EE263, Fall 2016-17
Oct 29-30, 2016

Midterm exam solutions

This is a 24 hour take-home midterm with 6 problems. Please turn it in at Bytes Cafe in the Packard building, 24 hours after you pick it up.

- You may use any books, notes, or computer programs (*e.g.*, matlab), but you may not discuss the exam with others until Nov. 10, after everyone has taken the exam. The only exception is that you can ask the course staff for clarification, by emailing to the staff email address `ee263-fall1617-staff@lists.stanford.edu`. We've tried pretty hard to make the exam unambiguous and clear, so we're unlikely to say much. Please do not post any exam related questions on Piazza.
- Since you have 24 hours, we expect your solutions to be legible, neat, and clear. Do not hand in your rough notes, and please try to simplify your solutions as much as you can. We will deduct points from solutions that are technically correct, but much more complicated than they need to be.
- Please check your email a few times during the exam, just in case we need to send out a clarification or other announcement. It's unlikely we'll need to do this, but you never know.
- Attach the official exam cover page (available when you pick up or drop off the exam) to your exam, and assemble your solutions to the problems in order, *i.e.*, problem 1, problem 2, …, problem 6. Start each solution on a new page.
- Please make a copy of your exam before handing it in. We have never lost one, but it might occur.
- If a problem asks for some specific answers, make sure they are obvious in your solutions. You might put a box around the answers, so they stand out from the surrounding discussion, justification, plots, etc.
- When a problem involves some computation (say, using matlab), we do not want just the final answers. We want a clear discussion and justification of exactly what you did, the matlab source code that produces the result, and the final numerical result. Be sure to show us your verification that your computed solution satisfies whatever properties it is supposed to, at least up to numerical precision. For example, if you compute a vector x that is supposed to satisfy $Ax = b$ (say), show us the matlab code that checks this, and the result. (This might be done by the matlab code `norm(A*x-b)`; be sure to show us the result, which should be very small.) *We will not check your numerical solutions for you, in cases where there is more than one solution.*

- In the portion of your solutions where you explain the mathematical approach, you *cannot* refer to matlab operators, such as the backslash operator. (You can, of course, refer to inverses of matrices, or any other standard mathematical construct.)

- Some of the problems are described in a practical setting, such as population dynamics, or signal processing. *You do not need to understand anything about the application area to solve these problems.* We've taken special care to make sure all the information and math needed to solve the problem is given in the problem description.

- Some of the problems require you to download and run a matlab file to generate the data needed. These files can be found at the URL

  ```
  http://ee263.stanford.edu/mterm16_mfiles/FILENAME
  ```

  where you should substitute the particular filename (given in the problem) for `FILENAME`. *There are no links on the course web page pointing to these files, so you'll have to type in the whole URL yourself.*

- Please respect the honor code. Although we encourage you to work on homework assignments in small groups, *you cannot discuss the midterm with anyone*, with the exception of EE263 course staff, until Nov. 10, when everyone has taken it an the solutions are posted online.

1. *Linear algebra done right*

   (a) Let $A \in \mathbf{R}^{n\times n}$, if $A^2 = A$ and $\text{rank}(A) = n$, must A be the identity matrix? Prove or disprove it with a counterexample.

   (b) Let $A \in \mathbf{R}^{n\times m}, B \in \mathbf{R}^{m\times p}$. Is the following always true:

   $$\dim(\text{range}(AB)) = \dim(\text{range}(B)) - \dim(\text{null}(A) \cap \text{range}(B)),$$

   where $\dim(\mathcal{V})$ gives the dimension of the vector space $\mathcal{V}$. Prove or disprove it with a counterexample.

   Note: The operation $\cap$ denotes intersection. That is, given any two sets A and B, $A \cap B$ is the set that contains elements both in A and in B. Consequently, $\text{null}(A) \cap \text{range}(B)$ in the problem means the set of all vectors that are in both the null space of A and the range of B.

   (c) Let $u_1, \dots, u_k \in \mathbf{R}^m$ and $v_1, \dots, v_k \in \mathbf{R}^n$ be column vectors. Consider the matrix $M = \sum_{i=1}^k u_i v_i^{\mathrm{T}}$. Compare the $\text{rank}(M)$ with k and prove your claim.

   (d) Let $M \in \mathbf{R}^{m\times n}$ be a rank k matrix. Is it always possible to find column vectors $u_1, \dots, u_k \in \mathbf{R}^m$ and $v_1, \dots, v_k \in \mathbf{R}^n$ such that $M = \sum_{i=1}^k u_i v_i^{\mathrm{T}}$? Prove or disprove (with a counter example) the claim.

   Note: You should only use material covered in EE263 lectures up till this point.

   (e) Let $G = (V, E)$ be a simple and undirected graph, where $V = \{1, 2, \dots, n\}$ is the set of n vertices and E is the set of all edges. Let the matrix $A \in \mathbf{R}^{n\times n}$ be the adjacency matrix of this graph G, defined as follows:

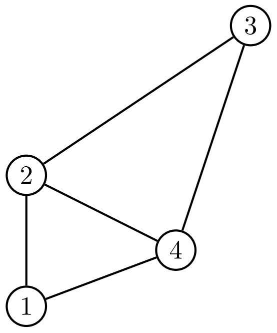
   $$A_{ij} = \begin{cases} 1, & (i,j) \text{ or } (j,i) \in E \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

   Note that since the graph G being simple means that there is no self-loop, hence $A_{ii} = 0$ for each i. Further, since the graph is undirected, $A_{ij} = A_{ji}, \forall i, j \in V$. A triangle in G is a set of three distinct vertices i, j, k where $(i,j) \in E, (j,k) \in E, (k,i) \in E$. Let T be the total number of triangles in the graph G. Prove the following statement:

   $$\text{Tr}(A^3) = 6T,$$

   where $\text{Tr}(A)$ gives the trace of the square matrix $A \in \mathbf{R}^{n\times n}$, which is equal to the sum of all its diagonal entries (*i.e.* $\text{Tr}(A) = \sum_{i=1}^n A_{ii}$).

   Note: To help you better understand the problem, consider the below example graph:



where the vertex set is $V = \{1, 2, 3, 4\}$, and the edge set is $E = \{(1, 2), (1, 4), (2, 3), (2, 4), (3, 4)\}$. The adjacency matrix is:

$$A = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

You can easily check that in this case the number of triangles is 2 and $\mathrm{Tr}(A^3) = 12$. As a final hint, in Review Session 4, we discussed what each entry in A^2 means. Think about what each entry on the diagonal of A^3 gives you.

Solution.

(a) A must be the identity matrix, because $\mathrm{rank}(A) = n$ implies that A is invertible. Consequently, let A^{-1} be A's inverse, and multiply A^{-1} on both sides of the equation $A^2 = A$ yields the result.

(b) True. Consider the linear transformation $T : \mathrm{range}(B) \to \mathrm{range}(AB)$ given by

$$T(x) = Ax, \qquad \forall x \in \mathrm{range}(B)$$

Note that $T(x) = 0$ if and only if $x \in \mathrm{range}(B)$ and $Ax = 0$. Consequently, the dimension of the nullspace (kernel) of T is $\dim(\mathrm{null}(A) \cap \mathrm{range}(B))$. By rank-nullity theorem, we have

$$\dim(\mathrm{range}(AB)) + \dim(\mathrm{null}(A) \cap \mathrm{range}(B)) = \dim(\mathrm{range}(B)),$$

which yields the result.

(c) $\mathrm{rank}(M) \leq k$. Note that $Mx = \sum_{i=1}^{k} u_i v_i^{\mathrm{T}} x$, which is a linear combination of k vectors $u_1, \ldots, u_k$. Consequently, the span has dimension at most k.

(d) It is always possible. Since M has rank k, pick k linearly independent columns (out of n columns in total) that span the column space of M: call them $u_1, \ldots, u_k$.

Consequently, each column in M can be written as a linear combination of those k columns, which leads to

$$M = UV,$$

where $U = [u_1, u_2, \ldots, u_k]$ and each column $\bar{v}_i$ $(i = 1, 2, \ldots, n)$ of V is the linear combination coefficient vector. Namely, for each column c_i of M, we have $c_i = U\bar{v}_i$. Denote each row of V to be v_i^{T}, we therefore have $M = \sum_{i=1}^{k} u_i v_i^{\mathrm{T}}$.

(e) First, by expanding the trace, we have

$$\mathrm{Tr}(A^3) = \sum_i \sum_j \sum_k A_{ij} A_{jk} A_{ki}$$

If $i = j$ or $j = k$ or $k = i$, then $A_{ij} A_{jk} A_{ki} = 0$. Consequently, the only case where $A_{ij} A_{jk} A_{ki}$ is not 0 is when $i \neq j$ and $j \neq k$ and $k \neq i$ and $A_{ij} = 1$, $A_{jk} = 1$, $A_{ki} = 1$, in which case i, j, k form a triangle. However, for each such triangle (i, j, k), there are six permutations of (i, j, k) that contributes to the sum. Consequently, one needs to multiply the total number of triangles by 6.

2. *Population dynamics*

An ecosystem consists of n species that interact (say, by eating other species, eating each other's food sources, eating each other's predators, and so on). We let $x(t) \in \mathbf{R}^n$ be the vector of deviations of the species populations (say, in thousands) from some equilibrium values (which don't matter here), in time period (say, month) t. In this model, time will take on the discrete values $t = 0, 1, 2, \ldots$. Thus $x_3(4) < 0$ means that the population of species 3 in time period 4 is below its equilibrium level. (It does not mean the population of species 3 is negative in time period 4.) The population (deviations) follows a discrete-time linear dynamical system:

$$x(t+1) = Ax(t).$$

We refer to $x(0)$ as the *initial population deviation*.

The questions below pertain to the specific case with $n = 10$ species, with matrix A given in `pop_dyn_data.m`.

(a) Suppose the initial deviation is $x(0) = e_4$ (meaning, we inject one thousand new creatures of species 4 into the ecosystem at $t = 0$). How long will it take to affect the other species populations? In other words, report a vector s, where s_i is the smallest t for which $x_i(t) \neq 0$. (We have $s_4 = 0$).

Note: We are not looking for an analytical answer here. You can answer this by running a matlab simulation.

(b) *Population control.* We can choose any initial deviation that satisfies $|x_i(0)| \leq 1$ for each $i = 1, \ldots, 10$. (We achieve this by introducing additional creatures and/or hunting and fishing.) What initial deviation $x(0)$ would you choose in order to maximize the population of species 1 at time $t = 10$? Explain your reasoning. Give the initial deviation, and using your selected initial deviation, give $x_1(10)$ and plot $x_1(t)$ versus t for $t = 0, \ldots, 40$.

Solution.

(a) We can simply simulate the first few time periods by iterating $x(t+1) = Ax(t)$ and check when the elements of the vectors $x(t)$ first become nonzero. The code for this method is given below.

We find that $s = (4, 2, 1, 0, 4, 3, 3, 2, 4, 3)$.

Alternatively, since the system evolves by repeated application of the matrix A, $x(t) = A^t x(0)$. Since $x(0) = e_4$, we will have $x_i(t) \neq 0$ exactly when $(A^t)_{i,4} \neq 0$. We can thus solve this problem by examining the 4th columns of each of the first few powers (say, 10) of A, looking to see when the elements first become nonzero.

(b) Let $B = A^{10}$, so $x(10) = A^{10}x(0) = Bx(0)$. We want to maximize $x_1(10)$, which is given by the inner product of the 1st row of B with $x(0)$. That is,

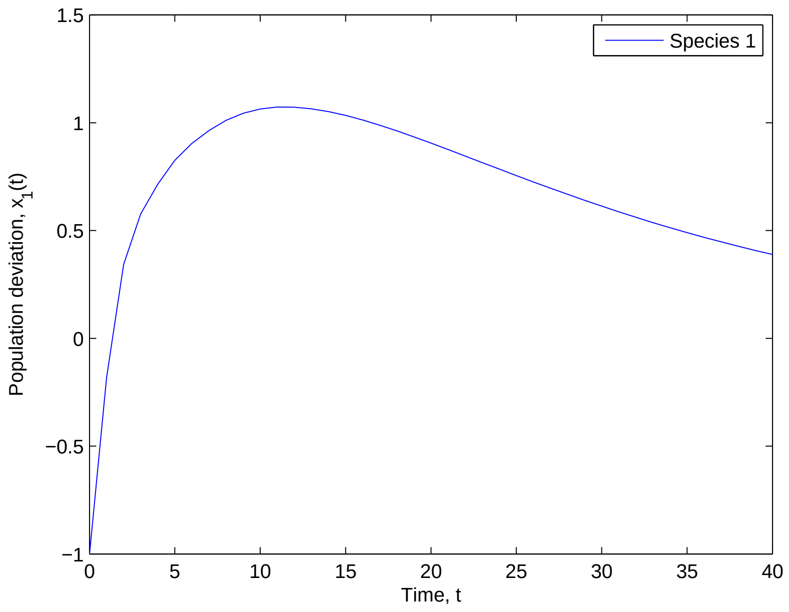
$$x_1(10) = \sum_{j=1}^{10} b_{1,j} x_j(0).$$

This sum is called *separable* because the jth term depends only on the variable $x_j(0)$. That means that the sum is maximized when each term is maximized separately. Since we are constrained to having the elements $|x_j(0)| \leq 1$, it's easy to see that the term $b_{1,j}x_j(0)$ is maximized when $x_j(0) = \text{sign}(b_{1,j})$, *i.e.*, $x_j(0)$ is either $+1$ or -1, with the sign matching the sign of $b_{1,j}$.

The code and the plot of $x_1(t)$ is given below. We find that

$$x(0) = (-1, -1, 1, -1, 1, 1, 1, 1, 1, -1)$$

and $x_1(10) = 1.06$. Note that, counterintuitively, the $x(0)$ which maximizes $x_1(10)$ sets the initial deviation of species 1 to -1!



```
pop_dyn_data

%part a
x0 = zeros(n,1);
x0(4) = 1;
T = 10;

s = zeros(n,1);
x = x0;
for i = 1:T
    s = s + (abs(x) <= 1e-6);
    x = A*x;
end

%part b
target_species = 1;
```

```
target_time = 10;

B = A^target_time;
x0 = sign(B(target_species,:))';
x_targ = B*x0;
fprintf('x_%d(t=%d) is %f\n',...
    target_species,target_time,x_targ(target_species))

%plotting for part b
T = 40;
xs = zeros(n,T+1);
x = x0;
for i = 1:T
    xs(:,i) = x;
    x = A*x;
end
xs(:,T+1) = x;

plot(0:T,xs(target_species,:))
xlabel('Time, t')
ylabel(['Population deviation, x_' num2str(target_species) '(t)'])
legend(['Species ' num2str(target_species)])
```

3. *Network design*

In this problem, you will be playing the role of a network engineer. You are given a fully connected network and your job is to set the bandwidths of the links in the network such that the state of the network gets as close as possible to a desired state, at a given point in time. Before we setup the problem, let's define some network terminology.

We consider a simple model of a communication network in which message packets of unit-length are carried between the nodes. Assume that you are given a 2-dimensional communication network formed by the set of nodes $N = \{(i,\, j) : i, j \in \mathbf{Z},\, 1 \leq i \leq m,\, 1 \leq j \leq n\}$. Note that each node (i, j) is indexed by two positive integers. It is your job to set up the links (*i.e.* connection wires) between the nodes. In this regard, for each node (i, j) you need to allocate the bandwidths $b_{(i,j)\to(k,l)}$, $(k, l) \in N$, $(k, l) \neq (i, j)$, for the wires connecting node (i, j) to other nodes (k, l). Note that the link that goes from node (i, j) to node (k, l) is distinct from the one going from node (k, l) to node (i, j). Also note that what we mean by $(k, l) \neq (i, j)$ is that either $k \neq i$, or $l \neq j$ or both. The bandwidths $b_{(i,j)\to(k,l)}$ determine the out-flow dividing coefficients at each node, meaning that when we have some out-flow at node (i, j), the following fraction of the out-flow will be carried to node (k, l)

$$\phi_{(i,j)\to(k,l)} = \frac{b_{(i,j)\to(k,l)}}{\sum_{(p,q)\in N,\,(p,q)\neq(i,j)} b_{(i,j)\to(p,q)}}.$$

(We assume that for all $(i, j) \in N$, $\sum_{(p,q)\in N,\,(p,q)\neq(i,j)} b_{(i,j)\to(p,q)} \neq 0$.) We study this network as a discrete-time system for $0 \leq t \leq 2$. Let $x_{(i,j)}(t)$ denote the total out-flow from node (i, j) at time t. In addition, let $u_{(i,j)}(t)$ denote the *known* in-flow entering the node (i, j) from outside the network at time t. One simple model for the network as it evolves toward a steady state through time is as follows:

$$x_{(i,j)}(t+1) = u_{(i,j)}(t) + \sum_{(k,l)\in N,\,(k,l)\neq(i,j)} \phi_{(k,l)\to(i,j)} x_{(k,l)}(t), \qquad 0 \leq t \leq 2.$$

Assume that at $t = 0$ there is no information out-flow in the network, which means that $x_{(i,j)}(0) = 0$ for all $(i, j) \in N$. Also, assume that a each time t, the in-flow entering the network is determined by known matrix $U(t) \in \mathbf{R}^{m\times n}$, such that $u_{(i,j)}(t) = U_{ij}(t)$. We define the state of the network at time t using a matrix $X(t) \in \mathbf{R}^{m\times n}$ for which $X_{ij}(t) = x_{(i,j)}(t)$. Finally, the problem of interest is as follows: as the network engineer, you want the state of the system at time $t = 2$ to be as close as possible to some desired state $Y \in \mathbf{R}^{m\times n}$ in the sense of minimizing the following cost function:

$$\|X(2) - Y\|_F^2, \tag{2}$$

where $\|X\|_F$ is the Frobenius norm of a matrix and is defined as:

$$\|X\|_F = \left(\sum_{i,j} X_{ij}^2\right)^{1/2}.$$

Remark: In reality, the in-flows and out-flows of the network should all be non-negative quantities. However, to simplify things, you need not enforce this constraint when solving this problem. In other words, it is ok if your solution produces negative bandwidths for some of the links.

(a) Assume that you are given a set of bandwidths $\{b^*_{(i,j)\to(k,l)} : (i,j),(k,l) \in N\}$ that minimizes the cost function (2). Determine whether this set of bandwidths would be the unique minimizer of our cost function, or we can find other solutions as well?

Hint: What would happen if you scale the optimal bandwidths $b^*_{(i,j)\to(k,l)}$?

Solution

Assume that the bandwidths $b^*_{(i,j)\to(k,l)}$'s minimize (2). Now we define some new bandwidths as follows:

$$b_{(i,j)\to(k,l)} = c_{(i,j)} b^*_{(i,j)\to(k,l)}, \; c_{(i,j)} \neq 0, \qquad \text{for all } (i,j),(k,l) \in N, \; (k,l) \neq (i,j).$$

It is easy to check that the dividing coefficients $\phi_{(i,j)\to(k,l)}$ do not change and as a result the value of (2) remains fixed. So we can conclude that there are infinite number of solutions to this minimization problem.

(b) Using the material covered in EE263 so far, propose a method to find the bandwidths $b_{(i,j)\to(k,l)}$ that minimize the cost function (2). Explicitly mention any assumptions that you make about the matrices that you use in your solution. Comment on whether or not your method gives a unique solution for optimal bandwidths $b_{(i,j)\to(k,l)}$ and why.

Hint: Note that your cost function (2) is not the standard least-squares cost function. However, there is a simple connection between the two. You can start by rearranging matrices $X(t)$ and Y into vectors. You will also need to somehow rearrange the bandwidths $b_{(i,j)\to(k,l)}$ into a vector. Now this is a little bit trickier. Think of how many entries this vector will have? To make this a little easier you can define $b_{(i,j)\to(i,j)} = 0$ for all $(i,j) \in N$ and include them as part of the bandwidths $b_{(i,j)\to(k,l)}$. Now try to rewrite you cost function (2) in terms of these new *vectorized* variables and see if it looks more like a standard least-squares cost function. Yes we know, the relationship between $\phi_{(i,j)\to(k,l)}$ and $b_{(i,j)\to(k,l)}$ is making the math kind of messy. That is why we added part (a) to this problem. It is an indirect hint on how to make the math simpler. Finally, remember you need to somehow ensure that in your final solution $b_{(i,j)\to(i,j)} = 0$ for all $(i,j) \in N$. How about adding some constraints to your minimization problem?

Solution.

First, a matrix $X \in \mathbf{R}^{m\times n}$ can be rearranged into a vector $v(X) \in \mathbf{R}^{mn}$ in the following manner:

$$X_{ij} = v_{i+(j-1)m}(X) \qquad \text{for } 1 \leq i \leq m, \; 1 \leq j \leq n.$$

In addition, let us define

$$b_{(i,j)\to(i,j)} = 0 \qquad \text{for all } (i,j) \in N. \tag{3}$$

Also, since multiplying all the bandwidths $b_{(i,j)\to(k,l)}$ by any nonzero constant $c_{(i,j)}$ does not change any thing, without loss of generality, we can assume that:

$$\sum_{(k,l)\in N} b_{(i,j)\to(k,l)} = 1 \qquad \text{for all } (i,j) \in N. \tag{4}$$

With this added constraint we now have $b_{(i,j)\to(k,l)} = \phi_{(i,j)\to(k,l)}$.Furthermore, let us define the vector $b \in \mathbf{R}^{(mn)^2}$ as follows:

$$b_{((l-1)m+(k-1))mn+(j-1)m+i} = b_{(i,j)\to(k,l)}$$

Finally we need to define the matrix $A \in \mathbf{R}^{mn\times(mn)^2}$ as follows:

$$A = \begin{bmatrix} v^T(U_0) & 0\dots0 & \dots & \dots & 0\dots0 \\ 0\dots0 & v^T(U_0) & 0\dots0 & \dots & 0\dots0 \\ 0\dots0 & 0\dots0 & v^T(U_0) & \dots & 0\dots0 \\ \vdots & \vdots & & \ddots & \vdots \\ 0\dots0 & 0\dots0 & \dots & & v^T(U_0) \end{bmatrix},$$

where $U_{0,ij} = u_{(i,j)}(0)$. Now we are ready to change (2) into something familiar for us. It is easily seen that

$$\|X(2) - Y\|_F^2 = \|Ab + v(U_1) - v(Y)\|_2^2. \tag{5}$$

As a result, from now on we try to minimize (5) instead of (2). But before we proceed, we note that there are two constraints (3) and (4) that have to be satisfied as well. So we need to write them in matrix notation. To do so, we define the matrices C_1, $C_2 \in \mathbf{R}^{mn\times(mn)^2}$ as follows:

$$C_{1,kl} = \begin{cases} 1 & \text{if } k = (j-1)m+i \text{ and } l = k(mn+1) - mn \text{ for all } (i,j) \in N \\ 0 & \text{o.w.} \end{cases}$$

$$C_2 = \begin{bmatrix} I_{mn} & I_{mn} & \cdots & I_{mn} \end{bmatrix}.$$

In addition we define the matrix C as follows:

$$C = \begin{bmatrix} C_1 \\ C_2 \end{bmatrix},$$

and the vector $d \in \mathbf{R}^{2mn}$ as follows:

$$d = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ \vdots \\ 1 \end{bmatrix}.$$

Now we can write the constraints in terms of C and d:

$$Cb = d.$$

In addition, for simplicity of notation let $y = v(Y) - v(U_1)$. We can wrap up everything as the following problem:

$$\begin{aligned} &\text{minimize } \|Ab - y\|_2^2 \\ &\text{subject to } Cb = d \end{aligned} \tag{6}$$

We know that $b \in \mathbf{R}^{(mn)^2}$ is optimal for this problem if and only if

$$\begin{bmatrix} A^T A & C^T \\ C & 0 \end{bmatrix} \begin{bmatrix} b \\ \lambda \end{bmatrix} = \begin{bmatrix} A^T y \\ d \end{bmatrix},$$

for some $\lambda \in \mathbf{R}^{2mn}$. Note that $A^T A$ and the block matrix are not invertible in this problem and hence, the formula in given pages 13 and 14 of lecture notes *Least-norm solutions of underdetermined equations* cannot be used. To combat this difficulty, we can have two different approaces:

i. **The first approach**: Since C is fat and full-rank, CC^T is invertible. Thus we can use the first block to solve and eliminate λ:

$$\begin{bmatrix} (I - C^\dagger C) A^T A \\ C \end{bmatrix} b = \begin{bmatrix} (I - C^\dagger C) A^T y \\ d \end{bmatrix}$$

It turns out that the coefficient matrix in this system is not full-rank (its rank is 26). However, because the normal equations are always consistent, we can still compute a solution using the pseudoinverse:

$$b = \begin{bmatrix} (I - C^\dagger C) A^T A \\ C \end{bmatrix}^\dagger \begin{bmatrix} (I - C^\dagger C) A^T y \\ d \end{bmatrix}$$

Since the coefficient matrix is singular, the optimal b is not unique. Note that we cannot use the singularity of normal equations to immediately conclude that the optimal b is not unique; the ambiguity could have been entirely in λ.

ii. **The second approach**: Consider the optimization problem (6) again. Since the matrix C is fat and full-rank any solution to this optimization problem has to be of the following form:

$$b = C^T(CC^T)^{-1}d + Fz,$$

where the matrix F constitutes some columns that are a basis for the $\mathcal{N}(C)$. If we plug this format into the objective function, the original optimization problem reduces to the following problem:

$$\text{minimize } \|AFz + AC^T(CC^T)^{-1}d - y\|_2^2,$$

The solution to which, based on the hint, is the following:

$$z = (AF)^\dagger \left(y - AC^T(CC^T)^{-1}d\right),$$

from which we can recover the vector b:

$$b = C^T(CC^T)^{-1}d + F(AF)^\dagger \left(y - AC^T(CC^T)^{-1}d\right).$$

(c) Implement your method on the data given in `network_design_data.m`. In this m-file, the matrices `U_0` $\in \mathbf{R}^{m\times n}$ and `U_1` $\in \mathbf{R}^{m\times n}$ are the in-flows at times $t = 0$, and $t = 1$, respectively. The matrix `Y` $\in \mathbf{R}^{m\times n}$ is the desired state at time $t = 2$. You only need to include your code and report the minimum value of cost function (2) for the given data.

Hint: The matlab function `pinv` can be used to find the pseudoinverse of a (not necessarily full-rank) matrix.

Solution.
The following matlab code solves the problem. The minimum value of the objective function is: 0.1520.

```
%% network_design_sol

run network_design_data

% creating the required matrices and vectors
v_Y = zeros(m*n,1);
v_U0 = zeros(m*n,1);
v_U1 = zeros(m*n,1);
A = zeros(m*n,m^2*n^2);
C1 = zeros(m*n,m^2*n^2);
C2 = repmat(eye(m*n),1,m*n);
d = [zeros(m*n,1);ones(m*n,1)];

for i = 1:m
    for j=1:n
```

```
        k = (j-1)*m+i;
        l = k*(m*n+1)-m*n;

        v_Y(k) = Y(i,j);
        v_U0(k) = U0(i,j);
        v_U1(k) = U1(i,j);

        C1(k,l) = 1;
    end
end

for i = 1:m*n
   A(i,(i-1)*m*n+1:i*m*n) = v_U0';
end

C = [C1;C2];
y = v_Y - v_U1;

% implenenting the solution method

% The first approach

BB = [(eye(size(C'*C)) - pinv(C)*C)*A'*A;C];
yy = [(eye(size(C'*C))-pinv(C)*C)*A'*y;d];

b = pinv(BB)*yy;

% computing the optimal value
norm(A*b-y)^2

% The second approach

F = null(C);
z = (A*F)\(y-A*C'/(C*C')*d);
norm(A*F*z-(y-A*C'/(C*C')*d))^2
```

4. *Grab life by the Llama*

You are the owner of an alpine mountaineering company, who offer tours with guide llamas. You want show your customers as much of the Llama Adventure WonderLand (L.A.W.L), but you don't want to over exert your animals. We can model a llama as a first order linear dynamical system:

$$x(t+1) = Ax(t) + Bu(t)$$

$$A = \begin{bmatrix} 1 & 0 & .1 & 0 \\ 0 & 1 & 0 & .1 \\ 0 & 0 & .5 & 0 \\ 0 & 0 & 0 & .5 \end{bmatrix} \qquad B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

where $x(t) \in \mathbf{R}^4$ is the state vector consisting of the llamas position and velocity, and the system input $u(t) \in \mathbf{R}^2$ is the velocity control effort.

$$x(t) = \begin{bmatrix} \text{pos}_x(t) \\ \text{pos}_y(t) \\ \text{vel}_x(t) \\ \text{vel}_y(t) \end{bmatrix} \qquad u(t) = \begin{bmatrix} u_x(t) \\ u_y(t) \end{bmatrix}$$

Our goal is minimize the overall effort put into controlling the llama between times 0 and T by minimizing the following cost function for a given T.

$$J = \sum_{t=0}^{T} \|u(t)\|^2$$

We also want to hit a set of positional waypoints $\{w_1, ..., w_n\}$, $w_i \in \mathbf{R}^2$ at target times $\{k_1, ..., k_n\}$. Hence, the minimization problem has the following constraints:

$$w_i = Hx(k_i) \quad i = 1, ..., n$$

where H is a matrix that when left multiplied by x returns the first two entries, since the waypoints are only positions.

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

Finally we get to the problem.

(a) Determine a single matrix equation that relates the control inputs $u(t)$ for $t = 0, ..., T$ to the waypoints w_i for $i = 1, ..., n$. Assume you are given an initial state $x(0) = x_0$ and that the final time is $T = k_n - 1$.

Hint: expand out the dynamics for $x(1)$, $x(2)$, ... until you find a pattern. Your final equation should express w_i's as an affine function of $u(t)$ for $t = 0, ..., T$ (remember, an affine function is a linear function plus a constant).

(b) Using the matrix equation of part (a), formulate the problem as a standard least-norm problem and find the optimal sequence of control inputs $u(t)$ that minimizes the cost function subject to the constraints. Leave your answer in terms of the problem data (A, B, H, x_0, w_i, k_i). Feel free to introduce new variables to simplify your math.

(c) Implement your solution with the data in the file `llama_data.m`. Each column of `w` gives the $(\text{pos}_x, \text{pos}_y)$ position waypoint for the corresponding time in `k`. Report the following plots:

- The trajectory of the llama with the waypoints clearly marked
- Each x and y velocity of the llama as a function of time

Also report your final minimized cost J.

(d) What are the conditions on matrices A and B that must hold in order to guarantee that the llama passes through all the waypoints at the target times? We are looking for a mathematical argument using what you have learned in EE263 up until this point.

Solution.

(a) We can expand out the dynamics equation to obtain

$$x(1) = Ax(0) + Bu(0)$$
$$x(2) = A^2x_0 + \begin{bmatrix} AB & B \end{bmatrix} \begin{bmatrix} u(0) \\ u(1) \end{bmatrix}$$

Continuing this for each way point yields

$$\begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix} = \begin{bmatrix} HA^{k_1} \\ \vdots \\ HA^{k_n} \end{bmatrix} x_0 + \begin{bmatrix} HA^{k_1-1}B & \dots & HB & \dots & 0 \\ \vdots & \vdots & \dots & \vdots & 0 \\ HA^{k_n-1}B & HA^{k_n-2}B & \dots & \dots & HB \end{bmatrix} \begin{bmatrix} u(0) \\ \vdots \\ u(k_n-1) \end{bmatrix}$$

where H is a matrix that when left multiplied returns the 1st two rows since the way points are only positions

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

(b) Let the above equation be

$$w = Fx_0 + Gu$$

Thus we want to solve the following optimization problem

$$\begin{array}{ll} \text{minimize} & \|u\| \\ \text{subject to} & Gu = w - Fx_0 \end{array}$$

The result is

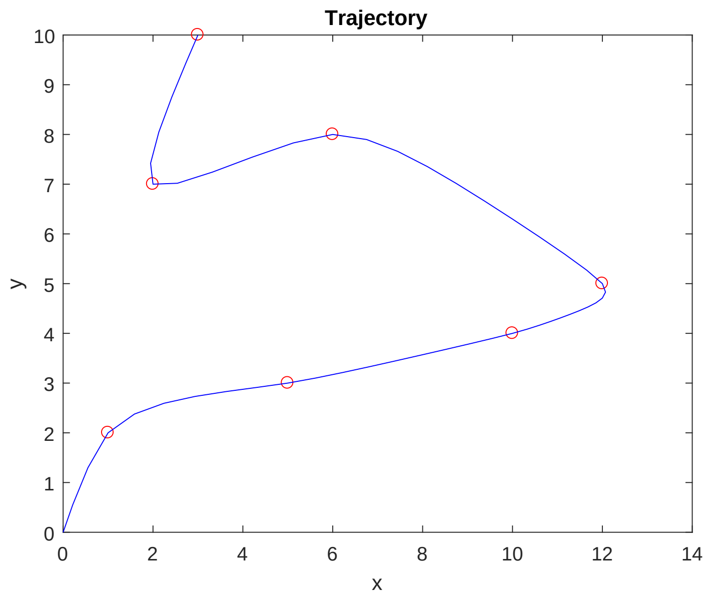
$$u = G^T(GG^T)^{-1}(w - Fx_0)$$

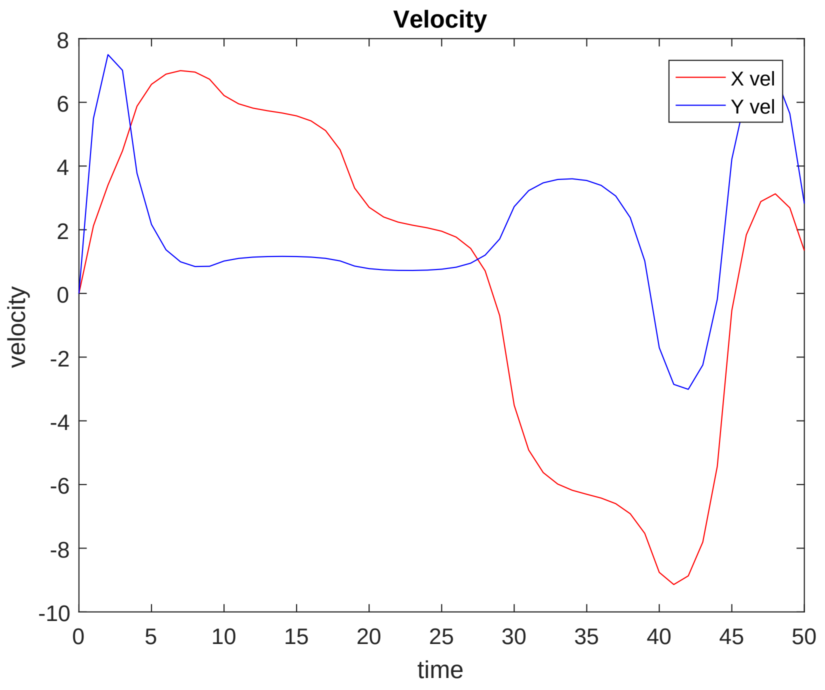
which is the least norm solution.

(c) the code that solves this problem

```
close all
clc

%% load data
run llama_data

%% build system
H=[1 0 0 0; 0 1 0 0];
G=zeros(length(k)*2, 2*k(end));
F=zeros(length(k)*2, 4);
L=zeros(length(k)*2, 1);
for i=1:length(k)
    %build G
   for j=1:k(i)
   G(2*i-1:2*i, 2*j-1:2*j)=H*A^(k(i)-j)*B;
   end
   %build F
   F(2*i-1:2*i, :)=H*A^(k(i));
   L(2*i-1:2*i)=w(:,i);
end

%% take least norm solution
U=G'*inv(G*G')*(L-F*x0);
U=reshape(U, 2, 50);

%% forward dynamics
T=k(end); %% final time
X=zeros(4, T+1);
X(:,1)=x0;

for t=1:T
    X(:, t+1)=A*X(:,t)+B*U(:,t);
end

%% plotting
%trajectory
plot(w(1,:), w(2,:), 'ro')
hold on
plot(X(1,:), X(2,:), 'b')
xlabel('x'); ylabel('y')
title('Trajectory')

%velocity
figure()
```

```
plot(0:T, X(3,:), 'r')
hold on
plot(0:T, X(4,:), 'b')
xlabel('time'); ylabel('velocity')
title('Velocity'); legend('X vel', 'Y vel')

J=norm(U, 'fro');
fprintf('The optimal cost is %.3f\n', J)
```





Optimal Cost is 23.458

(d) One condition that must hold is that $(w - Fx_0)$ is in the span of G.

However, an even more general condition relates to what is called the Controllability Matrix. If the matrix

$$\mathbf{C} = \begin{bmatrix} A^{n-1}B & A^{n-2}B & \dots & B \end{bmatrix}$$

is full rank then it is possible to reach any final state from any initial state within n steps, where n is the size of the state.

5. *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 conservation of various chemical species (metabolites) within the cell.

We focus on m metabolites in a cell, labeled $M_1, \ldots, M_m$. There are n (reversible) reactions going on, labeled $R_1, \ldots, R_n$, with reaction rates $v_1, \ldots, v_n \in \mathbf{R}$. A positive value of v_i means the reaction proceeds in the given direction, while a negative value of v_i means the reaction proceeds in the reverse direction. 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 $2v_j$. If v_j is negative, then metabolite M_i is produced at the rate $2|v_j|$.

As an example, suppose reaction R_1 has the form $M_1 \to M_2 + 2M_3$. 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 $M_4, \ldots, M_m$.) This corresponds to a first column of S of the form $(-1, 1, 2, 0, \ldots, 0)$.

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. (When $v_2 < 0$, it means that $|v_2|$ is the flow rate of the metabolite out of the cell.) This corresponds to a second column of S of the form $(1, 0, \ldots, 0)$.

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 (when the entry is negative) or created (when positive) 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 the *flux balance equation* $Sv = 0$.

Finally, we consider the effect of knocking out a gene. For simplicity, we'll assume that reactions $1, \ldots, n-1$ are each controlled by an associated gene, *i.e.*, gene G_k controls reaction R_k. Knocking out G_k has the effect of setting the associated reaction rate to zero.

Finally, we get to the point of all this. Suppose there is no $v \in \mathbf{R}^n$ that satisfies

$$Sv = 0, \quad v_k = 0, \quad v_n > 0.$$

This means there are no reaction rates consistent with cell growth, flux balance, and the gene knockout (remember, v_n is the cell growth rate). In this case, we predict that knocking out gene G_k will kill the cell, and call gene G_k an *essential gene.*

(a) Explain how to find all essential genes, given the stoichiometry matrix S. You can use any concepts from the class, *e.g.*, range, nullspace, least-squares.

(b) Carry out your method for the problem data given in `fba_data.m`. List all essential genes.

Solution. First we note that the condition $v_n > 0$ involves an inequality, which we have no way to deal with using 263 methods. But we can handle it, as follows. The conditions $Sv = 0$, $v_n > 0$ are homogeneous, that is, if v satisfies them, then so does αv, for any $\alpha > 0$. Using this, the condition is equivalent to $Sv = 0$, $v_n = 1$. (To see this, just choose $\alpha = 1/v_n$, where v is any vector that satisfies $Sv = 0$, $v_n > 0$.) So now we have a set of equations, which is indeed 263 material.

These conditions are the same as

$$s_n = -v_1 s_1 - \cdots - v_{n-1} s_{n-1},$$

where $s_1, \ldots, s_n$ are the columns of S. This in turn is the same as saying $s_n \in \operatorname{span}\{s_1, \ldots, s_{n-1}\}$.

Now let's bring in the idea of knocking out gene G_k. This sets v_k to zero, which simply means that we remove s_k from the list in the span above. That is, the growth condition with gene G_k knocked out is

$$s_n \in \operatorname{span}\{s_1, \ldots, s_{k-1}, s_{k+1}, \ldots, s_{n-1}\}.$$

So to find out which genes are essential, we check this condition for $k = 1, \ldots, n-1$. When the condition above does not hold, the gene is essential.

The matlab code below carries this out.

```
% find essential genes using flux based analysis
fba_data;

% first let's check that s_n is in range of the first n-1 columns

sn = S(:,n);
for k = 1:n-1
    % remove the kth and nth columns of S
    Stilde = [S(:,1:(k-1)) S(:,k+1:n-1)];
    if~(rank([Stilde sn]) == rank(Stilde))
        display(['gene ' num2str(k) ' is essential.']);
    end
end
```

Running the code, we find that the essential genes are G_4, G_8, and G_{11}.

Alternate approach. An alternative approach uses the nullspace. If G_k is an essential gene, there is no v such that $Sv = 0$, $v_k = 0$, and $v_n > 0$. In fact, the last condition $v_n > 0$ is equivalent to $v_n \neq 0$ since for any v satisfies $Sv = 0$, $v_k = 0$, and $v_n < 0$, its negative satisfies $Sv = 0$, $v_k = 0$, and $v_n > 0$. Therefore, if G_k is an essential gene, the conditions $Sv = 0$ and $v_k = 0$ imply $v_n = 0$.

To check this, we first consider all v that satisfy the conditions $Sv = 0$ and $v_k = 0$, that is, $v \in \mathcal{N}(A)$ with

$$A = \begin{bmatrix} S \\ e_k^T \end{bmatrix}.$$

Let $\dim(\mathcal{N}(A)) = r$ and $a_1, \ldots, a_r$ be a basis for $\mathcal{N}(A)$. To check if $v_n = 0$ for all v in the $\mathcal{N}(A)$, we simply check if the nth entry of each basis element a_k is 0.

6. *Iteratively reweighted least squares and total variation signal reconstruction*

In an ordinary least squares problem, we are given $A \in \mathbf{R}^{m\times n}$ (skinny and full rank) and $y \in \mathbf{R}^m$, and we choose $x \in \mathbf{R}^n$ in order to minimize

$$\|Ax - y\|_2^2 = \sum_{i=1}^m (\tilde{a}_i^T x - y_i)^2.$$

Note that the penalty that we assign to a measurement error does not depend on the sensor from which the measurement was taken. However, this is not always the right thing to do: if we believe that one sensor is more accurate than another, we might want to assign a larger penalty to an error in the measurement from the more accurate sensor. We can account for differences in the accuracies of our sensors by assigning sensor i a weight $w_i > 0$, and then minimizing

$$\sum_{i=1}^m w_i(\tilde{a}_i^T x - y_i)^2.$$

By giving larger weights to more accurate sensors, we can account for differences in the precision of our sensors.

(a) *Weighted least squares.* Explain how to choose x in order to minimize

$$\sum_{i=1}^m w_i(\tilde{a}_i^T x - y_i)^2,$$

where the weights $w_1, \dots, w_n > 0$ are given.

(b) *Iteratively reweighted least squares for ℓ_1-norm approximation.* Consider a cost function of the form

$$\sum_{i=1}^m w_i(x)(\tilde{a}_i^T x - y_i)^2. \tag{7}$$

There are multiple heuristics for minimizing a cost function of this form. Generally, a heuristic is an approach that is not guaranteed to work 100% of the time, but often produces very good results (You can assume the heuristic used for this problem always works). One heuristic for minimizing a cost function of the form given in (7) is *iteratively reweighted least squares*, which works as follows. First, we choose an initial point $x^{(0)} \in \mathbf{R}^n$. Then, we generate a sequence of points $x^{(1)}, x^{(2)}, \ldots \in \mathbf{R}^n$ by choosing $x^{(k+1)}$ in order to minimize

$$\sum_{i=1}^m w_i(x^{(k)})(\tilde{a}_i^T x^{(k+1)} - y_i)^2.$$

Each step of this algorithm involves updating our weights, and solving a weighted least squares problem. Suppose we want to use this method to minimize the

ℓ_1-norm approximation error, which is defined to be

$$\|Ax - y\|_1 = \sum_{i=1}^{m} |\tilde{a}_i^T x - y_i|,$$

where the matrix $A \in \mathbf{R}^{m \times n}$ and the vector $y \in \mathbf{R}^m$ are given. How should we choose the weights $w_i(x)$ to make the cost function in (7) equal to the ℓ_1-norm approximation error?

We now consider an important special case of the multi-objective least-squares problem. In *reconstruction problems*, we start with a *signal* represented by a vector $x \in \mathbf{R}^n$. The coefficients x_i correspond to the value of some function of time, evaluated (or *sampled*, in the language of signal processing) at evenly spaced points. It is usually assumed that the signal does not vary too rapidly, which means that usually, we have $x_i \approx x_{i+1}$. (In this problem we consider signals in one dimension, *e.g.*, audio signals, but the same ideas can be applied to signals in two or more dimensions, *e.g.*, images or video.)

The signal x is corrupted by an additive noise v:

$$x_{\text{cor}} = x + v.$$

The noise can be modeled in many different ways, but here we simply assume that it is unknown, small, and, unlike the signal, rapidly varying. The goal is to form an estimate $\hat{x}$ of the original signal x, given the corrupted signal x_{cor}. This process is called *signal reconstruction* (since we are trying to reconstruct the original signal from the corrupted version) or *de-noising* (since we are trying to remove the noise from the corrupted signal). Most reconstruction methods end up performing some sort of smoothing operation on x_{cor} to produce $\hat{x}$, so the process is also called *smoothing.*

One simple formulation of the reconstruction problem is the bi-objective problem

$$\text{minimize} \quad \left(\|\hat{x} - x_{\text{cor}}\|_2^2, \phi(\hat{x})\right), \tag{8}$$

where $\hat{x}$ is the variable and x_{cor} is a problem parameter. The function $\phi : \mathbf{R}^n \to \mathbf{R}$ is called the *regularization function* or *smoothing objective.* It is meant to measure the roughness, or lack of smoothness, of the estimate $\hat{x}$. The reconstruction problem (8) seeks signals that are close (in ℓ_2-norm) to the corrupted signal, and that are smooth, *i.e.*, for which $\phi(\hat{x})$ is small. We can find the optimal trade-off curve for the reconstruction problem (8) by minimizing the weighted-sum objective for different values of $\mu \geq 0$:

$$\text{minimize} \quad \|\hat{x} - x_{\text{cor}}\|_2^2 + \mu\phi(\hat{x}) \tag{9}$$

(c) *Quadratic smoothing.* The simplest reconstruction method uses the quadratic smoothing function

$$\phi_{\text{quad}}(x) = \sum_{i=1}^{n-1} (x_{i+1} - x_i)^2 = \|Dx\|_2^2,$$

We can obtain the optimal trade-off between $\|\hat{x}-x_{\text{cor}}\|_2$ and $\|D\hat{x}\|_2$ by minimizing

$$\|\hat{x} - x_{\text{cor}}\|_2^2 + \mu\|D\hat{x}\|_2^2,$$

where $\mu \geq 0$ parameterizes the optimal trade-off curve.

Explicitly specify the entries of matrix D and its dimension. The file `irwls_data_1.m` contains data for a corrupted signal `x_cor` $\in \mathbf{R}^{4000}$. Plot the optimal trade-off curve between $\|\hat{x} - x_{\text{cor}}\|_2$ and $\|D\hat{x}\|_2$. Based on the optimal trade-off curve, choose a reasonable value for μ and plot your reconstructed signal $\hat{x}$ for your choice of μ.

(d) The file `irwls_data_2.m` contains data for a corrupted signal `z_cor` $\in \mathbf{R}^{2000}$. In this case the signal is mostly smooth, but has several rapid variations or jumps in value; the noise is rapidly varying. Apply quadratic smoothing of part (c) to z_{cor} and plot the optimal trade-off curve between $\|\hat{z} - z_{\text{cor}}\|_2$ and $\|D\hat{z}\|_2$. Based on the optimal trade-off curve, choose a reasonable value for μ and plot your reconstructed signal $\hat{z}$ for your choice of μ.

(e) *Total variation reconstruction.* Simple quadratic smoothing works well as a reconstruction method when the original signal is very smooth, and the noise is rapidly varying. But, as seen in part (d), any rapid variations in the original signal will, obviously, be attenuated or removed by quadratic smoothing. In this part we describe a reconstruction method that can remove much of the noise, while still preserving occasional rapid variations in the original signal. The method is based on the smoothing function

$$\phi_{\text{tv}}(\hat{x}) = \sum_{i=1}^{n-1} |\hat{x}_{i+1} - \hat{x}_i| = \|D\hat{x}\|_1,$$

which is called the *total variation* of $x \in \mathbf{R}^n$. Like the quadratic smoothness measure ϕ_{quad}, the total variation function assigns large values to rapidly varying $\hat{x}$. The total variation measure, however, assigns relatively less penalty to large values of $|x_{i+1} - x_i|$.

Apply total variation reconstruction to the corrupted signal `z_cor` from part (d). plot the optimal trade-off curve between $\|\hat{z} - z_{\text{cor}}\|_2$ and $\|D\hat{z}\|_1$. Based on the optimal trade-off curve, choose a reasonable value for μ and plot your reconstructed signal $\hat{z}$ for your choice of μ. Compare the $\hat{z}$ you got from total variation reconstruction to what you got from quadratic smoothing. Plot the two reconstructed signals on the same graph and comment on the similarities and differences between them. Which method gives a better reconstruction of the original signal z?

Hint: First you need to use the results of parts (a) and (b) to reformulate the bi-objective problem

$$\text{minimize} \quad \|\hat{z} - z_{\text{cor}}\|_2^2 + \mu\phi_{\text{tv}}(\hat{z}), \tag{10}$$

as an iteratively reweighted least squares problem. You can choose your $\hat{z}^{(0)}$ to be equal to z_{cor} and then solve the problem for each value of μ in a few iterations. Generally, 5 to 10 iterations should be enough to get you decent convergence. Please note that generating the optimal tradeoff curve for this part can take considerably longer than the previous parts due to the iterative approach.

Solution.

(a) We can express the objective function as

$$\begin{aligned}\sum_{i=1}^{m} w_i(\tilde{a}_i^T x - y_i)^2 &= \sum_{i=1}^{m}(\sqrt{w_i}\tilde{a}_i^T x - \sqrt{w_i}y_i)^2 \\ &= \left\| \begin{bmatrix} \sqrt{w_1}\tilde{a}_1^T \\ \vdots \\ \sqrt{w_1}\tilde{a}_m^T \end{bmatrix} x - \begin{bmatrix} \sqrt{w_1}y_1 \\ \vdots \\ \sqrt{w_m}y_m \end{bmatrix} \right\|^2 \\ &= \|W^{\frac{1}{2}}Ax - W^{\frac{1}{2}}y\|^2,\end{aligned}$$

where $W^{\frac{1}{2}} = \mathbf{diag}(\sqrt{w_1}, \dots, \sqrt{w_m})$. Note that $(W^{\frac{1}{2}})^2 = W = \mathbf{diag}(w_1, \dots, w_m)$. Thus, minimizing this objective function is a least-squares problem; the solution is

$$x = ((W^{\frac{1}{2}}A)^T(W^{\frac{1}{2}}A))^{-1}(W^{\frac{1}{2}}A)^T(W^{\frac{1}{2}}y) = (A^TWA)^{-1}A^TWy.$$

(b) If we choose $w_i(x) = 1/|\tilde{a}_i^T x - y_i|$, then we have that

$$\sum_{i=1}^{m} w_i(x)(\tilde{a}_i^T x - y_i)^2 = \sum_{i=1}^{m} \frac{1}{|\tilde{a}_i^T x - y_i|}(\tilde{a}_i^T x - y_i)^2 = \sum_{i=1}^{m} |\tilde{a}_i^T x - y_i| = \|Ax - y\|_1.$$

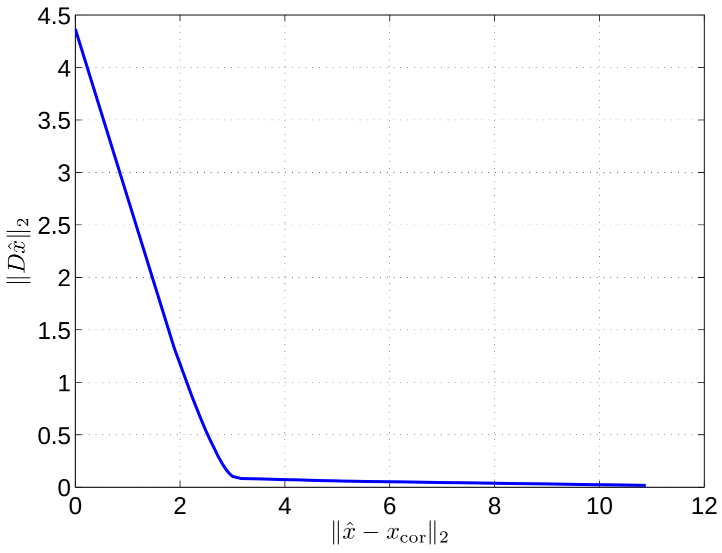
Note that $w_i(x)$ is undefined if $w_i(x) = |\tilde{a}_i^T x - y_i| = 0$. In this case, we can just take $w_i(x)$ to be some large value $w_{\max}$. However, note that in practice it is extremely unlikely that one of the residuals will be exactly equal to zero.

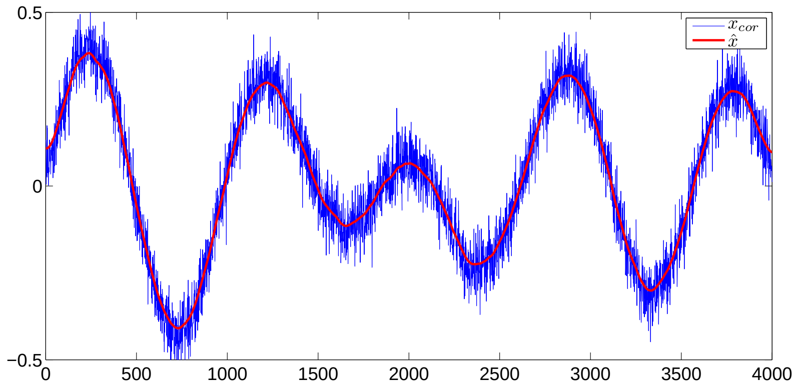
(c) Matrix $D \in \mathbf{R}^{(n-1)\times n}$ is the bidiagonal matrix

$$D = \begin{bmatrix} -1 & 1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & -1 & 1 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & -1 & 1 & 0 \\ 0 & 0 & 0 & \cdots & 0 & -1 & 1 \end{bmatrix}.$$

The following matlab code plots the optimal trade-off curve between the objectives $\|\hat{x} - x_{\text{cor}}\|_2$ and $\|D\hat{x}\|_2$. A reasonable value for μ is one that results in the values of objectives being close to the knee of the optimal trade-off curve. Values of μ between 500 and 2000 are acceptable for this part. Below is a plot of $\hat{x}$ for $\mu = 1000$.

```
clear all
close all

irwls_data_1
A = eye(length(x_cor));
g = zeros(length(x_cor)-1,1);
D = -eye(length(x_cor)) + diag(ones(length(x_cor)-1,1),1);
D(end,:) = [];

mu = [0:1:5, 10:5:100, 1000, 10000, 100000];

for i = 1:length(mu)
    A_tilde = [A; sqrt(mu(i))*D];
    y_tilde = [x_cor; sqrt(mu(i))*g];
    x_hat(:,i) = A_tilde\y_tilde;
    J1(i) = sqrt(sum((x_hat(:,i)-x_cor).^2));
    J2(i) = sqrt(sum((D*x_hat(:,i)).^2));
end

figure(1)
plot(J1,J2,'linewidth',2)
set(gca,'fontsize',16)
ylabel('$\|D\hat x\|_2$','interpreter','latex')
xlabel('$\|\hat x-x_{\rm cor}\|_2$','interpreter','latex')
grid on

figure(2)
plot(x_cor);
hold on
plot(x_hat(:,26),'r','linewidth',2)
ylim([-0.5 0.5])
legend({'$x_{cor}$','$\hat{x}$'},'Interpreter','latex')
set(gca,'fontsize',16)
```

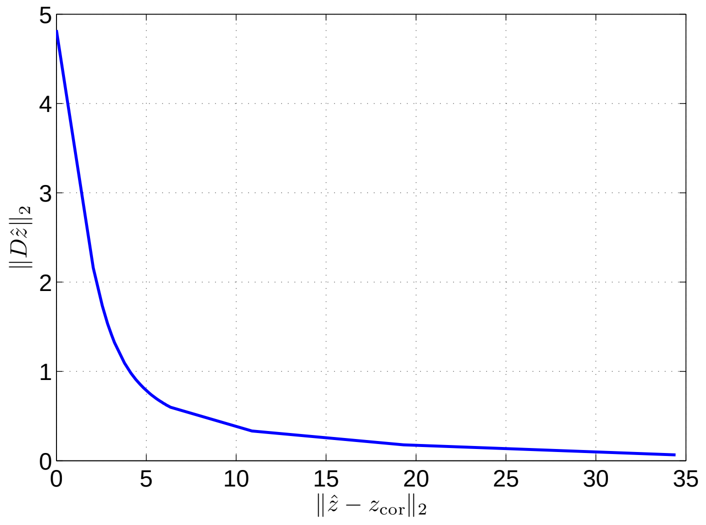


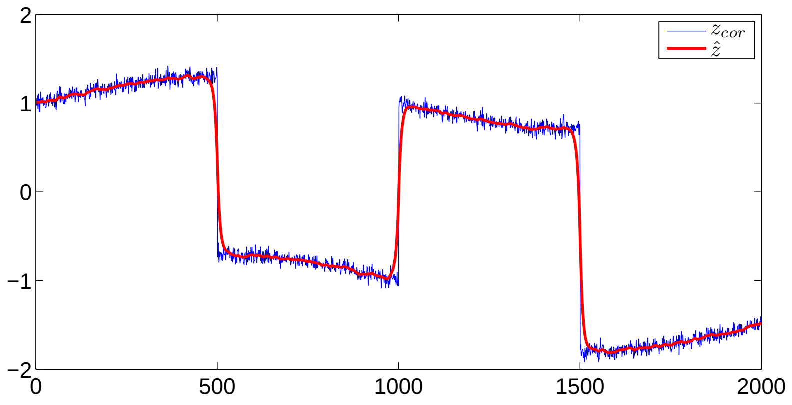


(d) Similar to part c, the following matlab code plots the optimal trade-off curve between the objectives $\|\hat{z} - z_{\mathrm{cor}}\|_2$ and $\|D\hat{z}\|_2$. A reasonable value for μ is one that results in the values of objectives being close to the knee of the optimal trade-off curve. Values of μ between 20 and 200 are acceptable for this part. Below is a plot of $\hat{z}$ for $\mu = 50$.

```
clear all
close all

irwls_data_2
A = eye(length(z_cor));
g = zeros(length(z_cor)-1,1);
D = -eye(length(z_cor)) + diag(ones(length(z_cor)-1,1),1);
D(end,:) = [];

mu = [0:1:5, 10:5:100, 1000, 10000, 100000];

for i = 1:length(mu)
```

```
    A_tilde = [A; sqrt(mu(i))*D];
    y_tilde = [z_cor; sqrt(mu(i))*g];
    z_hat(:,i) = A_tilde\y_tilde;
    J1(i) = sqrt(sum((z_hat(:,i)-z_cor).^2));
    J2(i) = sqrt(sum((D*z_hat(:,i)).^2));
end

figure(1)
plot(J1,J2,'linewidth',2)
set(gca,'fontsize',16)
ylabel('$\|D\hat z\|_2$','interpreter','latex')
xlabel('$\|\hat z-z_{\rm cor}\|_2$','interpreter','latex')
grid on

figure(2)
plot(z_cor);
hold on
plot(z_hat(:,15),'r','linewidth',2)
ylim([-2 2])
legend({'$z_{cor}$','$\hat{z}$'},'Interpreter','latex')
set(gca,'fontsize',16)
```

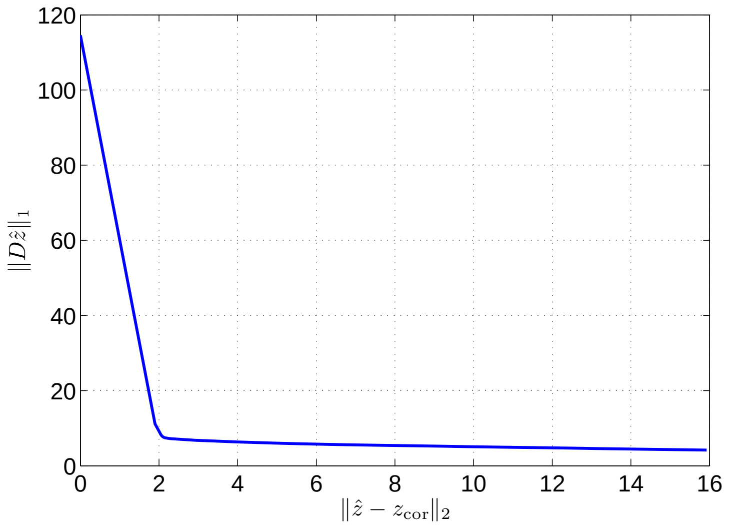


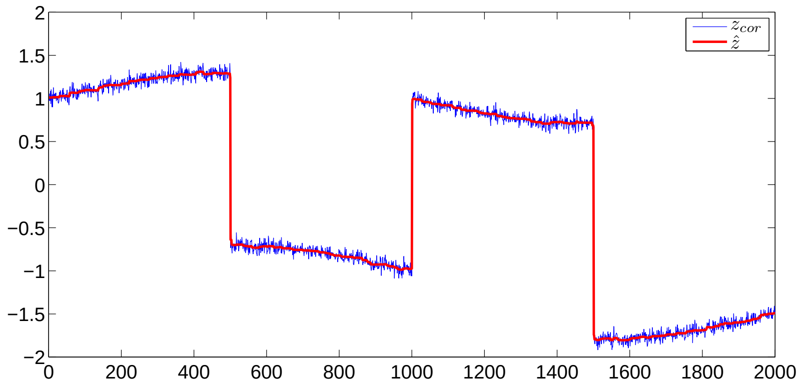


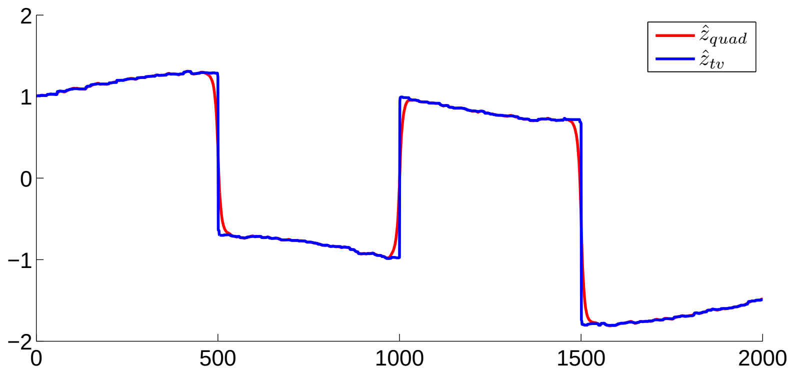
(e) The following matlab code plots the optimal trade-off curve between the objectives $\|\hat{z} - z_{\rm cor}\|_2$ and $\|D\hat{z}\|_1$. A reasonable value for μ is one that results in the values of objectives being close to the knee of the optimal trade-off curve. Values of μ between 0.2 and 2 are acceptable for this part. Below is a plot of $\hat{x}$ for $\mu = 0.2$. We can see that the total variation reconstruction does a much better job in preserving the sharp edges of the signal compared to quadratic smoothing. Both methods do get rid of the noise in a decent way.

```
clear all
close all

irwls_data_2
A = eye(length(z_cor));
g = zeros(length(z_cor)-1,1);
D = -eye(length(z_cor)) + diag(ones(length(z_cor)-1,1),1);
D(end,:) = [];
N_itr = 5; % Number of iterations for IRWLS
mu = [0:0.1:2, 5:5:100];

for i = 1:length(mu)
    z0 = z_cor;
    y_tilde = [z_cor; sqrt(mu(i))*g];
    for j = 1:N_itr
        w = 1./abs(D*z0);
        W = diag(sqrt(w));
        D_tilde = W*D;
        A_tilde = [A; sqrt(mu(i))*D_tilde];
        z_hat_temp = A_tilde\y_tilde;
        z0 = z_hat_temp;
    end
    z_hat(:,i) = A_tilde\y_tilde;
    J1(i) = sqrt(sum((z_hat(:,i)-z_cor).^2));
    J2(i) = sum(abs(D*z_hat(:,i)));
end

figure(1)
```

```
plot(J1,J2,'linewidth',2)
set(gca,'fontsize',16)
ylabel('$\|D\hat z\|_1$','interpreter','latex')
xlabel('$\|\hat z-z_{\rm cor}\|_2$','interpreter','latex')
grid on

figure(2)
plot(z_cor);
hold on
plot(z_hat(:,3),'r','linewidth',2)
ylim([-2 2])
legend({'$z_{cor}$','$\hat{z}$'},'Interpreter','latex')
set(gca,'fontsize',16)
```






$\hat{z}_{quad}$
$\hat{z}_{tv}$
2
1
0
−1
−2
0
500
1000
1500
2000