EE364a Homework 7 solutions

8.8 *Intersection and containment of polyhedra.* Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two polyhedra defined as

$$\mathcal{P}_1 = \{x \mid Ax \preceq b\}, \qquad \mathcal{P}_2 = \{x \mid Fx \preceq g\},$$

with $A \in \mathbf{R}^{m\times n}$, $b \in \mathbf{R}^m$, $F \in \mathbf{R}^{p\times n}$, $g \in \mathbf{R}^p$. Formulate each of the following problems as an LP feasibility problem, or a set of LP feasibility problems.

(a) Find a point in the intersection $\mathcal{P}_1 \cap \mathcal{P}_2$.

(b) Determine whether $\mathcal{P}_1 \subseteq P_2$.

For each problem, derive a set of linear inequalities and equalities that forms a strong alternative, and give a geometric interpretation of the alternative.

Repeat the question for two polyhedra defined as

$$\mathcal{P}_1 = \mathbf{conv}\{v_1, \ldots, v_K\}, \qquad \mathcal{P}_2 = \mathbf{conv}\{w_1, \ldots, w_L\}.$$

Solution

Inequality description.

(a) Solve

$$Ax \preceq b, \qquad Fx \preceq g.$$

The alternative is

$$A^T u + F^T v = 0, \qquad u \succeq 0, \qquad v \succeq 0, \qquad b^T u + g^T v < 0.$$

Interpretation: if the sets do not intersect, then they can be separated by a hyperplane with normal vector $a = A^T u = -F^T v$. If $Ax \preceq b$ and $Fy \preceq g$,

$$a^T x = u^T A x \le u^T b < -v^T g \le -v^T F x \le a^T y.$$

(b) $\mathcal{P}_1 \subseteq \mathcal{P}_2$ if and only if

$$\sup_{Ax\preceq b} f_i^T x \le g_i, \quad i = 1, \ldots, p.$$

We can solve p LPs, and compare the optimal values with g_i. Using LP duality we can write the same conditions as

$$\inf_{A^T z = f_i,\ z \succeq 0} b^T z \le g_i, \quad i = 1, \ldots, p,$$

which is equivalent to p (decoupled) LP feasibility problems

$$A^T z_i = f_i, \qquad z_i \succeq 0, \qquad b^T z_i \leq g_i$$

with variables z_i. The alternative for this system is

$$Ax \preceq \lambda b, \qquad f_i^T x > \lambda g_i, \qquad \lambda \geq 0.$$

If $\lambda > 0$, this means that $(1/\lambda)x \in \mathcal{P}_1$, $(1/\lambda)x \notin \mathcal{P}_2$.
If $\lambda = 0$, it means that if $\bar{x} \in \mathcal{P}1$, then $\bar{x} + tx \notin \mathcal{P}_2$ for t sufficiently large.

Vertex description.

(a) $\mathcal{P}_1 \cap \mathcal{P}_2 = \emptyset$? Solve

$$\lambda \succeq 0, \qquad \mathbf{1}^T \lambda = 1, \qquad \mu \succeq 0, \qquad \mathbf{1}^T \mu = 1, \qquad V\lambda = W\mu,$$

where V has columns v_i and W has columns w_i.
From Farkas' lemma the alternative is

$$V^T z + t\mathbf{1} \succeq 0, \qquad t < 0, \qquad -W^T z + u\mathbf{1} \succeq 0, \qquad u < 0,$$

i.e., $V^T z \succ 0$, $W^T z \prec 0$. Therefore z defines a separating hyperplane.

(b) $\mathcal{P}_1 \subseteq P_2$? For $i = 1, \ldots, K$,

$$w_i = V\mu_i, \qquad \mu_i \succeq 0, \qquad \mathbf{1}^T \mu_i = 1.$$

The alternative (from Farkas lemma) is

$$V^T z_i + t_i \mathbf{1} \succeq 0, \qquad w_i^T z_i + t_i < 0,$$

i.e., $w_i^T z_i \mathbf{1} < V^T z_i$. Thus, z_i defines a hyperplane separating w_i from $\mathcal{P}_2$.

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

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

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

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

The maximum volume rectangle is the solution of

$$\begin{array}{ll} \text{maximize} & \left(\prod_{i=1}^{n} (u_i - l_i)\right)^{1/n} \\ \text{subject to} & \sum_{i=1}^{n} (a_{ij}^{+} u_j - a_{ij}^{-} l_j) \le b_i, \quad i = 1, \ldots, m, \end{array}$$

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

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

A5.10 *Identifying a sparse linear dynamical system.* A linear dynamical system has the form

$$x(t+1) = Ax(t) + Bu(t) + w(t), \quad t = 1, \ldots, T-1,$$

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.*, $x(1), \ldots, x(T)$, $u(1), \ldots, u(T-1)$, 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

$$\sum_{t=1}^{T-1} \left\| W^{-1/2} \left(x(t+1) - \hat{A}x(t) - \hat{B}u(t) \right) \right\|_2^2 \le n(T-1) + 2\sqrt{2n(T-1)}.$$

(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

$$\begin{array}{ll} \text{minimize} & \mathbf{card}(\hat{A}) + \mathbf{card}(\hat{B}) \\ \text{subject to} & \sum_{t=1}^{T-1} \left\| W^{-1/2}\left(x(t+1) - \hat{A}x(t) - \hat{B}u(t)\right)\right\|_2^2 \in n(T-1) \pm 2\sqrt{2n(T-1)}, \end{array}$$

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

$$\begin{array}{ll} \text{minimize} & \|\mathbf{vec}(\hat{A})\|_1 + \|\mathbf{vec}(\hat{B})\|_1 \\ \text{subject to} & \sum_{t=1}^{T-1} \left\| W^{-1/2}\left(x(t+1) - \hat{A}x(t) - \hat{B}u(t)\right)\right\|_2^2 \leq n(T-1) + 2\sqrt{2n(T-1)}, \end{array}$$

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*us), '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.

A6.6 *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:

$$0 \leq x(1) \leq x(2) \leq \cdots \leq x(N).$$

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.m`. (This file contains the true signal `xtrue`, which of course you cannot use in creating your estimate.) Find the maximum likelihood estimate $\hat{x}_{\text{ml}}$, 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.*

Hint. The function `conv` (convolution) is overloaded to work with CVX.

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

$$\prod_{t=2}^{N+1} p\left(y(t) - \hat{y}_x(t)\right),$$

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,

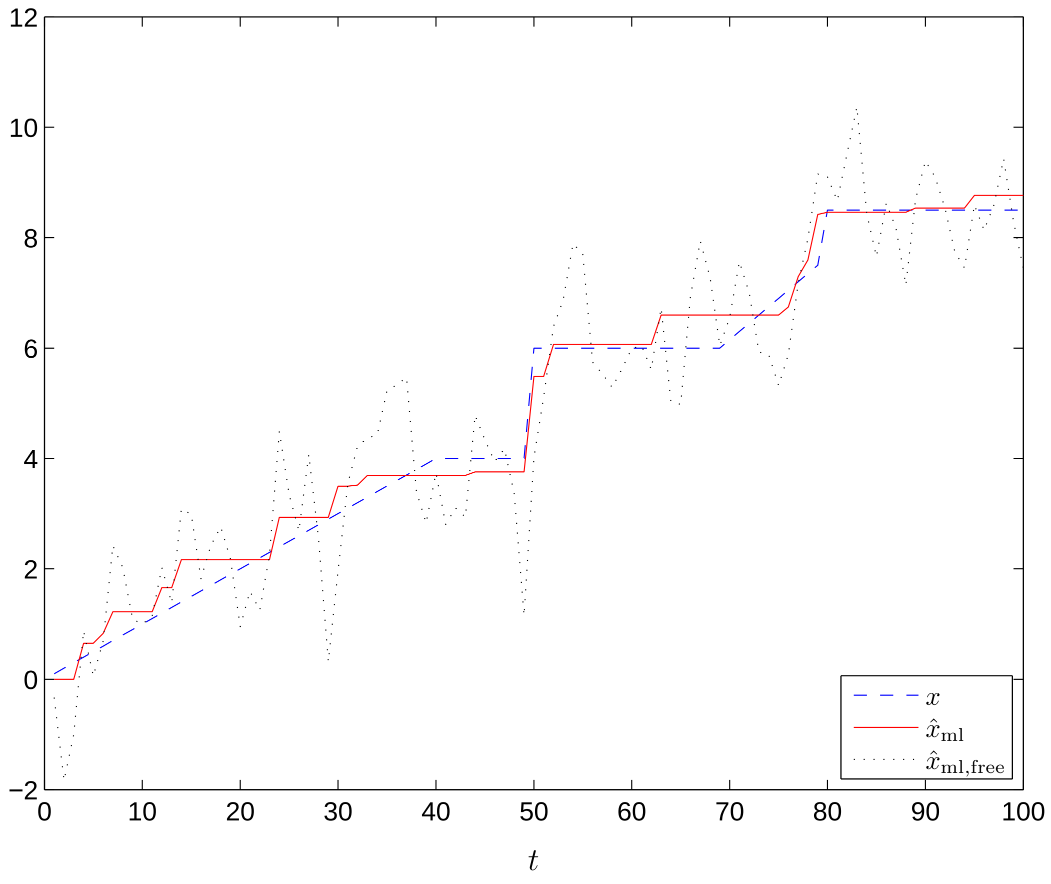
$$\hat{x}_{\text{ml,free}} = \operatorname*{argmin}_x \|\hat{y}_x - y\|_2^2.$$

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 $\hat{x}_{\text{ml}}$ by solving the QP

$$\begin{array}{ll} \text{minimize} & \|\hat{y}_x - y\|_2^2 \\ \text{subject to} & x(1) \geq 0 \\ & x(t) \leq x(t+1) \quad t = 1, \dots, N-1. \end{array}$$

(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 $\hat{x}_{\text{ml}}$, 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.



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
```

A8.5 *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 part (0a). 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 A and b with these dimensions. Form the normal equations as in part (0a), 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 $F^{-1}a_1, ..., F^{-1}a_m$, 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.

Solution.

(a) The objective function is

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

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

$$(A^TA + \delta\Delta + \eta I)x^\star = A^Tb. \qquad (1)$$

(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

$$x^\star = (\delta\Delta + \eta I)^{-1}g - (\delta\Delta + \eta I)^{-1}A^T(I + A(\delta\Delta + \eta I)^{-1}A^T)^{-1}A(\delta\Delta + \eta I)^{-1}g$$

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 $x^\star = z_1 - Z_2 z_3$ ($2nk$ 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) );
```

A13.15 *Sparse index tracking.* The (weekly, say) return of n stocks is given by a random variable $r \in \mathbf{R}^n$, with mean $\overline{r}$ and covariance $\mathbf{E}(r-\overline{r})(r-\overline{r})^T = \Sigma \succ 0$. 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 $\overline{r}$ 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 $\mathbf{card}(w)$, the number of nonzero entries in w. (To evaluate $\mathbf{card}(w)$, 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

$$\mathbf{E}(z - w^T r)^2 = \mathbf{E}((c-w)^T r)^2 = (c-w)^T(\Sigma + \overline{r}\overline{r}^T)(c-w),$$

so

$$E = \frac{\|(\Sigma + \overline{r}\overline{r}^T)^{1/2}(c-w)\|_2}{\|(\Sigma + \overline{r}\overline{r}^T)^{1/2}c\|_2}.$$

The heuristic is to choose w by solving the problem

$$\begin{array}{ll}\text{minimize} & \|w\|_1 \\ \text{subject to} & \|(\Sigma + \overline{r}\overline{r}^T)^{1/2}(c-w)\|_2 \leq 0.10\|(\Sigma + \overline{r}\overline{r}^T)^{1/2}c\|_2.\end{array}$$

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 $\mathbf{card}(w) = 50$. This can be compared to $\mathbf{card}(c) = 500$. 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%.

A14.8 *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 $p(T^{\rm td}) = 0$ and $\dot{p}(T^{\rm td}) = 0$, where $T^{\rm td}$ 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

$$\int_0^{T^{\rm td}} \gamma \|f(t)\|_2 \, dt,$$

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 $k = 1, \ldots, K$, where $T^{\rm td} = Kh$. Therefore we have

$$v_{k+1} = v_k + (1/m) f_k - g e_3, \quad p_{k+1} = p_k + (1/2)(v_k + v_{k+1}),$$

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 $t = 0, h, 2h, \ldots, Kh$.

(a) *Minimum fuel descent.* Explain how to find the thrust profile $f_1, \ldots, f_K$ that minimizes fuel consumption, given the touchdown time $T^{\rm td} = Kh$ 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.m`. Report the optimal total fuel consumption for part (a), and the minimum touchdown time for part (b). The data file also contains (commented out) code that plots your results. Use the code to plot the spacecraft trajectory and thrust profiles you obtained for parts (a) and (b).

Solution.

(a) To find the minimum fuel thrust profile for a given K, we solve

$$\begin{array}{ll} \text{minimize} & \sum_{k=1}^{K} \|f_k\|_2 \\ \text{subject to} & v_{k+1} = v_k + (1/m) f_k - g e_3, \quad p_{k+1} = p_k + (1/2)(v_k + v_{k+1}), \\ & \|f_k\|_2 \le F^{\max}, \quad (p_k)_3 \ge \alpha \|((p_k)_1, (p_k)_2)\|_2, \quad k = 1, \ldots, K \\ & p_{K+1} = 0, \quad v_{K+1} = 0, \quad p_1 = p(0), \quad v_1 = \dot{p}(0), \end{array}$$

with variables $p_1, \ldots, p_{K+1}$, $v_1, \ldots, v_{K+1}$, and $f_1, \ldots, f_K$. 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

$$\begin{array}{ll} \text{minimize} & 0 \\ \text{subject to} & v_{k+1} = v_k + (1/m) f_k - g e_3, \quad p_{k+1} = p_k + (1/2)(v_k + v_{k+1}), \\ & \|f_k\|_2 \le F^{\max}, \quad (p_k)_3 \ge \alpha \|((p_k)_1, (p_k)_2)\|_2, \quad k = 1, \ldots, K \\ & p_{K+1} = 0, \quad v_{K+1} = 0, \quad p_1 = p(0), \quad v_1 = \dot{p}(0), \end{array}$$

with variables $p_1, \ldots, p_{K+1}$, $v_1, \ldots, v_{K+1}$, and $f_1, \ldots, f_K$. 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 $(1/m)F^{\max} \ge g$, so we can use bisection to speed up our search.)

(c) The following code solves the problem:

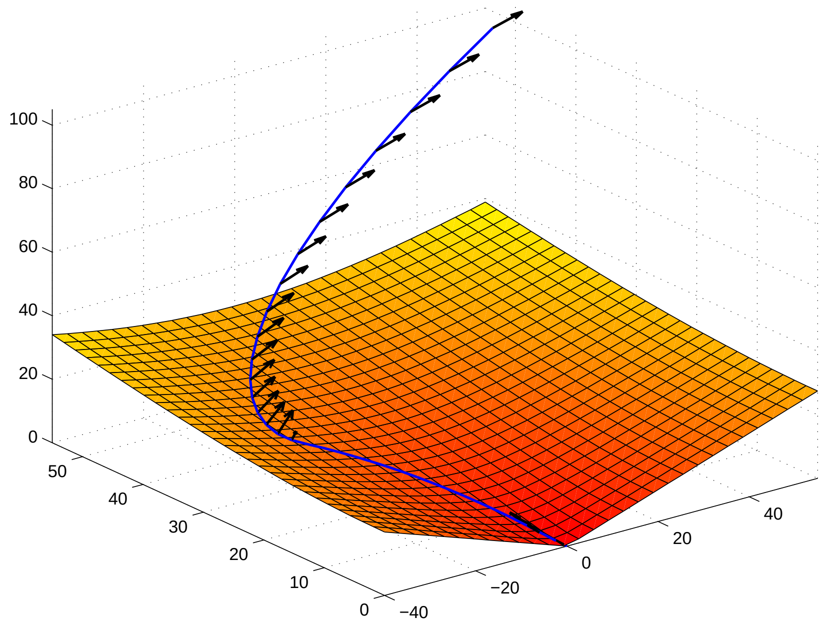
```
spacecraft_landing_data;

% 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)+(1/m)*f-g*repmat([0;0;1],1,K);
    p(:,2:K+1) == p(:,1:K)+(1/2)*(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)+(1/m)*f-g*repmat([0;0;1],1,Ki);
        p(:,2:Ki+1) == p(:,1:Ki)+(1/2)*(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');
```

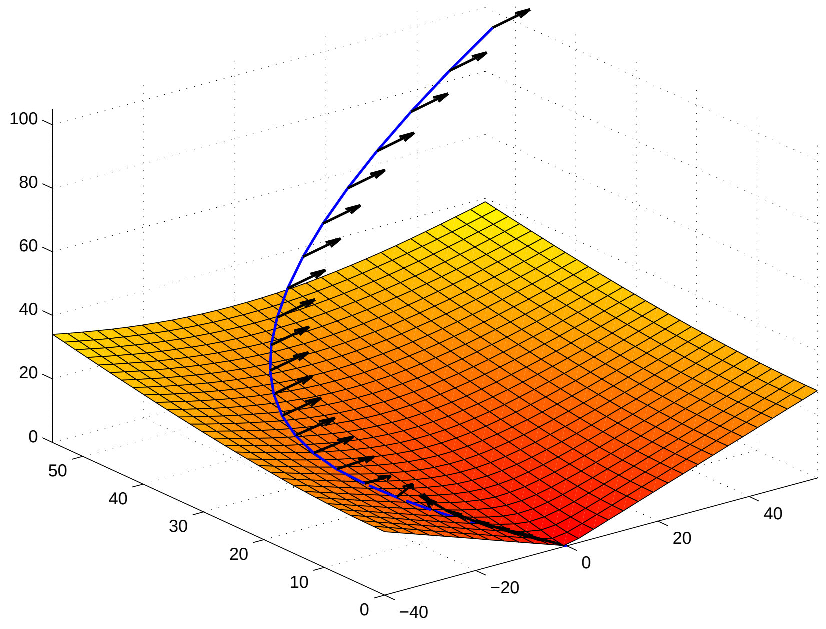
For part (a) the optimal total fuel consumption is 193.0. For part (b) the minimum touchdown time is $K = 25$. The following plots show the trajectories we obtain.

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).



A14.12 *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, $i = 1, \ldots, n$, at points equally spaced a distance d from each other along the given path. These are the variables to be chosen. (The heights $h_1, \ldots, h_n$ are called a *grading plan*.) We are given e_i, $i = 1, \ldots, n$, 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:

$$\begin{array}{rcll} |h_{i+1} - h_i|/d & \leq & D^{(1)}, & i = 1, \ldots, n-1 \\ |h_{i+1} - 2h_i + h_{i-1}|/d^2 & \leq & D^{(2)}, & i = 2, \ldots, n-1 \\ |h_{i+1} - 3h_i + 3h_{i-1} - h_{i-2}|/d^3 & \leq & D^{(3)}, & i = 3, \ldots, n-1, \end{array}$$

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

$$\phi^{\text{fill}}(u) = 2(u)_+^2 + 30(u)_+, \qquad \phi^{\text{cut}} = 12(u)_+^2 + (u)_+.$$

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.

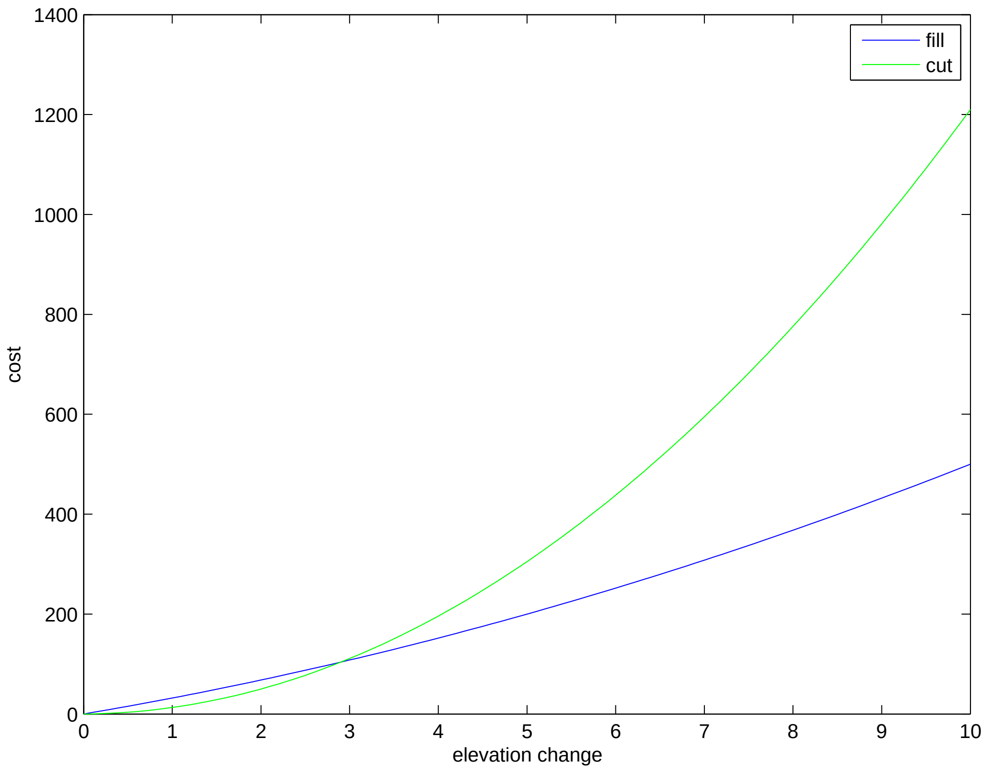
Solution.

(a) Fortunately, this problem in convex; C is convex since ϕ^{fill} and ϕ^{cut} are convex and increasing and max is convex. Therefore we simply solve the problem

$$\begin{array}{ll}\text{minimize} & C\\ \text{subject to} & |h_{i+1}-h_i|/d \le D^{(1)}, \quad i=1,\dots,n-1\\ & |h_{i+1}-2h_i+h_{i-1}|/d^2 \le D^{(2)}, \quad i=2,\dots,n-1\\ & |h_{i+1}-3h_i+3h_{i-1}-h_{i-2}|/d^3 \le D^{(3)}, \quad i=3,\dots,n-1,\end{array}$$

with optimization variables h_i.

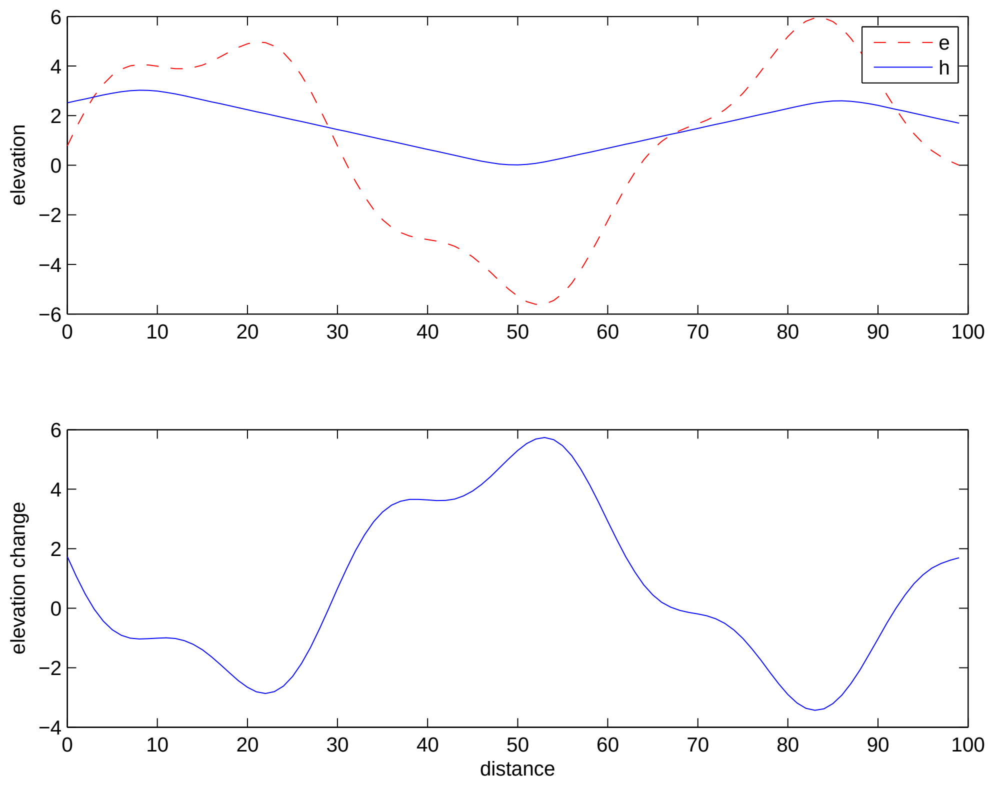
(b) For this problem instance we have cost functions shown below.



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.

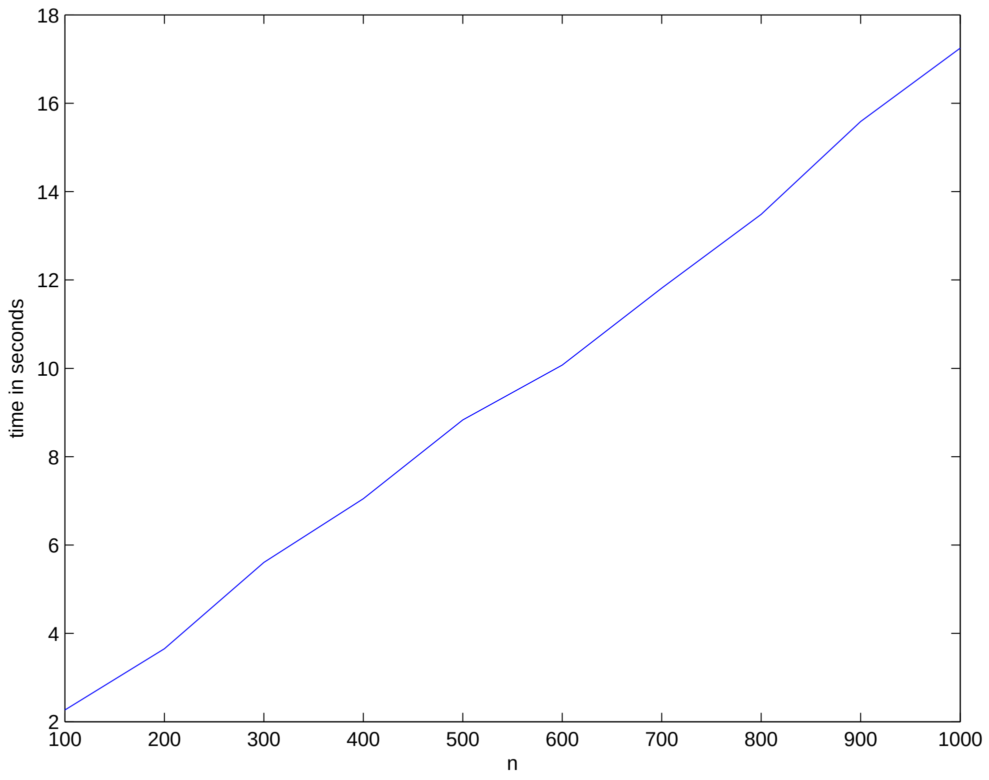
(c) Using our knowledge from EE364a we see immediately that this problem is well structured. Our objective is separable and our constraints have bandwidth of at most 4. For each Newton step we have to solve a banded system, which we know we can do in $\mathcal{O}(nk^2)$ flops, where $k = 4$ is the bandwidth. 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

$$\begin{array}{ll}\text{minimize} & tC - \sum_{i=1}^{n-1}\Big(\log(D^{(1)}d - h_{i+1} + h_i) + \log(D^{(1)}d + h_{i+1} - h_i)\Big) - \\ & \sum_{i=2}^{n-1}\Big(\log(D^{(2)}d^2 - h_{i+1} + 2h_i - h_{i-1})\Big) - \\ & \sum_{i=2}^{n-1}\Big(\log(D^{(2)}d^2 + h_{i+1} - 2h_i + h_{i-1})\Big) - \\ & \sum_{i=3}^{n-1}\Big(\log(D^{(3)}d^3 - h_{i+1} + 3h_i - 3h_{i-1} + h_{i-2})\Big) - \\ & \sum_{i=3}^{n-1}\Big(\log(D^{(3)}d^3 + h_{i+1} - 3h_i + 3h_{i-1} - h_{i-2})\Big).\end{array}$$

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).