

## EE364a Homework 6 additional problems

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  $R_{\text{lt}}^*$ . 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.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.

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_1n_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.)

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.)

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

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

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  $x_{\text{ls}}$  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.

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.

- (a) Express the optimality conditions for this problem as a set of linear equations involving  $x$ . (These are called the normal equations.)
- (b) Now assume that  $k \ll n$ . Describe an efficient method to solve the normal equations found in (5a). 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  $n = 2000$ , 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 \ A`. This will do the right

thing: factor the matrix  $F$  once, and carry out multiple back substitutions for the righthand sides.