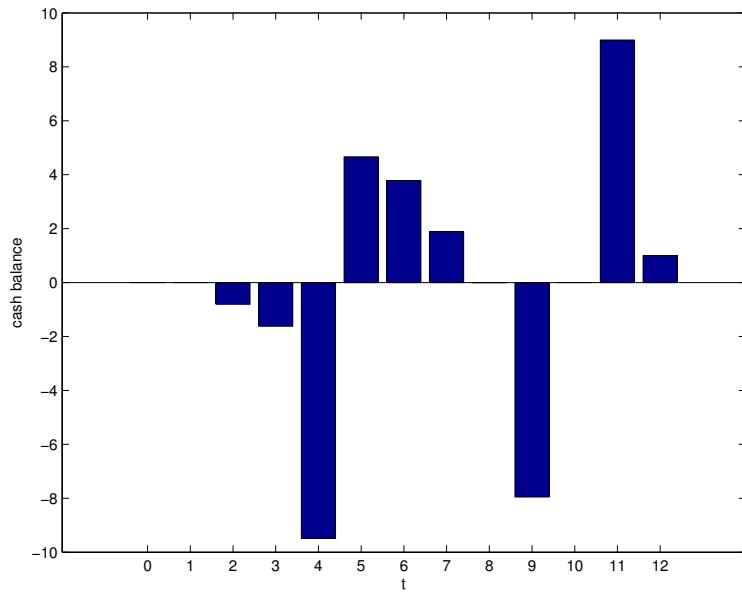


Figure 13: Cash balance with optimal bond purchase.

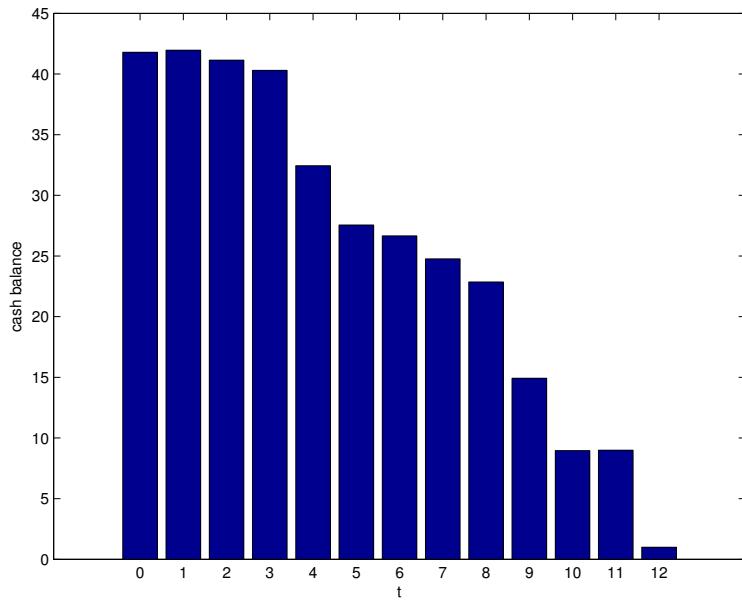


Figure 14: Cash balance with no bond purchase.

where $p \in \mathbf{R}_+^n$ is the vector of product prices. The profit is $p^T s - c^T r$. (Both d and q are real vectors; their entries need not be integers.)

You are given A , c , and p . The product demand, however, is not known. Instead, a set of K possible demand vectors, $d^{(1)}, \dots, d^{(K)}$, with associated probabilities π_1, \dots, π_K , is given. (These satisfy $\mathbf{1}^T \pi = 1$, $\pi \succeq 0$.)

You will explore two different optimization problems that arise in choosing r and q (the variables).

I. Choose r and q ahead of time. You must choose r and q , knowing only the data listed above. (In other words, you must order the raw materials, and commit to producing the chosen quantities of products, before you know the product demand.) The objective is to maximize the expected profit.

II. Choose r ahead of time, and q after d is known. You must choose r , knowing only the data listed above. Some time after you have chosen r , the demand will become known to you. This means that you will find out which of the K demand vectors is the true demand. Once you know this, you must choose the quantities to be manufactured. (In other words, you must order the raw materials before the product demand is known; but you can choose the mix of products to manufacture after you have learned the true product demand.) The objective is to maximize the expected profit.

- (a) Explain how to formulate each of these problems as a convex optimization problem. Clearly state what the variables are in the problem, what the constraints are, and describe the roles of any auxiliary variables or constraints you introduce.
- (b) Carry out the methods from part (a) on the problem instance with numerical data given in `planning_data.m`. This file will define A , D , K , c , m , n , p and π . The K columns of D are the possible demand vectors. For both of the problems described above, give the optimal value of r , and the expected profit.

Solution. We first consider the case when r and q must be decided before the demand is known. The variables are $r \in \mathbf{R}^m$ and $q \in \mathbf{R}^n$. The cost of the resources is deterministic; it's just $c^T r$. The revenue from product sales is a random variable, with values

$$p^T \min\{q, d^{(k)}\}, \quad k = 1, \dots, K,$$

(where $\min\{q, d^{(k)}\}$ is meant elementwise), with associated probabilities π_1, \dots, π_K . The expected profit is therefore

$$-c^T r + \sum_{k=1}^K \pi_k p^T \min\{q, d^{(k)}\}.$$

Our problem is thus

$$\begin{aligned} &\text{maximize} && -c^T r + \sum_{k=1}^K \pi_k p^T \min\{q, d^{(k)}\} \\ &\text{subject to} && q \succeq 0, \quad r \succeq 0, \quad r \succeq Aq, \end{aligned}$$

with variables r and q . The objective is concave (and piecewise-linear), and the constraints are all linear inequalities, so this is a convex optimization problem.

Now we turn to the case when we must commit to the resource order, but can choose the product mix after we know the demand. In this case we need to have a separate product quantity vector for each possible demand vector. Thus we have variables $r \in \mathbf{R}_+^m$ and

$$q^{(1)}, \dots, q^{(K)} \in \mathbf{R}_+^n.$$

Here $q^{(k)}$ is the product mix we produce if $d^{(k)}$ turns out to be the actual product demand. Our problem can be formulated as

$$\begin{aligned} & \text{maximize} && -c^T r + \sum_{k=1}^K \pi_k p^T \min\{q^{(k)}, d^{(k)}\} \\ & \text{subject to} && r \succeq 0 \\ & && q^{(k)} \succeq 0, \quad r \succeq Aq^{(k)}, \quad k = 1, \dots, K. \end{aligned}$$

Note that the variables r and $q^{(1)}, \dots, q^{(K)}$ must be decided at the same time, taking all of them into account; we cannot optimize them separately. Nor can we decide on r first, without taking the different $q^{(k)}$'s into account. (However, provided we have solved the problem above to get the right r , we can then optimize over q , once we know which demand vector is the true one. But there is no need for this, since we have already worked out the optimal q for each possible demand vector.)

We can simplify this a bit. In this case, we will not produce any excess product, because this wastes resources, and we know the demand before we decide how much to manufacture. So we will have $q \preceq d$, and we can write the problem as

$$\begin{aligned} & \text{maximize} && -c^T r + \sum_{k=1}^K \pi_k p^T q^{(k)} \\ & \text{subject to} && r \succeq 0 \\ & && d^{(k)} \succeq q^{(k)} \succeq 0, \quad r \succeq Aq^{(k)}, \quad k = 1, \dots, K. \end{aligned}$$

With the values from `planning_data.m`, the optimal values of r for the two problems, respectively, are

$$\begin{aligned} r_I &= (45.5, 65.8, 44.6, 76.7, 72.0, 77.8, 59.3, 61.9, 83.2, 73.1) \\ r_{II} &= (65.7, 59.6, 55.2, 70.8, 69.7, 68.4, 55.3, 57.4, 63.8, 66.2) \end{aligned}$$

These yield expected profits, respectively, of 114.3 and 133.0.

The CVX code that solves this problem is listed below.

```
disp('Problem I.')
cvx_begin
    variable r(m); % amount of each raw material to buy.
    variable q(n); % amount of each product to manufacture.
    r >= 0;
    q >= 0;
    r >= A*q;
    S = min(q*ones(1,K),D);
    maximize(p'*S*pi - c'*r)
cvx_end
rI = r; disp(r);
```

```

disp(['profit: ' num2str(cvx_optval)]);
profitI = p'*S - c'*r; % save for graphing.

disp('Problem II.')
cvx_begin
    variable r(m); % amount of each raw material to buy.
    variable q(n, K); % product to manufacture, for each demand.
    q >= 0;
    r >= 0;
    r*ones(1,K) >= A*q;
    S = min(q, D);
    maximize(p'*S*pi - c'*r)
cvx_end
rII = r; disp(rII);
disp(['profit: ' num2str(cvx_optval)]);
profitII = p'*S - c'*r; % save for graphing.

```

We mention two other optimization problems, that we didn't ask you to explore. First, suppose we committed to buy r_I resources under the first plan, but later learned the demand (before production started). The question then is: how suboptimal is the expected profit when using the r_I from the first problem instead of r_{II} ? For the given numerical instance, our expected profit drops from 133.0 to 128.7.

Finally, it's also interesting to look at performance when we know the demand before buying the resources. This is the full prescience, or full information case—we know everything beforehand. In this case we can optimize the choice of r and q for each possible demand vector separately. This obviously gives an upper bound on the profit available when we have less information. In this case, the expected profit with full prescience is 161.4.

The code that solves these two additional problems is listed below.

```

disp('Problem II but with rI.')
r = rI;
cvx_begin
    variable q(n, K); % product to manufacture, for each demand.
    q >= 0;
    r*ones(1,K) >= A*q;
    S = min(q, D);
    maximize(p'*S*pi - c'*r)
cvx_end
disp(['profit: ' num2str(p'*S*pi - c'*r)]);
profitIII = p'*S - c'*r; % save for graphing.

disp('Full prescience.')
cvx_begin
    variable r(m, K); % amount of each raw material to buy, for each demand..
    variable q(n, K); % product to manufacture, for each demand.

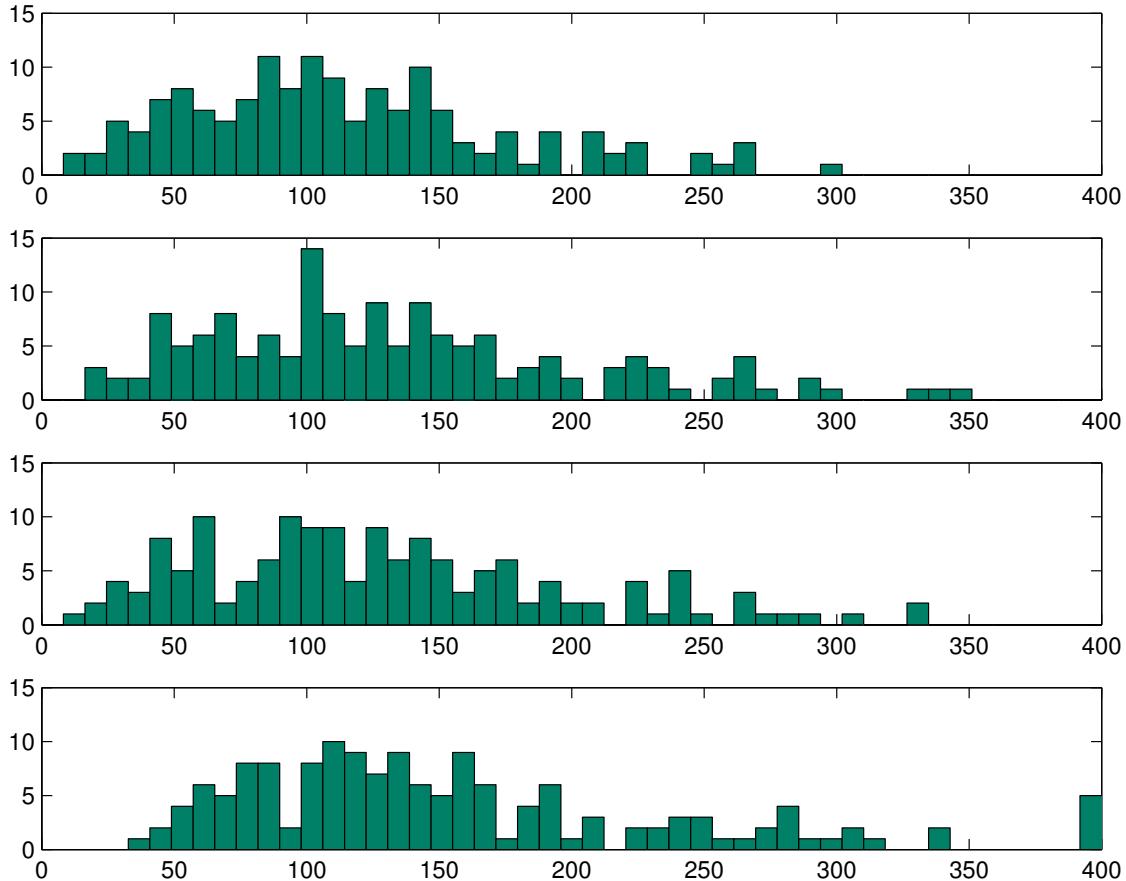
```

```

q >= 0;
r >= 0;
r >= A*q;
S = min(q, D);
maximize((p'*S - c'*r)*pi)
cvx_end
disp(['profit: ' num2str((p'*S*pi - c'*r)*pi)]);
profitIV = p'*S - c'*r; % save for graphing.

```

Histograms showing profits with the different possible outcomes are shown below.



13.12 Gini coefficient of inequality. Let x_1, \dots, x_n be a set of nonnegative numbers with positive sum, which typically represent the wealth or income of n individuals in some group. The *Lorenz curve* is a plot of the fraction f_i of total wealth held by the i poorest individuals,

$$f_i = (1/\mathbf{1}^T x) \sum_{j=1}^i x_{(j)}, \quad i = 0, \dots, n,$$

versus i/n , where $x_{(j)}$ denotes the j th smallest of the numbers $\{x_1, \dots, x_n\}$, and we take $f_0 = 0$. The Lorenz curve starts at $(0, 0)$ and ends at $(1, 1)$. Interpreted as a continuous curve (as, say,

$n \rightarrow \infty$) the Lorentz curve is convex and increasing, and lies on or below the straight line joining the endpoints. The curve coincides with this straight line, *i.e.*, $f_i = (i/n)$, if and only if the wealth is distributed equally, *i.e.*, the x_i are all equal.

The *Gini coefficient* is defined as twice the area between the straight line corresponding to uniform wealth distribution and the Lorentz curve:

$$G(x) = (2/n) \sum_{i=1}^n ((i/n) - f_i).$$

The Gini coefficient is used as a measure of wealth or income inequality: It ranges between 0 (for equal distribution of wealth) and $1 - 1/n$ (when one individual holds all wealth).

- (a) Show that G is a quasiconvex function on $x \in \mathbf{R}_+^n \setminus \{0\}$.
- (b) *Gini coefficient and marriage.* Suppose that individuals i and j get married ($i \neq j$) and therefore pool wealth. This means that x_i and x_j are both replaced with $(x_i + x_j)/2$. What can you say about the change in Gini coefficient caused by this marriage?

Solution.

- (a) Recall that the sum of the i smallest values of x , $S_i = \sum_{j=1}^i x_{(j)}$, is concave, so $Z(x) = \sum_{i=1}^n S_i$ is concave. We can express G as $G = (n+1)/n - (2/n)Z(x)/\mathbf{1}^T x$. Thus, the sublevel set is

$$\{x \mid G(x) \leq t\} = \{x \mid ((n+1)/n - t)\mathbf{1}^T x - (2/n)Z(x) \leq 0\}.$$

The righthand side is clearly convex, since the function $((n+1)/n - t)\mathbf{1}^T x - (2/n)Z(x)$ is convex.

- (b) Marriage cannot increase the Gini coefficient. Here's a snappy proof, using part (a). Let z be the same as x , but with i th and j th entries swapped. Then $(x+z)/2$ is precisely the income distribution after i and j are married. Clearly $G(z) = G(x)$. By quasiconvexity,

$$G((x+z)/2) \leq \max\{G(x), G(z)\} = G(x).$$

13.13 Internal rate of return for cash streams with a single initial investment. We use the notation of example 3.34 in the textbook. Let $x \in \mathbf{R}^{n+1}$ be a cash flow over n periods, with x indexed from 0 to n , where the index denotes period number. We assume that $x_0 < 0$, $x_j \geq 0$ for $j = 1, \dots, n$, and $x_0 + \dots + x_n > 0$. This means that there is an initial positive investment; thereafter, only payments are made, with the total of the payments exceeding the initial investment. (In the more general setting of example 3.34, we allow additional investments to be made after the initial investment.)

- (a) Show that $\text{IRR}(x)$ is quasilinear in this case.
- (b) *Blending initial investment only streams.* Use the result in part (a) to show the following. Let $x^{(i)} \in \mathbf{R}^{n+1}$, $i = 1, \dots, k$, be a set of k cash flows over n periods, each of which satisfies the conditions above. Let $w \in \mathbf{R}_+^k$, with $\mathbf{1}^T w = 1$, and consider the blended cash flow given by $x = w_1 x^{(1)} + \dots + w_k x^{(k)}$. (We can think of this as investing a fraction w_i in cash flow i .) Show that $\text{IRR}(x) \leq \max_i \text{IRR}(x^{(i)})$. Thus, blending a set of cash flows (with initial investment only) will not improve the IRR over the best individual IRR of the cash flows.

Solution.

- (a) Consider any cash flow with $x_0 < 0$, $x_j \geq 0$ for $j = 1, \dots, n$, and $x_0 + \dots + x_n > 0$. These assumptions imply that the present value function $\text{PV}(x, r)$ is monotone decreasing in r , so it crosses 0 at exactly one point. (This is not necessarily the case when some x_j are negative for $j > 0$, which corresponds to making additional investments in addition to the original investment; cf. example 3.34.) Therefore we can say that

$$\text{IRR} \leq R \iff \text{PV}(x, R) \leq 0.$$

The righthand side is a linear inequality in x , and so defines a convex set. It follows that IRR is quasiconvex; but we know from example 3.34 that it is quasiconcave, so it is quasilinear (on the set of x satisfying the assumptions above).

- (b) Now we can solve part (b). For a quasiconvex function, its value at any convex combination of some points cannot exceed the maximum value at the points. This gives us our desired conclusion.

13.14 Efficient solution of basic portfolio optimization problem. This problem concerns the simplest possible portfolio optimization problem:

$$\begin{aligned} &\text{maximize} && \mu^T w - (\lambda/2)w^T \Sigma w \\ &\text{subject to} && \mathbf{1}^T w = 1, \end{aligned}$$

with variable $w \in \mathbf{R}^n$ (the normalized portfolio, with negative entries meaning short positions), and data μ (mean return), $\Sigma \in \mathbf{S}_{++}^n$ (return covariance), and $\lambda > 0$ (the risk aversion parameter). The return covariance has the factor form $\Sigma = FQF^T + D$, where $F \in \mathbf{R}^{n \times k}$ (with rank K) is the *factor loading matrix*, $Q \in \mathbf{S}_{++}^k$ is the factor covariance matrix, and D is a diagonal matrix with positive entries, called the *idiosyncratic risk* (since it describes the risk of each asset that is independent of the factors). This form for Σ is referred to as a ‘ k -factor risk model’. Some typical dimensions are $n = 2500$ (assets) and $k = 30$ (factors).

- (a) What is the flop count for computing the optimal portfolio, if the low-rank plus diagonal structure of Σ is *not* exploited? You can assume that $\lambda = 1$ (which can be arranged by absorbing it into Σ).
(b) Explain how to compute the optimal portfolio more efficiently, and give the flop count for your method. You can assume that $k \ll n$. You do not have to give the best method; any method that has linear complexity in n is fine. You can assume that $\lambda = 1$.

Hints. You may want to introduce a new variable $y = F^T w$ (which is called the vector of factor exposures). You may want to work with the matrix

$$G = \begin{bmatrix} \mathbf{1} & F \\ 0 & -I \end{bmatrix} \in \mathbf{R}^{(n+k) \times (1+k)},$$

treating it as dense, ignoring the (little) exploitable structure in it.

- (c) Carry out your method from part (b) on some randomly generated data with dimensions $n = 2500$, $k = 30$. For comparison (and as a check on your method), compute the optimal portfolio using the method of part (a) as well. Give the (approximate) CPU time for each

method, using `tic` and `toc`. *Hints.* After you generate D and Q randomly, you might want to add a positive multiple of the identity to each, to avoid any issues related to poor conditioning. Also, to be able to invert a block diagonal matrix efficiently, you'll need to recast it as sparse.

- (d) *Risk return trade-off curve.* Now suppose we want to compute the optimal portfolio for M values of the risk aversion parameter λ . Explain how to do this efficiently, and give the complexity in terms of M , n , and k . Compare to the complexity of using the method of part (b) M times. *Hint.* Show that the optimal portfolio is an affine function of $1/\lambda$.

Solution.

- (a) We compute the optimal portfolio by solving the (linear) KKT system

$$\begin{bmatrix} \Sigma & \mathbf{1} \\ \mathbf{1}^T & 0 \end{bmatrix} \begin{bmatrix} w \\ \nu \end{bmatrix} = \begin{bmatrix} \mu \\ 1 \end{bmatrix},$$

where $\nu \in \mathbf{R}$ is the Lagrange multiplier associated with the budget constraint $\mathbf{1}^T w = 1$. The flop count is $(1/3)(n+1)^3$, which after dropping non-dominant terms is the same as $(1/3)n^3$.

- (b) There are several ways to describe an efficient method, and they are all related. Here is one. We introduce a new variable $y = F^T w$ (which represents the factor exposures for the portfolio w), and solve the problem

$$\begin{aligned} & \text{maximize} && \mu^T w - (1/2)w^T D w - (1/2)y^T Q y \\ & \text{subject to} && \mathbf{1}^T w = 1, \quad F^T w = y, \end{aligned}$$

with variables w, y . The KKT system is

$$\begin{bmatrix} D & 0 & \mathbf{1} & F \\ 0 & Q & 0 & -I \\ \mathbf{1}^T & 0 & 0 & 0 \\ F^T & -I & 0 & 0 \end{bmatrix} \begin{bmatrix} w \\ y \\ \nu \\ \kappa \end{bmatrix} = \begin{bmatrix} \mu \\ 0 \\ 1 \\ 0 \end{bmatrix}.$$

We will use standard block elimination, as described in §C.4 of the textbook. To match the notation of that section, we define

$$A_{11} = \begin{bmatrix} D & 0 \\ 0 & Q \end{bmatrix}, \quad A_{12} = \begin{bmatrix} \mathbf{1} & F \\ 0 & -I \end{bmatrix}, \quad A_{21} = A_{12}^T, \quad A_{22} = 0,$$

(with $A_{22} \in \mathbf{R}^{(k+1) \times (k+1)}$),

$$b_1 = \begin{bmatrix} \mu \\ 0 \end{bmatrix}, \quad b_2 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

(where the 0s in these expressions have dimension k), and

$$x_1 = \begin{bmatrix} w \\ y \end{bmatrix}, \quad x_2 = \begin{bmatrix} \nu \\ \kappa \end{bmatrix}.$$

Now we solve the linear equations above by elimination. The Schur complement is

$$\begin{aligned} S &= A_{22} - A_{21}A_{11}^{-1}A_{12} \\ &= - \begin{bmatrix} \mathbf{1}^T & 0 \\ F^T & -I \end{bmatrix} \begin{bmatrix} D^{-1} & 0 \\ 0 & Q^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{1} & F \\ 0 & -I \end{bmatrix} \\ &= - \begin{bmatrix} \mathbf{1}^T D^{-1} \mathbf{1} & \mathbf{1}^T D^{-1} F \\ F^T D^{-1} \mathbf{1} & F^T D^{-1} F + Q^{-1} \end{bmatrix}. \end{aligned}$$

The dominant part of forming S is computing $F^T D^{-1} F$, which costs nk^2 flops. (Computing Q^{-1} is order k^3 , which is non-dominant.) We also compute the reduced righthand side,

$$\begin{aligned} \tilde{b} &= b_2 - A_{21}A_{11}^{-1}b_1 \\ &= \begin{bmatrix} 1 - \mathbf{1}^T D^{-1} \mu \\ -F^T D^{-1} \mu \end{bmatrix}, \end{aligned}$$

with negligible cost (even ignoring the fact that $F^T D^{-1}$ was already computed in forming S). Now we solve the reduced system $Sx_2 = \tilde{b}$, which costs $(1/3)(k+1)^3$, and so is non-dominant. We break up x_2 into its first component, which is ν , and its last k components, κ . Finally, we find x_1 from

$$\begin{aligned} x_1 &= A_{11}^{-1}(b_1 - A_{12}x_2) \\ &= \begin{bmatrix} D^{-1}(\mu - \nu \mathbf{1} - F\kappa) \\ Q^{-1}\kappa \end{bmatrix}, \end{aligned}$$

which has negligible cost. In fact, we only need the top part of x_1 :

$$w = D^{-1}(\mu - \nu \mathbf{1} - F\kappa).$$

The overall flop count is nk^2 flops, quite a bit more efficient than the other method, which requires $(1/3)n^3$ flops.

(c) The Matlab code below carries out both methods.

```
% efficient solution of basic portfolio optimization problem
% generate some random data
n = 2500; % number of assets
k = 30; % number of factors
randn('state',0);
rand('state',0);
F = randn(n,k);
d = 0.1+rand(n,1);
Q = randn(k); Q=Q*Q'+eye(k);
Sigma = diag(d) + F*Q*F';
mu = rand(n,1);

% the slow way, solve full KKT
tic;
```

```

wnu = [Sigma ones(n,1); ones(n,1)' 0] \ [mu ; 1];
toc;
wslow = wnu(1:n);

% fast method (specific code)
tic;
% form Schur complement (some inefficiencies here...)
S= -[sum(1./d) (1./d)'*F; F'*(1./d) F'*diag(1./d)*F+inv(Q)];
btilde = [1- ones(1,n)*(mu./d); -F'*(mu./d)];
x2 = S\btilde;
nu = x2(1); kappa = x2(2:k+1);
wfast = (mu-nu*ones(n,1)-F*kappa)./d;
toc;
rel_err = norm(wfast-wslow)/norm(wslow)

% fast method (general code)
A11 = sparse([diag(d) zeros(n,k); zeros(k,n) Q]);
A12 = sparse([ones(n,1) F; zeros(k,1) -eye(k)]);
A21 = A12';
b1 = [mu; zeros(k,1)];
b2 = [1; zeros(k,1)];
tic;
R= chol(A11); % cholesky factorization; we'll need A11^-1 several times
S=-A21*(R\ (R'\A12));
btilde = b2-A21*(R\ (R'\b1));
x2 = S\btilde;
x1 = R\ (R'\(b1-A12*x2)); %
wfast = x1(1:n);
toc;

rel_err = norm(wfast-wslow)/norm(wslow)

```

The times for the two methods are around 2 seconds (slow method) and 60 milliseconds (fast method). With a real language (*e.g.*, C or C++), the difference would be even more dramatic. Here is the corresponding Julia code. As of Winter 2015, Julia's sparse Cholesky factorization still has some issues that are being worked out, so we do not include that method here. The times were roughly 3 seconds for the slow method and 1.5 seconds for the fast method.

```

# efficient solution of basic portfolio optimization problem
# generate some random data
n = 2500; # number of assets
k = 30; # number of factors
srand(0);
F = randn(n,k);
d = 0.1 + rand(n);
Q = randn(k,k);

```

```

Q = Q*Q' + eye(k);
Sigma = diagm(d) + F*Q*F';
mu = rand(n);

# the slow way, solve full KKT
tic();
wnu = [Sigma ones(n); ones(1,n) 0] \ [mu ; 1];
toc();
wslow = wnu[1:n];

# fast method (specific code)
tic();
# form Schur complement (some inefficiencies here...)
inv_d = 1./d;
S= -[sum(inv_d) inv_d'*F; F'*inv_d F'*diagm(inv_d)*F+inv(Q)];
btilde = [1-ones(1,n)*(mu./d); -F*(mu./d)];
x2 = S\btilde;
nu = x2[1]; kappa = x2[2:k+1];
wfast = (mu-nu*ones(n)-F*kappa)./d;
toc();
rel_err = norm(wfast-wslow)/norm(wslow);
println(rel_err)

```

Here is the solution in Python. The times were roughly 0.34 seconds for the slow method and 0.02 seconds for the fast method.

```

# efficient solution of basic portfolio optimization problem
# generate some random data

import numpy as np
import scipy.sparse as sps
import scipy.sparse.linalg as splinalg
import time

n = 2500 # number of assets
k = 30 # number of factors
np.random.seed(0)

F = np.random.randn(n,k)
F = np.matrix(F)
d = 0.1 + np.random.rand(n)
d = np.matrix(d).T
Q = np.random.randn(k)
Q = np.matrix(Q).T
Q = Q * Q.T + np.eye(k)
Sigma = np.diag(d.A1) + F*Q*F.T

```

```

mu = np.random.rand(n)
mu = np.matrix(mu).T

# the slow way, solve full KKT
t = time.time()
kkt_matrix = np.vstack(
    (np.hstack((Sigma, np.ones((n,1)))),
     np.hstack((np.ones(n), [0.])))))
wnu = np.linalg.solve(kkt_matrix, np.vstack((mu, [1.])))
print "Elapsed time is %f seconds." % (time.time() - t)
wslow = wnu[:n]

# fast method (specific code)
t = time.time()
# form Schur complement (some inefficiencies here...)
S = -np.vstack(
    (np.hstack(([np.sum(1./d)], ((1./d).T*F).A1)),
     np.hstack((F.T*(1./d), F.T*np.diag(1./d.A1)*F+np.linalg.inv(Q))))
    )
btilde = np.vstack(
    ([1-np.sum(mu/d)],
     -F.T*(mu/d))
    )
x2 = np.linalg.solve(S,btilde)
nu = x2[0][0,0]
kappa = np.matrix(x2[1:k+1])
wfast = (mu-nu*np.ones((n,1))-F*kappa)/d
print "Elapsed time is %f seconds." % (time.time() - t)
rel_err = np.sqrt(np.sum((wfast-wslow).A1**2)/np.sum(wslow.A1**2))
print rel_err

# fast method (general code)
A11 = sps.vstack(
    sps.hstack((sps.diags([d.A1],[0]), np.zeros((n,k))),
               sps.hstack((sps.csc_matrix((k,n)), Q)))
    )
A12 = sps.vstack(
    sps.hstack((np.ones((n,1)), sps.csr_matrix(F))),
    sps.hstack((sps.csc_matrix((k,1)), sps.eye(k)))
    )
A21 = A12.T
b1 = np.vstack((mu, np.zeros((k,1))))
b2 = np.vstack((1., np.zeros((k,1))))
t = time.time()
factor_solve = splinalg.factorized(A11)

```

```

S = -A21*factor_solve(A12.todense())
btilde = b2-A21*factor_solve(b1)
x2 = np.linalg.solve(S,btilde)
x1 = factor_solve(b1-A12*x2)
wfast = x1[:n]
print "Elapsed time is %f seconds." % (time.time() - t)
rel_err = np.sqrt(np.sum((wfast-wslow).A1**2)/np.sum(wslow.A1**2))
print rel_err

```

- (d) Now we put λ back in to the KKT system:

$$\begin{bmatrix} \lambda \Sigma & \mathbf{1} \\ \mathbf{1}^T & 0 \end{bmatrix} \begin{bmatrix} w \\ \nu \end{bmatrix} = \begin{bmatrix} \mu \\ 1 \end{bmatrix},$$

Divide the first block row by λ , and change variables to $\tilde{\nu} = \nu/\lambda$ to get

$$\begin{bmatrix} \Sigma & \mathbf{1} \\ \mathbf{1}^T & 0 \end{bmatrix} \begin{bmatrix} w \\ \tilde{\nu} \end{bmatrix} = \begin{bmatrix} \mu/\lambda \\ 1 \end{bmatrix}.$$

This shows that the optimal portfolio is an affine function of $1/\lambda$. Therefore, w has the form $w = w_0 + (1/\lambda)w_1$, where w_0 is the optimal portfolio for $\mu = 0$ (or $\lambda = \infty$), and $w_0 + w_1$ is the optimal portfolio for $\lambda = 1$. We just solve for the optimal portfolio for these *two* values of λ (or any other two, for that matter!), and then we can construct the solution for any other value with $2n$ flops.

Thus we perform two solves, which costs $(2/3)nk^2$; then we need $2nM$ flops to find all the optimal portfolios. The complexity couldn't be any smaller, since to just write down the M optimal portfolios, we touch nM real numbers.

13.15 Sparse index tracking. The (weekly, say) return of n stocks is given by a random variable $r \in \mathbf{R}^n$, with mean \bar{r} and covariance $\mathbf{E}(r - \bar{r})(r - \bar{r})^T = \Sigma \succ 0$. An index (such as S&P 500 or Wilshire 5000) is a weighted sum of these returns, given by $z = c^T r$, where $c \in \mathbf{R}_+^n$. (For example, the vector c is nonzero only for the stocks in the index, and the coefficients c_i might be proportional to some measure of market capitalization of stock i .) We will assume that the index weights $c \in \mathbf{R}^n$, as well as the return mean and covariance \bar{r} and Σ , are known and fixed.

Our goal is to find a *sparse* weight vector $w \in \mathbf{R}^n$, which can include negative entries (meaning, short positions), so that the RMS index tracking error, defined as

$$E = \left(\frac{\mathbf{E}(z - w^T r)^2}{\mathbf{E} z^2} \right)^{1/2},$$

does not exceed 0.10 (*i.e.*, 10%). Of course, taking $w = c$ results in $E = 0$, but we are interested in finding a weight vector with (we hope) many fewer nonzero entries than c has.

Remark. This is the idea behind an *index fund*: You find a sparse portfolio that replicates or tracks the return of the index (within some error tolerance). Acquiring (and rebalancing) the sparse tracking portfolio will incur smaller transactions costs than trading in the full index.

- (a) Propose a (simple) heuristic method for finding a sparse weight vector w that satisfies $E \leq 0.10$.

- (b) Carry out your method on the problem instance given in `sparse_idx_track_data.m`. Give $\text{card}(w)$, the number of nonzero entries in w . (To evaluate $\text{card}(w)$, use `sum(abs(w)>0.01)`, which treats weight components smaller than 0.01 as zero.) (You might want to compare the index weights and the weights you find by typing `[c w]`. No need to print or turn in the resulting output, though.)

Solution. We have

$$\mathbf{E}(z - w^T r)^2 = \mathbf{E}((c - w)^T r)^2 = (c - w)^T (\Sigma + \bar{r}\bar{r}^T)(c - w),$$

so

$$E = \frac{\|(\Sigma + \bar{r}\bar{r}^T)^{1/2}(c - w)\|_2}{\|(\Sigma + \bar{r}\bar{r}^T)^{1/2}c\|_2}.$$

The heuristic is to choose w by solving the problem

$$\begin{aligned} &\text{minimize} && \|w\|_1 \\ &\text{subject to} && \|(\Sigma + \bar{r}\bar{r}^T)^{1/2}(c - w)\|_2 \leq 0.10 \|(\Sigma + \bar{r}\bar{r}^T)^{1/2}c\|_2. \end{aligned}$$

The code below carries this out for the given data.

```

sparse_idx_track_data
sigma_rr=Sigma+rbar*rbar';

cvx_begin
    variable w(n);
    minimize(norm(w,1))
    subject to
        quad_form(c-w,sigma_rr)<=0.10^2*quad_form(c,sigma_rr);
cvx_end
card_w = sum(abs(w)>0.01)

% we didn't ask you to do the following.
% we fix the chosen stocks, and optimize the weights for best tracking
zeros_idx = (abs(w)<0.01);
cvx_begin
    variable w2(n);
    minimize (quad_form(c-w2,sigma_rr));
    subject to
        w2=zeros_idx == 0;
cvx_end

E2 = sqrt(cvx_optval/quad_form(c,sigma_rr));

```

The method finds a weight vector with $\text{card}(w) = 50$. This can be compared to $\text{card}(c) = 500$. This weight vector (naturally) achieves a tracking error $E = 0.10$.

In fact we can improve this result, even though we didn't expect you to do any more. For example, we can just take the selected stocks from the basic method, and then minimize E over the weights for those. This reduces the tracking error to under 4%.

13.16 Option price bounds. In this problem we use the methods and results of example 5.10 to give bounds on the arbitrage-free price of an option. (See exercise 5.38 for a simple version of option pricing.) We will use all the notation and definitions from Example 5.10.

We consider here options on an underlying asset (such as a stock); these have a payoff or value that depends on S , the value of the underlying asset at the end of the investment period. We will assume that the underlying asset can only take on m different values, $S^{(1)}, \dots, S^{(m)}$. These correspond to the m possible scenarios or outcomes described in Example 5.10.

A risk-free asset has value $r > 1$ in every scenario.

A *put option* at *strike price* K gives the owner the right to sell one unit of the underlying stock at price K . At the end of the investment period, if the stock is trading at a price S , then the put option has payoff $(K - S)_+ = \max\{0, K - S\}$ (since the option is exercised only if $K > S$). Similarly a *call option* at strike price K gives the buyer the right to buy a unit of stock at price K . A call option has payoff $(S - K)_+ = \max\{0, S - K\}$.

A *collar* is an option with payoff

$$\begin{cases} C - S_0 & S > C \\ S - S_0 & F \leq S \leq C \\ F - S_0 & S < F \end{cases}$$

where F is the *floor*, C is the *cap* and S_0 is the price of the underlying at the start of the investment period. This option limits both the upside and downside of payoff.

Now we consider a specific problem. The price of the risk-free asset, with $r = 1.05$, is 1. The price of the underlying asset is $S_0 = 1$. We will use $m = 200$ scenarios, with $S^{(i)}$ uniformly spaced from $S^{(1)} = 0.5$ to $S^{(200)} = 2$. The following options are traded on an exchange, with prices listed below.

Type	Strike	Price
Call	1.1	0.06
Call	1.2	0.03
Put	0.8	0.02
Put	0.7	0.01

A collar with floor $F = 0.9$ and cap $C = 1.15$ is not traded on an exchange. Find the range of prices for this collar, consistent with the absence of arbitrage and the prices given above.

Solution. This problem is exactly the one described at the end of Example 5.10. Let's label the risk-free asset as asset 1, the underlying asset as asset 2, the four options traded on the exchange as assets 3–6, and the collar as asset 7. A set of 7 prices for these, denoted p , is consistent with the no-arbitrage assumption if and only if there is a $y \succeq 0$ with $V^T y = p$, where V is the value matrix as defined in Example 5.10. We are given the first 6 entries of p , and need to bound the last entry. The lower bound is found by solving the LP

$$\begin{array}{ll} \text{minimize} & p_n \\ \text{subject to} & V^T y = p \\ & y \succeq 0, \end{array}$$

with variables $p_n \in \mathbf{R}$ and $y \in \mathbf{R}^m$. To find the upper bound we maximize p_n instead. The following code solves the problem:

```
% ee364a option price bounds
% additional exercise solution

r=1.05; % risk-free rate
m=200; % scenarios
n=7; % assets
V=zeros(m,n); % value/payoff matrix
V(:,1) = r; % risk-free asset
V(:,2) = linspace(0.5,2,m); % underlying
% the four exchange traded options:
V(:,3) = pos(V(:,2) - 1.1);
V(:,4) = pos(V(:,2) - 1.2);
V(:,5) = pos(0.8-V(:,2));
V(:,6) = pos(0.7-V(:,2));
% collar option:
F=0.9;C=1.15;
V(:,7) = min(max(V(:,2)-1,F-1),C-1);

p = [1; 1; 0.06; 0.03; 0.02; 0.01]; % asset prices (from exchange)

cvx_begin
variables p_collar y(m)
minimize p_collar
%maximize p_collar
y>=0
V'*y== [p; p_collar]
cvx_end
```

The bounds were as follows:

```
[lb ub]=
0.033 0.065
```

We didn't ask you to carry out the following check, but had this been a real pricing exercise, you should have done it. We vary both the range of final underlying asset prices (which we arbitrarily took to be 0.5 and 2, representing a halving or doubling of value), and m , the number of samples of the final underlying asset value (which arbitrarily took to be the round number 200). Sure enough, varying these arbitrary parameters doesn't change the call option price bounds very much.

13.17 Portfolio optimization with qualitative return forecasts. We consider the risk-return portfolio optimization problem described on pages 155 and 185 of the book, with one twist: We don't precisely know the mean return vector \bar{p} . Instead, we have a range of possible values for each asset, *i.e.*, we have $l, u \in \mathbf{R}^n$ with $l \preceq \bar{p} \preceq u$. We use l and u to encode various qualitative forecasts we have

about the mean return vector \bar{p} . For example, $l_7 = 0.02$ and $u_7 = 0.20$ means that we believe the mean return for asset 7 is between 2% and 20%.

Define the *worst-case mean return* R^{wc} , as a function of portfolio vector x , as the worst (minimum) value of $\bar{p}^T x$, over all \bar{p} consistent with the given bounds l and u .

- (a) Explain how to find a portfolio x that maximizes R^{wc} , subject to a budget constraint and risk limit,

$$\mathbf{1}^T x = 1, \quad x^T \Sigma x \leq \sigma_{\max}^2,$$

where $\Sigma \in \mathbf{S}_{++}^n$ and $\sigma_{\max} \in \mathbf{R}_{++}$ are given.

- (b) Solve the problem instance given in `port_qual_forecasts_data.m`. Give the optimal worst-case mean return achieved by the optimal portfolio x^* .

In addition, construct a portfolio x^{mid} that maximizes $c^T x$ subject to the budget constraint and risk limit, where $c = (1/2)(l + u)$. This is the optimal portfolio assuming that the mean return has the midpoint value of the forecasts. Compare the midpoint mean returns $c^T x^{\text{mid}}$ and $c^T x^*$, and the worst-case mean returns of x^{mid} and x^* .

Briefly comment on the results.

Solution.

- (a) There are several ways to solve this problem. All depend on coming up with an explicit expression for R^{wc} that is clearly concave.

Here is one way. The value R^{wc} is the optimal value of the box-constrained LP

$$\begin{aligned} & \text{minimize} && \bar{p}^T x \\ & \text{subject to} && l \preceq \bar{p} \preceq u, \end{aligned}$$

with variable \bar{p} . As in exercise 4.8(c), we can choose an optimal \bar{p} as

$$\bar{p}_i^* = \begin{cases} l_i & x_i \geq 0 \\ u_i & x_i \leq 0. \end{cases}$$

The worst case mean return can be written as

$$\begin{aligned} R^{wc}(x) &= \bar{p}^* x \\ &= \sum_{i=1}^n (l_i \max\{0, x_i\} - u_i \max\{0, -x_i\}) \\ &= \frac{1}{2} \sum_{i=1}^n (l_i(x_i + |x_i|) + u_i(x_i - |x_i|)) \\ &= c^T x - \sum_{i=1}^n r_i |x_i|, \end{aligned}$$

where $c = (1/2)(l + u)$ and $r = (1/2)(u - l)$.

Thus the problem is

$$\begin{aligned} & \text{maximize} && c^T x - \sum_{i=1}^n r_i |x_i| \\ & \text{subject to} && \mathbf{1}^T x = 1, \quad x^T \Sigma x \leq \sigma_{\max}^2, \end{aligned}$$

with variable $x \in \mathbf{R}^n$, which is convex. This problem has a very nice interpretation. The objective is to maximize the midpoint mean return, minus a regularization term, which is a weighted ℓ_1 norm; the weights are one half the ‘spread’ in the forecast returns.

Here is another way to derive a nice expression for R^{wc} . We first note that the problem of finding the worst-case return splits across the assets, and always occurs with either the lower or upper limit of return. When $x_i \geq 0$, the worst-case return is obtained using l_i ; when $x_i \leq 0$, it is obtained using u_i . The associated worst case return for asset i is then $\min(l_i x_i, u_i x_i)$. Thus we have

$$R^{\text{wc}} = \sum_{i=1}^n \min(l_i x_i, u_i x_i).$$

This is evidently concave (indeed, it is DCP compliant). We get the problem

$$\begin{aligned} & \text{maximize} && \sum_{i=1}^n \min(l_i x_i, u_i x_i) \\ & \text{subject to} && \mathbf{1}^T x = 1, \quad x^T \Sigma x \leq \sigma_{\max}^2. \end{aligned}$$

Following the same idea, we can split x into a positive and negative vector, as $x = x_+ - x_-$, with $x_+, x_- \succeq 0$. Then we write the objective as $l^T x_+ - u^T x_-$, which gives

$$\begin{aligned} & \text{maximize} && l^T x_+ - u^T x_- \\ & \text{subject to} && \mathbf{1}^T(x_+ - x_-) = 1 \\ & && (x_+ - x_-)^T \Sigma (x_+ - x_-) \leq \sigma_{\max}^2 \\ & && x_+ \succeq 0, \quad x_- \succeq 0. \end{aligned}$$

Of course all these methods are equivalent; they just look a little different.

- (b) The following code constructs an optimal portfolio, as well as the optimal portfolio assuming the mean return takes on the midpoint forecast value. These two portfolios are

x	x^{mid}
0.0000	-0.1550
-0.9395	-1.0726
0.0000	0.9360
0.3559	0.8118
-0.0000	-0.7599
0.0813	0.5137
1.2577	1.0699
0.2249	-0.4942
0.0000	-0.1051
0.0197	0.2553

They achieve midpoint returns 0.38, 0.55, and worst-case returns 0.17, -0.08.

Brief comments. If we assume the return will be the midpoint value, the worst-case return is actually a loss! It’s also interesting to note that the portfolio that maximizes worst-case return is sparse; it doesn’t invest in 4 assets. These happen to be ones that have large ranges in the forecast mean return.

```

%% portfolio optimization with qualitative return forecasts

port_qual_forecasts_data

c = (1 + u)/2;
r = (u - 1)/2;

cvx_begin
    variable x(n)
    maximize(c'*x - r'*abs(x))
    subject to
        sum(x) == 1;
        quad_form(x,Sigma) <= sigma_max^2;
cvx_end

% assume mean return takes on midpoint forecast value
cvx_begin
    variable x_mid(n)
    maximize(c'*x_mid)
    subject to
        sum(x_mid) == 1;
        quad_form(x_mid,Sigma) <= sigma_max^2;
cvx_end

[ x x_mid ]

fprintf('Midpoint return for x: %.2f\n', ...
    (c'*x));
fprintf('Midpoint return for x_mid: %.2f\n', ...
    (c'*x_mid));
fprintf('Worst-case return for x: %.2f\n', ...
    (c'*x - r'*abs(x)));
fprintf('Worst-case return for x_mid: %.2f\n', ...
    (c'*x_mid - r'*abs(x_mid)));

```

- 13.18 De-leveraging.** We consider a multi-period portfolio optimization problem, with n assets and T time periods, where $x_t \in \mathbf{R}^n$ gives the holdings (say, in dollars) at time t , with negative entries denoting, as usual, short positions. For each time period the return vector has mean $\mu \in \mathbf{R}^n$ and covariance $\Sigma \in \mathbf{S}_{++}^n$. (These are known.)

The initial portfolio x_0 maximizes the risk-adjusted expected return $\mu^T x - \gamma x^T \Sigma x$, where $\gamma > 0$, subject to the leverage limit constraint $\|x\|_1 \leq L^{\text{init}}$, where $L^{\text{init}} > 0$ is the given initial leverage limit. (There are several different ways to measure leverage; here we use the sum of the total short and long positions.) The final portfolio x_T maximizes the risk-adjusted return, subject to $\|x\|_1 \leq L^{\text{new}}$, where $L^{\text{new}} > 0$ is the given final leverage limit (with $L^{\text{new}} < L^{\text{init}}$). This uniquely determines x_0 and x_T , since the objective is strictly concave.

The question is how to move from x_0 to x_T , *i.e.*, how to choose x_1, \dots, x_{T-1} . We will do this so as to maximize the objective

$$J = \sum_{t=1}^T \left(\mu^T x_t - \gamma x_t^T \Sigma x_t - \phi(x_t - x_{t-1}) \right),$$

which is the total risk-adjusted expected return, minus the total transaction cost. The transaction cost function ϕ has the form

$$\phi(u) = \sum_{i=1}^n (\kappa_i |u_i| + \lambda_i u_i^2),$$

where $\kappa \succeq 0$ and $\lambda \succeq 0$ are known parameters. We will require that $\|x_t\|_1 \leq L^{\text{init}}$, for $t = 1, \dots, T-1$. In other words, the leverage limit is the initial leverage limit up until the deadline T , when it drops to the new lower value.

- (a) Explain how to find the portfolio sequence x_1^*, \dots, x_{T-1}^* that maximizes J subject to the leverage limit constraints.
- (b) Find the optimal portfolio sequence x_t^* for the problem instance with data given in `deleveraging_data.m`. Compare this sequence with two others: $x_t^{\text{LP}} = x_0$ for $t = 1, \dots, T-1$ (*i.e.*, one that does all trading at the last possible period), and the linearly interpolated portfolio sequence

$$x_t^{\text{lin}} = (1-t/T)x_0 + (t/T)x_T, \quad t = 1, \dots, T-1.$$

For each of these three portfolio sequences, give the objective value obtained, and plot the risk and transaction cost adjusted return,

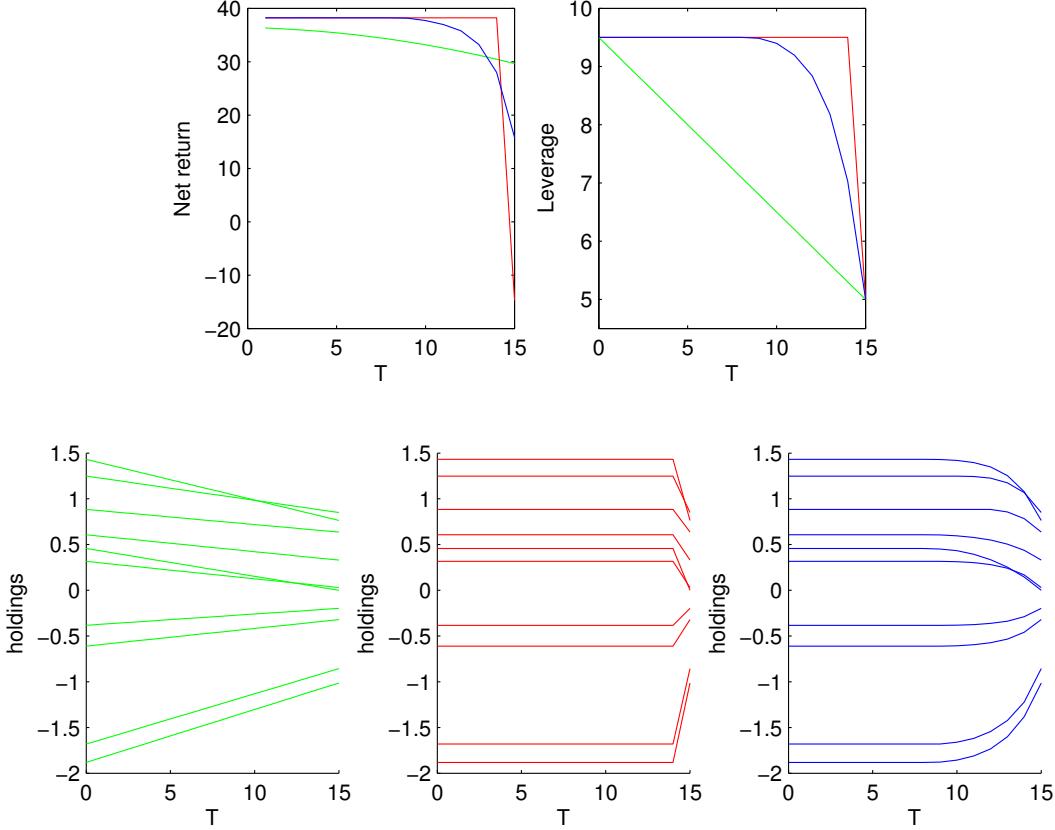
$$\mu^T x_t - \gamma x_t^T \Sigma x_t - \phi(x_t - x_{t-1}),$$

and the leverage $\|x_t\|_1$, versus t , for $t = 0, \dots, T$. Also, for each of the three portfolio sequences, generate a single plot that shows how the holdings $(x_t)_i$ of the n assets change over time, for $i = 1, \dots, n$.

Give a *very short* (one or two sentence) intuitive explanation of the results.

Solution.

- (a) It is easy to see that ϕ is convex in u . By the composition rule, it follows that each period cost at time t is concave in x_1, \dots, x_{T-1} . Then, maximizing J subject to the leverage limit constraints is a convex optimization problem. The constraints for this optimization problem are straightforward: $\|x_i\|_1 \leq L^{\text{init}}, i = 1, \dots, T-1$, $x_0 = x_0^*$ and $x_T = x_T^*$, where x_0^* and x_T^* are both given by maximizing the risk adjusted return (without transaction cost) subject to initial and last leverage limit.
- (b) The total net returns for the linearly interpolated portfolio sequence, the portfolio sequence that does all the trading at the last period, and the optimal sequence are 506.18, 520.42 and 531.38 respectively. In the following graphs, the three sequences are drawn in green, red, and blue, in this order.



Intuitive explanation. The linearly interpolated portfolio smoothly changes its positions, so it doesn't incur much transaction cost. On the other hand, it starts deleveraging well before the deadline, so it loses some opportunity to improve the objective in the early steps. When we make all trades in the last period, we get a good objective value up until then, but then we pay for it in transaction costs in the last period, and the total objective suffers. When you transition just right, you combine the best of both strategies: you keep the portfolio near the optimal one for high leverage for about 10 periods, then smoothly transition to the new portfolio, in such a way that the total objective is maximized.

The following code solve the problem.

```
% de-leveraging
deleveraging_data;

% compute initial and final portfolio
cvx_begin quiet
    variable x0_star(n)
    maximize (mu'*x0_star - gamma*quad_form(x0_star, Sigma))
    subject to
        norm(x0_star, 1) <= Linit;
cvx_end

cvx_begin quiet
```

```

variable xT_star(n)
maximize (mu'*xT_star - gamma*quad_form(xT_star, Sigma))
subject to
    norm(xT_star, 1) <= Lnew;
cvx_end

% linear interpolation
Xlin = zeros(n, T+1);
for i = 1:T+1
    Xlin(:, i) = (1-(i-1)/T)*x0_star + (i-1)/T*xT_star;
end

% trading at the last period
Xlp = [repmat(x0_star, 1, T) xT_star];

% optimization
cvx_begin quiet
    variable X(n, T+1)
    expressions dX(n, T) ret trs
    dX = X(:, 2:T+1) - X(:, 1:T);
    ret = 0; trs = 0;
    for t = 2:T+1
        ret = ret + mu'*X(:, t) - gamma*quad_form(X(:, t), Sigma);
        trs = trs + kappa'*abs(dX(:, t-1)) ...
            + quad_form(dX(:, t-1), diag(lambda));
    end

    maximize ret - trs
    subject to
        X(:, 1) == x0_star;
        X(:, T+1) == xT_star;
        norms(X(:, 2:T), 1) <= Linit;
cvx_end

levOpt = norms(X, 1);
levLin = norms(Xlin, 1);
levLp = norms(Xlp, 1);

retOpt = X'*mu - gamma*diag(X'*Sigma*X);
retLin = Xlin'*mu - gamma*diag(Xlin'*Sigma*Xlin);
retLp = Xlp'*mu - gamma*diag(Xlp'*Sigma*Xlp);

dXlin = Xlin(:, 2:T+1) - Xlin(:, 1:T);
dXLp = Xlp(:, 2:T+1) - Xlp(:, 1:T);

```

```

trsOpt = abs(dX)'*kappa + diag(dX'*diag(lambda)*dX);
trsLin = abs(dXlin)'*kappa + diag(dXlin'*diag(lambda)*dXlin);
trsLp = abs(dXlp)'*kappa + diag(dXlp'*diag(lambda)*dXlp);

netReturnOpt = retOpt(2:T+1) - trsOpt;
netReturnLin = retLin(2:T+1) - trsLin;
netReturnLp = retLp(2:T+1) - trsLp;

fprintf('J of x^lin : %f\n', sum(netReturnLin));
fprintf('J of x^lp : %f\n', sum(netReturnLp));
fprintf('J of x^star: %f\n', sum(netReturnOpt));

% plot
clf;
subplot(2, 3, 1.5);
plot(1:T, netReturnLin, 'g', 1:T, netReturnLp, 'r', ...
      1:T, netReturnOpt, 'b');
xlabel('T'); ylabel('Net return'); xlim([0, T]);

subplot(2, 3, 2.5);
plot(0:T, levLin, 'g', 0:T, levLp, 'r', 0:T, levOpt, 'b');
xlabel('T'); ylabel('Leverage'); axis([0 T Lnew-0.5 Linit+0.5]);

subplot(2, 3, 3); % x^lin
hold on;
for i = 1:n
    plot(0:T, Xlin(i, :), 'g');
end
hold off;
xlabel('T'); ylabel('holdings'); xlim([0 T]);

subplot(2, 3, 4); % x^lp
hold on;
for i = 1:n
    plot(0:T, Xlp(i, :), 'r');
end
hold off;
xlabel('T'); ylabel('holdings'); xlim([0 T]);

subplot(2, 3, 5); % x^star
hold on;
for i = 1:n
    plot(0:T, X(i, :), 'b');
end
hold off;

```

```

xlabel('T'); ylabel('holdings'); xlim([0 T]);
print -depsc deleveraging.eps;

```

- 13.19 Worst-case variance.** Suppose Z is a random variable on \mathbf{R}^n with covariance matrix $\Sigma \in \mathbf{S}_+^n$. Let $c \in \mathbf{R}^n$. The variance of $Y = c^T Z$ is $\text{var}(Y) = c^T \Sigma c$. We define the *worst-case variance* of Y , denoted $\text{wcvar}(Y)$, as the maximum possible value of $c^T \tilde{\Sigma} c$, over all $\tilde{\Sigma} \in \mathbf{S}_+^n$ that satisfy $\Sigma_{ii} = \tilde{\Sigma}_{ii}$, $i = 1, \dots, n$. In other words, the worst-case variance of Y is the maximum possible variance, if we are allowed to arbitrarily change the correlations between Z_i and Z_j . Of course we have $\text{wcvar}(Y) \geq \text{var}(Y)$.

- (a) Find a simple expression for $\text{wcvar}(Y)$ in terms of c and the diagonal entries of Σ . You must justify your expression.
- (b) *Portfolio optimization.* Explain how to find the portfolio $x \in \mathbf{R}^n$ that maximizes the expected return $\mu^T x$ subject to a limit on risk, $\text{var}(r^T x) = x^T \Sigma x \leq R$, and a limit on worst-case risk $\text{wcvar}(r^T x) \leq R^{\text{wc}}$, where $R > 0$ and $R^{\text{wc}} > R$ are given. Here $\mu = \mathbf{E} r$ and $\Sigma = \mathbf{E}(r - \mu)(r - \mu)^T$ are the (given) mean and covariance of the (random) return vector $r \in \mathbf{R}^n$.
- (c) Carry out the method of part (b) for the problem instance with data given in `wc_risk_portfolio_opt_data.m`. Also find the optimal portfolio when the worst-case risk limit is ignored. Find the expected return and worst-case risk for these two portfolios.

Remark. If a portfolio is highly leveraged, and the correlations in the returns change drastically, you (the portfolio manager) can be in big trouble, since you are now exposed to much more risk than you thought you were. And yes, this (almost exactly) has happened.

Solution.

- (a) We will show that

$$\text{wcvar}(Y) = \left(\sum_{i=1}^n |c_i| \sqrt{\Sigma_{ii}} \right)^2.$$

This worst-case variance value is obtained by the matrix $\Sigma^{\text{wc}} = dd^T$, where $d \in \mathbf{R}^n$ is defined as

$$d_i = \text{sign}(c_i) \sqrt{\Sigma_{ii}}, \quad i = 1, \dots, n.$$

Note that $\Sigma^{\text{wc}} \in \mathbf{S}_+^n$ and its diagonal entries are the same as Σ .

To show this (and also, to derive it) we proceed as follows. For any $\tilde{\Sigma} \in \mathbf{S}_+^n$ that satisfies $\Sigma_{ii} = \tilde{\Sigma}_{ii}$ for all i , we have

$$\begin{aligned} c^T \tilde{\Sigma} c &= \sum_{i=1}^n c_i^2 \tilde{\Sigma}_{ii} + \sum_{i \neq j} c_i c_j \tilde{\Sigma}_{ij} \\ &= \sum_{i=1}^n c_i^2 \Sigma_{ii} + \sum_{i \neq j} c_i c_j \tilde{\Sigma}_{ij}. \end{aligned}$$

Since $\tilde{\Sigma} \succeq 0$, we must have

$$|\tilde{\Sigma}_{ij}| \leq \sqrt{\tilde{\Sigma}_{ii} \tilde{\Sigma}_{jj}} = \sqrt{\Sigma_{ii} \Sigma_{jj}}$$

for $i \neq j$. If we maximize $c^T \tilde{\Sigma} c$ over the off-diagonal elements of $\tilde{\Sigma}$ subject to this limit, we find that the maximum occurs when

$$\tilde{\Sigma}_{ij} = \mathbf{sign}(c_i c_j) \sqrt{\Sigma_{ii} \Sigma_{jj}}$$

for $i \neq j$. We can re-write this as

$$\tilde{\Sigma}_{ij} = \sqrt{\Sigma_{ii} \Sigma_{jj}} \mathbf{sign}(c_i) \mathbf{sign}(c_j), \quad i, j = 1, \dots, n.$$

So far, we have not taken into account the constraint that $\tilde{\Sigma} \succeq 0$. Fortunately, the matrix above satisfies this, since we can express it as $\tilde{\Sigma} = dd^T$.

- (b) Given the data μ , Σ , R , and R^{wc} , we solve the optimization problem

$$\begin{aligned} & \text{maximize} && \mu^T x \\ & \text{subject to} && x^T \Sigma x \leq R \\ & && \sum_{i=1}^n |x_i| \sqrt{\Sigma_{ii}} \leq \sqrt{R^{wc}}. \end{aligned}$$

- (c) The optimal portfolio with the worst-case risk limit is

$$x^* = (2.263, 0, 0, 0, 0, 0, 0, 0.698, -1.673, 0),$$

with expected return $\mu^T x^* = 8.195$, and worst case risk $\mathbf{wcvvar}(r^T x^*) = 10$.

The optimal portfolio with the worst-case risk limit ignored is

$$x^* = (1.625, 1.214, -0.672, -1.821, 2.395, -0.892, 0.533, 3.967, -0.565, -2.769),$$

with expected return $\mu^T x^* = 13.512$, and worst case risk $\mathbf{wcvvar}(r^T x^*) = 117.197$.

The following code solves the problem.

```
% worst-case variance
wc_risk_portfolio_opt_data;

cvx_begin quiet
variable x(n)
maximize (mu'*x)
subject to
    quad_form(x, Sigma) <= R;
    norm(sqrt(diag(Sigma)).*x, 1) <= sqrt(R_wc);
cvx_end

fprintf('Portfolio with the worst-case risk limit:\n')
fprintf(' Expected return: %.3f\n', mu'*x);
fprintf(' Variance: %.3f\n', quad_form(x, Sigma));
fprintf(' Worst case risk: %.3f\n', norm(sqrt(diag(Sigma)).*x, 1)^2);
fprintf(' x = (');

for i = 1:n
    fprintf('%.3f, ', x(i));
```

```

end
fprintf(')\n\n');

cvx_begin quiet
variable x(n)
maximize (mu'*x)
subject to
    quad_form(x, Sigma) <= R;
cvx_end

fprintf('Portfolio without the worst-case risk limit:\n')
fprintf(' Expected return: %.3f\n', mu'*x);
fprintf(' Variance: %.3f\n', quad_form(x, Sigma));
fprintf(' Worst case risk: %.3f\n', norm(sqrt(diag(Sigma)).*x, 1)^2);
fprintf(' x = (');
for i = 1:n
    fprintf('%.3f, ', x(i));
end
fprintf(')\n');

```

- 13.20 Risk budget allocation.** Suppose an amount $x_i > 0$ is invested in n assets, labeled $i = 1, \dots, n$, with asset return covariance matrix $\Sigma \in \mathbf{S}_{++}^n$. We define the *risk* of the investments as the standard deviation of the total return, $R(x) = (x^T \Sigma x)^{1/2}$.

We define the (relative) *risk contribution* of asset i (in the portfolio x) as

$$\rho_i = \frac{\partial \log R(x)}{\partial \log x_i} = \frac{\partial R(x)}{R(x)} \frac{x_i}{\partial x_i}, \quad i = 1, \dots, n.$$

Thus ρ_i gives the fractional increase in risk per fractional increase in investment i . We can express the risk contributions as

$$\rho_i = \frac{x_i(\Sigma x)_i}{x^T \Sigma x}, \quad i = 1, \dots, n,$$

from which we see that $\sum_{i=1}^n \rho_i = 1$. For general x , we can have $\rho_i < 0$, which means that a small increase in investment i decreases the risk. Desirable investment choices have $\rho_i > 0$, in which case we can interpret ρ_i as the fraction of the total risk contributed by the investment in asset i . Note that the risk contributions are homogeneous of degree zero, *i.e.*, scaling x by a positive constant does not affect ρ_i .

In the *risk budget allocation problem*, we are given Σ and a set of desired risk contributions $\rho_i^{\text{des}} > 0$ with $\mathbf{1}^T \rho^{\text{des}} = 1$; the goal is to find an investment mix $x \succ 0$, $\mathbf{1}^T x = 1$, with these risk contributions. When $\rho^{\text{des}} = (1/n)\mathbf{1}$, the problem is to find an investment mix that achieves so-called *risk parity*.

- (a) Explain how to solve the risk budget allocation problem using convex optimization.

Hint. Minimize $(1/2)x^T \Sigma x - \sum_{i=1}^n \rho_i^{\text{des}} \log x_i$.

- (b) Find the investment mix that achieves risk parity for the return covariance matrix

$$\Sigma = \begin{bmatrix} 6.1 & 2.9 & -0.8 & 0.1 \\ 2.9 & 4.3 & -0.3 & 0.9 \\ -0.8 & -0.3 & 1.2 & -0.7 \\ 0.1 & 0.9 & -0.7 & 2.3 \end{bmatrix}.$$

For your convenience, this is contained in `risk_alloc_data.m`.

Solution.

- (a) Consider the problem of minimizing $f(x)$, with

$$f(x) = (1/2)x^T \Sigma x - \sum_{i=1}^n \rho_i^{\text{des}} \log x_i.$$

The optimality condition for this problem is

$$\nabla f(x) = \Sigma x - \text{diag}(x)^{-1} \rho^{\text{des}} = 0.$$

Hence, at the optimal point x^* , we have

$$x_i^* (\Sigma x^*)_i = \rho_i^{\text{des}}.$$

The constraint $\mathbf{1}^T \rho^{\text{des}} = 1$ implies that $x^{*T} \Sigma x^* = \sum_{i=1}^n x_i^* (\Sigma x^*)_i = 1$. Therefore, we also have

$$\rho_i^{\text{des}} = \frac{x_i^* (\Sigma x^*)_i}{x^{*T} \Sigma x^*}.$$

Note that this equation is homogeneous in x . So choosing $\hat{x} = x^*/(\mathbf{1}^T x^*)$ will do the trick. Since the objective function is strictly convex, the minimizer is unique; hence, the optimal investment mix is unique.

- (b) The following code implements the solution:

```

risk_alloc_data
n = size(Sigma,1);
rho = ones(n,1)/n;

cvx_begin
    variable x(n)
    minimize (1/2*quad_form(x,Sigma) - log( geo_mean(x)))
%    could also use:
%    minimize (1/2*quad_form(x,Sigma) - rho'*log(x))
cvx_end
x_hat = x/sum(x);

% verify risk parity
x_hat.*((Sigma*x_hat)./quad_form(x_hat,Sigma))

% compare with uniform allocation
u = ones(n,1)/n;
u.*((Sigma*u)./quad_form(u,Sigma))

```

The optimal investment mix is: $\hat{x} = (.1377, .1134, .4759, .2731)$.

13.21 Portfolio rebalancing. We consider the problem of rebalancing a portfolio of assets over multiple periods. We let $h_t \in \mathbf{R}^n$ denote the vector of our dollar value holdings in n assets, at the beginning of period t , for $t = 1, \dots, T$, with negative entries meaning short positions. We will work with the portfolio weight vector, defined as $w_t = h_t / (\mathbf{1}^T h_t)$, where we assume that $\mathbf{1}^T h_t > 0$, i.e., the total portfolio value is positive.

The *target portfolio weight vector* w^* is defined as the solution of the problem

$$\begin{aligned} &\text{maximize} && \mu^T w - \frac{\gamma}{2} w^T \Sigma w \\ &\text{subject to} && \mathbf{1}^T w = 1, \end{aligned}$$

where $w \in \mathbf{R}^n$ is the variable, μ is the mean return, $\Sigma \in \mathbf{S}_{++}^n$ is the return covariance, and $\gamma > 0$ is the risk aversion parameter. The data μ , Σ , and γ are given. In words, the target weights maximize the risk-adjusted expected return.

At the beginning of each period t we are allowed to rebalance the portfolio by buying and selling assets. We call the post-trade portfolio weights \tilde{w}_t . They are found by solving the (rebalancing) problem

$$\begin{aligned} &\text{maximize} && \mu^T w - \frac{\gamma}{2} w^T \Sigma w - \kappa^T |w - w_t| \\ &\text{subject to} && \mathbf{1}^T w = 1, \end{aligned}$$

with variable $w \in \mathbf{R}^n$, where $\kappa \in \mathbf{R}_+^n$ is the vector of (so-called linear) transaction costs for the assets. (For example, these could model bid/ask spread.) Thus, we choose the post-trade weights to maximize the risk-adjusted expected return, minus the transactions costs associated with rebalancing the portfolio. Note that the pre-trade weight vector w_t is known at the time we solve the problem. If we have $\tilde{w}_t = w_t$, it means that no rebalancing is done at the beginning of period t ; we simply hold our current portfolio. (This happens if $w_t = w^*$, for example.)

After holding the rebalanced portfolio over the investment period, the dollar value of our portfolio becomes $h_{t+1} = \text{diag}(r_t)\tilde{h}_t$, where $r_t \in \mathbf{R}_{++}^n$ is the (random) vector of asset returns over period t , and \tilde{h}_t is the post-trade portfolio given in dollar values (which you do not need to know). The next weight vector is then given by

$$w_{t+1} = \frac{\text{diag}(r_t)\tilde{w}_t}{r_t^T \tilde{w}_t}.$$

(If $r_t^T \tilde{w}_t \leq 0$, which means our portfolio has negative value after the investment period, we have gone bust, and all trading stops.) The standard model is that r_t are IID random variables with mean and covariance μ and Σ , but this is not relevant in this problem.

(a) *No-trade condition.* Show that $\tilde{w}_t = w_t$ is optimal in the rebalancing problem if

$$\gamma |\Sigma(w_t - w^*)| \leq \kappa$$

holds, where the absolute value on the left is elementwise.

Interpretation. The lefthand side measures the deviation of w_t from the target portfolio w^* ; when this deviation is smaller than the cost of trading, you do not rebalance.

Hint. Find dual variables, that with $w = w_t$ satisfy the KKT conditions for the rebalancing problem.

- (b) Starting from $w_1 = w^*$, compute a sequence of portfolio weights \tilde{w}_t for $t = 1, \dots, T$. For each t , find \tilde{w}_t by solving the rebalancing problem (with w_t a known constant); then generate a vector of returns r_t (using our supplied function) to compute w_{t+1} (The sequence of weights is random, so the results won't be the same each time you run your script. But they should look similar.)

Report the fraction of periods in which the no-trade condition holds and the fraction of periods in which the solution has only zero (or negligible) trades, defined as $\|\tilde{w}_t - w_t\|_\infty \leq 10^{-3}$. Plot the sequence \tilde{w}_t for $t = 1, 2, \dots, T$.

The file `portf_weight_rebalance_data.*` provides the data, a function to generate a (random) vector r_t of market returns, and the code to plot the sequence \tilde{w}_t . (The plotting code also draws a dot for every non-negligible trade.)

Carry this out for two values of κ , $\kappa = \kappa_1$ and $\kappa = \kappa_2$. Briefly comment on what you observe.

Hint. In CVXPY we recommend using the solver ECOS. But if you use SCS you should increase the default accuracy, by passing `eps=1e-4` to the `cvxpy.Problem.solve()` method.

Solution.

- (a) *No-trade condition.* The solution w^* of the problem without transaction costs satisfies the KKT conditions

$$-\mu + \gamma \Sigma w^* + \nu^* \mathbf{1} = 0, \quad \mathbf{1}^T w^* = 1,$$

where $\nu^* \in \mathbf{R}$ is an optimal dual variable for the constraint $\mathbf{1}^T w = 1$. (This is a set of linear equations that we can easily solve, but we don't need this fact for this problem.)

Now we derive optimality conditions for the rebalancing problem. First we express it in the form

$$\begin{aligned} & \text{maximize} && \mu^T w - \frac{\gamma}{2} w^T \Sigma w - \kappa^T s \\ & \text{subject to} && \mathbf{1}^T w = 1, \\ & && w - w_t \preceq s \\ & && w_t - w \preceq s, \end{aligned}$$

with additional (slack) variable $s \in \mathbf{R}^n$. Defining dual variables ν , λ_+ , and λ_- for the three constraints, the optimality conditions are

$$\begin{aligned} -\mu + \gamma \Sigma w + \nu \mathbf{1} + \lambda_+ - \lambda_- &= 0 \\ \kappa - \lambda_+ - \lambda_- &= 0 \\ \lambda_+ &\succeq 0 \\ \lambda_- &\succeq 0 \\ \lambda_+^T (w - w_t - s) &= 0 \\ \lambda_-^T (w_t - w - s) &= 0 \\ \mathbf{1}^T w &= 1 \\ w - w_t &\preceq s \\ w_t - w &\preceq s. \end{aligned}$$

We want to find a condition under which $w = w_t$, $s = 0$ is optimal. (This means we do not trade.) With these choices for primal variables, the last 5 conditions hold. So we need to

find dual variables so that the first 4 conditions hold. We'll use the dual variable ν^* from the original problem, and seek λ_+ and λ_- so that the following 4 conditions hold:

$$\begin{aligned} -\mu + \gamma \Sigma w_t + \nu^* \mathbf{1} + \lambda_+ - \lambda_- &= 0 \\ \kappa - \lambda_+ - \lambda_- &= 0 \\ \lambda_+ &\succeq 0 \\ \lambda_- &\succeq 0. \end{aligned}$$

The first and second conditions can be written (subtracting the optimal solution of the problem without transaction costs)

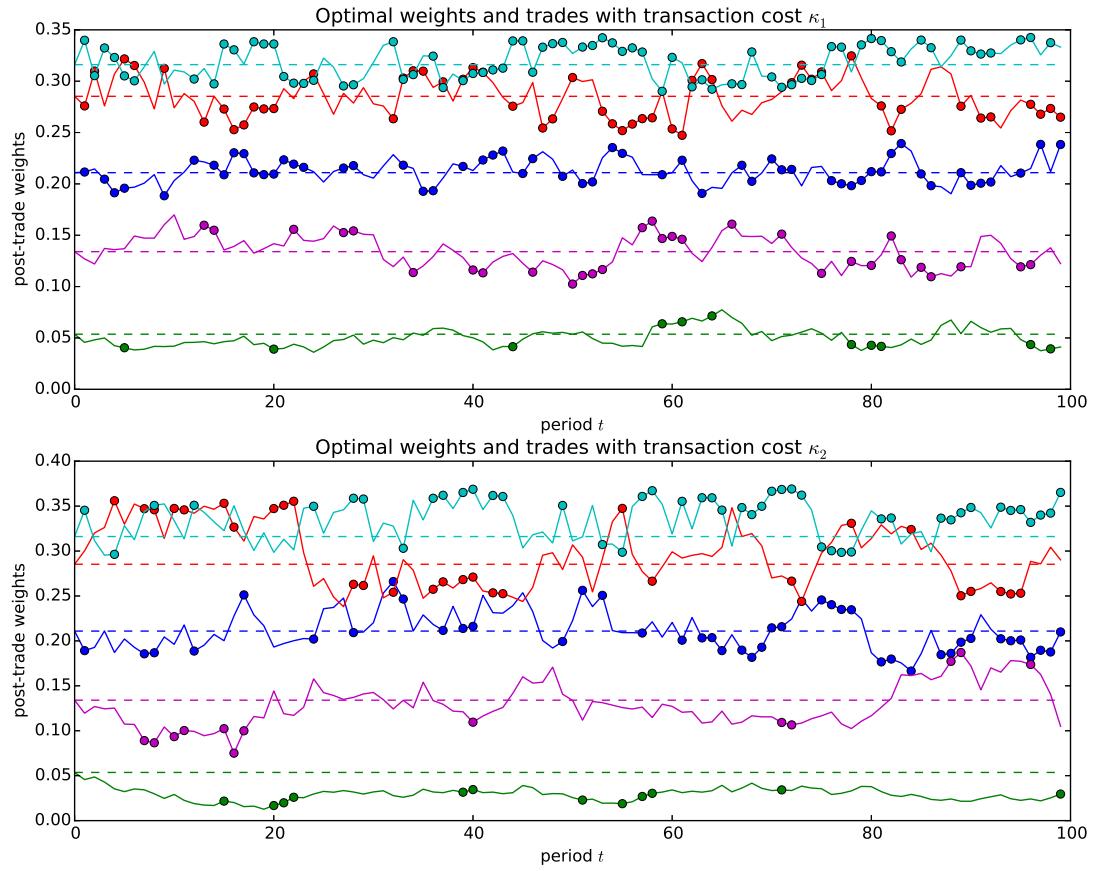
$$\gamma \Sigma (w_t - w^*) + \lambda_+ - \lambda_- = 0, \quad \kappa = \lambda_+ + \lambda_-.$$

This holds for some $\lambda_+ \succeq 0$ and $\lambda_- \succeq 0$ if and only if

$$|\gamma \Sigma (w_t - w^*)| \preceq \kappa.$$

This is what we were supposed to show. Under this condition, $w = w_t$ is optimal for the rebalancing problem, *i.e.*, we do not trade.

- (b) We report the average plus or minus one standard deviations of the results, computed over multiple runs of the solution script. (The results vary since the returns are generated randomly.) With $\kappa = \kappa_1$, the no-trade condition holds $(10 \pm 3)\%$ of the times and the trades are negligible $(17 \pm 4)\%$ of the times. With $\kappa = \kappa_2$ the no-trade condition holds $(27 \pm 6)\%$ of the times and the trades are negligible $(38 \pm 7)\%$ of the times. We observe that the no-trade condition holds for less periods than the periods with negligible trades. This is correct since the no-trade condition is a sufficient but not necessary condition for having negligible trades. We also observe that with transaction costs κ_2 we rebalance less than with κ_1 , since $\kappa_2 \succeq \kappa_1$. The plots show that the sequence of \tilde{w}_t deviate randomly from w^* and the trades tend to bring it closer to w^* (that is in fact rebalancing). By increasing the transaction costs, $\kappa_2 \succeq \kappa_1$, we rebalance less and let the weights diverge farther from the target w^* .



The following Python code solves the problem:

```

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

T = 100
n = 5
gamma = 8.0
Sigma = np.array([[ 1.512e-02,  1.249e-03,  2.762e-04, -5.333e-03, -7.938e-04],
                 [ 1.249e-03,  1.030e-02,  6.740e-05, -1.301e-03, -1.937e-04],
                 [ 2.762e-04,  6.740e-05,  1.001e-02, -2.877e-04, -4.283e-05],
                 [-5.333e-03, -1.301e-03, -2.877e-04,  1.556e-02,  8.271e-04],
                 [-7.938e-04, -1.937e-04, -4.283e-05,  8.271e-04,  1.012e-02]])
mu = np.array([ 1.02 , 1.028, 1.01 , 1.034, 1.017])
kappa_1 = np.array([ 0.002, 0.002, 0.002, 0.002, 0.002])
kappa_2 = np.array([ 0.004, 0.004, 0.004, 0.004, 0.004])
threshold = 0.001

```

```

## Solve target weights problem
w = cvx.Variable(n)
cvx.Problem(cvx.Maximize(w.T*mu - (gamma/2.)*cvx.quad_form(w, Sigma)),
            [cvx.sum_entries(w) == 1]).solve()
w_star = w.value.A1

generateReturns = lambda: np.random.multivariate_normal(mu,Sigma)

## Generate market scenario
plt.figure(figsize=(13,10))
for i, kappa in [(1,kappa_1), (2,kappa_2)]:
    ws = np.zeros((T,n))
    us = np.zeros((T,n))
    no_trade_cond = np.zeros(T)
    w_t = w_star
    for t in range(T):
        # check if no-trade condition holds
        no_trade_cond[t]= max(gamma*np.abs(np.dot(Sigma, w_t - w_star)) -
                               kappa) <= 0
        w = cvx.Variable(n)
        cvx.Problem(cvx.Maximize(w.T*mu -
                                  (gamma/2.)*cvx.quad_form(w, Sigma) -
                                  cvx.sum_entries(cvx.abs(w - w_t).T*kappa)),
                    [cvx.sum_entries(w) == 1]).solve()
        ws[t,:] = w.value.A1
        us[t,:] = w.value.A1 - w_t
        w_t = w.value.A1 * generateReturns()
        w_t /= sum(w_t)

    neglig_trades = np.max(np.abs(us),1) < threshold
    print "The no-trade condition holds %.1f%% of the times."%(sum(no_trade_cond)*100./T)
    print "The optimal solution has neglig. trades %.1f%% of the times."%(sum(neglig_trades)*100./T)

    plt.subplot(210+i)
    colors = ['b','r','g','c','m']
    for j in range(n):
        plt.plot(range(T), ws[:,j], colors[j])
        plt.plot(range(T), [w_star[j]]*T, colors[j]+'--')
        non_zero_trades = abs(us[:,j]) > threshold
        plt.plot(np.arange(T)[non_zero_trades],
                 ws[non_zero_trades, j], colors[j]+'\o')
    plt.ylabel('post-trade weights')

```

```

plt.xlabel('period $t$')
plt.title('Optimal weights and trades with transaction cost $\kappa_{\%d\%i}$')
plt.savefig("portfolio_weights.eps")

The following Matlab code solves the problem:

T = 100;
n = 5;
gamma = 8.0;
threshold = 0.001;
Sigma = [[ 1.512e-02, 1.249e-03, 2.762e-04, -5.333e-03, -7.938e-04],
          [ 1.249e-03, 1.030e-02, 6.740e-05, -1.301e-03, -1.937e-04],
          [ 2.762e-04, 6.740e-05, 1.001e-02, -2.877e-04, -4.283e-05],
          [ -5.333e-03, -1.301e-03, -2.877e-04, 1.556e-02, 8.271e-04],
          [ -7.938e-04, -1.937e-04, -4.283e-05, 8.271e-04, 1.012e-02]];
mu = [ 1.02 , 1.028, 1.01 , 1.034, 1.017];
kappa_1 = [ 0.002, 0.002, 0.002, 0.002, 0.002];
kappa_2 = [ 0.004, 0.004, 0.004, 0.004, 0.004];

% Solve target weights problem
cvx_begin quiet
    variable w_star(n)
    maximize(mu*w_star - (gamma/2.)*w_star'*Sigma*w_star)
    subject to
        sum(w_star) == 1
cvx_end

generateReturns = @() mu + randn(1, length(mu)) * chol(Sigma);

kappas = {kappa_1, kappa_2};
figure('position', [0, 0, 900, 800]);
for i = 1:2
    kappa = kappas{i};
    ws = zeros(T,n);
    us = zeros(T,n);
    no_trade_cond = zeros(T,1);
    w_t = w_star;
    for t = 1:T
        % check if no-trade condition holds
        no_trade_cond(t) = max(gamma*abs(Sigma*(w_t - w_star))-kappa') <= 0;

        cvx_begin quiet
            variable w_tilde(n)
            maximize(mu*w_tilde - (gamma/2.)*w_tilde'*Sigma*w_tilde - ...
                      kappa*(abs(w_tilde - w_t)))
            subject to
                sum(w_tilde) == 1

```

```

cvx_end
ws(t,:) = w_tilde';
us(t,:) = (w_tilde - w_t)';
w_t = w_tilde.*generateReturns()' ;
w_t = w_t/sum(w_t);
end

neglig_trades = max(abs(us')) < threshold;
fprintf('The no-trade condition holds %f%% of the times.\n', ...
    sum(no_trade_cond)*100./T);
fprintf(['The optimal solution has neglig. ' ...
    'trades %f%% of the times.\n'], ...
    sum(neglig_trades)*100./T);
colors = ['b','r','g','c','m'];
range = 1:T;
subplot(2,1,i)
for j = 1:n
    hold on
    plot(range, ws(:,j), [':' colors(j)]);
    plot(range, ones(T)*w_star(j), [':-' colors(j)]);
    non_zero_trades = abs(us(:,j)) > threshold;
    plot(range(non_zero_trades), ws(non_zero_trades,j), ...
        ['o' colors(j)], 'MarkerFaceColor', colors(j));
    xlabel('Period t');
    ylabel('Post-trade weights w_t tilde');
    title(['Optimal weights and trades with transaction cost kappa_',...
        num2str(i)]);
end
print -depsc portfolio_weights

```

The following Julia code solves the problem:

```

# data and starter code for multiperiod portfolio rebalancing problem
T = 100;
n = 5;
gamma = 8.0;
threshold = 0.001;
Sigma = [[ 1.512e-02   1.249e-03   2.762e-04  -5.333e-03  -7.938e-04]
          [ 1.249e-03   1.030e-02   6.740e-05  -1.301e-03  -1.937e-04]
          [ 2.762e-04   6.740e-05   1.001e-02  -2.877e-04  -4.283e-05]
          [ -5.333e-03  -1.301e-03  -2.877e-04   1.556e-02   8.271e-04]
          [ -7.938e-04  -1.937e-04  -4.283e-05   8.271e-04   1.012e-02]];
mu = [ 1.02 , 1.028, 1.01 , 1.034, 1.017];
kappa_1 = [ 0.002, 0.002, 0.002, 0.002, 0.002];
kappa_2 = [ 0.004, 0.004, 0.004, 0.004, 0.004];

```

```

using Distributions, Convex, SCS, PyPlot
solver = SCSSolver(verbose=0)

# compute target weights
w_star = Variable(n);
problem = maximize(mu'*w_star - (gamma/2.)*quad_form(w_star,Sigma),
    sum(w_star) == 1);
solve!(problem, solver);
w_star = w_star.value;

generateReturns() = rand(MvNormal(mu, Sigma));

kappas = {kappa_1, kappa_2};
figure(figsize=(13,10));
for i = 1:2
    kappa = kappas[i];
    us = zeros(T,n);
    ws = zeros(T,n);
    no_trade_cond = zeros(T);
    w_t = w_star;
    for t = 1:T
        # check if no-trade condition holds
        no_trade_cond[t] = maximum(gamma*abs(Sigma*(w_t - w_star))
            - kappa) <= 0;
        w_tilde = Variable(n);
        problem = maximize(mu'*w_tilde - (gamma/2.)*quad_form(w_tilde,Sigma) -
            kappa*(abs(w_tilde - w_t)), sum(w_tilde) == 1);
        solve!(problem, solver);
        w_tilde = w_tilde.value;
        ws[t,:] = w_tilde;
        us[t,:] = (w_tilde - w_t)';
        w_t = w_tilde.*generateReturns();
        w_t = w_t/sum(w_t);
    end

    neglig_trades = maximum(abs(us),2) .< threshold;
    @printf("The no-trade condition holds %.1f%% of the times.\n",
        sum(no_trade_cond)*100/T);
    @printf("The optimal solution has neglig. trades %.1f%% of the times.\n",
        sum(neglig_trades)*100./T);

    colors = ["b","r","g","c","m"];
    subplot(210+i);
    for j = 1:n
        plot(1:T, ws[:,j], colors[j]);
    end
end

```

```

plot(1:T, w_star[j]*ones(T), colors[j]*"--");
non_zero_trades = abs(us[:,j]) .> threshold;
plot((1:T)[non_zero_trades], ws[non_zero_trades,j], colors[j]*"o");
end
ylabel("post-trade weights");
xlabel("period \$t\$");
title(@sprintf "Opt. weights and trades with trans. cost \$\kappa_i\$");
end
savefig(@sprintf "portfolio_weights.eps")

```

13.22 Portfolio optimization using multiple risk models. Let $w \in \mathbf{R}^n$ be a vector of portfolio weights, where negative values correspond to short positions, and the weights are normalized such that $\mathbf{1}^T w = 1$. The expected return of the portfolio is $\mu^T w$, where $\mu \in \mathbf{R}^n$ is the (known) vector of expected asset returns. As usual we measure the risk of the portfolio using the variance of the portfolio return. However, in this problem we do not know the covariance matrix Σ of the asset returns; instead we assume that Σ is one of M (known) covariance matrices $\Sigma^{(k)} \in \mathbf{S}_{++}^n$, $k = 1, \dots, M$. We can think of the $\Sigma^{(k)}$ as representing M different risk models, associated with M different market regimes (say). For a weight vector w , there are M different possible values of the risk: $w^T \Sigma^{(k)} w$, $k = 1, \dots, M$. The worst-case risk, across the different models, is given by $\max_{k=1, \dots, M} w^T \Sigma^{(k)} w$. (This is the same as the worst-case risk over all covariance matrices in the convex hull of $\Sigma^{(1)}, \dots, \Sigma^{(M)}$.)

We will choose the portfolio weights in order to maximize the expected return, adjusted by the worst-case risk, *i.e.*, as the solution w^* of the problem

$$\begin{aligned} &\text{maximize} && \mu^T w - \gamma \max_{k=1, \dots, M} w^T \Sigma^{(k)} w \\ &\text{subject to} && \mathbf{1}^T w = 1, \end{aligned}$$

with variable w , where $\gamma > 0$ is a given risk-aversion parameter. We call this the mean-worst-case-risk portfolio problem.

- (a) Show that there exist $\gamma_1, \dots, \gamma_M \geq 0$ such that $\sum_{k=1}^M \gamma_k = \gamma$ and the solution w^* of the mean-worst-case-risk portfolio problem is also the solution of the problem

$$\begin{aligned} &\text{maximize} && \mu^T w - \sum_{k=1}^M \gamma_k w^T \Sigma^{(k)} w \\ &\text{subject to} && \mathbf{1}^T w = 1, \end{aligned}$$

with variable w .

Remark. The result above has a beautiful interpretation: We can think of the γ_k as allocating our total risk aversion γ in the mean-worst-case-risk portfolio problem across the M different regimes.

Hint. The values γ_k are not easy to find: you have to solve the mean-worst-case-risk problem to get them. Thus, this result does not help us solve the mean-worst-case-risk problem; it simply gives a nice interpretation of its solution.

- (b) Find the optimal portfolio weights for the problem instance with data given in `multi_risk_portfolio_data.*`. Report the weights and the values of γ_k , $k = 1, \dots, M$. Give the M possible values of the risk associated with your weights, and the worst-case risk.

Solution.

- (a) We can reformulate the mean-worst-case-risk portfolio problem as

$$\begin{aligned} & \text{minimize} && -\mu^T w + \gamma t \\ & \text{subject to} && w^T \Sigma^{(k)} w \leq t, \quad k = 1, \dots, M, \\ & && \mathbf{1}^T w = 1. \end{aligned}$$

with variables $w \in \mathbf{R}^n$ and $t \in \mathbf{R}$. Let λ_k be a dual variable for the k th inequality constraint, and ν be a dual variable for the equality constraint. The KKT conditions are then

$$\begin{aligned} -\mu + \nu \mathbf{1} + \sum_k 2\lambda_k \Sigma^{(k)} w &= 0 \\ \gamma - \sum_k \lambda_k &= 0 \\ \mathbf{1}^T w &= 1 \\ w^T \Sigma^{(k)} w &\leq t \\ \lambda_k (w^T \Sigma^{(k)} w - t) &= 0, \quad k = 1, \dots, M \\ \gamma &\geq 0. \end{aligned}$$

Similarly, the KKT conditions for the problem

$$\begin{aligned} & \text{maximize} && \mu^T w - \sum_{k=1}^M \gamma_k w^T \Sigma^{(k)} w \\ & \text{subject to} && \mathbf{1}^T w = 1 \end{aligned} \tag{58}$$

are

$$\begin{aligned} -\mu + \alpha \mathbf{1} + \sum_k 2\gamma_k \Sigma^{(k)} w &= 0 \\ \mathbf{1}^T w &= 1, \end{aligned} \tag{59}$$

where α is a dual variable for the equality constraint. Let $(w^*, t^*, \nu^*, \lambda^*)$ be optimal for the mean-worst-case-risk portfolio problem, and choose $\gamma_k = \lambda_k^*$, $k = 1, \dots, M$. With this choice of the γ_k , we have that $\sum_k \gamma_k = \sum_k \lambda_k^* = \gamma$ from the optimality conditions for the mean-worst-case-risk portfolio problem. Moreover, $(w, \alpha) = (w^*, \nu^*)$ satisfy (59), so w^* is optimal for (58).

- (b) The results are

```
weights:
0.42474
0.66427
-0.11469
1.38055
1.42423
-1.52706
-0.61401
-0.49879
-0.25407
0.11484
```

```
gamma_k values:
```

```
0.29232  
0.00000  
0.00000  
0.46580  
0.14230  
0.09958
```

```
risk values:
```

```
0.12188  
0.08454  
0.08247  
0.12188  
0.12188  
0.12188
```

```
worst case risk:
```

```
0.12188
```

The following Matlab code solves the problem.

```
clear all; clc  
multi_risk_portfolio_data;  
  
cvx_begin quiet  
    variables t w(n)  
    dual variables gamma_dual{M}  
    maximize(mu*w - gamma*t)  
    subject to  
        gamma_dual{1}: w'*Sigma_1*w <= t  
        gamma_dual{2}: w'*Sigma_2*w <= t  
        gamma_dual{3}: w'*Sigma_3*w <= t  
        gamma_dual{4}: w'*Sigma_4*w <= t  
        gamma_dual{5}: w'*Sigma_5*w <= t  
        gamma_dual{6}: w'*Sigma_6*w <= t  
        sum(w) == 1  
cvx_end  
  
% weights  
w  
  
% dual variables  
disp('gamma_k values');  
disp(gamma_dual{1});  
disp(gamma_dual{2});  
disp(gamma_dual{3});
```

```

disp(gamma_dual{4});
disp(gamma_dual{5});
disp(gamma_dual{6});

% risk values
disp('risk values');
disp(w'*Sigma_1*w);
disp(w'*Sigma_2*w);
disp(w'*Sigma_3*w);
disp(w'*Sigma_4*w);
disp(w'*Sigma_5*w);
disp(w'*Sigma_6*w);

```

```

% worst case risk
t

```

The following Python code solves the problem.

```

import numpy as np
import cvxpy as cvx

from multi_risk_portfolio_data import *

w = cvx.Variable(n)
t = cvx.Variable()
risks = [cvx.quad_form(w, Sigma) for Sigma in
         (Sigma_1, Sigma_2, Sigma_3, Sigma_4, Sigma_5, Sigma_6)]
risk_constraints = [risk <= t for risk in risks]
prob = cvx.Problem(cvx.Maximize(w.T*mu - gamma * t),
                    risk_constraints + [cvx.sum_entries(w) == 1])
prob.solve()

print('\nweights:')
print('\n'.join(['{:5f}'.format(weight) for weight in w.value.A1]))

print('\ngamma_k values:')
print('\n'.join(['{:5f}'.format(risk.dual_value)
                for risk in risk_constraints]))

print('\nrisk values:')
print('\n'.join(['{:5f}'.format(risk.value) for risk in risks]))

print('\nworst case risk:\n{:5f}'.format(t.value))

```

The following Julia code solves the problem.

```

using Convex, SCS
set_default_solver(SCSSolver(verbose=false))

```

```

include("multi_risk_portfolio_data.jl");

w = Variable(n)
t = Variable()

risk_1 = quadform(w, Sigma_1)
risk_2 = quadform(w, Sigma_2)
risk_3 = quadform(w, Sigma_3)
risk_4 = quadform(w, Sigma_4)
risk_5 = quadform(w, Sigma_5)
risk_6 = quadform(w, Sigma_6)

p = maximize(mu*w - gamma * t, [sum(w) == 1])

p.constraints += risk_1 <= t
p.constraints += risk_2 <= t
p.constraints += risk_3 <= t
p.constraints += risk_4 <= t
p.constraints += risk_5 <= t
p.constraints += risk_6 <= t;

solve!(p)

println("\nweights:")
println(round(w.value, 5))

println("\ngamma_k values:")
for i = 2:7
    println(round(p.constraints[i].dual,5))
end

println("\nrisk values:")
for sigma in (Sigma_1, Sigma_2, Sigma_3, Sigma_4, Sigma_5, Sigma_6)
    println(round(w.value'*sigma*w.value,5))
end

println("\nworst case risk:")
println(round(t.value, 5))

```

- 13.23 Computing market-clearing prices.** We consider n commodities or goods, with $p \in \mathbf{R}_{++}^n$ the vector of prices (per unit quantity) of them. The (nonnegative) demand for the products is a function of the prices, which we denote $D : \mathbf{R}^n \rightarrow \mathbf{R}^n$, so $D(p)$ is the demand when the product prices are p . The (nonnegative) supply of the products (*i.e.*, the amounts that manufacturers are willing to produce) is also a function of the prices, which we denote $S : \mathbf{R}^n \rightarrow \mathbf{R}^n$, so $S(p)$ is the supply when the product prices are p . We say that the market *clears* if $S(p) = D(p)$, *i.e.*, supply equals

demand, and we refer to p in this case as a set of *market-clearing prices*.

Elementary economics courses consider the special case $n = 1$, i.e., a single commodity, so supply and demand can be plotted (vertically) against the price (on the horizontal axis). It is assumed that demand decreases with increasing price, and supply increases; the market clearing price can be found ‘graphically’, as the point where the supply and demand curves intersect. In this problem we examine some cases in which market-clearing prices (for the general case $n > 1$) can be computed using convex optimization.

We assume that the demand function is *Hicksian*, which means it has the form $D(p) = \nabla E(p)$, where $E : \mathbf{R}^n \rightarrow \mathbf{R}$ is a differentiable function that is concave and increasing in each argument, called the *expenditure function*. (While not relevant in this problem, Hicksian demand arises from a model in which consumers make purchases by maximizing a concave utility function.)

We will assume that the producers are independent, so $S(p)_i = S_i(p_i)$, $i = 1, \dots, n$, where $S_i : \mathbf{R} \rightarrow \mathbf{R}$ is the supply function for good i . We will assume that the supply functions are positive and increasing on their domain \mathbf{R}_+ .

- (a) Explain how to use convex optimization to find market-clearing prices under the assumptions given above. (You do not need to worry about technical details like zero prices, or cases in which there are no market-clearing prices.)
- (b) Compute market-clearing prices for the specific case with $n = 4$,

$$E(p) = \left(\prod_{i=1}^4 p_i \right)^{1/4},$$

$$S(p) = (0.2p_1 + 0.5, 0.02p_2 + 0.1, 0.04p_3, 0.1p_4 + 0.2).$$

Give the market-clearing prices and the demand and supply (which should match) at those prices.

Hint: In CVX and CVXPY, `geo_mean` gives the geometric mean of the entries of a vector argument. Julia does not yet have a vector argument `geom_mean` function, but you can get the geometric mean of 4 variables a, b, c, d using `geomean(geomean(a, b), geomean(c, d))`.

Solution.

- (a) Consider the function $L(p) = \sum_{i=1}^n \int_0^{p_i} S_i(t)dt$. We claim that $L(p)$ is a convex function of p . Indeed, each separate term $\int_0^{p_i} S_i(t)dt$ is convex in p because it is convex in p_i (its derivative is $S_i(p_i)$ which is increasing by assumption). Thus, $L(p)$ is convex in p being a sum of convex functions. Moreover, note that $S(p) = \nabla L(p)$. We also know that $E(p)$ is concave in p and $D(p) = \nabla E(p)$ by assumption.

We now claim that the market-clearing prices p are exactly the optimal point of the following convex optimization problem:

$$\text{minimize } L(p) - E(p),$$

with variable p . This is a convex problem because L is convex and E is concave. The optimal point p^* of this problem has to satisfy the condition

$$\nabla(L(p) - E(p)) \Big|_{p=p^*} = 0.$$

Since $S(p) = \nabla L(p)$ and $D(p) = \nabla E(p)$, we conclude that p^* satisfies $S(p^*) = D(p^*)$ which means that p^* is the market-clearing price vector.

(b) For this problem instance we have

$$L(p) = \sum_{i=1}^4 \int_0^{p_i} S_i(t) dt = 0.1p_1^2 + 0.5p_1 + 0.01p_2^2 + 0.1p_2 + 0.02p_3^2 + 0.05p_4^2 + 0.2p_4,$$

so all we need to do is solve the problem

$$\text{minimize } L(p) - E(p),$$

with L as above and E as given, *i.e.*, the geometric mean.

Solving this problem we find the following market-clearing prices:

$$p = (0.6363, 2.6193, 3.1589, 1.2341).$$

The supply and demand at these prices are:

$$S(p) = D(p) = (0.6272, 0.1523, 0.1263, 0.3234).$$

The following Matlab code solves the problem.

```
cvx_begin quiet
    variable p(4)
    minimize(0.1 * square(p(1)) + 0.5 * p(1) ...
        + 0.01 * square(p(2)) + 0.1 * p(2) ...
        + 0.02 * square(p(3)) ...
        + 0.05 * square(p(4)) + 0.2 * p(4) ...
        - geo_mean(p))

cvx_end
fprintf('The market-clearing prices are: %f, %f, %f, %f\n', p)
fprintf('The supply at these prices: %f, %f, %f, %f\n', ...
    [0.2 * p(1) + 0.5, 0.02 * p(2) + 0.1, 0.04 * p(3), 0.1 * p(4) + 0.2])
g = geo_mean(p);
fprintf('The demand at these prices: %f, %f, %f, %f\n', ...
    [g / (4.0 * p(1)), g / (4.0 * p(2)), g / (4.0 * p(3)), g / (4.0 * p(4))])
```

The following Python code solves the problem.

```
import cvxpy as cvx
import numpy as np

p = cvx.Variable(4)
obj = 0.1 * cvx.square(p[0]) + 0.5 * p[0] \
    + 0.01 * cvx.square(p[1]) + 0.1 * p[1] \
    + 0.02 * cvx.square(p[2]) \
    + 0.05 * cvx.square(p[3]) + 0.2 * p[3] \
    - cvx.geo_mean(p)
```

```

problem = cvx.Problem(cvx.Minimize(obj))
problem.solve()

prices = [p.value.A1[i] for i in range(4)]
print('The market-clearing prices are: {}'.format(prices))
supply = [0.2 * prices[0] + 0.5, 0.02 * prices[1] + 0.1, \
          0.04 * prices[2], 0.1 * prices[3] + 0.2]
print('The supply at these prices: {}'.format(supply))
g = np.power(np.prod(prices), 1.0 / 4)
demand = [g / (4 * prices[i]) for i in range(4)]
print('The demand at these prices: {}'.format(demand))

```

The following Julia code solves the problem.

```

using Convex, SCS
set_default_solver(SCSSolver(verbose=false))

p = Variable(4)
obj = 0.1 * square(p[1]) + 0.5 * p[1] +
      0.01 * square(p[2]) + 0.1 * p[2] +
      0.02 * square(p[3]) +
      0.05 * square(p[4]) + 0.2 * p[4] -
      geomean(geomean(p[1], p[2]), geomean(p[3], p[4]))

problem = minimize(obj)
solve!(problem)
prices = vec(p.value)
println("Market-clearing prices are: $prices")
supply = [0.2 * prices[1] + 0.5, 0.02 * prices[2] + 0.1,
          0.04 * prices[3], 0.1 * prices[4] + 0.2]
println("Supply at these prices: $supply")
g = (prices[1] * prices[2] * prices[3] * prices[4])^(1.0 / 4)
demand = [g / (4 * prices[1]), g / (4 * prices[2]),
          g / (4 * prices[3]), g / (4 * prices[4])]
println("Demand at these prices: $demand")

```

14 Mechanical and aerospace engineering

- 14.1 Optimal design of a tensile structure.** A tensile structure is modeled as a set of n masses in \mathbf{R}^2 , some of which are fixed, connected by a set of N springs. The masses are in equilibrium, with spring forces, connection forces for the fixed masses, and gravity balanced. (This equilibrium occurs when the position of the masses minimizes the total energy, defined below.)

We let $(x_i, y_i) \in \mathbf{R}^2$ denote the position of mass i , and $m_i > 0$ its mass value. The first p masses are fixed, which means that $x_i = x_i^{\text{fixed}}$ and $y_i = y_i^{\text{fixed}}$, for $i = 1, \dots, p$. The gravitational potential energy of mass i is $gm_i y_i$, where $g \approx 9.8$ is the gravitational acceleration.

Suppose spring j connects masses r and s . Its elastic potential energy is

$$(1/2)k_j ((x_r - x_s)^2 + (y_r - y_s)^2),$$

where $k_j \geq 0$ is the stiffness of spring j .

To describe the topology, *i.e.*, which springs connect which masses, we will use the incidence matrix $A \in \mathbf{R}^{n \times N}$, defined as

$$A_{ij} = \begin{cases} 1 & \text{head of spring } j \text{ connects to mass } i \\ -1 & \text{tail of spring } j \text{ connects to mass } i \\ 0 & \text{otherwise.} \end{cases}$$

Here we arbitrarily choose a head and tail for each spring, but in fact the springs are completely symmetric, and the choice can be reversed without any effect. (Hopefully you will discover why it is convenient to use the incidence matrix A to specify the topology of the system.)

The total energy is the sum of the gravitational energies, over all the masses, plus the sum of the elastic energies, over all springs. The equilibrium positions of the masses is the point that minimizes the total energy, subject to the constraints that the first p positions are fixed. (In the equilibrium positions, the total force on each mass is zero.) We let E_{\min} denote the total energy of the system, in its equilibrium position. (We assume the energy is bounded below; this occurs if and only if each mass is connected, through some set of springs with positive stiffness, to a fixed mass.)

The total energy E_{\min} is a measure of the stiffness of the structure, with larger E_{\min} corresponding to stiffer. (We can think of $E_{\min} = -\infty$ as an infinitely unstiff structure; in this case, at least one mass is not even supported against gravity.)

- (a) Suppose we know the fixed positions $x_1^{\text{fixed}}, \dots, x_p^{\text{fixed}}, y_1^{\text{fixed}}, \dots, y_p^{\text{fixed}}$, the mass values m_1, \dots, m_n , the spring topology A , and the constant g . You are to choose nonnegative k_1, \dots, k_N , subject to a budget constraint $\mathbf{1}^T k = k_1 + \dots + k_N = k^{\text{tot}}$, where k^{tot} is given. Your goal is to maximize E_{\min} .

Explain how to do this using convex optimization.

- (b) Carry out your method for the problem data given in `tens_struct_data.m`. This file defines all the needed data, and also plots the equilibrium configuration when the stiffness is evenly distributed across the springs (*i.e.*, $k = (k^{\text{tot}}/N)\mathbf{1}$).

Report the optimal value of E_{\min} . Plot the optimized equilibrium configuration, and compare it to the equilibrium configuration with evenly distributed stiffness. (The code for doing this is in the file `tens_struct_data.m`, but commented out.)

Solution.

- (a) $A^T x$ gives a vector of the x -displacements of the springs, and $A^T y$ gives a vector of the y -displacements of the springs.

The energy as a function of x , y , and k is given by

$$E(x, y, k) = (1/2)x^T A \text{diag}(k) A^T x + (1/2)y^T A \text{diag}(k) A^T y + c^T y,$$

where $c_i = gm_i$. This is an affine function of k . Thus the minimum energy is the minimum of this function, over all x and y that satisfy the fixed constraints. It follows immediately that E_{\min} is a concave function of k . That's good, since we want to maximize it.

We'll need an explicit formula for E_{\min} . To do this we partition A into A_1 and A_2 , where $A_1 \in \mathbf{R}^{p \times N}$ is made up of the first p rows of A , while $A_2 \in \mathbf{R}^{(n-p) \times N}$ is made up of the last $n - p$ rows of A .

Let $\bar{x}, \bar{y}, \bar{c} \in \mathbf{R}^{n-p}$ denote the last $n - p$ (*i.e.*, free) elements of x , y and c respectively. The minimum energy can be written as

$$E_{\min}(k) = \min_{\bar{x}, \bar{y}} \left((1/2)z_x^T \text{diag}(k) z_x + (1/2)z_y^T \text{diag}(k) z_y + \bar{c}^T \bar{y} + C \right),$$

where $C = \sum_{i=1}^p gm_i y_i^{\text{fixed}}$ and

$$\begin{aligned} z_x &= A_2^T \bar{x} + b_x \\ z_y &= A_2^T \bar{y} + b_y \\ b_x &= A_1^T x^{\text{fixed}} \\ b_y &= A_1^T y^{\text{fixed}}. \end{aligned}$$

Thus to evaluate E_{\min} we need to evaluate the minimum of an unconstrained quadratic in \bar{x} and \bar{y} . This gives us

$$E_{\min}(k) = (1/2)(b_x^T D b_x - v_x^T Q^{-1} v_x) + (1/2)(b_y^T D b_y - v_y^T Q^{-1} v_y) + C,$$

where

$$\begin{aligned} D &= \text{diag}(k) \\ Q &= A_2 D A_2^T \\ v_x &= A_2 D b_x \\ v_y &= A_2 D b_y + \bar{c}. \end{aligned}$$

Note that all these terms are affine in k .

We can therefore write down the problem of re-allocating stiffness as

$$\begin{aligned} &\text{maximize} && b_x^T D b_x - v_x^T Q^{-1} v_x + b_y^T D b_y - v_y^T Q^{-1} v_y \\ &\text{subject to} && \mathbf{1}^T k = k^{\text{tot}}, \quad k \succeq 0. \end{aligned}$$

This is a convex optimization problem. The constraints are obviously convex in k . The objective has two terms which are affine (and therefore concave) in k and two terms which

are the negatives of matrix fractionals of affine terms in k (and thus also concave). Thus the objective is concave in k .

Here is another solution to this problem. The stiffness allocation problem is

$$\begin{aligned} & \text{maximize} && E_{\min}(k) \\ & \text{subject to} && \mathbf{1}^T k = k^{\text{tot}}, \quad k \succeq 0. \end{aligned}$$

Let (x^*, y^*) be the equilibrium mass coordinates for stiffness k and let μ be the Lagrange multiplier associated with the equality constraint of k . Then the Lagrangian of the stiffness allocation problem is

$$\begin{aligned} L(k, \mu) &= (1/2)(x^{*T} A \mathbf{diag}(k) A^T x^* + y^{*T} A \mathbf{diag}(k) A^T y^*) + c^T y^* + \mu(\mathbf{1}^T k - k^{\text{tot}}) \\ &= \sum_j k_j ((1/2)(x^{*T} a_j a_j^T x^* + y^{*T} a_j a_j^T y^*) + \mu) + c^T y^* - \mu k^{\text{tot}}, \end{aligned}$$

where a_j is the j th column of A . However, this is the same as the Lagrangian of the following optimization problem

$$\begin{aligned} & \text{minimize} && c^T y - \mu k^{\text{tot}} \\ & \text{subject to} && (1/2)x^T a_j a_j^T x + (1/2)y^T a_j a_j^T y + \mu \leq 0, \quad j = 1, \dots, N \\ & && x_i = x_i^{\text{fixed}}, \quad i = 1, \dots, p \\ & && y_i = y_i^{\text{fixed}}, \quad i = 1, \dots, p. \end{aligned}$$

The value of this optimization problem is equal to the optimal E_{\min} . The optimal spring stiffness k^* is equal to the optimal Lagrange multipliers corresponding to the inequality constraints.

Here is something that doesn't work (but was proposed by a large number of people): alternating between minimizing over x and y and maximizing over k . Some people argued that this would converge to a local optimum of the problem, thus a global optimum since the problem is convex. However, this is not true.

- (b) The following code solves the stiffness re-allocation problem.

```
tens_struct_data;

c = g*m;
% Optimum stiffness allocation problem
A1 = A(1:p,:); A2 = A(p+1:n,:); cbar = c(p+1:n);
cvx_begin
    variable k(N)
    D = diag(k);
    bx = A1'*x_fixed;
    by = A1'*y_fixed;
    vx = A2*D*bx;
    vy = A2*D*by+cbar;
    maximize(bx'*D*bx-matrix_frac(vx,A2*D*A2')+...
              by'*D*by-matrix_frac(vy,A2*D*A2'))
    subject to
```

```

k >= 0; sum(k) == k_tot;
cvx_end

% Compute Emin
Eunif = 0.5*x_unif'*A*diag(k_unif)*A'*x_unif;
Eunif = Eunif + 0.5*y_unif'*A*diag(k_unif)*A'*y_unif;
Eunif = Eunif + c'*y_unif
Emin = 0.5*cvx_optval+c(1:p)'*y_fixed

% Form x and y
xmin = -(A2*D*A2')\ (A2*D*A1'*x_fixed);
ymin = -(A2*D*A2')\ (A2*D*A1'*y_fixed+cbar);

xopt = zeros(n,1);
xopt(1:p) = x_fixed;
xopt(p+1:n) = xmin;

yopt = zeros(n,1);
yopt(1:p) = y_fixed;
yopt(p+1:n) = ymin;

% Plot optimized structure and structure with uniform stiffness
figure
ind_ex = find(k_unif < 1e-2); %do not show springs with k < 1e-2
Aadj = A(:,setdiff(1:N,ind_ex));
Aadj2 = double(Aadj*Aadj'-diag(diag(Aadj*Aadj')) ~= 0);
gplot(Aadj2,[x_unif y_unif], 'o-');
hold on
plot(x_fixed,y_fixed, 'ro');
xlabel('x'); ylabel('y');
axis([0.1 0.8 -1.5 1])

figure
ind_ex = find(k < 1e-2); %do not show springs with k < 1e-2
Aadj = A(:,setdiff(1:N,ind_ex));
Aadj2 = double(Aadj*Aadj'-diag(diag(Aadj*Aadj')) ~= 0);
gplot(Aadj2,[xopt yopt], 'o-');
hold on
plot(x_fixed,y_fixed, 'ro');
xlabel('x'); ylabel('y');
axis([0.1 0.8 -1.5 1])
hold off

```

The optimal energy is $E_{\min}(k^*) = 57.84$. On the other hand the minimum energy is 18.37 when stiffness is uniformly allocated. The mass configurations for both uniform and optimal stiffness allocations are shown in figure 15. Springs with stiffness less than 10^{-2} are not shown.

14.2 Equilibrium position of a system of springs. We consider a collection of n masses in \mathbf{R}^2 , with locations $(x_1, y_1), \dots, (x_n, y_n)$, and masses m_1, \dots, m_n . (In other words, the vector $x \in \mathbf{R}^n$ gives the x-coordinates, and $y \in \mathbf{R}^n$ gives the y-coordinates, of the points.) The masses m_i are, of course, positive.

For $i = 1, \dots, n-1$, mass i is connected to mass $i+1$ by a spring. The potential energy in the i th spring is a function of the (Euclidean) distance $d_i = \|(x_i, y_i) - (x_{i+1}, y_{i+1})\|_2$ between the i th and $(i+1)$ st masses, given by

$$E_i = \begin{cases} 0 & d_i < l_i \\ (k_i/2)(d_i - l_i)^2 & d_i \geq l_i \end{cases}$$

where $l_i \geq 0$ is the rest length, and $k_i > 0$ is the stiffness, of the i th spring. The gravitational potential energy of the i th mass is $g m_i y_i$, where g is a positive constant. The total potential energy of the system is therefore

$$E = \sum_{i=1}^{n-1} E_i + g m^T y.$$

The locations of the first and last mass are fixed. The equilibrium location of the other masses is the one that minimizes E .

- (a) Show how to find the equilibrium positions of the masses $2, \dots, n-1$ using convex optimization. Be sure to justify convexity of any functions that arise in your formulation (if it is not obvious). The problem data are m_i , k_i , l_i , g , x_1 , y_1 , x_n , and y_n .
- (b) Carry out your method to find the equilibrium positions for a problem with $n = 10$, $m_i = 1$, $k_i = 10$, $l_i = 1$, $x_1 = y_1 = 0$, $x_n = y_n = 10$, with g varying from $g = 0$ (no gravity) to $g = 10$ (say). Verify that the results look reasonable. Plot the equilibrium configuration for several values of g .

Solution. The only tricky part here is to show that E_i is a convex function of (x, y) . Define $g(u) = k_i/2(u - l_i)^2$ for $u \geq l_i$, and $g(u) = 0$ for $u < l_i$. This function is convex and nondecreasing on its extended domain, so

$$E_i = g(\|(x_i, y_i) - (x_{i+1}, y_{i+1})\|_2)$$

is a convex function of (x, y) . So E is convex, and we can minimize it subject to equality constraints on (x_1, y_1) and (x_n, y_n) . The CVX code below solves the problem for the given problem instance.

```
% equilibrium position of mass spring systems.
n=10; % number of masses
m=ones(n,1); % mass values
l=ones(n-1,1); % spring resting lengths
k=10*ones(n-1,1); % spring stiffnesses
g=3;
x1=0; y1=0;
xn=10; yn=10;
```

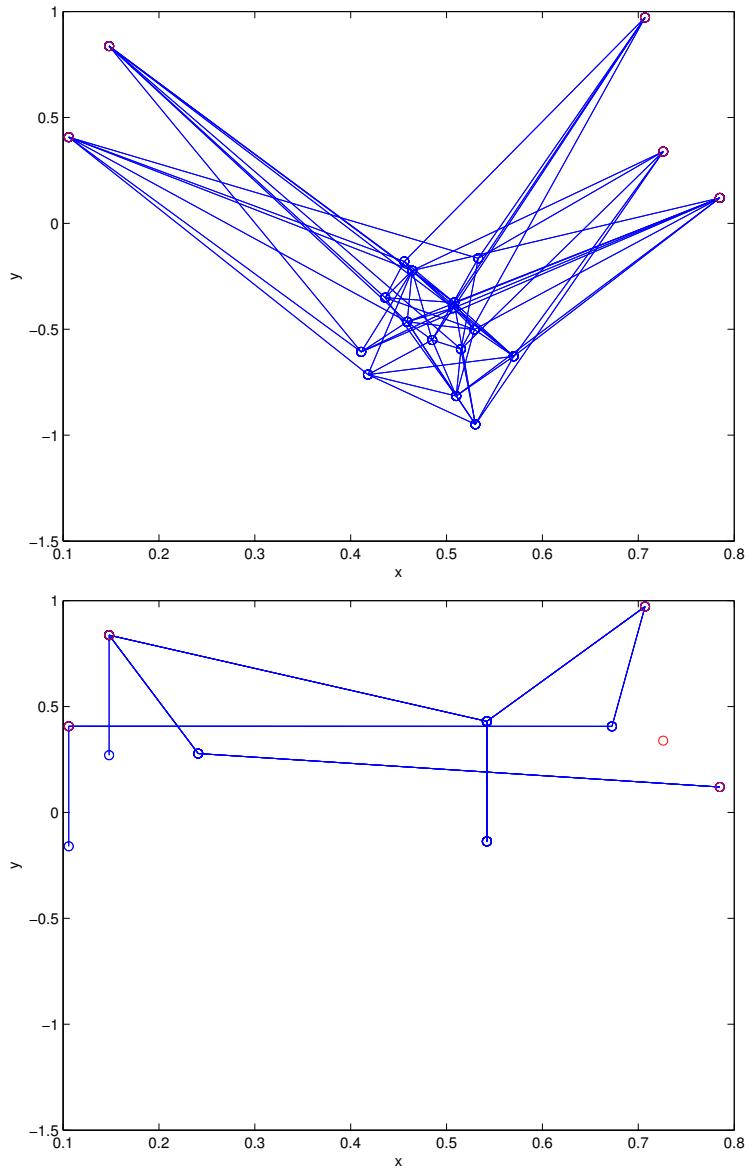


Figure 15: Mass configuration for uniform (top) and optimal (bottom) stiffness allocation.

```

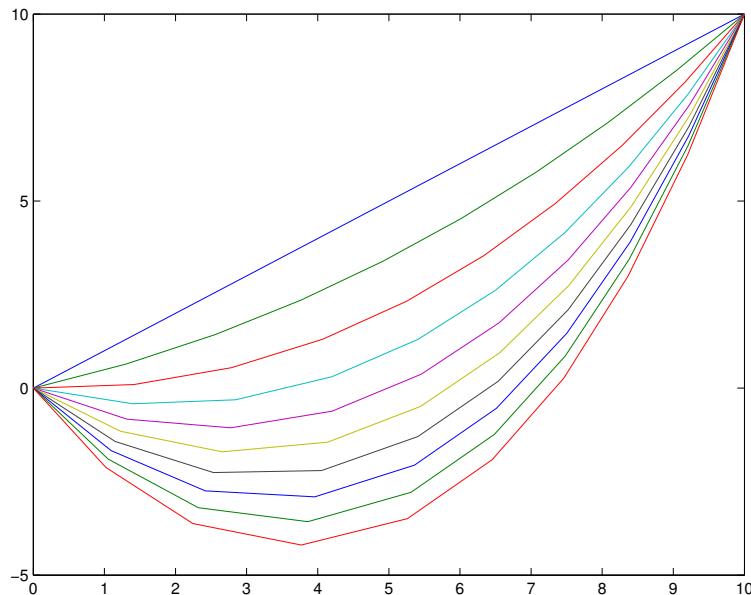
cvx_quiet('true');

K = 10; % number of values of g
g= linspace(0,10,K);
X = zeros(n,K); % hold optimal X positions for K values of g
Y = zeros(n,K); % hold optimal Y positions for K values of g
for i=1:K
cvx_begin
variables x(n) y(n)
x(1) == x1;
x(n) == xn;
y(1) == y1;
y(n) == yn;
d = norms([x(2:n)'-x(1:n-1)'; y(2:n)'-y(1:n-1)' ]); % distances between masses
minimize (sum(k'.*square_pos(d-1'))+g(i)*m'*y)
cvx_end
X(:,i)=x;
Y(:,i)=y;
end

plot(X,Y)
print -depsc mass_spr_pos

```

The plot below shows the equilibrium positions for 10 values of g between $g = 0$ and $g = 10$. We can see that with no gravity, the equilibrium positions are on a straight line between the fixed endpoints; as gravity increases, we see more sag in the positions.



14.3 Elastic truss design. In this problem we consider a truss structure with m bars connecting a set

of nodes. Various external forces are applied at each node, which cause a (small) displacement in the node positions. $f \in \mathbf{R}^n$ will denote the vector of (components of) external forces, and $d \in \mathbf{R}^n$ will denote the vector of corresponding node displacements. (By ‘corresponding’ we mean if f_i is, say, the z -coordinate of the external force applied at node k , then d_i is the z -coordinate of the displacement of node k .) The vector f is called a *loading* or *load*.

The structure is linearly elastic, *i.e.*, we have a linear relation $f = Kd$ between the vector of external forces f and the node displacements d . The matrix $K = K^T \succ 0$ is called the *stiffness matrix* of the truss. Roughly speaking, the ‘larger’ K is (*i.e.*, the stiffer the truss) the smaller the node displacement will be for a given loading.

We assume that the geometry (unloaded bar lengths and node positions) of the truss is fixed; we are to design the cross-sectional areas of the bars. These cross-sectional areas will be the design variables x_i , $i = 1, \dots, m$. The stiffness matrix K is a linear function of x :

$$K(x) = x_1 K_1 + \dots + x_m K_m,$$

where $K_i = K_i^T \succeq 0$ depend on the truss geometry. You can assume these matrices are given or known. The total weight W_{tot} of the truss also depends on the bar cross-sectional areas:

$$W_{\text{tot}}(x) = w_1 x_1 + \dots + w_m x_m,$$

where $w_i > 0$ are known, given constants (density of the material times the length of bar i). Roughly speaking, the truss becomes stiffer, but also heavier, when we increase x_i ; there is a tradeoff between stiffness and weight.

Our goal is to design the stiffest truss, subject to bounds on the bar cross-sectional areas and total truss weight:

$$l \leq x_i \leq u, \quad i = 1, \dots, m, \quad W_{\text{tot}}(x) \leq W,$$

where l , u , and W are given. You may assume that $K(x) \succ 0$ for all feasible vectors x . To obtain a specific optimization problem, we must say how we will measure the stiffness, and what model of the loads we will use.

- (a) There are several ways to form a scalar measure of how stiff a truss is, for a given load f . In this problem we will use the *elastic stored energy*

$$\mathcal{E}(x, f) = \frac{1}{2} f^T K(x)^{-1} f$$

to measure the stiffness. Maximizing stiffness corresponds to minimizing $\mathcal{E}(x, f)$.

Show that $\mathcal{E}(x, f)$ is a convex function of x on $\{x \mid K(x) \succ 0\}$.

Hint. Use Schur complements to prove that the epigraph is a convex set.

- (b) We can consider several different scenarios that reflect our knowledge about the possible loadings f that can occur. The simplest is that f is a single, fixed, known loading. In more sophisticated formulations, the loading f might be a random vector with known distribution, or known only to lie in some set \mathcal{F} , etc.

Show that each of the following four problems is a convex optimization problem, with x as variable.

- *Design for a fixed known loading.* The vector f is known and fixed. The design problem is

$$\begin{aligned} & \text{minimize} && \mathcal{E}(x, f) \\ & \text{subject to} && l \leq x_i \leq u, \quad i = 1, \dots, m \\ & && W_{\text{tot}}(x) \leq W. \end{aligned}$$

- *Design for multiple loadings.* The vector f can take any of N known values $f^{(i)}$, $i = 1, \dots, N$, and we are interested in the worst-case scenario. The design problem is

$$\begin{aligned} & \text{minimize} && \max_{i=1, \dots, N} \mathcal{E}(x, f^{(i)}) \\ & \text{subject to} && l \leq x_i \leq u, \quad i = 1, \dots, m \\ & && W_{\text{tot}}(x) \leq W. \end{aligned}$$

- *Design for worst-case, unknown but bounded load.* Here we assume the vector f can take arbitrary values in a ball $B = \{f \mid \|f\|_2 \leq \alpha\}$, for a given value of α . We are interested in minimizing the worst-case stored energy, *i.e.*,

$$\begin{aligned} & \text{minimize} && \sup_{\|f\|_2 \leq \alpha} \mathcal{E}(x, f^{(i)}) \\ & \text{subject to} && l \leq x_i \leq u, \quad i = 1, \dots, m \\ & && W_{\text{tot}}(x) \leq W. \end{aligned}$$

- *Design for a random load with known statistics.* We can also use a stochastic model of the uncertainty in the load, and model the vector f as a random variable with known mean and covariance:

$$\mathbf{E} f = f^{(0)}, \quad \mathbf{E}(f - f^{(0)})(f - f^{(0)})^T = \Sigma.$$

In this case we would be interested in minimizing the expected stored energy, *i.e.*,

$$\begin{aligned} & \text{minimize} && \mathbf{E} \mathcal{E}(x, f^{(i)}) \\ & \text{subject to} && l \leq x_i \leq u, \quad i = 1, \dots, m \\ & && W_{\text{tot}}(x) \leq W. \end{aligned}$$

Hint. If v is a random vector with zero mean and covariance Σ , then $\mathbf{E} v^T A v = \mathbf{E} \mathbf{tr} A v v^T = \mathbf{tr} A \mathbf{E} v v^T = \mathbf{tr} A \Sigma$.

- (c) Formulate the four problems in (b) as semidefinite programming problems.

Solution. There are several correct answers for each subproblem. We give only one or two.

- (a) The epigraph of $\mathcal{E}(x, f)$ is

$$\{(x, t) \mid \frac{1}{2} f^T K(x)^{-1} f \leq t, K(x) \succ 0\}.$$

Using Schur complements we can express this as the solution set of the linear matrix inequality

$$\begin{bmatrix} K(x) & f \\ f^T & 2t \end{bmatrix} \succeq 0,$$

i.e., as a convex set.

- (b) • *Fixed known load.* The objective function is convex (see part a). The constraints are linear inequalities.
- *Multiple loads.* The objective function is the pointwise maximum of N convex functions, hence convex.
- *Unknown but bounded load.* The objective function is the pointwise supremum of an (infinite) number of convex functions, and hence convex.
- *Random load.* We have

$$\begin{aligned}\mathbf{E} f^T K(x)^{-1} f &= f_0^T K(x)^{-1} f_0 + \mathbf{E}(f - f_0)^T K(x)^{-1}(f - f_0) \\ &= f_0^T K(x)^{-1} f_0 + \mathbf{tr} K(x)^{-1} \Sigma.\end{aligned}$$

We have already shown that the first term in the sum is convex. To show convexity of the second term, we write the eigenvalue decomposition of Σ as $\Sigma = \sum_{i=1}^n \lambda_i q_i q_i^T$. ($\lambda_i \geq 0$, since a covariance matrix is always positive semidefinite.) Then we can write

$$\begin{aligned}\mathbf{tr} K(x)^{-1} \Sigma &= \mathbf{tr} K(x)^{-1} \sum_i \lambda_i q_i q_i^T \\ &= \sum_{i=1}^n \lambda_i \mathbf{tr} K(x)^{-1} q_i q_i^T = \sum_{i=1}^n \lambda_i q_i^T K(x)^{-1} q_i,\end{aligned}$$

i.e., a nonnegative sum of convex functions of x .

- (c) • *Fixed known load.*

$$\begin{aligned}&\text{minimize} \quad t/2 \\ &\text{subject to} \quad \begin{bmatrix} K(x) & f \\ f^T & t \end{bmatrix} \succeq 0 \\ &\quad W_{\text{tot}}(x) \leq W \\ &\quad l \preceq x \preceq u.\end{aligned}$$

- *Multiple loadings.*

$$\begin{aligned}&\text{minimize} \quad t/2 \\ &\text{subject to} \quad \begin{bmatrix} K(x) & f_i \\ f_i^T & t \end{bmatrix} \succeq 0, \quad i = 1, \dots, N \\ &\quad W_{\text{tot}}(x) \leq W \\ &\quad l \preceq x \preceq u.\end{aligned}$$

- *Unknown but bounded load.* Note that

$$\begin{aligned}\sup_{\|f\|_2 \leq \alpha} \left(\frac{1}{2} f^T K(x)^{-1} f \right) &= \frac{\alpha^2}{2} \sup_{\|f\|_2 \leq 1} f^T K(x)^{-1} f \\ &= \frac{\alpha^2}{2} \lambda_{\max}(K(x)^{-1}) \\ &= \frac{\alpha^2}{2} (\lambda_{\min}(K(x)))^{-1},\end{aligned}$$

where $\lambda_{\max}(K(x)^{-1})$ is the largest eigenvalue of $K(x)^{-1}$ and $\lambda_{\min}(K(x))$ is the smallest eigenvalue of $K(x)$.

This observation leads to the SDP formulation:

$$\begin{aligned} & \text{maximize} && t \\ & \text{subject to} && K(x) \succeq tI \\ & && W_{\text{tot}}(x) \leq W \\ & && l \preceq x \preceq u. \end{aligned}$$

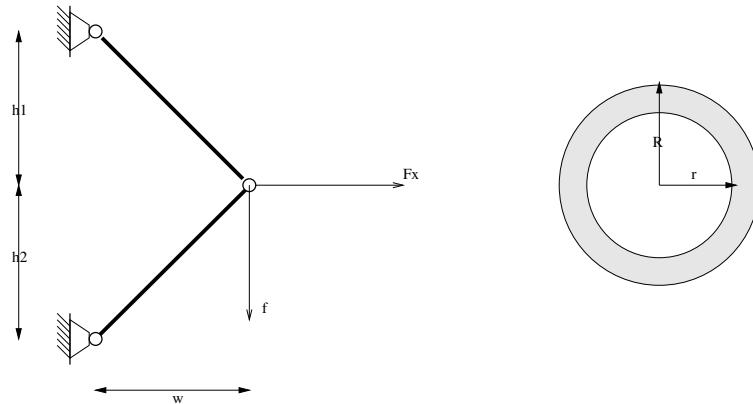
(See page 7-9 of the lecture notes). The variables are x, t . From the optimal t we obtain the optimal value of the original problem as $\alpha^2/(2t)$.

- *Random load.* There are several ways to express the problem as an SDP. One possibility is to follow our proof of convexity of the objective function, and to write the problem as

$$\begin{aligned} & \text{minimize} && \frac{1}{2}(t_0 + \sum_{i=1}^n \lambda_i t_i) \\ & \text{subject to} && \begin{bmatrix} K(x) & f_0 \\ f_0^T & t_0 \end{bmatrix} \succeq 0 \\ & && \begin{bmatrix} K(x) & q_i \\ q_i^T & t_i \end{bmatrix} \succeq 0, \quad i = 1, \dots, n \\ & && W_{\text{tot}}(x) \leq W \\ & && l \preceq x \preceq u. \end{aligned}$$

The variables are x and t_0, t_1, \dots, t_n .

- 14.4 A structural optimization problem.** [?] The figure shows a two-bar truss with height $2h$ and width w . The two bars are cylindrical tubes with inner radius r and outer radius R . We are interested in determining the values of r, R, w , and h that minimize the weight of the truss subject to a number of constraints. The structure should be strong enough for two loading scenarios. In the first scenario a vertical force F_1 is applied to the node; in the second scenario the force is horizontal with magnitude F_2 .



The weight of the truss is proportional to the total volume of the bars, which is given by

$$2\pi(R^2 - r^2)\sqrt{w^2 + h^2}$$

This is the cost function in the design problem.

The first constraint is that the truss should be strong enough to carry the load F_1 , i.e., the stress caused by the external force F_1 must not exceed a given maximum value. To formulate this constraint, we first determine the forces in each bar when the structure is subjected to the vertical load F_1 . From the force equilibrium and the geometry of the problem we can determine that the magnitudes of the forces in two bars are equal and given by

$$\frac{\sqrt{w^2 + h^2}}{2h} F_1.$$

The maximum force in each bar is equal to the cross-sectional area times the maximum allowable stress σ (which is a given constant). This gives us the first constraint:

$$\frac{\sqrt{w^2 + h^2}}{2h} F_1 \leq \sigma \pi (R^2 - r^2).$$

The second constraint is that the truss should be strong enough to carry the load F_2 . When F_2 is applied, the magnitudes of the forces in two bars are again equal and given by

$$\frac{\sqrt{w^2 + h^2}}{2w} F_2,$$

which gives us the second constraint:

$$\frac{\sqrt{w^2 + h^2}}{2w} F_2 \leq \sigma \pi (R^2 - r^2).$$

We also impose limits $w_{\min} \leq w \leq w_{\max}$ and $h_{\min} \leq h \leq h_{\max}$ on the width and the height of the structure, and limits $1.1r \leq R \leq R_{\max}$ on the outer radius.

In summary, we obtain the following problem:

$$\begin{aligned} & \text{minimize} && 2\pi(R^2 - r^2)\sqrt{w^2 + h^2} \\ & \text{subject to} && \frac{\sqrt{w^2 + h^2}}{2h} F_1 \leq \sigma \pi (R^2 - r^2) \\ & && \frac{\sqrt{w^2 + h^2}}{2w} F_2 \leq \sigma \pi (R^2 - r^2) \\ & && w_{\min} \leq w \leq w_{\max} \\ & && h_{\min} \leq h \leq h_{\max} \\ & && 1.1r \leq R \leq R_{\max} \\ & && R > 0, \quad r > 0, \quad w > 0, \quad h > 0. \end{aligned}$$

The variables are R , r , w , h .

Formulate this as a geometric programming problem.

Solution.

We can introduce new variables

$$u = R^2 - r^2, \quad L = \sqrt{w^2 + h^2}$$

and write the problem as

$$\begin{aligned} & \text{minimize} && 2\pi uL \\ & \text{subject to} && (F_1/(2\sigma\pi))Lh^{-1}u^{-1} \leq 1 \\ & && (F_2/(2\sigma\pi))Lw^{-1}u^{-1} \leq 1 \\ & && (1/w_{\max})w \leq 1, \quad w_{\min}w^{-1} \leq 1 \\ & && (1/h_{\max})h \leq 1, \quad h_{\min}h^{-1} \leq 1 \\ & && 0.21r^2u^{-1} \leq 1 \\ & && (1/R_{\max}^2)u + (1/R_{\max}^2)r^2 \leq 1 \\ & && w^2L^{-2} + h^2L^{-2} \leq 1 \\ & && r > 0, \quad w > 0, \quad h > 0, \quad u > 0, \quad L > 0, \end{aligned}$$

with scalar variables r , w , h , u , and L . The desired values can be recovered from the GP by calculating $R = \sqrt{u + r^2}$.

A geometric programming problem can only have *monomial* equality constraints, so we cannot add an equality constraint $L^2 = w^2 + h^2$. Therefore we changed it to an inequality

$$L^2 \geq w^2 + h^2,$$

i.e., $w^2L^{-2} + h^2L^{-2} \leq 1$. To see why this works, notice that L appears only in the objective and the first two inequality constraints. Each of these involves an expression that is monotonically increasing in L , so at the optimum L will be equal to $\sqrt{w^2 + h^2}$. If this is not the case, then we could make L smaller, which maintains feasibility and strictly decreases the objective.

Also, note that we replaced the inequality $R \geq 1.1r$ with $u \geq (1.1^2 - 1)r^2 = 0.21r^2$.

14.5 Optimizing the inertia matrix of a 2D mass distribution. An object has density $\rho(z)$ at the point $z = (x, y) \in \mathbf{R}^2$, over some region $\mathcal{R} \subset \mathbf{R}^2$. Its mass $m \in \mathbf{R}$ and center of gravity $c \in \mathbf{R}^2$ are given by

$$m = \int_{\mathcal{R}} \rho(z) dx dy, \quad c = \frac{1}{m} \int_{\mathcal{R}} \rho(z) z dx dy,$$

and its inertia matrix $M \in \mathbf{R}^{2 \times 2}$ is

$$M = \int_{\mathcal{R}} \rho(z)(z - c)(z - c)^T dx dy.$$

(You do not need to know the mechanics interpretation of M to solve this problem, but here it is, for those interested. Suppose we rotate the mass distribution around a line passing through the center of gravity in the direction $q \in \mathbf{R}^2$ that lies in the plane where the mass distribution is, at angular rate ω . Then the total kinetic energy is $(\omega^2/2)q^T M q$.)

The goal is to choose the density ρ , subject to $0 \leq \rho(z) \leq \rho^{\max}$ for all $z \in \mathcal{R}$, and a fixed total mass $m = m^{\text{given}}$, in order to maximize $\lambda_{\min}(M)$.

To solve this problem numerically, we will discretize \mathcal{R} into N pixels each of area a , with pixel i having constant density ρ_i and location (say, of its center) $z_i \in \mathbf{R}^2$. We will assume that the integrands above don't vary too much over the pixels, and from now on use instead the expressions

$$m = a \sum_{i=1}^N \rho_i, \quad c = \frac{a}{m} \sum_{i=1}^N \rho_i z_i, \quad M = a \sum_{i=1}^N \rho_i (z_i - c)(z_i - c)^T.$$

The problem below refers to these discretized expressions.

- (a) Explain how to solve the problem using convex (or quasiconvex) optimization.
- (b) Carry out your method on the problem instance with data in `inertia_dens_data.m`. This file includes code that plots a density. Give the optimal inertia matrix and its eigenvalues, and plot the optimal density.

Solution.

- (a) We first express M as

$$M = a \sum_{i=1}^N \rho_i z_i z_i^T - m c c^T,$$

so $\lambda_{\min}(M) \geq \gamma$ if and only if

$$a \sum_{i=1}^N \rho_i z_i z_i^T - m c c^T \succeq \gamma I.$$

Using Schur complements, and the given fixed mass, this can be written as

$$\begin{bmatrix} a \sum_{i=1}^N \rho_i z_i z_i^T - \gamma I & c \\ c^T & 1/m^{\text{given}} \end{bmatrix} \succeq 0,$$

which is an LMI in ρ and γ .

We can express therefore express the problem as the SDP

$$\begin{aligned} & \text{maximize} && \gamma \\ & \text{subject to} && \begin{bmatrix} a \sum_{i=1}^N \rho_i z_i z_i^T - \gamma I & c \\ c^T & 1/m^{\text{given}} \end{bmatrix} \succeq 0 \\ & && a \mathbf{1}^T \rho = m^{\text{given}}, \quad 0 \preceq \rho \preceq \rho^{\max} \mathbf{1} \\ & && c = (a/m^{\text{given}}) \sum_{i=1}^N \rho_i z_i, \end{aligned}$$

with variables $\rho \in \mathbf{R}^N$, $c \in \mathbf{R}^2$, and $\gamma \in \mathbf{R}$.

We can also express the problem another way, without explicit LMIs. We can write the LMI as

$$c^T \left(a \sum_{i=1}^N \rho_i z_i z_i^T - \gamma I \right)^{-1} c \leq 1/m^{\text{given}},$$

where we assume the matrix inverted here is positive definite. This leads to the problem

$$\begin{aligned} & \text{maximize} && \gamma \\ & \text{subject to} && c^T \left(a \sum_{i=1}^N \rho_i z_i z_i^T - \gamma I \right)^{-1} c \leq 1/m^{\text{given}} \\ & && a \mathbf{1}^T \rho = m^{\text{given}}, \quad 0 \preceq \rho \preceq \rho^{\max} \mathbf{1} \\ & && c = (a/m^{\text{given}}) \sum_{i=1}^N \rho_i z_i. \end{aligned}$$

(b) The code below solves the problem. The optimal inertia matrix is

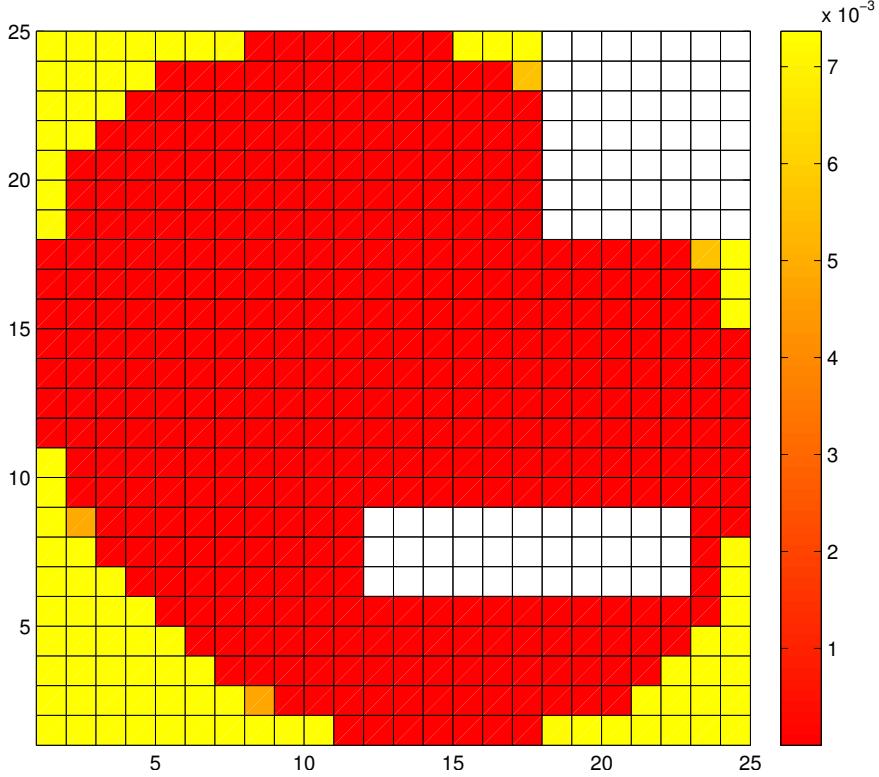
$$M = \begin{bmatrix} 0.6619 & -0.0001 \\ -0.0001 & 0.6619 \end{bmatrix},$$

which has two equal eigenvalues. The optimal mass distribution is shown below.

```
% inertia density optimization
inertia_dens_data;

cvx_begin
variables rho(N) c(2) gamma
maximize (gamma)
[a*Z*diag(rho)*Z' - gamma*eye(2) c; c' 1/mgiven] == semidefinite(3);
a*sum(rho) == mgiven;
0 <= rho;
rho <= rhomax;
c == (a/mgiven)*Z*rho;
cvx_end
M = a*Z*diag(rho)*Z'-mgiven*c*c';

%%%%% Plot Solution %%%%%%
P = nan(n,n) ; P(ind)=rho;
pcolor(P); axis square;
colormap autumn ; colorbar ;
print -depsc inertia_dens
```



14.6 Truss loading analysis. A truss (in 2D, for simplicity) consists of a set of n nodes, with positions $p^{(1)}, \dots, p^{(n)} \in \mathbf{R}^2$, connected by a set of m bars with tensions $t_1, \dots, t_m \in \mathbf{R}$ ($t_j < 0$ means bar j operates in compression).

Each bar puts a force on the two nodes which it connects. Suppose bar j connects nodes k and l . The tension in this bar applies a force

$$\frac{t_j}{\|p^{(l)} - p^{(k)}\|_2} (p^{(l)} - p^{(k)}) \in \mathbf{R}^2$$

to node k , and the opposite force to node l . In addition to the forces imparted by the bars, each node has an external force acting on it. We let $f^{(i)} \in \mathbf{R}^2$ be the external force acting on node i . For the truss to be in equilibrium, the total force on each node, *i.e.*, the sum of the external force and the forces applied by all of the bars that connect to it, must be zero. We refer to this constraint as force balance.

The tensions have given limits, $T_j^{\min} \leq t_j \leq T_j^{\max}$, with $T_j^{\min} \leq 0$ and $T_j^{\max} \geq 0$, for $j = 1, \dots, m$. (For example, if bar j is a cable, then it can only apply a nonnegative tension, so $T_j^{\min} = 0$, and we interpret T_j^{\max} as the maximum tension the cable can carry.)

The first p nodes, $i = 1, \dots, p$, are *free*, while the remaining $n - p$ nodes, $i = p + 1, \dots, n$, are *anchored* (*i.e.*, attached to a foundation). We will refer to the external forces on the free nodes as *load forces*, and external forces at the anchor nodes as *anchor forces*. The anchor forces are unconstrained. (More accurately, the foundations at these points are engineered to withstand any total force that the bars attached to it can deliver.) We will assume that the load forces are just

dead weight, *i.e.*, have the form

$$f^{(i)} = \begin{bmatrix} 0 \\ -w_i \end{bmatrix}, \quad i = 1, \dots, p,$$

where $w_i \geq 0$ is the weight supported at node i .

The set of weights $w \in \mathbf{R}_+^p$ is *supportable* if there exists a set of tensions $t \in \mathbf{R}^m$ and anchor forces $f^{(p+1)}, \dots, f^{(n)}$ that, together with the given load forces, satisfy the force balance equations and respect the tension limits. (The tensions and anchor forces in a real truss will adjust themselves to have such values when the load forces are applied.) If there does not exist such a set of tensions and anchor forces, the set of load forces is said to be *unsupportable*. (In this case, a real truss will fail, or collapse, when the load forces are applied.)

Finally, we get to the questions.

- (a) Explain how to find the maximum total weight, $\mathbf{1}^T w$, that is supportable by the truss.
- (b) Explain how to find the minimum total weight that is not supportable by the truss. (Here we mean: Find the minimum value of $\mathbf{1}^T w$, for which $(1 + \epsilon)w$ is not supportable, for all $\epsilon > 0$.)
- (c) Carry out the methods of parts (a) and (b) on the data given in `truss_load_data.m`. Give the critical total weights from parts (a) and (b), as well as the individual weight vectors.

Notes.

- In parts (a) and (b), we don't need a fully formal mathematical justification; a clear argument or explanation of anything not obvious is fine.
- The force balance equations can be expressed in the compact and convenient form

$$At + \begin{bmatrix} f^{\text{load,x}} \\ f^{\text{load,y}} \\ f^{\text{anch}} \end{bmatrix} = 0,$$

where

$$\begin{aligned} f^{\text{load,x}} &= (f_1^{(1)}, \dots, f_1^{(p)}) \in \mathbf{R}^p, \\ f^{\text{load,y}} &= (f_2^{(1)}, \dots, f_2^{(p)}) \in \mathbf{R}^p, \\ f^{\text{anch}} &= (f_1^{(p+1)}, \dots, f_1^{(n)}, f_2^{(p+1)}, \dots, f_2^{(n)}) \in \mathbf{R}^{2(n-p)}, \end{aligned}$$

and $A \in \mathbf{R}^{2n \times m}$ is a matrix that can be found from the geometry data (truss topology and node positions). You may refer to A in your solutions to parts (a) and (b). For part (c), *we have very kindly provided the matrix A for you in the m-file*, to save you the time and trouble of working out the force balance equations from the geometry of the problem.

Solution.

- (a) The first step is to decompose A into three submatrices

$$A = \begin{bmatrix} A_x \\ A_y \\ A_{\text{anch}} \end{bmatrix},$$

(with p , p , and $2(n - p)$ rows, respectively), so force balance with the dead weight loads is

$$A_x t = 0, \quad A_y t = w, \quad A_{\text{anch}} t + f^{\text{anch}} = 0.$$

Since f^{anch} is unconstrained, the last set of force balance equations will always be satisfied, no matter what value t has (indeed, just take $f^{\text{anch}} = -A_{\text{anch}}t$). So we can just ignore these equations. So the weight vector w is supportable provided there exists t which satisfies

$$A_x t = 0, \quad A_y t = w, \quad T^{\min} \preceq t \preceq T^{\max}.$$

To find the maximum supportable total weight, we can solve the LP

$$\begin{aligned} & \text{maximize} && \mathbf{1}^T w \\ & \text{subject to} && A_x t = 0, \quad A_y t = w, \quad w \succeq 0 \\ & && T^{\min} \preceq t \preceq T^{\max}, \end{aligned}$$

with variables t and w . This tells us that the set of supportable weight vectors is a polyhedron, which we'll denote as $\mathcal{S} \subseteq \mathbf{R}_+^n$. Here we are maximizing a linear function ($\mathbf{1}^T w$) over \mathcal{S} , which is an LP.

- (b) Finding the minimum unsupportable total weight is more difficult than finding the maximum supportable total weight. The set of unsupportable weight vectors is

$$\mathcal{U} = \mathbf{R}_+^n \setminus \mathcal{S},$$

which is not convex. Minimizing over \mathcal{U} the total weight, a linear function of w , is not an LP. The key insight is this: The minimum unsupportable weight will be *concentrated* at one of the free nodes (which has the nice interpretation as the weakest node in the truss). Once we know this, we solve the problem by considering each node in turn, finding the maximum supportable weight at each node, by solving the LPs

$$\begin{aligned} & \text{maximize} && \alpha \\ & \text{subject to} && A_x t = 0, \quad A_y t = \alpha e_k \\ & && T^{\min} \preceq t \preceq T^{\max}, \end{aligned}$$

with variables t and α , for $k = 1, \dots, n$. This number is the same as the minimum unsupportable weight at the node. The minimum of these numbers give us the minimum unsupportable total weight.

The tricky part is to show that a minimum of $\mathbf{1}^T w$ over $\bar{\mathcal{U}}$ occurs at a w of the form $w = w_k e_k$, where e_k is the k th unit vector. (The bar above \mathcal{U} means its closure.) Here is one argument for this, which relies on knowing some analysis. Suppose that w^* minimizes $\mathbf{1}^T w$ over $\bar{\mathcal{U}}$, with associated total weight $W^* = \mathbf{1}^T w^*$. This means that w^* is actually supportable, but just barely; $(1 + \epsilon)w^*$ is unsupportable for all $\epsilon > 0$. We claim that one of the points $W^* e_k$ also minimizes $\mathbf{1}^T w$ over $\bar{\mathcal{U}}$. (This is what we want to show.) Let $\epsilon > 0$, and consider the point $(1 + \epsilon)w^*$, which is unsupportable. But the vector $(1 + \epsilon)w^*$ is a convex combination of the vectors $(1 + \epsilon)W^* e_k$; so if all these vectors were supportable, we would have $(1 + \epsilon)w^*$ supportable, since the set of supportable weights is convex. So at least one of the vectors $(1 + \epsilon)W^* e_k$ is unsupportable. Since ϵ was arbitrary, we conclude that one of the vectors $W^* e_k$ is also on the boundary of \mathcal{U} (and \mathcal{S}), and so also minimizes the total weight over $\bar{\mathcal{U}}$.

Another proof. The set of weights is *not* supportable if the equations above have no solution. We'll derive the alternative inequalities. We form the Lagrangian

$$L = -\nu^T A_x t - \kappa^T (A_y t - w) + \lambda^T (T^{\min} - t) + \mu^T (t - T^{\max}),$$

with $\lambda \succeq 0, \mu \succeq 0$. We minimize over the variable t to get

$$-A_x^T \nu - A_y^T \kappa - \lambda + \mu = 0,$$

which leaves

$$g(\nu, \kappa, \lambda, \mu) = \kappa^T w + \lambda^T T^{\min} - \mu^T T^{\max}.$$

Thus, the alternative is:

$$\begin{aligned} -A_x^T \nu - A_y^T \kappa - \lambda + \mu &= 0, \\ \kappa^T w + \lambda^T T^{\min} - \mu^T T^{\max} &= 1, \\ \lambda \succeq 0, \quad \mu \succeq 0. \end{aligned}$$

where the 1 on righthand side of the second equation comes from homogeneity. We can now say: w is unsupportable if and only if there are $\nu, \kappa, \lambda, \mu$ satisfying the equations and inequalities above.

Now suppose w is unsupportable, with total weight $W = \mathbf{1}^T w$, and let $\nu, \kappa, \lambda, \mu$ be the corresponding dual variables satisfying the equations and inequalities above.

Since $\mu, \lambda \succeq 0$ and $T^{\min} \preceq 0 \preceq T^{\max}$, we have $\lambda^T T^{\min} - \mu^T T^{\max} \leq 0$. So $\kappa^T w \geq 1$, which implies that κ has at least one positive component, since w is elementwise positive. Let k be such that $\kappa_k = \max_j \kappa_j$. From above, we know that $\kappa_k > 0$. Now define $\tilde{w} = \alpha e_k$, where $\alpha = \kappa^T w / \kappa_k$. Then the equations and inequalities above hold for \tilde{w} , which tells us that \tilde{w} is also unsupportable. Now we will show that $\mathbf{1}^T \tilde{w} \leq W$, which means the new weight distribution has a total weight not exceeding the original weight. Note that

$$\mathbf{1}^T \tilde{w} = \alpha = (\kappa / \kappa_k)^T w \leq \mathbf{1}^T w = W,$$

because the vector κ / κ_k has all entries ≤ 1 (and $w \succeq 0$).

So we've shown that given any unsupportable weight vector w , we can construct an unsupportable weight vector, with no more total weight, that is concentrated at a single node.

We gave varying amounts of partial credit for explanations of why the least total weight that is unsupportable should be concentrated at a single node.

- (c) We find that the maximum total weight that is supportable is 5.79, with corresponding weight vector

`max_supp_w =`

```
1.2053
0.8166
0.7599
0.9818
0.5918
1.4369.
```

We also find that the minimum unsupportable weight is 1.2 (placed at node 4). The following code solves this problem.

```
% truss load analysis
truss_load_data;

Ax = A(1:p,:);
Ay = A(p+1:2*p,:);

% maximum supportable total weight
cvx_begin quiet
    variables w(p) t(m)
    maximize( sum(w) )
    subject to
        w >= 0;
        t >= Tmin;
        t <= Tmax;
        Ax*t == 0;
        Ay*t == w;
cvx_end

max_supp_w = w
max_supp_weight = sum(w)

% minimum unsupportable total weight
% to find this, we loop over the free nodes
w_min_single = zeros(p,1);
for k = 1:p
    ek = zeros(p,1);
    ek(k) = 1;
    cvx_begin quiet
        variables alpha t(m)
        maximize( alpha )
        subject to
            alpha >= 0;
            t >= Tmin;
            t <= Tmax;
            Ax*t == 0;
            Ay*t == alpha*ek;
    cvx_end
    % store max supportable weight at node k
    % (= min unsupportable weight at node k)
    w_min_single(k) = alpha;
end

[min_unsupp_weight weakest_node] = min(w_min_single);
```

```

min_usupp_w = zeros(p,1);
min_usupp_w(weakest_node) = min_unsupp_weight
min_unsupp_weight

```

14.7 Quickest take-off. This problem concerns the braking and thrust profiles for an airplane during take-off. For simplicity we will use a discrete-time model. The position (down the runway) and the velocity in time period t are p_t and v_t , respectively, for $t = 0, 1, \dots$. These satisfy $p_0 = 0$, $v_0 = 0$, and $p_{t+1} = p_t + hv_t$, $t = 0, 1, \dots$, where $h > 0$ is the sampling time period. The velocity updates as

$$v_{t+1} = (1 - \eta)v_t + h(f_t - b_t), \quad t = 0, 1, \dots,$$

where $\eta \in (0, 1)$ is a friction or drag parameter, f_t is the engine thrust, and b_t is the braking force, at time period t . These must satisfy

$$0 \leq b_t \leq \min\{B^{\max}, f_t\}, \quad 0 \leq f_t \leq F^{\max}, \quad t = 0, 1, \dots,$$

as well as a constraint on how fast the engine thrust can be changed,

$$|f_{t+1} - f_t| \leq S, \quad t = 0, 1, \dots$$

Here B^{\max} , F^{\max} , and S are given parameters. The initial thrust is $f_0 = 0$. The take-off time is $T^{\text{to}} = \min\{t \mid v_t \geq V^{\text{to}}\}$, where V^{to} is a given take-off velocity. The take-off position is $P^{\text{to}} = p_{T^{\text{to}}}$, the position of the aircraft at the take-off time. The length of the runway is $L > 0$, so we must have $P^{\text{to}} \leq L$.

- (a) Explain how to find the thrust and braking profiles that minimize the take-off time T^{to} , respecting all constraints. Your solution can involve solving more than one convex problem, if necessary.
- (b) Solve the quickest take-off problem with data

$$h = 1, \quad \eta = 0.05, \quad B^{\max} = 0.5, \quad F^{\max} = 4, \quad S = 0.8, \quad V^{\text{to}} = 40, \quad L = 300.$$

Plot p_t , v_t , f_t , and b_t versus t . Comment on what you see. Report the take-off time and take-off position for the profile you find.

Solution. To check if $T^{\text{to}} = T$ is possible, we solve the (convex) feasibility inequalities

$$\begin{aligned} p_0 &= 0, \quad p_{t+1} = p_t + hv_t, \quad t = 0, \dots, T-1 \\ v_0 &= 0, \quad v_{t+1} = (1 - \eta)v_t + h(f_t - b_t), \quad t = 0, \dots, T-1 \\ 0 \leq b_t &\leq \min\{B^{\max}, f_t\}, \quad t = 0, \dots, T \\ f_0 &= 0, \quad 0 \leq f_t \leq F^{\max}, \quad t = 0, \dots, T \\ |f_{t+1} - f_t| &\leq S, \quad t = 0, 1, \dots, T-1 \\ v_T &\geq V^{\text{to}}, \quad p_T \leq L, \end{aligned}$$

with variables $p, v, f, b \in \mathbf{R}^{T+1}$. To find the quickest take-off, we start with $T = 1$ (say) and increment it until the constraints above are feasible.

The following code solves the problem.

```

% solution for quickest take-off problem
h = 1;
eta = 0.05;
Bmax = 0.5;
Fmax = 4;
S = 0.8;
Vto = 40;
L = 300;

T = 17; % infeasible for T=1,...,16
cvx_begin
    variables p(T+1) v(T+1) f(T+1) b(T+1)
    minimize (p(T+1)) % objective not needed
    subject to
        p(1) == 0;
        p(2:T+1) == p(1:T) + h*v(1:T);
        p(T+1) <= L;
        v(1) == 0;
        v(T+1) >= Vto;
        v(2:T+1) == (1-eta)*v(1:T) + h*(f(1:T)-b(1:T));
        f(1) == 0;
        0 <= b;
        b <= Bmax;
        b <= f;
        0 <= f;
        f <= Fmax;
        abs(f(2:T+1)-f(1:T)) <= S;
cvx_end

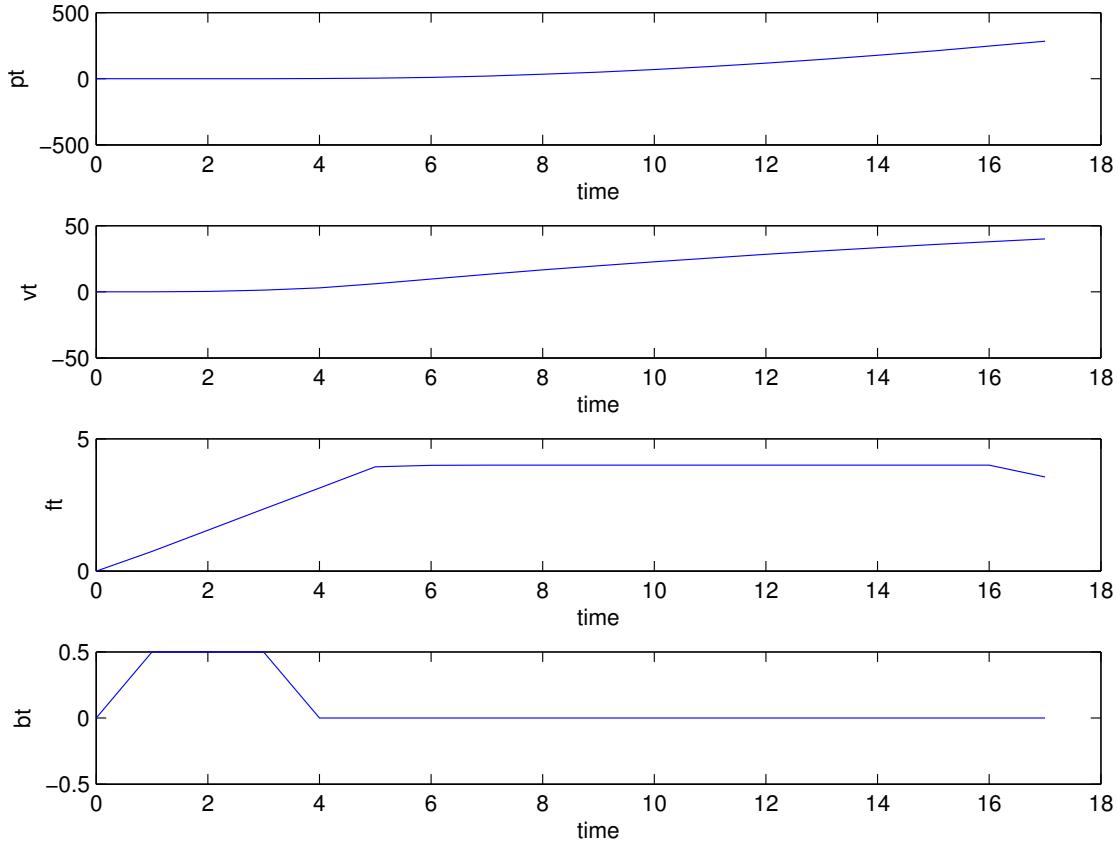
% plots
subplot(4,1,1); plot([0:T]', p); xlabel('time'); ylabel('pt');
subplot(4,1,2); plot([0:T]', v); xlabel('time'); ylabel('vt');
subplot(4,1,3); plot([0:T]', f); xlabel('time'); ylabel('ft');
subplot(4,1,4); plot([0:T]', b); xlabel('time'); ylabel('bt');
print -depsc quickest_takeoff

```

The quickest take-off time is $T^{\text{to}*} = 17$. The profiles for quickest take-off are shown below. You can check that the use of the brake is not optional; the problem becomes infeasible when you disable it (say, by setting $B^{\text{max}} = 0$). At first it seems strange that a brake (which, after all, is used to slow an airplane down) is critical in take-off. But a bit of thought reveals why it's needed: You lock the aircraft down while the engine thrusts up. If you don't, you're already moving down the runway at less than full thrust, which means you'll be farther along the runway when you take off. By the way, if you make the runway a bit longer, the brake isn't needed; you just thrust up to max thrust right from the start.

We can also note that if you can take off at time T , then you can take off in any later time period. To see this, suppose that $s \geq 0$, and f_t, b_t are trajectories that allow take-off at time T . Then we

set $\tilde{f}_t = \tilde{b}_t = 0$ for $t = 0, \dots, s$, and $\tilde{f}_t = f_{t-s}$, $\tilde{b}_t = f_{t-s}$ for $t = s, \dots, T+s$. This trajectory is also feasible, and leads to take-off time $T+s$ and the same take-off location.



14.8 Optimal spacecraft landing. We consider the problem of optimizing the thrust profile for a spacecraft to carry out a landing at a target position. The spacecraft dynamics are

$$m\ddot{p} = f - mge_3,$$

where $m > 0$ is the spacecraft mass, $p(t) \in \mathbf{R}^3$ is the spacecraft position, with 0 the target landing position and $p_3(t)$ representing height, $f(t) \in \mathbf{R}^3$ is the thrust force, and $g > 0$ is the gravitational acceleration. (For simplicity we assume that the spacecraft mass is constant. This is not always a good assumption, since the mass decreases with fuel use. We will also ignore any atmospheric friction.) We must have $p(T^{\text{td}}) = 0$ and $\dot{p}(T^{\text{td}}) = 0$, where T^{td} is the touchdown time. The spacecraft must remain in a region given by

$$p_3(t) \geq \alpha \|(p_1(t), p_2(t))\|_2,$$

where $\alpha > 0$ is a given minimum glide slope. The initial position $p(0)$ and velocity $\dot{p}(0)$ are given. The thrust force $f(t)$ is obtained from a single rocket engine on the spacecraft, with a given maximum thrust; an attitude control system rotates the spacecraft to achieve any desired direction

of thrust. The thrust force is therefore characterized by the constraint $\|f(t)\|_2 \leq F^{\max}$. The fuel use rate is proportional to the thrust force magnitude, so the total fuel use is

$$\int_0^{T^{\text{td}}} \gamma \|f(t)\|_2 dt,$$

where $\gamma > 0$ is the fuel consumption coefficient. The thrust force is discretized in time, *i.e.*, it is constant over consecutive time periods of length $h > 0$, with $f(t) = f_k$ for $t \in [(k-1)h, kh)$, for $k = 1, \dots, K$, where $T^{\text{td}} = Kh$. Therefore we have

$$v_{k+1} = v_k + (h/m)f_k - hge_3, \quad p_{k+1} = p_k + (h/2)(v_k + v_{k+1}),$$

where p_k denotes $p((k-1)h)$, and v_k denotes $\dot{p}((k-1)h)$. We will work with this discrete-time model. For simplicity, we will impose the glide slope constraint only at the times $t = 0, h, 2h, \dots, Kh$.

- (a) *Minimum fuel descent.* Explain how to find the thrust profile f_1, \dots, f_K that minimizes fuel consumption, given the touchdown time $T^{\text{td}} = Kh$ and discretization time h .
- (b) *Minimum time descent.* Explain how to find the thrust profile that minimizes the touchdown time, *i.e.*, K , with h fixed and given. Your method can involve solving several convex optimization problems.
- (c) Carry out the methods described in parts (a) and (b) above on the problem instance with data given in `spacecraft_landing_data.*`. Report the optimal total fuel consumption for part (a), and the minimum touchdown time for part (b). The data files also contain plotting code (commented out) to help you visualize your solution. Use the code to plot the spacecraft trajectory and thrust profiles you obtained for parts (a) and (b).

Hints.

- In Julia, the plot will come out rotated.

Remarks. If you'd like to see the ideas of this problem in action, watch these videos:

- <http://www.youtube.com/watch?v=2t15vP1PyoA>
- <https://www.youtube.com/watch?v=orUjSkc2pG0>
- <https://www.youtube.com/watch?v=1B6oiLNyKKI>
- <https://www.youtube.com/watch?v=ZCBE8oc0kAQ>

Solution.

- (a) To find the minimum fuel thrust profile for a given K , we solve

$$\begin{aligned} & \text{minimize} && \sum_{k=1}^K \|f_k\|_2 \\ & \text{subject to} && v_{k+1} = v_k + (h/m)f_k - hge_3, \quad p_{k+1} = p_k + (h/2)(v_k + v_{k+1}), \\ & && \|f_k\|_2 \leq F^{\max}, \quad (p_k)_3 \geq \alpha \|((p_k)_1, (p_k)_2)\|_2, \quad k = 1, \dots, K \\ & && p_{K+1} = 0, \quad v_{K+1} = 0, \quad p_1 = p(0), \quad v_1 = \dot{p}(0), \end{aligned}$$

with variables p_1, \dots, p_{K+1} , v_1, \dots, v_{K+1} , and f_1, \dots, f_K . This is a convex optimization problem.

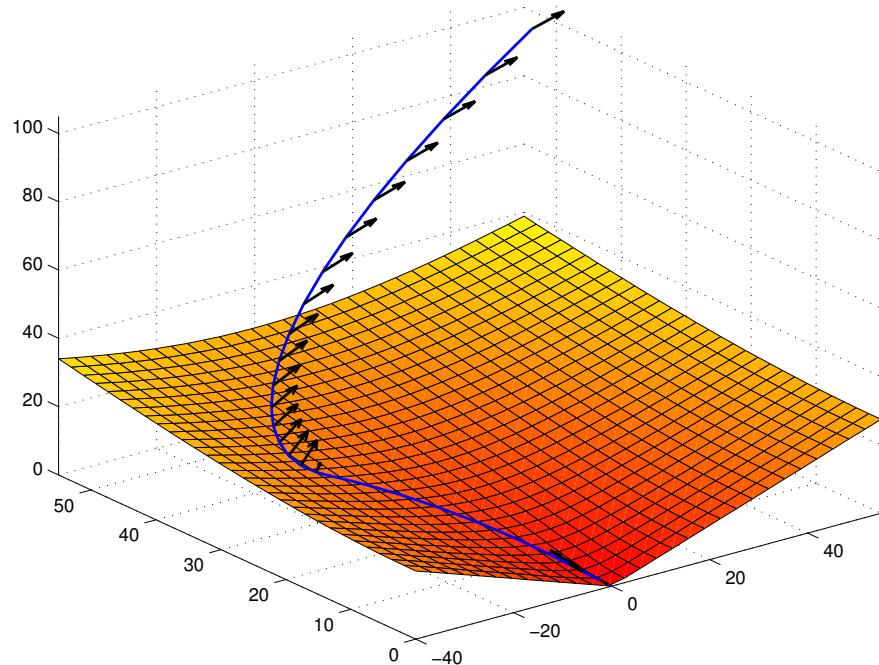
- (b) We can solve a sequence of convex feasibility problems to find the minimum touchdown time.
For each K we solve

$$\begin{aligned} & \text{minimize} && 0 \\ & \text{subject to} && v_{k+1} = v_k + (h/m)f_k - hge_3, \quad p_{k+1} = p_k + (h/2)(v_k + v_{k+1}), \\ & && \|f_k\|_2 \leq F^{\max}, \quad (p_k)_3 \geq \alpha \|((p_k)_1, (p_k)_2)\|_2, \quad k = 1, \dots, K \\ & && p_{K+1} = 0, \quad v_{K+1} = 0, \quad p_1 = p(0), \quad v_1 = \dot{p}(0), \end{aligned}$$

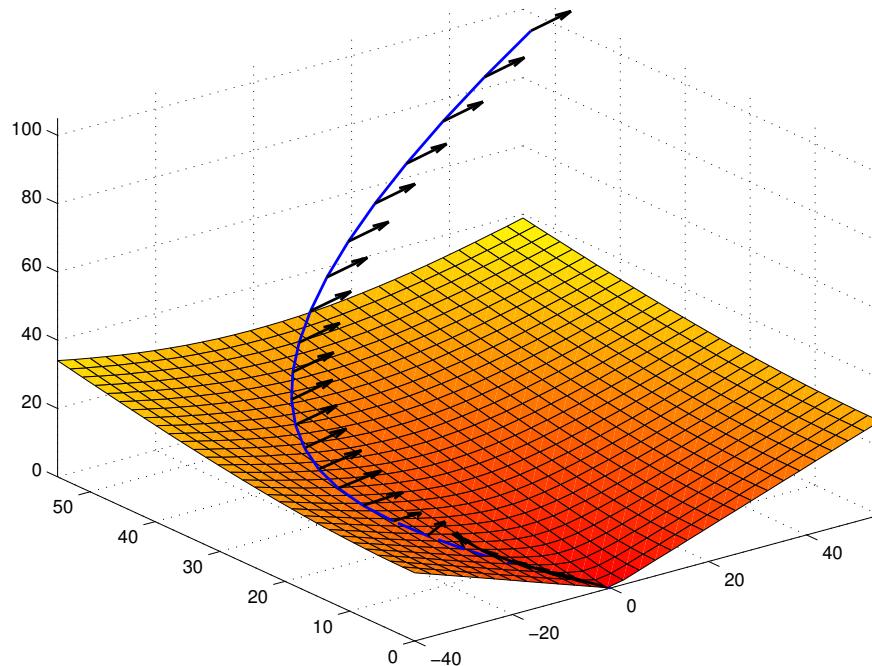
with variables p_1, \dots, p_{K+1} , v_1, \dots, v_{K+1} , and f_1, \dots, f_K . If the problem is feasible, we reduce K , otherwise we increase K . We iterate until we find the smallest K for which a feasible trajectory can be found. (In fact, the problem is quasiconvex as long as $(1/m)F^{\max} \geq g$, so we can use bisection to speed up our search.)

- (c) For part (a) the optimal total fuel consumption is 193.0. For part (b) the minimum touchdown time is $K = 25$. The following plots show the trajectories we obtain. The blue line shows the position of the spacecraft, the black arrows show the thrust profile, and the colored surface shows the glide slope constraint.

Here is the minimum fuel trajectory for part (a). Notice that for a portion of the trajectory the thrust is exactly equal to zero (which we would expect, given our cost function).



Here is a minimum time trajectory for part (b).



The following code solves the problem. In Matlab:

```

spacecraft_landing_data;

% solve part (a) (find minimum fuel trajectory)
cvx_solver sdpt3;
cvx_begin
    variables p(3,K+1) v(3,K+1) f(3,K)
    v(:,2:K+1) == v(:,1:K)+(h/m)*f-h*g*repmat([0;0;1],1,K);
    p(:,2:K+1) == p(:,1:K)+(h/2)*(v(:,1:K)+v(:,2:K+1));
    p(:,1) == p0; v(:,1) == v0;
    p(:,K+1) == 0; v(:,K+1) == 0;
    p(3,:) >= alpha*norms(p(1:2,:));
    norms(f) <= Fmax;
    minimize(sum(norms(f)))
cvx_end
min_fuel = cvx_optval*gamma*h;
p_minf = p; v_minf = v; f_minf = f;

% solve part (b) (find minimum K)
% we will use a linear search, but bisection is faster
Ki = K;
while(1)
    cvx_begin
        variables p(3,Ki+1) v(3,Ki+1) f(3,Ki)
        v(:,2:Ki+1) == v(:,1:Ki)+(h/m)*f-h*g*repmat([0;0;1],1,Ki);

```

```

p(:,2:Ki+1) == p(:,1:Ki)+(h/2)*(v(:,1:Ki)+v(:,2:Ki+1));
p(:,1) == p0; v(:,1) == v0;
p(:,Ki+1) == 0; v(:,Ki+1) == 0;
p(3,:) >= alpha*norms(p(1:2,:));
norms(f) <= Fmax;
minimize(sum(norms(f)))
cvx_end
if(strcmp(cvx_status,'Infeasible') == 1)
    Kmin = Ki+1;
    break;
end
Ki = Ki-1;
p_mink = p; v_mink = v; f_mink = f;
end

% plot the glide cone
x = linspace(-40,55,30); y = linspace(0,55,30);
[X,Y] = meshgrid(x,y);
Z = alpha*sqrt(X.^2+Y.^2);
figure; colormap autumn; surf(X,Y,Z);
axis([-40,55,0,55,0,105]);
grid on; hold on;

% plot minimum fuel trajectory for part (a)
plot3(p_minf(1,:),p_minf(2,:),p_minf(3,:),'b','linewidth',1.5);
quiver3(p_minf(1,1:K),p_minf(2,1:K),p_minf(3,1:K),...
         f_minf(1,:),f_minf(2,:),f_minf(3,:),0.3,'k','linewidth',1.5);
print('-depsc','spacecraft_landing_a.eps');

% plot minimum time trajectory for part (b)
figure; colormap autumn; surf(X,Y,Z);
axis([-40,55,0,55,0,105]); grid on; hold on;
plot3(p_mink(1,:),p_mink(2,:),p_mink(3,:),'b','linewidth',1.5);
quiver3(p_mink(1,1:Kmin),p_mink(2,1:Kmin),p_mink(3,1:Kmin),...
         f_mink(1,:),f_mink(2,:),f_mink(3,:),0.3,'k','linewidth',1.5);
print('-depsc','spacecraft_landing_b.eps');

```

In Python:

```

import numpy as np
import cvxpy as cvx
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm

h = 1.
g = 0.1

```

```

m = 10.
Fmax = 10.
p0 = np.matrix('50;50;100')
v0 = np.matrix('-10;0;-10')
alpha = 0.5
gamma = 1.
K = 35

#Solution begins
#-----

e3 = np.matrix('0; 0; 1')

p = cvx.Variable(3, K+1)
v = cvx.Variable(3, K+1)
f = cvx.Variable (3, K)

fuel_use = h*gamma*sum([cvx.norm(f[:,i]) for i in range(K)])

const = [v[:,i+1] == v[:,i] + (h/m)*f[:,i]-h*g*e3 for i in range(K)]
const += [p[:,i+1] == p[:,i] + h/2*(v[:,i]+v[:,i+1]) for i in range(K)]
const += [p[:,0]==p0, v[:,0]==v0]
const += [p[:,K]==0, v[:,K]==0]
const += [p[2,i] >= alpha*cvx.norm(p[0:2,i]) for i in range(K+1)]
const += [cvx.norm(f[:,i]) <= Fmax for i in range(K)]

prob = cvx.Problem(cvx.Minimize(fuel_use), const)
prob.solve()
print 'Minimum fuel use is %.2f' % fuel_use.value

# Minimum fuel trajectory and glide cone
fig = plt.figure()
ax = fig.gca(projection='3d')

X = np.linspace(-40, 55, num=30)
Y = np.linspace(0, 55, num=30)
X, Y = np.meshgrid(X, Y)
Z = alpha*np.sqrt(X**2+Y**2);
ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap=cm.coolwarm, linewidth=0)

ax.plot(xs=p.value[0,:].A1,ys=p.value[1,:].A1,zs=p.value[2,:].A1)
ax.set_xlabel('x'); ax.set_ylabel('y'); ax.set_zlabel('z')

#For minimum time descent, we do a linear search but bisection would be faster.

```

```

K = 1
while True:
    p = cvx.Variable(3, K+1)
    v = cvx.Variable(3, K+1)
    f = cvx.Variable(3, K)

    const = [v[:,i+1] == v[:,i] + (h/m)*f[:,i]-h*g*e3 for i in range(K)]
    const += [p[:,i+1] == p[:,i] + h/2*(v[:,i]+v[:,i+1]) for i in range(K)]
    const += [p[:,0]==p0, v[:,0]==v0]
    const += [p[:,K]==0, v[:,K]==0]
    const += [p[2,i] >= alpha*cvx.norm(p[0:2,i]) for i in range(K+1)]
    const += [cvx.norm(f[:,i]) <= Fmax for i in range(K)]

    prob = cvx.Problem(cvx.Minimize(0), const)
    prob.solve()

    if prob.status=='optimal':
        break
    K += 1

print 'The minimum touchdown time is', K

#Minimum time trajectory and glide cone
fig = plt.figure()
ax = fig.gca(projection='3d')

X = np.linspace(-40, 55, num=30)
Y = np.linspace(0, 55, num=30)
X, Y = np.meshgrid(X, Y)
Z = alpha*np.sqrt(X**2+Y**2);
ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap=cm.coolwarm, linewidth=0)
ax.plot(xs=p.value[0,:].A1,ys=p.value[1,:].A1,zs=p.value[2,:].A1)
ax.set_xlabel('x'); ax.set_ylabel('y'); ax.set_zlabel('z')
plt.show()

```

In Julia:

```

include("spacecraft_landing_data.jl");
using Convex, SCS
solver = SCSSolver(max_iters=20000, verbose=false);

# solve part (a) (find minimum fuel trajectory)
p = Variable(3, K+1);
v = Variable(3, K+1);
f = Variable(3, K);

```

```

constraints = [];
constraints += v[:,2:K+1] == v[:,1:K] + (h/m)*f - h*g*repmat([0,0,1], 1, K);
constraints += p[:,2:K+1] == p[:,1:K] + (h/2)*(v[:,1:K] + v[:,2:K+1]);
constraints += p[:,1] == p0;
constraints += v[:,1] == v0;
constraints += p[:,K+1] == 0;
constraints += v[:,K+1] == 0;
for i = 1:K+1
    constraints += p[3,i] >= alpha*norm(p[1:2,i]);
end
objective = 0;
for i = 1:K
    constraints += norm(f[:,i]) <= Fmax;
    objective += norm(f[:,i]);
end
problem = minimize(objective, constraints);
solve!(problem, solver);
min_fuel = problem.optval*gamma*h;
p_minf = p.value'; v_minf = v.value'; f_minf = f.value';

# solve part (b) (find minimum K)
# we will use a linear search, but bisection is faster
Kmin = K;
p_mink = nothing;
v_mink = nothing;
f_mink = nothing;
while true
    p = Variable(3, Kmin+1);
    v = Variable(3, Kmin+1);
    f = Variable(3, Kmin);
    constraints = [];
    constraints += v[:,2:Kmin+1] == v[:,1:Kmin] + (h/m)*f - h*g*repmat([0,0,1], 1, Kmin);
    constraints += p[:,2:Kmin+1] == p[:,1:Kmin] + (h/2)*(v[:,1:Kmin] + v[:,2:Kmin+1]);
    constraints += p[:,1] == p0;
    constraints += v[:,1] == v0;
    constraints += p[:,Kmin+1] == 0;
    constraints += v[:,Kmin+1] == 0;
    for i = 1:Kmin+1
        constraints += p[3,i] >= alpha*norm(p[1:2,i]);
    end
    objective = 0;
    for i = 1:Kmin
        constraints += norm(f[:,i]) <= Fmax;
        objective += norm(f[:,i]);
    end

```

```

problem = minimize(objective, constraints);
solve!(problem, solver);

if problem.status != :Optimal
    Kmin += 1;
    break;
end
Kmin -= 1;
p_mink = p.value'; v_mink = v.value'; f_mink = f.value';
end

# plot the glide cone
using PyPlot
x = linspace(-40,55,30); y = linspace(0,55,30);
X = repmat(x', length(x), 1);
Y = repmat(y, 1, length(y));
Z = alpha*sqrt(X.^2+Y.^2);
figure();
grid(true);
hold(true);
surf(X, Y, Z, cmap=get_cmap("autumn"));
xlim([-40, 55]);
ylim([0, 55]);
zlim([0, 105]);

# plot minimum fuel trajectory for part (a)
plot(p_minf[:,1], p_minf[:,2], p_minf[:,3]);
heads = p_minf[1:K,:] + f_minf;
quiver(heads[:,1], heads[:,2], heads[:,3],
        f_minf[:,1], f_minf[:,2], f_minf[:,3], length=10);

# plot minimum time trajectory for part (b)
figure();
grid(true);
hold(true);
surf(X, Y, Z, cmap=get_cmap("autumn"));
xlim([-40, 55]);
ylim([0, 55]);
zlim([0, 105]);
plot(p_mink[:,1], p_mink[:,2], p_mink[:,3]);
heads_k = p_mink[1:Kmin,:] + f_mink;
quiver(heads_k[:,1], heads_k[:,2], heads_k[:,3],
        f_mink[:,1], f_mink[:,2], f_mink[:,3], length=10);

```

14.9 Feedback gain optimization. A system (such as an industrial plant) is characterized by $y = Gu + v$, where $y \in \mathbf{R}^n$ is the output, $u \in \mathbf{R}^n$ is the input, and $v \in \mathbf{R}^n$ is a disturbance signal. The matrix

$G \in \mathbf{R}^{n \times n}$, which is known, is called the system input-output matrix. The input signal u is found using a linear feedback (control) policy: $u = Fy$, where $F \in \mathbf{R}^{n \times n}$ is the feedback (gain) matrix, which is what we need to determine. From the equations given above, we have

$$y = (I - GF)^{-1}v, \quad u = F(I - GF)^{-1}v.$$

(You can simply assume that $I - GF$ will be invertible.)

The disturbance v is random, with $\mathbf{E} v = 0$, $\mathbf{E} vv^T = \sigma^2 I$, where σ is known. The objective is to minimize $\max_{i=1,\dots,n} \mathbf{E} y_i^2$, the maximum mean square value of the output components, subject to the constraint that $\mathbf{E} u_i^2 \leq 1$, $i = 1, \dots, n$, i.e., each input component has a mean square value not exceeding one. The variable to be chosen is the matrix $F \in \mathbf{R}^{n \times n}$.

(a) Explain how to use convex (or quasi-convex) optimization to find an optimal feedback gain matrix. As usual, you must fully explain any change of variables or other transformations you carry out, and why your formulation solves the problem described above. A few comments:

- You can assume that matrices arising in your change of variables are invertible; you do not need to worry about the special cases when they are not.
- You can assume that G is invertible if you need to, but we will deduct a few points from these answers.

(b) Carry out your method for the problem instance with data

$$\sigma = 1, \quad G = \begin{bmatrix} 0.3 & -0.1 & -0.9 \\ -0.6 & 0.3 & -0.3 \\ -0.3 & 0.6 & 0.2 \end{bmatrix}.$$

Give an optimal F , and the associated optimal objective value.

Solution. We first note that

$$\mathbf{E} yy^T = \sigma^2(I - GF)^{-1}(I - GF)^{-T}, \quad \mathbf{E} uu^T = \sigma^2 F(I - GF)^{-1}(I - GF)^{-T}F^T.$$

The diagonal entries of these are $(\mathbf{E} y_1^2, \dots, \mathbf{E} y_n^2)$ and $(\mathbf{E} u_1^2, \dots, \mathbf{E} u_n^2)$, respectively. We can also express these quantities as the ℓ_2 -norm squared of the rows of the matrices $(I - GF)^{-1}$ and $F(I - GF)^{-1}$, respectively.

The problem is therefore

$$\begin{aligned} & \text{minimize} && \sigma^2 \max_{i=1,\dots,n} \left((I - GF)^{-1}(I - GF)^{-T} \right)_{ii} \\ & \text{subject to} && \sigma^2 \left(F(I - GF)^{-1}(I - GF)^{-T}F^T \right)_{ii} \leq 1, \quad i = 1, \dots, n, \end{aligned}$$

with variable $F \in \mathbf{R}^{n \times n}$. In this form it is not a convex problem.

We are clearly going to have to change variables to obtain a convex problem. In fact, several changes of variables lead to a convex problem.

Approach #1. One change of variables uses $Z = F(I - GF)^{-1}$. First let's see how we can recover F from Z . We start with $Z(I - GF) = F$, from which we then get $F = (I + ZG)^{-1}Z$. (Here we simply assume that $I + ZG$ is invertible.)

The constraints are $(ZZ^T)_{ii} \leq 1/\sigma^2$, or equivalently, that the rows of Z should have ℓ_2 norm not exceeding $1/\sigma$. So the constraints are convex in Z . Now let's handle the objective. We need to express $(I - GF)^{-1}$ in terms of $Z = F(I - GF)^{-1}$. We start with

$$I = (I - GF)(I - GF)^{-1} = (I - GF)^{-1} - GF(I - GF)^{-1}.$$

We rewrite this as

$$(I - GF)^{-1} = I + GZ.$$

Thus the objective (ignoring the constant σ^2) is the maximum of the ℓ_2 norm squared of the rows of $I + GZ$, which is clearly convex. Whew!

Here is the final (convex) problem we solve:

$$\begin{aligned} & \text{minimize} && \sigma^2 \max_{i=1,\dots,n} ((I + GZ)(I + GZ)^T)_{ii} \\ & \text{subject to} && \sigma^2 (ZZ^T)_{ii} \leq 1, \quad i = 1, \dots, n, \end{aligned}$$

with variable Z . (The two expressions are really just the ℓ_2 -norms squared of the rows of the matrices.) We recover an optimal F from an optimal Z using $F = (I + ZG)^{-1}Z$.

Approach #2. Here we use the variable $Y = (I - GF)^{-1}$. We recover F using $F = G^{-1}(I - Y^{-1})$, so this approach requires that G be invertible.

In this case the objective is simple and convex; it is the maximum of the ℓ_2 norms of the rows of Y (squared, and multiplied by σ^2).

We'll have to work to handle the constraints, though. Using the formula above for F , we have

$$F(I - GF)^{-1} = FY = G^{-1}(I - Y^{-1})Y = G^{-1}(Y - I).$$

So now we see that the constraints are that the rows of the matrix $G^{-1}(Y - I)$ have ℓ_2 norm not exceeding $1/\sigma$, which is convex.

Our convex problem is then

$$\begin{aligned} & \text{minimize} && \sigma^2 \max_{i=1,\dots,n} (YY^T)_{ii} \\ & \text{subject to} && \sigma^2 (G^{-1}(Y - I)(G^{-1}(Y - I))^T)_{ii} \leq 1, \quad i = 1, \dots, n, \end{aligned}$$

with variable Y , with F recovered from $F = G^{-1}(I - Y^{-1})$.

Approach #3. Here is another approach used by several people, which is a hybrid of the approaches given above. This approach uses both the new variables:

$$Y = (I - GF)^{-1}, \quad Z = F(I - GF)^{-1},$$

with the constraint that $Z = FY = G^{-1}(Y - I)$ (assuming that G is invertible here), which we can also write as $GZ = Y - I$, which avoids the assumption that G is invertible. This change of variables is not a bijection, that is, a one-to-one correspondence between the old and new variables. The correct statement is that for each F , there exists a pair Z and Y satisfying $GZ = Y - I$, $Y = (I - GF)^{-1}$, $Z = F(I - GF)^{-1}$; conversely, for each pair Z , Y satisfying $GZ = Y - I$, there exists an F for which $Y = (I - GF)^{-1}$, $Z = F(I - GF)^{-1}$. (Here we are informal about the existence of inverses.)

We end up with the convex problem

$$\begin{aligned} & \text{minimize} && \sigma^2 \max_{i=1,\dots,n} (YY^T)_{ii} \\ & \text{subject to} && \sigma^2 (ZZ^T)_{ii} \leq 1, \quad i = 1, \dots, n, \\ & && GZ = Y - I, \end{aligned}$$

with variables Z and Y , with F recovered from $F = ZY^{-1}$.

Numerical instance. The code below solves the problem (using approaches 1 and 2, and sure enough, the results agree). The optimal value is 0.1647, and an optimal F is

$$F = \begin{bmatrix} -1.2990 & 3.6886 & 3.4530 \\ 0.2103 & -0.8963 & -3.3598 \\ 15.5258 & 7.3230 & -9.1520 \end{bmatrix}.$$

```
% feedback gain optimization
G = [0.3 -0.1 -0.9; -0.6 0.3 -0.3; -0.3 0.6 0.2];
sigma = 1;

% approach 1, using Z = F(I-GF)^{-1}
cvx_begin
    variable Z(3,3);
    minimize (sigma*max(norms((eye(3)+G*Z)')));
    norms(Z') <= 1/sigma; % rows of Z have norm less than 1/sigma
cvx_end
% recover feedback matrix
opt_val = cvx_optval^2
F = inv(eye(3)+Z*G)*Z

% approach 2, using Y=(I-GF)^{-1}
cvx_begin
    variable Y(3,3);
    minimize (sigma*max(norms(Y')));
    norms((inv(G)*(Y-eye(3)))') <= 1/sigma;
cvx_end
% recover feedback matrix
opt_val2 = cvx_optval^2
F2 = inv(G)*(eye(3)-inv(Y))
```

14.10 Fuel use as function of distance and speed. A vehicle uses fuel at a rate $f(s)$, which is a function of the vehicle speed s . We assume that $f : \mathbf{R} \rightarrow \mathbf{R}$ is a positive increasing convex function, with $\text{dom } f = \mathbf{R}_+$. The physical units of s are m/s (meters per second), and the physical units of $f(s)$ are kg/s (kilograms per second).

- (a) Let $g(d, t)$ be the total fuel used (in kg) when the vehicle moves a distance $d \geq 0$ (in meters) in time $t > 0$ (in seconds) at a constant speed. Show that g is convex.
- (b) Let $h(d)$ be the minimum fuel used (in kg) to move a distance d (in m) at a constant speed s (in m/s). Show that h is convex.

Solution.

- (a) $g(d, t) = tf(d/t)$ is the perspective of f , and therefore convex.
- (b) $h(d) = \inf_{t>0} g(d, t)$ is obtained by partial minimization of g , so is convex.

14.11 Minimum time speed profile along a road. A vehicle of mass $m > 0$ moves along a road in \mathbf{R}^3 , which is piecewise linear with given knot points $p_1, \dots, p_{N+1} \in \mathbf{R}^3$, starting at p_1 and ending at p_{N+1} . We let $h_i = (p_i)_3$, the z -coordinate of the knot point; these are the heights of the knot points (above sea-level, say). For your convenience, these knot points are equidistant, i.e., $\|p_{i+1} - p_i\|_2 = d$ for all i . (The points give an arc-length parametrization of the road.) We let $s_i > 0$ denote the (constant) vehicle speed as it moves along road segment i , from p_i to p_{i+1} , for $i = 1, \dots, N$, and $s_{N+1} \geq 0$ denote the vehicle speed after it passes through knot point p_{N+1} . Our goal is to minimize the total time to traverse the road, which we denote T .

We let $f_i \geq 0$ denote the total fuel burnt while traversing the i th segment. This fuel burn is turned into an increase in vehicle energy given by ηf_i , where $\eta > 0$ is a constant that includes the engine efficiency and the energy content of the fuel. While traversing the i th road segment the vehicle is subject to a drag force, given by $C_D s_i^2$, where $C_D > 0$ is the coefficient of drag, which results in an energy loss $dC_D s_i^2$.

We derive equations that relate these quantities via energy balance:

$$\frac{1}{2}ms_{i+1}^2 + mgh_{i+1} = \frac{1}{2}ms_i^2 + mgh_i + \eta f_i - dC_D s_i^2, \quad i = 1, \dots, N,$$

where $g = 9.8$ is the gravitational acceleration. The lefthand side is the total vehicle energy (kinetic plus potential) after it passes through knot point p_{i+1} ; the righthand side is the total vehicle energy after it passes through knot point p_i , plus the energy gain from the fuel burn, minus the energy lost to drag. To set up the first vehicle speed s_1 requires an additional initial fuel burn f_0 , with $\eta f_0 = \frac{1}{2}ms_1^2$.

Fuel is also used to power the on-board system of the vehicle. The total fuel used for this purpose is f_{ob} , where $\eta f_{\text{ob}} = TP$, where $P > 0$ is the (constant) power consumption of the on-board system. We have a fuel capacity constraint: $\sum_{i=0}^N f_i + f_{\text{ob}} \leq F$, where $F > 0$ is the total initial fuel.

The problem data are m , d , h_1, \dots, h_{N+1} , η , C_D , P , and F . (You don't need the knot points p_i .)

- (a) Explain how to find the fuel burn levels f_0, \dots, f_N that minimize the time T , subject to the constraints.

- (b) Carry out the method described in part (a) for the problem instance with data given in `min_time_speed_data.m`. Give the optimal time T^* , and compare it to the time T^{unif} achieved if the fuel for propulsion were burned uniformly, *i.e.*, $f_0 = \dots = f_N$. For each of these cases, plot speed versus distance along the road, using the plotting code in the data file as a template.

Solution.

- (a) The time to traverse the i th segment is d/s_i , so the total time is

$$T = \sum_{i=1}^N \frac{d}{s_i}.$$

We are to solve the problem

$$\begin{aligned} & \text{minimize} && \sum_{i=1}^N d/s_i \\ & \text{subject to} && \frac{1}{2}ms_{i+1}^2 + mgh_{i+1} = \frac{1}{2}ms_i^2 + mgh_i + \eta f_i - dC_D s_i^2, \quad i = 1, \dots, N \\ & && \sum_{i=0}^N f_i + PT/\eta \leq F \\ & && f_i \geq 0, \quad i = 0, \dots, N \\ & && \eta f_0 = \frac{1}{2}ms_1^2, \end{aligned}$$

with variables f_i and s_i . The domain of the objective gives the implicit constraint $s_i > 0$, $i = 1, \dots, N$. The objective is convex, but unfortunately, the equality constraints are not linear because of the s_i^2 terms. So we will try introducing the new variables $z_i = s_i^2$. Since $s_i \geq 0$, we can recover speed from the new variables using $s_i = \sqrt{z_i}$. Note that z_i is proportional to the kinetic energy; therefore this change of variables can be thought of as solving the problem in terms of kinetic energy instead of speed. Our equality constraints now become

$$\frac{1}{2}mz_{i+1} + mgh_{i+1} = \frac{1}{2}mz_i + mgh_i + \eta f_i - dC_D z_i, \quad i = 1, \dots, N$$

and $\eta f_0 = \frac{1}{2}mz_1$, which are linear equations in the variables z_i and f_i . We now return to our objective which, under this change of variables, becomes

$$T = \sum_{i=1}^N \frac{d}{s_i} = \sum_{i=1}^N dz_i^{-1/2},$$

which is convex since $z_i^{-1/2}$ is convex. Having shown that T is convex in our new variables, it is clear that the fuel capacity constraint

$$\sum_{i=0}^N f_i + PT/\eta = \sum_{i=0}^N f_i + P/\eta \sum_{i=1}^N dz_i^{-1/2} \leq F$$

is a convex constraint. We can therefore find f_0, \dots, f_N by solving the convex optimization problem

$$\begin{aligned} & \text{minimize} && \sum_{i=1}^N dz_i^{-1/2} \\ & \text{subject to} && \frac{1}{2}mz_{i+1} + mgh_{i+1} = \frac{1}{2}mz_i + mgh_i + \eta f_i - dC_D z_i, \quad i = 1, \dots, N \\ & && \sum_{i=0}^N f_i + P/\eta \sum_{i=1}^N d(z_i)^{-1/2} \leq F \\ & && f_i \geq 0, \quad i = 0, \dots, N \\ & && \eta f_0 = \frac{1}{2}mz_1, \end{aligned}$$

with variables f_0, \dots, f_N and z_1, \dots, z_N . Then we recover s_i using $s_i = \sqrt{z_i}$.

(b) The following code solves the problem:

```
% Minimum time speed profile along a road.
min_time_speed_data;

% minimum time
cvx_begin
variables z(N+1) f(N+1)
minimize sum(d*inv_pos(sqrt(z(1:N))))
subject to
.5*m*z(2:N+1)+m*g*h(2:N+1) ==...
.5*m*z(1:N)+m*g*h(1:N)+eta*f(2:N+1)-d*C_D*z(1:N)
sum(f)+P/eta*sum(d*inv_pos(sqrt(z(1:N)))) <= F
f >= 0
eta*f(1) == .5*m*z(1)
cvx_end

T = cvx_optval

% constant fuel burn
cvx_solver sdpt3
% sedumi fails, but only on this 1 part of 1 problem
% from the entire exam, and only if you replaced
% the vector f, by the variable fc as below
cvx_begin
variables zc(N+1) fc
minimize sum(d*inv_pos(sqrt(zc(1:N))))
subject to
.5*m*zc(2:N+1)+m*g*h(2:N+1) ==...
.5*m*zc(1:N)+m*g*h(1:N)+eta*fc-d*C_D*zc(1:N)
(N+1)*fc+P/eta*sum(d*inv_pos(sqrt(zc(1:N)))) <= F
fc >= 0
eta*fc == .5*m*zc(1)
cvx_end

T_unif = cvx_optval

figure
subplot(3,1,1)
plot((0:N)*d,h);
ylabel('height');
subplot(3,1,2)
stairs((0:N)*d,sqrt(z),'b');
hold on
stairs((0:N)*d,sqrt(zc),'--r');
ylabel('speed')
```

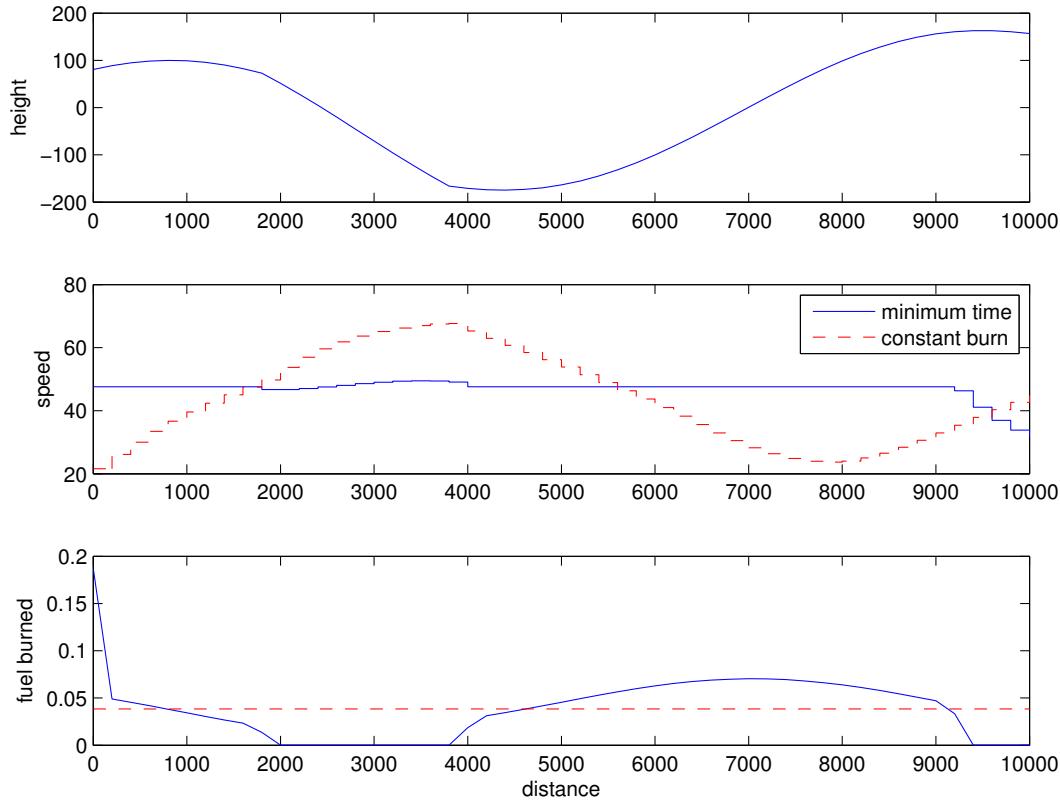
```

legend('minimum time', 'constant burn')
subplot(3,1,3)
plot((0:N)*d,f,'b');
hold on
plot((0:N)*d,fc*ones(N+1,1),'--r')
xlabel('distance')
ylabel('fuel burned')

print -depsc min_time_speed;

```

We get $T^* = 213.26$ and $T^{\text{unif}} = 258.48$. Note that you can find T^{unif} by replacing the f_i 's by a single parameter f_c as we did, or by constraining all of the f_i 's to be equal. The plot below shows the speed and fuel burn profiles for the two cases.



We see that the optimal fuel burn attempts to keep a constant velocity, except near the end of the trajectory, where it coasts to the finish line. The uniform fuel burn wastes fuel (and therefore loses time) by burning fuel on the downhill parts, leading to a large speed, and therefore, large loss due to drag.

14.12 Least-cost road grading. A road is to be built along a given path. We must choose the height of the roadbed (say, above sea level) along the path, minimizing the total cost of grading, subject to some constraints. The cost of grading (*i.e.*, moving earth to change the height of the roadbed from the existing elevation) depends on the difference in height between the roadbed and the existing

elevation. When the roadbed is below the existing elevation it is called a *cut*; when it is above it is called a *fill*. Each of these incurs engineering costs; for example, fill is created in a series of *lifts*, each of which involves dumping just a few inches of soil and then compacting it. Deeper cuts and higher fills require more work to be done on the road shoulders, and possibly, the addition of reinforced concrete structures to stabilize the earthwork. This explains why the marginal cost of cuts and fills increases with their depth/height.

We will work with a discrete model, specifying the road height as $h_i, i = 1, \dots, n$, at points equally spaced a distance d from each other along the given path. These are the variables to be chosen. (The heights h_1, \dots, h_n are called a *grading plan*.) We are given $e_i, i = 1, \dots, n$, the existing elevation, at the points. The grading cost is

$$C = \sum_{i=1}^n \left(\phi^{\text{fill}}((h_i - e_i)_+) + \phi^{\text{cut}}((e_i - h_i)_+) \right),$$

where ϕ^{fill} and ϕ^{cut} are the fill and cut cost functions, respectively, and $(a)_+ = \max\{a, 0\}$. The fill and cut functions are increasing and convex. The goal is to minimize the grading cost C .

The road height is constrained by given limits on the first, second, and third derivatives:

$$\begin{aligned} |h_{i+1} - h_i|/d &\leq D^{(1)}, \quad i = 1, \dots, n-1 \\ |h_{i+1} - 2h_i + h_{i-1}|/d^2 &\leq D^{(2)}, \quad i = 2, \dots, n-1 \\ |h_{i+1} - 3h_i + 3h_{i-1} - h_{i-2}|/d^3 &\leq D^{(3)}, \quad i = 3, \dots, n-1, \end{aligned}$$

where $D^{(1)}$ is the maximum allowable road slope, $D^{(2)}$ is the maximum allowable curvature, and $D^{(3)}$ is the maximum allowable third derivative.

- (a) Explain how to find the optimal grading plan.
- (b) Find the optimal grading plan for the problem with data given in `road_grading_data.m`, and fill and cut cost functions

$$\phi^{\text{fill}}(u) = 2(u)_+^2 + 30(u)_+, \quad \phi^{\text{cut}} = 12(u)_+^2 + (u)_+.$$

Plot $h_i - e_i$ for the optimal grading plan and report the associated cost.

- (c) Suppose the optimal grading problem with $n = 1000$ can be solved on a particular machine (say, with one, or just a few, cores) in around one second. Assuming the author of the software took EE364a, about how long will it take to solve the optimal grading problem with $n = 10000$? Give a very brief justification of your answer, no more than a few sentences.

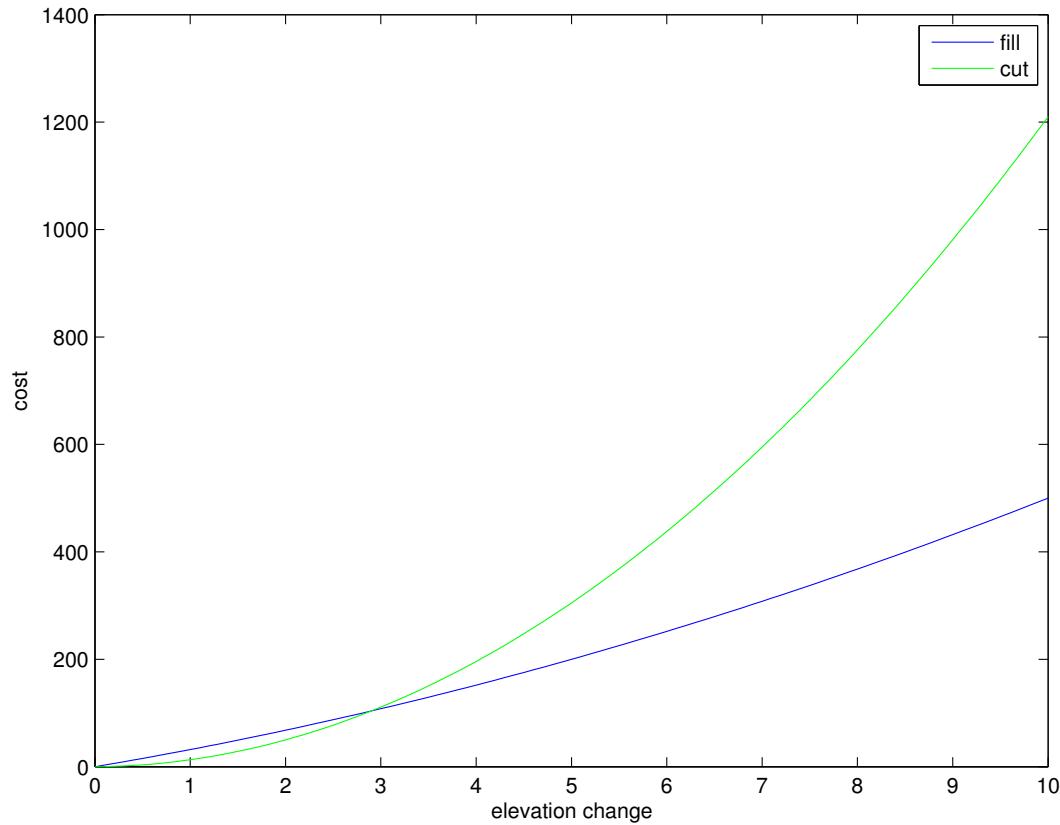
Solution.

- (a) Fortunately, this problem is convex; C is convex since ϕ^{fill} and ϕ^{cut} are convex and increasing and \max is convex. Therefore we simply solve the problem

$$\begin{aligned} &\text{minimize} && C \\ &\text{subject to} && |h_{i+1} - h_i|/d \leq D^{(1)}, \quad i = 1, \dots, n-1 \\ &&& |h_{i+1} - 2h_i + h_{i-1}|/d^2 \leq D^{(2)}, \quad i = 2, \dots, n-1 \\ &&& |h_{i+1} - 3h_i + 3h_{i-1} - h_{i-2}|/d^3 \leq D^{(3)}, \quad i = 3, \dots, n-1, \end{aligned}$$

with optimization variables h_i .

(b) For this problem instance we have cost functions shown below.



The following code solves the problem:

```
% Least-cost road grading.
road_grading_data;

cvx_begin
variables h(n);
minimize sum(alpha_fill*square_pos(h-e)+beta_fill*pos(h-e)+...
alpha_cut*square_pos(e-h)+beta_cut*pos(e-h))
subject to
abs(h(2:n)-h(1:n-1)) <= D1*d
abs(h(3:n)-2*h(2:n-1)+h(1:n-2)) <= D2*d^2
abs(-h(4:n)+3*h(3:n-1)-3*h(2:n-2)+h(1:n-3)) <= D3*d^3
cvx_end

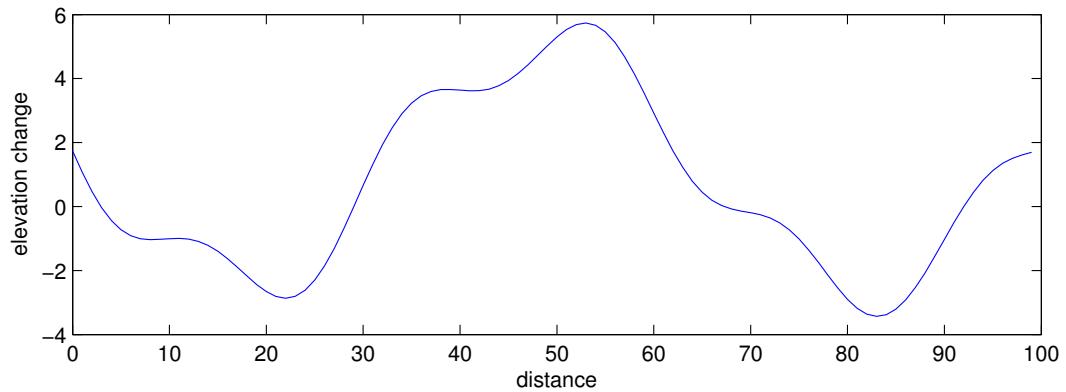
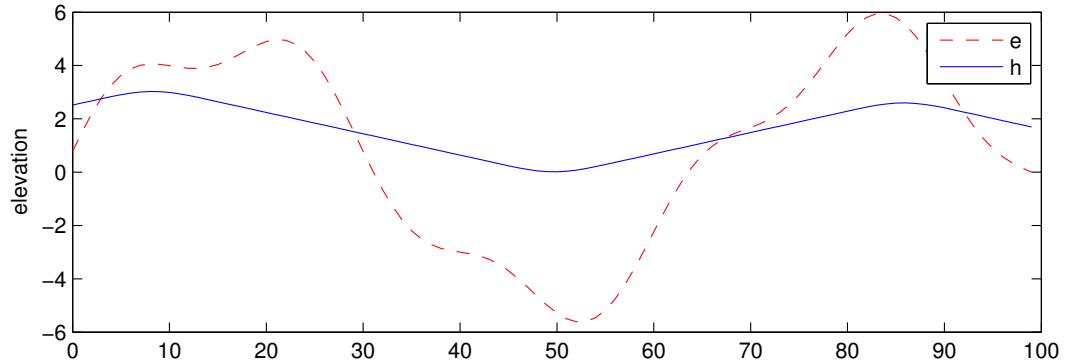
figure
subplot(2,1,1)
plot((0:n-1)*d,e,'--r');
ylabel('elevation');
hold on
```

```

plot((0:n-1)*d,h, 'b')
legend('e', 'h');
subplot(2,1,2)
plot((0:n-1)*d,h-e)
ylabel('elevation change')
xlabel('distance')

print -depsc road_grading;

```



We find that the optimal cost is 7562.82.

- (c) Using our knowledge from EE364a we see immediately that this problem is well structured. Our objective is separable and our constraints have bandwidth of at most 4. For each Newton step we have to solve a banded system, which we know we can do in $\mathcal{O}(nk^2)$ flops, where $k = 4$ is the bandwidth. We can, therefore, take a Newton step in $\mathcal{O}(n)$ flops. We have seen that the number of iterations required to solve a problem with an interior point method is practically independent of problem size. Thus, if an optimal grading problem with $n = 1000$ can be solved in about 1 second, we can solve a problem with $n = 10000$ in approximately 10 seconds.

To see how this banded system arises, using EE364a knowledge you might solve this problem using an interior point barrier method. For a given t you are now solving the unconstrained

minimization problem

$$\begin{aligned} \text{minimize} \quad & tC - \sum_{i=1}^{n-1} \left(\log(D^{(1)}d - h_{i+1} + h_i) + \log(D^{(1)}d + h_{i+1} - h_i) \right) - \\ & \sum_{i=2}^{n-1} \left(\log(D^{(2)}d^2 - h_{i+1} + 2h_i - h_{i-1}) \right) - \\ & \sum_{i=2}^{n-1} \left(\log(D^{(2)}d^2 + h_{i+1} - 2h_i + h_{i-1}) \right) - \\ & \sum_{i=3}^{n-1} \left(\log(D^{(3)}d^3 - h_{i+1} + 3h_i - 3h_{i-1} + h_{i-2}) \right) - \\ & \sum_{i=3}^{n-1} \left(\log(D^{(3)}d^3 + h_{i+1} - 3h_i + 3h_{i-1} - h_{i-2}) \right). \end{aligned}$$

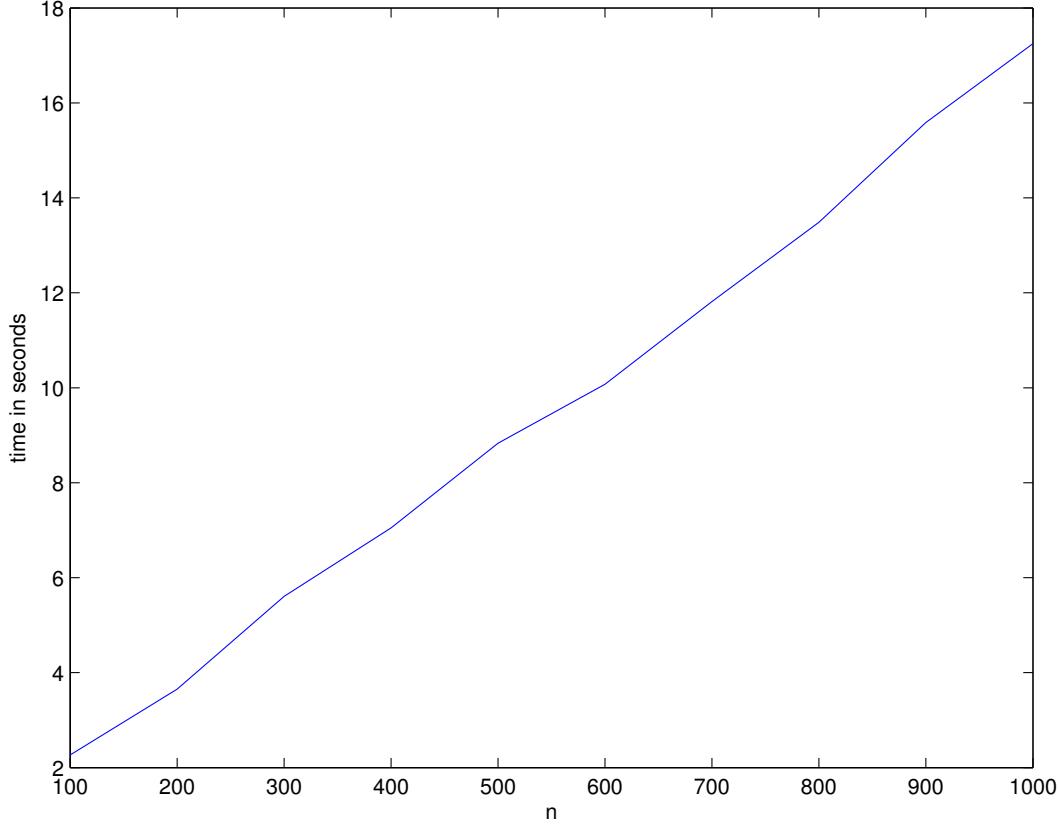
We can represent the Hessian of this function as the sum of 4 matrices: the Hessian relating to C and the Hessians relating to constraints with $D^{(1)}$, $D^{(2)}$, and $D^{(3)}$. Since C is separable in the optimization variables h_i , its Hessian is diagonal. The constraints with $D^{(1)}$ will contribute a matrix of bandwidth 2 to the Hessian, as there is only coupling between h_i and h_{i+1} . Similarly the terms related to $D^{(2)}$ will contribute a matrix of bandwidth 3, and the terms related to $D^{(3)}$ will contribute a matrix of bandwidth 4. Therefore at each Newton step, as already stated we must solve a system with bandwidth 4. As the problem size increases, the bandwidth remains constant, so we expect the problem to scale linearly in n until we run into system related issues such as memory limitations. The code below generates problem instances from $n = 100$ to $n = 1000$:

```
% Least-cost road grading timing for different problem sizes.
road_grading_data

N = 10;
times = zeros(N,1);
e2 = [];
for i = 1:N
    e2 = [e2; e];
    tic
    cvx_begin
    variables h(i*n);
    minimize sum(alpha_fill*square_pos(h-e2)+...
        beta_fill*pos(h-e2)+...
        alpha_cut*square_pos(e2-h)+beta_cut*pos(e2-h))
    subject to
    abs(h(2:end)-h(1:end-1)) <= D1*d
    abs(h(3:end)-2*h(2:end-1)+h(1:end-2)) <= D2*d^2
    abs(-h(4:end)+3*h(3:end-1)-3*h(2:end-2)+h(1:end-3)) <= D3*d^3
    cvx_end
    times(i) = toc;
end

figure
plot((1:N)*n,times);
xlabel('n');
ylabel('time in seconds');
```

```
print -depsc road_grading_timing.eps
```



We see that in SDPT3 the timing is indeed linear with problem size (with an offset for the CVX overhead).

14.13 Lightest structure that resists a set of loads. We consider a mechanical structure in 2D (for simplicity) which consists of a set of m nodes, with known positions $p_1, \dots, p_m \in \mathbf{R}^2$, connected by a set of n bars (also called struts or elements), with cross-sectional areas $a_1, \dots, a_n \in \mathbf{R}_+$, and internal tensions $t_1, \dots, t_n \in \mathbf{R}$.

Bar j is connected between nodes r_j and s_j . (The indices r_1, \dots, r_n and s_1, \dots, s_n give the structure topology.) The length of bar j is $L_j = \|p_{r_j} - p_{s_j}\|_2$, and the total volume of the bars is $V = \sum_{j=1}^n a_j L_j$. (The total weight is proportional to the total volume.)

Bar j applies a force $(t_j/L_j)(p_{r_j} - p_{s_j}) \in \mathbf{R}^2$ to node s_j and the negative of this force to node r_j . Thus, positive tension in a bar pulls its two adjacent nodes towards each other; negative tension (also called compression) pushes them apart. The ratio of the tension in a bar to its cross-sectional area is limited by its yield strength, which is symmetric in tension and compression: $|t_j| \leq \sigma a_j$, where $\sigma > 0$ is a known constant that depends on the material.

The nodes are divided into two groups: free and fixed. We will take nodes $1, \dots, k$ to be free, and nodes $k + 1, \dots, m$ to be fixed. Roughly speaking, the fixed nodes are firmly attached to the ground, or a rigid structure connected to the ground; the free ones are not.

A *loading* consists of a set of external forces, $f_1, \dots, f_k \in \mathbf{R}^2$ applied to the free nodes. Each free node must be in equilibrium, which means that the sum of the forces applied to it by the bars and the external force is zero. The structure can *resist* a loading (without collapsing) if there exists a set of bar tensions that satisfy the tension bounds and force equilibrium constraints. (For those with knowledge of statics, these conditions correspond to a structure made entirely with pin joints.)

Finally, we get to the problem. You are given a set of M loadings, *i.e.*, $f_1^{(i)}, \dots, f_k^{(i)} \in \mathbf{R}^2$, $i = 1, \dots, M$. The goal is to find the bar cross-sectional areas that minimize the structure volume V while resisting all of the given loadings. (Thus, you are to find *one* set of bar cross-sectional areas, and M sets of tensions.) Using the problem data provided in `lightest_struct_data.m`, report V^* and V^{unif} , the smallest feasible structure volume when all bars have the same cross-sectional area. The node positions are given as a $2 \times m$ matrix P , and the loadings as a $2 \times k \times M$ array F . Use the code included in the data file to visualize the structure with the bar cross-sectional areas that you find, and provide the plot in your solution.

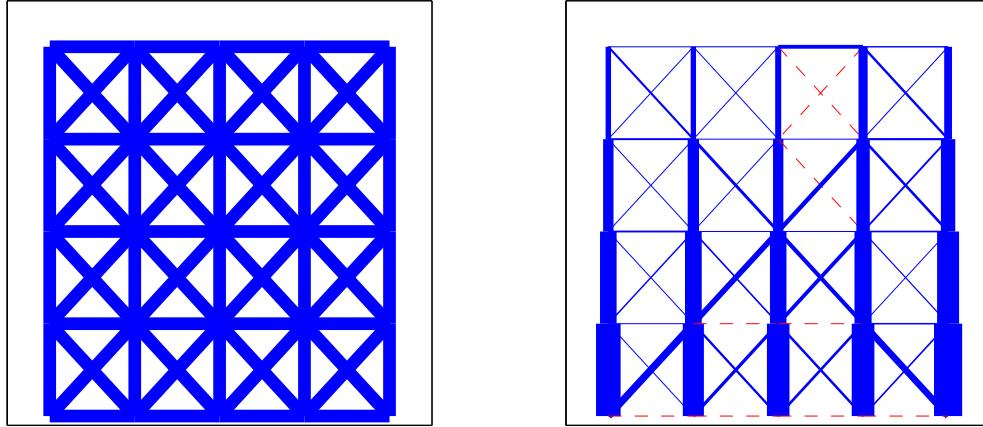
Hint. You might find the graph incidence matrix $A \in \mathbf{R}^{m \times n}$ useful. It is defined as

$$A_{ij} = \begin{cases} +1 & i = r_j \\ -1 & i = s_j \\ 0 & \text{otherwise.} \end{cases}$$

Remark. You could reasonably ask, ‘Does a mechanical structure really solve a convex optimization problem to determine whether it should collapse?’ It sounds odd, but the answer is, yes it does.

Solution. We can use the graph incidence matrix A to express the force exerted by the bars on each node. For a loading f_1, \dots, f_k , define $G \in \mathbf{R}^{2 \times m}$ such that g_i , the i th column of G , is the sum of the forces from each of the bars connected to node i . Then $G = -P A D A^T$, where $P \in \mathbf{R}^{2 \times m}$ is a matrix whose i th column is p_i , and $D \in \mathbf{R}^{n \times n}$ is a diagonal matrix such that $D_{jj} = t_j/L_j$. (To see this, note that the i th column of PAD is $(t_i/L_i)(p_{r_i} - p_{s_i})$, the force that bar i applies to the adjacent node r_i .) The force equilibrium constraints for the loading can then be written as $g_i + f_i = 0$, for $i = 1, \dots, k$. A single set of bar cross-sectional areas must satisfy the equilibrium constraints for each of the M loadings. The remaining constraints are easily formulated.

We find that $V^* = 188.55$, and $V^{\text{unif}} = 492.00$.



The following code solves the problem.

```

% lightest structure that resists a set of loads
lightest_struct_data;

% form incidence matrix
A = zeros(m, n);
for i = 1:n
    A(r(i), i) = +1;
    A(s(i), i) = -1;
end

L = norms(P*A)';

% solve with all bars having same cross-sectional area
cvx_begin quiet
    variables a(n) t(n, M)
    expression G(2, m, M) % force due to bars
    minimize (a'*L)
    subject to
        a == mean(a);
        for i = 1:M
            abs(t(:, i)) <= sigma.*a;
            G(:, :, i) = -P*A*diag(t(:, i)./L)*A';
            G(:, 1:k, i) + F(:, :, i) == 0;
        end
    cvx_end
    fprintf('V^unif = %f\n', cvx_optval);

% plot
clf;
subplot(1,2,1); hold on;
for i = 1:n
    p1 = r(i); p2 = s(i);
    plt_str = 'b-';
    if a(i) < 0.001
        plt_str = 'r--';
    end
    plot([P(1, p1) P(1, p2)], [P(2, p1) P(2, p2)], ...
        plt_str, 'LineWidth', a(i));
end
axis([-0.5 N-0.5 -0.1 N-0.5]); axis square; box on;
set(gca, 'xtick', [], 'ytick', []);
hold off;

% solve with bars having different cross-sectional areas
cvx_begin quiet

```

```

variables a(n) t(n, M)
expression G(2, m, M) % force due to bars
minimize (a'*L)
subject to
    for i = 1:M
        abs(t(:, i)) <= sigma.*a;
        G(:, :, i) = -P*A*diag(t(:, i)./L)*A';
        G(:, 1:k, i) + F(:, :, i) == 0;
    end
cvx_end
fprintf('V^star = %f\n', cvx_optval);

% plot
subplot(1,2,2); hold on;
for i = 1:n
    p1 = r(i); p2 = s(i);
    plt_str = 'b-';
    width = a(i);
    if a(i) < 0.001
        plt_str = 'r--';
        width = 1;
    end
    plot([P(1, p1) P(1, p2)], [P(2, p1) P(2, p2)], ...
        plt_str, 'LineWidth', width);
end
axis([-0.5 N-0.5 -0.1 N-0.5]); axis square; box on;
set(gca, 'xtick', [], 'ytick', []);
hold off;

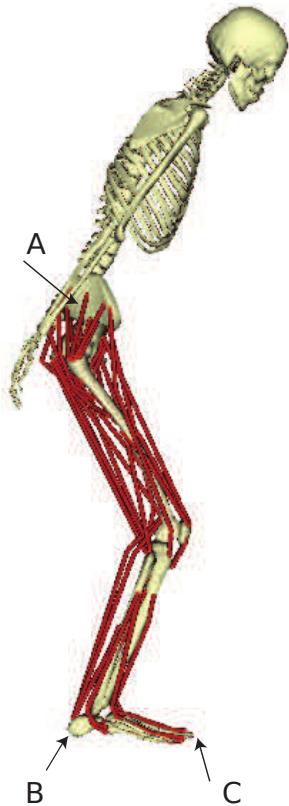
print -depsc lightest_struct.eps;

```

14.14 Maintaining static balance. In this problem we study a human's ability to maintain balance against an applied external force. We will use a planar (two-dimensional) model to characterize the set of push forces a human can sustain before he or she is unable to maintain balance. We model the human as a linkage of 4 body segments, which we consider to be rigid bodies: the foot, lower leg, upper leg, and pelvis (into which we lump the upper body). The pose is given by the joint angles, but this won't matter in this problem, since we consider a fixed pose. A set of 40 muscles act on the body segments; each of these develops a (scalar) tension t_i that satisfies $0 \leq t_i \leq T_i^{\max}$, where T_i^{\max} is the maximum possible tension for muscle i . (The maximum muscle tensions depend on the pose, and the person, but here they are known constants.) An external pushing force $f^{\text{push}} \in \mathbf{R}^2$ acts on the pelvis. Two (ground contact) forces act on the foot: $f^{\text{heel}} \in \mathbf{R}^2$ and $f^{\text{toe}} \in \mathbf{R}^2$. (These are shown at right.) These must satisfy

$$|f_1^{\text{heel}}| \leq \mu f_2^{\text{heel}}, \quad |f_1^{\text{toe}}| \leq \mu f_2^{\text{toe}},$$

where $\mu > 0$ is the coefficient of friction of the ground. There are also joint forces that act at the joints between the body segments, and gravity forces for each body segment, but we won't need them explicitly in this problem.



To maintain balance, the net force and torque on each body segment must be satisfied. These equations can be written out from the geometry of the body (*e.g.*, attachment points for the muscles) and the pose. They can be reduced to a set of 6 linear equations:

$$A^{\text{musc}}t + A^{\text{toe}}f^{\text{toe}} + A^{\text{heel}}f^{\text{heel}} + A^{\text{push}}f^{\text{push}} = b,$$

where $t \in \mathbf{R}^{40}$ is the vector of muscle tensions, and A^{musc} , A^{toe} , A^{heel} , and A^{push} are known matrices and $b \in \mathbf{R}^6$ is a known vector. These data depend on the pose, body weight and dimensions, and muscle lines of action. Fortunately for you, our biomechanics expert Apoorva has worked them out; you will find them in `static_balance_data.*` (along with T^{\max} and μ).

We say that the push force f^{push} can be *resisted* if there exist muscle tensions and ground contact forces that satisfy the constraints above. (This raises a philosophical question: Does a person solve an optimization to decide whether he or she should lose their balance? In any case, this approach makes good predictions.)

Find $\mathcal{F}^{\text{res}} \subset \mathbf{R}^2$, the set of push forces that can be resisted. Plot it as a shaded region.

Hints. Show that \mathcal{F}^{res} is a convex set. For the given data, $0 \in \mathcal{F}^{\text{res}}$. Then for $\theta = 1^\circ, 2^\circ, \dots, 360^\circ$, determine the maximum push force, applied in the direction θ , that can be resisted. To make a filled region on a plot, you can use the command `fill()` in Matlab. For Python and Julia, `fill()` is also available through PyPlot. In Julia, make sure to use the ECOS solver with `solver = ECOSolver(verbose=false)`.

Remark. A person can resist a much larger force applied to the hip than you might think.

Solution. The set of vectors $(t, f^{\text{toe}}, f^{\text{heel}}, f^{\text{push}})$ which satisfy the constraints

$$\begin{aligned} A^{\text{musc}}t + A^{\text{toe}}f^{\text{toe}} + A^{\text{heel}}f^{\text{heel}} + A^{\text{push}}f^{\text{push}} &= b \\ |f_1^{\text{toe}}| &\leq \mu f_2^{\text{toe}} \\ |f_1^{\text{heel}}| &\leq \mu f_2^{\text{heel}} \\ 0 \preceq t \preceq T^{\max}, \end{aligned}$$

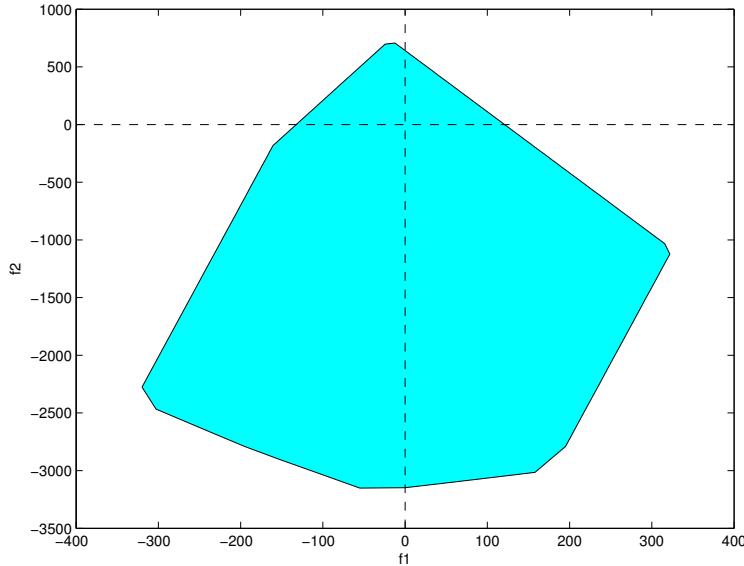
forms a convex set, a polyhedron. It follows the set of push forces that can be resisted, \mathcal{F}^{res} , is a convex set, since it is the projection of this set onto the variable f^{push} . In fact, \mathcal{F}^{res} is also a polyhedron.

To find the maximum push force for a given direction θ , we solve the LP

$$\begin{aligned} \text{maximize } & z \\ \text{subject to } & A^{\text{musc}}t + A^{\text{toe}}f^{\text{toe}} + A^{\text{heel}}f^{\text{heel}} + A^{\text{push}}z(\cos \theta, \sin \theta) = b \\ & |f_1^{\text{toe}}| \leq \mu f_2^{\text{toe}} \\ & |f_1^{\text{heel}}| \leq \mu f_2^{\text{heel}} \\ & 0 \preceq t \preceq T^{\max}. \end{aligned}$$

with variables $t, f^{\text{toe}}, f^{\text{heel}}$ and $z \in \mathbf{R}$. We solve this problem for a number of values of θ , and for each one we record the value $f^{\text{push}} = z^*(\cos \theta, \sin \theta)$, which is on the boundary of \mathcal{F}^{res} . We use these values to fill out the set when plotting.

The set \mathcal{F}^{res} for the given data is shown below.



We see some interesting things from the plot. One is that it's much easier to make someone lose their static balance by pulling them up, instead of pushing them down. Another interpretation (maybe more positive) is that a 75 kg person can maintain this posture even with a load of 3150 Newtons (321 kg, 708 lbs) attached to their hips. This is over 4 times body weight! And it's a little easier to make them lose their balance pushing them forward, compared to pulling them backward. This analysis does not take into account other factors such as the maximum compressive load that

can be supported by bones and joints. That said, the human body can produce and withstand extremely high forces. For reference, in running, the force in the Achilles tendon can be 6 to 8 times body weight and compressive forces in the lower leg can be 10 to 14 times body weight.

The following Matlab code solves the problem.

```
static_balance_data

theta_push = pi/180.* (0:1:360);
f_push_max = zeros(size(theta_push));
t_muscle = zeros(n_musc, length(theta_push));

for i = 1:length(theta_push)
    theta = theta_push(i);
    cvx_begin
        cvx_quiet(true)
        variable f_push;
        variable t(40,1);
        variables f_toe(2,1) f_heel(2,1);

        maximize(f_push);

        A_musc*t + A_heel*f_heel + A_toe*f_toe + ...
            A_push*f_push*[cos(theta); sin(theta)] == b;
        abs(f_toe(1)) <= mu*f_toe(2);
        abs(f_heel(1)) <= mu*f_heel(2);
        0 <= t;
        t <= T_max;
    cvx_end
    f_push_max(i) = f_push;
    t_muscle(:,i) = t;
end

% plot results
figure
fill(f_push_max.*cos(theta_push), f_push_max.*sin(theta_push), 'c'), hold on
xlabel('f^{push}_1 (Newtons)')
ylabel('f^{push}_2 (Newtons)')
plot([-400,400], [0,0], 'k--'), hold on
plot([0,0], [-3500,1000], 'k--')

print -depsc static_balance_fres_mat
```

The following Python code solves the problem.

```
# solution to static balance problem
import numpy as np
```

```

import cvxpy as cvx
import matplotlib.pyplot as plt
import matplotlib

from static_balance_data import *

theta_push = np.pi/180. * np.arange(360)
f_push_max = np.zeros(len(theta_push));

for i in range(len(theta_push)):
    theta = theta_push[i];
    f_push = cvx.Variable(1)
    t = cvx.Variable(40,1)
    f_toe = cvx.Variable(2,1)
    f_heel = cvx.Variable(2,1)

    constr = [A_musc*t + A_heel*f_heel + A_toe*f_toe \
              + A_push*f_push*np.array([np.cos(theta), np.sin(theta)]) == b,
              cvx.abs(f_toe[0]) <= mu*f_toe[1],
              cvx.abs(f_heel[0]) <= mu*f_heel[1],
              0 <= t,
              t <= T_max]

    p = cvx.Problem(cvx.Maximize(f_push), constr)

    p.solve(verbose = False)

    f_push_max[i] = f_push.value

# plot results
plt.figure(1)
plt.fill(f_push_max*np.cos(theta_push), f_push_max*np.sin(theta_push), 'c')
plt.plot(np.array([-400,400]), np.array([0,0]), 'k--')
plt.plot(np.array([0,0]), np.array([-3500,1000]), 'k--')
plt.xlabel('$f^{\mathrm{push}}_1$ (Newtons)')
plt.ylabel('$f^{\mathrm{push}}_2$ (Newtons)')

plt.savefig('static_balance_fres_py.eps')

plt.show()

```

The following Julia code solves the problem.

```
include("static_balance_data.jl");
```

```

using Convex, ECOS, PyPlot
solver = ECOSolver(verbose=false);

theta_push = pi/180 * [0:359];
f_push_max = zeros(length(theta_push));

for i = 1:length(theta_push)
    theta = theta_push[i];
    f_push = Variable(1);
    t = Variable(40);
    f_toe = Variable(2);
    f_heel = Variable(2)

    constraints = A_musc*t + A_heel*f_heel + A_toe*f_toe +
                  A_push*f_push*[cos(theta) sin(theta)]' == b;
    constraints += abs(f_toe[1]) <= mu*f_toe[2];
    constraints += abs(f_heel[1]) <= mu*f_heel[2];
    constraints += 0 <= t;
    constraints += t <= T_max;

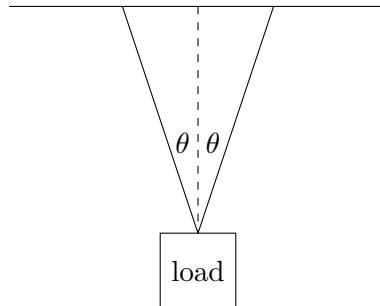
    prob = maximize(f_push, constraints);
    solve!(prob, solver)

    f_push_max[i] = prob.optval
end

# plot results
fill(f_push_max.*cos(theta_push), f_push_max.*sin(theta_push),"c")
plot([-400,400], [0,0], "k--")
plot([0,0], [-3500,1000], "k--")
xlabel("\$f^{\mathrm{push}}_1\$ (Newtons)")
ylabel("\$f^{\mathrm{push}}_2\$ (Newtons)")
savefig("static_balance_fres_jl.eps")

```

- 14.15** *Minimum time maneuver for a crane.* A crane manipulates a load with mass $m > 0$ in two dimensions using two cables attached to the load. The cables maintain angles $\pm\theta$ with respect to vertical, as shown below.



The (scalar) tensions T^{left} and T^{right} in the two cables are independently controllable, from 0 up to a given maximum tension T^{\max} . The total force on the load is

$$F = T^{\text{left}} \begin{bmatrix} -\sin \theta \\ \cos \theta \end{bmatrix} + T^{\text{right}} \begin{bmatrix} \sin \theta \\ \cos \theta \end{bmatrix} + mg,$$

where $g = (0, -9.8)$ is the acceleration due to gravity. The acceleration of the load is then F/m .

We approximate the motion of the load using

$$p_{i+1} = p_i + hv_i, \quad v_{i+1} = v_i + (h/m)F_i, \quad i = 1, 2, \dots,$$

where $p_i \in \mathbf{R}^2$ is the position of the load, $v_i \in \mathbf{R}^2$ is the velocity of the load, and $F_i \in \mathbf{R}^2$ is the force on the load, at time $t = ih$. Here $h > 0$ is a small (given) time step.

The goal is to move the load, which is initially at rest at position p^{init} to the position p^{des} , also at rest, in minimum time. In other words, we seek the smallest k for which

$$p_1 = p^{\text{init}}, \quad p_k = p^{\text{des}}, \quad v_1 = v_k = (0, 0)$$

is possible, subject to the constraints described above.

- (a) Explain how to solve this problem using convex (or quasiconvex) optimization.
- (b) Carry out the method of part (a) for the problem instance with

$$m = 0.1, \quad \theta = 15^\circ, \quad T^{\max} = 2, \quad p^{\text{init}} = (0, 0), \quad p^{\text{des}} = (10, 2),$$

with time step $h = 0.1$. Report the minimum time k^* . Plot the tensions versus time, and the load trajectory, *i.e.*, the points p_1, \dots, p_k in \mathbf{R}^2 . Does the load move along the line segment between p^{init} and p^{des} (*i.e.*, the shortest path from p^{init} and p^{des})? Comment briefly.

Solution.

- (a) The problem as stated is quasiconvex: To see if $k^* \leq k$, we simply check if there exists a set of variables that satisfy the constraints, together with $p_k = p^{\text{des}}$, $v_k = 0$.

For a given value for k , we can solve a convex feasibility problem (in fact, an LP) to determine if such a trajectory exists. Let $T \in \mathbf{R}^{2 \times k-1}$ be a matrix of the tensions, so that T_{1i} , T_{2i} are T^{left} , T^{right} at time ih , respectively. Then the force applied to the load at time ih be $F_i = MT_i + mg$ where

$$M = \begin{bmatrix} -\sin \theta & \sin \theta \\ \cos \theta & \cos \theta \end{bmatrix}.$$

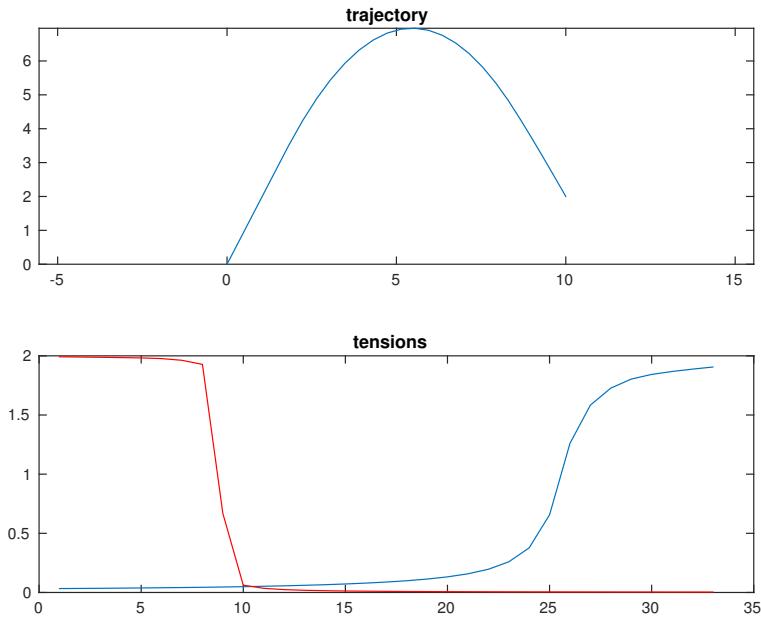
To find a feasible trajectory, we solve the LP

$$\begin{aligned} & \text{minimize} && 0 \\ & \text{subject to} && 0 \preceq T \preceq T^{\max}, \\ & && v_{i+1} = v_i + (h/m)F_i, \quad i = 1, \dots, k-1, \\ & && p_{i+1} = p_i + hv_i, \quad i = 1, \dots, k-1, \\ & && p_1 = p^{\text{init}}, \quad p_k = p^{\text{des}}, \quad v_1 = v_k = 0. \end{aligned}$$

We can then find the minimum time by finding the smallest k for which the above problem is feasible. This can be done by bisection, or by simply increasing k until the problem becomes feasible.

- (b) We find that $k^* = 34$, corresponding to $t = 3.4$ seconds. From the trajectory plot, we see that the load does not travel along the line between the initial and final positions. Since the load must cross a large horizontal distance, we maximize the horizontal force which is accomplished by setting the tension in the right cable to T^{\max} . As the tension produces a force along the line of the cable, the load rises up in addition to accelerating in the horizontal direction.

The code is shown below in Matlab.



```

clear;
% Angle of cables with respect to vertical
% For convenience, we put the coefficients in a matrix
theta = 15*pi/180;
M = [-sin(theta), sin(theta); cos(theta), cos(theta)];
T_max = 2; % Max tension that each cable can apply (kNewtons)
m = 0.1; % Mass of the load (metric tons)
g = [0;-9.8]; % Gravity (m/s^2)
p_init = [0;0]; % Init position (m)
p_des = [10;2]; % Desired position (m)
h = 0.1; % Simulation timestep (s)

T_feasible = 0;
p_feasible = 0;

%% Run the problem
lower = 10; % A lowerbound obtained by rough check (infeasible)
upper = 50; % A upperbound obtained by rough check (feasible)
while lower + 1 ~= upper
    k = floor((lower+upper)/2);
    disp(['checking: k=' num2str(k) ...

```

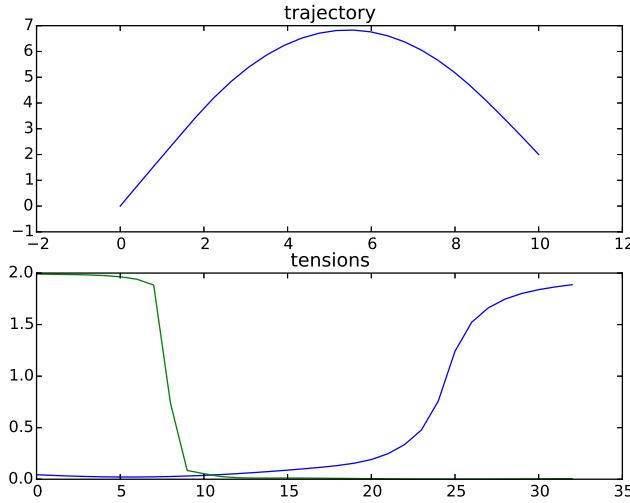
```

        ', lower=' num2str(lower) ...
        ', upper=' num2str(upper)]);
cvx_begin quiet
    variables T(2,k-1) p(2,k) v(2,k)
    F = M*T + m*repmat(g,1,k-1);
    minimize 0
    subject to
        p(:,1) == p_init; p(:,end) == p_des;
        v(:,1) == 0; v(:,end) == 0;
        0 <= T <= T_max;
        v(:,2:end) == v(:,1:end-1) + h/m*F;
        p(:,2:end) == p(:,1:end-1) + h*v(:,1:end-1);
cvx_end
if cvx_optval == 0
    upper = k;
    T_feasible = T;
    p_feasible = p;
else
    lower = k;
end
end
k = upper

%% Plotting
figure(1); clf;
subplot(2,1,1); plot(p(1,:),p(2,:)); axis equal;
title('trajectory');
subplot(2,1,2); plot(T_feasible(1,:)); hold on; plot(T_feasible(2,:),'r');
title('tensions');
print -depsc crane_no_constraint

```

For Python, the code is given below.



```

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

theta = 15*3.141592/180.0
M = np.matrix([[-np.sin(theta), np.sin(theta)],
               [np.cos(theta), np.cos(theta)]])
T_max = 2.0 # Max tension that each cable can apply (kNewtons)
m = 0.1 # Mass of the load (Metric tons)
g = np.matrix('0;-9.8') # Gravity (m/s^2)
p_init = np.matrix('0.0;0.0') # Init position (m)
p_des = np.matrix('10.0;2.0') # Desired position (m)
h = 0.1 # Simulation timestep (s)

T_feasible = 0
p_feasible = 0

# Run bisection
lower = 10 # Determined by rough check, infeasible
upper = 50 # Determined by rough check, feasible
while not lower + 1 == upper:
    k = int((upper+lower)/2)
    print('checking k=' + str(k) +
          ', lower=' + str(lower) +
          ', upper=' + str(upper))

    T = cvx.Variable(2,k-1)
    v = cvx.Variable(2,k)
    p = cvx.Variable(2,k)

```

```

F = M*T + m*np.tile(g,(1,k-1))

constraints = [0 <= T, T <= T_max]
constraints += [p[:,0] == p_init, p[:,k-1] == p_des]
constraints += [v[:,0] == 0, v[:,k-1] == 0]
constraints += [v[:,1:k] == v[:,0:k-1] + (h/m)*F]
constraints += [p[:,1:k] == p[:,0:k-1] + h*v[:,0:k-1]]

prob = cvx.Problem(cvx.Minimize(0),constraints)

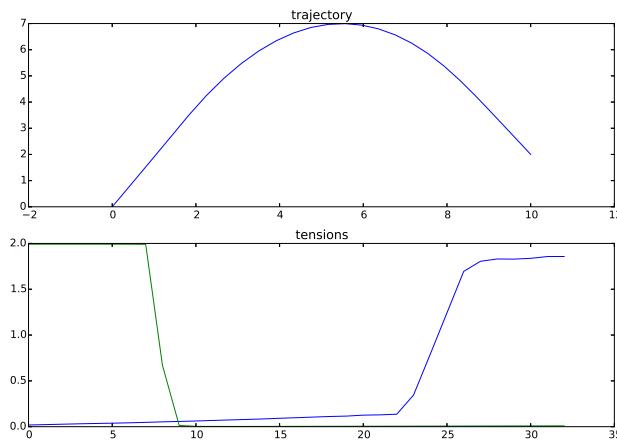
opt_val = prob.solve(solver=cvx.ECOS,verbose=False)

if opt_val == 0:
    upper = k
    T_feasible = T.value
    p_feasible = p.value
else: lower = k

k = upper;
print('minimum is ' + str(k))
plt.subplot(2,1,1)
plt.plot(p_feasible[0,:].T,p_feasible[1,:].T)
plt.title('trajectory')
plt.subplot(2,1,2)
plt.plot(T_feasible.T)
plt.title('tensions')
plt.savefig('crane_no_constraint.eps',format='ps')

```

The Julia code follows.



```
using Convex, SCS, PyPlot;
```

```

using ECOS;

theta = 15*3.141592/180;
M = [-sin(theta) sin(theta); cos(theta) cos(theta)];
T_max = 2.0; # Max tension that each cable can apply (kNewtons)
m = 0.1; # Mass of the load (Metric tons)
g = [0; -9.8]; # Gravity (m/s^2)
p_init = zeros(2,1); # Init position (m)
p_des = [10.0; 2]; # Desired position (m)
h = 0.1; # Simulation timestep (s)

T_feasible = 0;
p_feasible = 0;

lower = 10; # Determined by rough check (infeasible)
upper = 50; # Determined by rough check (feasible)
while lower + 1 != upper
    k = int((lower+upper)/2);
    println(string("checking k=",k," lower=",lower, " upper=", upper));
    T = Variable(2,k-1);
    v = Variable(2,k);
    p = Variable(2,k);

    F = M*T + m*repmat(g,1,k-1);

    constraints = [0 <= T, T <= T_max];
    constraints += [p[:,1] == p_init, p[:,end] == p_des];
    constraints += [v[:,1] == 0, v[:,end] == 0];
    constraints += v[:,2:end] == v[:,1:end-1] + (h/m)*F;
    constraints += p[:,2:end] == p[:,1:end-1] + h*v[:,1:end-1];

    prob = satisfy(constraints);

    #solve!(prob,SCSSolver(verbose=false));
    solve!(prob,ECOSSolver(maxit=20000,eps=1e-4,verbose=false));

    println(prob.status);
    if prob.status == :Optimal
        upper = k;
        T_feasible = T.value;
        p_feasible = p.value;
    elseif prob.status == :Infeasible
        lower = k;
    else
        println("solve failed!!!!");
    end
end

```

```
    end
end
k = upper;
println(string("minimum is ",k));

fig = figure("fig1",figsize=(12,8));
subplot(211);
plot(p_feasible[1,:]',p_feasible[2,:]');
title("trajectory")
subplot(212);
plot(T_feasible');
title("tensions")
savefig("crane_no_constraint.eps",format="ps");
```

15 Graphs and networks

- 15.1** A *hypergraph* with nodes $1, \dots, m$ is a set of nonempty subsets of $\{1, 2, \dots, m\}$, called *edges*. An ordinary graph is a special case in which the edges contain no more than two nodes.

We consider a hypergraph with m nodes and assume coordinate vectors $x_j \in \mathbf{R}^p$, $j = 1, \dots, m$, are associated with the nodes. Some nodes are fixed and their coordinate vectors x_j are given. The other nodes are free, and their coordinate vectors will be the optimization variables in the problem. The objective is to place the free nodes in such a way that some measure of the physical size of the nets is small.

As an example application, we can think of the nodes as modules in an integrated circuit, placed at positions $x_j \in \mathbf{R}^2$. Every edge is an interconnect network that carries a signal from one module to one or more other modules.

To define a measure of the size of a net, we store the vectors x_j as columns of a matrix $X \in \mathbf{R}^{p \times m}$. For each edge S in the hypergraph, we use X_S to denote the $p \times |S|$ submatrix of X with the columns associated with the nodes of S . We define

$$f_S(X) = \inf_y \|X_S - y\mathbf{1}^T\|. \quad (60)$$

as the *size* of the edge S , where $\|\cdot\|$ is a matrix norm, and $\mathbf{1}$ is a vector of ones of length $|S|$.

- (a) Show that the optimization problem

$$\text{minimize } \sum_{\text{edges } S} f_S(X)$$

is convex in the free node coordinates x_j .

- (b) The size $f_S(X)$ of a net S obviously depends on the norm used in the definition (60). We consider five norms.

- *Frobenius norm:*

$$\|X_S - y\mathbf{1}^T\|_F = \left(\sum_{j \in S} \sum_{i=1}^p (x_{ij} - y_i)^2 \right)^{1/2}.$$

- *Maximum Euclidean column norm:*

$$\|X_S - y\mathbf{1}^T\|_{2,1} = \max_{j \in S} \left(\sum_{i=1}^p (x_{ij} - y_i)^2 \right)^{1/2}.$$

- *Maximum column sum norm:*

$$\|X_S - y\mathbf{1}^T\|_{1,1} = \max_{j \in S} \sum_{i=1}^p |x_{ij} - y_i|.$$

- *Sum of absolute values norm:*

$$\|X_S - y\mathbf{1}^T\|_{\text{sav}} = \sum_{j \in S} \sum_{i=1}^p |x_{ij} - y_i|$$

- *Sum-row-max norm:*

$$\|X_s - y\mathbf{1}^T\|_{\text{srm}} = \sum_{i=1}^p \max_{j \in S} |x_{ij} - y_i|$$

For which of these norms does f_S have the following interpretations?

- (i) $f_S(X)$ is the radius of the smallest Euclidean ball that contains the nodes of S .
- (ii) $f_S(X)$ is (proportional to) the perimeter of the smallest rectangle that contains the nodes of S :

$$f_S(X) = \frac{1}{4} \sum_{i=1}^p (\max_{j \in S} x_{ij} - \min_{j \in S} x_{ij}).$$

- (iii) $f_S(X)$ is the squareroot of the sum of the squares of the Euclidean distances to the mean of the coordinates of the nodes in S :

$$f_S(X) = \left(\sum_{j \in S} \|x_j - \bar{x}\|_2^2 \right)^{1/2} \quad \text{where } \bar{x}_i = \frac{1}{|S|} \sum_{k \in S} x_{ik}, \quad i = 1, \dots, p.$$

- (iv) $f_S(X)$ is the sum of the ℓ_1 -distances to the (coordinate-wise) median of the coordinates of the nodes in S :

$$f_S(X) = \sum_{j \in S} \|x_j - \hat{x}\|_1 \quad \text{where } \hat{x}_i = \text{median}(\{x_{ik} \mid k \in S\}), \quad i = 1, \dots, p.$$

Solution.

- (a) Follows from the fact that $\|X - y\mathbf{1}^T\|$ is convex jointly in X and y .
- (b) (i) The maximum Euclidean column norm:

$$\|X_S - y\mathbf{1}^T\|_{2,1} = \max_{j \in S} \|x_j - y\|_2$$

is minimized by putting y at the center of the smallest Euclidean ball containing x_j , $j \in S$.

- (ii) The sum-row-max-norm

$$\|X_s - y\mathbf{1}^T\|_{\text{srm}} = \sum_{i=1}^p \max_{j \in S} |x_{ij} - y_i|$$

is minimized by placing y at the midpoint $y_i = (1/2)(\max_{j \in S} x_{ij} + \min_{j \in S} x_{ij})$.

- (iii) The Frobenius norm

$$\|X_s - y\mathbf{1}^T\|_F^2 = \sum_{j \in S} \|x_j - y\|_2^2.$$

is minimized by $y = (1/|S|) \sum_{j \in S} x_j$.

- (iv) The sum of absolute values norm.

$$\|X_s - y\mathbf{1}^T\|_{\text{sav}} = \sum_{i=1}^p \sum_{j \in S} |x_{ij} - y_i|$$

can be minimized for each i independently. The minimizer of $\sum_{j \in S} |x_{ij} - y_i|$ is the median of $\{x_{ij} \mid j \in S\}$.

15.2 Let $W \in \mathbf{S}^n$ be a symmetric matrix with nonnegative elements w_{ij} and zero diagonal. We can interpret W as the representation of a weighted undirected graph with n nodes. If $w_{ij} = w_{ji} > 0$, there is an edge between nodes i and j , with weight w_{ij} . If $w_{ij} = w_{ji} = 0$ then nodes i and j are not connected. The *Laplacian* of the weighted graph is defined as

$$L(W) = -W + \mathbf{diag}(W\mathbf{1}).$$

This is a symmetric matrix with elements

$$L_{ij}(W) = \begin{cases} \sum_{k=1}^n w_{ik} & i = j \\ -w_{ij} & i \neq j. \end{cases}$$

The Laplacian has the useful property that

$$y^T L(W)y = \sum_{i \leq j} w_{ij}(y_i - y_j)^2$$

for all vectors $y \in \mathbf{R}^n$.

- (a) Show that the function $f : \mathbf{S}^n \rightarrow \mathbf{R}$,

$$f(W) = \inf_{\mathbf{1}^T x = 0} n \lambda_{\max}(L(W) + \mathbf{diag}(x)),$$

is convex.

- (b) Give a simple argument why $f(W)$ is an upper bound on the optimal value of the combinatorial optimization problem

$$\begin{aligned} & \text{maximize} && y^T L(W)y \\ & \text{subject to} && y_i \in \{-1, 1\}, \quad i = 1, \dots, n. \end{aligned}$$

This problem is known as the *max-cut* problem, for the following reason. Every vector y with components ± 1 can be interpreted as a partition of the nodes of the graph in a set $S = \{i \mid y_i = 1\}$ and a set $T = \{i \mid y_i = -1\}$. Such a partition is called a *cut* of the graph. The objective function in the max-cut problem is

$$y^T L(W)y = \sum_{i \leq j} w_{ij}(y_i - y_j)^2.$$

If y is ± 1 -vector corresponding to a partition in sets S and T , then $y^T L(W)y$ equals four times the sum of the weights of the edges that join a point in S to a point in T . This is called the weight of the cut defined by y . The solution of the max-cut problem is the cut with the maximum weight.

- (c) The function f defined in part 1 can be evaluated, for a given W , by solving the optimization problem

$$\begin{aligned} & \text{minimize} && n \lambda_{\max}(L(W) + \mathbf{diag}(x)) \\ & \text{subject to} && \mathbf{1}^T x = 0, \end{aligned}$$

with variable $x \in \mathbf{R}^n$. Express this problem as an SDP.

- (d) Derive an alternative expression for $f(W)$, by taking the dual of the SDP in part 3. Show that the dual SDP is equivalent to the following problem:

$$\begin{aligned} & \text{maximize} && \sum_{i \leq j} w_{ij} \|p_i - p_j\|_2^2 \\ & \text{subject} && \|p_i\|_2 = 1, \quad i = 1, \dots, n, \end{aligned}$$

with variables $p_i \in \mathbf{R}^n$, $i = 1, \dots, n$. In this problem we place n points p_i on the unit sphere in \mathbf{R}^n in such a way that the weighted sum of their squared pair-wise distances is maximized.

Solution.

- (a) Follows from the fact that $\lambda_{\max}(L(W) + \mathbf{diag}(x))$ is jointly convex in W and x .
(b) If $\mathbf{1}^T x = 0$, then

$$\begin{aligned} \sup_{y \in \{-1, +1\}^n} y^T L(W)y &= \sup_{y \in \{-1, +1\}^n} y^T (L(W) + \mathbf{diag}(x))y \\ &\leq \sup_{y^T y = n} y^T (L(W) + \mathbf{diag}(x))y \\ &= n \lambda_{\max}(L(W) + \mathbf{diag}(x)). \end{aligned}$$

(c)

$$\begin{aligned} & \text{minimize} && nt \\ & \text{subject to} && L(W) + \mathbf{diag}(x) \preceq tI \\ & && \mathbf{1}^T x = 0. \end{aligned}$$

- (d) The Lagrangian is

$$\begin{aligned} G(t, x, Z) &= nt + \mathbf{tr}(Z(L(W) + \mathbf{diag}(x) - tI)) - \nu \mathbf{1}^T x \\ &= (n - \mathbf{tr} Z)t + (\mathbf{diag}(Z) - \nu \mathbf{1})^T x + \mathbf{tr}(L(W)Z). \end{aligned}$$

G is unbounded below unless $\mathbf{tr} Z = n$ and $\mathbf{diag}(Z) = \nu \mathbf{1}$, so the dual SDP is

$$\begin{aligned} & \text{maximize} && \mathbf{tr}(L(W)Z) \\ & \text{subject to} && \mathbf{tr} Z = n \\ & && \mathbf{diag}(Z) = \nu \mathbf{1} \\ & && Z \succeq 0. \end{aligned}$$

This simplifies to

$$\begin{aligned} & \text{maximize} && \mathbf{tr}(L(W)Z) \\ & \text{subject to} && \mathbf{diag}(Z) = \mathbf{1} \\ & && Z \succeq 0. \end{aligned}$$

In the geometric interpretation we interpret Z as a Gram matrix with elements $z_{ij} = p_i^T p_j$ for some set of n vectors $p_i \in \mathbf{R}^n$. The constraint $\mathbf{diag}(Z) = \mathbf{1}$ means that $\|p_i\|_2 = 1$. The

objective can be written in terms of the vectors p_i using the definition of $L(W)$:

$$\begin{aligned}
\mathbf{tr}(L(W)Z) &= -\mathbf{tr}(WZ) - \mathbf{diag}(Z)^T(W\mathbf{1}) \\
&= -2 \sum_{i < j} w_{ij} p_i^T p_j + \sum_{i=1}^n (\sum_{j=1}^n w_{ij}) \|p_i\|_2^2 \\
&= \sum_{i < j} w_{ij} (-2p_i^T p_j + \|p_i\|_2^2 + \|p_j\|_2^2) \\
&= \sum_{i < j} w_{ij} \|p_i - p_j\|_2^2.
\end{aligned}$$

15.3 Utility versus latency trade-off in a network. We consider a network with m edges, labeled $1, \dots, m$, and n flows, labeled $1, \dots, n$. Each flow has an associated nonnegative flow rate f_j ; each edge or link has an associated positive capacity c_i . Each flow passes over a fixed set of links (its route); the total traffic t_i on link i is the sum of the flow rates over all flows that pass through link i . The flow routes are described by a routing matrix $R \in \mathbf{R}^{m \times n}$, defined as

$$R_{ij} = \begin{cases} 1 & \text{flow } j \text{ passes through link } i \\ 0 & \text{otherwise.} \end{cases}$$

Thus, the vector of link traffic, $t \in \mathbf{R}^m$, is given by $t = Rf$. The link capacity constraint can be expressed as $Rf \leq c$. The (logarithmic) network utility is defined as $U(f) = \sum_{j=1}^n \log f_j$.

The (average queuing) delay on link i is given by

$$d_i = \frac{1}{c_i - t_i}$$

(multiplied by a constant, that doesn't matter to us). We take $d_i = \infty$ for $t_i = c_i$. The delay or latency for flow j , denoted l_j , is the sum of the link delays over all links that flow j passes through. We define the maximum flow latency as

$$L = \max\{l_1, \dots, l_n\}.$$

We are given R and c ; we are to choose f .

- (a) How would you find the flow rates that maximize the utility U , ignoring flow latency? (In particular, we allow $L = \infty$.) We'll refer to this maximum achievable utility as U^{\max} .
- (b) How would you find the flow rates that minimize the maximum flow latency L , ignoring utility? (In particular, we allow $U = -\infty$.) We'll refer to this minimum achievable latency as L^{\min} .
- (c) Explain how to find the optimal trade-off between utility U (which we want to maximize) and latency L (which we want to minimize).
- (d) Find U^{\max} , L^{\min} , and plot the optimal trade-off of utility versus latency for the network with data given in `net_util_data.m`, showing L^{\min} and U^{\max} on the same plot. Your plot should cover the range from $L = 1.1L^{\min}$ to $L = 11L^{\min}$. Plot U vertically, on a linear scale, and L horizontally, using a log scale.

Note. For parts (a), (b), and (c), your answer can involve solving one or more convex optimization problems. But if there is a simpler solution, you should say so.

Solution.

- (a) To maximize utility we solve the convex problem

$$\begin{aligned} & \text{maximize} && \sum_{j=1}^n \log f_j \\ & \text{subject to} && Rf \preceq c. \end{aligned}$$

with variable f , and the implicit constraint $f \succeq 0$.

- (b) Link delay is monotonically increasing in traffic, so it is minimized with zero traffic. Since flow latency is the sum of link delays, it is also minimized by the choice $f = 0$. So zero flow minimizes each flow latency. With $f = 0$ we have $d_i = 1/c_i$. To sum these delays over the routes, we multiply by R^T to get $l = R^T(1/c_1, \dots, 1/c_m)$, where l is the vector of flow latencies. Thus we have

$$L^{\min} = \max \left(R^T(1/c_1, \dots, 1/c_m) \right),$$

where the max is the maximum over the entries of the vector $R^T(1/c_1, \dots, 1/c_m)$.

- (c) Let b_1^T, \dots, b_m^T denote the rows of R . To find the optimal trade-off between U and L , we solve the problem

$$\begin{aligned} & \text{maximize} && \sum_{j=1}^n \log f_j \\ & \text{subject to} && Rf \preceq c, \\ & && \sum_{i=1}^m \frac{R_{ij}}{c_i - b_i^T f} \leq L, \quad j = 1, \dots, n, \end{aligned}$$

for a range of values of L . The variable here is f , and there is an implicit constraint $f \succeq 0$. Evidently, if $Rf \preceq c$ we must have $c_i - b_i^T f \geq 0$, so the constraints

$$\sum_{i=1}^m \frac{R_{ij}}{c_i - b_i^T f} \leq L, \quad j = 1, \dots, n,$$

are convex. This is therefore a convex optimization problem.

- (d) The following code computes the utility versus latency trade-off.

```
% solution for network utility problem
clear all;
net_util_data;

% let's find max utility with no delay constraint
cvx_begin
    variable f(n)
    maximize geomean(f)
    R*f <= c
    f >= 0; % not needed; enforced by geomean domain
cvx_end
Umax=sum(log(f));
```

```

% let's find min latency with no utility constraint
% can be done analytically: just take f=0
Lmin = max(R'*(1./c));

% now let's do pareto curve

N = 20;
ds = 1.10*Lmin*logspace(0,1,N); % go from 10% above L
Uopt = [];

for d = ds
    cvx_begin
        variable f(n)
        maximize geomean(f);
        R'*inv_pos(c-R*f) <= d*ones(n,1)
        f >= 0; % not needed; enforced by geomean domain
        R*f <= c; % not needed; enforced by inv_pos domain
    cvx_end
    Uopt = [Uopt n*log(cvx_optval)];
end

semilogx(ds,Uopt,'k-',[Lmin,ds],[Umax,ones(1,N)*Umax],...
    'k--',[1,1]*Lmin,[Uopt(1),Umax],'k--')
% axis([ds(1), ds(N), -480, -340]);
xlabel('L'); ylabel('U');

```

The trade-off curve is shown in figure 16.

15.4 Allocation of interdiction effort. A smuggler moves along a directed acyclic graph with m edges and n nodes, from a source node (which we take as node 1) to a destination node (which we take as node n), along some (directed) path. Each edge k has a detection failure probability p_k , which is the probability that the smuggler passes over that edge undetected. The detection events on the edges are independent, so the probability that the smuggler makes it to the destination node undetected is $\prod_{j \in \mathcal{P}} p_j$, where $\mathcal{P} \subset \{1, \dots, m\}$ is (the set of edges on) the smuggler's path. We assume that the smuggler knows the detection failure probabilities and will take a path that maximizes the probability of making it to the destination node undetected. We let P^{\max} denote this maximum probability (over paths). (Note that this is a function of the edge detection failure probabilities.)

The edge detection failure probability on an edge depends on how much interdiction resources are allocated to the edge. Here we will use a very simple model, with $x_j \in \mathbf{R}_+$ denoting the effort (say, yearly budget) allocated to edge j , with associated detection failure probability $p_j = e^{-a_j x_j}$, where $a_j \in \mathbf{R}_{++}$ are given. The constraints on x are a maximum for each edge, $x \leq x^{\max}$, and a total budget constraint, $\mathbf{1}^T x \leq B$.

- (a) Explain how to solve the problem of choosing the interdiction effort vector $x \in \mathbf{R}^m$, subject to the constraints, so as to minimize P^{\max} . Partial credit will be given for a method that involves an enumeration over all possible paths (in the objective or constraints). *Hint.* For

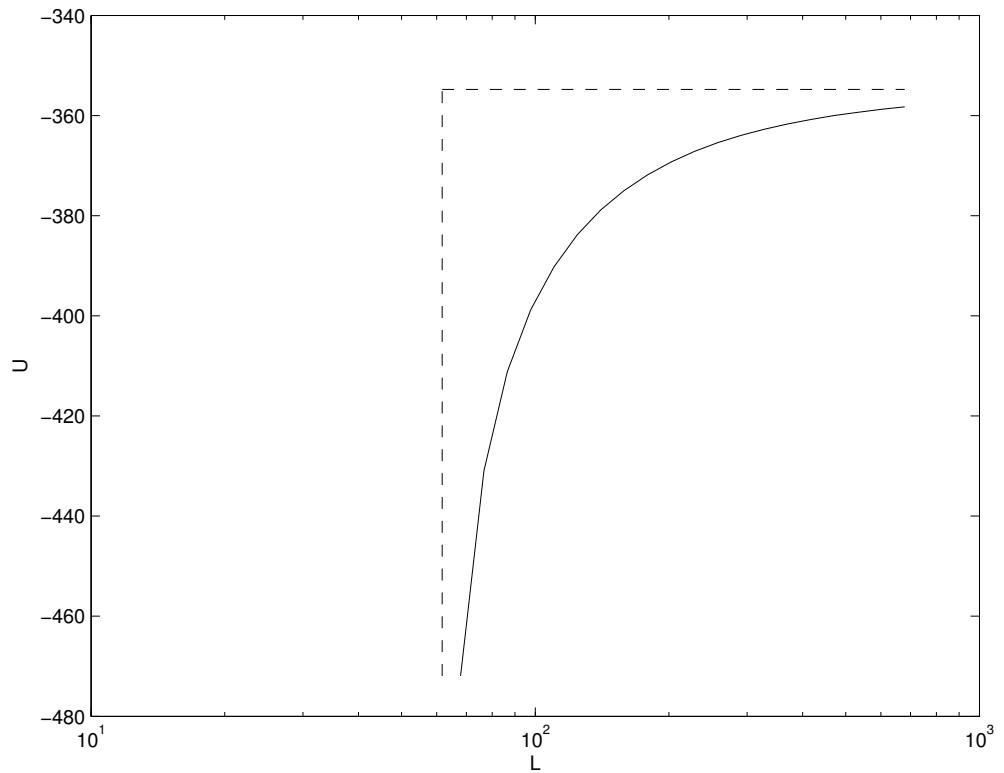


Figure 16: Utility versus maximum latency trade-off. Solid: trade-off curve, dashed: L_{\min} and U_{\max}

each node i , let P_i denote the maximum of $\prod_{k \in \mathcal{P}} p_k$ over all paths \mathcal{P} from the source node 1 to node i (so $P^{\max} = P_n$).

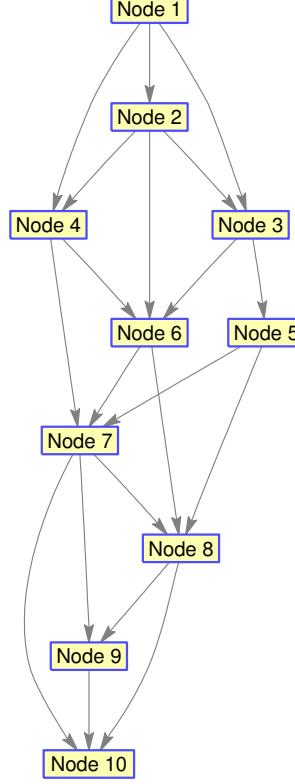
- (b) Carry out your method on the problem instance given in `interdict_alloc_data.m`. The data file contains the data a , x^{\max} , B , and the graph incidence matrix $A \in \mathbf{R}^{n \times m}$, where

$$A_{ij} = \begin{cases} -1 & \text{if edge } j \text{ leaves node } i \\ +1 & \text{if edge } j \text{ enters node } i \\ 0 & \text{otherwise.} \end{cases}$$

Give $P^{\max*}$, the optimal value of P^{\max} , and compare it to the value of P^{\max} obtained with uniform allocation of resources, *i.e.*, with $x = (B/m)\mathbf{1}$.

Hint. Given a vector $z \in \mathbf{R}^n$, $A^T z$ is the vector of edge differences: $(A^T z)_j = z_k - z_l$ if edge j goes from node l to node k .

The following figure shows the topology of the graph in question. (The data file contains A ; this figure, which is not needed to solve the problem, is shown here so you can visualize the graph.)



Solution. We will work with the logs of the detection failure probabilities, $y_j = \log p_j = -a_j x_j$. Consider these as edge weights on the graph. The smuggler chooses a path from source to destination that maximizes the total path weight (*i.e.*, the sum of weights along its edges). Thus $\log P^{\max}$ is the maximum, over the set of paths from source to destination, of a sum of linear functions of x ; so it is a piecewise-linear convex function of x . It follows that minimizing $\log P^{\max}$ subject to the constraints $x^{\max} \succeq x \succeq 0$, $\mathbf{1}^T x \leq B$, is a convex optimization problem. Unfortunately, this formulation requires an enumeration of all paths. So let's find one that does not.

Following the given hint, we can write a recursion for P_i as follows:

$$P_i = \max_{k \in \text{pred}(i)} (p_{ik} P_k),$$

where $\text{pred}(i)$ denotes the predecessor nodes of i , and p_{ik} is the detection failure probability on the edge from k to i . We also have $P_1 = 1$, $P_n = P^{\max}$.

Letting $z_i = \log P_i$ we have the problem

$$\begin{aligned} & \text{minimize} && z_n \\ & \text{subject to} && z_i = \max_{k \in \text{pred}(i)} (-a_{ik} x_{ik} + z_k), \quad i = 2, \dots, n \\ & && z_1 = 0 \\ & && x^{\max} \succeq x \succeq 0, \quad \mathbf{1}^T x \leq B, \end{aligned}$$

with variables $x \in \mathbf{R}^m$ and $z \in \mathbf{R}^n$. In the second line, we use x_{ik} to denote x_j , where j corresponds to the edge from node k to node i .

The objective and constraints in this problem are all monotone nondecreasing in z_i , so it follows that we can relax it to the convex problem

$$\begin{aligned} & \text{minimize} && z_n \\ & \text{subject to} && z_i \geq \max_{k \in \text{pred}(i)} (-a_{ik}x_{ik} + z_k), \quad i = 2, \dots, n \\ & && z_1 = 0 \\ & && x^{\max} \succeq x \succeq 0, \quad \mathbf{1}^T x \leq B. \end{aligned}$$

Using the incidence matrix this can be written as the following LP:

$$\begin{aligned} & \text{minimize} && z_n \\ & \text{subject to} && A^T z \succeq -\mathbf{diag}(a)x \\ & && z_1 = 0 \\ & && x^{\max} \succeq x \succeq 0, \quad \mathbf{1}^T x \leq B. \end{aligned}$$

GP formulation. Equivalently, we can formulate the problem as a GP:

$$\begin{aligned} & \text{minimize} && P_n \\ & \text{subject to} && P_i \geq \max_{k \in \text{pred}(i)} (P_k y_{ik}^{-a_{ik}}), \quad i = 2, \dots, n \\ & && P_1 = 1 \\ & && y \succeq 1 \\ & && \prod_{i=1}^m y_i \leq \exp(B), \end{aligned}$$

over $P \in \mathbf{R}^n$ and $y \in \mathbf{R}^m$ where $y_i = \exp(x_i)$ (we use the notation y_{ik} as defined above).

The optimal probability of detection failure is $P^{\max} = 0.043$, so the smuggler will get through with probability 4.3%. Using uniform allocation of resources, we obtain $P^{\max} = 0.247$, so the smuggler gets through with probability 24.7%.

The following code was used to solve the problem.

```
interdict_alloc_data

% dynamic programming solution
cvx_begin
variables x(m) z(n)
minimize(z(n))
z(1) == 0
A'*z >= -diag(a)*x
x >= 0
x <= x_max
sum(x) <= B
cvx_end

% uniform allocation
```

```

x_unif=B/m*ones(m,1);
cvx_begin
variables z_unif(n)
minimize(z_unif(n))
z_unif(1) == 0
A'*z_unif >= -diag(a)*x_unif
cvx_end

```

15.5 Network sizing. We consider a network with n directed arcs. The flow through arc k is denoted x_k and can be positive, negative, or zero. The flow vector x must satisfy the network constraint $Ax = b$ where A is the node-arc incidence matrix and b is the external flow supplied to the nodes.

Each arc has a positive capacity or width y_k . The quantity $|x_k|/y_k$ is the flow density in arc k . The cost of the flow in arc k depends on the flow density and the width of the arc, and is given by $y_k \phi_k(|x_k|/y_k)$, where ϕ_k is convex and nondecreasing on \mathbf{R}_+ .

- (a) Define $f(y, b)$ as the optimal value of the network flow optimization problem

$$\begin{aligned} &\text{minimize} && \sum_{k=1}^n y_k \phi_k(|x_k|/y_k) \\ &\text{subject to} && Ax = b \end{aligned}$$

with variable x , for given values of the arc widths $y \succ 0$ and external flows b . Is f a convex function (jointly in y, b)? Carefully explain your answer.

- (b) Suppose b is a discrete random vector with possible values $b^{(1)}, \dots, b^{(m)}$. The probability that $b = b^{(j)}$ is π_j . Consider the problem of sizing the network (selecting the arc widths y_k) so that the expected cost is minimized:

$$\text{minimize } g(y) + \mathbf{E} f(y, b). \quad (61)$$

The variable is y . Here g is a convex function, representing the installation cost, and $\mathbf{E} f(y, b)$ is the expected optimal network flow cost

$$\mathbf{E} f(y, b) = \sum_{j=1}^m \pi_j f(y, b^{(j)}),$$

where f is the function defined in part 1. Is (61) a convex optimization problem?

Solution.

- (a) We first note that $\phi_k(|x_k|)$ is convex as a function of x_k . This follows from the composition rules. The function is a composition $h(g(x_k))$ of two functions:
- the convex function $g(x_k) = |x_k|$
 - the convex nondecreasing function $h(u)$ defined as $h(u) = \phi_k(u)$ if $u \geq 0$ and $h(u) = \phi_k(0)$ for $u \leq 0$.

The function $y_k \phi_k(|x_k|/y_k)$ is jointly convex in x_k , y_k because it is the perspective of a convex function.

The function f can be expressed as $\inf_x F(x, y, b)$ where

$$F(x, y, b) = \begin{cases} \sum_k y_k \phi_k(|x_k|/y_k) & Ax = b \\ +\infty & \text{otherwise.} \end{cases}$$

This function F is jointly convex in (x, y, b) . This implies that $f(y, b) = \inf_x f(x, y, b)$ is convex.

(b) The expression

$$\mathbf{E} f(y, b) = \sum_{j=1}^m \pi_j f(y, b^{(j)})$$

shows that $\mathbf{E} f(y, b)$ is the sum of convex functions of y .

15.6 Maximizing algebraic connectivity of a graph. Let $G = (V, E)$ be a weighted undirected graph with $n = |V|$ nodes, $m = |E|$ edges, and weights $w_1, \dots, w_m \in \mathbf{R}_+$ on the edges. If edge k connects nodes i and j , then define $a_k \in \mathbf{R}^n$ as $(a_k)_i = 1$, $(a_k)_j = -1$, with other entries zero. The *weighted Laplacian* (matrix) of the graph is defined as

$$L = \sum_{k=1}^m w_k a_k a_k^T = A \mathbf{diag}(w) A^T,$$

where $A = [a_1 \cdots a_m] \in \mathbf{R}^{n \times m}$ is the *incidence matrix* of the graph. Nonnegativity of the weights implies $L \succeq 0$.

Denote the eigenvalues of the Laplacian L as

$$\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n,$$

which are functions of w . The minimum eigenvalue λ_1 is always zero, while the second smallest eigenvalue λ_2 is called the *algebraic connectivity* of G and is a measure of the connectedness of a graph: The larger λ_2 is, the better connected the graph is. It is often used, for example, in analyzing the robustness of computer networks.

Though not relevant for the rest of the problem, we mention a few other examples of how the algebraic connectivity can be used. These results, which relate graph-theoretic properties of G to properties of the spectrum of L , belong to a field called *spectral graph theory*. For example, $\lambda_2 > 0$ if and only if the graph is connected. The eigenvector v_2 associated with λ_2 is often called the *Fiedler vector* and is widely used in a graph partitioning technique called *spectral partitioning*, which assigns nodes to one of two groups based on the sign of the relevant component in v_2 . Finally, λ_2 is also closely related to a quantity called the *isoperimetric number* or *Cheeger constant* of G , which measures the degree to which a graph has a bottleneck.

The problem is to choose the edge weights $w \in \mathbf{R}_+^m$, subject to some linear inequalities (and the nonnegativity constraint) so as to maximize the algebraic connectivity:

$$\begin{aligned} & \text{maximize} && \lambda_2 \\ & \text{subject to} && w \succeq 0, \quad Fw \preceq g, \end{aligned}$$

with variable $w \in \mathbf{R}^m$. The problem data are A (which gives the graph topology), and F and g (which describe the constraints on the weights).

- (a) Describe how to solve this problem using convex optimization.
- (b) *Numerical example.* Solve the problem instance given in `max_alg_conn_data.m`, which uses $F = \mathbf{1}^T$ and $g = 1$ (so the problem is to allocate a total weight of 1 to the edges of the graph). Compare the algebraic connectivity for the graph obtained with the optimal weights w^* to the one obtained with $w^{\text{unif}} = (1/m)\mathbf{1}$ (*i.e.*, a uniform allocation of weight to the edges).

Use the function `plotgraph(A,xy,w)` to visualize the weighted graphs, with weight vectors w^* and w^{unif} . You will find that the optimal weight vector v^* has some zero entries (which due to the finite precision of the solver, will appear as small weight values); you may want to round small values (say, those under 10^{-4}) of w^* to exactly zero. Use the `gplot` function to visualize the original (given) graph, and the subgraph associated with nonzero weights in w^* . *Briefly* comment on the following (incorrect) intuition: “The more edges a graph has, the more connected it is, so the optimal weight assignment should make use of all available edges.”

Solution.

- (a) The only question is how to ensure that λ_2 is a concave function of w . In general, the second lowest eigenvalue of a symmetric matrix is not a concave function. The trick is to form a new matrix whose smallest eigenvalue is exactly λ_2 .

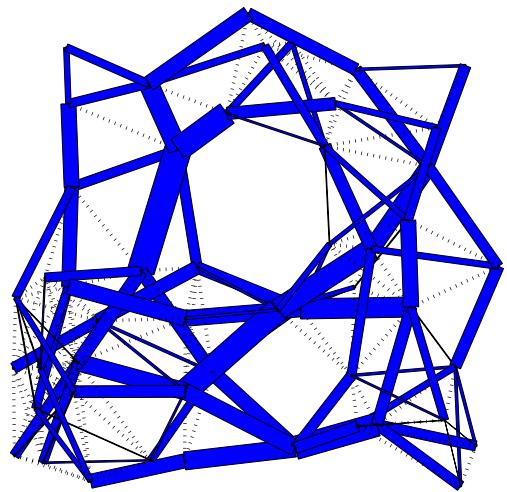
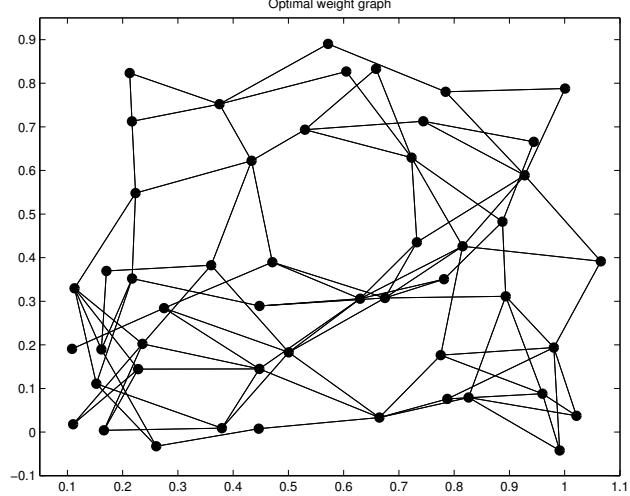
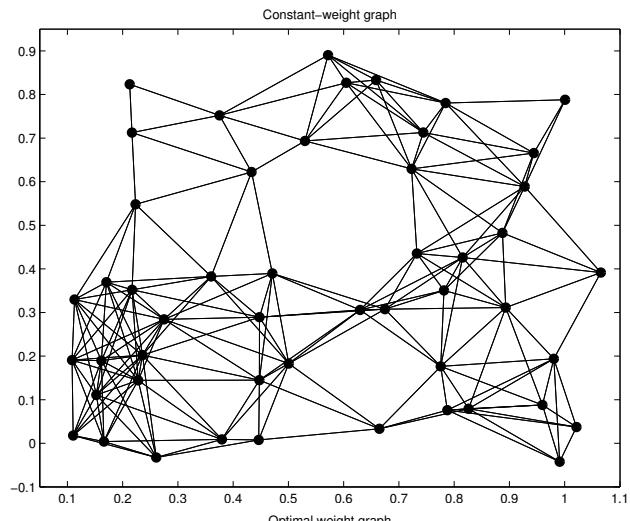
First, note that $\mathbf{1}$ is an eigenvector of L corresponding to λ_1 ; this follows because $\mathbf{1}^T a_k = 0$ for all k . We can thus express the algebraic connectivity λ_2 as the minimum eigenvalue of L , restricted to the subspace $\mathbf{1}^\perp$, *i.e.*, $\lambda_2 = \lambda_{\min}(Q^T L Q)$, where $Q \in \mathbf{R}^{n \times (n-1)}$ is a matrix whose columns form an orthonormal basis for $\mathcal{N}(\mathbf{1}^T) = \mathbf{1}^\perp$, so that $Q^T L Q$ has eigenvalues $\lambda_2(L), \dots, \lambda_n(L)$. Since $\lambda_{\min}(Q^T L Q)$ is concave in w , we’re done. We end up with the problem

$$\begin{aligned} & \text{maximize} && \lambda_{\min}(Q^T L Q) \\ & \text{subject to} && w \succeq 0, \quad Fw \preceq g, \end{aligned}$$

with variable $w \in \mathbf{R}^m$.

- (b) The solution and plots are below. From the first two plots, it is clear that the solution is sparse, in the sense that many edges that would be active in the uniform weight graph have been disabled in the optimal weight graph. In particular, many edges have been removed from very densely connected sections of the graph.

We find that the algebraic connectivities of the uniform and optimal weights are 0.002204 and 0.005018, respectively. In other words, the optimal graph is better connected despite there being fewer total edges, because additional weight (*e.g.*, capacity) is assigned to particularly important edges that substantially improve the global connectivity of the graph.



```

%% maximizing algebraic connectivity of a graph

max_alg_conn_data;

Q = null(ones(1,n)); % columns of Q are orthonormal basis for 1^\perp

cvx_begin
    variable w(m)
    L = A*diag(w)*A';
    maximize (lambda_min(Q'*L*Q))
    subject to
        w >= 0;
        F*w <= g;
cvx_end

w(abs(w) < 1e-4) = 0;

% compare algebraic connectivities
L_unif = (1/m)*A*A';
dunif = eig(L_unif);
dopt = eig(L);
fprintf(1, 'Algebraic connectivity of L_unif: %f\n', dunif(2));
fprintf(1, 'Algebraic connectivity of L_opt: %f\n', dopt(2));

% plot topology of constant-weight graph
figure(1), clf
gplot(L_unif,xy);
hold on;
plot(xy(:,1), xy(:,2), 'ko','LineWidth',4, 'MarkerSize',4);
axis([0.05 1.1 -0.1 0.95]);
title('Constant-weight graph')
hold off;
print -deps graph_plot1.eps;

% plot topology of optimal weight graph
figure(2), clf
gplot(L,xy);
hold on;
plot(xy(:,1), xy(:,2), 'ko','LineWidth',4, 'MarkerSize',4);
axis([0.05 1.1 -0.1 0.95]);
title('Optimal weight graph')
hold off;
print -deps graph_plot2.eps;

% plot optimal weight graph with edge thickness proportional to weight

```

```

figure(3), clf
plotgraph(A,xy,w);
text(0.3,1.05,'Optimal weights')
print -deps graph_plot3.eps;

```

15.7 Graph isomorphism via linear programming. An (undirected) graph with n vertices can be described by its adjacency matrix $A \in \mathbf{S}^n$, given by

$$A_{ij} = \begin{cases} 1 & \text{there is an edge between vertices } i \text{ and } j \\ 0 & \text{otherwise.} \end{cases}$$

Two (undirected) graphs are *isomorphic* if we can permute the vertices of one so it is the same as the other (*i.e.*, the same pairs of vertices are connected by edges). If we describe them by their adjacency matrices A and B , isomorphism is equivalent to the existence of a permutation matrix $P \in \mathbf{R}^{n \times n}$ such that $PAP^T = B$. (Recall that a matrix P is a permutation matrix if each row and column has exactly one entry 1, and all other entries 0.) Determining if two graphs are isomorphic, and if so, finding a suitable permutation matrix P , is called the *graph isomorphism problem*.

Remarks (not needed to solve the problem). It is not currently known if the graph isomorphism problem is NP-complete or solvable in polynomial time. The graph isomorphism problem comes up in several applications, such as determining if two descriptions of a molecule are the same, or whether the physical layout of an electronic circuit correctly reflects the given circuit schematic diagram.

- (a) Find a set of linear equalities and inequalities on $P \in \mathbf{R}^{n \times n}$, that together with the Boolean constraint $P_{ij} \in \{0, 1\}$, are necessary and sufficient for P to be a permutation matrix satisfying $PAP^T = B$. Thus, the graph isomorphism problem is equivalent to a Boolean feasibility LP.
- (b) Consider the relaxed version of the Boolean feasibility LP found in part (a), *i.e.*, the LP that results when the constraints $P_{ij} \in \{0, 1\}$ are replaced with $P_{ij} \in [0, 1]$. When this LP is infeasible, we can be sure that the two graphs are not isomorphic. If a solution of the LP is found that satisfies $P_{ij} \in \{0, 1\}$, then the graphs are isomorphic and we have solved the graph isomorphism problem. This of course does not always happen, even if the graphs are isomorphic.

A standard trick to encourage the entries of P to take on the values 0 and 1 is to add a random linear objective to the relaxed feasibility LP. (This doesn't change whether the problem is feasible or not.) In other words, we minimize $\sum_{i,j} W_{ij}P_{ij}$, where W_{ij} are chosen randomly (say, from $\mathcal{N}(0, 1)$). (This can be repeated with different choices of W .)

Carry out this scheme for the two isomorphic graphs with adjacency matrices A and B given in `graph_isomorphism_data.*` to find a permutation matrix P that satisfies $PAP^T = B$. Report the permutation vector, given by the matrix-vector product Pv , where $v = (1, 2, \dots, n)$. Verify that all the required conditions on P hold. To check that the entries of the solution of the LP are (close to) $\{0, 1\}$, report $\max_{i,j} P_{ij}(1 - P_{ij})$. And yes, you might have to try more than one instance of the randomized method described above before you find a permutation that establishes isomorphism of the two graphs.

Solution.

- (a) P is a permutation matrix if and only if $P\mathbf{1} = \mathbf{1}$, $P^T\mathbf{1} = \mathbf{1}$, and $P_{ij} \in \{0, 1\}$. The condition $PAP^T = B$ is a quadratic equality on P . But we observe that since P is a permutation matrix, we have $P^{-1} = P^T$. Multiplying $PAP^T = B$ on the right by P we get $PA = BP$, a set of linear equations in P . So, P is a permutation matrix that satisfies $PAP^T = B$ if and only if

$$P\mathbf{1} = \mathbf{1}, \quad P^T\mathbf{1} = \mathbf{1}, \quad PA = BP, \quad P_{ij} \in \{0, 1\}.$$

This is a set of linear equations in P , together with the Boolean condition $P_{ij} \in \{0, 1\}$.

- (b) The LP relaxation, with the random cost function suggested, is

$$\begin{aligned} & \text{minimize} && \mathbf{tr}(W^T P) \\ & \text{subject to} && P\mathbf{1} = \mathbf{1}, \quad P^T\mathbf{1} = \mathbf{1}, \quad PA = BP \\ & && 0 \leq P_{ij} \leq 1. \end{aligned}$$

(The constraints $P_{ij} \leq 1$ are redundant and can be removed.)

When we solve this problem for the given data, we find that there are two different permutation vectors π_1 and π_2 that relate A and B . The quantity $\max_{i,j} P_{ij}(1 - P_{ij})$ is suitably small, on the order of 10^{-6} . These two vectors are

$$\begin{aligned} \pi_1 = & (16, 22, 27, 9, 5, 13, 1, 25, 21, 23, 19, 14, 26, 6, 2, 7, \\ & 30, 11, 10, 17, 15, 24, 8, 18, 20, 3, 12, 28, 4, 29), \end{aligned}$$

and

$$\begin{aligned} \pi_2 = & (6, 22, 27, 19, 15, 3, 11, 25, 21, 23, 9, 4, 26, 16, 12, 17, \\ & 30, 1, 20, 7, 5, 24, 18, 8, 10, 13, 2, 28, 14, 29). \end{aligned}$$

It is interesting to notice that solving the feasibility problem with itself (*i.e.*, with objective function 0) usually doesn't end up finding a permutation matrix. However adding the linear random objective does help us find a permutation matrix.

The following MATLAB code solves the problem

```
graph_isomorphism_data
n = size(A,1);
W = randn(n);

cvx_begin quiet
variable P(n,n)
minimize trace(W*P)
subject to
    P'*ones(n,1) == ones(n,1);
    P*ones(n,1) == ones(n,1);
    P*A - B*P == 0;
    0 <= P <= 1
cvx_end
fprintf(['maximum p(1-p) where p is an entry of P is ' ...
    '%d\n'], max(max(P.*(1-P))))
```

```

fprintf(['norm of the residual for first constraint is ' ...
        '%d\n'], norm(P'*ones(n,1) - ones(n,1)))
fprintf(['norm of the residual for second constraint is ' ...
        '%d\n'], norm(P*ones(n,1) - ones(n,1)))
fprintf(['norm of the residual for third constraint is ' ...
        '%d\n'], norm(P*A - B*P))
P*(1:n),

```

The following Python code solves the problem

```

from cvxpy import *
from graph_isomorphism_data import *

n = A.shape[0]
W = np.random.randn(n, n)

P = Variable(n,n)
objective = Minimize(trace(W*P))
constraints = [ P*np.ones([n,1]) == np.ones([n,1]),
                 P.T*np.ones([n,1]) == np.ones([n,1]),
                 P*A == B*P,
                 0 <= P,
                 P <= 1]

prob = Problem(objective, constraints)
result = prob.solve()
P = P.value

print('maximum p(1-p) where p is an entry of P is '
      + str(np.max(np.multiply(P,1-P))))
print('norm of the residual for first constraint is '
      + str(np.linalg.norm(P*np.ones([n,1])- np.ones([n,1]))))
print('norm of the residual for second constraint is '
      + str(np.linalg.norm(P.T*np.ones([n,1]) - np.ones([n,1]))))
print('norm of the residual for third constraint is '
      + str(np.linalg.norm(P*A-B*P)))
print(np.dot(P,np.arange(n)+1))

```

The following Julia code solves the problem

```

using Convex, SCS
include("graph_isomorphism_data.jl");

n = size(A,1);
W = randn(n,n);

P = Variable(n, n);
obj = trace(W*P);

```

```
constraints = [
    sum(P, 2) == ones(n),
    sum(P, 1) == ones(1, n),
    P*A == B*P,
    P >= 0,
    P <= 1
];
prob = minimize(obj, constraints);
solve!(prob, SCSSolver(verbose=false, max_iters=20000));
P = P.value;
println(maximum(P.*(1-P)));
println(norm(sum(P, 2) - ones(n)));
println(norm(sum(P, 1) - ones(1, n)));
println(vecnorm(P*A-B*P));
println(P);
```

16 Energy and power

16.1 Power flow optimization with ‘ $N - 1$ ’ reliability constraint. We model a network of power lines as a graph with n nodes and m edges. The power flow along line j is denoted p_j , which can be positive, which means power flows along the line in the direction of the edge, or negative, which means power flows along the line in the direction opposite the edge. (In other words, edge orientation is only used to determine the direction in which power flow is considered positive.) Each edge can support power flow in either direction, up to a given maximum capacity P_j^{\max} , i.e., we have $|p_j| \leq P_j^{\max}$.

Generators are attached to the first k nodes. Generator i provides power g_i to the network. These must satisfy $0 \leq g_i \leq G_i^{\max}$, where G_i^{\max} is a given maximum power available from generator i . The power generation costs are $c_i > 0$, which are given; the total cost of power generation is $c^T g$.

Electrical loads are connected to the nodes $k + 1, \dots, n$. We let $d_i \geq 0$ denote the demand at node $k + i$, for $i = 1, \dots, n - k$. We will consider these loads as given. In this simple model we will neglect all power losses on lines or at nodes. Therefore, power must balance at each node: the total power flowing into the node must equal the sum of the power flowing out of the node. This power balance constraint can be expressed as

$$Ap = \begin{bmatrix} -g \\ d \end{bmatrix},$$

where $A \in \mathbf{R}^{n \times m}$ is the node-incidence matrix of the graph, defined by

$$A_{ij} = \begin{cases} +1 & \text{edge } j \text{ enters node } i, \\ -1 & \text{edge } j \text{ leaves node } i, \\ 0 & \text{otherwise.} \end{cases}$$

In the basic power flow optimization problem, we choose the generator powers g and the line flow powers p to minimize the total power generation cost, subject to the constraints listed above. The (given) problem data are the incidence matrix A , line capacities P^{\max} , demands d , maximum generator powers G^{\max} , and generator costs c .

In this problem we will add a basic (and widely used) reliability constraint, commonly called an ‘ $N - 1$ constraint’. (N is not a parameter in the problem; ‘ $N - 1$ ’ just means ‘all-but-one’.) This states that the system can still operate even if any one power line goes out, by re-routing the line powers. The case when line j goes out is called ‘failure contingency j ’; this corresponds to replacing P_j^{\max} with 0. The requirement is that there must exist a contingency power flow vector $p^{(j)}$ that satisfies all the constraints above, with $p_j^{(j)} = 0$, using the same given generator powers. (This corresponds to the idea that power flows can be re-routed quickly, but generator power can only be changed more slowly.) The ‘ $N - 1$ reliability constraint’ requires that for each line, there is a contingency power flow vector. The ‘ $N - 1$ reliability constraint’ is (implicitly) a constraint on the generator powers.

The questions below concern the specific instance of this problem with data given in `rel_pwr_flow_data.*`. (Executing this file will also generate a figure showing the network you are optimizing.) Especially for part (b) below, you must explain exactly how you set up the problem as a convex optimization problem.

- (a) *Nominal optimization.* Find the optimal generator and line power flows for this problem instance (without the $N - 1$ reliability constraint). Report the optimal cost and generator powers. (You do not have to give the power line flows.)
- (b) *Nominal optimization with $N - 1$ reliability constraint.* Minimize the nominal cost, but you must choose generator powers that meet the $N - 1$ reliability requirement as well. Report the optimal cost and generator powers. (You do not have to give the nominal power line flows, or any of the contingency flows.)

Solution.

- (a) To find the optimal generators and line power flows we solve the LP

$$\begin{aligned} & \text{minimize} && c^T g \\ & \text{subject to} && Ap = \begin{bmatrix} -g \\ d \end{bmatrix} \\ & && -P^{\max} \leq p \leq P^{\max} \\ & && 0 \leq g \leq G^{\max}, \end{aligned}$$

with variables g and p .

- (b) To handle the additional $N - 1$ reliability constraint, we must introduce a set of power flow vectors for each contingency. We then solve the LP

$$\begin{aligned} & \text{minimize} && c^T g \\ & \text{subject to} && Ap^{(j)} = \begin{bmatrix} -g \\ d \end{bmatrix}, \quad j = 1, \dots, m \\ & && p_j^{(j)} = 0, \quad j = 1, \dots, m \\ & && -P^{\max} \leq p^{(j)} \leq P^{\max}, \quad j = 1, \dots, m \\ & && 0 \leq g \leq G^{\max}, \end{aligned}$$

with variables $g \in \mathbf{R}^k$ and $p^{(1)}, \dots, p^{(m)} \in \mathbf{R}^m$.

The optimal costs are 44.60 and 56.20 for parts (a) and (b) respectively. The optimal generator powers are

$$g_{\text{nom}} = \begin{bmatrix} 3.0 \\ 0.0 \\ 2.3 \\ 7.0 \end{bmatrix}, \quad g_{\text{rel}} = \begin{bmatrix} 1.9 \\ 1.9 \\ 4.0 \\ 4.5 \end{bmatrix}.$$

We can say a little about these results. In the nominal case, it turns out that the line capacities are not tight; that is, we have no congestion on the transmission lines. So we must select a set of generator powers that deliver the required power, which is the sum of the demands (12.32 power units). To do this most efficiently, we start with generator 4, which has the lowest cost, and we set it to its maximum, which gives us a total of 7 power units. We then go the second cheapest generator, generator 1, and set it to its maximum (3 power units), which gives us a total of 10 power units. Finally, we go to the third cheapest generator, generator 3, and use it to satisfy the remaining demand.

When we impose the additional $N - 1$ reliability constraint, we are forced to shift some generation to more expensive generators.

The following MATLAB code solves parts (a) and (b).

```

rel_pwr_flow_data;

% nominal case
cvx_begin
    variables p(m) g_nom(k)
    minimize (c'*g_nom)
    subject to
        A*p == [-g_nom;d];
        abs(p) <= Pmax;
        g_nom <= Gmax;
        g_nom >= 0;
cvx_end
nom_cost = cvx_optval;

% N-1 case
cvx_begin
    variables P(m,m) g_rel(k)
    minimize (c'*g_rel)
    subject to
        A*P == [-g_rel;d]*ones(1,m);
        diag(P) == 0;
        abs(P) <= Pmax*ones(1,m);
        g_rel <= Gmax;
        g_rel >= 0;
cvx_end
rel_cost = cvx_optval;

% show nominal optimal and reliable optimal cost
[nom_cost rel_cost]
% show nominal optimal and reliable optimal generator powers
[g_nom g_rel]

```

The following Python code solves parts (a) and (b).

```

import cvxpy as cvx
from rel_pwr_flow_data import *

# nominal case
p = cvx.Variable(m)
g_nom = cvx.Variable(k)
nom_cost = cvx.Problem(cvx.Minimize(c.T*g_nom),
                       [A[:, :]*p == -g_nom,

```

```

A[k:,:]*p == d.T,
cvx.abs(p) <= Pmax.T,
g_nom <= Gmax,
g_nom >= 0.]).solve()

# N-1 case
P = cvx.Variable(m,m)
g_rel = cvx.Variable(k)
rel_cost = cvx.Problem(cvx.Minimize(c.T*g_rel),
    [A[:k,:]*P == -g_rel*np.ones((1,m)),
     A[k:,:]*P == d.T*np.ones((1,m)),
     cvx.diag(P) == 0,
     cvx.abs(P) <= Pmax.T*np.ones((1,m)),
     g_rel <= Gmax,
     g_rel >= 0.]).solve()

# show nominal optimal and reliable optimal cost
print "nom_cost", nom_cost
print "rel_cost", rel_cost
# show nominal optimal and reliable optimal generator powers
print "g_nom:", g_nom.value.A1
print "g_rel:", g_rel.value.A1

```

The following Julia code solves parts (a) and (b).

```

include("rel_pwr_flow_data.jl")

using Convex, SCS

# nominal case
p = Variable(m);
g_nom = Variable(k);
constraints = [];
constraints += A*p == [-g_nom; d];
constraints += abs(p) <= Pmax;
constraints += g_nom <= Gmax;
constraints += g_nom >= 0;
problem = minimize(c'*g_nom, constraints);
solve!(problem, SCSSolver(max_iters=20000));
nom_cost = problem.optval;
println("nominal cost: ", nom_cost)
println("nominal generator powers: ")
println(g_nom.value)

# N-1 case
P = Variable(m,m);

```

```

g_rel = Variable(k);
constraints = [];
constraints += A*P == [-g_rel; d]*ones(1,m);
constraints += abs(P) <= Pmax * ones(1,m);
constraints += diag(P) == 0;
constraints += g_rel <= Gmax;
constraints += g_rel >= 0;
problem = minimize(c'*g_rel, constraints);
solve!(problem, SCSSolver(max_iters=20000));
rel_cost = problem.optval;
println("reliable cost: ", rel_cost)
println("reliable generator powers: ")
println(g_rel.value)

```

16.2 Optimal generator dispatch. In the *generator dispatch problem*, we schedule the electrical output power of a set of generators over some time interval, to minimize the total cost of generation while exactly meeting the (assumed known) electrical demand. One challenge in this problem is that the generators have dynamic constraints, which couple their output powers over time. For example, every generator has a maximum rate at which its power can be increased or decreased.

We label the generators $i = 1, \dots, n$, and the time periods $t = 1, \dots, T$. We let $p_{i,t}$ denote the (nonnegative) power output of generator i at time interval t . The (positive) electrical demand in period t is d_t . The total generated power in each period must equal the demand:

$$\sum_{i=1}^n p_{i,t} = d_t, \quad t = 1, \dots, T.$$

Each generator has a minimum and maximum allowed output power:

$$P_i^{\min} \leq p_{i,t} \leq P_i^{\max}, \quad i = 1, \dots, n, \quad t = 1, \dots, T.$$

The cost of operating generator i at power output u is $\phi_i(u)$, where ϕ_i is an increasing strictly convex function. (Assuming the cost is mostly fuel cost, convexity of ϕ_i says that the thermal efficiency of the generator decreases as its output power increases.) We will assume these cost functions are quadratic: $\phi_i(u) = \alpha_i u + \beta_i u^2$, with α_i and β_i positive.

Each generator has a maximum ramp-rate, which limits the amount its power output can change over one time period:

$$|p_{i,t+1} - p_{i,t}| \leq R_i, \quad i = 1, \dots, n, \quad t = 1, \dots, T-1.$$

In addition, changing the power output of generator i from u_t to u_{t+1} incurs an additional cost $\psi_i(u_{t+1} - u_t)$, where ψ_i is a convex function. (This cost can be a real one, due to increased fuel use during a change of power, or a fictitious one that accounts for the increased maintenance cost or decreased lifetime caused by frequent or large changes in power output.) We will use the power change cost functions $\psi_i(v) = \gamma_i |v|$, where γ_i are positive.

Power plants with large capacity (*i.e.*, P_i^{\max}) are typically more efficient (*i.e.*, have smaller α_i , β_i), but have smaller ramp-rate limits, and higher costs associated with changing power levels. Small

gas-turbine plants ('peakers') are less efficient, have less capacity, but their power levels can be rapidly changed.

The total cost of operating the generators is

$$C = \sum_{i=1}^n \sum_{t=1}^T \phi_i(p_{i,t}) + \sum_{i=1}^n \sum_{t=1}^{T-1} \psi_i(p_{i,t+1} - p_{i,t}).$$

Choosing the generator output schedules to minimize C , while respecting the constraints described above, is a convex optimization problem. The problem data are d_t (the demands), the generator power limits P_i^{\min} and P_i^{\max} , the ramp-rate limits R_i , and the cost function parameters α_i , β_i , and γ_i . We will assume that problem is feasible, and that $p_{i,t}^*$ are the (unique) optimal output powers.

- (a) *Price decomposition.* Show that there are power prices Q_1, \dots, Q_T for which the following holds: For each i , $p_{i,t}^*$ solves the optimization problem

$$\begin{aligned} & \text{minimize} && \sum_{t=1}^T (\phi_i(p_{i,t}) - Q_t p_{i,t}) + \sum_{t=1}^{T-1} \psi_i(p_{i,t+1} - p_{i,t}) \\ & \text{subject to} && P_i^{\min} \leq p_{i,t} \leq P_i^{\max}, \quad t = 1, \dots, T \\ & && |p_{i,t+1} - p_{i,t}| \leq R_i, \quad t = 1, \dots, T-1. \end{aligned}$$

The objective here is the portion of the objective for generator i , minus the revenue generated by the sale of power at the prices Q_t . Note that this problem involves *only* generator i ; it can be solved independently of the other generators (once the prices are known). How would you find the prices Q_t ?

You do not have to give a full formal proof; but you must explain your argument fully. You are welcome to use results from the text book.

- (b) Solve the generator dispatch problem with the data given in `gen_dispatch_data.m`, which gives (fake, but not unreasonable) demand data for 2 days, at 15 minute intervals. This file includes code to plot the demand, optimal generator powers, and prices. (You must replace these variables with their correct values.) Comment on anything you see in your solution that might at first seem odd. Using the prices found, solve the problems in part (a) for the generators separately, to be sure they give the optimal powers (up to some small numerical errors).

Remark. While beyond the scope of this course, we mention that there are very simple price update mechanisms that adjust the prices in such a way that when the generators independently schedule themselves using the prices (as described above), we end up with the total power generated in each period matching the demand, *i.e.*, the optimal solution of the whole (coupled) problem. This gives a decentralized method for generator dispatch.

Solution.

- (a) We start by forming the partial Lagrangian, introducing dual variable $\nu \in \mathbf{R}^T$ for the equality constraint:

$$L(p, \nu) = C + \sum_{t=1}^T \nu_t \left(d_t - \sum_{i=1}^n p_{i,t} \right),$$

restricted to the set of p that satisfy the various constraints. To get the dual function, we minimize over p . But this is separable, so we can minimize the power schedule for each generator separately. For each i we solve the problem

$$\begin{aligned} & \text{minimize} && \sum_{t=1}^T (\phi_i(p_{i,t}) - \nu_t p_{i,t}) + \sum_{t=1}^{T-1} \psi_i(p_{i,t+1} - p_{i,t}) \\ & \text{subject to} && P_i^{\min} \leq p_{i,t} \leq P_i^{\max}, \quad t = 1, \dots, T \\ & && |p_{i,t+1} - p_{i,t}| \leq R_i, \quad t = 1, \dots, T-1. \end{aligned}$$

Now we observe that strong duality holds, since the constraints are linear and the problem is feasible. This means that when ν is (dual) optimal, the dual function equals the optimal (primal) objective function. Since the Lagrangian, with optimal dual variable, is strictly convex in p , it has a unique solution; this must be the solution of the original problem. Thus we see that the prices are none other than optimal dual variables for the power balance equations.

- (b) The code below solves the problem.

```
% solve generator dispatch problem
gen_dispatch_data;

cvx_begin
variable p(n,T);
dual variable Q;
p <= Pmax'*ones(1,T);
p >= Pmin'*ones(1,T);
Q: sum(p,1) == d; % get lagrange multipliers, which are the prices
abs(p(:,2:T)-p(:,1:T-1)) <= R'*ones(1,T-1);
power_cost= sum(alpha*p+beta*(p.^2));
change_power_cost = sum(gamma*abs(p(:,2:T)-p(:,1:T-1)));
minimize (power_cost+change_power_cost)
cvx_end

subplot(3,1,1)
plot(t,d);
title('demand')
subplot(3,1,2)
plot(t,p);
title('generator powers')
subplot(3,1,3)
plot(t,Q);
title('power prices')

print -depsc gen_dispatch

% now let's solve the problems separately, with the given prices
% we'll solve in one big problem, which is separable, so we're really
% solving n separate problems, one for each generator
```

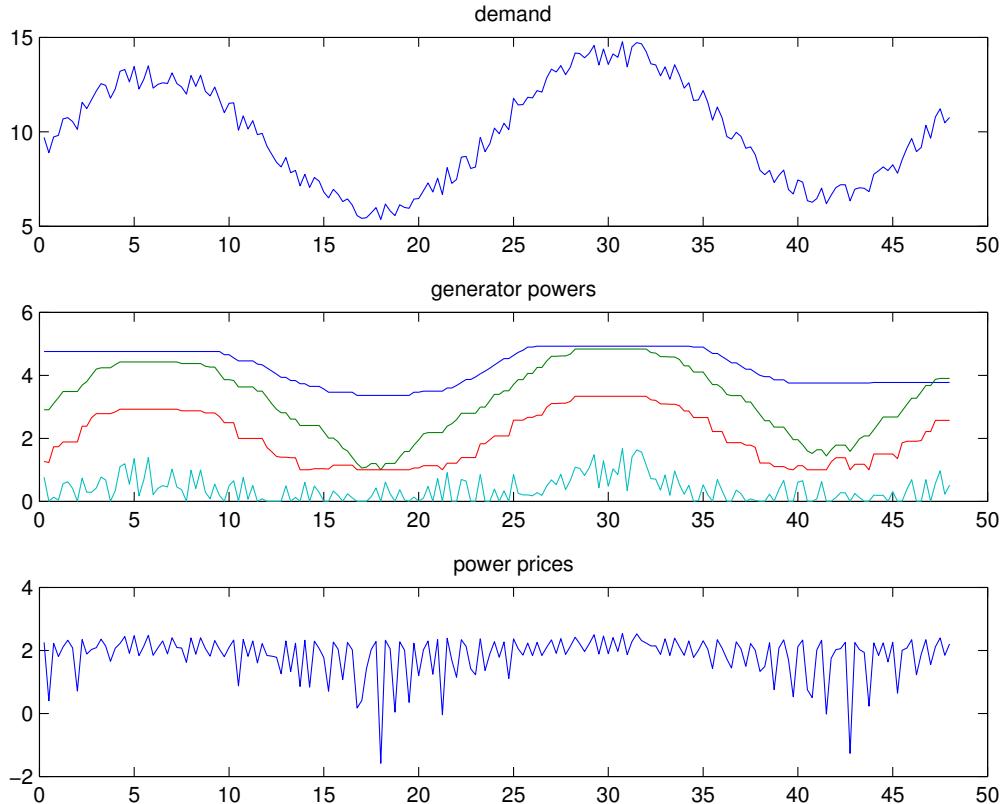
```

cvx_begin
variable pp(n,T);
pp <= Pmax'*ones(1,T);
pp >= Pmin'*ones(1,T);
%sum(pp,1) == d;
abs(pp(:,2:T)-pp(:,1:T-1)) <= R'*ones(1,T-1);
power_cost= sum(alpha*pp+beta*(pp.^2));
change_power_cost = sum(gamma*abs(pp(:,2:T)-pp(:,1:T-1)));
minimize (power_cost+change_power_cost-sum(pp*Q'))
cvx_end

rel_error = norm(pp-p)/norm(p)

```

This produces the following plots. Note that in two periods, power prices become *negative*. This is correct: the negative price is an incentive to the generators to rapidly ramp down power levels at that time period, which has low demand. The relative error in computing the powers using the prices found is on the order of 10^{-7} .



16.3 Optimizing a portfolio of energy sources. We have n different energy sources, such as coal-fired plants, several wind farms, and solar farms. Our job is to size each of these, *i.e.*, to choose its capacity. We will denote by c_i the capacity of plant i ; these must satisfy $c_i^{\min} \leq c_i \leq c_i^{\max}$, where c_i^{\min} and c_i^{\max} are given minimum and maximum values.

Each generation source has a cost to build and operate (including fuel, maintenance, government subsidies and taxes) over some time period. We lump these costs together, and assume that the cost is proportional to c_i , with (given) coefficient b_i . Thus, the total cost to build and operate the energy sources is $b^T c$ (in, say, \$/hour).

Each generation source is characterized by an availability a_i , which is a random variable with values in $[0, 1]$. If source i has capacity c_i , then the power available from the plant is $c_i a_i$; the total power available from the portfolio of energy sources is $c^T a$, which is a random variable. A coal fired plant has $a_i = 1$ almost always, with $a_i < 1$ when one of its units is down for maintenance. A wind farm, in contrast, is characterized by strong fluctuations in availability with $a_i = 1$ meaning a strong wind is blowing, and $a_i = 0$ meaning no wind is blowing. A solar farm has $a_i = 1$ only during peak sun hours, with no cloud cover; at other times (such as night) we have $a_i = 0$.

Energy demand $d \in \mathbf{R}_+$ is also modeled as a random variable. The components of a (the availabilities) and d (the demand) are *not* independent. Whenever the total power available falls short of the demand, the additional needed power is generated by (expensive) peaking power plants at a fixed positive price p . The average cost of energy produced by the peakers is

$$\mathbf{E} p(d - c^T a)_+,$$

where $x_+ = \max\{0, x\}$. This average cost has the same units as the cost $b^T c$ to build and operate the plants.

The objective is to choose c to minimize the overall cost

$$C = b^T c + \mathbf{E} p(d - c^T a)_+.$$

Sample average approximation. To solve this problem, we will minimize a cost function based on a sample average of peaker cost,

$$C^{\text{sa}} = b^T c + \frac{1}{N} \sum_{j=1}^N p(d^{(j)} - c^T a^{(j)})_+$$

where $(a^{(j)}, d^{(j)})$, $j = 1, \dots, N$, are (given) samples from the joint distribution of a and d . (These might be obtained from historical data, weather and demand forecasting, and so on.)

Validation. After finding an optimal value of c , based on the set of samples, you should double check or validate your choice of c by evaluating the overall cost on another set of (validation) samples, $(\tilde{a}^{(j)}, \tilde{d}^{(j)})$, $j = 1, \dots, N^{\text{val}}$,

$$C^{\text{val}} = b^T c + \frac{1}{N^{\text{val}}} \sum_{j=1}^{N^{\text{val}}} p(\tilde{d}^{(j)} - c^T \tilde{a}^{(j)})_+.$$

(These could be another set of historical data, held back for validation purposes.) If $C^{\text{sa}} \approx C^{\text{val}}$, our confidence that each of them is approximately the optimal value of C is increased.

Finally we get to the problem. Get the data in `energy_portfolio_data.m`, which includes the required problem data, and the samples, which are given as a $1 \times N$ row vector `d` for the scalars

$d^{(j)}$, and an $n \times N$ matrix \mathbf{A} for $a^{(j)}$. A second set of samples is given for validation, with the names `d_val` and `A_val`.

Carry out the optimization described above. Give the optimal cost obtained, C^{sa} , and compare to the cost evaluated using the validation data set, C^{val} .

Compare your solution with the following naive ('certainty-equivalent') approach: Replace a and d with their (sample) means, and then solve the resulting optimization problem. Give the optimal cost obtained, C^{ce} (using the average values of a and d). Is this a lower bound on the optimal value of the original problem? Now evaluate the cost for these capacities on the validation set, $C^{\text{ce},\text{val}}$. Make a brief statement.

Solution. The problem can be formulated as

$$\begin{aligned} & \text{minimize} && b^T c + \mathbf{E} p(d - c^T a)_+ \\ & \text{subject to} && c^{\min} \preceq c \preceq c^{\max}, \end{aligned}$$

with variable $c \in \mathbf{R}^n$. It is convex since $p(d - c^T a)_+$ is a convex function of c for each a and d (using $p > 0$), and convexity is preserved under expectation.

The sample average approximation problem is

$$\begin{aligned} & \text{minimize} && b^T c + (1/N) \sum_{j=1}^N p(d^{(j)} - c^T a^{(j)})_+ \\ & \text{subject to} && c^{\min} \preceq c \preceq c^{\max}, \end{aligned}$$

with variable c .

The naive ('certainty-equivalent') problem is

$$\begin{aligned} & \text{minimize} && b^T c + p(\bar{d} - c^T \bar{a})_+ \\ & \text{subject to} && c^{\min} \preceq c \preceq c^{\max}, \end{aligned}$$

with variable c , where

$$\bar{d} = (1/N) \sum_{j=1}^N d^{(j)}, \quad \bar{a} = (1/N) \sum_{j=1}^N a^{(j)}.$$

By Jensen's inequality, the objective function of this problem is less than or equal to the objective for the original problem, so the optimal cost obtained by solving the naive problem is a lower bound on the optimal cost of the original problem.

The code to solve this problem is shown below. We obtain the following numerical results. The sampled problem has cost $C_{\text{sa}} = 10176$. Evaluating the objective for the validation set gives $C^{\text{val}} = 9906$. Since these two values are reasonably close we can guess that our solution is close to optimal for the original problem. The optimal cost for the naive approach is $C^{\text{ce}} = 6695$, which is indeed less than the value obtained by the sample average approximation. Evaluating the objective on the validation set gives $C^{\text{ce},\text{val}} = 12441$, which shows that the naive approach is not very good. By solving the stochastic problem (well, OK, approximately), we've saved around 22% in cost.

The following code solves the problem.

```
% solution to energy portfolio optimization problem
energy_portfolio_data;
```

```

% sample average approximation problem
cvx_begin
    variable c(n)
    minimize (b'*c + (1/N)*sum(p*pos(d-c'*A)));
    cmin <= c;
    c <= cmax;
cvx_end

Csa = cvx_optval % sample average optimal cost

% now evaluate objective on validation data
Cval = b'*c + (1/Nval)*sum(p*pos(dval-c'*Aval))

% naive method
dbar = (1/N)*sum(d);
Abar = (1/N)*sum(A,2);
cvx_begin
    variable cce(n)
    minimize (b'*cce+ p*pos(dbar-cce'*Abar))
    cmin <= cce;
    cce <= cmax;
cvx_end

Cce = cvx_optval % naive method optimal cost

% now evaluate objective on validation data
Cceval = b'*cce + (1/Nval)*sum(p*pos(dval-cce'*Aval))

```

16.4 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, \dots, 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 $f : \mathbf{R} \rightarrow \mathbf{R}$ is positive, increasing, and convex. Therefore, the total energy consumed is

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

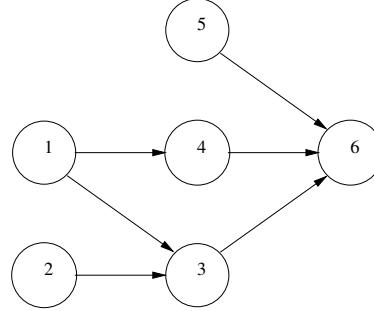
(Here we ignore the energy used to transfer data between processors, and assume the processors are powered down when they are not active.)

There is a set of *precedence constraints* for the tasks, which is a set of m ordered pairs $\mathcal{P} \subseteq \{1, \dots, n\} \times \{1, \dots, n\}$. 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} , $\alpha_1, \dots, \alpha_n$, E_{\max} , and the function f . The variables are s_1, \dots, 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 `proc_speed_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

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

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

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

We formulate the problem as

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

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

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

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

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

(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);
```

```

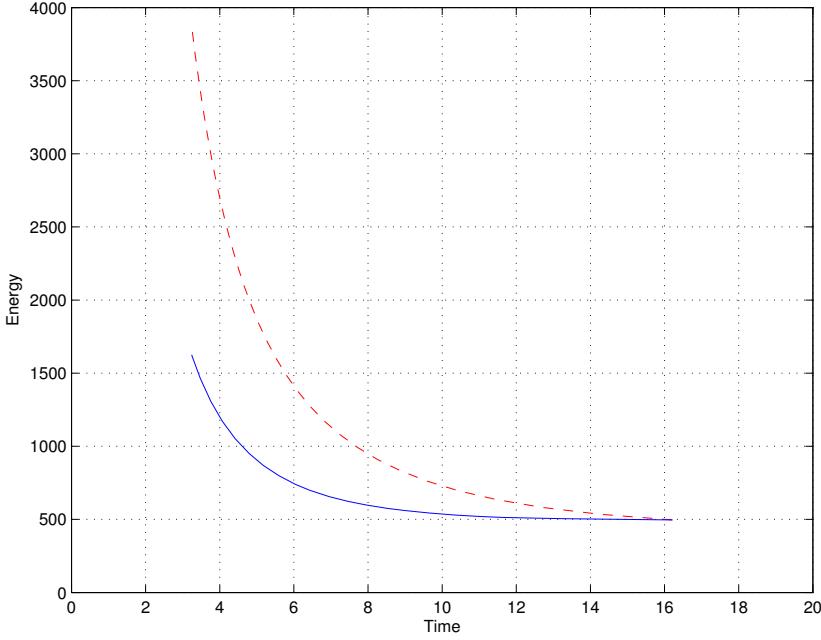
T = max(t);
Eopt = [Eopt E];
Topt = [Topt T];
end

% 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

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

To get the tradeoff curve we can solve the problem

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

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.

16.5 Minimum energy processor speed scheduling. A single processor can adjust its speed in each of T time periods, labeled $1, \dots, T$. Its speed in period t will be denoted s_t , $t = 1, \dots, T$. The speeds must lie between given (positive) minimum and maximum values, S^{\min} and S^{\max} , respectively, and must satisfy a slew-rate limit, $|s_{t+1} - s_t| \leq R$, $t = 1, \dots, T-1$. (That is, R is the maximum allowed period-to-period change in speed.) The energy consumed by the processor in period t is given by $\phi(s_t)$, where $\phi : \mathbf{R} \rightarrow \mathbf{R}$ is increasing and convex. The total energy consumed over all the periods is $E = \sum_{t=1}^T \phi(s_t)$.

The processor must handle n jobs, labeled $1, \dots, n$. Each job has an availability time $A_i \in \{1, \dots, T\}$, and a deadline $D_i \in \{1, \dots, T\}$, with $D_i \geq A_i$. The processor cannot start work on job i until period $t = A_i$, and must complete the job by the end of period D_i . Job i involves a (nonnegative) total work W_i . You can assume that in each time period, there is at least one job available, *i.e.*, for each t , there is at least one i with $A_i \leq t$ and $D_i \geq t$.

In period t , the processor allocates its effort across the n jobs as θ_t , where $\mathbf{1}^T \theta_t = 1$, $\theta_t \succeq 0$. Here θ_{ti} (the i th component of θ_t) gives the fraction of the processor effort devoted to job i in period t . Respecting the availability and deadline constraints requires that $\theta_{ti} = 0$ for $t < A_i$ or $t > D_i$. To complete the jobs we must have

$$\sum_{t=A_i}^{D_i} \theta_{ti} s_t \geq W_i, \quad i = 1, \dots, n.$$

- (a) Formulate the problem of choosing the speeds s_1, \dots, s_T , and the allocations $\theta_1, \dots, \theta_T$, in order to minimize the total energy E , as a convex optimization problem. The problem data are S^{\min} , S^{\max} , R , ϕ , and the job data, A_i , D_i , W_i , $i = 1, \dots, n$. Be sure to justify any change of variables, or introduction of new variables, that you use in your formulation.
- (b) Carry out your method on the problem instance described in `proc_sched_data.m`, with quadratic energy function $\phi(s_t) = \alpha + \beta s_t + \gamma s_t^2$. (The parameters α , β , and γ are given in the data file.) Executing this file will also give a plot showing the availability times and deadlines for the jobs.

Give the energy obtained by your speed profile and allocations. Plot these using the command `bar((s*ones(1,n)).*theta, 1, 'stacked')`, where s is the $T \times 1$ vector of speeds, and θ is the $T \times n$ matrix of allocations with components θ_{ti} . This will show, at each time period, how much effective speed is allocated to each job. The top of the plot will show the speed s_t . (You don't need to turn in a color version of this plot; B&W is fine.)

Solution. The trick is to work with the variables $x_{ti} = \theta_{ti} s_t$, which must be nonnegative. Let $X \in \mathbf{R}^{T \times n}$ denote the matrix with components x_{ti} . The job completion constraint can be expressed as $X^T \mathbf{1} \succeq W$. The availability and deadline constraints can be expressed as $x_{ti} = 0$ for $t < A_i$ or $t > D_i$, which are linear constraints. The speed can be expressed as $s = X \mathbf{1}$, a linear function of our variable X . The objective E is clearly a convex function of s (and so, also of X).

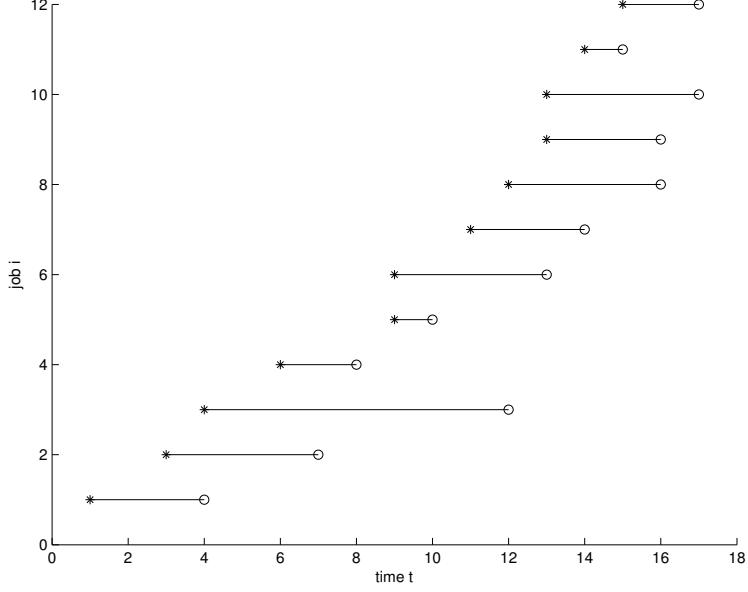


Figure 17: Job availability diagram. Stars indicate the time in which each job becomes available. Open circles indicate the required end of the job, which is at the end of the deadline time period.

Our convex optimization problem is

$$\begin{aligned}
 & \text{minimize} \quad E = \sum_{t=1}^T \phi(s_t) \\
 & \text{subject to} \quad S^{\min} \preceq s \preceq S^{\max}, \quad s = X\mathbf{1}, \quad X^T \mathbf{1} \succeq W, \quad X \succeq 0 \\
 & \quad |s_{t+1} - s_t| \leq R, \quad t = 1, \dots, T-1 \\
 & \quad X_{ti} = 0, \quad t = 1, \dots, A_i - 1, \quad i = 1, \dots, n \\
 & \quad X_{ti} = 0, \quad t = D_i + 1, \dots, T, \quad i = 1, \dots, n
 \end{aligned}$$

with variables $s \in \mathbf{R}^T$ and $X \in \mathbf{R}^{T \times n}$. All generalized inequalities here, including $X \succeq 0$, are elementwise nonnegativity. Evidently this is a convex problem.

Once we find an optimal X^* and s^* , we can recover θ_t^* as

$$\theta_{ti}^* = (1/s_t^*)x_{ti}^*, \quad t = 1, \dots, T, \quad i = 1, \dots, n.$$

For our data, the availability and deadlines of jobs are plotted in figure 17. The job duration lines show a start time at the beginning of the available time period and a termination time at the very end of the deadline time period. Thus, the length of the line shows the actual duration within which the job is allowed to be completed.

The code that solves the problem is as follows.

```

proc_sched_data;

cvx_begin
variable X(T,n)

```

```

X >= 0;
s = sum(X')';
minimize(sum(alpha+beta*s+gamma*square(s)))
s >= Smin;
s <= Smax;
abs(s(2:end)-s(1:end-1))<=R; % slew rate constraint

% start/stop constraints
for i=1:n
    for t=1:A(i)-1
        X(t,i)==0;
    end
    for t=D(i)+1:T
        X(t,i)==0;
    end
end

sum(X)>=W';

cvx_end
theta = X./(s*ones(1,n));

figure;
bar((s*ones(1,n)).*theta,1,'stacked');
xlabel('t');
ylabel('st');

```

The optimal total energy is $E = 162.125$, which is obtained by allocating speeds according to the plot shown in figure 18.

16.6 AC power flow analysis via convex optimization. This problem concerns an AC (alternating current) power system consisting of m transmission lines that connect n nodes. We describe the topology by the node-edge incidence matrix $A \in \mathbf{R}^{n \times m}$, where

$$A_{ij} = \begin{cases} +1 & \text{line } j \text{ leaves node } i \\ -1 & \text{line } j \text{ enters node } i \\ 0 & \text{otherwise.} \end{cases}$$

The power flow on line j is p_j (with positive meaning in the direction of the line as defined in A , negative meaning power flow in the opposite direction).

Node i has voltage phase angle ϕ_i , and external power input s_i . (If a generator is attached to node i we have $s_i > 0$; if a load is attached we have $s_i < 0$; if the node has neither, $s_i = 0$.) Neglecting power losses in the lines, and assuming power is conserved at each node, we have $Ap = s$. (We must have $\mathbf{1}^T s = 0$, which means that the total power pumped into the network by generators balances the total power pulled out by the loads.)

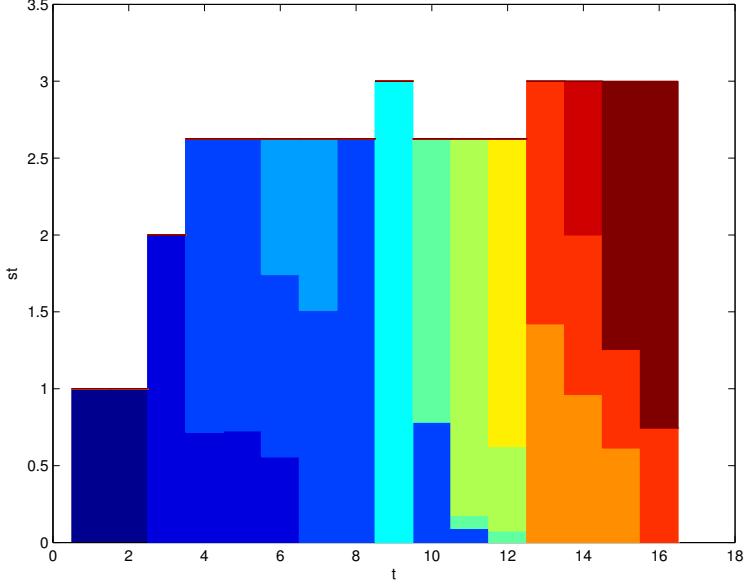


Figure 18: Optimal speed allocation profile. The top of the plot gives the processor speed. The colored regions indicate the portion of the speed allocated to a particular job at each time.

The line power flows are a nonlinear function of the difference of the phase angles at the nodes they connect to:

$$p_j = \kappa_j \sin(\phi_k - \phi_l),$$

where line j goes from node k to node l . Here κ_j is a known positive constant (related to the inductance of the line). We can write this in matrix form as $p = \text{diag}(\kappa) \sin(A^T \phi)$, where \sin is applied elementwise.

The *DC power flow equations* are

$$Ap = s, \quad p = \text{diag}(\kappa) \sin(A^T \phi).$$

In the power analysis problem, we are given s , and want to find p and ϕ that satisfy these equations. We are interested in solutions with voltage phase angle differences that are smaller than $\pm 90^\circ$. (Under normal conditions, real power lines are never operated with voltage phase angle differences more than $\pm 20^\circ$ or so.)

You will show that the DC power flow equations can be solved by solving the convex optimization problem

$$\begin{aligned} & \text{minimize} && \sum_{i=j}^m \psi_j(p_j) \\ & \text{subject to} && Ap = s, \end{aligned}$$

with variable p , where

$$\psi_j(u) = \int_0^u \sin^{-1}(v/\kappa_j) dv = u \sin^{-1}(u/\kappa_j) + \kappa_j (\sqrt{1 - (u/\kappa_j)^2} - 1),$$

with domain $\text{dom } \psi_j = (-\kappa_j, \kappa_j)$. (The second expression will be useless in this problem.)

- (a) Show that the problem above is convex.
- (b) Suppose the problem above has solution p^* , with optimal dual variable ν^* associated with the equality constraint $Ap = s$. Show that $p^*, \phi = \nu^*$ solves the DC power flow equation. Hint. Write out the optimality conditions for the problem above.

Solution. We have $\psi'_i(u) = \sin^{-1}(u/\kappa_j)$, which is a strictly increasing function of u , so ψ_j is strictly convex.

The optimality conditions for the problem above are

$$Ap^* = s, \quad \nabla\psi(p^*) - A^T\nu^* = 0.$$

The second equation (dual feasibility) can be written as

$$\psi'_j(p_j^*) = a_j^T \nu^*, \quad j = 1, \dots, m,$$

where a_j is the j th column of A . Thus we have

$$\sin^{-1}(p_j^*/\kappa_j) = a_j^T \nu^*, \quad j = 1, \dots, m,$$

and so

$$p_j^* = \kappa_j \sin(a_j^T \nu^*), \quad j = 1, \dots, m.$$

But this is exactly the same as $p^* = \text{diag}(\kappa) \sin(A^T \phi)$, with $\phi = \nu^*$.

16.7 Power transmission with losses. A power transmission grid is modeled as a set of n nodes and m directed edges (which represent transmission lines), with topology described by the node-edge incidence matrix $A \in \mathbf{R}^{n \times m}$, defined by

$$A_{ij} = \begin{cases} +1 & \text{edge } j \text{ enters node } i, \\ -1 & \text{edge } j \text{ leaves node } i, \\ 0 & \text{otherwise.} \end{cases}$$

We let $p_j^{\text{in}} \geq 0$ denote the power that flows into the tail of edge j , and $p_j^{\text{out}} \geq 0$ the power that emerges from the head of edge j , for $j = 1, \dots, m$. Due to transmission losses, the power that flows into each edge is more than the power that emerges:

$$p_j^{\text{in}} = p_j^{\text{out}} + \alpha(L_j/R_j^2)(p_j^{\text{out}})^2, \quad j = 1, \dots, m,$$

where $L_j > 0$ is the length of transmission line j , $R_j > 0$ is the radius of the conductors on line j , and $\alpha > 0$ is a constant. (The second term on the righthand side above is the transmission line power loss.) In addition, each edge has a maximum allowed input power, that also depends on the conductor radius: $p_j^{\text{in}} \leq \sigma R_j^2$, $j = 1, \dots, m$, where $\sigma > 0$ is a constant.

Generators are attached to nodes $i = 1, \dots, k$, and loads are attached to nodes $i = k+1, \dots, n$. We let g_i denote the (nonnegative) power injected into node i by its generator, for $i = 1, \dots, k$. We let l_i denote the (nonnegative) power pulled from node i by the load, for $i = k+1, \dots, n$. These load powers are known and fixed.

We must have power balance at each node. For $i = 1, \dots, k$, the sum of all power entering the node from incoming transmission lines, plus the power supplied by the generator, must equal the sum of all power leaving the node on outgoing transmission lines:

$$\sum_{j \in \mathcal{E}(i)} p_j^{\text{out}} + g_i = \sum_{j \in \mathcal{L}(i)} p_j^{\text{in}}, \quad i = 1, \dots, k,$$

where $\mathcal{E}(i)$ ($\mathcal{L}(i)$) is the set of edge indices for edges entering (leaving) node i . For the load nodes $i = k+1, \dots, n$ we have a similar power balance condition:

$$\sum_{j \in \mathcal{E}(i)} p_j^{\text{out}} = \sum_{j \in \mathcal{L}(i)} p_j^{\text{in}} + l_i, \quad i = k+1, \dots, n.$$

Each generator can vary its power g_i over a given range $[0, G_i^{\max}]$, and has an associated cost of generation $\phi_i(g_i)$, where ϕ_i is convex and strictly increasing, for $i = 1, \dots, k$.

- (a) *Minimum total cost of generation.* Formulate the problem of choosing generator and edge input and output powers, so as to minimize the total cost of generation, as a convex optimization problem. (All other quantities described above are known.) Be sure to explain any additional variables or terms you introduce, and to justify any transformations you make.

Hint: You may find the matrices $A_+ = (A)_+$ and $A_- = (-A)_+$ helpful in expressing the power balance constraints.

- (b) *Marginal cost of power at load nodes.* The (marginal) cost of power at node i , for $i = k+1, \dots, n$, is the partial derivative of the minimum total power generation cost, with respect to varying the load power l_i . (We will simply assume these partial derivatives exist.) Explain how to find the marginal cost of power at node i , from your formulation in part (a).
- (c) *Optimal sizing of lines.* Now suppose that you can optimize over generator powers, edge input and output powers (as above), and the power line radii R_j , $j = 1, \dots, m$. These must lie between given limits, $R_j \in [R_j^{\min}, R_j^{\max}]$ ($R_j^{\min} > 0$), and we must respect a total volume constraint on the lines,

$$\sum_{j=1}^m L_j R_j^2 \leq V^{\max}.$$

Formulate the problem of choosing generator and edge input and output powers, as well as power line radii, so as to minimize the total cost of generation, as a convex optimization problem. (Again, explain anything that is not obvious.)

- (d) *Numerical example.* Using the data given in `ptrans_loss_data.m`, find the minimum total generation cost and the marginal cost of power at nodes $k+1, \dots, n$, for the case described in parts (a) and (b) (i.e., using the fixed given radii R_j), and also for the case described in part (c), where you are allowed to change the transmission line radii, keeping the same total volume as the original lines. For the generator costs, use the quadratic functions

$$\phi_i(g_i) = a_i g_i + b_i g_i^2, \quad i = 1, \dots, k,$$

where $a, b \in \mathbf{R}_+^k$. (These are given in the data file.)

Remark: In the m-file, we give you a load vector $l \in \mathbf{R}^{n-k}$. For consistency, the i th entry of this vector corresponds to the load at node $k+i$.

Solution.

(a) The problem as stated,

$$\begin{aligned} & \text{minimize} && \sum_{i=1}^k \phi_i(g_i) \\ & \text{subject to} && p_j^{\text{in}} = p_j^{\text{out}} + \alpha(L_j/R_j^2)(p_j^{\text{out}})^2, \quad j = 1, \dots, m \\ & && 0 \leq p_j^{\text{in}} \leq \sigma R_j^2, \quad j = 1, \dots, m \\ & && 0 \leq p^{\text{out}} \\ & && 0 \leq g \leq G^{\max} \\ & && (A_+ p^{\text{out}})_i + g_i = (A_- p^{\text{in}})_i, \quad i = 1, \dots, k \\ & && (A_+ p^{\text{out}})_i = (A_- p^{\text{in}})_i + l_i, \quad i = k+1, \dots, n, \end{aligned}$$

with variables p^{in} , p^{out} , and g , is not convex, since the power line equations are not affine. However, the the power line loss equations can be relaxed to *inequalities*,

$$p_j^{\text{in}} \geq p_j^{\text{out}} + \alpha(L_j/R_j^2)(p_j^{\text{out}})^2, \quad j = 1, \dots, m,$$

which are convex, and we can show that they must be tight at the optimal point. First we argue intuitively. These inequalities allow us the option of putting more energy into a power line than is needed; in other words, it allows us to throw away energy. Energy is valuable, so we'd be foolish to do this. That's not really an argument, though. A more formal argument goes like this. Suppose that the line loss inequality for line j holds strictly. Then we can reduce the power input to line j without affecting its power output. Now trace a directed path back to a generator. We move along the path from the node that feeds line j back to the generator. Along each line, we can reduce the input and output power slightly, while maintaining power balance. Finally, we reduce the generator power as well, which reduces our cost function, since ϕ_i are strictly increasing. Thus, we have constructed a new feasible point with lower cost, which means that the powers were not optimal.

In summary, we solve the problem by solving the convex problem

$$\begin{aligned} & \text{minimize} && \sum_{i=1}^k \phi_i(g_i) \\ & \text{subject to} && p_j^{\text{in}} \geq p_j^{\text{out}} + \alpha(L_j/R_j^2)(p_j^{\text{out}})^2, \quad j = 1, \dots, m \\ & && 0 \leq p_j^{\text{in}} \leq \sigma R_j^2, \quad j = 1, \dots, m \\ & && 0 \leq p^{\text{out}} \\ & && 0 \leq g \leq G^{\max} \\ & && (A_+ p^{\text{out}})_i + g_i = (A_- p^{\text{in}})_i, \quad i = 1, \dots, k \\ & && (A_+ p^{\text{out}})_i = (A_- p^{\text{in}})_i + l_i, \quad i = k+1, \dots, n. \end{aligned}$$

At the optimal point, the power line inequalities here will be tight, *i.e.*, they will hold with equality.

(b) The marginal cost of power load at node i is exactly equal to the negative of the Lagrange multiplier associated with the constraint

$$(A_+ p^{\text{out}})_i = (A_- p^{\text{in}})_i + l_i,$$

since perturbing this equality constraint is the same as perturbing l_i , the load power.

- (c) The key insight is that the problem is convex in the cross-sectional area, $s_j = R_j^2$ (neglecting the constant π), but not R_j . With the variable s instead of R , our optimization problem becomes

$$\begin{aligned} & \text{minimize} && \sum_{i=1}^k \phi_i(g_i) \\ & \text{subject to} && p_j^{\text{in}} \geq p_j^{\text{out}} + \alpha L_j (p_j^{\text{out}})^2 / s_j, \quad j = 1, \dots, m \\ & && (R_j^{\text{min}})^2 \leq s_j \leq (R_j^{\text{max}})^2, \quad j = 1, \dots, m \\ & && 0 \leq p_j^{\text{in}} \leq \sigma s_j, \quad j = 1, \dots, m \\ & && 0 \preceq p^{\text{out}} \\ & && 0 \preceq g \preceq G^{\text{max}} \\ & && (A_+ p^{\text{out}})_i + g_i = (A_- p^{\text{in}})_i, \quad i = 1, \dots, k \\ & && (A_+ p^{\text{out}})_i = (A_- p^{\text{in}})_i + l_i, \quad i = k+1, \dots, n \\ & && L^T s \leq V^{\text{max}}, \end{aligned}$$

with the $s \geq 0$ constraint implicit from the first set of constraints. This is a convex problem in the variables p^{in} , p^{out} , and s . We recover the optimal R_j via $R_j^* = \sqrt{s_j^*}$.

- (d) We find that the optimal cost of power generation for the unoptimized network is 113.96 and the optimal cost of power generation for the optimized network is 99.82. We also find that the marginal cost of load powers for the unoptimized network are

`marginalUnif =`

```
14.4256
15.7128
14.3973
12.7591
11.8504
15.3929
11.4307
14.3772,
```

while the network optimized marginal costs are

`marginalOpt =`

```
7.1644
12.6325
6.5118
11.7082
5.8231
13.0379
5.8685
12.3710.
```

Lastly, the unoptimized radii R vs the optimized radii R_{opt} are

`[R Ropt] =`

1.0000	1.4731
1.0000	0.5000

```

1.0000    1.3480
1.0000    0.7491
1.0000    0.5000
1.0000    0.5000
1.0000    0.5000
1.0000    0.5000
1.0000    0.5000
1.0000    0.5000
1.0000    1.2073
1.0000    0.5000
1.0000    1.3503
1.0000    1.3478
1.0000    0.5000
1.0000    1.5000
1.0000    1.3828
1.0000    0.5000
1.0000    0.5000
1.0000    1.5000.

```

The following code solves the problem

```

% optimal power transmission with losses

ptrans_loss_data;

Ap = max(A,0);
Am = max(-A,0);

% parts a and b
cvx_begin
    variables p_in(m) p_out(m) g(k);
    dual variable lambda;
    minimize (a'*g + sum(b.*(g.^2)))
    subject to
        p_in <= sigma*R.^2;
        p_in >= 0;
        p_out >= 0;
        g <= Gmax;
        g >= 0;
        for j = 1:m
            p_in(j) >= p_out(j) + alpha*L(j)/(R(j)^2)*p_out(j).^2;
        end
        Ap(1:k,:)*p_out + g == Am(1:k,:)*p_in;
        lambda : Ap(k+1:n,:)*p_out == Am(k+1:n,:)*p_in + l;
cvx_end

pinUnif = p_in;
poutUnif = p_out;

```

```

gUnif = g;
valUnif = cvx_optval;
marginalUnif = lambda;

% part c
cvx_begin
    variables p_in(m) p_out(m) g(k) s(m);
    dual variable lambda;
    minimize (a'*g + sum(b.*(g.^2)))
    subject to
        p_in <= sigma*s;
        p_in >= 0;
        p_out >= 0;
        g <= Gmax;
        g >= 0;
        s <= Rmax.^2;
        s >= Rmin.^2;
        L'*s <= Vmax;
        for j = 1:m
            p_in(j) >= p_out(j) + alpha*L(j)*quad_over_lin(p_out(j),s(j));
        end
        Ap(1:k,:)*p_out + g == Am(1:k,:)*p_in;
        lambda : Ap(k+1:n,:)*p_out == Am(k+1:n,:)*p_in + l;
cvx_end

pinOpt = p_in;
poutOpt = p_out;
gOpt = g;
valOpt = cvx_optval;
marginalOpt = lambda;
Ropt = sqrt(s);

```

- 16.8 Utility/power trade-off in a wireless network.** In this problem we explore the trade-off between total utility and total power usage in a wireless network in which the link transmit powers can be adjusted. The network consists of a set of nodes and a set of links over which data can be transmitted. There are n routes, each corresponding to a sequence of links from a source to a destination node. Route j has a data flow rate $f_j \in \mathbf{R}_+$ (in units of bits/sec, say). The total utility (which we want to maximize) is

$$U(f) = \sum_{j=1}^n U_j(f_j),$$

where $U_j : \mathbf{R} \rightarrow \mathbf{R}$ are concave increasing functions.

The network topology is specified by the routing matrix $R \in \mathbf{R}^{m \times n}$, defined as

$$R_{ij} = \begin{cases} 1 & \text{route } j \text{ passes over link } i \\ 0 & \text{otherwise.} \end{cases}$$

The total traffic on a link is the sum of the flows that pass over the link. The traffic (vector) is thus $t = Rf \in \mathbf{R}^m$. The traffic on each link cannot exceed the capacity of the link, *i.e.*, $t \preceq c$, where $c \in \mathbf{R}_+^m$ is the vector of link capacities.

The link capacities, in turn, are functions of the link transmit powers, given by $p \in \mathbf{R}_+^m$, which cannot exceed given limits, *i.e.*, $p \preceq p^{\max}$. These are related by

$$c_i = \alpha_i \log(1 + \beta_i p_i),$$

where α_i and β_i are positive parameters that characterize link i . The second objective (which we want to minimize) is $P = \mathbf{1}^T p$, the total (transmit) power.

- (a) Explain how to find the optimal trade-off curve of total utility and total power, using convex or quasiconvex optimization.
- (b) Plot the optimal trade-off curve for the problem instance with $m = 20$, $n = 10$, $U_j(x) = \sqrt{x}$ for $j = 1, \dots, n$, $p_i^{\max} = 10$, $\alpha_i = \beta_i = 1$ for $i = 1, \dots, m$, and network topology generated using

```
rand('seed',3);
R = round(rand(m,n));
```

Your plot should have the total power on the horizontal axis.

Solution.

- (a) We can formulate the problem as follows

$$\begin{aligned} & \text{maximize} && \sum_{j=1}^n U_j(f_j) - \lambda \mathbf{1}^T p \\ & \text{subject to} && t = Rf, \quad p \preceq p^{\max}, \\ & && t_i \leq \alpha_i \log(1 + \beta_i p_i), \quad i = 1, \dots, m, \end{aligned}$$

with variables $t, p \in \mathbf{R}^m$, $f \in \mathbf{R}^n$, and tradeoff parameter λ . The objective and constraints are convex. To compute the optimal trade-off curve, we solve the optimization problem over instances with $0 < \lambda < \infty$.

- (b) The following Matlab script solves the problem and plots the tradeoff curve.

```
clear all, close all;
% generate data
m = 20; n = 10;
pmax = 10;
alpha = 1; beta = 1;
rand('seed', 3);
R = round(rand(m,n));

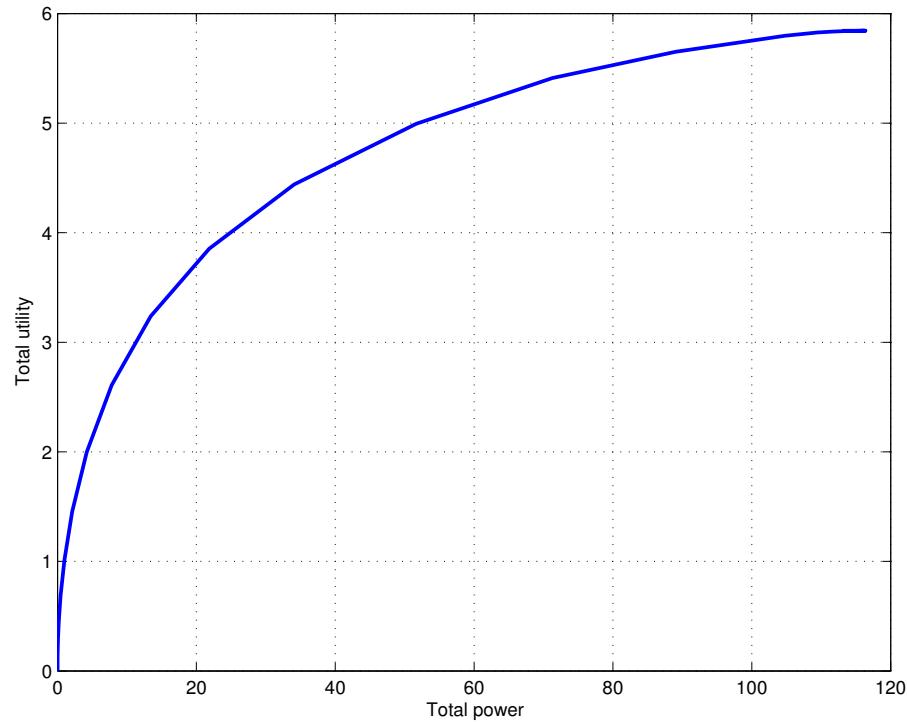
% tradeoff curve
utility = [];
power = [];
num_evals = 50;
```

```

for lambda=logspace(-6,3,num_evals)
    % compute optimal powers
    cvx_begin;
        variables t(m) p(m) f(n);
        maximize( sum(sqrt(f)) - lambda*sum(p) );
        subject to
            t == R*f;
            p <= pmax;
            t <= alpha * log(1 + beta*p);
    cvx_end;
    utility = [utility, sum(sqrt(f))];
    power = [power, sum(p)];
end

% plot results
plot(power, utility);
xlabel('Total power');
ylabel('Total utility');
grid on;
print -depsc util_pwr_tradeoff

```



16.9 Energy storage trade-offs. We consider the use of a storage device (say, a battery) to reduce the total cost of electricity consumed over one day. We divide the day into T time periods, and let p_t denote the (positive, time-varying) electricity price, and u_t denote the (nonnegative) usage or consumption, in period t , for $t = 1, \dots, T$. Without the use of a battery, the total cost is $p^T u$.

Let q_t denote the (nonnegative) energy stored in the battery in period t . For simplicity, we neglect energy loss (although this is easily handled as well), so we have $q_{t+1} = q_t + c_t$, $t = 1, \dots, T - 1$, where c_t is the charging of the battery in period t ; $c_t < 0$ means the battery is discharged. We will require that $q_1 = q_T + c_T$, *i.e.*, we finish with the same battery charge that we start with. With the battery operating, the net consumption in period t is $u_t + c_t$; we require this to be nonnegative (*i.e.*, we do not pump power back into the grid). The total cost is then $p^T(u + c)$.

The battery is characterized by three parameters: The capacity Q , where $q_t \leq Q$; the maximum charge rate C , where $c_t \leq C$; and the maximum discharge rate D , where $c_t \geq -D$. (The parameters Q , C , and D are nonnegative.)

- (a) Explain how to find the charging profile $c \in \mathbf{R}^T$ (and associated stored energy profile $q \in \mathbf{R}^T$) that minimizes the total cost, subject to the constraints.
- (b) Solve the problem instance with data p and u given in `storage_tradeoff_data.*`, $Q = 35$, and $C = D = 3$. Plot u_t , p_t , c_t , and q_t versus t .
- (c) *Storage trade-offs.* Plot the minimum total cost versus the storage capacity Q , using p and u from `storage_tradeoff_data.*`, and charge/discharge limits $C = D = 3$. Repeat for charge/discharge limits $C = D = 1$. (Put these two trade-off curves on the same plot.) Give an interpretation of the endpoints of the trade-off curves.

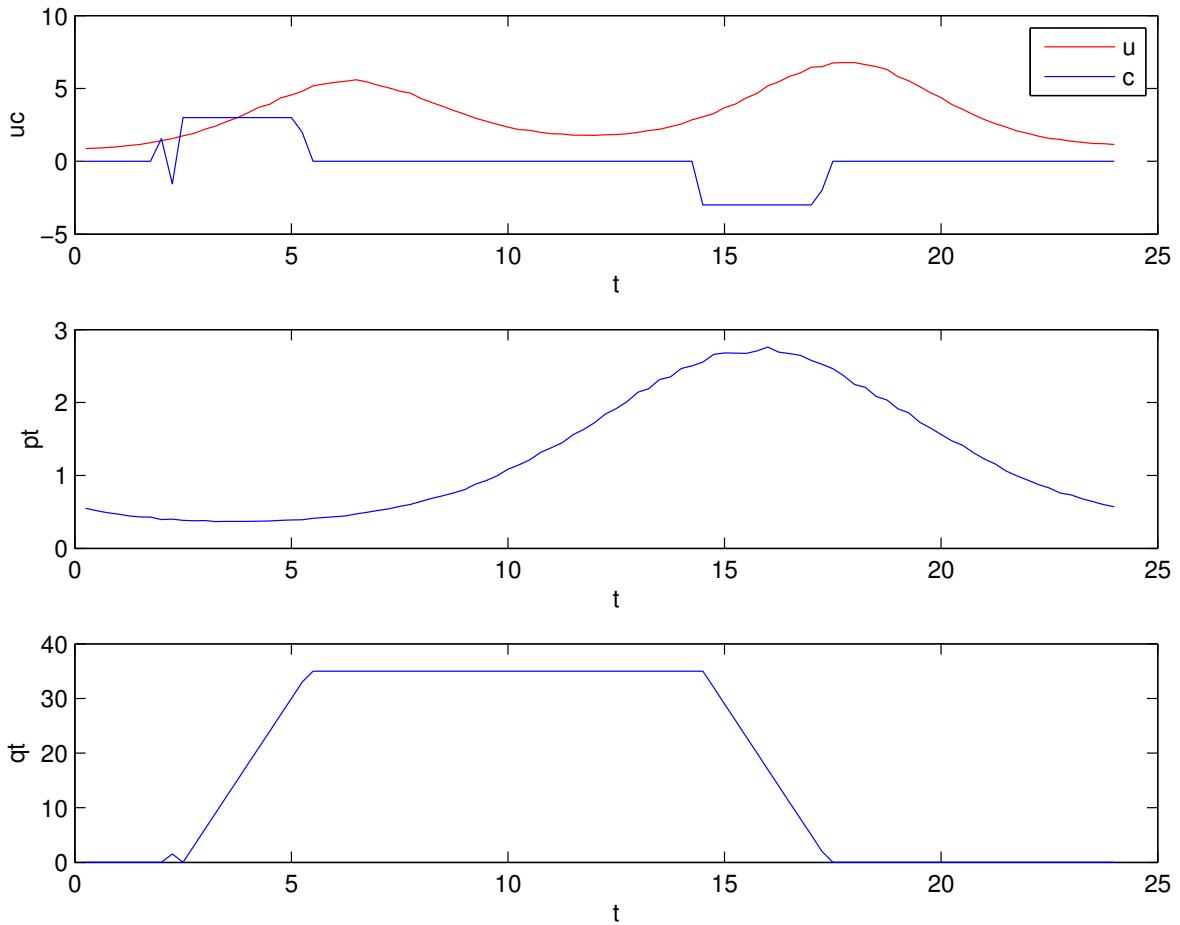
Solution.

- (a) The problem is an LP,

$$\begin{aligned} \text{minimize} \quad & p^T(u + c) \\ \text{subject to} \quad & -D\mathbf{1} \preceq c \preceq C\mathbf{1}, \quad u + c \succeq 0, \quad 0 \preceq q \preceq Q\mathbf{1} \\ & q_{t+1} = q_t + c_t, \quad t = 1, \dots, T \\ & q_1 = q_T + c_T \end{aligned}$$

with variables $c, q \in \mathbf{R}^T$.

- (b) The code for the problem is given in part (c). The results are shown below.



- (c) We vary the range of Q from 1 to 150 and solve the LP for the two cases of high charge/discharge limit and low charge/discharge limit.

The following Matlab code solves the problem.

```
clear all; close all; storage_tradeoff_data;
Q = 35;
C = 3; D = 3;
cvx_quiet(true);
cvx_begin
    variables q(T,1) c(T,1);
    minimize(p'*(u+c));
    subject to
    c >= -D; c <= C;
    q >= 0; q <= Q;
    q(2:T) == q(1:T-1) + c(1:T-1);
    q(1) == q(T) + c(T);
    u+c >= 0;
cvx_end
```

```

figure;
ts = (1:T)/4;
subplot(3,1,1);
plot(ts,u, 'r'); hold on
plot(ts,c,'b');
legend('u','c');
xlabel('t');
ylabel('uc');
subplot(3,1,2);
plot(ts, p);
ylabel('pt');
xlabel('t');
subplot(3,1,3);
plot(ts,q);
ylabel('qt');
xlabel('t');
print -depsc storage_tradeoff_time_trace.eps

```

```

%% Plot the trade-off curves
N = 31; Qs = linspace(0, 150,N);
C = 1; D = 1;
cvx_quiet(true);
for i=1:N
    Q = Qs(i);
    cvx_begin
        variables q(T,1) c(T,1);
        minimize(p'*(u+c));
        subject to
        c >= -D;           c <= C;
        q >= 0;             q <= Q;
        q(2:T) == q(1:T-1) + c(1:T-1);
        q(1) == q(T) + c(T);
        u + c >= 0;
    cvx_end
    qStore1(:,i) = q;
    cStore1(:,i) = c;
    cost1(i) = cvx_optval;
end
figure;
plot(Qs,cost1, 'b.-');
hold on

C = 3; D = 3;

```

```

for i=1:N
    Q = Qs(i);
    cvx_begin
        variables q(T,1) c(T,1);
        minimize(p'* (u+c));
        subject to
            c >= -D;           c <= C;
            q >= 0;             q <= Q;
            q(2:T) == q(1:T-1) + c(1:T-1);
            q(1) == q(T) + c(T);
            u + c >= 0;
    cvx_end
    qStore2(:,i) = q;
    cStore2(:,i) = c;
    cost2(i) = cvx_optval;
end
plot(Qs,cost2, 'g--');
xlabel('Q');
ylabel('cost');
print -depsc storage_tradeoff_curve.eps

```

The following Python code solves the problem.

```

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

np.random.seed(1)

T = 96
t = np.linspace(1, T, num=T).reshape(T,1)
p = np.exp(-np.cos((t-15)*2*np.pi/T)+0.01*np.random.randn(T,1))
u = 2*np.exp(-0.6*np.cos((t+40)*np.pi/T) - \
              0.7*np.cos(t*4*np.pi/T)+0.01*np.random.randn(T,1))

plt.figure(1)
plt.plot(t/4, p);
plt.plot(t/4, u, 'r');

Q = 35
C, D = 3, 3

q = cvx.Variable(T,1)
c = cvx.Variable(T,1)
obj = p.T*(u+c)
cons = [c >= -D]

```

```

cons += [c <= C]
cons += [q >= 0]
cons += [q <= Q]
cons += [q[1:] == q[:T-1] + c[:T-1]]
cons += [q[0] == q[T-1] + c[T-1]]
cons += [u+c >= 0]
pstar = cvx.Problem(cvx.Minimize(obj), cons).solve()

plt.figure(2)
ts = np.linspace(1, T, num=T).reshape(T,1)/4
plt.subplot(3,1,1)
plt.plot(ts, u, 'r');
plt.plot(ts, c.value, 'b');
plt.xlabel('t')
plt.ylabel('uc')
plt.legend(['u', 'c'])
plt.subplot(3,1,2)
plt.plot(ts, p, 'b');
plt.xlabel('t')
plt.ylabel('pt')
plt.subplot(3,1,3)
plt.plot(ts, q.value, 'b');
plt.xlabel('t')
plt.ylabel('qt')
plt.ylim((0, 40))
plt.savefig('storage_tradeoff_time_trace.eps')

# Plot the tradeoff curves
N = 31
Qs = np.linspace(0, 150, num=N).reshape(N,1)
C, D = 1, 1
cost1 = np.zeros((N,1))
cost2 = np.zeros((N,1))

for i in range(N):
    Q = Qs[i]
    q = cvx.Variable(T,1)
    c = cvx.Variable(T,1)
    obj = p.T*(u+c)
    cons = [c >= -D]
    cons += [c <= C]
    cons += [q >= 0]
    cons += [q <= Q]
    cons += [q[1:] == q[:T-1] + c[:T-1]]

```

```

cons += [q[0] == q[T-1] + c[T-1]]
cons += [u+c >= 0]
pstar = cvx.Problem(cvx.Minimize(obj), cons).solve()
cost1[i] = pstar

C, D = 3, 3

for i in range(N):
    Q = Qs[i]
    q = cvx.Variable(T,1)
    c = cvx.Variable(T,1)
    obj = p.T*(u+c)
    cons = [c >= -D]
    cons += [c <= C]
    cons += [q >= 0]
    cons += [q <= Q]
    cons += [q[1:] == q[:T-1] + c[:T-1]]
    cons += [q[0] == q[T-1] + c[T-1]]
    cons += [u+c >= 0]
    pstar = cvx.Problem(cvx.Minimize(obj), cons).solve()
    cost2[i] = pstar

plt.figure(3)
plt.plot(Qs, cost2, 'g--');
plt.plot(Qs, cost1, 'b.-');
plt.xlabel('Q')
plt.ylabel('cost')
plt.savefig('storage_tradeoff_curve.eps')

plt.show()

```

The following Julia code solves the problem.

```

using Convex, Gadfly
include("storage_tradeoff_data.jl")

Q = 35;
C = 3;
D = 3;

q = Variable(T);
c = Variable(T);

constraints = c >= -D;
constraints += c <= C;
constraints += q >= 0;
constraints += q <= Q;

```

```

constraints += q[2:T] == q[1:T-1] + c[1:T-1];
constraints += q[1] == q[T] + c[T];
constraints += u+c >= 0;

prob = minimize(p*(u+c), constraints);
solve!(prob);

ts = [1:T]/4;

p1 = plot(
    layer(
        x = ts,
        y = u,
        Geom.line,
        Theme(default_color = color("red"))
    ),
    layer(
        x = ts,
        y = c.value,
        Geom.line,
        Theme(default_color = color("blue"))
    ),
    Guide.xlabel("t"),
    Guide.ylabel("u<sub>t</sub>,c<sub>t</sub>"),
    Guide.manual_color_key("", [
        ["u<sub>t</sub>", "c<sub>t</sub>"],
        ["red", "blue"]
    ])
);

p2 = plot(
    layer(
        x = ts,
        y = p,
        Geom.line,
        Theme(default_color = color("blue"))
    ),
    Guide.xlabel("t"),
    Guide.ylabel("p<sub>t</sub>")
);

p3 = plot(
    layer(
        x = ts,
        y = q.value,
        Geom.line,

```

```

Theme(default_color = color("blue"))
),
Guide.xlabel("t"),
Guide.ylabel("q<sub>t</sub>")
);
draw(PS("storage_tradeoff_time_trace.eps", 6inch, 6inch),
      vstack(p1, p2, p3))

# Plot the trade-off curves
N = 31;
Qs = [0:5:150];
C = 1;
D = 1;
cost1 = zeros(N,1);

for i = 1:N
    Q = Qs[i];
    q = Variable(T);
    c = Variable(T);

    constraints = c >= -D;
    constraints += c <= C;
    constraints += q >= 0;
    constraints += q <= Q;
    constraints += q[2:T] == q[1:T-1] + c[1:T-1];
    constraints += q[1] == q[T] + c[T];
    constraints += u+c >= 0;

    prob = minimize(p'*(u+c), constraints);
    solve!(prob);
    println(prob.status)
    cost1[i] = prob.optval;
end

C = 3;
D = 3;
cost2 = zeros(N,1);

for i = 1:N
    Q = Qs[i];
    q = Variable(T);
    c = Variable(T);

    constraints = c >= -D;
    constraints += c <= C;

```

```

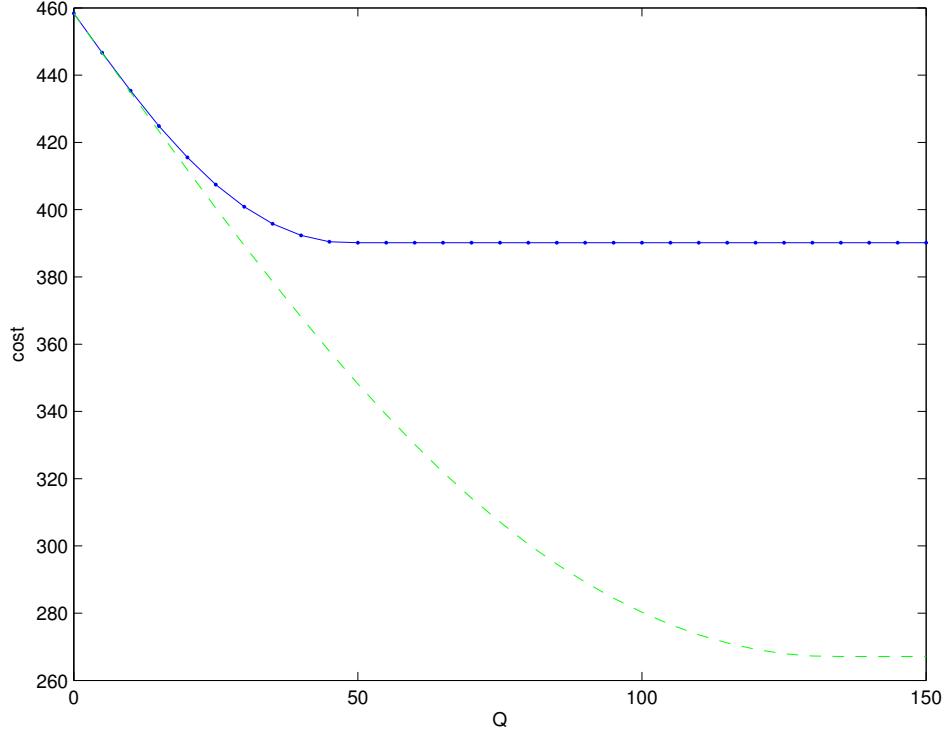
constraints += q >= 0;
constraints += q <= Q;
constraints += q[2:T] == q[1:T-1] + c[1:T-1];
constraints += q[1] == q[T] + c[T];
constraints += u+c >= 0;

prob = minimize(p*(u+c), constraints);
solve!(prob);
println(prob.status)
cost2[i] = prob.optval;
end

p4 = plot(
    layer(
        x = Qs,
        y = cost1,
        Geom.line,
        Theme(default_color = color("blue"))
    ),
    layer(
        x = Qs,
        y = cost2,
        Geom.point,
        Theme(default_color = color("green"))
    ),
    Guide.xlabel("Q"),
    Guide.ylabel("pTu"),
    Guide.manual_color_key("", ["C=1, D=1", "C=3, D=3"], ["blue", "green"])
);
display(p4);
draw(PS("storage_tradeoff_curve.eps", 6inch, 3inch), p4)

```

The trade-off curves are shown below, where the blue solid curve corresponds to $C = D = 1$ and the green dashed curve corresponds to $C = D = 3$. The intersection of the trade-off curves with the y axis (corresponding to $Q = 0$) gives the total cost if there were no battery, $p^T u$. On the right end of the trade-off curves, the battery capacity constraint is no longer active, so no further reduction in total cost is obtained. The total cost reduction here is limited by the charge/discharge limits.



16.10 Cost-comfort trade-off in air conditioning. A heat pump (air conditioner) is used to cool a residence to temperature T_t in hour t , on a day with outside temperature T_t^{out} , for $t = 1, \dots, 24$. These temperatures are given in Kelvin, and we will assume that $T_t^{\text{out}} \geq T_t$.

A total amount of heat $Q_t = \alpha(T_t^{\text{out}} - T_t)$ must be removed from the residence in hour t , where α is a positive constant (related to the quality of thermal insulation).

The electrical energy required to pump out this heat is given by $E_t = Q_t/\gamma_t$, where

$$\gamma_t = \eta \frac{T_t}{T_t^{\text{out}} - T_t}$$

is the *coefficient of performance* of the heat pump and $\eta \in (0, 1]$ is the efficiency constant. The efficiency is typically around 0.6 for a modern unit; the theoretical limit is $\eta = 1$. (When $T_t = T_t^{\text{out}}$, we take $\gamma_t = \infty$ and $E_t = 0$.)

Electrical energy prices vary with the hour, and are given by $P_t > 0$ for $t = 1, \dots, 24$. The total energy cost is $C = \sum_t P_t E_t$. We will assume that the prices are known.

Discomfort is measured using a piecewise-linear function of temperature,

$$D_t = (T_t - T^{\text{ideal}})_+,$$

where T^{ideal} is an ideal temperature, below which there is no discomfort. The total daily discomfort is $D = \sum_{t=1}^{24} D_t$. You can assume that $T^{\text{ideal}} < T_t^{\text{out}}$.

To get a point on the optimal cost-comfort trade-off curve, we will minimize $C + \lambda D$, where $\lambda > 0$. The variables to be chosen are T_1, \dots, T_{24} ; all other quantities described above are given.

Show that this problem has an analytical solution of the form $T_t = \psi(P_t, T_t^{\text{out}})$, where $\psi : \mathbf{R}^2 \rightarrow \mathbf{R}$. The function ψ can depend on the constants $\alpha, \eta, T^{\text{ideal}}, \lambda$. Give ψ explicitly. You are free (indeed, encouraged) to check your formula using CVX, with made up values for the constants.

Disclaimer. The focus of this course is *not* on deriving 19th century pencil and paper solutions to problems. But every now and then, a practical problem will actually have an analytical solution. This is one of them.

Solution. We use the expression for E_t and the efficiency to get

$$E_t = (\alpha/\eta) \frac{(T_t^{\text{out}} - T_t)^2}{T_t},$$

which is a convex function of T_t . For any practical problem we can regard the denominator as a constant, but it's cool to note that we can handle the nonlinearity of the thermodynamic efficiency exactly. It follows that the cost C , which is a positive weighted sum of E_t , is convex. The discomfort is evidently convex in T_t , so the composite objective $C + \lambda D$ is convex in T_t . So we have a convex problem here.

The composite objective $C + \lambda D$ is *separable* in T_t , i.e., a sum of functions of T_t :

$$C + \lambda D = \sum_{t=1}^{24} \left(P_t(\alpha/\eta) \frac{(T_t^{\text{out}} - T_t)^2}{T_t} + \lambda(T_t - T^{\text{ideal}})_+ \right).$$

It follows that we can find each T_t (separately) by minimizing

$$P_t(\alpha/\eta) \frac{(T_t^{\text{out}} - T_t)^2}{T_t} + \lambda(T_t - T^{\text{ideal}})_+.$$

The derivative of the first term is

$$P_t(\alpha/\eta) \frac{T_t^2 - (T_t^{\text{out}})^2}{T_t^2}.$$

First assume that $T_t > T^{\text{ideal}}$. Then the optimality condition is

$$P_t(\alpha/\eta) \frac{T_t^2 - (T_t^{\text{out}})^2}{T_t^2} + \lambda = 0,$$

which gives

$$T_t = (1 + \eta\lambda/(P_t\alpha))^{-1/2} T_t^{\text{out}}.$$

If $T_t < T^{\text{ideal}}$, the optimality condition is

$$P_t(\alpha/\eta) \frac{T_t^2 - (T_t^{\text{out}})^2}{T_t^2} = 0,$$

which gives $T_t = T_t^{\text{out}}$, which contradicts $T_t < T^{\text{ideal}}$. So this case cannot happen. But we can have $T_t = T^{\text{ideal}}$; this happens when

$$(1 + \eta\lambda/(P_t\alpha))^{-1/2} T_t^{\text{out}} \leq T^{\text{ideal}}.$$

So the optimal choice of temperature is simply

$$T_t = \psi(P_t, T_t^{\text{out}}) = \max \left\{ (1 + \eta\lambda/(P_t\alpha))^{-1/2} T_t^{\text{out}}, T^{\text{ideal}} \right\}.$$

Let's test our formula using CVX.

```

% Cost-comfort trade-off in air conditioning.
N = 24;
Tout = 3*sin(2*pi*(1:N)'/24-pi/2)+29; Tideal = 25;
Tout = Tout + 273.15; Tideal = Tideal + 273.15;

eta = 0.6; alpha = 1.8; lambda = 1;
price = 6*[ones(8,1);1.5*ones(9,1);ones(7,1)];
k = price*alpha/eta;

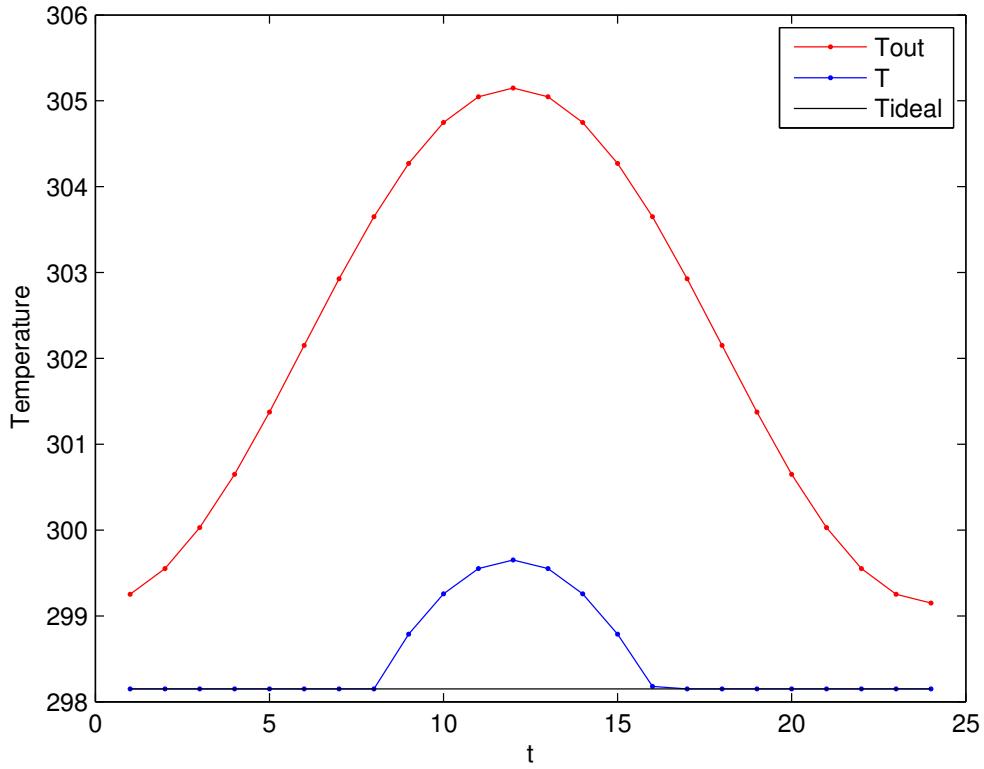
cvx_begin
    variable T(N)
    C = k'*quad_over_lin(Tout-T,T,2);
    D = sum(pos(T-Tideal));
    minimize ( C + lambda*D )
    subject to
        T >= 0; T <= Tout; % they are not necessary
cvx_end

T_analytic = max(sqrt(k./(k+lambda)).*Tout, Tideal);
C_analytic = k'*quad_over_lin(Tout-T_analytic,T_analytic,2);
display(['Total cost obtained by cvx: ' num2str(C)]);
display(['Total cost obtained analytically: ' num2str(C_analytic)]);

plot((1:N)',Tout,'.-r',(1:N)',T,'.-b',(1:N)',Tideal*ones(N,1),'-k');
xlabel('t'); ylabel('Temperature'); legend('Tout','T','Tideal');

print -depsc air_cond.eps

```



The total energy cost obtained from the analytical solution is 31.838. CVX returns the same answer (31.838, when using sdpt3; and 31.8204, when using Sedumi).

- 16.11 Optimal electric motor drive currents.** In this problem you will design the drive current waveforms for an AC (alternating current) electric motor. The motor has a magnetic rotor which spins with constant angular velocity $\omega \geq 0$ inside the stationary stator. The stator contains three circuits (called *phase windings*) with (vector) current waveform $i : \mathbf{R} \rightarrow \mathbf{R}^3$ and (vector) voltage waveform $v : \mathbf{R} \rightarrow \mathbf{R}^3$, which are 2π -periodic functions of the angular position θ of the rotor. The circuit dynamics are

$$v(\theta) = Ri(\theta) + \omega L \frac{d}{d\theta} i(\theta) + \omega k(\theta),$$

where $R \in \mathbf{S}_{++}^3$ is the resistance matrix, $L \in \mathbf{S}_{++}^3$ is the inductance matrix, and $k : \mathbf{R} \rightarrow \mathbf{R}^3$, a 2π -periodic function of θ , is the back-EMF waveform (which encodes the electromagnetic coupling between the rotor permanent magnets and the phase windings). The angular velocity ω , the matrices R and L , and the back-EMF waveform k , are known.

We must have $|v_i(\theta)| \leq v^{\text{supply}}$, $i = 1, 2, 3$, where v^{supply} is the (given) supply voltage. The output torque of the motor at rotor position θ is $\tau(\theta) = k(\theta)^T i(\theta)$. We will require the torque to have a given constant nonnegative value: $\tau(\theta) = \tau^{\text{des}}$ for all θ .

The average power loss in the motor is

$$P^{\text{loss}} = \frac{1}{2\pi} \int_0^{2\pi} i(\theta)^T R i(\theta) d\theta.$$

The mechanical output power is $P^{\text{out}} = \omega\tau^{\text{des}}$, and the motor efficiency is

$$\eta = P^{\text{out}} / (P^{\text{out}} + P^{\text{loss}}).$$

The objective is to choose the current and voltage waveforms to maximize η .

Discretization. To solve this problem we consider a discretized version in which θ takes on the N values $\theta = h, 2h, \dots, Nh$, where $h = 2\pi/N$. We impose the voltage and torque constraints for these values of θ . We approximate the power loss as

$$P^{\text{loss}} = (1/N) \sum_{j=1}^N i(jh)^T R i(jh).$$

The circuit dynamics are approximated as

$$v(jh) = Ri(jh) + \omega L \frac{i((j+1)h) - i(jh)}{h} + \omega k(jh), \quad j = 1, \dots, N,$$

where here we take $i((N+1)h) = i(h)$ (by periodicity).

Find optimal (discretized) current and voltage waveforms for the problem instance with data given in `ac_motor_data.m`. The back-EMF waveform is given as a $3 \times N$ matrix K . Plot the three current waveform components on one plot, and the three voltage waveforms on another. Give the efficiency obtained.

Solution. We first note that all of the constraints are convex as stated. To maximize the efficiency, we minimize the power loss, which is a sum of convex quadratic terms. The following code will solve the problem.

```
% optimal electric motor drive currents
ac_motor_data;

Rhalf = chol(R);
cvx_begin
    variables I(3, N) V(3, N)
    minimize (norm(Rhalf*I, 'fro')) % sqrt(N*Ploss)
    subject to
        V == R*I + omega*K + omega*L*(I(:, [2:N, 1])-I)/h; % dynamics
        tau_des == sum(K.*I); % torque constraint
        abs(V) <= V_supply; % voltage limits
cvx_end

Ploss = cvx_optval^2/N;
Pout = omega*tau_des;
eta = Pout/(Pout+Ploss);

fprintf('Maximum efficiency: %f\n', eta);

% plot
subplot(2, 1, 1);
```

```

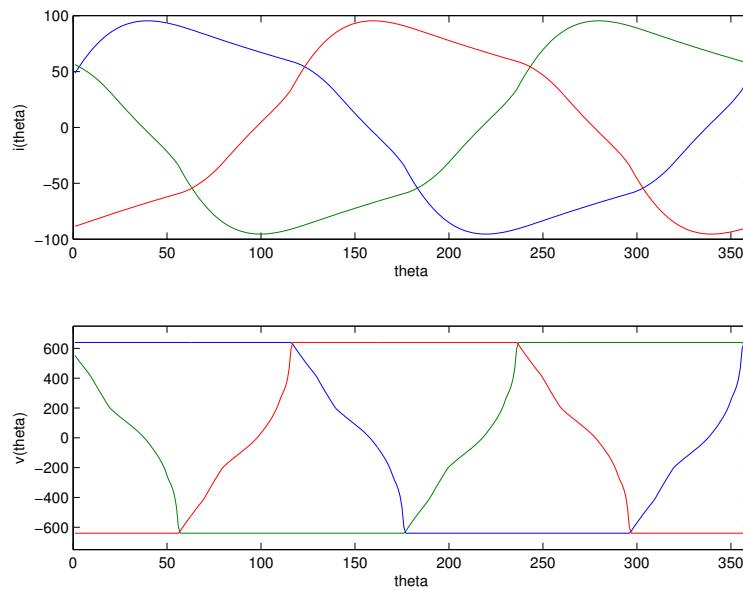
plot(I(:, 1:N)');
xlabel('theta'); ylabel('i(theta)');
axis([0, 360, -100, 100]);

subplot(2, 1, 2);
plot(V');
xlabel('theta'); ylabel('v(theta)');
axis([0, 360, -750, 750]);

print -depsc ac_motor.eps;

```

The maximum efficiency is 90.2%. The optimal current and voltage waveforms are shown below.



17 Miscellaneous applications

17.1 Earth mover's distance. In this exercise we explore a general method for constructing a distance between two probability distributions on a finite set, called the *earth mover's distance*, *Wasserstein metric*, or *Dubroshkin metric*. Let x and y be two probability distributions on $\{1, \dots, n\}$, i.e., $\mathbf{1}^T x = \mathbf{1}^T y = 1$, $x \succeq 0$, $y \succeq 0$. We imagine that x_i is the amount of earth stored at location i ; our goal is to move the earth between locations to obtain the distribution given by y . Let C_{ij} be the cost of moving one unit of earth from location j to location i . We assume that $C_{ii} = 0$, and $C_{ij} = C_{ji} > 0$ for $i \neq j$. (We allow $C_{ij} = \infty$, which means that earth cannot be moved directly from node j to node i .) Let $S_{ij} \geq 0$ denote the amount of earth moved from location j to location i . The total cost is $\sum_{i,j=1}^n S_{ij} C_{ij} = \text{tr } C^T S$. The shipment matrix S must satisfy the balance equations,

$$\sum_{j=1}^n S_{ij} = y_i, \quad i = 1, \dots, n, \quad \sum_{i=1}^n S_{ij} = x_j, \quad j = 1, \dots, n,$$

which we can write compactly as $S\mathbf{1} = y$, $S^T\mathbf{1} = x$. (The first equation states that the total amount shipped into location i equals y_i ; the second equation states that the total shipped out from location j is x_j .) The earth mover's distance between x and y , denoted $d(x, y)$, is given by the minimal cost of earth moving required to transform x to y , i.e., the optimal value of the problem

$$\begin{aligned} &\text{minimize} && \text{tr } C^T S \\ &\text{subject to} && S_{ij} \geq 0, \quad i, j = 1, \dots, n \\ & && S\mathbf{1} = y, \quad S^T\mathbf{1} = x, \end{aligned}$$

with variables $S \in \mathbf{R}^{n \times n}$.

We can also give a probability interpretation of $d(x, y)$. Consider a random variable Z on $\{1, \dots, n\}^2$ with values C_{ij} . We seek the joint distribution S that minimizes the expected value of the random variable Z , with given marginals x and y .

The earth mover's distance is used to compare, for example, 2D images, with C_{ij} equal to the distance between pixels i and j . If x and y represent two photographs of the same scene, from slightly different viewpoints and with an offset in camera position (say), $d(x, y)$ will be small, but the distance between x and y measured by most common norms (e.g., $\|x - y\|_1$) will be large.

- (a) Show that d satisfies the following. *Symmetry*: $d(x, y) = d(y, x)$; *Nonnegativity*: $d(x, y) \geq 0$; *Definiteness*: $d(x, x) = 0$, and $d(x, y) > 0$ for $x \neq y$.
- (b) Show that $d(x, y)$ is the optimal value of the problem

$$\begin{aligned} &\text{maximize} && \nu^T x + \mu^T y \\ &\text{subject to} && \nu_i + \mu_j \leq C_{ij}, \quad i, j = 1, \dots, n, \end{aligned}$$

with variables $\nu, \mu \in \mathbf{R}^n$.

Solution.

- (a) First let's show that $d(x, y) \geq 0$. S and C are both elementwise nonnegative, so $\text{tr } C^T S \geq 0$. So the objective is always nonnegative.

Let's now show that $d(x, y) = 0$ when $x = y$. To do this, we observe that $S = \mathbf{diag}(x)$ is feasible; the objective is then $\mathbf{tr} C^T S = 0$, since the diagonal of C is zero.

We have $d(x, y) > 0$ when $x \neq y$. If $x \neq y$, then $S\mathbf{1} \neq S^T\mathbf{1}$, so S cannot be symmetric. A necessary condition for this is that S have some non-zero off-diagonal elements. Since $S_{ij} \geq 0$, S must have some positive off-diagonal elements. We are given that $C_{ij} > 0$ for $i \neq j$. So, any feasible S for $x \neq y$ must satisfy $\mathbf{tr} C^T S > 0$.

To show symmetry, suppose S is feasible for the problem. Then S^T is feasible for the problem with x and y swapped. The two costs are the same, since $\mathbf{tr} C^T S = \mathbf{tr} C^T S^T$ (using symmetry of C).

Finally, we need to show the triangle inequality. From the definition of d as the optimal value of a convex optimization problem, with constraints that are jointly convex in S , y , x , we deduce that d is a convex function of (x, y) .

- (b) We show that the optimization problem in part (b) is the dual of the optimization problem in the question statement, which defines $d(x, y)$. The Lagrangian of the problem is $L(S, \Lambda, \nu, \mu) = \mathbf{tr} C^T S - \mathbf{tr} \Lambda^T S + \nu^T(y - S\mathbf{1}) + \mu^T(x - S^T\mathbf{1})$. The dual function is

$$\begin{aligned} g(\Lambda, \nu, \mu) &= \inf_S L(S, \Lambda, \nu, \mu) \\ &= \nu^T y + \mu^T x + \inf_S (\mathbf{tr} C^T S - \mathbf{tr} \Lambda^T S - \nu^T S\mathbf{1} - \mu^T S^T\mathbf{1}) \\ &= \nu^T y + \mu^T x + \inf_S (\mathbf{tr}(C - \Lambda - \nu\mathbf{1}^T - \mathbf{1}\mu^T)^T S) \end{aligned}$$

Note that $\mathbf{tr}(C - \Lambda - \nu\mathbf{1}^T - \mathbf{1}\mu^T)^T S$ is linear in S , so $L(S, \Lambda, \nu, \mu)$ is bounded below only when $C - \Lambda - \nu\mathbf{1}^T - \mathbf{1}\mu^T = 0$. So,

$$g(\Lambda, \nu, \mu) = \begin{cases} \nu^T y + \mu^T x & C - \Lambda - \nu\mathbf{1}^T - \mathbf{1}\mu^T = 0 \\ -\infty & \text{otherwise} \end{cases}$$

So, the dual problem is

$$\begin{aligned} &\text{maximize} && \nu^T y + \mu^T x \\ &\text{subject to} && \Lambda \geq 0 \\ & && C - \Lambda - \nu\mathbf{1}^T - \mathbf{1}\mu^T = 0, \end{aligned}$$

or equivalently

$$\begin{aligned} &\text{maximize} && \nu^T x + \mu^T y \\ &\text{subject to} && \Lambda_{ij} \geq 0, \quad i, j = 1, \dots, n \\ & && \nu_i + \mu_j + \Lambda_{ij} = C_{ij}, \quad i, j = 1, \dots, n. \end{aligned}$$

We treat Λ as a slack variable and can therefore write the problem as

$$\begin{aligned} &\text{maximize} && \nu^T x + \mu^T y \\ &\text{subject to} && \nu_i + \mu_j \leq C_{ij}, \quad i, j = 1, \dots, n. \end{aligned}$$

By strong duality, $d(x, y)$, the optimal value of the original problem, is also the optimal value of this problem.

17.2 Radiation treatment planning. In radiation treatment, radiation is delivered to a patient, with the goal of killing or damaging the cells in a tumor, while carrying out minimal damage to other tissue. The radiation is delivered in beams, each of which has a known pattern; the level of each beam can be adjusted. (In most cases multiple beams are delivered at the same time, in one ‘shot’, with the treatment organized as a sequence of ‘shots’.) We let b_j denote the level of beam j , for $j = 1, \dots, n$. These must satisfy $0 \leq b_j \leq B^{\max}$, where B^{\max} is the maximum possible beam level. The exposure area is divided into m voxels, labeled $i = 1, \dots, m$. The dose d_i delivered to voxel i is linear in the beam levels, *i.e.*, $d_i = \sum_{j=1}^n A_{ij} b_j$. Here $A \in \mathbf{R}_+^{m \times n}$ is a (known) matrix that characterizes the beam patterns. We now describe a simple radiation treatment planning problem.

A (known) subset of the voxels, $\mathcal{T} \subset \{1, \dots, m\}$, corresponds to the tumor or target region. We require that a minimum radiation dose D^{target} be administered to each tumor voxel, *i.e.*, $d_i \geq D^{\text{target}}$ for $i \in \mathcal{T}$. For all other voxels, we would like to have $d_i \leq D^{\text{other}}$, where D^{other} is a desired maximum dose for non-target voxels. This is generally not feasible, so instead we settle for minimizing the penalty

$$E = \sum_{i \notin \mathcal{T}} ((d_i - D^{\text{other}})_+)^2,$$

where $(\cdot)_+$ denotes the nonnegative part. We can interpret E as the sum of the squares of the nontarget excess doses.

- (a) Show that the treatment planning problem is convex. The optimization variable is $b \in \mathbf{R}^n$; the problem data are B^{\max} , A , \mathcal{T} , D^{target} , and D^{other} .
- (b) Solve the problem instance with data given in the file `treatment_planning_data.m`. Here we have split the matrix A into `Atarget`, which contains the rows corresponding to the target voxels, and `Aother`, which contains the rows corresponding to other voxels. Give the optimal value. Plot the dose histogram for the target voxels, and also for the other voxels. Make a brief comment on what you see. *Remark.* The beam pattern matrix in this problem instance is randomly generated, but similar results would be obtained with realistic data.

Solution. There’s not much to say here, except that the constraints are linear, and the objective is convex. Clearly the function $(a)_+^2$, which is zero for $a \leq 0$, and a^2 for $a \geq 0$, is convex; our objective E is simply a sum of functions of this form, with affine argument.

The optimum penalalty is $E^* = 0.308$.

Here is the code to solve the problem:

```
% radiation treatment planning
treatment_planning_data;

cvx_begin
variable b(n); % beam intensities
0 <= b;
b <= Bmax;
Atarget*b >= Dttarget % deliver minimum does to tumor voxels
minimize (sum(square_pos(Aother*b-Dother))) % minimize square excess dose delivered to others
cvx_end
```

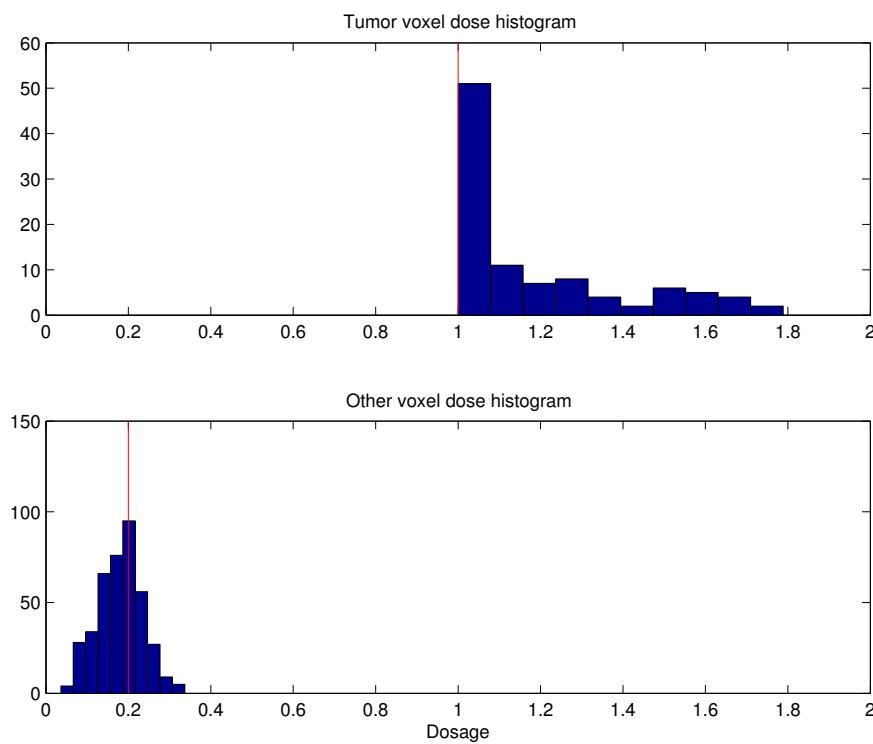
```

subplot(2,1,1);
hist(Atarget*b);
axis([0 2 0 60]);
hold on; plot([Dttarget Dttarget],[0 60],'r')
title('Tumor voxel dose histogram')
subplot(2,1,2);
hist(Aother*b);
axis([0 2 0 150]);
hold on; plot([Dother Dother],[0 150],'r')
title('Other voxel dose histogram')
xlabel('Dosage')

print -depsc dose_histos

```

The resulting dose histograms are shown below, along with vertical red lines showing D^{target} and D^{other} .



17.3 Flux balance analysis in systems biology. Flux balance analysis is based on a very simple model of the reactions going on in a cell, keeping track only of the gross rate of consumption and production of various chemical species within the cell. Based on the known stoichiometry of the reactions, and known upper bounds on some of the reaction rates, we can compute bounds on the other reaction rates, or cell growth, for example.

We focus on m metabolites in a cell, labeled M_1, \dots, M_m . There are n reactions going on, labeled R_1, \dots, R_n , with nonnegative reaction rates v_1, \dots, v_n . Each reaction has a (known) stoichiometry, which tells us the rate of consumption and production of the metabolites per unit of reaction rate. The stoichiometry data is given by the *stoichiometry matrix* $S \in \mathbf{R}^{m \times n}$, defined as follows: S_{ij} is the rate of production of M_i due to unit reaction rate $v_j = 1$. Here we consider consumption of a metabolite as negative production; so $S_{ij} = -2$, for example, means that reaction R_j causes metabolite M_i to be consumed at a rate $2v_j$.

As an example, suppose reaction R_1 has the form $M_1 \rightarrow M_2 + 2M_3$. The consumption rate of M_1 , due to this reaction, is v_1 ; the production rate of M_2 is v_1 ; and the production rate of M_3 is $2v_1$. (The reaction R_1 has no effect on metabolites M_4, \dots, M_m .) This corresponds to a first column of S of the form $(-1, 1, 2, 0, \dots, 0)$.

Reactions are also used to model flow of metabolites into and out of the cell. For example, suppose that reaction R_2 corresponds to the flow of metabolite M_1 into the cell, with v_2 giving the flow rate. This corresponds to a second column of S of the form $(1, 0, \dots, 0)$.

The last reaction, R_n , corresponds to biomass creation, or cell growth, so the reaction rate v_n is the cell growth rate. The last column of S gives the amounts of metabolites used or created per unit of cell growth rate.

Since our reactions include metabolites entering or leaving the cell, as well as those converted to biomass within the cell, we have conservation of the metabolites, which can be expressed as $Sv = 0$. In addition, we are given upper limits on *some* of the reaction rates, which we express as $v \preceq v^{\max}$, where we set $v_j^{\max} = \infty$ if no upper limit on reaction rate j is known. The goal is to find the maximum possible cell growth rate (*i.e.*, largest possible value of v_n) consistent with the constraints

$$Sv = 0, \quad v \succeq 0, \quad v \preceq v^{\max}.$$

The questions below pertain to the data found in `fba_data.m`.

- (a) Find the maximum possible cell growth rate G^* , as well as optimal Lagrange multipliers for the reaction rate limits. How sensitive is the maximum growth rate to the various reaction rate limits?
- (b) *Essential genes and synthetic lethals.* For simplicity, we'll assume that each reaction is controlled by an associated gene, *i.e.*, gene G_i controls reaction R_i . Knocking out a set of genes associated with some reactions has the effect of setting the reaction rates (or equivalently, the associated v^{\max} entries) to zero, which of course reduces the maximum possible growth rate. If the maximum growth rate becomes small enough or zero, it is reasonable to guess that knocking out the set of genes will kill the cell. An *essential gene* is one that when knocked out reduces the maximum growth rate below a given threshold G^{\min} . (Note that G_n is always an essential gene.) A *synthetic lethal* is a pair of non-essential genes that when knocked out reduces the maximum growth rate below the threshold. Find all essential genes and synthetic lethals for the given problem instance, using the threshold $G^{\min} = 0.2G^*$.

Solution. There's not much to do here other than write the code for the problem. For determining the essential genes and synthetic lethals, we simply loop over pairs of reactions to zero out.

Here is the solution in Matlab.

```

% flux balance analysis in systems biology

fba_data;

% part (a): find maximum growth rate
cvx_begin
variable v(n);
dual variable lambda;
maximize (v(n));
S*v == 0;
v >= 0;
lambda: v <= vmax;
cvx_end
Gstar=cvx_optval;
vopt=v;

[v vmax lambda]

% part (b): find essential genes and synthetic lethals
cvx_quiet('true');
G = zeros(n,n);
for i=1:n
for j=i:n
cvx_begin
variable v(n);
S*v == 0;
v >= 0;
v <= vmax;
v(i) == 0;
v(j) == 0;
maximize (v(n))
cvx_end
G(i,j)=cvx_optval;
G(j,i)=cvx_optval;
end
end

G<0.2*Gstar

```

Here it is in Python.

```

# flux balance analysis in systems biology

from fba_data import *
import numpy as np
import cvxpy as cvx

```

```

v = cvx.Variable(n)
constraints = [v <= vmax, v >= 0, S*v == 0]
prob = cvx.Problem(cvx.Maximize(v[n-1]),constraints)
Gstar = prob.solve()
vopt = v.value

# Look at optimal value, and see which constraints are tight
# Check if the dual value for the first constraint agrees
print(Gstar)
print(vopt)
print(vmax)
print(constraints[0].dual_value)

# Now see which genes are essential, and which pairs are synthetic lethals
G = np.zeros((n,n))
for i in range(n):
    for j in range(i+1,n):
        v = cvx.Variable(n)
        constraints = [v <= vmax, v >= 0, S*v == 0, v[i] == 0, v[j] == 0]
        prob = cvx.Problem(cvx.Maximize(v[n-1]),constraints)
        val = prob.solve()
        G[i,j] = val
        G[j,i] = val
print(G < 0.2*Gstar)

```

Here is a Julia version.

```

# flux balance analysis in systems biology

include("fba_data.jl");
using Convex, SCS

# part (a): find maximum growth rate
v = Variable(n);
constraints = [
    S*v == 0,
    v >= 0,
    v <= vmax
];
p = maximize(v[n], constraints);
solve!(p);
lambda = constraints[3].dual;
Gstar = p.optval;
vopt = v.value;

```

```

println("G*: ", Gstar)
println("duals: ", lambda)

# part (b): find essential genes and synthetic lethals
G = zeros(n,n);
for i=1:n
    push!(constraints, v[i] == 0)
    for j=i:n
        push!(constraints, v[j] == 0)
        p = maximize(v[n], constraints)
        solve!(p, SCSSolver(verbose=false))
        G[i,j] = p.optval;
        G[j,i] = p.optval;
        pop!(constraints)
    end
    pop!(constraints)
end

# diagonal entries are true for e
println(int(diag(G .< 0.2*Gstar)))
# entries close to 0 are synthetic lethals
println(int(abs(G) .< 0.001))

```

We find the maximum growth rate to be $G^* = 13.55$. Only three reactions are limited: R_1 , R_3 , and R_5 . All other Lagrange multipliers are zero, of course; these have optimal values 0.5, 0.5, and 1.5, respectively. So it appears that the limit on reaction R_5 is the one that would have the largest effect on the optimal growth rate, for a small change in the limit.

We see that the essential genes are G_1 and G_9 . The synthetic lethals are

$$\{G_2, G_3\}, \quad \{G_2, G_7\}, \quad \{G_4, G_7\}, \quad \{G_5, G_7\}.$$

17.4 Online advertising displays. When a user goes to a website, one of a set of n ads, labeled $1, \dots, n$, is displayed. This is called an *impression*. We divide some time interval (say, one day) into T periods, labeled $t = 1, \dots, T$. Let $N_{it} \geq 0$ denote the number of impressions in period t for which we display ad i . In period t there will be a total of $I_t > 0$ impressions, so we must have $\sum_{i=1}^n N_{it} = I_t$, for $t = 1, \dots, T$. (The numbers I_t might be known from past history.) You can treat all these numbers as real. (This is justified since they are typically very large.)

The revenue for displaying ad i in period t is $R_{it} \geq 0$ per impression. (This might come from click-through payments, for example.) The total revenue is $\sum_{t=1}^T \sum_{i=1}^n R_{it} N_{it}$. To maximize revenue, we would simply display the ad with the highest revenue per impression, and no other, in each display period.

We also have in place a set of m contracts that require us to display certain numbers of ads, or mixes of ads (say, associated with the products of one company), over certain periods, with a penalty for any shortfalls. Contract j is characterized by a set of ads $\mathcal{A}_j \subseteq \{1, \dots, n\}$ (while it does not affect

the math, these are often disjoint), a set of periods $\mathcal{T}_j \subseteq \{1, \dots, T\}$, a target number of impressions $q_j \geq 0$, and a shortfall penalty rate $p_j > 0$. The *shortfall* s_j for contract j is

$$s_j = \left(q_j - \sum_{t \in \mathcal{T}_j} \sum_{i \in A_j} N_{it} \right)_+,$$

where $(u)_+$ means $\max\{u, 0\}$. (This is the number of impressions by which we fall short of the target value q_j .) Our contracts require a total penalty payment equal to $\sum_{j=1}^m p_j s_j$. Our net profit is the total revenue minus the total penalty payment.

- (a) Explain how to find the display numbers N_{it} that maximize net profit. The data in this problem are $R \in \mathbf{R}^{n \times T}$, $I \in \mathbf{R}^T$ (here I is the vector of impressions, not the identity matrix), and the contract data A_j , \mathcal{T}_j , q_j , and p_j , $j = 1, \dots, m$.
- (b) Carry out your method on the problem with data given in `ad_disp_data.m`. `ad_disp_data.py`. The data A_j and \mathcal{T}_j , for $j = 1, \dots, m$ are given by matrices $A^{\text{contr}} \in \mathbf{R}^{n \times m}$ and $T^{\text{contr}} \in \mathbf{R}^{T \times m}$, with

$$A_{ij}^{\text{contr}} = \begin{cases} 1 & i \in A_j \\ 0 & \text{otherwise,} \end{cases} \quad T_{tj}^{\text{contr}} = \begin{cases} 1 & t \in \mathcal{T}_j \\ 0 & \text{otherwise.} \end{cases}$$

Report the optimal net profit, and the associated revenue and total penalty payment. Give the same three numbers for the strategy of simply displaying in each period only the ad with the largest revenue per impression.

Solution.

- (a) The constraints $N_{it} \geq 0$, $\sum_{i=1}^n N_{it} = I_t$ are linear. The contract shortfalls s_j are convex functions of N , so the net profit

$$\sum_{t=1}^T \sum_{i=1}^n R_{it} N_{it} - \sum_{j=1}^m p_j s_j$$

is concave. So we have a convex optimization problem.

- (b) We can express the vector of shortfalls as

$$s = \left(q - \mathbf{diag}(A^{\text{contr}}{}^T N T^{\text{contr}}) \right)_+$$

The following code solves the problem:

```
clear all;
% ad display problem
ad_disp_data;

% optimal display
cvx_begin
    variable N(n,T)
    s = pos(q-diag(Acontr'*N*Tcontr));
    maximize(R(:)*N(:)-p'*s)
```

```

subject to
N >= 0;
sum(N)' == I;
cvx_end
opt_s=s;
opt_net_profit = cvx_optval
opt_penalty = p'*opt_s
opt_revenue = opt_net_profit+opt_penalty

% display, ignoring contracts
cvx_begin
variable N(n,T)
maximize(R(:)'*N(:))
subject to
N >= 0;
sum(N)' == I;
cvx_end
greedy_s = pos(q-diag(Acontr'*N*Tcontr));
greedy_net_profit = cvx_optval-p'*greedy_s
greedy_penalty = p'*greedy_s
greedy_revenue = cvx_optval

import cvxpy as cvx
import numpy as np

np.random.seed(0)
n = 100 #number of ads
m = 30 #number of contracts
T = 60 #number of periods

#number of impressions in each period
I = 10*np.random.rand(T, 1); I = np.asmatrix(I)
#revenue rate for each period and ad
R = np.random.rand(n, T); R = np.asmatrix(R)
#contract target number of impressions
q = T/float(n)*50*np.random.rand(m, 1); q = np.asmatrix(q)
#penalty rate for shortfall
p = np.random.rand(m, 1); p = np.asmatrix(p)
#one column per contract. 1's at the periods to be displayed
Tcontr = np.matrix(np.random.rand(T,m)>.8, dtype = float)
Acontr = np.zeros((n ,m)); Acont = np.asmatrix(Acontr)
for i in range(n):
    contract=int(np.floor(m*np.random.rand(1)))
    #one column per contract. 1's at the ads to be displayed
    Acontr[i,contract]=1

```

```

#Solution begins
#-----
N = cvx.Variable(n, T)
s = cvx.pos(q-cvx.diag(Acontr.T*N*Tcontr))

penalty = cvx.pos(p).T*s
revenue = cvx.sum_entries(cvx.mul_elemwise(R,N))

constraints = [N >= 0]
constraints += [cvx.sum_entries(N[:, t]) == I[t] for t in range(T)]
prob = cvx.Problem([], constraints)

#Optimal solution
prob.objective = cvx.Maximize(revenue - penalty)
optimal_profit = prob.solve()
optimal_revenue = revenue.value
optimal_penalty = penalty.value

#Greedy solution
prob.objective = cvx.Maximize(revenue)
greedy_revenue = prob.solve()
greedy_penalty = penalty.value
greedy_profit = greedy_revenue - greedy_penalty

print [optimal_revenue, optimal_penalty, optimal_profit]
print [greedy_revenue, greedy_penalty, greedy_profit]

```

The performance of the optimal and greedy strategies is given in the table below. The optimal strategy gives up a bit of (gross) revenue, in order to pay substantially less penalty, and ends up way ahead of the greedy strategy.

strategy	revenue	penalty	net profit
optimal	268.23	37.66	230.57
greedy	305.10	232.26	72.84

strategy	revenue	penalty	net profit
optimal	280.94	21.94	259.00
greedy	306.37	159.49	146.88

17.5 Ranking by aggregating preferences. We have n objects, labeled $1, \dots, n$. Our goal is to assign a real valued rank r_i to the objects. A *preference* is an ordered pair (i, j) , meaning that object i is preferred over object j . The ranking $r \in \mathbf{R}^n$ and preference (i, j) are *consistent* if $r_i \geq r_j + 1$. (This sets the scale of the ranking: a gap of one in ranking is the threshold for preferring one item over another.) We define the *preference violation* of preference (i, j) with ranking $r \in \mathbf{R}^n$ as

$$v = (r_j + 1 - r_i)_+ = \max\{r_j + 1 - r_i, 0\}.$$

We have a set of m preferences among the objects, $(i^{(1)}, j^{(1)}), \dots, (i^{(m)}, j^{(m)})$. (These may come from several different evaluators of the objects, but this won't matter here.)

We will select our ranking r as a minimizer of the total preference violation penalty, defined as

$$J = \sum_{k=1}^m \phi(v^{(k)}),$$

where $v^{(k)}$ is the preference violation of $(i^{(k)}, j^{(k)})$ with r , and ϕ is a nondecreasing convex penalty function that satisfies $\phi(u) = 0$ for $u \leq 0$.

- (a) Make a (simple, please) suggestion for ϕ for each of the following two situations:
 - (i) We don't mind some small violations, but we really want to avoid large violations.
 - (ii) We want as many preferences as possible to be consistent with the ranking, but will accept some (hopefully, few) larger preference violations.
- (b) Find the rankings obtained using the penalty functions proposed in part (a), on the data set found in `rank_aggr_data.m`. Plot a histogram of preference violations for each case and briefly comment on the differences between them. Give the number of positive preference violations for each case. (Use `sum(v>0.001)` to determine this number.)

Remark. The objects could be candidates for a position, papers at a conference, movies, websites, courses at a university, and so on. The preferences could arise in several ways. Each of a set of evaluators provides some preferences, for example by rank ordering a subset of the objects. The problem can be thought of as aggregating the preferences given by the evaluators, to come up with a composite ranking.

Solution.

- (a) Here are some simple suggestions. For the first case (i), we take a quadratic penalty (for positive violation): $\phi(u) = u_+^2$. For the second case (ii), we take a linear penalty (for positive violations): $\phi(u) = u_+$. If the violations were two-sided, these would correspond to an ℓ_2 norm and an ℓ_1 norm, respectively.
- (b) The problem that we need to solve is

$$\begin{aligned} &\text{minimize} && \sum_{k=1}^m \phi((v^{(k)})_+) \\ &\text{subject to} && v^{(k)} = r_{j^{(k)}} + 1 - r_{i^{(k)}}. \end{aligned}$$

The following Matlab code solves the problem for the particular problem instance using the penalty functions proposed in part (a):

```
%ranking aggregation problem
%load data
rank_aggr_data;

%find rankings with sum of squared violations penalty
cvx_begin
variables r(n) v(m)
minimize (sum(square_pos(v)));

```

```

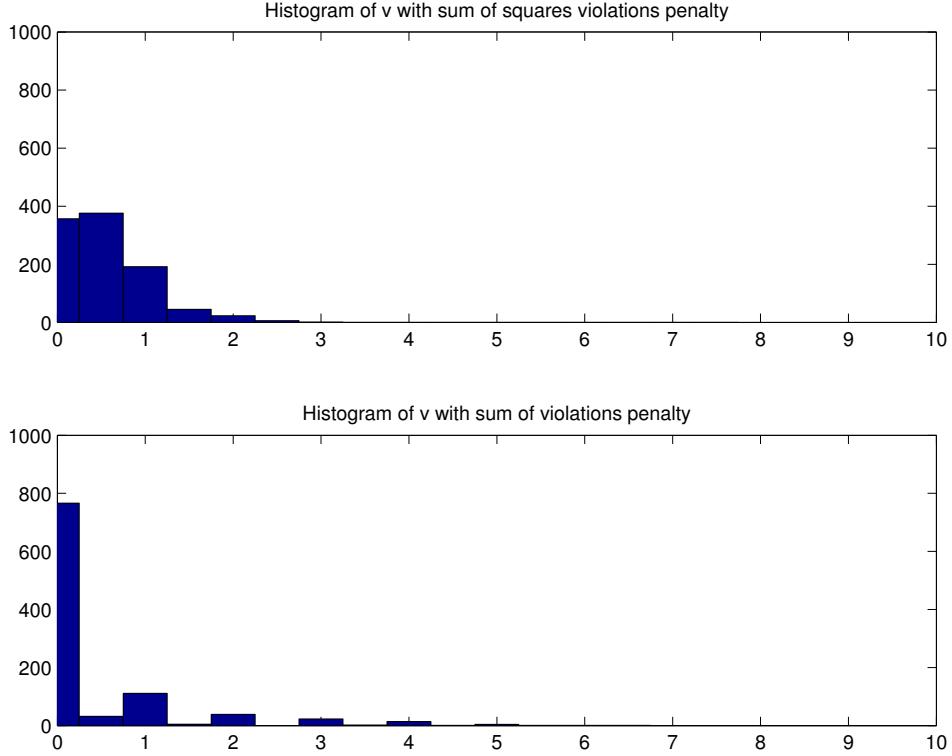
subject to
    v == r(preferences(:,2)) + 1 - r(preferences(:,1))
cvx_end
v_sq = pos(v);
num_viol_sq = sum(v_sq>0.001)

%find rankings with sum of violations penalty
cvx_begin
variables r(n) v(m)
minimize (sum(pos(v)));
subject to
    v == r(preferences(:,2)) + 1 - r(preferences(:,1))
cvx_end
v_lin = pos(v);
num_viol_lin = sum(v_lin>0.001)

%plot results
subplot(2,1,1);
hist(v_sq,[0:0.5:7.5]);
xlim([0 10]);ylim([0 1000]);
title('Histogram of v with sum of squares violations penalty');
subplot(2,1,2);
hist(v_lin,[0:0.5:7.5]);
xlim([0 10]);ylim([0 1000]);
title('Histogram of v with sum of violations penalty');
print -deps rank_aggr_hist

```

The resulting residual histograms are shown below.



For case (i), the violations are often small, but rarely zero; of the 1000 preferences given, 781 are violated. But the largest violation is below 3. For case (ii), only 235 preferences are violated, but we have a few larger violations, with values up to around 7.

17.6 Time release formulation. A patient is treated with a drug (say, in pill form) at different times. Each treatment (or pill) contains (possibly) different amounts of various formulations of the drug. Each of the formulations, in turn, has a characteristic pattern as to how quickly it releases the drug into the bloodstream. The goal is to optimize the blend of formulations that go into each treatment, in order to achieve a desired drug concentration in the bloodstream over time.

We will use discrete time, $t = 1, 2, \dots, T$, representing hours (say). There will be K treatments, administered at known times $1 = \tau_1 < \tau_2 < \dots < \tau_K < T$. We have m drug formulations; each treatment consists of a mixture of these m formulations. We let $a^{(k)} \in \mathbf{R}_+^m$ denote the amounts of the m formulations in treatment k , for $k = 1, \dots, K$.

Each formulation i has a time profile $p_i(t) \in \mathbf{R}_+$, for $t = 1, 2, \dots$. If an amount $a_i^{(k)}$ of formulation i from treatment k is administered at time t_0 , the drug concentration in the bloodstream (due to this formulation) is given by $a_i^{(k)} p_i(t - t_0)$ for $t > t_0$, and 0 for $t \leq t_0$. To simplify notation, we will define $p_i(t)$ to be zero for $t = 0, -1, -2, \dots$. We assume the effects of the different formulations and different treatments are additive, so the total bloodstream drug concentration is given by

$$c(t) = \sum_{k=1}^K \sum_{i=1}^m p_i(t - \tau_k) a_i^{(k)}, \quad t = 1, \dots, T.$$

(This is just a vector convolution.) Recall that $p_i(t - \tau_k) = 0$ for $t \leq \tau_k$, which means that the effect of treatment k does not show up until time $\tau_k + 1$.

We require that $c(t) \leq c^{\max}$ for $t = 1, \dots, T$, where c^{\max} is a given maximum permissible concentration. We define the therapeutic time T^{ther} as

$$T^{\text{ther}} = \min\{t \mid c(\tau) \geq c^{\min} \text{ for } \tau = t, \dots, T\},$$

with $T^{\text{ther}} = \infty$ if $c(t) < c^{\min}$ for $t = 1, \dots, T$. Here, c^{\min} is the minimum concentration for the drug to have therapeutic value. Thus, T^{ther} is the first time at which the drug concentration reaches, and stays above, the minimum therapeutic level.

Finally, we get to the problem. The optimization variables are the treatment formulation vectors $a^{(1)}, \dots, a^{(K)}$. There are two objectives: T^{ther} (which we want to be small), and

$$J^{\text{ch}} = \sum_{k=1}^{K-1} \|a^{(k+1)} - a^{(k)}\|_{\infty}$$

(which we also want to be small). This second objective is a penalty for changing the formulation amounts in the treatments.

The rest of the problem concerns the specific instance with data given in the file `time_release_form_data.m`. This gives data for $T = 168$ (one week, starting from 8AM Monday morning), with treatments occurring 3 times each day, at 8AM, 2PM, and 11PM, so we have a total of $K = 21$ treatments. We have $m = 6$ formulations, with profiles with length 96 (*i.e.*, $p_i(t) = 0$ for $t > 96$).

- Explain how to find the optimal trade-off curve of T^{ther} versus J^{ch} . Your method may involve solving several convex optimization problems.
- Plot the trade-off curve over a reasonable range, and be sure to explain or at least comment on the endpoints of the trade-off curve.
- Plot the treatment formulation amounts versus k , and the bloodstream concentration versus t , for the two trade-off curve endpoints, and one corresponding to $T^{\text{ther}} = 8$.

Warning. We've found that CVX can experience numerical problems when solving this problem (depending on how it is formulated). In one case, `cvx_status` is "Solved/Inaccurate" when in fact the problem has been solved (just not to the tolerances SeDuMi likes to see). You can ignore this status, taking it to mean Optimal. You can also try switching to the SDPT3 solver. In any case, please do not spend much time worrying about, or dealing with, these numerical problems.

Solution. We formulate the solution as the following bi-criterion optimization problem:

$$\begin{aligned} & \text{minimize} && (J^{\text{ch}}, T^{\text{ther}}) \\ & \text{subject to} && c(t) \leq c^{\max}, \quad t = 1, \dots, T \\ & && c(t) \geq c^{\min}, \quad t = T^{\text{ther}}, \dots, T \\ & && a^{(k)} \succeq 0, \quad k = 1, \dots, K. \end{aligned}$$

The key to this problem is to recognize that the objective T^{ther} is quasiconvex. The problem as stated is convex for fixed values of T^{ther} . To solve the problem, then, we will solve a sequence of T LPs, with $T^{\text{ther}} = 1, \dots, T$. The code to do this is given below.

Note that an acceptable solution to this problem added the constraint that $c(T^{\text{ther}} - 1) < c^{\min}$, such that the fixed value of T^{ther} satisfies the definition. Points on the trade-off curve found this way need not be Pareto optimal (*i.e.*, minimal).

```

% solution for time release formulation
close all; clear all
time_release_form_data;

Tther = 1:T;

fastest = -1;
for i = 1:length(Tther)
    cvx_begin quiet
        variable a(m,K)
        c = zeros(1,T);

        % vector convolution performed manually
        for k = 1:K,
            P_shift = zeros(m,T);
            P_shift(:, tau(k):tau(k) + Tp-1) = P;
            P_shift = P_shift(:,1:T);    % truncate if exceed "week"
            c = c + sum((a(:,k)*ones(1,T)).*P_shift);
        end

        Jch = sum(norms(a(:,2:K) - a(:,1:K-1), Inf));

        minimize (Jch)
        subject to
            c <= cmax
            c(Tther(i):end) >= cmin
            a >= 0
    cvx_end

    if (strcmpi(cvx_status, 'Solved') && fastest == -1)
        fastest = i;
    end

    Jch_vals(i) = cvx_optval;
    C(i,:) = c;
    A{i} = a;
    if(abs(cvx_optval) <= 1e-6)
        break
    end
end

% make plots
figure
plot(Tther(1:length(A)), Jch_vals)
xlabel('Tther'); ylabel('Jch');

```

```

print -depsc 'time_release_tradeoff.eps'

p = 8;

figure
plot(1:T,C(fastest,:), 1:T, C(end,:), 1:T, C(p,:), ...
     1:T, cmin*ones(T,1), 'k--', 1:T, cmax*ones(T,1), 'k--' )
axis([0 T 0 5.5]);
xlabel('t'); ylabel('ct')
print -depsc 'time_release_bloodstream.eps'

figure
subplot(3,1,1)
plot(1:K, A{fastest}, 'k')
ylabel(['Tther' num2str(fastest)]); axis([1 K 0 40]);
subplot(3,1,2)
plot(1:K, A{p}, 'k')
ylabel('Tther8'); axis([1 K 0 40]);
subplot(3,1,3)
plot(1:K, A{end}, 'k')
ylabel(['Tther' num2str(length(A))]); axis([1 K 0 40]);
xlabel('k');
print -depsc 'time_release_formulation.eps'

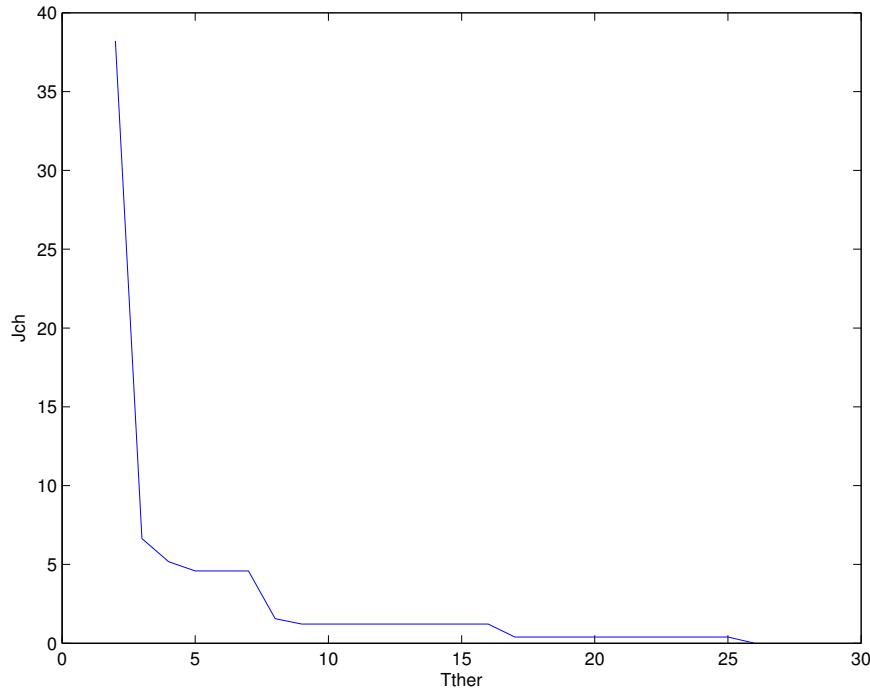
```

For small values of T^{ther} , the LP will be infeasible. This means that we cannot formulate a treatment that achieves the minimum therapeutic level by time T^{ther} . As we increase the value of T^{ther} , the LP becomes feasible.

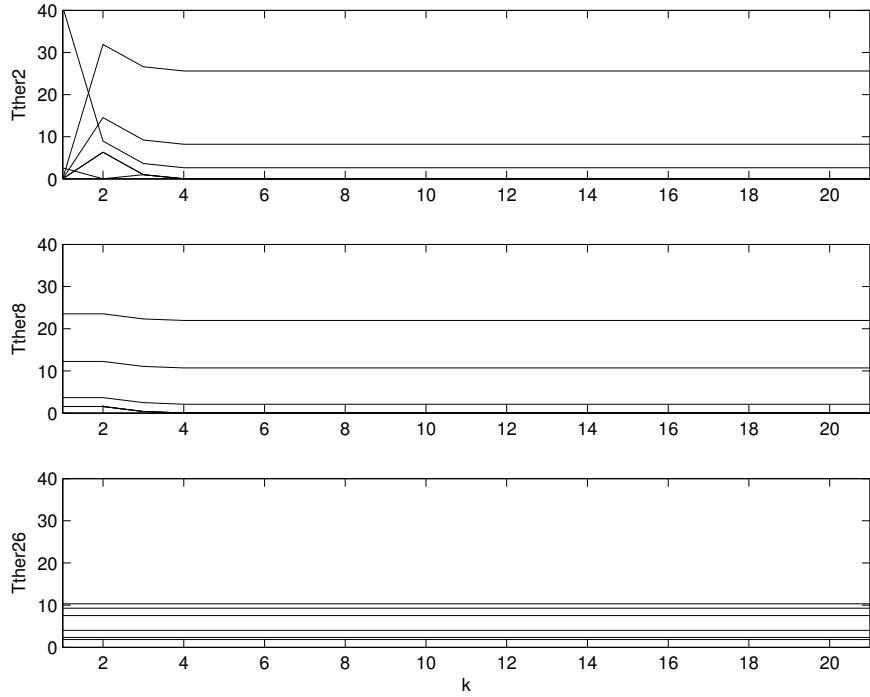
Once the LP becomes feasible, our solution gives the treatment that achieves the minimum therapeutic level as quickly as possible. However, J^{ch} may be large. This means that in order to achieve the minimum therapeutic level as quickly as possible, we must often change the formulation amounts in our treatments.

If we let T^{ther} become large enough, J^{ch} goes to 0, since we no longer have to achieve a minimum therapeutic level. At this point, our treatment consists of a constant drug formulation.

The figure below shows the optimal tradeoff curve. The problem is infeasible for $T^{\text{ther}} = 1$, so we cannot achieve the minimum therapeutic level after an hour of taking the pill. However, for $T^{\text{ther}} \geq 2$, we are able to achieve the minimum therapeutic level by some blend of our 6 drug formulations. Therefore, $T^{\text{ther}} = 2$ gives the fastest acting blend of the formulations; however, the price we pay is that we must form 4 different pills. For $T^{\text{ther}} \geq 26$, $J^{\text{ch}} = 0$: a single pill (constant blend) suffices to achieve the minimum therapeutic level after 26 hours. Note that the Pareto optimal frontier is *not* convex.

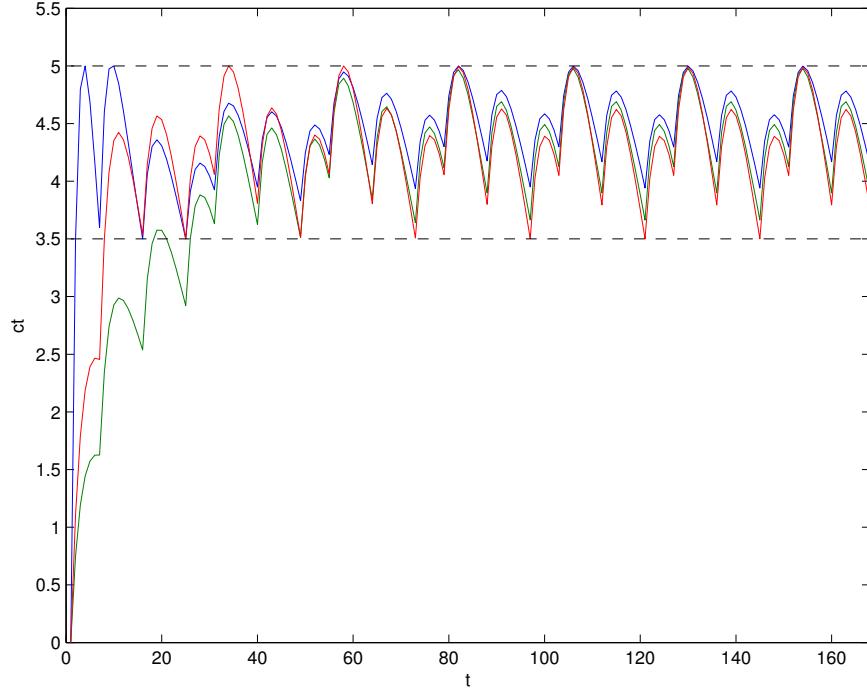


The next plot shows the treatment formulation amounts versus k for the three selected values of T^{ther} . Note that as T^{ther} increases, the formulation amounts vary less. For $T^{\text{ther}} = 2$, there are 4 different pills; for $T^{\text{ther}} = 8$, we have 2 or 3 different pills; and for $T^{\text{ther}} = 26$, we have a single pill.



The last plot shows the bloodstream concentration $c(t)$ versus t . The bloodstream concentrating

corresponding to $T^{\text{ther}} = 2$ (the fastest acting blend) is in blue; $T^{\text{ther}} = 8$ is in red; and $T^{\text{ther}} = 26$ (the constant blend) is in green.



17.7 Sizing a gravity feed water supply network. A water supply network connects water supplies (such as reservoirs) to consumers via a network of pipes. Water flow in the network is due to gravity (as opposed to pumps, which could also be added to the formulation). The network is composed of a set of n nodes and m directed edges between pairs of nodes. The first k nodes are supply or reservoir nodes, and the remaining $n - k$ are consumer nodes. The edges correspond to the pipes in the water supply network.

We let $f_j \geq 0$ denote the water flow in pipe (edge) j , and h_i denote the (known) altitude or height of node i (say, above sea level). At nodes $i = 1, \dots, k$, we let $s_i \geq 0$ denote the flow into the network from the supply. For $i = 1, \dots, n - k$, we let $c_i \geq 0$ denote the water flow taken out of the network (by consumers) at node $k + i$. Conservation of flow can be expressed as

$$Af = \begin{bmatrix} -s \\ c \end{bmatrix},$$

where $A \in \mathbf{R}^{n \times m}$ is the incidence matrix for the supply network, given by

$$A_{ij} = \begin{cases} -1 & \text{if edge } j \text{ leaves node } i \\ +1 & \text{if edge } j \text{ enters node } i \\ 0 & \text{otherwise.} \end{cases}$$

We assume that each edge is oriented from a node of higher altitude to a node of lower altitude; if edge j goes from node i to node l , we have $h_i > h_l$. The pipe flows are determined by

$$f_j = \frac{\alpha \theta_j R_j^2 (h_i - h_l)}{L_j},$$

where edge j goes from node i to node l , $\alpha > 0$ is a known constant, $L_j > 0$ is the (known) length of pipe j , $R_j > 0$ is the radius of pipe j , and $\theta_j \in [0, 1]$ corresponds to the valve opening in pipe j .

Finally, we have a few more constraints. The supply feed rates are limited: we have $s_i \leq S_i^{\max}$. The pipe radii are limited: we have $R_j^{\min} \leq R_j \leq R_j^{\max}$. (These limits are all known.)

- (a) *Supportable consumption vectors.* Suppose that the pipe radii are fixed and known. We say that $c \in \mathbf{R}_+^{n-k}$ is supportable if there is a choice of f , s , and θ for which all constraints and conditions above are satisfied. Show that the set of supportable consumption vectors is a polyhedron, and explain how to determine whether or not a given consumption vector is supportable.
- (b) *Optimal pipe sizing.* You must select the pipe radii R_j to minimize the cost, which we take to be (proportional to) the total volume of the pipes, $L_1 R_1^2 + \dots + L_m R_m^2$, subject to being able to support a set of consumption vectors, denoted $c^{(1)}, \dots, c^{(N)}$, which we refer to as consumption scenarios. (This means that any consumption vector in the convex hull of $\{c^{(1)}, \dots, c^{(N)}\}$ will be supportable.) Show how to formulate this as a convex optimization problem. *Note.* You are asked to choose *one* set of pipe radii, and N sets of valve parameters, flow vectors, and source vectors; one for each consumption scenario.
- (c) Solve the instance of the optimal pipe sizing problem with data defined in the file `grav_feed_network_data.m`, and report the optimal value and the optimal pipe radii. The columns of the matrix C in the data file are the consumption vectors $c^{(1)}, \dots, c^{(N)}$.

Hint. $-A^T h$ gives a vector containing the height differences across the edges.

Solution.

- (a) Let P be the set of vectors $(f, \theta, s, c) \in \mathbf{R}^m \times \mathbf{R}^m \times \mathbf{R}^k \times \mathbf{R}^{n-k}$ that satisfy

$$Af = \begin{bmatrix} -s \\ c \end{bmatrix}, \quad 0 \preceq \theta \preceq 1, \quad 0 \preceq f, \quad 0 \preceq c, \quad 0 \preceq s \preceq S^{\max}, \quad f = D\theta,$$

where $D = D_1 D_2$, with $D_1 = -\alpha \mathbf{diag}(A^T h) \mathbf{diag}(1/L_1, \dots, 1/L_m)$ and $D_2 = \mathbf{diag}(R_1^2, \dots, R_m^2)$. The set P is clearly a polyhedron. Therefore, the set of supportable consumption vectors c , which is just a projection of P , is also a polyhedron.

To determine whether or not a consumption vector $c \in \mathbf{R}_+^{n-k}$ is supportable, we can solve the following feasibility problem:

$$\begin{aligned} \text{find} \quad & (f, \theta, s) \\ \text{subject to} \quad & 0 \preceq f \\ & 0 \preceq \theta \preceq 1 \\ & Af = (-s, c) \\ & 0 \preceq s \preceq S^{\max} \\ & f = D\theta. \end{aligned}$$

(b) The problem that we would like to solve is

$$\begin{aligned}
& \text{minimize} && L_1 R_1^2 + \cdots + L_m R_m^2 \\
& \text{subject to} && R^{\min} \leq R \leq R^{\max} \\
& && 0 \leq f^{(p)}, \quad p = 1, \dots, N \\
& && 0 \leq \theta^{(p)} \leq 1, \quad p = 1, \dots, N \\
& && A f^{(p)} = (-s^{(p)}, c^{(p)}) \quad p = 1, \dots, N \\
& && 0 \leq s^{(p)} \leq S^{\max}, \quad p = 1, \dots, N \\
& && f^{(p)} = D_1 D_2 \theta^{(p)}, \quad p = 1, \dots, N.
\end{aligned}$$

This problem, however, is not convex in the variables $R, f^{(p)}, s^{(p)}, \theta^{(p)}$ since the last inequality constraint is not a convex constraint.

The first trick is to not use the valve parameters directly and replace θ with $\mathbf{1}$, i.e.

$$f^{(p)} \leq D_1 D_2 \mathbf{1}.$$

After finding the flows and pipe radii, we can simply set

$$\theta^{(p)} = (D_1 D_2)^{-1} f^{(p)} \in [0, 1]^m.$$

This is a valid transformation, since D_1 and D_2 are diagonal and invertible, so θ can be recovered uniquely.

However, the constraint $f^{(p)} \leq D_1 D_2 \mathbf{1}$ is still not convex because D_2 is quadratic in R . The second trick is to do a change of variables $z = D_2 \mathbf{1} = (R_1^2, \dots, R_m^2)$. This is a valid change of variables since $R \geq 0$.

Using these transformations, we obtain the equivalent problem

$$\begin{aligned}
& \text{minimize} && L_1 z_1 + \cdots + L_m z_m \\
& \text{subject to} && (R_j^{\min})^2 \leq z_j \leq (R_j^{\max})^2, \quad j = 1, \dots, m \\
& && 0 \leq f^{(p)}, \quad p = 1, \dots, N \\
& && A f^{(p)} = (-s^{(p)}, c^{(p)}) \quad p = 1, \dots, N \\
& && 0 \leq s^{(p)} \leq S^{\max}, \quad p = 1, \dots, N \\
& && f^{(p)} \leq D_1 z, \quad p = 1, \dots, N,
\end{aligned}$$

which is now a convex problem.

(c) The following Matlab script solves the problem.

```

grav_feed_network_data;
D1 = diag(alpha*(-A'*h))*diag(1./L);

cvx_begin
variables z(m) F(m,N) S(k,N)
minimize(L'*z)
subject to
F >= 0
Rmin.^2 <= z
z <= Rmax.^2

```

```

A*F == [-S;C]
S >= 0
max(S')' <= Smax
max(F')' <= D1*z
cvx_end
R = sqrt(z); % pipe radii
D2 = diag(R.^2);
theta = inv(D1*D2)*F; % valve opening

```

The optimal value was found to be 313.024 and the pipe radii are given in the table below.

pipe	radius
1	0.5
2	0.5
3	2.21
4	0.5
5	0.5
6	0.5
7	1.93
8	1.23
9	0.5
10	2.5
11	2.5
12	0.5
13	0.5
14	1.20
15	0.5
16	2.5
17	1.05
18	1.89
19	1.73
20	0.5

17.8 Optimal political positioning. A political constituency is a group of voters with similar views on a set of political issues. The electorate (*i.e.*, the set of voters in some election) is partitioned (by a political analyst) into K constituencies, with (nonnegative) populations P_1, \dots, P_K . A candidate in the election has an initial or prior position on each of n issues, but is willing to consider (presumably small) deviations from her prior positions in order to maximize the total number of votes she will receive. We let $x_i \in \mathbf{R}$ denote the change in her position on issue i , measured on some appropriate scale. (You can think of $x_i < 0$ as a move to the ‘left’ and $x_i > 0$ as a move to the ‘right’ on the issue, if you like.) The vector $x \in \mathbf{R}^n$ characterizes the changes in her position on all issues; $x = 0$ represents the prior positions. On each issue she has a limit on how far in each direction she is willing to move, which we express as $l \preceq x \preceq u$, where $l \prec 0$ and $u \succ 0$ are given.

The candidate’s position change x affects the fraction of voters in each constituency that will vote for her. This fraction is modeled as a logistic function,

$$f_k = g(w_k^T x + v_k), \quad k = 1, \dots, K.$$

Here $g(z) = 1/(1 + \exp(-z))$ is the standard logistic function, and $w_k \in \mathbf{R}^n$ and $v_k \in \mathbf{R}$ are given data that characterize the views of constituency k on the issues. Thus the total number of votes the candidate will receive is

$$V = P_1 f_1 + \cdots + P_K f_K.$$

The problem is to choose x (subject to the given limits) so as to maximize V . The problem data are l , u , and P_k , w_k , and v_k for $k = 1, \dots, K$.

- (a) *The general political positioning problem.* Show that the objective function V need not be quasiconcave. (This means that the general optimal political positioning problem is not a quasiconvex problem, and therefore also not a convex problem.) In other words, choose problem data for which V is not a quasiconcave function of x .
- (b) *The partisan political positioning problem.* Now suppose the candidate focuses only on her core constituencies, *i.e.*, those for which a significant fraction will vote for her. In this case we interpret the K constituencies as her core constituencies; we assume that $v_k \geq 0$, which means that with her prior position $x = 0$, at least half of each of her core constituencies will vote for her. We add the constraint that $w_k^T x + v_k \geq 0$ for each k , which means that she will not take positions that alienate a majority of voters from any of her core constituencies. Show that the partisan political positioning problem (*i.e.*, maximizing V with the additional assumptions and constraints) is convex.
- (c) *Numerical example.* Find the optimal positions for the partisan political positioning problem with data given in `opt_pol_pos_data.m`. Report the number of votes from each constituency under the politician's prior positions ($x = 0$) and optimal positions, as well as the total number of votes V in each case.

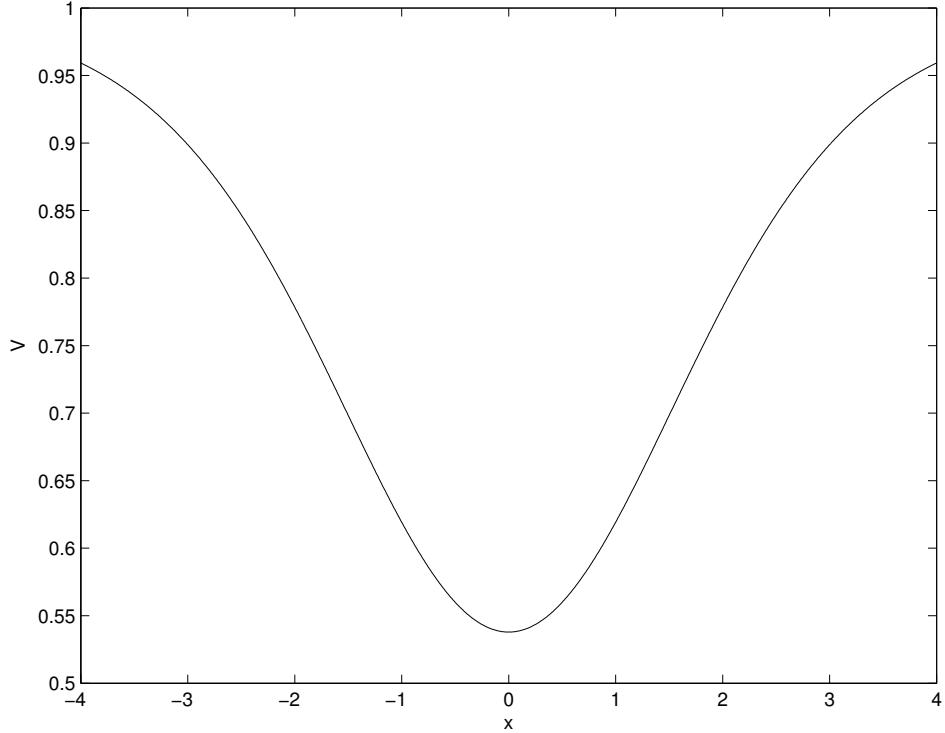
You may use the function

$$g_{\text{approx}}(z) = \min\{1, g(i) + g'(i)(z - i) \text{ for } i = 0, 1, 2, 3, 4\}$$

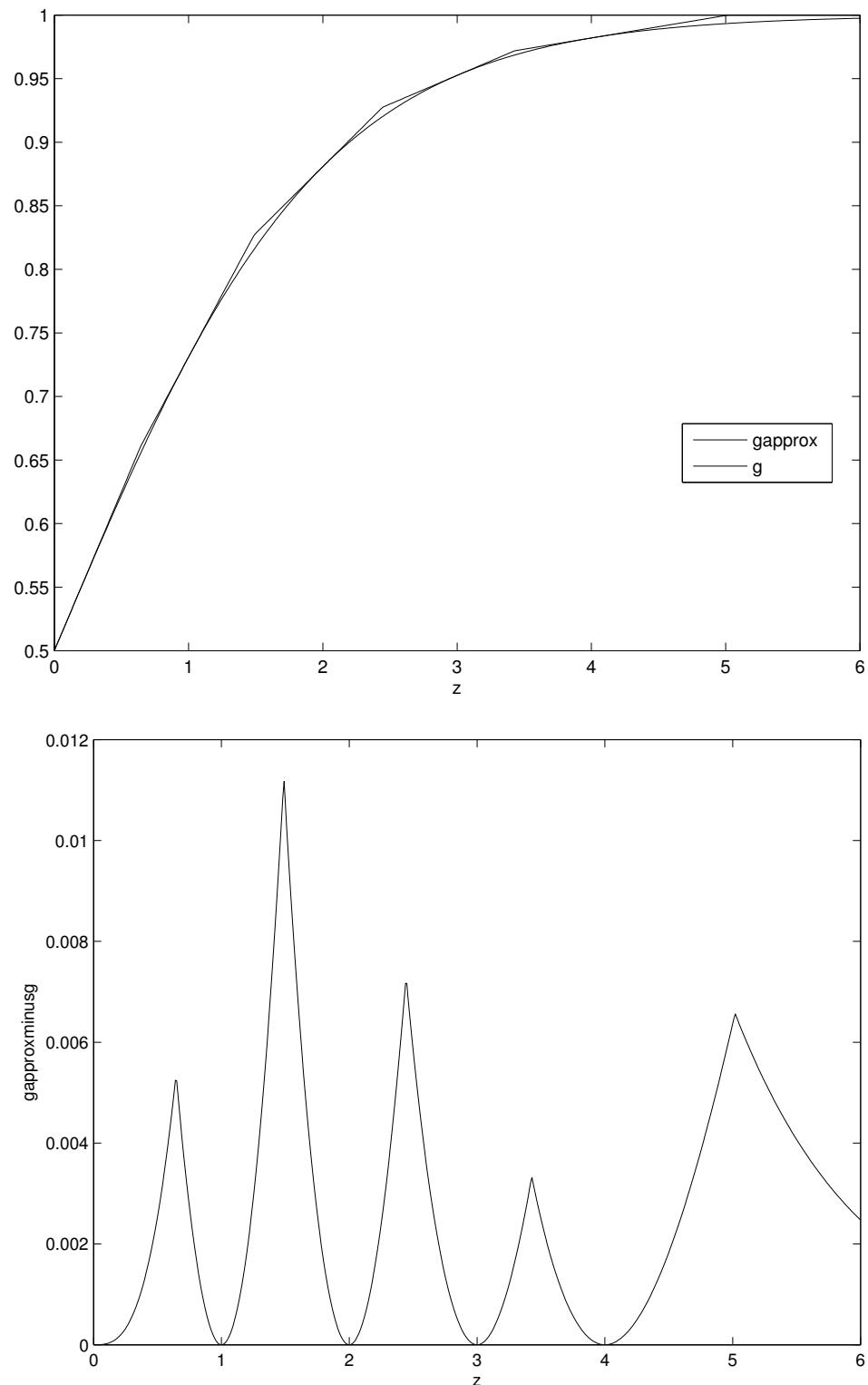
as an approximation of g for $z \geq 0$. (The function g_{approx} is also an upper bound on g for $z \geq 0$.) For your convenience, we have included function definitions for g and g_{approx} (`g` and `gapx`, respectively) in the data file. You should report the results (votes from each constituency and total) using g , but be sure to check that these numbers are close to the results using g_{approx} (say, within one percent or so).

Solution.

- (a) The simplest counterexample has only one issue and two constituencies. We can take (for example) $w_1 = 1$, $w_2 = -1$, and $v_1 = v_2 < 0$, and (say) $P_1 = P_2 = 1$. This means that neither constituency is very supportive of the position $x = 0$; one wants the position to be negative, and the other positive. As a result, the total number of votes V increases for either x decreasing or increasing. It follows that V is not quasiconcave. The plot below shows V as a function of x .



- (b) The logistic function g is neither convex nor concave. But if we restrict the argument to nonnegative values, it is concave. In the partisan positioning problem, our constraint of not alienating any constituency, *i.e.*, $w_k^T x + v_k \geq 0$, means that for feasible x , $g(w_k^T x + v_k)$ is concave. Thus the total number of votes V , which is a nonnegative weighted sum of the concave functions $g(w_k^T x + v_k)$, is also concave. The constraints, which are the lower and upper bounds on x , as well as the constraints $w_k^T x + v_k \geq 0$, are clearly convex.
- (c) We didn't ask you to check the given approximation of g , but we do so here for the record. The first plot below shows g and g_{approx} ; the second shows the error $g_{\text{approx}} - g$. The maximum error is around .01. (We could easily reduce this adding more terms to the maximum.)



The following code solves the problem:

```
% optimal political positioning.
```

```

% ee364a
opt_pol_pos_data;

% Counterexample to quasiconvexity for general political positioning problem
plot_counterexample = true;
if plot_counterexample
    x = (-4:.01:4)';
    v_counterexample = -1;
    figure;
    plot(x,g(x+v_counterexample)+g(-x+v_counterexample))
    xlabel('x')
    ylabel('V')
    print -deps opt_pol_pos_counterexample.eps
end

% Plot approximation quality
plot_apx_quality = true
if plot_apx_quality
    x = 0:.01:6;
    figure;
    plot(x,gapx(x), 'r', x, g(x), 'b')
    legend('gapprox', 'g', 'Location', 'Best')
    xlabel('z')
    print -deps opt_pol_pos_logit_approx.eps;

    figure;
    plot(x,gapx(x)-g(x))
    ylabel('gapproxminusg')
    xlabel('z')
    print -deps opt_pol_pos_logit_error.eps;
end

% Compute optimal positions for partisan political positioning problem
cvx_begin
    variable x(n)
    maximize(P'*gapx(W*x + v))
    subject to
        x >= l;
        x <= u;
        W*x + v >= 0;
cvx_end

% Vote before position optimization
Vi = P'*g(v)

```

```

Viapx = P'*gapx(v)
% Vote after position optimization
Vf = P'*g(W*x + v)
Vfapx = P'*gapx(W*x + v)
% Change in vote per constituency
delta = [P.*g(v),P.*g(W*x+v)]

```

The total vote improves by 16% from 373432 to 433884. The changes in votes for each constituency are given below. We can see that the change in positions increases the number of votes from most constituencies, but also lowers the number of votes in a few.

These numbers are computed using g , not the approximation g_{approx} . But if we calculate these numbers using g_{approx} , they differ by less than 1%. A stronger statement can be made by noting that g_{approx} is an upper bound on g when the argument is nonnegative, so the optimal value of our problem using g_{approx} , which is 435579, is an upper bound on the true optimal value. Since the objective value we obtained is 433884, we are guaranteed that this is no more than 0.4% suboptimal. (We didn't ask you to do this analysis.)

Initial Vote	Final Vote
48295	52894
21352	28223
27384	40443
26583	22150
46370	47055
41511	35228
25744	34561
14850	10463
31419	43309
24806	21445
35174	41241
29945	56873

17.9 Resource allocation in stream processing. A large data center is used to handle a stream of J types of jobs. The traffic (number of instances per second) of each job type is denoted $t \in \mathbf{R}_+^J$. Each instance of each job type (serially) invokes or calls a set of processes. There are P types of processes, and we describe the job-process relation by the $P \times J$ matrix

$$R_{pj} = \begin{cases} 1 & \text{job } j \text{ invokes process } p \\ 0 & \text{otherwise.} \end{cases}$$

The process loads (number of instances per second) are given by $\lambda = Rt \in \mathbf{R}^P$, i.e., λ_p is the sum of the traffic from the jobs that invoke process p .

The latency of a process or job type is the average time that it takes one instance to complete. These are denoted $l^{\text{proc}} \in \mathbf{R}^P$ and $l^{\text{job}} \in \mathbf{R}^J$, respectively, and are related by $l^{\text{job}} = R^T l^{\text{proc}}$, i.e., l_j^{job} is the sum of the latencies of the processes called by j . Job latency is important to users, since l_j^{job} is the average time the data center takes to handle an instance of job type j . We are given a maximum allowed job latency: $l^{\text{job}} \preceq l^{\max}$.

The process latencies depend on the process load and also how much of n different resources are made available to them. These resources might include, for example, number of cores, disk storage, and network bandwidth. Here, we represent amounts of these resources as (nonnegative) real numbers, so $x_p \in \mathbf{R}_+^n$ represents the resources allocated to process p . The process latencies are given by

$$l_p^{\text{proc}} = \psi_p(x_p, \lambda_p), \quad p = 1, \dots, P,$$

where $\psi_p : \mathbf{R}^n \times \mathbf{R} \rightarrow \mathbf{R} \cup \{\infty\}$ is a known (extended-valued) convex function. These functions are nonincreasing in their first (vector) arguments, and nondecreasing in their second arguments (*i.e.*, more resources or less load cannot increase latency). We interpret $\psi_p(x_p, \lambda_p) = \infty$ to mean that the resources given by x_p are not sufficient to handle the load λ_p .

We wish to allocate a total resource amount $x^{\text{tot}} \in \mathbf{R}_{++}^n$ among the P processes, so we have $\sum_{p=1}^P x_p \preceq x^{\text{tot}}$. The goal is to minimize the objective function

$$\sum_{j=1}^J w_j(t_j^{\text{tar}} - t_j)_+,$$

where t_j^{tar} is the target traffic level for job type j , $w_j > 0$ give the priorities, and $(u)_+$ is the nonnegative part of a vector, *i.e.*, $u_i = \max\{u_i, 0\}$. (Thus the objective is a weighted penalty for missing the target job traffic.) The variables are $t \in \mathbf{R}_+^J$ and $x_p \in \mathbf{R}_+^n$, $p = 1, \dots, P$. The problem data are the matrix R , the vectors l^{max} , x^{tot} , t^{tar} , and w , and the functions ψ_p , $p = 1, \dots, P$.

- (a) Explain why this is a convex optimization problem.
- (b) Solve the problem instance with data given in `res_alloc_stream_data.m`, with latency functions

$$\psi_p(x_p, \lambda_p) = \begin{cases} 1/(a_p^T x_p - \lambda_p) & a_p^T x_p > \lambda_p, \quad x_p \succeq x_p^{\min} \\ \infty & \text{otherwise} \end{cases}$$

where $a_p \in \mathbf{R}_{++}^n$ and $x_p^{\min} \in \mathbf{R}_{++}^n$ are given data. The vectors a_p and x_p^{\min} are stored as the columns of the matrices \mathbf{A} and $\mathbf{x_min}$, respectively.

Give the optimal objective value and job traffic. Compare the optimal job traffic with the target job traffic.

Solution.

- (a) The problem is

$$\begin{aligned} & \text{minimize} && \sum_{j=1}^J w_j(t_j^{\text{tar}} - t_j)_+ \\ & \text{subject to} && \sum_p x_p \preceq x^{\text{tot}} \\ & && x_p \succeq 0, \quad p = 1, \dots, P \\ & && t \succeq 0 \\ & && \lambda = Rt \\ & && l^{\text{job}} = R^T l^{\text{proc}} \\ & && l^{\text{job}} \preceq l^{\text{max}} \end{aligned}$$

with variables x_p and t , where $l_p^{\text{proc}} = \psi_p(x_p, \lambda_p)$.

The variables are x_p and t , so the constraints $t \succeq 0$, $x_p \succeq 0$, and $\sum_p x_p \preceq x^{\text{tot}}$ are convex, as are the implicit constraints $\psi_p(x_p, \lambda_p) < \infty$. The objective is a positive weighted sum of

convex functions of the variables, and so is convex. The job traffic t is affine in the variables, so each entry of l^{proc} is a convex function of the variables, and therefore (since the coefficients of R^T are nonnegative) so is every entry of l^{job} . Thus the constraints $l^{\text{job}} \preceq l^{\text{max}}$ are convex.

The constraint $x_p \succeq x_p^{\min}$ used in part (b) is also convex.

- (b) The following code solves the given problem instance.

```
% resource allocation for stream processing.

res_alloc_stream_data;

% solution

cvx_begin
    variable x(n,P) % resource allocation
    variable t(J)   % job traffic
    t >= 0
    sum(x') <= x_tot'; % resource limits
    lambda = R*t; % process loads
    x >= x_min % minimum allowable resources for the processes
    lproc = inv_pos(sum(A.*x)-lambda)'; % process latencies
    ljob = R'*lproc; % job latencies
    ljob <= l_max; % job latency limit
    minimize (w'*pos(t_tar-t))
cvx_end

cvx_optval

[t t_tar]

[ljob l_max]
```

The optimal objective value is 7.74. All job types are handled at their target loads, except for job 4. (We should not be surprised by sparsity of the job load missed target vector.) All the latency limits are respected (as we require); some of the jobs have latency smaller than the maximum allowed value.

17.10 Optimal parimutuel betting. In *parimutuel betting*, participants bet nonnegative amounts on each of n outcomes, exactly one of which will actually occur. (For example, the outcome can be which of n horses wins a race.) The total amount bet by all participants on all outcomes is called the *pool* or *tote*. The house takes a commission from the pool (typically around 20%), and the remaining pool is divided among those who bet on the outcome that occurs, in proportion to their bets on the outcome. This problem concerns the choice of the amount to bet on each outcome.

Let $x_i \geq 0$ denote the amount we bet on outcome i , so the total amount we bet on all outcomes is $\mathbf{1}^T x$. Let $a_i > 0$ denote the amount bet by all other participants on outcome i , so after the house commission, the remaining pool is $P = (1 - c)(\mathbf{1}^T a + \mathbf{1}^T x)$, where $c \in (0, 1)$ is the house commission

rate. Our *payoff* if outcome i occurs is then

$$p_i = \left(\frac{x_i}{x_i + a_i} \right) P.$$

The goal is to choose x , subject to $\mathbf{1}^T x = B$ (where B is the total amount to be bet, which is given), so as to maximize the expected utility

$$\sum_{i=1}^n \pi_i U(p_i),$$

where π_i is the probability that outcome i occurs, and U is a concave increasing utility function, with $U(0) = 0$. You can assume that a_i , π_i , c , B , and the function U are known.

- (a) Explain how to find an optimal x using convex or quasiconvex optimization. If you use a change of variables, be sure to explain how your variables are related to x .
- (b) Suggest a fast method for computing an optimal x . You can assume that U is strictly concave, and that scalar optimization problems involving U (such as evaluating the conjugate of $-U$) are easily and quickly solved.

Remarks.

- To carry out this betting strategy, you'd need to know a_i , and then be the last participant to place your bets (so that a_i don't subsequently change). You'd also need to know the probabilities π_i . These could be estimated using sophisticated machine learning techniques or insider information.
- The formulation above assumes that the total amount to bet (*i.e.*, B) is known. If it is not known, you could solve the problem above for a range of values of B and use the value of B that yields the largest optimal expected utility.

Solution.

- (a) It turns out the problem is convex, exactly as stated. The constraints $\mathbf{1}^T x = B$ and $x \succeq 0$ are clearly convex. The objective is concave, which can be seen as follows. Since we know $\mathbf{1}^T x = B$, the remaining pool is a (known) constant, $P = (1 - c)(\mathbf{1}^T a + B)$. The objective is a convex combination of terms of the form

$$U \left(\frac{Px_i}{x_i + a_i} \right),$$

which are concave functions of x . To see this we note that $x_i/(x_i + a_i)$ is concave in x_i for $x_i \geq 0$, and by the composition rules, a concave increasing function of a concave function is concave.

- (b) The problem can be expressed as

$$\begin{aligned} & \text{maximize} && \sum_{i=1}^n f_i(x_i) \\ & \text{subject to} && \mathbf{1}^T x = B, \quad x \succeq 0, \end{aligned}$$

where

$$f_i(x_i) = \pi_i U(Px_i/(x_i + a_i))$$

(which are concave functions of x_i , as noted in part (a)). We can use a water-filling method to solve the problem.

We form the partial Lagrangian (keeping $x \succeq 0$ implicit)

$$L(x, \nu) = \sum_{i=1}^n f_i(x_i) + \nu(B - \mathbf{1}^T x),$$

with $\text{dom } L = \mathbf{R}_+^n \times \mathbf{R}$. This is separable in x , so to maximize it over x we simply maximize over each x_i separately:

$$x_i = \underset{x_i \geq 0}{\operatorname{argmax}} (f_i(x_i) - \nu x_i).$$

Our assumption that U is strictly concave implies that f_i is strictly concave, so this has a unique solution. This is a condition sufficient to allow us to solve the primal problem by solving the dual. These scalar maximization problems are easily solved (that's our assumption), so we need only adjust ν so that $\mathbf{1}^T x = B$. Since each x_i is monotone nonincreasing in ν , we can use bisection.

When ν is negative, the problems are unbounded above, so we have $\nu \geq 0$. When $\nu \geq \max_i f'_i(0)$ (we assume here f_i are differentiable at 0), we have $b = 0$. So we can start bisection with the initial interval $\nu \in [0, \max_i f'_i(0)]$.

Each step of the bisection requires solving n scalar optimization problems, and we have to perform a modest number (say, 20) bisection steps.

17.11 Perturbing a Hamiltonian to maximize an energy gap. A finite dimensional approximation of a quantum mechanical system is described by its Hamiltonian matrix $H \in \mathbf{S}^n$. We label the eigenvalues of H as $\lambda_1 \leq \dots \leq \lambda_n$, with corresponding orthonormal eigenvectors v_1, \dots, v_n . In this context the eigenvalues are called the energy levels of the system, and the eigenvectors are called the eigenstates. The eigenstate v_1 is called the ground state, and λ_1 is the ground energy. The energy gap (between the ground and next state) is $\eta = \lambda_2 - \lambda_1$.

By changing the environment (say, applying external fields), we can perturb a nominal Hamiltonian matrix to obtain the perturbed Hamiltonian, which has the form

$$H = H^{\text{nom}} + \sum_{i=1}^k x_i H_i.$$

Here $H^{\text{nom}} \in \mathbf{S}^n$ is the nominal (unperturbed) Hamiltonian, $x \in \mathbf{R}^k$ gives the strength or value of the perturbations, and $H_1, \dots, H_k \in \mathbf{S}^n$ characterize the perturbations. We have limits for each perturbation, which we express as $|x_i| \leq 1$, $i = 1, \dots, k$. The problem is to choose x to maximize the gap η of the perturbed Hamiltonian, subject to the constraint that the perturbed Hamiltonian H has the same ground state (up to scaling, of course) as the unperturbed Hamiltonian H^{nom} . The problem data are the nominal Hamiltonian matrix H^{nom} and the perturbation matrices H_1, \dots, H_k .

- (a) Explain how to formulate this as a convex or quasiconvex optimization problem. If you change variables, explain the change of variables clearly.

- (b) Carry out the method of part (a) for the problem instance with data given in `hamiltonian_gap_data.m`. Give the optimal perturbations, and the energy gap for the nominal and perturbed systems. The data H_i are given as a cell array; $H\{i\}$ gives H_i .

Solution.

- (a) Let v^{nom} be the ground state of the nominal Hamiltonian. The condition that it be an eigenstate of the perturbed Hamiltonian is $Hv^{\text{nom}} = \lambda v^{\text{nom}}$ for some $\lambda \in \mathbf{R}$, which is a linear equality constraint on x and λ . The condition $\lambda_{\min}(H) \geq \lambda$, which is convex, ensures that λ is the ground energy level of the perturbed Hamiltonian. (It can be shown that this constraint is not needed.)

We still have to deal with the gap. In general, the second lowest eigenvalue of a symmetric matrix is not a concave function. But here we know the eigenvector associated with the lowest eigenvalue. The trick is to form a new matrix whose smallest eigenvalue is exactly λ_2 for the perturbed Hamiltonian.

One way is to project onto the subspace orthogonal to v^{nom} , which we do by finding a matrix $Q \in \mathbf{R}^{(n-1) \times n}$ whose columns are an orthonormal basis for $\mathcal{N}(v^{\text{nom}})$. The matrix $Q^T HQ$, which is in \mathbf{S}^{n-1} , has eigenvalues $\lambda_2(H), \dots, \lambda_n(H)$, so $\lambda_{\min}(Q^T HQ) = \lambda_2$. This is a concave function of x , so we're done. We end up with the problem

$$\begin{aligned} &\text{maximize} && \lambda_{\min}(Q^T HQ) - \lambda \\ &\text{subject to} && Hv^{\text{nom}} = \lambda v^{\text{nom}}, \quad \|x\|_\infty \leq 1, \end{aligned}$$

with variables x and λ .

There are several other ways to get a handle on λ_2 . The matrix

$$\tilde{H} = H + \alpha v^{\text{nom}}(v^{\text{nom}})^T$$

has eigenvalues

$$\lambda_1(H) + \alpha, \quad \lambda_2(H), \dots, \lambda_n(H).$$

Thus we have

$$\lambda_{\min}(\tilde{H}) = \min\{\lambda_1(H) + \alpha, \lambda_2(H)\}.$$

This is a concave function of x and α ; if we maximize this function over α , we can take any value that ends up with

$$\lambda_{\min}(\tilde{H}) = \lambda_2(H),$$

which is just what we wanted. So another formulation is

$$\begin{aligned} &\text{maximize} && \lambda_{\min}(H + \alpha v^{\text{nom}}(v^{\text{nom}})^T) - \lambda \\ &\text{subject to} && Hv^{\text{nom}} = \lambda v^{\text{nom}}, \quad \|x\|_\infty \leq 1, \end{aligned}$$

with variables x , α , and λ .

- (b) The following code solves the problem (using both ways, just to check).

```
% hamiltonian perturbation optimization
% ee364a, s. boyd
hamiltonian_gap_data;
```

```

[V,D]=eig(Hnom);
v_nom = V(:,1); % nominal ground state

cvx_begin
    variables x(k) lambda
    variable alpha
    x >= -1;
    x <= 1;
    H = Hnom;
    for i=1:k
        H=H+x(i)*Hpert{i};
    end
    H*v_nom == lambda*v_nom; % preserve ground state
    % first method
    Q= null(v_nom'); % columns of Q are o.n. basis for v_nom^\perp
    maximize (lambda_min(Q'*H*Q)-lambda)
    % second method
    %maximize (lambda_min(H + alpha*v_nom*v_nom') -lambda)
cvx_end

```

x

```

nom_energies = eig(Hnom);
pert_energies = eig(H);

[nom_energies pert_energies]

nom_gap = nom_energies(2)-nom_energies(1)

pert_gap = pert_energies(2)-pert_energies(1)

```

The optimal perturbations are given below. We can see that several of the perturbations are at the limits. The energies for the unperturbed and perturbed systems are also given. To increase the gap the perturbations have reduced the ground energy and also increased the second energy level. The gap increases by a factor around two, from 3.56 to 7.14.

```

x =

0.3546
0.1552
0.8463
-0.5188
0.1474
-0.7314
0.3515
-1.0000

```

```

-0.4400
-0.2201
-0.3730
 0.6952
-0.5168
-1.0000
-1.0000

```

`ans =`

```

-7.2432 -21.2974
-3.6805 -14.1566
-2.1072 -12.7372
-1.3870 -10.2474
-1.0429 -5.4235
-0.2360 -1.8341
 2.4893 -1.2973
 3.6856  3.1430
 5.5773 11.8985
 8.2111 22.3972

```

`nom_gap =`

3.5627

`pert_gap =`

7.1408

17.12 *Theory-applications split in a course.* A professor teaches an advanced course with 20 lectures, labeled $i = 1, \dots, 20$. The course involves some interesting theoretical topics, and many practical applications of the theory. The professor must decide how to split each lecture between theory and applications. Let T_i and A_i denote the fraction of the i th lecture devoted to theory and applications, for $i = 1, \dots, 20$. (We have $T_i \geq 0$, $A_i \geq 0$, and $T_i + A_i = 1$.)

A certain amount of theory has to be covered before the applications can be taught. We model this in a crude way as

$$A_1 + \cdots + A_i \leq \phi(T_1 + \cdots + T_i), \quad i = 1, \dots, 20,$$

where $\phi : \mathbf{R} \rightarrow \mathbf{R}$ is a given nondecreasing function. We interpret $\phi(u)$ as the cumulative amount of applications that can be covered, when the cumulative amount of theory covered is u . We will

use the simple form $\phi(u) = a(u - b)_+$, with $a, b > 0$, which means that no applications can be covered until b lectures of the theory is covered; after that, each lecture of theory covered opens the possibility of covering a lectures on applications.

The theory-applications split affects the emotional state of students differently. We let s_i denote the emotional state of a student after lecture i , with $s_i = 0$ meaning neutral, $s_i > 0$ meaning happy, and $s_i < 0$ meaning unhappy. Careful studies have shown that s_i evolves via a linear recursion (dynamics)

$$s_i = (1 - \theta)s_{i-1} + \theta(\alpha T_i + \beta A_i), \quad i = 1, \dots, 20,$$

with $s_0 = 0$. Here α and β are parameters (naturally interpreted as how much the student likes or dislikes theory and applications, respectively), and $\theta \in [0, 1]$ gives the emotional volatility of the student (*i.e.*, how quickly he or she reacts to the content of recent lectures). The student's terminal emotional state is s_{20} .

Now consider a specific instance of the problem, with course material parameters $a = 2$, $b = 3$, and three groups of students, with emotional dynamics parameters given as follows.

	Group 1	Group 2	Group 3
θ	0.05	0.1	0.3
α	-0.1	0.8	-0.3
β	1.4	-0.3	0.7

Find (four different) theory-applications splits that maximize the terminal emotional state of the first group, the terminal emotional state of the second group, the terminal emotional state of the third group, and, finally, the minimum of the terminal emotional states of all three groups.

For each case, plot T_i and the emotional state s_i for the three groups, versus i . Report the numerical values of the terminal emotional states for each group, for each of the four theory-applications splits.

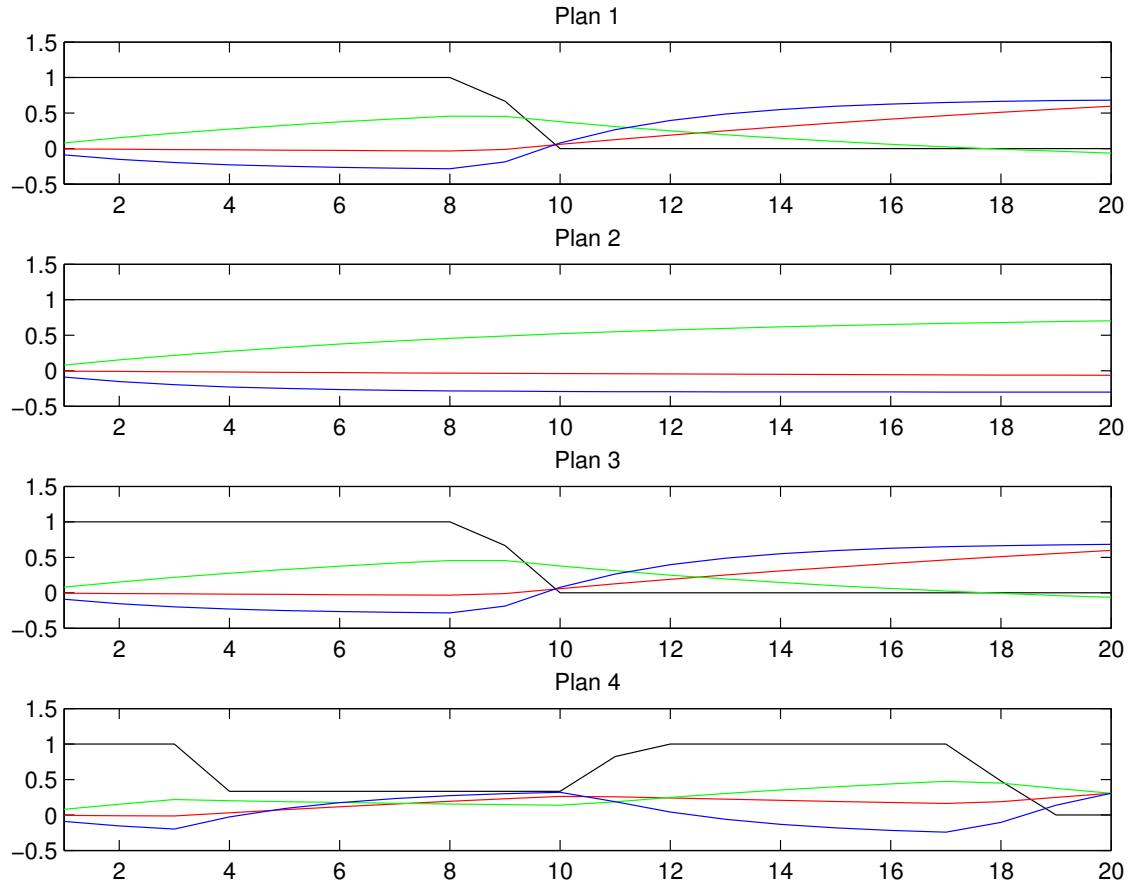
Solution. Because of the way that ϕ was chosen, the first b lectures have to be theory only, *i.e.*, $T_i = 1$, $A_i = 0$ for $i = 1, \dots, b$. Using this observation, we can rewrite the condition on the theory-applications split:

$$A_{b+1} + \dots + A_i \leq a(T_{b+1} + \dots + T_i), \quad i = b+1, \dots, 20.$$

This is a linear inequality in the variables T_{b+1}, \dots, T_{20} , and A_{b+1}, \dots, A_{20} .

Note that each s_i is also linear in the same set of variables. Thus, maximizing the given objective functions is a linear program.

Solving the numerical instance gives the following plots. The black curve shows T_i . The red, green, and blue curves show the emotional states of the three student groups.



The terminal emotional states for the three groups under the four different lecture plans are given by the table below.

	Group 1	Group 2	Group 3
Plan 1	0.597	-0.064	0.682
Plan 2	-0.064	0.703	-0.300
Plan 3	0.597	-0.064	0.682
Plan 4	0.306	0.306	0.306

The following code solves the problem and generates the plots shown above.

```
% theory-applications split in a course
clear all; clf;

% set up the parameters
a = 2; b = 3;
theta = [ 0.05; 0.1; 0.3];
alpha = [-0.1; 0.8; -0.3];
beta = [ 1.4; -0.3; 0.7];
n = 20; % number of lectures
```

```

m = 3; % number of student groups

for plan = 1:m+1
    cvx_begin quiet
        variable T(n)
        expressions s(m, n+1) obj

        % compute the emotional states
        for i = 1:n
            s(:, i+1) = (1-theta).*s(:, i) ...
                + theta.*((alpha*T(i)+beta*(1-T(i))));
        end

        if plan == 4
            obj = min(s(:, n+1));
        else
            obj = s(plan, n+1);
        end

        maximize obj
        subject to
            T >= 0;
            T <= 1;
            T(1:b) == 1;
            cumsum(1-T(b+1:n)) <= a*cumsum(T(b+1:n));
    cvx_end

    % plot
    subplot(4, 1, plan);
    plot(1:n, T, 'k', ...
        1:n, s(1, 2:n+1), 'r', ...
        1:n, s(2, 2:n+1), 'g', ...
        1:n, s(3, 2:n+1), 'b');
    title(sprintf('Plan %d', plan));
    axis([1 n -0.5 1.5]);
    fprintf('Plan %d: %f %f %f\n', plan, ...
        s(1, n+1), s(2, n+1), s(3, n+1));
end

print -depsc theory_appls.eps;

```

17.13 Lyapunov analysis of a dynamical system. We consider a discrete-time time-varying linear dynamical system with state $x_t \in \mathbf{R}^n$. The state propagates according to the linear recursion $x_{t+1} = A_t x_t$, for $t = 0, 1, \dots$, where the matrices A_t are unknown but satisfy $A_t \in \mathcal{A} = \{A^{(1)}, \dots, A^{(K)}\}$, where $A^{(1)}, \dots, A^{(K)}$ are known. (In computer science, this would be called a non-deterministic linear automaton.) We call the sequence x_0, x_1, \dots a *trajectory* of the system. There are infinitely many

trajectories, one for each sequence A_0, A_1, \dots .

The *Lyapunov exponent* κ of the system is defined as

$$\kappa = \sup_{A_0, A_1, \dots} \limsup_{t \rightarrow \infty} \|x_t\|_2^{1/t}.$$

(If you don't know what sup and lim sup mean, you can replace them with max and lim, respectively.) Roughly speaking, this means that all trajectories grow no faster than κ^t . When $\kappa < 1$, the system is called *exponentially stable*.

It is a hard problem to determine the Lyapunov exponent of the system, or whether the system is exponentially stable, given the data $A^{(1)}, \dots, A^{(K)}$. In this problem we explore a powerful method for computing an upper bound on the Lyapunov exponent.

- (a) Let $P \in \mathbf{S}_{++}^n$ and define $V(x) = x^T P x$. Suppose V satisfies

$$V(A^{(i)}x) \leq \gamma^2 V(x) \text{ for all } x \in \mathbf{R}^n, \quad i = 1, \dots, K.$$

Show that $\kappa \leq \gamma$. Thus γ is an upper bound on the Lyapunov exponent κ . (The function V is called a quadratic *Lyapunov function* for the system.)

- (b) Explain how to use convex or quasiconvex optimization to find a matrix $P \in \mathbf{S}_{++}^n$ with the smallest value of γ , *i.e.*, with the best upper bound on κ . You must justify your formulation.
- (c) Carry out the method of part (b) for the specific problem with data given in `lyap_exp_bound_data.m`. Report the best upper bound on κ , to a tolerance of 0.01. The data $A^{(i)}$ are given as a cell array; `A{i}` gives $A^{(i)}$.
- (d) *Approximate worst-case trajectory simulation.* The quadratic Lyapunov function found in part (c) can be used to generate sequences of A_t that tend to result in large values of $\|x_t\|_2^{1/t}$. Start from a random vector x_0 . At each t , generate x_{t+1} by choosing $A_t = A^{(i)}$ that maximizes $V(A^{(i)}x_t)$, where P is computed from part (c). Do this for 50 time steps, and generate 5 such trajectories. Plot $\|x_t\|_2^{1/t}$ and γ against t to verify that the bound you obtained in the previous part is valid. Report the lower bound on the Lyapunov exponent that the trajectories suggest.

Solution.

- (a) Suppose V satisfies the conditions given above, and x_0, x_1, \dots is any trajectory of the system. Then, $V(x_{t+1}) \leq \gamma^2 V(x_t)$ for all $t \geq 0$. It follows that $V(x_t) \leq \gamma^{2t} V(x_0)$. So we have

$$\|x_t\|_2^2 = \frac{x_t^T x_t}{x_t^T P x_t} x_t^T P x_t \leq \sup_{x \neq 0} \frac{x^T x}{x^T P x} V(x_t) \leq \lambda_{\min}(P)^{-1} \gamma^{2t} V(x_0),$$

and thus,

$$\|x_t\|_2^{1/t} \leq \lambda_{\min}(P)^{-1/2t} \gamma V(x_0)^{1/2t}.$$

Taking the limit as $t \rightarrow \infty$ we get $\kappa \leq \gamma$.

- (b) The given condition on V is equivalent to the matrix inequalities

$$A^{(i)T} P A^{(i)} \preceq \gamma^2 P, \quad i = 1, \dots, K.$$

For any fixed γ , these are linear matrix inequalities in P , hence convex constraints.

Note that the LMIs above are homogeneous in P . Therefore, without loss of generality, we can require $P \succeq I$ in order to enforce P to be positive definite. Thus, there is a quadratic Lyapunov function that establishes the bound γ if and only if the LMIs

$$A^{(i)T} P A^{(i)} \preceq \gamma^2 P, \quad i = 1, \dots, K, \quad P \succeq I$$

are feasible. This defines a convex set of P , so finding the smallest possible value of γ is a quasiconvex problem. Then, we can use bisection on γ to solve this problem.

- (c) We find that the best bound obtained by the method above is $\gamma = 0.97$. The following code solves the problem.

```
% lyapunov analysis of a dynamical system
lyap_exp_bound_data;

% compute lower and upper bounds for bisection
l = 0; u = 0;
for i = 1:K
    l = max(l, max(abs(eig(A{i}))));
    u = max(u, norm(A{i}));
end
bisection_tol = 1e-4;

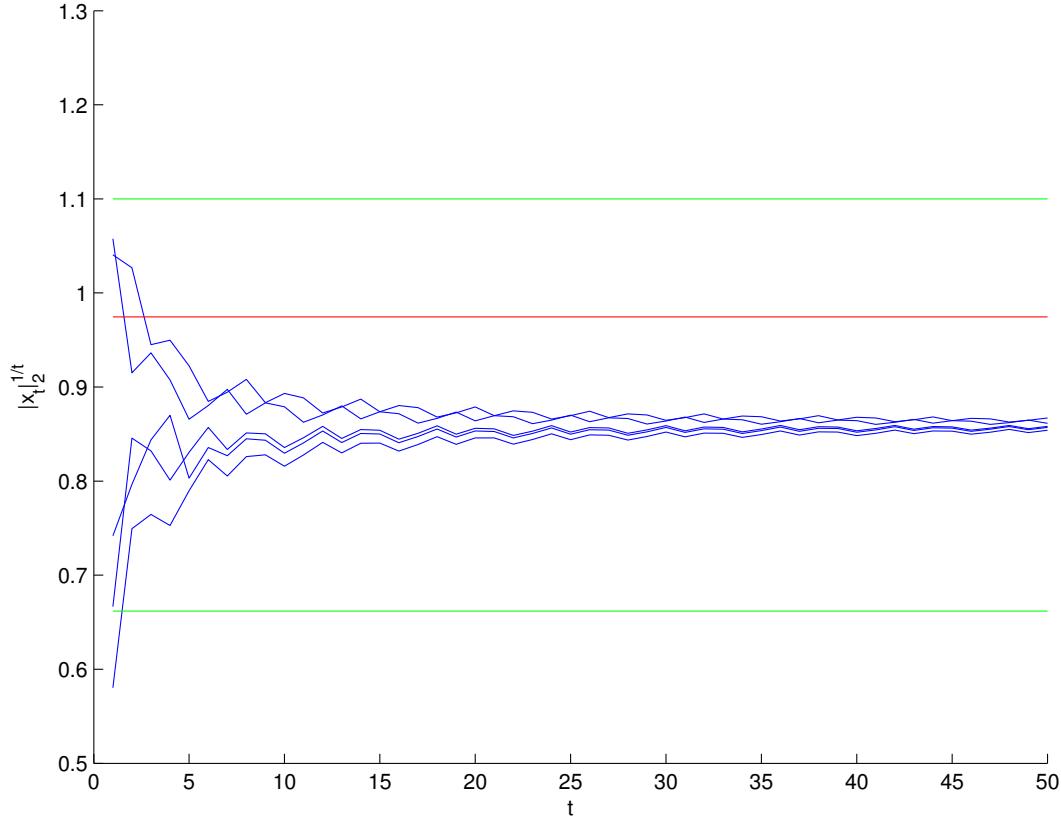
while u-l >= bisection_tol
    fprintf('bisection bounds: %f %f\n', l, u);
    gamma = (l+u)/2;
    cvx_begin quiet
        variable P(n, n) symmetric
        minimize trace(P)
        subject to
            for i = 1:K
                gamma^2*P - A{i}'*P*A{i} == semidefinite(n)
            end
            P-eye(n) == semidefinite(n)
    cvx_end

    if strcmp(cvx_status, 'Solved')
        u = gamma;
        bound = gamma;
        P_opt = P;
    else
        l = gamma;
    end
end

fprintf('smallest value of gamma = %f\n', bound);
```

We mention the initial lower and upper bound used in the code, that we didn't ask you to explore. Suppose that the initial state was an eigenvector of some $A^{(i)}$ with eigenvalue λ , and that this particular $A^{(i)}$ was chosen at every time step. Under this condition, it is easy to see that $|\lambda|$ gives a lower bound on γ . On the other hand, the ratio $\|x_{t+1}\|_2/\|x_t\|_2$ is bounded by $\max_i \|A^{(i)}\|$, so we obtain an upper bound on γ .

- (d) The figure shows five random trajectories in blue, and the γ bound in red. The initial lower and upper bound used in the bisection method are shown in green. The trajectories suggest that the Lyapunov exponent of the system is 0.86 or higher.



The following code generates the plot.

```
% approximate worst-case trajectory
% simulation of a dynamical system
clf; hold on;
T = 50;
for traj = 1:5
    x = randn(n, 1); % random initial state
    x = 1.1*x/norm(x);
    ys = [];
    for t = 1:T
        best = 0;
        for i = 1:K
            if best < x'*A{i}'*P_opt*A{i}*x

```

```

        best = x'*A{i}'*P_opt*A{i}*x;
        x_next = A{i}*x;
    end
end
x = x_next;
ys(end+1) = norm(x)^(1/t);
end
plot(1:T, ys);
end

l = 0; u = 0;
for i = 1:K
    l = max(l, max(abs(eig(A{i}))));
    u = max(u, norm(A{i}));
end

plot([1 T], [bound bound], 'r', ...
      [1 T], [l l], 'g', ...
      [1 T], [u u], 'g');

xlabel('t'); ylabel('|x_t|_2^{1/t}');
hold off;

print -depsc lyap_exp_bound.eps;

```

17.14 Optimal material blending. A standard industrial operation is to blend or mix raw materials (typically fluids such as different grades of crude oil) to create blended materials or products. This problem addresses optimizing the blending operation. We produce n blended materials from m raw materials. Each raw and blended material is characterized by a vector that gives the concentration of each of q constituents (such as different octane hydrocarbons). Let $c_1, \dots, c_m \in \mathbf{R}_+^q$ and $\tilde{c}_1, \dots, \tilde{c}_n \in \mathbf{R}_+^q$ be the concentration vectors of the raw materials and the blended materials, respectively. We have $\mathbf{1}^T c_j = \mathbf{1}^T \tilde{c}_i = 1$ for $i = 1, \dots, n$ and $j = 1, \dots, m$. The raw material concentrations are given; the blended product concentrations must lie between some given bounds, $\tilde{c}_i^{\min} \leq \tilde{c}_i \leq \tilde{c}_i^{\max}$.

Each blended material is created by pumping raw materials (continuously) into a vat or container where they are mixed to produce the blended material (which continuously flows out of the mixing vat). Let $f_{ij} \geq 0$ denote the flow of raw material j (say, in kg/s) into the vat for product i , for $i = 1, \dots, n$, $j = 1, \dots, m$. These flows are limited by the total availability of each raw material: $\sum_{i=1}^n f_{ij} \leq F_j$, $j = 1, \dots, m$, where $F_j > 0$ is the maximum total flow of raw material j available. Let $\tilde{f}_i \geq 0$ denote the flow rates of the blended materials. These also have limits: $\tilde{f}_i \leq \tilde{F}_i$, $i = 1, \dots, n$.

The raw and blended material flows are related by the (mass conservation) equations

$$\sum_{j=1}^m f_{ij} c_j = \tilde{f}_i \tilde{c}_i, \quad i = 1, \dots, n.$$

(The lefthand side is the vector of incoming constituent mass flows and the righthand side is the vector of outgoing constituent mass flows.)

Each raw and blended material has a (positive) price, p_j , $j = 1, \dots, m$ (for the raw materials), and \tilde{p}_i , $i = 1, \dots, n$ (for the blended materials). We pay for the raw materials, and get paid for the blended materials. The total profit for the blending process is

$$-\sum_{i=1}^n \sum_{j=1}^m f_{ij} p_j + \sum_{i=1}^n \tilde{f}_i \tilde{p}_i.$$

The goal is to choose the variables f_{ij} , \tilde{f}_i , and \tilde{c}_i so as to maximize the profit, subject to the constraints. The problem data are c_j , \tilde{c}_i^{\min} , \tilde{c}_i^{\max} , F_j , \tilde{F}_i , p_j , and \tilde{p}_j .

- (a) Explain how to solve this problem using convex or quasi-convex optimization. You must justify any change of variables or problem transformation, and explain how you recover the solution of the blending problem from the solution of your proposed problem.
- (b) Carry out the method of part (a) on the problem instance given in `material_blending_data.*`. Report the optimal profit, and the associated values of f_{ij} , \tilde{f}_i , and \tilde{c}_i .

Solution.

- (a) The problem we are to solve is

$$\begin{aligned} & \text{maximize} && -\sum_{i,j} f_{ij} p_j + \sum_i \tilde{f}_i \tilde{p}_i \\ & \text{subject to} && \sum_j f_{ij} c_j = \tilde{f}_i \tilde{c}_i \\ & && \mathbf{1}^T \tilde{c}_i = 1 \\ & && \tilde{c}_i^{\min} \leq \tilde{c}_i \leq \tilde{c}_i^{\max} \\ & && 0 \leq \tilde{f}_i \leq \tilde{F}_i \\ & && 0 \leq f_{ij} \\ & && \sum_i f_{ij} \leq F_j \end{aligned}$$

with variables f_{ij} , \tilde{f}_i , and \tilde{c}_i . Each constraint that is indexed by i must hold for $i = 1, \dots, n$, and each constraint is indexed by j must hold for $j = 1, \dots, m$.

The objective and all constraints except the first set of equality constraints are linear. On the right hand side of the first set of inequalities we have the product of two variables \tilde{f}_i and \tilde{c}_i , so these constraints are not convex.

To deal with this, we introduce new variables $m_i = \tilde{f}_i \tilde{c}_i \in \mathbf{R}^q$ for $i = 1, \dots, n$, and reformulate the problem as an optimization problem with decision variables f_{ij} , \tilde{f}_i , and m_i , removing the variables \tilde{c}_i . The vectors m_i are the blended product constituent mass flows.

The variables \tilde{c}_i only appear in the first three sets of constraints. In the first set of equality constraints, we can simply replace $\tilde{f}_i \tilde{c}_i$ with m_i , which results in a set of linear equality constraints. We express $\mathbf{1}^T \tilde{c}_i = 1$ as $\mathbf{1}^T m_i = \tilde{f}_i$; these are equivalent since $\mathbf{1}^T m_i = \tilde{f}_i \mathbf{1}^T \tilde{c}_i = \tilde{f}_i$. The third set of constraints is equivalent to $\tilde{f}_i \tilde{c}_i^{\min} \leq m_i \leq \tilde{f}_i \tilde{c}_i^{\max}$. Therefore, the problem

becomes

$$\begin{aligned}
 & \text{maximize} && -\sum_{i,j} f_{ij} p_j + \sum_i \tilde{f}_i \tilde{p}_i \\
 & \text{subject to} && \sum_j f_{ij} c_j = m_i, \quad i = 1, \dots, n \\
 & && \mathbf{1}^T m_i = \tilde{f}_i, \quad i = 1, \dots, n \\
 & && \tilde{f}_i c_i^{\min} \leq m_i \leq \tilde{f}_i c_i^{\max}, \quad i = 1, \dots, n \\
 & && 0 \leq f_{ij}, \quad i = 1, \dots, n \quad j = 1, \dots, m \\
 & && 0 \leq \tilde{f}_i \leq \tilde{F}_i, \quad i = 1, \dots, n \\
 & && \sum_i f_{ij} \leq F_j, \quad j = 1, \dots, m,
 \end{aligned}$$

with variables f_{ij} , \tilde{f}_i , and m_i . This is an LP. In order to reconstruct the solution to the original problem, we find \tilde{c}_i by $\tilde{c}_i = m_i / \tilde{f}_i$.

- (b) The following MATLAB code solves the problem.

```

clear all
material_blending_data

cvx_begin
variables f(2,4) ftild(2) m(3,2)
maximize -sum(f*p)+ sum(m*pTilde)
subject to
    m == C*f'
    sum(m) == ftild'
    0 <= f
    0 <= ftild <= FTilde
    c_minTilde*diag(ftild) <= m <= c_maxTilde*diag(ftild)
    sum(f)' <= F
cvx_end

```

The following Python code solves the problem.

```

from cvxpy import *
from material_blending_data import *

f = Variable(2,4)
ftild = Variable(2)
m = Variable(3,2)

objective = Maximize(-sum_entries(f * p)+ sum_entries(m * pTilde))
constraints = [m == C*f.T,
               np.ones([1,3])*m == ftild.T,
               0 <= f,
               0 <= ftild,
               ftild <= FTilde,
               c_minTilde * diag(ftild) <= m,
               m <= c_maxTilde * diag(ftild),
               (np.ones([1,2]) * f).T <= F]

```

```

prob = Problem(objective, constraints)

result = prob.solve()
print prob.value

```

The following Julia code solves the problem.

```

using Convex, ECOS
include("material_blending_data.jl");

f = Variable(2,4)
ftilde = Variable(2)
m = Variable(3,2)

obj = (-sum(f * p) + sum(m * pTilde))
prob = maximize(obj)
prob.constraints += [m == C*f',
                     ones(1,3)*m == ftildes',
                     0 <= f,
                     0 <= ftildes,
                     ftildes <= FTilde,
                     c_minTilde * diagm(ftildes) <= m,
                     m <= c_maxTilde * diagm(ftildes),
                     f'*ones(2,1) <= F]

solve!(prob, ECOSolver(verbose=0,max_iters=20000))

println(prob.optval)

```

We find that the optimal value is 127, with a solution

$$f = \begin{bmatrix} 6.0555 & 0.9167 & 0.7065 & 0.3213 \\ 0.9445 & 1.0833 & 5.2935 & 2.6787 \end{bmatrix},$$

$$\tilde{f} = \begin{bmatrix} 8 \\ 10 \end{bmatrix}, \quad \tilde{c}_1 = \begin{bmatrix} 0.8588 \\ 0.1000 \\ 0.0412 \end{bmatrix}, \quad \tilde{c}_2 = \begin{bmatrix} 0.7029 \\ 0.1800 \\ 0.1171 \end{bmatrix}.$$

- 17.15 Optimal evacuation planning.** We consider the problem of evacuating people from a dangerous area in a way that minimizes risk exposure. We model the area as a connected graph with n nodes and m edges; people can assemble or collect at the nodes, and travel between nodes (in either direction) over the edges. We let $q_t \in \mathbf{R}_+^n$ denote the vector of the numbers of people at the nodes, in time period t , for $t = 1, \dots, T$, where T is the number of periods we consider. (We will consider the entries of q_t as real numbers, not integers.) The initial population distribution q_1 is given. The nodes have capacity constraints, given by $q_t \leq Q$, where $Q \in \mathbf{R}_+^n$ is the vector of node capacities.

We use the incidence matrix $A \in \mathbf{R}^{n \times m}$ to describe the graph. We assign an arbitrary reference direction to each edge, and take

$$A_{ij} = \begin{cases} +1 & \text{if edge } j \text{ enters node } i \\ -1 & \text{if edge } j \text{ exits node } i \\ 0 & \text{otherwise.} \end{cases}$$

The population dynamics are given by $q_{t+1} = A f_t + q_t$, $t = 1, \dots, T-1$ where $f_t \in \mathbf{R}^m$ is the vector of population movement (flow) across the edges, for $t = 1, \dots, T-1$. A positive flow denotes movement in the direction of the edge; negative flow denotes population flow in the reverse direction. Each edge has a capacity, *i.e.*, $|f_t| \leq F$, where $F \in \mathbf{R}_+^m$ is the vector of edge capacities, and $|f_t|$ denotes the elementwise absolute value of f_t .

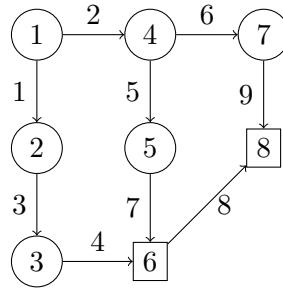
An *evacuation plan* is a sequence q_1, q_2, \dots, q_T and f_1, f_2, \dots, f_{T-1} obeying the constraints above. The goal is to find an evacuation plan that minimizes the total risk exposure, defined as

$$R_{\text{tot}} = \sum_{t=1}^T (r^T q_t + s^T q_t^2) + \sum_{t=1}^{T-1} (\tilde{r}^T |f_t| + \tilde{s}^T f_t^2),$$

where $r, s \in \mathbf{R}_+^n$ are given vectors of risk exposure coefficients associated with the nodes, and $\tilde{r}, \tilde{s} \in \mathbf{R}_+^m$ are given vectors of risk exposure coefficients associated with the edges. The notation q_t^2 and f_t^2 refers to elementwise squares of the vectors. Roughly speaking, the risk exposure is a quadratic function of the occupancy of a node, or the (absolute value of the) flow of people along an edge. The linear terms can be interpreted as the risk exposure per person; the quadratic terms can be interpreted as the additional risk associated with crowding.

A subset of nodes have zero risk ($r_i = s_i = 0$), and are designated as *safe nodes*. The population is considered *evacuated* at time t if $r^T q_t + s^T q_t^2 = 0$. The *evacuation time* t_{evac} of an evacuation plan is the smallest such t . We will assume that T is sufficiently large and that the total capacity of the safe nodes exceeds the total initial population, so evacuation is possible.

Use CVX* to find an optimal evacuation plan for the problem instance with data given in `opt_evac_data.*`. (We display the graph below, with safe nodes denoted as squares.)



Report the associated optimal risk exposure R_{tot}^* . Plot the time period risk

$$R_t = r^T q_t + s^T q_t^2 + \tilde{r}^T |f_t| + \tilde{s}^T f_t^2$$

versus time. (For $t = T$, you can take the edge risk to be zero.) Plot the node occupancies q_t , and edge flows f_t versus time. Briefly comment on the results you see. Give the evacuation time t_{evac} (considering any $r^T q_t + s^T q_t^2 \leq 10^{-4}$ to be zero).

Hint. With CVXPY, use the ECOS solver with `p.solve(solver=cvxpy.ECOS)`.

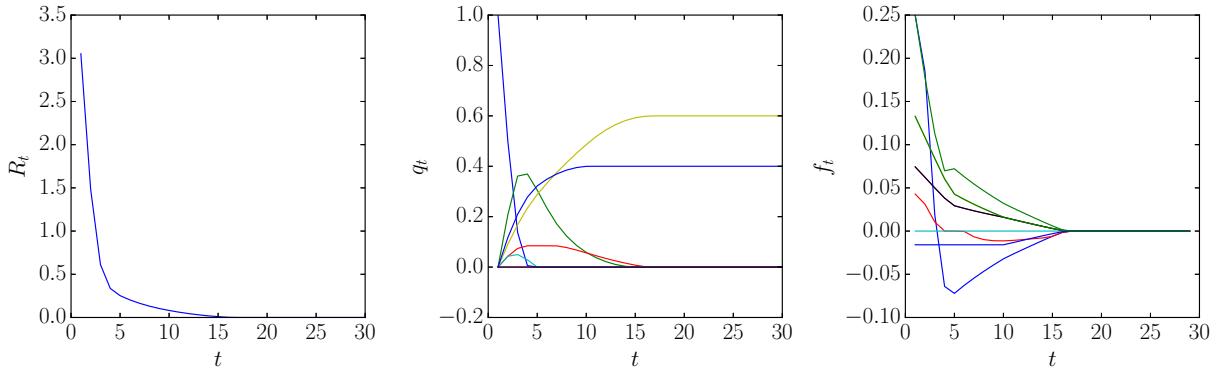
Solution. The optimization problem is given by

$$\begin{aligned} \text{minimize} \quad & \sum_{t=1}^T \left(r^T q_t + s^T q_t^2 \right) + \sum_{t=1}^{T-1} \left(\tilde{r}^T |f_t| + \tilde{s}^T f_t^2 \right) \\ \text{subject to} \quad & q_{t+1} = A f_t + q_t, \quad t = 1, \dots, T-1 \\ & 0 \preceq q_t \preceq Q, \quad t = 2, \dots, T \\ & |f_t| \leq F, \quad t = 1, \dots, T-1, \end{aligned}$$

with variables q_2, \dots, q_T and f_1, \dots, f_{T-1} . This is evidently a convex optimization problem, since the objective is convex and the constraints are all linear.

The optimal evacuation time is $t_{\text{evac}} = 17$, with total risk exposure $R_{\text{tot}}^* = 6.59$.

Note that two of the edge flows reverse direction during the evacuation. This is because the entire population starts at node 1, but not everyone can move to the safe nodes immediately, due to the edge capacity constraints. To avoid accumulating risk, some people move to the safer nodes 2 and 3. Once the bottleneck clears, people flow back in the reverse direction, past node 1, and towards the safe nodes.



The following Python code solves the problem.

```
# solution to optimal evacuation problem
import numpy as np
import cvxpy as cvx
import matplotlib.pyplot as plt
import matplotlib

from opt_evac_data import *

def opt_evac(A, Q, F, q1, r, s, rtild, stild, T):
    n,m = A.shape
    q = cvx.Variable(n,T)
    f = cvx.Variable(m,T-1)
```

```

node_risk = q.T*r + cvx.square(q).T*s
edge_risk = cvx.vstack(cvx.abs(f).T*rtild + cvx.square(f).T*stild,0)
risk = node_risk + edge_risk

constr = [q[:,0] == q1,
          q[:,1:] == A*f + q[:,:-1],
          0 <= q, q <= np.tile(Q,(T,1)).T,
          cvx.abs(f) <= np.tile(F,(T-1,1)).T]

p = cvx.Problem(cvx.Minimize(sum(risk)), constr)
p.solve(verbose=True, solver=cvx.ECOS)

arr = lambda _: np.array(_.value)
q, f, risk, node_risk = map(arr, (q, f, risk, node_risk))

print "Total risk: ", p.value
print "Evacuated at t =", (node_risk <= 1e-4).nonzero()[0][0] + 1

return q, f, risk, node_risk

# solve
q, f, risk, node_risk = opt_evac(A, Q, F, q1, r, s, rtild, stild, T)

# plot
plt.rc('text', usetex=True)
plt.rcParams.update({'font.size': 20})
fig, axs = plt.subplots(1,3,figsize=(15,5))

axs[0].plot(np.arange(1,T+1), risk)
axs[0].set_ylabel('$R_t$')

axs[1].plot(np.arange(1,T+1), q.T)
axs[1].set_ylabel('$q_t$')

axs[2].plot(np.arange(1,T), f.T)
axs[2].set_ylabel('$f_t$')

for ax in axs:
    ax.set_xlabel('$t$')

if matplotlib.get_backend().lower() in ['agg', 'macosx']:
    fig.set_tight_layout(True)
else:
    fig.tight_layout()
#plt.tight_layout()

```

```

fig.savefig('opt_evac.pdf')
fig.savefig('opt_evac.eps')

```

The following MATLAB code solves the problem.

```

% solution to optimal evacuation problem
opt_evac_data

[n, m] = size(A);

cvx_begin
    variable q(n,T)
    variable f(m,T-1)
    risk = q'*r + square(q)*s + [abs(f)']*rtild + square(f)*stild; 0]
    minimize( sum(risk) )
    subject to
        q(:,2:end) == A*f + q(:,1:end-1)
        q(:,1) == q1
        0 <= q
        q <= repmat(Q,1,T)
        abs(f) <= repmat(F,1,T-1)
cvx_end

fprintf('Total risk: %f\n', sum(risk))
fprintf('Evacuated at t = %d\n', find(q'*r + (q.^2)*s < 1e-4,1))

subplot(1,3,1)
plot(risk)
ylabel('R_t')
xlabel('t')
subplot(1,3,2)
plot(q')
ylabel('q_t')
xlabel('t')
subplot(1,3,3)
plot(f')
ylabel('f_t')
xlabel('t')
print(gcf,'-deps','opt_evac.eps')

```

The following Julia code solves the problem.

```

# solution to optimal evacuation problem
using Convex, ECOS, PyPlot
include("opt_evac_data.jl");

```

```

n,m = size(A)
q = Variable(n,T)
f = Variable(m,T-1)
risk = q'*r + square(q)'*s + [abs(f)'*rtild + square(f)'*stild,0]
p = minimize(sum(risk))
p.constraints += [q[:,1] == q1,
                  q[:,2:end] == A*f + q[:,1:end-1],
                  0 <= q, q <= repmat(Q,1,T),
                  abs(f) <= repmat(F,1,T-1)]
solve!(p, ECOSolver(verbose=1))

risk = evaluate(risk)
q = q.value
f = f.value
println("Total risk: $(round(sum(risk),2))")
println("Evacuated at t = $(findfirst(q'*r + (q.*q)'*s .<= 1e-4))")

fig = figure("stuff",figsize=(22,5))
subplot(131)
plot(risk)
ylabel(L"R_t")
xlabel(L"t")
subplot(132)
plot(q')
ylabel(L"q_t")
xlabel(L"t")
subplot(133)
plot(f')
ylabel(L"f_t")
xlabel(L"t")
savefig("opt_evac.eps")

```

17.16 *Ideal preference point.* A set of K choices for a decision maker is parametrized by a set of vectors $c^{(1)}, \dots, c^{(K)} \in \mathbf{R}^n$. We will assume that the entries c_i of each choice are normalized to lie in the range $[0, 1]$. The *ideal preference point model* posits that there is an ideal choice vector c^{ideal} with entries in the range $[0, 1]$; when the decision maker is asked to choose between two candidate choices c and \tilde{c} , she will choose the one that is closest (in Euclidean norm) to her ideal point. Now suppose that the decision maker has chosen between all $K(K - 1)/2$ pairs of given choices $c^{(1)}, \dots, c^{(K)}$. The decisions are represented by a list of pairs of integers, where the pair (i, j) means that $c^{(i)}$ is chosen when given the choices $c^{(i)}, c^{(j)}$. You are given these vectors and the associated choices.

- (a) How would you determine if the decision maker's choices are consistent with the ideal preference point model?
- (b) Assuming they are consistent, how would you determine the bounding box of ideal choice vectors consistent with her decisions? (That is, how would you find the minimum and maximum

values of c_i^{ideal} , for c^{ideal} consistent with being the ideal preference point.)

- (c) Carry out the method of part (b) using the data given in `ideal_pref_point_data.*`. These files give the points $c^{(1)}, \dots, c^{(K)}$ and the choices, and include the code for plotting the results. Report the width and the height of the bounding box and include your plot.

Solution.

- (a) The decision that c^{ideal} is closer to $c^{(i)}$ than $c^{(j)}$ means that c^{ideal} lies in the half-space

$$\left\{ x \in \mathbf{R}^n \mid (c^{(j)} - c^{(i)})^T x \leq \frac{1}{2}(c^{(i)} + c^{(j)})^T (c^{(j)} - c^{(i)}) \right\}.$$

Thus, the decision maker's choices are consistent with an ideal preference point model if and only if the polyhedron obtained by intersecting the hypercube $[0, 1]^n$ and those half-spaces for all decisions (i, j) is nonempty.

Remark: It is insufficient to only check whether the decisions satisfy transitivity (i.e., if (i, j) and (j, k) are decisions, then so is (i, k)). Consider $c^{(i)} = [i] \in \mathbf{R}^1$, $i = 1, 2, 3$, then the decisions $(1, 2), (1, 3), (3, 2)$ satisfy transitivity, though there is no c^{ideal} satisfying the constraints.

- (b) The minimum/maximum value of c_k^{ideal} can be obtained by minimizing/maximizing c_k^{ideal} subject to the constraint that c^{ideal} lies in the aforementioned polyhedron. In other words, for each $k = 1, \dots, n$, to find the minimum value of c_k^{ideal} , we solve the problem

$$\begin{aligned} & \text{minimize} && c_k^{\text{ideal}} \\ & \text{subject to} && 0 \preceq c^{\text{ideal}} \preceq \mathbf{1}, \\ & && (c^{(j)} - c^{(i)})^T c^{\text{ideal}} \leq \frac{1}{2}(c^{(i)} + c^{(j)})^T (c^{(j)} - c^{(i)}) \text{ for decisions } (i, j). \end{aligned}$$

The maximum value is obtained by using `maximize` instead of `minimize`.

Remark : It is possible to reduce the number of decisions needed to be considered in the convex problem from $K(K-1)/2$ to $K-1$, as long as the decisions satisfy transitivity (which should be the case if there is no tie). Assume the candidate choices are ordered as $c^{(k_1)}, \dots, c^{(k_K)}$ from closest to farthest from the ideal point (the ordering can be obtained by counting the number of times $c^{(i)}$ is preferred in the decisions (i, j)), then we need only to consider the decisions $(k_1, k_2), \dots, (k_{K-1}, k_K)$.

- (c) The width and the height of the bounding box are 0.140 and 0.098 respectively.

The following Matlab code solves the problem:

```
% Problem data
K = 8;
n = 2;

% List of candidate choices as row vectors
c = [0.314 0.509; 0.185 0.282; 0.670 0.722; 0.116 0.253; ...
      0.781 0.382; 0.519 0.952; 0.953 0.729; 0.406 0.110];

% List of decisions. Row [i j] means c(i) preferred over c(j)
d = [1 2; 3 1; 3 2; 1 4; 2 4; 3 4; 5 1; ...
      5 2; 3 5; 5 4; 1 6; 6 2; 3 6; 6 4; ...]
```

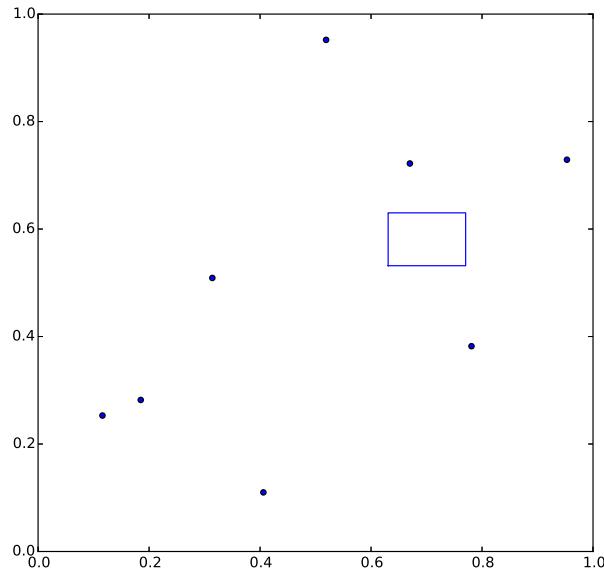


Figure 19: Plot of the points $c^{(i)}$ and the bounding box.

```

5 6; 7 1; 7 2; 3 7; 7 4; 5 7; 7 6; ...
1 8; 8 2; 3 8; 8 4; 5 8; 6 8; 7 8];

box = zeros(n, 2);

% Put your code for finding the bounding box here.
% box(i, 1) and box(i, 2) should be the lower and upper bounds
% of the i-th coordinate respectively.
for a = 1:n
    for s = [-1, 1]
        cvx_begin
            variables c_ideal(n)
            maximize c_ideal(a) * s
            subject to
                0 <= c_ideal;
                c_ideal <= 1;
                for b = 1:size(d,1)
                    c_ideal' * (c(d(b,2),:) - c(d(b,1),:))' <= (c(d(b,1),:)
                        + c(d(b,2),*)) * (c(d(b,2),:) - c(d(b,1),:))' / 2;
                end
        cvx_end
        box(a, (s + 3) / 2) = cvx_optval * s;
    end
end

```

```

end

% Drawing the bounding box
figure;
scatter(c(:,1), c(:,2));
hold on
plot([box(1,1);box(1,2);box(1,2);box(1,1);box(1,1)], ...
      [box(2,1);box(2,1);box(2,2);box(2,2);box(2,1)]);
hold off
xlim([0 1]);
ylim([0 1]);
disp(['Width of bounding box = ' num2str(box(1,2) - box(1,1))])
disp(['Height of bounding box = ' num2str(box(2,2) - box(2,1))])

```

The following Python code solves the problem:

```

from cvxpy import *
import numpy as np
import matplotlib.pyplot as plt

# Problem data
K = 8
n = 2

# List of candidate choices
c = [[0.314, 0.509], [0.185, 0.282], [0.670, 0.722], [0.116, 0.253],
      [0.781, 0.382], [0.519, 0.952], [0.953, 0.729], [0.406, 0.110]]
c = [np.array(x) for x in c]

# List of decisions. [i, j] means c[i] preferred over c[j]
d = [[0, 1], [2, 0], [2, 1], [0, 3], [1, 3], [2, 3], [4, 0],
      [4, 1], [2, 4], [4, 3], [0, 5], [5, 1], [2, 5], [5, 3],
      [4, 5], [6, 0], [6, 1], [2, 6], [6, 3], [4, 6], [6, 5],
      [0, 7], [7, 1], [2, 7], [7, 3], [4, 7], [5, 7], [6, 7]]

box = [[0] * 2 for a in range(n)]

# Put your code for finding the bounding box here.
# box[i][0] and box[i][1] should be the lower and upper bounds
# of the i-th coordinate respectively.
for a in range(n):
    for s in [-1, 1]:
        c_ideal = Variable(n)
        objective = Maximize(c_ideal[a] * s)
        constraints = [0 <= c_ideal, c_ideal <= 1]
        for b in d:
            if a < b:
                constraints.append(c_ideal[b] - c_ideal[a] >= 0)
            else:
                constraints.append(c_ideal[a] - c_ideal[b] >= 0)
        prob = Problem(objective, constraints)
        prob.solve()
        box[a][0] = max(box[a][0], c_ideal.value)
        box[a][1] = min(box[a][1], c_ideal.value)

```

```

        constraints.append(c_ideal.T * (c[b[1]] - c[b[0]]))
        <= np.dot(c[b[0]] + c[b[1]], c[b[1]] - c[b[0]]) / 2)
prob = Problem(objective, constraints)
box[a][(s + 1) // 2] = prob.solve() * s

# Drawing the bounding box
plt.figure()
plt.xlim(0, 1)
plt.ylim(0, 1)
plt.scatter([c[i][0] for i in range(K)], [c[i][1] for i in range(K)])
plt.plot([box[0][0], box[0][1], box[0][1], box[0][0], box[0][0]],
         [box[1][0], box[1][0], box[1][1], box[1][1], box[1][0]])
plt.gca().set_aspect('equal', adjustable='box')
plt.show()
print('Width of bounding box = ' + str(box[0][1] - box[0][0]))
print('Height of bounding box = ' + str(box[1][1] - box[1][0]))

```

The following Julia code solves the problem:

```

using Convex
using PyPlot

# Problem data
K = 8
n = 2

# List of candidate choices as row vectors
c = [0.314 0.509; 0.185 0.282; 0.670 0.722; 0.116 0.253;
      0.781 0.382; 0.519 0.952; 0.953 0.729; 0.406 0.110]

# List of decisions. Row [i j] means c[i] preferred over c[j]
d = [1 2; 3 1; 3 2; 1 4; 2 4; 3 4; 5 1;
      5 2; 3 5; 5 4; 1 6; 6 2; 3 6; 6 4;
      5 6; 7 1; 7 2; 3 7; 7 4; 5 7; 7 6;
      1 8; 8 2; 3 8; 8 4; 5 8; 6 8; 7 8]

box = zeros(n, 2)

# Put your code for finding the bounding box here.
# box[i, 1] and box[i, 2] should be the lower and upper bounds
# of the i-th coordinate respectively.
for a in 1:n
    for s in [-1, 1]
        c_ideal = Variable(n)
        constraints = [0 <= c_ideal; c_ideal <= 1]
        for b = 1:size(d,1)

```

```

        push!(constraints, c_ideal' * (c[d[b,2],:] - c[d[b,1],:])'
              <= (c[d[b,1],:] + c[d[b,2],:]) * (c[d[b,2],:] - c[d[b,1],:])' / 2)
    end
    prob = maximize(c_ideal[a] * s, constraints)
    solve!(prob)
    box[a, (s + 3) / 2] = prob.optval * s
end
end

# Drawing the bounding box
figure()
xlim(0, 1)
ylim(0, 1)
scatter(c[:,1], c[:,2])
plot([box[1,1];box[1,2];box[1,2];box[1,1];box[1,1],
      [box[2,1];box[2,1];box[2,2];box[2,2];box[2,1]])
println("Width of bounding box = $(box[1,2] - box[1,1])")
println("Height of bounding box = $(box[2,2] - box[2,1])")

```

17.17 Matrix equilibration. We say that a matrix is ℓ_p *equilibrated* if each of its rows has the same ℓ_p norm, and each of its columns has the same ℓ_p norm. (The row and column ℓ_p norms are related by m , n , and p .) Suppose we are given a matrix $A \in \mathbf{R}^{m \times n}$. We seek diagonal invertible matrices $D \in \mathbf{R}^{m \times m}$ and $E \in \mathbf{R}^{n \times n}$ for which DAE is ℓ_p equilibrated.

- Explain how to find D and E using convex optimization. (Some matrices cannot be equilibrated. But you can assume that all entries of A are nonzero, which is enough to guarantee that it can be equilibrated.)
- Equilibrate the matrix A given in the file `matrix_equilibration_data.*`, with

$$m = 20, \quad n = 10, \quad p = 2.$$

Print the row ℓ_p norms and the column ℓ_p norms of the equilibrated matrix as vectors to check that each matches.

Hints.

- Work with the matrix B , with $B_{ij} = |A_{ij}|^p$.
- Consider the problem of minimizing $\sum_{i=1}^m \sum_{j=1}^n B_{ij} e^{u_i + v_j}$ subject to $\mathbf{1}^T u = 0$, $\mathbf{1}^T v = 0$. (Several variations on this idea will work.)
- We have found that expressing the terms in the objective as $e^{\log B_{ij} + u_i + v_j}$ leads to fewer numerical problems.

Solution. Following the hint, we find the optimality conditions for the suggested problem. The Lagrangian is

$$L(u, v, \nu, \omega) = \sum_{i=1}^m \sum_{j=1}^n B_{ij} e^{u_i + v_j} + \nu \mathbf{1}^T u + \omega \mathbf{1}^T v,$$

with dual variables ν and ω . The optimality conditions are

$$\frac{\partial L}{\partial u_i} = \sum_{j=1}^n B_{ij} e^{u_i + v_j} + \nu = 0, \quad i = 1, \dots, m,$$

and

$$\frac{\partial L}{\partial v_j} = \sum_{i=1}^m B_{ij} e^{u_i + v_j} + \omega = 0, \quad j = 1, \dots, n,$$

along with $\mathbf{1}^T u = \mathbf{1}^T v = 0$. Defining $D = \text{diag}(e^{u/p})$ and $E = \text{diag}(e^{v/p})$, where the exponentials are elementwise, we can write the optimality conditions as

$$\mathbf{1}^T D^p B E^p = -\nu \mathbf{1}^T, \quad D^p B E^p \mathbf{1} = -\omega \mathbf{1},$$

i.e., $D^p B E^p$ has all column sums equal to $-\nu$, and all row sums equal to $-\omega$. Therefore, the matrix DAE is ℓ_p equilibrated, since $|(DAE)_{ij}^p| = (D^p B E^p)_{ij}$.

For the given matrix A , we find the equilibrated matrix has row norms and column norms as follows.

Code solutions for each language follow.

Matlab:

```
matrix_equilibration_data;
B = abs(A).^p;

cvx_begin
    variables u(m) v(n);
    expression obj;
    for i = 1:m
        for j = 1:n
            obj = obj + exp(log(B(i,j))+u(i)+v(j));
        end
    end
    minimize(obj);
    subject to
        sum(u) == 0;
        sum(v) == 0;
cvx_end

D = diag(exp(u./p));
E = diag(exp(v./p));
A_eq= D*A*E;

row_norms = norms(A_eq,p,2)
col_norms = norms(A_eq,p,1)',
```

Python:

```

from cvxpy import *
import numpy as np

from matrix_equilibration_data import *
B = np.power(np.abs(A), p)

u = Variable(m)
v = Variable(n)

obj = 0
for i in range(m):
    for j in range(n):
        obj += exp(log(B[i, j]) + u[i] + v[j])
obj = Minimize(obj)

constraints = [sum(u) == 0, sum(v) == 0]

prob = Problem(obj, constraints)
prob.solve(verbose=True)

D = np.diagflat(np.exp(u.value / p))
E = np.diagflat(np.exp(v.value / p))
A_eq = D * A * E

row_norms = np.linalg.norm(A_eq, p, 1)
col_norms = np.linalg.norm(A_eq.T, p, 1)

print(row_norms)
print(col_norms)

```

Julia:

```

# Compute the equilibration
include("matrix_equilibration_data.jl");
B = abs(A).^p;

using Convex;
u = Variable(m); v = Variable(n);

objective = 0;
for i = 1:m
    for j = 1:n
        objective += exp(log(B[i,j])+u[i]+v[j]);
    end
end
constraints = [sum(u) == 0, sum(v) == 0];

```

```
problem = minimize(objective,constraints);
solve!(problem);

D = diagm(exp(u.value[:,1]./p));
E = diagm(exp(v.value[:,1]./p));
A_eq = D*A*E;

row_norms = sum(abs(A_eq).^p,2).^(1/p)
col_norms = sum(abs(A_eq').^p,2).^(1/p)
```