

### EE364a Homework 6 solutions

4.60 *Log-optimal investment strategy.* We consider a portfolio problem with  $n$  assets held over  $N$  periods. At the beginning of each period, we re-invest our total wealth, redistributing it over the  $n$  assets using a fixed, constant, allocation strategy  $x \in \mathbf{R}^n$ , where  $x \succeq 0$ ,  $\mathbf{1}^T x = 1$ . In other words, if  $W(t-1)$  is our wealth at the beginning of period  $t$ , then during period  $t$  we invest  $x_i W(t-1)$  in asset  $i$ . We denote by  $\lambda(t)$  the total return during period  $t$ , *i.e.*,  $\lambda(t) = W(t)/W(t-1)$ . At the end of the  $N$  periods our wealth has been multiplied by the factor  $\prod_{t=1}^N \lambda(t)$ . We call

$$\frac{1}{N} \sum_{t=1}^N \log \lambda(t)$$

the *growth rate* of the investment over the  $N$  periods. We are interested in determining an allocation strategy  $x$  that maximizes growth of our total wealth for large  $N$ .

We use a discrete stochastic model to account for the uncertainty in the returns. We assume that during each period there are  $m$  possible scenarios, with probabilities  $\pi_j$ ,  $j = 1, \dots, m$ . In scenario  $j$ , the return for asset  $i$  over one period is given by  $p_{ij}$ . Therefore, the return  $\lambda(t)$  of our portfolio during period  $t$  is a random variable, with  $m$  possible values  $p_1^T x, \dots, p_m^T x$ , and distribution

$$\pi_j = \mathbf{prob}(\lambda(t) = p_j^T x), \quad j = 1, \dots, m.$$

We assume the same scenarios for each period, with (identical) independent distributions. Using the law of large numbers, we have

$$\lim_{N \rightarrow \infty} \frac{1}{N} \log \left( \frac{W(N)}{W(0)} \right) = \lim_{N \rightarrow \infty} \frac{1}{N} \sum_{t=1}^N \log \lambda(t) = \mathbf{E} \log \lambda(t) = \sum_{j=1}^m \pi_j \log(p_j^T x).$$

In other words, with investment strategy  $x$ , the long term growth rate is given by

$$R_{\text{lt}} = \sum_{j=1}^m \pi_j \log(p_j^T x).$$

The investment strategy  $x$  that maximizes this quantity is called the *log-optimal investment strategy*, and can be found by solving the optimization problem

$$\begin{aligned} & \text{maximize} && \sum_{j=1}^m \pi_j \log(p_j^T x) \\ & \text{subject to} && x \succeq 0, \quad \mathbf{1}^T x = 1, \end{aligned}$$

with variable  $x \in \mathbf{R}^n$ .

Show that this is a convex optimization problem.

**Solution.** Actually, there's not much to do in this problem. The constraints,  $x \succeq 0$ ,  $\mathbf{1}^T x = 1$ , are clearly convex, so we just need to show that the objective is concave (since it is to be maximized). We can do that in just a few steps: First, note that log is concave, so  $\log(p_j^T x)$  is concave in  $x$  (on the domain, which is the open halfspace  $\{x \mid p_j^T x > 0\}$ ). Since  $\pi_j \geq 0$ , we conclude that the sum of concave functions

$$\sum_{j=1}^m \pi_j \log(p_j^T x)$$

is concave.

4.61 *Optimization with logistic model.* A random variable  $X \in \{0, 1\}$  satisfies

$$\mathbf{prob}(X = 1) = p = \frac{\exp(a^T x + b)}{1 + \exp(a^T x + b)},$$

where  $x \in \mathbf{R}^n$  is a vector of variables that affect the probability, and  $a$  and  $b$  are known parameters. We can think of  $X = 1$  as the event that a consumer buys a product, and  $x$  as a vector of variables that affect the probability, *e.g.*, advertising effort, retail price, discounted price, packaging expense, and other factors. The variable  $x$ , which we are to optimize over, is subject to a set of linear constraints,  $Fx \preceq g$ .

Formulate the following problems as convex optimization problems.

- (a) *Maximizing buying probability.* The goal is to choose  $x$  to maximize  $p$ .
- (b) *Maximizing expected profit.* Let  $c^T x + d$  be the profit derived from selling the product, which we assume is positive for all feasible  $x$ . The goal is to maximize the expected profit, which is  $p(c^T x + d)$ .

**Solution.**

- (a) The function  $e^u/(1 + e^u)$  is monotonically increasing in  $u$ , so we can maximize  $\exp(a^T x + b)/(1 + \exp(a^T x + b))$  by maximizing  $a^T x + b$ , which leads to the LP

$$\begin{aligned} & \text{maximize} && a^T x + b \\ & \text{subject to} && Fx \preceq g. \end{aligned}$$

- (b) Here we have to maximize  $p(c^T x + d)$ , or equivalently, its logarithm:

$$\begin{aligned} & \text{maximize} && a^T x + b - \log(1 + \exp(a^T x + b)) + \log(c^T x + d) \\ & \text{subject to} && Fx \preceq g. \end{aligned}$$

This is a convex problem, since the objective is a concave function of  $x$ . (Recall that  $f(x) = \log \sum_{i=1}^m \exp(a_i^T x + b_i)$  is convex.)

5.38 *Option pricing.* We apply the results of example 5.10, page 263, to a simple problem with three assets: a riskless asset with fixed return  $r > 1$  over the investment period of interest (for example, a bond), a stock, and an option on the stock. The option gives us the right to purchase the stock at the end of the period, for a predetermined price  $K$ .

We consider two scenarios. In the first scenario, the price of the stock goes up from  $S$  at the beginning of the period, to  $Su$  at the end of the period, where  $u > r$ . In this scenario, we exercise the option only if  $Su > K$ , in which case we make a profit of  $Su - K$ . Otherwise, we do not exercise the option, and make zero profit. The value of the option at the end of the period, in the first scenario, is therefore  $\max\{0, Su - K\}$ .

In the second scenario, the price of the stock goes down from  $S$  to  $Sd$ , where  $d < 1$ . The value at the end of the period is  $\max\{0, Sd - K\}$ .

In the notation of example 5.10,

$$V = \begin{bmatrix} r & uS & \max\{0, Su - K\} \\ r & dS & \max\{0, Sd - K\} \end{bmatrix}, \quad p_1 = 1, \quad p_2 = S, \quad p_3 = C,$$

where  $C$  is the price of the option.

Show that for given  $r, S, K, u, d$ , the option price  $C$  is uniquely determined by the no-arbitrage condition. In other words, the market for the option is complete.

**Solution.** The condition  $V^T y = p$  reduces to

$$y_1 + y_2 = 1/r, \quad uy_1 + dy_2 = 1, \quad y_1 \max\{0, Su - K\} + y_2 \max\{0, Sd - K\} = C.$$

The first two equations determine  $y_1$  and  $y_2$  uniquely:

$$y_1 = \frac{r - d}{r(u - d)}, \quad y_2 = \frac{u - r}{r(u - d)},$$

and these values are positive because  $u > r > d$ . Hence

$$C = \frac{(r - d) \max\{0, Su - K\} + (u - r) \max\{0, Sd - K\}}{r(u - d)}.$$

8.16 *Maximum volume rectangle inside a polyhedron.* Formulate the following problem as a convex optimization problem. Find the rectangle

$$\mathcal{R} = \{x \in \mathbf{R}^n \mid l \preceq x \preceq u\}$$

of maximum volume, enclosed in a polyhedron  $\mathcal{P} = \{x \mid Ax \preceq b\}$ . The variables are  $l, u \in \mathbf{R}^n$ . Your formulation should not involve an exponential number of constraints.

**Solution.** A straightforward, but very inefficient, way to express the constraint  $\mathcal{R} \subseteq \mathcal{P}$  is to use the set of  $m2^n$  inequalities  $Av^i \preceq b$ , where  $v^i$  are the  $(2^n)$  corners of  $\mathcal{R}$ . (If the

corners of a box lie inside a polyhedron, then the box does.) Fortunately it is possible to express the constraint in a far more efficient way. Define

$$a_{ij}^+ = \max\{a_{ij}, 0\}, \quad a_{ij}^- = \max\{-a_{ij}, 0\}.$$

Then we have  $\mathcal{R} \subseteq \mathcal{P}$  if and only if

$$\sum_{i=1}^n (a_{ij}^+ u_j - a_{ij}^- l_j) \leq b_i, \quad i = 1, \dots, m,$$

The maximum volume rectangle is the solution of

$$\begin{aligned} & \text{maximize} && (\prod_{i=1}^n (u_i - l_i))^{1/n} \\ & \text{subject to} && \sum_{i=1}^n (a_{ij}^+ u_j - a_{ij}^- l_j) \leq b_i, \quad i = 1, \dots, m, \end{aligned}$$

with implicit constraint  $u \succeq l$ . Another formulation can be found by taking the log of the objective, which yields

$$\begin{aligned} & \text{maximize} && \sum_{i=1}^n \log(u_i - l_i) \\ & \text{subject to} && \sum_{i=1}^n (a_{ij}^+ u_j - a_{ij}^- l_j) \leq b_i, \quad i = 1, \dots, m. \end{aligned}$$

## Solutions to additional exercises

1. *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.m`, 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  $R_{it}^*$ . 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.m`; you just have to add the code needed to find  $x^*$ .

*Hint:* The current version of `cvx` doesn't handle the logarithm directly. You can use `geo_mean()` to solve the problem exactly.

### Solution.

- (a) The following 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

```

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) The following code was used to generate the random event sequences and the trajectory plots:

```

% 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:



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.

2. *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  $\ell_1$  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  $\lambda$  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  $\ell_1$  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,:);

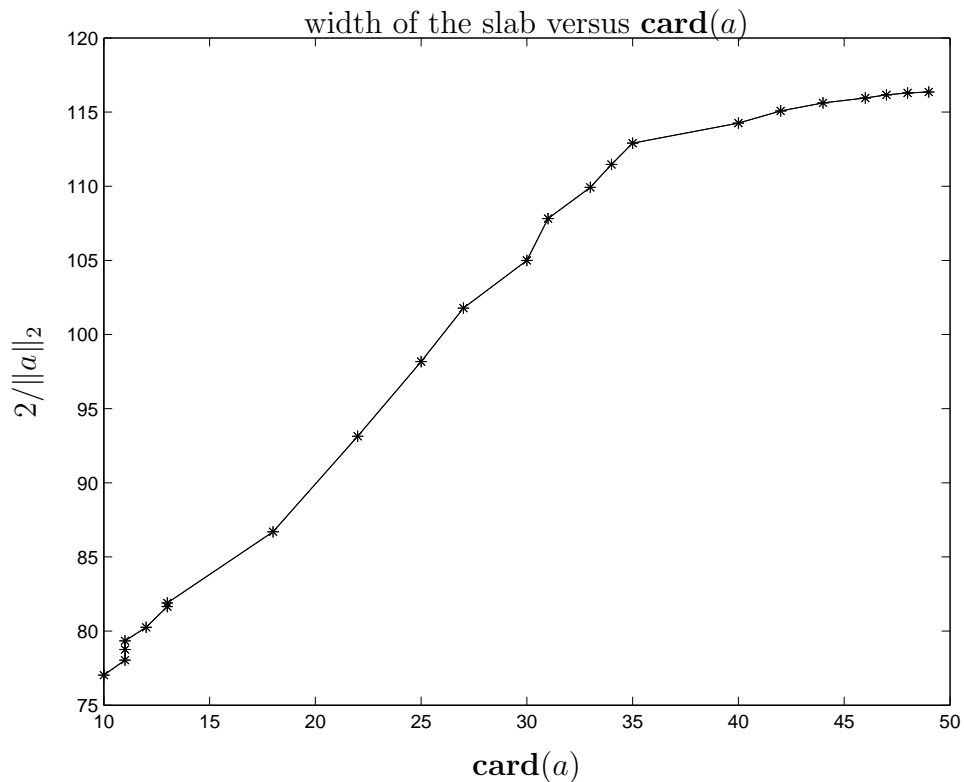
```

```

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.

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

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

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  $\alpha$  and  $\beta$  are known positive constants, with  $\alpha < \beta$ .) We want to find a maximum likelihood estimate of  $x$  and  $\phi$ , given  $y_i$ . (We also know  $a_i$ ,  $\sigma$ ,  $\alpha$ , and  $\beta$ .)

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  $\phi$  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  $\phi$  (*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  $\hat{x}_{\text{ml}}$ , 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) The measurement equations can be written

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

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

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

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

The log-likelihood function has the form

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

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

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

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

(b) The following Matlab code solve the given problem

```

nonlin_meas_data

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

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

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

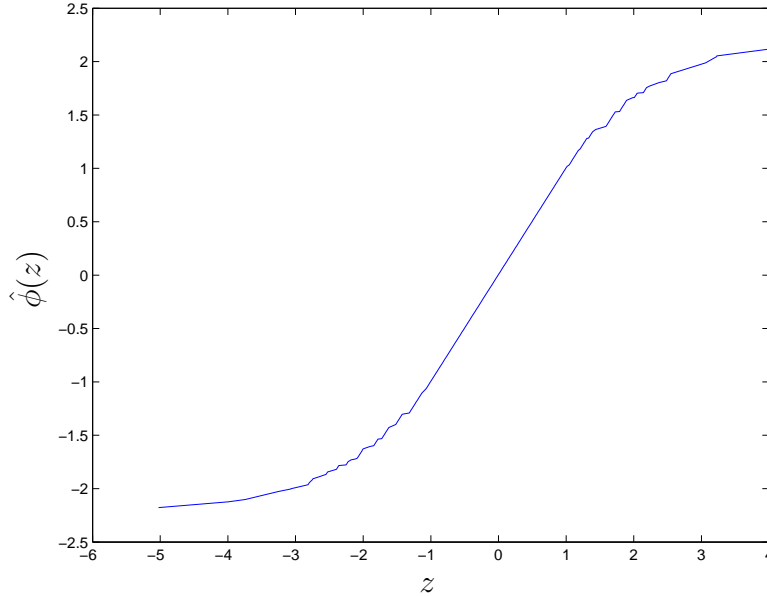
plot(z,y)
ylabel('y')
xlabel('z')
title('ML estimate of \phi')

```

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

4. *Robust least-squares with interval coefficient matrix.* An *interval matrix* in  $\mathbf{R}^{m \times n}$  is a matrix whose entries are intervals:

$$\mathcal{A} = \{A \in \mathbf{R}^{m \times n} \mid |A_{ij} - \bar{A}_{ij}| \leq R_{ij}, \quad i = 1, \dots, m, \quad j = 1, \dots, n\}.$$



**Figure 1** Maximum likelihood estimate of  $\phi$ .

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  $x_{\text{ls}}$  of the nominal problem (*i.e.*, minimize  $\|\bar{A}x - b\|_2$ ), and robust least-squares solution  $x_{\text{rls}}$ . 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  $x$ . Let  $f(x) = \sup_{A \in \mathcal{A}} \|Ax - b\|_2^2$ .

$$\begin{aligned} f(x) &= \sup\{\|Ax - b\|_2^2 \mid A = \bar{A} + \Delta, |\Delta_{ij}| \leq R_{ij}, i = 1, \dots, m, j = 1, \dots, n\} \\ &= \sup\{\|\bar{A}x + \Delta x - b\|_2^2 \mid |\Delta_{ij}| \leq R_{ij}, i = 1, \dots, m, j = 1, \dots, n\} \\ &= \sup\{\|r + \Delta x\|_2^2 \mid |\Delta_{ij}| \leq R_{ij}, i = 1, \dots, m, j = 1, \dots, n\} \end{aligned}$$

where  $r = \bar{A}x - b$ .

Since  $\|r + \Delta x\|_2^2 = \sum_{i=1}^m (r_i + \sum_{j=1}^n \Delta_{ij} x_j)^2 = \sum_{i=1}^m |r_i + \sum_{j=1}^n \Delta_{ij} x_j|^2$ , it's easy to see that this expression is separable in the rows of  $\Delta$ . So in order to find an expression for  $f(x)$ , we just need to find an expression for  $\sup\{|r_i + \sum_{j=1}^n \Delta_{ij} x_j| \mid |\Delta_{ij}| \leq R_{ij}, j = 1, \dots, n\}$ . The supremum is achieved by taking  $\Delta_{ij} = R_{ij} \mathbf{sign}(x_j)$  if  $r_i \geq$

0 and  $\Delta_{ij} = -R_{ij} \mathbf{sign}(x_j)$  if  $r_i < 0$ . The supremum is equal to  $|r_i| + \sum_{j=1}^n R_{ij}|x_j|$ . Therefore

$$\begin{aligned} f(x) &= \sum_{i=1}^m (|r_i| + \sum_{j=1}^n R_{ij}|x_j|)^2 \\ &= \|\bar{A}x - b + R|x|\|_2^2 \end{aligned}$$

The robust least-square problem then be reformulated as

$$\text{minimize } \|\bar{A}x - b + R|x|\|_2$$

Note that the objective function is convex since the Euclidian norm is convex and increasing on  $\mathbf{R}_+^m$  and  $|\bar{A}x - b + R|x||$  is convex and nonnegative.

- (b) The following script 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

```

This script returns the following results:

```

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

```

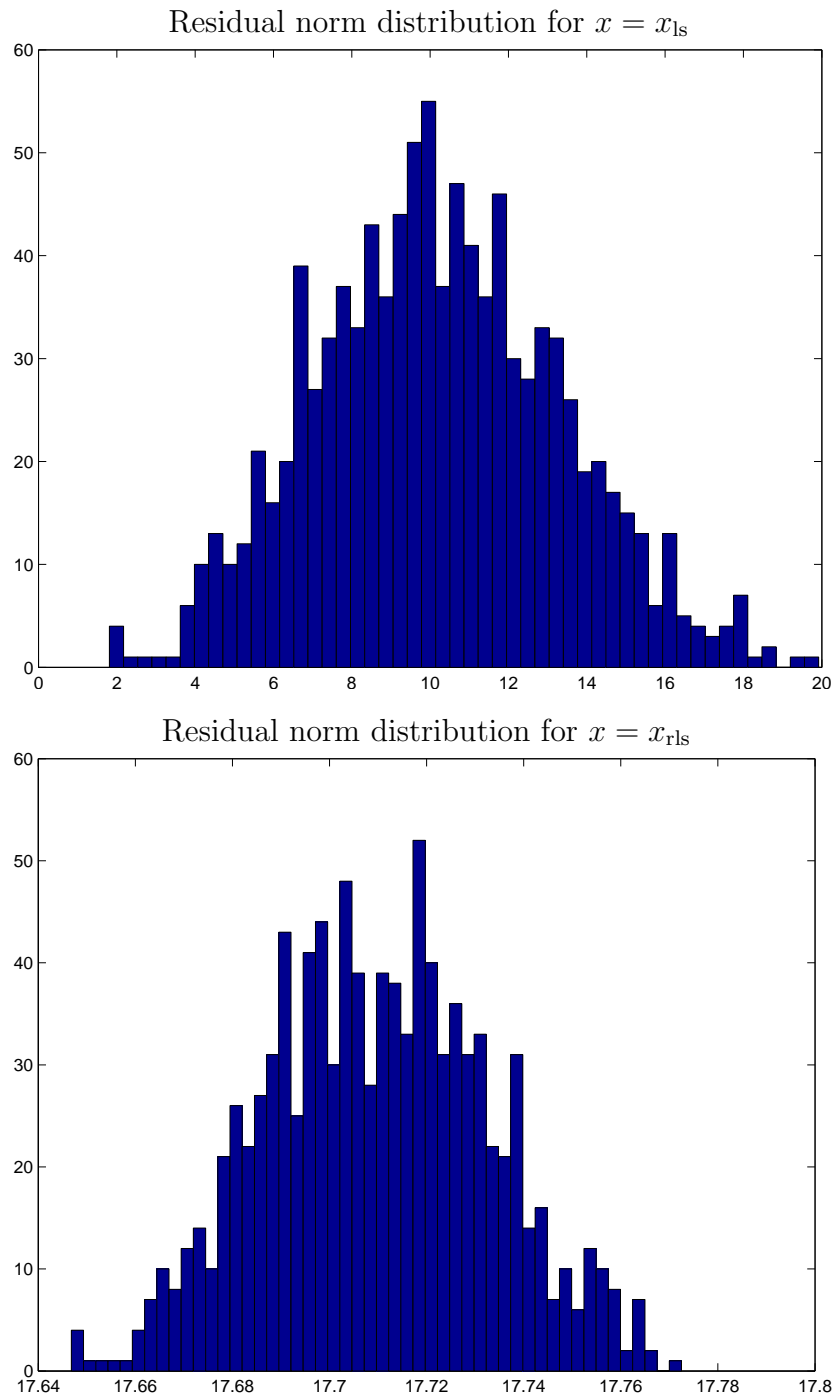
```

Residual norms for the worst-case problem when using robust solution:
    17.7940

```

We also generated, for fun, the following histograms showing the distribution of the residual norms for the case where  $x = x_{ls}$  and  $x = x_{rls}$ . 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 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);

```

5. *Efficient numerical method for a regularized least-squares problem.* We consider a regularized least squares problem with smoothing,

$$\text{minimize} \quad \sum_{i=1}^k (a_i^T x - b_i)^2 + \delta \sum_{i=1}^{n-1} (x_i - x_{i+1})^2 + \eta \sum_{i=1}^n x_i^2,$$

where  $x \in \mathbf{R}^n$  is the variable, and  $\delta, \eta > 0$  are parameters.

- Express the optimality conditions for this problem as a set of linear equations involving  $x$ . (These are called the normal equations.)
- Now assume that  $k \ll n$ . Describe an efficient method to solve the normal equations found in (5a). Give an approximate flop count for a general method that does not exploit structure, and also for your efficient method.
- A numerical instance.* In this part you will try out your efficient method. We'll choose  $k = 100$  and  $n = 2000$ , and  $\delta = \eta = 1$ . First, randomly generate  $A$  and  $b$  with these dimensions. Form the normal equations as in (5a), 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

$$\begin{bmatrix} F^{-1}a_1 & \cdots & F^{-1}a_m \end{bmatrix} = F^{-1} \begin{bmatrix} a_1 & \cdots & a_m \end{bmatrix}.$$

To compute this matrix using Matlab, you should collect the righthand sides into one matrix (as above) and use Matlab's backslash operator:  $F \setminus A$ . 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

$$x^T(A^T A + \delta\Delta + \eta I)x - 2b^T Ax + b^T b,$$

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

$$(A^T A + \delta\Delta + \eta I)x^* = A^T b. \tag{1}$$

(b) If no structure is exploited, then solving (1) 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 (1) 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 = 2000;
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) );
```