Final exam solutions

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

Please read the following instructions carefully.

• You may use any books, notes, or computer programs (e.g., Matlab), but you may not discuss the exam with anyone until Dec. 6, after everyone has taken the exam. The only exception is that you can ask the TAs or Stephen Boyd for clarification, by emailing to the staff email address. We’ve tried pretty hard to make the exam unambiguous and clear, so we’re unlikely to say much.

• 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 7. Start each solution on a new page. Do not collect all plots or code (for example) at the end of the exam; plots for problem 3 (say) should be with your solution to problem 3.

• Please make a copy of your exam before handing it in. We have never lost one, but it might occur.

• 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 with the Matlab code $\text{norm}(A*x-b)$; 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 (what appears to be) a practical setting. 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://www.stanford.edu/class/ee263/final_10_11_mfiles/FIENAME

   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 final with anyone, with the exception of Stephen Boyd and the TAs, until everyone has taken it.
1. Some attributes of a stable system. This problem concerns the autonomous linear dynamical system \( \dot{x} = Ax \), with \( x(t) \in \mathbb{R}^n \), which we assume is stable (i.e., all trajectories \( x(t) \) converge to zero as \( t \to \infty \)).

- **Peaking factor.** We define the peaking factor of the system as the largest possible value of \( \|x(t+\tau)/\|x(t)\| \), for any nonzero trajectory \( x \), any \( t \), and any \( \tau \geq 0 \).

- **Halving time.** We define the halving time of the system as the smallest \( \tau \geq 0 \) for which \( \|x(t+\tau)\| \leq \|x(t)\|/2 \) always holds, for all trajectories.

- **Minimum decorrelation time.** We define the minimum decorrelation time as the smallest possible \( \tau \geq 0 \) for which \( x(t+\tau) \perp x(t) \) can hold for some (nonzero) trajectory \( x \). This is the smallest possible time the state can rotate 90\(^\circ\). (If \( x(t+\tau) \perp x(t) \) never occurs for \( \tau \geq 0 \), then the minimum decorrelation time is \(+\infty\).)

(a) Explain how to find each of these quantities. Your method can involve some numerical simulation, such as a search over a (fine) grid of values of \( \tau \). You can assume that you do not need to search over \( \tau \) greater than \( \tau^{\text{max}} \), where \( \tau^{\text{max}} \) is known.

(b) Carry out your method for the specific case with

\[
A = \begin{bmatrix}
-1 & -5 & 0 & 0 \\
5 & 0 & 0 & 0 \\
0.4 & -1 & -0.6 & -6 \\
1 & 0 & 6 & 0
\end{bmatrix},
\]

with \( \tau^{\text{max}} = 10 \). We’d like all quantities to an accuracy of around 0.01.

**Solution:** We have \( x(t+\tau) = e^{\tau A}x(t) \), so the peaking factor is the maximum possible value of

\[
\frac{\|e^{\tau A}x(t)\|}{\|x(t)\|},
\]

over all trajectories and \( \tau \geq 0 \). The maximum of this over \( x(t) \) is just \( \|e^{\tau A}\| \). To find the maximum value of this over \( \tau \geq 0 \), we can just search, for example by plotting \( \|e^{\tau A}\| \) versus \( \tau \) over \([0, \tau^{\text{max}}]\).

The exact same plot can be used to find the halving time: We simply look for the smallest \( \tau \geq 0 \) for which \( \|e^{\tau A}\| \leq 1/2 \). (Since \( \|e^{\tau A}\| \) is a continuous function of \( \tau \), this is the same as the smallest \( \tau \geq 0 \) for which \( \|e^{\tau A}\| = 1/2 \).)

The condition \( x(t) \perp x(t+\tau) \) is equivalent to

\[
0 = x(t)^T e^{\tau A}x(t) = x(t)^T (1/2) (e^{\tau A} + (e^{\tau A})^T) x(t).
\]

(Yes, this is one of those times when failing to symmetrize a quadratic form leads to an error.) Let \( B(\tau) = (1/2) (e^{\tau A} + (e^{\tau A})^T) \). If \( B(\tau) \) is singular, say, \( B(\tau)z = 0 \) with
\( z \neq 0, \) then with \( x(t) = z, \) we have \( x(t + \tau) \perp x(t). \) If \( B(\tau) \) is nonsingular, then we cannot have \( x(t + \tau) \perp x(t) \) for any trajectory. So this means we can simply search for the smallest \( \tau \) for which \( B(\tau) \) is singular. We can do this by plotting its smallest singular value versus \( \tau, \) and finding the first time (if any) this plot goes to zero. The smallest singular value of \( B(0) = I \) is 1.

Since the singular values are the absolute values of the eigenvalues for a symmetric matrix, we can also plot the eigenvalues of \( B(\tau) \) versus \( \tau. \) The eigenvalues of \( B(0) = I \) are all one, we can just plot the minimum singular value versus \( \tau \) (since the minimum will be first one to cross zero).

In summary: We plot the minimum eigenvalue or singular value of \( B(\tau) \) to see when (or if!) it crosses zero, and that gives us the minimum decorrelation time.

The Matlab code below carries this out.

```matlab
 clears all; clc;
 A = [-1 -5 0 0; 5 0 0 0; 0.4 -1 -0.6 -6; 1 0 6 0];
 tau_max = 10;
 tauvals = 0:0.01:tau_max;
 norm_tau = []; eig_tau = [];
 for tau = tauvals
     norm_tau = [normTau norm(expm(tau*A))];
     eigTau = [eig Tau min(eig(0.5*(expm(tau*A)+expm(tau*A'))))];
 end
 subplot(2,1,1);
 plot(tauvals,norm_tau,'b-',[0 tau_max],[0.5 0.5],'r:');
 xlabel('tau'); ylabel('norm');
 subplot(2,1,2);
 plot(tauvals,eig Tau,'b-',[0 tau_max],[0 0],'r:');
 xlabel('tau'); ylabel('eig');

 [peak_val, peak_ind] = max(norm_tau);
 halving_ind = find(norm_tau <= 0.5);
 min_decorr_ind = find(eig Tau <= 0);

 display(['Peaking factor = ' num2str(peak_val) ' with tau = ' ...
         num2str(tauvals(peak_ind))]);
 display(['Halving time (tau) = ' num2str(tauvals(halving_ind(1)))]);
 display(['Minimum decorrelation time (tau) = ' ...
         num2str(tauvals(min_decorr_ind(1)))]);
```
The peaking factor is 1.1466 when $\tau = 1.1170$, the halving time is $\tau = 4.1278$, and the minimum decorrelation time is $\tau = 0.2465$. 

![Graphs showing the behavior of $\|e^{\tau A}\|$ and $\lambda_n(\frac{1}{2}(e^{\tau A} + e^{\tau A^T}))$ over $\tau$.]
3. **System with level alarms.** A linear dynamical system evolves according to
\[ \dot{x}(t) = Ax(t), \quad y(t) = Cx(t), \]
where \( x(t) \in \mathbb{R}^n \) is the state and \( y(t) \in \mathbb{R}^p \) is the output at time \( t \). You know \( A \) and \( C \), but not \( x(t) \) or \( y(t) \), except as described below.

The output is monitored using level alarms with thresholds. These tell us when \( y_i(t) \geq l_i \), where \( l_i \) is the threshold level for output component \( i \). (The threshold levels \( l_i \) are known.)

You have alarm data over the time interval \([0, T]\), of the following format. For each output component \( i = 1, \ldots, p \), you are given the (possibly empty) set of the intervals in \([0, T]\) over which \( y_i(t) \geq l_i \).

We now consider the specific problem with
\[
A = \begin{bmatrix} -0.9 & -4.2 & -2 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & 1 \end{bmatrix},
\]
\( T = 10, \ l_1 = l_2 = 1 \), and alarm intervals given below:
\[
y_1 : \quad [0, 1.0195], \quad [3.0288, 4.0863], \quad [6.4176, 6.9723]
\]
\[
y_2 : \quad [0.9210, 1.9402].
\]

The problem is to find an upper bound on how large \( \|x(T)\| \) can be, while being consistent with the given alarm data. We allow \(+\infty\) as an answer here; this means that there are trajectories with arbitrarily large values of \( \|x(T)\| \) that are consistent with the given alarm data. (We will deduct points for solutions that give bounds that are correct, but higher than they need to be.)

Give your bound on \( \|x(T)\| \). If it is \(+\infty\), explain. Of course, you must explain your method.

**Solution:** We have \( y(t) = Ce^{tA}x(0) \). The key observation is that at any endpoint of the alarm intervals, except for \( t = 0 \) and \( t = T \), the associated output component is exactly equal to the threshold level. If \( t_i \) is one of the endpoints (not equal to 0 or \( T \)), for output component \( k \), we have
\[
l_k = y_k(t_i) = c_k^T e^{t_iA}x(0),
\]
where \( c_k^T \) is the \( k \)th row of \( C \). We collect these to form the combined linear equations
\[
Fx(0) = b,
\]
where the \( i \)th row of \( F \) has the form \( c_k^T e^{t_iA} \), where \( k \) is the index of the output and the \( i \)th entry of \( b \) is \( l_k \). The number of rows of \( F \) (indexed by \( i \)) is equal to the total number of endpoints in the intervals, not including 0 or \( T \).
If this set of linear equations has a unique solution, we can determine $x(0)$ exactly, which in turn means we know $x(T)$ exactly. And so in this case we know $\|x(T)\|$ exactly; this is the best bound, of course.

In the other case, the problem is much harder to analyze exactly. But fortunately for the specific instance given, this does not occur. (This much harder problem can be partly addressed using methods from convex optimization, but obviously that’s not relevant here.)

By the way, it’s a bit counterintuitive that you can actually find the state trajectory exactly, given only the alarm data, which would seem to be pretty crude. It’s definitely not always possible, but in some cases (including the one here) it is.

The matlab code below forms the matrix $F$, and verifies that it has zero nullspace, so we can deduce $x(T)$ exactly. We find that

$$\|x(T)\| = 0.5023.$$ 

The initial condition and final state that give this value are

$$x(0) = \begin{bmatrix} 2.8 \\ -1.9 \\ 0.4 \end{bmatrix}$$

and

$$x(T) = \begin{bmatrix} 0.4868 \\ 0.0342 \\ -0.1190 \end{bmatrix}.$$ 

(We check that the smallest singular value of $F$ is $\sigma_{\min}(F) = 0.4057$, which is far enough from zero.) A plot of the simulated trajectory components $y(t)$ over $[0, T]$ for the initial condition shows that our computation of $x(0)$ is consistent with the crossing times.

\[
\begin{align*}
A &= \begin{bmatrix} -0.9 & -4.2 & -2 \\
1 & 0 & 0 \\
0 & 1 & 0 \end{bmatrix}; \\
C &= \begin{bmatrix} 1 & 0 & -1 \\
0 & 1 & 1 \end{bmatrix}; \\
n &= 3; \quad T = 10;
\end{align*}
\]

% crossings of y1 and y2, except t=0 and t=T
% t1 = [ 1.0195; 3.0288; 4.0863; 6.4176; 6.9723 ];
t2 = [ 0.9210; 1.9402 ];

% form the matrix F and vector b
F = [ C(1,:) * expm(t1(1)*A) 
     C(1,:) * expm(t1(2)*A) 
     C(1,:) * expm(t1(3)*A) 
     C(1,:) * expm(t1(4)*A) 
     C(1,:) * expm(t1(5)*A) ];
\[
\begin{align*}
C(2,:) \ast \exp(t2(1) \ast A) \\
C(2,:) \ast \exp(t2(2) \ast A) \end{align*}
\]

\[
b = \text{ones} \left( \text{length}(t1) + \text{length}(t2), 1 \right);
\]

% check that columns of F are linearly independent
sigma_min = \text{min} \left( \text{svd}(F) \right) \% gives 0.40569

% now solve for initial condition x0
x0 = F \setminus b;
% and check, just to be sure
\text{norm} \left( F \ast x0 - b \right)

% find the norm of x(T)
\begin{align*}
xT &= \exp(T \ast A) \ast x0; \\
\text{norm} \left( xT \right) &\% \text{ gives } 0.5023
\end{align*}
4. Minimum energy control with delayed destination knowledge. We consider a vehicle moving in $\mathbb{R}^2$, with dynamics

$$x(t + 1) = Ax(t) + Bu(t), \quad p(t) = Cx(t), \quad t = 1, 2, \ldots, \quad x(1) = 0,$$

where $x(t) \in \mathbb{R}^n$ is the state, $u(t) \in \mathbb{R}^m$ is the input, and $p(t) \in \mathbb{R}^2$ is the position of the vehicle, at time $t$. The matrices $A \in \mathbb{R}^{n \times n}$, $B \in \mathbb{R}^{n \times m}$ and $C \in \mathbb{R}^{2 \times n}$, and the initial state $x(1)$, are given.

The vehicle must reach a destination $d \in \mathbb{R}^2$ at time $t = M + N + 1$, where $M > 0$ and $N > 0$, i.e., we must have $p(M + N + 1) = d$. The subtlety here is that we are not told what $d$ is at $t = 1$; we simply know that it is one of $K$ possible destinations $d^{(1)}, \ldots, d^{(K)}$ (which we are given). At time $t = M + 1$, the destination (which is one of $d^{(1)}, \ldots, d^{(K)}$) will be revealed to you.

Thus, you must choose the inputs up to time $M$, $u(1), u(2), \ldots, u(M)$, independent of the actual final destination; but you can choose $u(M + 1), \ldots, u(M + N)$ depending on the final destination. We will denote the choice of these inputs, in the case when the final destination is $d^{(k)}$, as $u^{(k)}(M + 1), \ldots, u^{(k)}(M + N)$.

We will choose the inputs to minimize the cost function

$$\sum_{t=1}^{M} \|u(t)\|^2 + \frac{1}{K} \sum_{k=1}^{K} \sum_{t=M+1}^{M+N} \|u^{(k)}(t)\|^2,$$

which is the sum of squared-norm costs, averaged over all destinations.

(a) Explain how to find $u(1), \ldots, u(M)$ and $u^{(k)}(M + 1), \ldots, u^{(k)}(M + N)$, for $k = 1, \ldots, K$.

(b) Carry out your method on the data given in `delayed_dest_data.m`. Report the optimal cost function value, and for each possible destination plot the position of the vehicle $p^{(k)}(1), \ldots, p^{(k)}(M + N + 1)$. (The data file contains commented-out code for producing your plots.)

Comment briefly on the following statement: “Since we do not know where we are supposed to go until $t = M + 1$, there’s no point using the input (for which we are charged) until then.”

Solution:

(a) Let $u^{(k)} = (u^{(k)}(M + 1), \ldots, u^{(k)}(M + N))$ denote the input sequence chosen when $d = d^{(k)}$, $k = 1, \ldots, K$, and let $u = (u(1), \ldots, u(M))$. We can write

$$x(M + 1) = A^{M-1}Bu(1) + \cdots + ABu(M - 1) + Bu(M) = F^{(M)}u,$$

where

$$F^{(M)} = \begin{bmatrix} A^{M-1}B & \cdots & AB & B \end{bmatrix}.$$
For each destination $d^{(k)}$ we must have
\[
d^{(k)} = Cx^{(k)}(M + N + 1)
\]
\[
= CA^{N-1}Bu^{(k)}(M + 1) + \cdots + CBu^{(k)}(M + N) + CA^N x(M + 1)
\]
\[
= CF^{(N)}u^{(k)} + CGF^{(M)}u,
\]
where $G = A^N$. Defining $\bar{u}^{(k)} = (1/\sqrt{K})u^{(k)}$, our objective is to minimize the average cost
\[
E = \sum_{t=1}^{M} \|u(t)\|^2 + \frac{1}{K} \sum_{k=1}^{K} \sum_{t=M+1}^{M+N} \|u^{(k)}(t)\|^2
\]
subject to the constraint
\[
\begin{bmatrix}
  d^{(1)} \\
  d^{(2)} \\
  \vdots \\
  d^{(K)}
\end{bmatrix}
= \begin{bmatrix}
  \sqrt{K}CF^{(N)} & 0 & \cdots & 0 & CGF^{(M)} \\
  0 & \sqrt{K}CF^{(N)} & \cdots & 0 & CGF^{(M)} \\
  \vdots & \vdots & \ddots & \vdots & \vdots \\
  0 & 0 & \cdots & \sqrt{K}CF^{(N)} & CGF^{(M)}
\end{bmatrix}
\begin{bmatrix}
  \bar{u}^{(1)} \\
  \bar{u}^{(2)} \\
  \vdots \\
  \bar{u}^{(k)} \\
  u
\end{bmatrix}.
\]
This is clearly a least-norm problem, with variables $u, \bar{u}^{(1)}, \ldots, \bar{u}^{(k)}$.

Alternate solutions. We saw several alternate solutions. Here’s one. We observe that everything depends on $x(M+1)$, the state when the final destination is revealed. If you know this state, then the problem decouples into two parts: We calculate $u(1), \ldots, u(M)$ as the minimum energy control to get to this state; and we calculate $u^{(k)}(M + 1), \ldots, u^{(k)}(M + N)$ separately as the minimum energy control to get from there to final target.

But you don’t know the state $x(M+1)$. Instead, we can optimize over it. To do this, we calculate the average optimal cost to get to the destination from $x(M+1)$, and we calculate the optimal cost to get to $x(M+1)$ from zero. These are both quadratic functions, so their sum is a quadratic function. We set the gradient to zero, and we have the solution. We will spare you the matrices and calculations.

Some approximate solutions. We saw several approximate solutions, including some that give pretty good answers, judged numerically, at least for the particular problem instance considered in part (b).

Here is a typical one. Since at first we do not know where we are going, we should act as if we are headed towards the average destination $\bar{d} = (1/K) \sum k = 1^K d^{(k)}$. We do this using standard least-norm method (minimum energy control), with a single destination. Then, at time $t = M + 1$, when the actual destination is revealed, we simply re-calculate of input as the minimum-energy trajectory from where we are to the final destination. This is not a bad method, at least in some cases. But it doesn’t minimize the cost function we gave you.
An interesting observation. Some of those who took the alternative approach made an interesting observation: The optimal $u(1), \ldots, u(M)$ in fact depends only on the average destination $\overline{d}$. (You can get to this conclusion using our method, too.) While we didn’t ask you to show this, it’s a pretty interesting fact.

By the way, this does not mean that the approximate solution described above is correct. While the optimal $u(1), \ldots, u(M)$ depend only on the average destination $\overline{d}$, it is not as simple as just a minimum energy control to get to $\overline{d}$.

(b) Carrying out our method with the data in `delayed_dest_data.m` we find that the optimal energy is $E = 1.13 \times 10^{-3}$. The trajectories are shown below. The solid blue line shows the first portion of the trajectory (before the destination is revealed), and the dotted red line shows the second portion. In fact, the input for $t = 1, \ldots, M$ is nonzero; that is, we do use the input even before we know the destination. You could say that before we know the destination, we still apply an input that puts us in a good situation, once the destination is revealed.

The matlab code is given below.

```
delayed_dest_data;

% form matrices FN, FM, and G
```
G = A^N; FM = [];
for i = 1:M FM = [A^(i-1)*B, FM]; end;
FN = []; for i = 1:N FN = [A^(i-1)*B, FN]; end;

% form least-norm matrix
H = [];
for i = 1:K
    H = [H; [zeros(2,(i-1)*m*N),sqrt(K)*C*FN,zeros(2,(K-i)*m*N),C*G*FM]];
end
d = reshape(D,2*K,1);
U = pinv(H)*d;
E = U'*U

U2 = reshape(U(1:m*N*K),m*N,K);
U1 = U(m*N*K+1:end);

% plot the trajectories
X = zeros(n,M+N+1,K); P = zeros(2,M+N+1,K);
for k = 1:K
    for i = 1:M
        X(:,i+1,k) = A*X(:,i,k)+B*U1((i-1)*m+1:i*m);
P(:,i,k) = C*X(:,i,k);
    end
    for i = M+1:M+N
        X(:,i+1,k) = A*X(:,i,k)+sqrt(K)*B*U2((i-M-1)*m+1:(i-M)*m,k);
P(:,i,k) = C*X(:,i,k);
    end
    P(:,M+N+1,k) = C*X(:,M+N+1,k);
end

figure; hold on;
plot(P(1,1:M+1,1),P(2,1:M+1,1),'b-','linewidth',1.5);
plot(P(1,1,1),P(2,1,1),'bo','MarkerSize',5,'MarkerFaceColor','b');
axis([-4, 7, -4, 1]);
for k = 1:K
    plot(P(1,M+1:M+N+1,k),P(2,M+1:M+N+1,k),'r-','linewidth',1.5);
    plot(P(1,M+N+1,k),P(2,M+N+1,k),'ro','MarkerSize',5,'MarkerFaceColor','r');
end
plot(P(1,M+1,1),P(2,M+1,1),'bo','MarkerSize',5,'MarkerFaceColor','b');
5. One of these vectors doesn’t fit. The file one_of_these_data.m contains an \( n \times m \) matrix \( X \), whose columns we denote as \( x^{(1)}, \ldots, x^{(m)} \in \mathbb{R}^n \). The columns are (vector) data collected in some application. The ordering of the vectors isn’t relevant; in other words, permuting the columns would make no difference.

One of the vectors doesn’t fit with the others.

Find the index of the vector that doesn’t fit. Carefully explain your method, and especially, in what way the vector you’ve chosen doesn’t fit with the others. Your explanation can be algebraic, or geometric (or both), but it should be simple to state, and involve ideas and methods from this course.

Since the question is vague, clarity in your explanation of your method and approach is very important. In particular, we want a nice, short explanation. We will not read a long, complicated, or rambling explanation.

**Solution.** A quick check of the column norms, means, and few other quantities doesn’t reveal any one that stands out. Yes, of course we arranged it that way.

One general approach to find the one that doesn’t fit is to remove each vector, one by one, from the data set, and then carry out some test on the remaining vectors. We’re looking for a case in which we remove a vector, and the result changes a lot.

Let’s start with the singular value decomposition \( U \Sigma V^T \) of the data matrix \( X^{(i)} = [x^{(1)} \ldots x^{(i-1)} x^{(i+1)} \ldots x^{(m)}] \). If \( \sigma_n \), the smallest singular value of \( X^{(i)} \), is small, then the columns lie approximately in a hyperplane. (Algebraically, it means that the components of the vectors almost satisfy a nonzero linear equation.)

When we carry this out we find that, sure enough, when we remove the \( i = 147 \) vector, \( \sigma_n \) drops considerably, from a number around 7.24 for other values of \( i \) to around 0.0345.

We can explain this in two ways. Algebraically, we can say that all vectors except the 147th approximately satisfy the linear equation \( u_n^T x^{(j)} \approx 0 \) (here we can use the \( u_n \) found from the SVD after we remove the 147th vector). Geometrically, we can say that all vectors lie within a distance around 0.0345 from the hyperplane defined by \( u_n^T z = 0 \) except \( x^{(147)} \), which is a distance approximately 7.24 from it.

In fact, it is not necessary to remove the vector before taking the SVD of the data matrix. If \( X = U \Sigma V^T \), then we simply take a look at the component \( u_n^T x_i = \sigma_n (v_n)_i \) of each \( x_i \) in the space spanned by the “weakest” direction defined by the entire data set, which is enough to spot the offending vector index.

We saw several other methods that found the vector that didn’t fit, and gave a clear explanation of how it didn’t fit. Here is one: You calculate the angles between all pairs of the vectors, and then consider the smallest angle between each vector and the others. Doing this, you’ll find that \( x^{(147)} \) makes a substantially larger minimum angle to the others. This was fine (assuming it was described clearly) — we didn’t say how you were to identify the one that didn’t fit, but we did ask that you clearly explain how you found it, and in what way it didn’t fit.
one_of_these_data;

% let’s check sigma_min when we remove each column
for i=1:m
    % remove column i
    XX = X(:,[1:i-1] [i+1:m]);
    s = svd(XX);
    sigma_min(i) = s(n);
end

plot(sigma_min)

% another method: take SVD of whole matrix
[U,S,V]=svd(X);
q=U(:,n);
figure;
plot(q'*X);
grid on;
% sure enough, we see that vector 147 is not like the others.
6. Tax policies. In this problem we explore a dynamic model of an economy, including the effects of government taxes and spending, which we assume (for simplicity) takes place at the beginning of each year. Let \( x(t) \in \mathbb{R}^n \) represent the pre-tax economic activity at the beginning of year \( t \), across \( n \) sectors, with \( x(t)_i \) being the pre-tax activity level in sector \( i \). We let \( \tilde{x}(t) \in \mathbb{R}^n \) denote the post-tax economic activity, across \( n \) sectors, at the beginning of year \( t \). We will assume that all entries of \( x(0) \) are positive, which will imply that all entries of \( x(t) \) and \( \tilde{x}(t) \) are positive, for all \( t \geq 0 \).

The pre- and post-tax activity levels are related as follows. The government taxes the sector activities at rates given by \( r \in \mathbb{R}^n \), with \( r_i \) the tax rate for sector \( i \). These rates all satisfy \( 0 \leq r_i < 1 \). The total government revenue is then \( R(t) = r^T x(t) \). This total revenue is then spent in the sectors proportionally, with \( s \in \mathbb{R}^n \) giving the spending proportions in the sectors. These spending proportions satisfy \( s_i \geq 0 \) and \( \sum_{i=1}^n s_i = 1 \); the spending in sector \( i \) is \( s_i R(t) \). The post-tax economic activity in sector \( i \), which accounts for the government taxes and spending, is then given by

\[
\tilde{x}(t)_i = x(t)_i - r_i x(t)_i + s_i R(t), \quad i = 1, \ldots, n, \quad t = 0, 1, \ldots
\]

Economic activity propagates from year to year as \( x(t+1) = E \tilde{x}(t) \), where \( E \in \mathbb{R}^{n \times n} \) is the input-output matrix of the economy. You can assume that all entries of \( E \) are positive.

We let \( S(t) = \sum_{i=1}^n x(t)_i \) denote the total economic activity in year \( t \), and we let

\[
G = \lim_{t \to \infty} \frac{S(t+1)}{S(t)}
\]

denote the (asymptotic) growth rate (assuming it exceeds one) of the economy.

(a) Explain why the growth rate does not depend on \( x(0) \) (unless it exactly satisfies a single linear equation, which we rule out as essentially impossible). Express the growth rate \( G \) in terms of the problem data \( r \), \( s \), and \( E \), using ideas from the course. You may assume that a matrix that arises in your analysis is diagonalizable and has a single dominant eigenvalue, i.e., an eigenvalue \( \lambda_1 \) that satisfies \( |\lambda_1| > |\lambda_i| \) for \( i = 2, \ldots, n \). (These assumptions aren’t actually needed—they’re just to simplify the problem for you.)

(b) Consider the problem instance with data

\[
E = \begin{bmatrix}
0.3 & 0.4 & 0.1 & 0.6 \\
0.2 & 0.7 & 0.2 & 0.1 \\
0.1 & 0.2 & 0.2 & 0.1 \\
0.4 & 0.2 & 0.3 & 0.2
\end{bmatrix}, \quad r = \begin{bmatrix}
0.45 \\
0.25 \\
0.1 \\
0.1
\end{bmatrix}, \quad s = \begin{bmatrix}
0.15 \\
0.3 \\
0.4 \\
0.15
\end{bmatrix}.
\]

Find the growth rate. Now find the growth rate with the tax rate set to zero, i.e., \( r = 0 \) (in which case \( s \) doesn’t matter). You are welcome (even, encouraged) to simulate the economic activity to double-check your answer, but we want the value using the expression found in part (a).
Solution. Tracing through the equations, we get $x(t + 1) = Ax(t)$, where

$$A = E(I - \text{diag}(r) + sr^T).$$

Alternatively, we can express $A$ by its entries as

$$A_{ij} = E_{ij}(1 - r_j) + (Es)_j r_j, \quad i, j = 1, \ldots, n.$$ 

It follows that $x(t) = A^t x(0)$.

Using our assumption that $A$ is diagonalizable, we have

$$x(t) = \sum_{i=1}^{n} \lambda_i t v_i w_i^T x(0),$$

where $\lambda_i$ are the eigenvalues of $A$, $v_i$ the associated eigenvectors, and $w_i$ the associated left eigenvectors (suitably normalized). We sort the eigenvalues so $\lambda_1$ is dominant: $|\lambda_1| > |\lambda_i|$ for $i = 2, \ldots, n$. Then assuming that $w_1^T x(0) \neq 0$, we can approximate $x(t)$ as

$$x(t) \approx \lambda_1 t v_1 (w_1^T x(0)).$$

Now we see that $\lambda_1$ must be positive; if not, then the equation above shows $x(t)$ would eventually have negative entries, which is impossible. (The same argument shows that the entries of $v_1$ must be nonnegative.) Thus we have

$$S(t) \approx \lambda_1^t (1^T v_1)(w_1^T x(0)).$$

(Note that since the entries of $v_1$ are nonnegative, we cannot have $1^T v_1 = 0$; if that were true, then $v_1 = 0$.) Thus, we have

$$S(t+1)/S(t) \approx \lambda_1.$$

So $G = \lambda_1$.

Now evaluating $G$ for the given data, we find that $G = 1.0908$ (9.08% growth) without taxes, and $G = 1.1237$ (12.37%) with taxes. So, the taxes have actually boosted the asymptotic economic growth rate! Tell that to the next person who tells you that taxes are always bad (if you know any such people).

A script for computing the growth rates is given below.

```matlab
E = [0.3 0.4 0.1 0.6;
     0.2 0.3 0.7 0.2;
     0.1 0.2 0.2 0.1;
     0.4 0.2 0.3 0.2];

r = [0.45 ; 0.25 ; 0.1; 0.1 ];
s = [0.15 ; 0.3 ; 0.4; 0.15 ];

G_no_tax = max(abs(eig(E)))
A = E*(eye(n)-diag(r)+s*r');
G_with_tax = max(abs(eig(A)))
```

21
7. Regulation using ternary inputs. Consider a discrete-time linear dynamical system

\[ x(t+1) = Ax(t) + bu(t), \quad t = 1, 2, \ldots, \]

with \( x(t) \in \mathbb{R}^n \), \( u(t) \in \{-1, 0, 1\} \), and \( b \in \mathbb{R}^n, b \neq 0 \). (The problem title comes from the restriction that the input can only take three possible values.) Our goal is to regulate the system, i.e., choose the inputs \( u(t) \) so as to drive the state \( x(t) \) towards zero. We will adopt a greedy strategy: At each time \( t \), we will choose \( u(t) \) so as to minimize \( \|x(t+1)\| \).

(a) Show that \( u(t) \) has the form \( u(t) = \text{round}(kx(t)) \), where \( k \in \mathbb{R}^{1 \times n} \), and \( \text{round}(a) \) rounds the number \( a \) to the closest of \( \{-1, 0, 1\} \), i.e.,

\[
\text{round}(a) = \begin{cases} 
1 & a > 1/2 \\
0 & |a| \leq 1/2 \\
-1 & a < -1/2.
\end{cases}
\]

(We don’t care about what happens when there are ties; we have arbitrarily broken ties in favor of \( a = 0 \).) Give an explicit expression for \( k \).

(b) Consider the specific problem with data

\[
A = \begin{bmatrix}
1 & .2 & -2 & 0 \\
-2 & 1 & 0 & .15 \\
.2 & 0 & .9 & 0 \\
0 & -.15 & 0 & 1
\end{bmatrix}, \quad b = \begin{bmatrix}
.1 \\
-.1 \\
.1 \\
-.1
\end{bmatrix}, \quad x(1) = \begin{bmatrix}
-4 \\
0 \\
0 \\
-4
\end{bmatrix}.
\]

Give \( k \), and plot \( \|x(t)\| \) and \( u(t) \) for \( t = 1, \ldots, 100 \). Use the matlab function \( \text{stairs} \) to plot \( u(t) \).

Solution:

(a) We first give the solution that is the simplest, and most straightforward. There are only three possible choices for \( u(t) \): \(-1, 0, \) and \( 1 \), which result in \( x(t+1) \) becoming \( Ax(t) - b \), \( Ax(t) \), and \( Ax(t) + b \), respectively. The greedy strategy chooses the one among these that has least norm.

Let’s work out the conditions under which we choose \( u(t) = 0 \). This occurs when we have

\[
\|Ax(t)\| \leq \|Ax(t) + b\|, \quad \|Ax(t)\| \leq \|Ax(t) - b\|,
\]

which (squaring both sides of each inequality) is equivalent to

\[
0 \leq 2b^T Ax(t) + b^T b, \quad 0 \leq -2b^T Ax(t) + b^T b.
\]

This can be written as

\[
|(1/\|b\|^2)(b^T A)x(t)| \leq 1/2.
\]
We can use a similar analysis for the case when we choose \( u(t) = 1 \): the condition is
\[
\|Ax(t) + b\| \leq \|Ax(t)\|, \quad \|Ax(t) + b\| \leq \|Ax(t) - b\|,
\]
which is equivalent to
\[
2b^TAx(t) + b^Tb \leq 0, \quad 2b^TAx(t) \leq -2b^TAx(t).
\]
The first inequality implies the second one, so the condition under which we choose \( u(t) = 1 \) is just
\[
(1/\|b\|^2)(b^TA)x(t) \leq -1/2.
\]
A similar argument shows that we choose \( u(t) = -1 \) when
\[
(1/\|b\|^2)(b^TA)x(t) \geq 1/2.
\]
We define \( k = -(1/\|b\|^2)b^TA \). Putting it all together (and noting that ties don’t matter), we have \( u(t) = \text{round}(kx(t)) \).

A variation. A number of people used the following argument. We ignore the requirement that \( u(t) \in \{-1, 0, 1\} \). Then the problem is a simple least-squares problem, with solution \( u(t) = kx(t) \). So far, this is correct.
Now comes the tricky part. If you just say ‘now let’s round \( u(t) \) to the nearest valid value’, it’s not correct. Suppose you need to solve a general least-squares problem with variable \( z \) (say), which is restricted to be in some finite set of allowed values. Then the approach above is to first solve the least-squares problem ignoring the restriction on \( z \), then round \( z \) to the closest valid value. This is wrong in general. This approach does work when the variable \( z \) is a scalar, which of course was the case in this problem. But you had to at least make an argument why you’d round, and why that’s the correct thing to do. We were generous with credit for those who made an attempt to say why rounding does the right thing (in this case anyway).

(b) We carry out this method on the problem data and obtain \( k = [-3.5 1.625 -1.75 2.875] \). The matlab code and the generated plots are given below.

\[
A = \begin{bmatrix} 1 & 0.2 & -0.2 & 0; \\
-0.2 & 1 & 0 & 0.15; \\
0.2 & 0 & 0.9 & 0; \\
0 & -0.15 & 0 & 1 \end{bmatrix};
\]
\[
b = [0.1; -0.1; 0.1; -0.1];
\]
\[
T = 100;
\]
\[
x(:,1) = [-4; 0; 0; -4];
\]
\[
\% \text{compute } k
\]

23
\[ k = -(b' \cdot A) / (b' \cdot b) \]

% Simulate the greedy policy for T time steps
for t = 1 : T
    normx(t) = norm(x(:,t));
    if (abs(k*x(:,t)) <= 1/2)
        u(t) = 0;
    end
    if (k*x(:,t) > 1/2)
        u(t) = 1;
    end
    if (k*x(:,t) < -1/2)
        u(t) = -1;
    end
    x(:,t+1) = A*x(:,t) + b*u(t);
end

% Plots
figure;
subplot(2,1,1);
plot(1:T,normx);
xlabel('t');
ylabel('||x(t)||');
subplot(2,1,2);
stairs(1:T, u(1:T));
xlabel('t');
ylabel('u(t)');
$\|x(t)\|$ vs $t$

$u(t)$ vs $t$