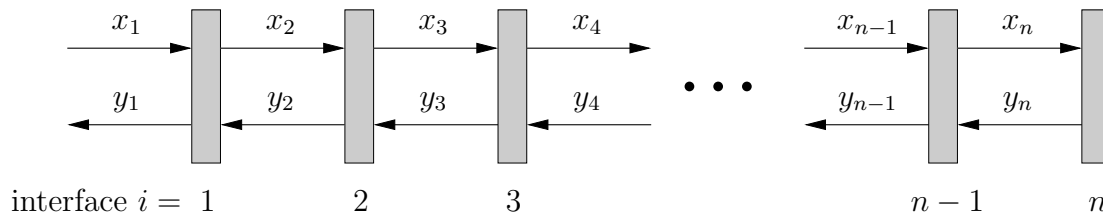


Midterm exam solutions

1. *Layered medium.* In this problem we consider a generic model for (incoherent) transmission in a layered medium. The medium is modeled as a set of n layers, separated by n dividing interfaces, shown as shaded rectangles in the figure below.



We let $x_i \in \mathbf{R}$ denote the right-traveling wave amplitude in layer i , and we let $y_i \in \mathbf{R}$ denote the left-traveling wave amplitude in layer i , for $i = 1, \dots, n$. The right-traveling wave in the first layer is called the incident wave, and the left-traveling wave in the first layer is called the reflected wave. The scattering coefficient for the medium is defined as the ratio $S = y_1/x_1$ (assuming $x_1 \neq 0$).

The right- and left-traveling waves on each side of an interface are related by transmission and reflection. The right-traveling wave of amplitude x_i contributes the amplitude $t_i x_i$ to x_{i+1} , where $t_i \in [0, 1]$ is the transmission coefficient of the i th interface. It also contributes the amplitude $r_i x_i$ to y_i , the left-traveling wave, where $r_i \in [0, 1]$ is the reflection coefficient of the i th interface. We will assume that the interfaces are symmetric, so the left-traveling wave with amplitude y_{i+1} contributes the wave amplitude $t_i y_{i+1}$ to y_i (via transmission) and wave amplitude $r_i y_{i+1}$ to x_{i+1} (via reflection). Thus we have

$$x_{i+1} = t_i x_i + r_i y_{i+1}, \quad y_i = r_i x_i + t_i y_{i+1}, \quad i = 1, 2, \dots, n-1.$$

We model the last interface as totally reflective, which means that $y_n = x_n$.

- (a) Explain how to find the scattering coefficient S , given the transmission and reflection coefficients for the first $n-1$ layers.
- (b) Carry out your method for a medium with $n = 20$ layers, and $t_i = 0.96$, $r_i = 0.02$ for $i = 1, \dots, n-1$. Plot the left- and right-traveling wave amplitudes x_i, y_i versus i , and report the value of S you find.

Hint: You may find the matlab function `diag(x,k)` useful.

- (c) *Fault location.* A fault in interface k results in a reversal: $t_k = 0.02$, $r_k = 0.96$, with all other interfaces having their nominal values $t_i = 0.96$, $r_i = 0.02$. You measure the scattering coefficient $S = S^{\text{fault}}$ with the fault (but you don't have access to the left- or right-traveling waves with the faulted interface). Explain how to find which interface is faulted. Carry out your method with $S^{\text{fault}} = 0.70$. You may assume that the last (fully reflective) interface is not faulty. Be sure to give the value of k that is most consistent with the measurement.

Solution.

- (a) The equations are homogeneous in x and y , so we can find the scattering coefficient by fixing the incident wave amplitude as $x_1 = 1$; the scattering coefficient is then $S = y_1/x_1 = y_1$. The transmission and reflection conditions lead to the following set of $2n$ (homogeneous) equations:

$$\begin{aligned}
 x_1 &= 1 \\
 x_2 &= t_1 x_1 + r_1 y_2 \\
 x_3 &= t_2 x_2 + r_2 y_3 \\
 &\vdots \\
 x_{n-1} &= t_{n-2} x_{n-2} + r_{n-2} y_{n-1} \\
 x_n &= t_{n-1} x_{n-1} + r_{n-1} y_n \\
 y_1 &= r_1 x_1 + t_1 y_2 \\
 y_2 &= r_2 x_2 + t_2 y_3 \\
 y_3 &= r_3 x_3 + t_3 y_4 \\
 &\vdots \\
 y_{n-1} &= r_{n-1} x_{n-1} + t_{n-1} y_n \\
 y_n &= x_n
 \end{aligned}$$

for the $2n$ unknowns $x_1, \dots, x_n, y_1, \dots, y_n$. These can be rewritten in block matrix form as

$$\underbrace{\begin{bmatrix} A & B \\ C & D \end{bmatrix}}_F \underbrace{\begin{bmatrix} x \\ y \end{bmatrix}}_z = \underbrace{\begin{bmatrix} e_1 \\ 0 \end{bmatrix}}_b,$$

where $e_1, 0 \in \mathbf{R}^n$ and $A, B, C, D \in \mathbf{R}^{n \times n}$ are defined as

$$A = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 0 \\ t_1 & -1 & 0 & \cdots & 0 & 0 \\ 0 & t_2 & -1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & -1 & 0 \\ 0 & 0 & 0 & \cdots & t_{n-1} & -1 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & r_1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & r_1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & r_{n-2} & 0 \\ 0 & 0 & 0 & \cdots & 0 & r_{n-1} \end{bmatrix},$$

$$C = \begin{bmatrix} r_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & r_2 & 0 & \cdots & 0 & 0 \\ 0 & 0 & r_3 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & r_{n-1} & 0 \\ 0 & 0 & 0 & \cdots & 0 & 1 \end{bmatrix}, \quad D = \begin{bmatrix} -1 & t_1 & 0 & \cdots & 0 & 0 \\ 0 & -1 & t_2 & \cdots & 0 & 0 \\ 0 & 0 & -1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & -1 & t_{n-1} \\ 0 & 0 & 0 & \cdots & 0 & -1 \end{bmatrix}.$$

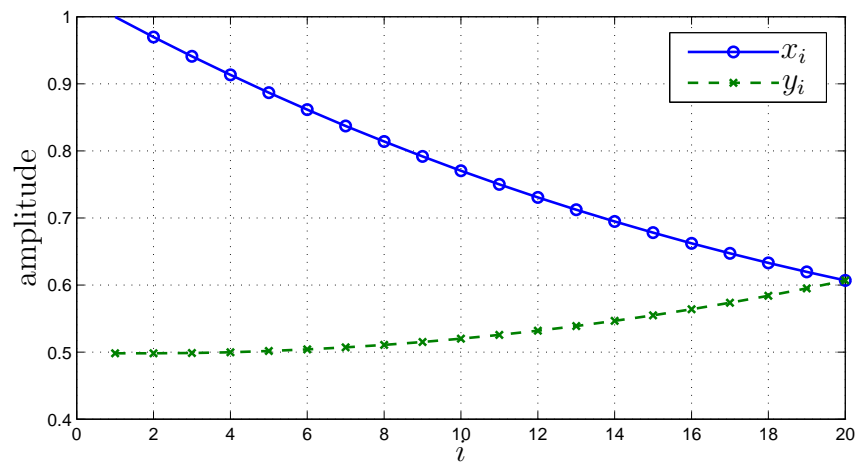
To find the scattering coefficient, we solve the system $Fz = b$ for $z \in \mathbf{R}^{2n}$, and let $S = z_{n+1}$.

(b) The following code solves the system and predicts $S = 0.4982$.

```
n = 20;
t = 0.96*ones(n-1,1);
r = 0.02*ones(n-1,1);

A = diag([1; -ones(n-1,1)]) + diag(t, -1);
B = diag([0; r]);
C = diag([r; 1]);
D = diag(-ones(n,1)) + diag(t, 1);
b = [1; zeros(2*n-1,1)];
F = [A B; C D];
z = F \ b;
S = z(n+1);

plot(1:n, z(1:n), 'o-', 1:n, z(n+1:end), 'x--');
xlabel('i'); ylabel('amplitude');
legend('xi', 'yi');
grid on;
```



(c) To figure out which interface is faulty, we find S_k , the reflection coefficient resulting

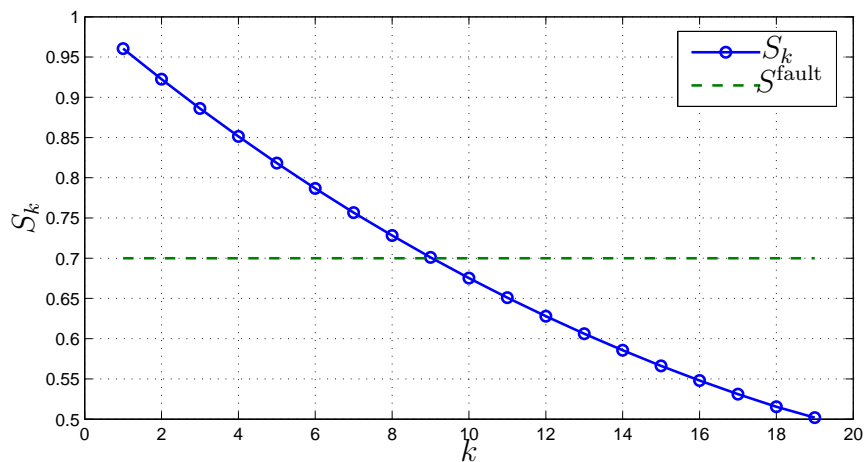
from setting $t_k = 0.02$, $r_k = 0.96$ for each $k = 1, \dots, n - 1$. The following code computes and plots S_k for each k . From the graph, interface $k = 9$ is most consistent with the measurement $S^{\text{fault}} = 0.70$.

```
n = 20;
Sfault = 0.70;
S = zeros(n-1,1);

for k=1:n-1
    t = 0.96*ones(n-1,1);
    r = 0.02*ones(n-1,1);
    t(k) = 0.02;
    r(k) = 0.96;

    A = diag([1; -ones(n-1,1)]) + diag(t, -1);
    B = diag([0; r]);
    C = diag([r; 1]);
    D = diag(-ones(n,1)) + diag(t, 1);
    b = [1; zeros(2*n-1,1)];
    F = [A B; C D];
    z = F \ b;
    S(k) = z(n+1);
end

plot(1:n-1, S, 'o-', 1:n-1, Sfault*ones(n-1,1), '--');
xlabel('k'); ylabel('Sk');
legend('Sk', 'Sfault');
grid on;
```



Alternative solution. Some people found an equivalent recursive way to compute

the left- and right- traveling wave amplitudes, for which we gave full credit. From the given homogeneous relations among the x_i and y_i , we rewrite the x_i and y_i as a function of x_{i+1} and y_{i+1} .

$$\begin{aligned} x_{i+1} &= t_i x_i + r_i y_{i+1} \\ y_i &= r_i x_i + t_i y_{i+1}, \quad i = 1, \dots, n-1. \end{aligned}$$

Solving the first equation for x_i and substituting into the second equation, gives the following recursion:

$$\begin{bmatrix} x_i \\ y_i \end{bmatrix} = \underbrace{\begin{bmatrix} \frac{1}{t_i} & -\frac{r_i}{t_i} \\ \frac{r_i}{t_i} & \frac{t_i^2 - r_i^2}{t_i} \end{bmatrix}}_{A_i} \begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix}, \quad i = 1, \dots, n-1,$$

where we assume that $t_i \neq 0$, $i = 1, \dots, n-1$ and the arbitrary initial condition $x_n = y_n = 1$. Expanding the recursion gives

$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = A_1 A_2 \cdots A_{n-1} \begin{bmatrix} 1 \\ 1 \end{bmatrix},$$

from where we get $S = y_1/x_1$. Finally, to find the faulted interface, we replace A_k by A^{fault} , where

$$A^{\text{fault}} = \begin{bmatrix} \frac{1}{(0.02)} & -\frac{(0.96)}{(0.02)} \\ \frac{(0.96)}{(0.02)} & \frac{(0.02)^2 - (0.96)^2}{(0.02)} \end{bmatrix},$$

and compute the corresponding reflection coefficient S_k . The value of k for which the corresponding $|S_k - S^{\text{fault}}|$ has the smallest value is the faulted interface.

2. *State trajectory estimation.* We consider a discrete-time linear dynamical system

$$x(t+1) = Ax(t) + Bu(t) + w(t), \quad y(t) = Cx(t) + v(t), \quad t = 1, 2, \dots,$$

with state $x(t) \in \mathbf{R}^n$, input $u(t) \in \mathbf{R}^m$ and output $y(t) \in \mathbf{R}^p$. The signal $w(t)$ is called the process noise, and the signal $v(t)$ is the measurement noise. You know the matrices A , B , C , the inputs $u(t)$, $t = 1, 2, \dots, T-1$, and outputs $y(t)$, $t = 1, 2, \dots, T$. You do not know $x(t)$, $w(t)$, or $v(t)$. Your job is to estimate the state trajectory $x(t)$, for $t = 1, \dots, T$. We will denote your estimate of the state trajectory as $\hat{x}(t)$, $t = 1, \dots, T$. When we guess the state trajectory $\hat{x}(t)$, $t = 1, \dots, T$, we have two sets of residuals,

$$\hat{x}(t+1) - (A\hat{x}(t) + Bu(t)), \quad t = 1, \dots, T-1, \quad y(t) - C\hat{x}(t), \quad t = 1, \dots, T,$$

which correspond to (implicit) estimates of $w(t)$ and $v(t)$, respectively.

You will choose $\hat{x}(t)$ so as to minimize the overall objective

$$J = \sum_{t=1}^{T-1} \|\hat{x}(t+1) - (A\hat{x}(t) + Bu(t))\|^2 + \rho \sum_{t=1}^T \|y(t) - C\hat{x}(t)\|^2,$$

where $\rho > 0$ is a given parameter (related to our guess of the relative sizes of $w(t)$ and $v(t)$). The objective J is a weighted sum of norm squares of our two residuals.

- Explain how to find the state trajectory estimate $\hat{x}(t)$, $t = 1, \dots, T$, using any concepts from the course. If one or more matrices must satisfy a rank condition for your method to work, say so.
- Carry out your method from part (a) using `state_traj_estim_data.m`, which gives A , B , C , the dimensions n , m , p , the parameter ρ , and the time horizon T . The input and output trajectories are given as $m \times T$ and $p \times T$ matrices, respectively. (The t th column gives the vector at the t th period.)

Give the value of J corresponding to your estimate.

The mfile includes the true value of the state trajectory, $x(t)$, (of course you may not use it in forming your estimate $\hat{x}(t)$). Plot $x_1(t)$ (the true first state component) and $\hat{x}_1(t)$ (the estimated first state component) on the same plot.

Matlab hints:

- The matlab command `x = X(:)`, where X is an n by m matrix, stacks the columns of X into a vector of dimension nm . You may then recover X with the command `X = reshape(x,n,m)`.
- You might find the matlab function `blkdiag` useful.

Solution.

- By stacking up $\hat{x}(t)$, $t = 1, \dots, T$ into a single vector $\hat{x} = (\hat{x}(1), \dots, \hat{x}(T)) \in \mathbf{R}^{nT}$ we can express J as

$$J = \|P\hat{x} + q\|^2,$$

where

$$P = \begin{bmatrix} -A & I & 0 & \cdots & 0 \\ 0 & -A & I & \cdots & 0 \\ \vdots & & \ddots & \ddots & \\ 0 & 0 & \cdots & -A & I \\ -\rho^{1/2}C & 0 & 0 & \cdots & 0 \\ 0 & -\rho^{1/2}C & 0 & \cdots & 0 \\ 0 & 0 & -\rho^{1/2}C & \cdots & 0 \\ \vdots & & & \ddots & \\ 0 & 0 & 0 & \cdots & -\rho^{1/2}C \end{bmatrix}, \quad q = \begin{bmatrix} -Bu(1) \\ -Bu(2) \\ \vdots \\ -Bu(T-1) \\ \rho^{1/2}y(1) \\ \rho^{1/2}y(3) \\ \rho^{1/2}y(3) \\ \vdots \\ \rho^{1/2}y(T) \end{bmatrix}.$$

In the general case, we can take $\hat{x} = -P^\dagger q$, which minimizes the objective even when there are multiple values of \hat{x} that minimize the objective. If P is skinny and full rank, there is a unique solution, given by $\hat{x}_{ls} = -(P^T P)^{-1} P^T q$.

- (b) The following matlab code implements the method and plots the true and estimated state trajectories for $t = 1, \dots, T$. The minimum value of J is found to be $J = 337.6$.

```
state_traj_estim_data;

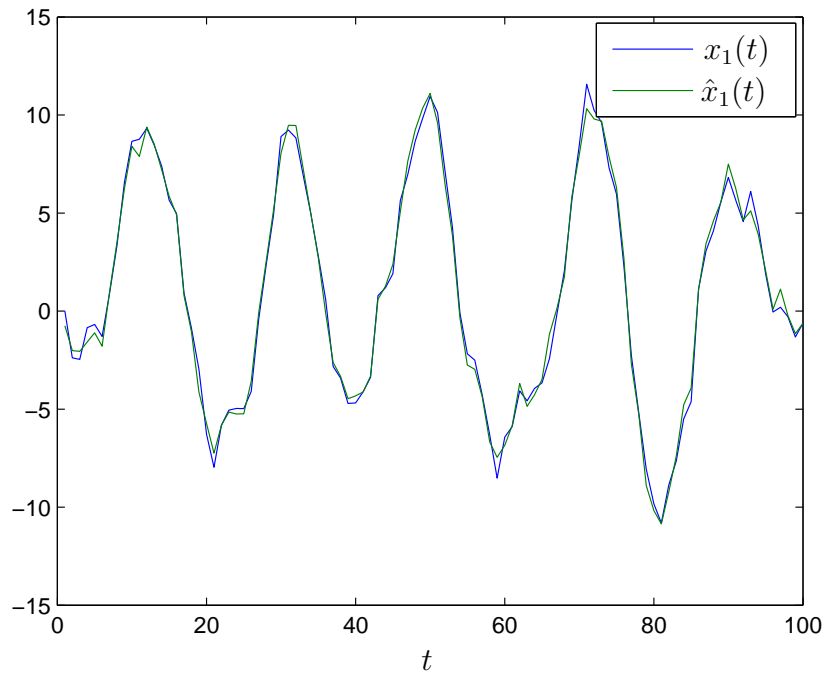
%Construct matrix P
diagA = [];
Plow = -sqrt(rho)*C;
for i=2:T
    diagA = blkdiag(diagA, -A);
    Plow = blkdiag(Plow, -sqrt(rho)*C);
end
Ptop = [diagA zeros((T-1)*n,n)] + [zeros((T-1)*n,n) eye((T-1)*n)];
P = [Ptop;Plow];

%Construct vector q
aux = -B*u;
q = [aux(:) ; sqrt(rho)*y(:)];

%Find LS approximation
xhat = -P\q;
xhat = reshape(xhat, n, T);

%Plot first component of output and error
plot(1:T, x_true(1,:) , 1:T, xhat(1,:));
legend('xtrue(t)', 'hatx(t)');
xlabel('t');

J = norm(P*xhat(:) + q)^2
```



3. *Reverse engineering a smoothing filter.* A smoothing filter takes an input vector $u \in \mathbf{R}^n$ and produces an output vector $y \in \mathbf{R}^n$. (We will assume that $n \geq 3$.) The output y is obtained as the minimizer of the objective

$$J = J^{\text{track}} + \lambda J^{\text{norm}} + \mu J^{\text{cont}} + \kappa J^{\text{smooth}},$$

where λ , μ , and κ are positive constants (weights), and

$$J^{\text{track}} = \sum_{i=1}^n (u_i - y_i)^2, \quad J^{\text{norm}} = \sum_{i=1}^n y_i^2$$

are the tracking error and norm-squared of y , respectively, and

$$J^{\text{cont}} = \sum_{i=2}^n (y_i - y_{i-1})^2, \quad J^{\text{smooth}} = \sum_{i=2}^{n-1} (y_{i+1} - 2y_i + y_{i-1})^2$$

are measures of the continuity and smoothness of y , respectively.

Here is the problem: You have access to one input-output pair, *i.e.*, an input u , and the associated output y . Your goal is to find the weights λ , μ , and κ . In other words, you will reverse engineer the smoothing filter, working from an input-output pair.

- (a) Explain how to find λ , μ , and κ . (You do not need to worry about ensuring that these are positive; you can assume this will occur automatically.)
- (b) Carry out your method on the data found in `rev_eng_smooth_data.m`. Give the values of the weights.

Solution.

We first define matrices $D_1 \in \mathbf{R}^{(n-1) \times n}$ and $D_2 \in \mathbf{R}^{(n-2) \times n}$ as

$$D_1 = \begin{bmatrix} -1 & 1 & 0 & \cdots & 0 \\ 0 & -1 & 1 & \cdots & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & \cdots & 0 & -1 & 1 \end{bmatrix}, \quad D_2 = \begin{bmatrix} 1 & -2 & 1 & 0 & \cdots & 0 \\ 0 & 1 & -2 & 1 & \cdots & 0 \\ \vdots & & & \ddots & & \vdots \\ 0 & \cdots & 0 & 1 & -2 & 1 \end{bmatrix}.$$

Multiplying a vector by these matrices gives the first and second differences:

$$(D_1 y)_i = y_{i+1} - y_i, \quad (D_2 y)_i = y_{i+2} - 2y_{i+1} + y_i.$$

Now we can write the objective components in the compact form

$$J^{\text{track}} = \|y - u\|^2, \quad J^{\text{norm}} = \|y\|^2, \quad J^{\text{cont}} = \|D_1 y\|^2, \quad J^{\text{smooth}} = \|D_2 y\|^2.$$

Since y minimizes J , the gradient of J with respect to y must be zero, *i.e.*,

$$\nabla_y J = 2(y - u) + 2\lambda y + 2\mu D_1^T D_1 y + 2\kappa D_2^T D_2 y = 0.$$

We are interested in finding λ , μ , and κ , so we re-write these equations as a set of linear equations involving λ , μ , and κ :

$$\begin{bmatrix} y & D_1^T D_1 y & D_2^T D_2 y \end{bmatrix} \begin{bmatrix} \lambda \\ \mu \\ \kappa \end{bmatrix} = u - y.$$

This is a set of n equations in the three unknowns λ , μ , and κ . For $n > 3$ this is a set of overdetermined linear equations; but we know that the equations are solvable.

We didn't require you to do so, but we can say when the matrix above is full rank (in which case, we can recover λ , μ , and κ exactly). It is full rank precisely when y does not satisfy a two-term recursion, *i.e.*, $y_{t+1} = \alpha y_t + \beta y_{t-1}$ for some $\alpha, \beta \in \mathbf{R}$. It was OK with us for you to simply assume the matrix is full rank, or to check that for the given y , it is full rank.

We can calculate λ , μ , κ as

$$\begin{bmatrix} \lambda \\ \mu \\ \kappa \end{bmatrix} = \begin{bmatrix} y & D_1^T D_1 y & D_2^T D_2 y \end{bmatrix}^\dagger (u - y).$$

(It's good practice to check that the computed (λ, μ, κ) do indeed satisfy the equations above, as they must.)

The following matlab script implements the solution.

```

rev_eng_smooth_data;

% construct the first difference operator, D1
D1 = zeros(n-1,n);
for i = 2 : n
    D1(i-1,i-1) = -1;
    D1(i-1,i) = 1;
end

% construct the second difference operator, D2
D2 = zeros(n-2,n);
for i = 2:n-1
    D2(i-1,i-1) = 1;
    D2(i-1,i) = -2;
    D2(i-1,i+1) = 1;
end

% solve for the weights (i.e., lambda, mu, kappa)
weights = ([y D1'*D1*y D2'*D2*y])\ (u-y);

% let's check that the equations are satisfied exactly (up to numerical errors)
norm([y D1'*D1*y D2'*D2*y]*weights-(u-y))

lambda = weights(1)
mu = weights(2)
kappa = weights(3)

```

We find that $\lambda = 0.1$, $\mu = 2$ and $\kappa = 10$. (The script verifies that the overdetermined equations are indeed satisfied.)

4. *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, \dots, M_m . There are n (reversible) reactions going on, labeled R_1, \dots, R_n , with reaction rates $v_1, \dots, v_n \in \mathbf{R}$. 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 $M_1 \rightarrow M_2 + 2M_3$. 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, \dots, M_m .) This corresponds to a first column of S of the form $(-1, 1, 2, 0, \dots, 0)$.

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 $(1, 0, \dots, 0)$.

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 $1, \dots, n - 1$ 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. 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.

Remark. This is a very simple version of the problem. In EE364a, you'll see more sophisticated versions of the same problem, that incorporate lower and upper limits on reactions rates and other realistic constraints.

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

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

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

So to find out which genes are essential, we check this condition for $k = 1, \dots, n - 1$. 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
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, \dots, 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 n th entry of each basis element a_k is 0.

5. *Portfolio selection with sector neutrality constraints.* We consider the problem of selecting a portfolio composed of n assets. We let $x_i \in \mathbf{R}$ denote the investment (say, in dollars) in asset i , with $x_i < 0$ meaning that we hold a short position in asset i . We normalize our total portfolio as $\mathbf{1}^T x = 1$, where $\mathbf{1}$ is the vector with all entries 1. (With normalization, the x_i are sometimes called *portfolio weights*.)

The portfolio (mean) return is given by $r = \mu^T x$, where $\mu \in \mathbf{R}^n$ is a vector of asset (mean) returns. We want to choose x so that r is large, while avoiding risk exposure, which we explain next.

First we explain the idea of *sector exposure*. We have a list of k economic sectors (such as manufacturing, energy, transportation, defense, ...). A matrix $F \in \mathbf{R}^{k \times n}$, called the *factor loading matrix*, relates the portfolio x to the *factor exposures*, given as $R^{\text{fact}} = Fx \in \mathbf{R}^k$. The number R_i^{fact} is the portfolio risk exposure to the i th economic sector. If R_i^{fact} is large (in magnitude) our portfolio is exposed to risk from changes in that sector; if it is small, we are less exposed to risk from that sector. If $R_i^{\text{fact}} = 0$, we say that the portfolio is *neutral* with respect to sector i .

Another type of risk exposure is due to fluctuations in the returns of the individual assets. The *idiosyncratic risk* is given by

$$R^{\text{id}} = \sum_{i=1}^n \sigma_i^2 x_i^2,$$

where $\sigma_i > 0$ are the standard deviations of the asset returns. (You can take the formula above as a definition; you do not need to understand the statistical interpretation.)

We will choose the portfolio weights x so as to maximize $r - \lambda R^{\text{id}}$, which is called the *risk-adjusted return*, subject to neutrality with respect to all sectors, *i.e.*, $R^{\text{fact}} = 0$. Of course we also have the normalization constraint $\mathbf{1}^T x = 1$. The parameter λ , which is positive, is called the *risk aversion parameter*. The (known) data in this problem are $\mu \in \mathbf{R}^n$, $F \in \mathbf{R}^{k \times n}$, $\sigma = (\sigma_1, \dots, \sigma_n) \in \mathbf{R}^n$, and $\lambda \in \mathbf{R}$.

- (a) Explain how to find x , using methods from the course. You are welcome (even encouraged) to express your solution in terms of block matrices, formed from the given data.
- (b) Using the data given in `sector_neutral_portfolio_data.m`, find the optimal portfolio. Report the associated values of r (the return), and R^{id} (the idiosyncratic risk). Verify that $\mathbf{1}^T x = 1$ (or very close) and $R^{\text{fact}} = 0$ (or very small).

Solution.

- (a) We define $\Sigma \in \mathbf{R}^{n \times n}$ to be a diagonal matrix with $\Sigma_{ii} = \sigma_i^2$, so $R^{\text{id}} = x^T \Sigma x$. The problem we are trying to solve is

$$\begin{aligned} & \text{maximize} && \mu^T x - \lambda x^T \Sigma x \\ & \text{subject to} && \mathbf{1}^T x = 1, \quad Fx = 0 \end{aligned}$$

with variable $x \in \mathbf{R}^n$. Maximizing an objective is equivalent to minimizing the negative of the objective, so we can rewrite this as

$$\begin{aligned} & \text{minimize} && -\mu^T x + \lambda x^T \Sigma x \\ & \text{subject to} && \mathbf{1}^T x = 1, \quad Fx = 0. \end{aligned}$$

We introduce Lagrange multipliers $\kappa \in \mathbf{R}$ and $\nu \in \mathbf{R}^k$ for the two constraints, and write the Lagrangian of this problem as

$$L(x, \nu, \kappa) = -\mu^T x + \lambda x^T \Sigma x + \nu^T (Fx) + \kappa(\mathbf{1}^T x - 1).$$

The optimality conditions are then given by

$$\nabla_x L = -\mu + 2\lambda \Sigma x + F^T \nu + \kappa \mathbf{1} = 0, \quad \nabla_\nu L = Fx = 0, \quad \nabla_\kappa L = \mathbf{1}^T x - 1 = 0,$$

which we can write in block matrix form as

$$\begin{bmatrix} 2\lambda \Sigma & F^T & \mathbf{1} \\ F & 0 & 0 \\ \mathbf{1}^T & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \nu \\ \kappa \end{bmatrix} = \begin{bmatrix} \mu \\ 0 \\ 1 \end{bmatrix}.$$

To find the optimal x we solve this set of $n + k + 1$ linear equations in $n + k + 1$ variables.

- (b) Using the data provided we arrive at optimal values $r = 26.71$, and $R^{\text{id}} = 133.6$, with the optimal objective value of the original problem being 13.3. The following matlab code implements the method of part (a).

```
sector_neutral_portfolio_data

% construct the relevant matrices from the data
M=[2*lambda*diag(sigmas.^2) F' ones(n,1);
```

```

F = zeros(k,k) zeros(k,1);
ones(1,n) zeros(1,k) 0];
v=[mu;zeros(k,1);1];

% this solves the set of linear equations M*all_vars = v
all_vars = M\v;

% extract the portfolio weights
x=all_vars(1:n)
% plug in to find the optimal objective value
objective = -(-mu'*x + lambda*x'*diag(sigmas.^2)*x)
r = mu'*x
R_id = x'*diag(sigmas.^2)*x

```

6. *Memory of a linear time-invariant system.* An input signal (sequence) $u_t \in \mathbf{R}$, $t \in \mathbf{Z}$ (i.e., $t = \dots, -1, 0, 1, \dots$) and output signal $y_t \in \mathbf{R}$, $t \in \mathbf{Z}$, are related by a convolution operator

$$y_t = \sum_{\tau=1}^M h_{\tau} u_{t-\tau}, \quad t \in \mathbf{Z},$$

where $h = (h_1, \dots, h_M)$ are the *impulse response coefficients* of the convolution system. (Convolution systems are also called linear time-invariant systems.) When $h_M \neq 0$, the integer M is called the *memory* of the system.

Now for the problem. You are given the input and output signal values over a time interval $t = 1, \dots, T$:

$$(u_1, \dots, u_T), \quad (y_1, \dots, y_T).$$

The goal is to find the smallest memory M consistent with this data. Note that *you do not know* u_{τ} or y_{τ} for $\tau \leq 0$ or $\tau > T$, and of course, you do not know h .

- Explain how to solve this problem, using any concepts from the course. You may assume that $T > 2M$.
- Use your method from part (a) on the data found in `mem_lti_data.m`. Give the value of M found. *Hint.* You may find the matlab functions `toeplitz` and `fliplr` useful.

Solution. Suppose the memory is M . Then y_1, \dots, y_M depend on values of u_t that we do not know, i.e., $t \leq 0$. These unknown values of u_t can cause y_1, \dots, y_M to take on any values at all (assuming $h \neq 0$); this means that we cannot use these values to find h . Thus, we must discard the output values y_1, \dots, y_M ; we will only use (y_{M+1}, \dots, y_T) . We rearrange the convolution equations $y_t = \sum_{\tau=1}^M h_{\tau} u_{t-\tau}$ for $t = M+1, \dots, T$ as a

set of $T - M$ linear equations in the M coefficients h :

$$\begin{bmatrix} y_{M+1} \\ y_{M+2} \\ y_{M+3} \\ \vdots \\ y_T \end{bmatrix} = \begin{bmatrix} u_M & u_{M-1} & u_{M-2} & \cdots & u_1 \\ u_{M+1} & u_M & u_{M-1} & \cdots & u_2 \\ u_{M+2} & u_{M+1} & u_M & \cdots & u_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ u_{T-1} & u_{T-2} & u_{T-3} & \cdots & u_{T-M} \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ \vdots \\ h_M \end{bmatrix}.$$

These equations must have a solution for the data to be consistent with memory M . So we simply increase M until the equations above have a solution. One good way to do this is to use the pseudo-inverse, and then check if the h found solves the equations above (to numerical precision).

The matlab script below carries this out.

```
mem_lti_data;

% Loop over M from 1 to floor(T/2)
for M = 1:floor(T/2)
    ytilde = y(M+1:T);
    A = toeplitz(u(M:T-1),fliplr(u(1:M)'));
    % check if M is feasible
    x = pinv(A)*ytilde;
    if(norm(A*x-ytilde) <= 10^(-5)*norm(ytilde))
        display(['M = ' num2str(M)]);
        break;
    end
end
```

Running the code, we find that the input-output data is consistent with a memory $M = 7$ (but not consistent with any smaller memory).