1. A simple power control algorithm for a wireless network. First some background.

We consider a network of \( n \) transmitter/receiver pairs. Transmitter \( i \) transmits at power level \( p_i \) (which is positive). The path gain from transmitter \( j \) to receiver \( i \) is \( G_{ij} \) (which are all nonnegative, and \( G_{ii} \) are positive). The signal power at receiver \( i \) is given by \( s_i = G_{ii}p_i \). The noise plus interference power at receiver \( i \) is given by

\[
q_i = \sigma^2 + \sum_{j \neq i} G_{ij}p_j
\]

where \( \sigma^2 > 0 \) is the self-noise power of the receivers (assumed to be the same for all receivers).

The signal to interference plus noise ratio (SINR) at receiver \( i \) is defined as \( S_i = s_i/q_i \). For signal reception to occur, the SINR must exceed some threshold value \( \gamma \) (which is often in the range \( 3 - 10 \)). Various power control algorithms are used to adjust the powers \( p_i \) to ensure that \( S_i \geq \gamma \) (so that each receiver can receive the signal transmitted by its associated transmitter). In this problem, we consider a simple power control update algorithm. The powers are all updated synchronously at a fixed time interval, denoted by \( t = 0, 1, 2, \ldots \). Thus the quantities \( p, q, \) and \( S \) are discrete-time signals, so for example \( p_3(5) \) denotes the transmit power of transmitter 3 at time epoch \( t = 5 \). What we’d like is

\[
S_i(t) = s_i(t)/q_i(t) = \alpha\gamma,
\]

where \( \alpha > 1 \) is an SINR safety margin (of, for example, one or two dB). Note that increasing \( p_i(t) \) (power of the \( i \)th transmitter) increases \( S_i \) but decreases all other \( S_j \). A very simple power update algorithm is given by

\[
p_i(t + 1) = p_i(t)(\alpha\gamma/S_i(t)). \tag{1}
\]

This scales the power at the next time step to be the power that would achieve \( S_i = \alpha\gamma \), if the interference plus noise term were to stay the same. But unfortunately, changing the transmit powers also changes the interference powers, so it’s not that simple! Finally, we get to the problem.

a) Show that the power control algorithm can be expressed as a linear dynamical system with constant input, i.e., in the form

\[
p(t + 1) = Ap(t) + b,
\]

where \( A \in \mathbb{R}^{n \times n} \) and \( b \in \mathbb{R}^n \) are constant. Describe \( A \) and \( b \) explicitly in terms of \( \sigma, \gamma, \alpha \) and the components of \( G \).
b) simulation Simulate the power control algorithm, starting from various initial (positive) power levels. Use the problem data

\[
G = \begin{bmatrix} 1 & 0.2 & 0.1 \\ 0.1 & 2 & 0.1 \\ 0.3 & 1 & 3 \end{bmatrix}, \quad \gamma = 3, \quad \alpha = 1.2, \quad \sigma = 0.1.
\]

Plot \( S_i \) and \( p \) as a function of \( t \), and compare it to the target value \( \alpha \gamma \). Repeat for \( \gamma = 5 \). Comment briefly on what you observe. Comment: You’ll understand what you see later in the course.

Solution.

a) The power update rule for a single transmitter can be found by manipulating the definitions given in the problem.

\[
p_i(t+1) = \frac{\alpha \gamma p_i(t)}{S_i(t)} = \frac{\alpha \gamma p_i(t) S_i(t)}{p_i(t)} = \frac{\alpha \gamma p_i(t) \left[ \sigma^2 + \sum_{j \neq i} G_{ij} p_j(t) \right]}{G_{ii} p_i(t)}
\]

In matrix form the equations look like this:

\[
\begin{bmatrix}
p_1(t+1) \\
p_2(t+1) \\
p_3(t+1) \\
... \\
p_n(t+1)
\end{bmatrix} =
\begin{bmatrix}
0 & \frac{\alpha \gamma G_{12}}{G_{11}} & \frac{\alpha \gamma G_{13}}{G_{11}} & \cdots & \frac{\alpha \gamma G_{1n}}{G_{11}} \\
\frac{\alpha \gamma G_{21}}{G_{22}} & 0 & \frac{\alpha \gamma G_{23}}{G_{22}} & \cdots & \frac{\alpha \gamma G_{2n}}{G_{22}} \\
\frac{\alpha \gamma G_{31}}{G_{33}} & \frac{\alpha \gamma G_{32}}{G_{33}} & 0 & \cdots & \frac{\alpha \gamma G_{3n}}{G_{33}} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
\frac{\alpha \gamma G_{n1}}{G_{nn}} & \frac{\alpha \gamma G_{n2}}{G_{nn}} & \frac{\alpha \gamma G_{n3}}{G_{nn}} & \cdots & 0
\end{bmatrix}
\begin{bmatrix}
p_1(t) \\
p_2(t) \\
p_3(t) \\
... \\
p_n(t)
\end{bmatrix} +
\begin{bmatrix}
\frac{\alpha \gamma \sigma^2}{G_{11}} \\
\frac{\alpha \gamma \sigma^2}{G_{22}} \\
\frac{\alpha \gamma \sigma^2}{G_{33}} \\
\vdots \\
\frac{\alpha \gamma \sigma^2}{G_{nn}}
\end{bmatrix}.
\]

b) The following matlab code simulates the system for \( \gamma = 3 \) and an initial power of 0.1 for each transmitter.

```matlab
clear all; close all; 
G = [1 .2 .1; .1 2 .1; .3 .1 3];
% Gain matrix
gamma = 3;
% minimum SINR
alpha = 1.2;
% safety margin
sigma = 0.01;
% Noise power (same for all receivers)
A = zeros(3,3); for i = 1:3 
    for j = 1:3
        if (i ~= j)
            A(i,j) = alpha*gamma*G(i,j)/G(i,i);
        end
    end
end
```

2
\[ b = \text{zeros}(3,1); \text{for } i = 1:3 \]
\[ b(i) = \alpha \gamma \sigma / G(i,i); \]
end
num_iterations = 20;
p_i = [.1; .1; .1]; % Initialized to p(0)
S = \[ G(1,1) p_i(1) / (\sigma + G(1,2) p_i(2) + G(1,3) p_i(3)); G(2,2) p_i(2) / (\sigma + G(2,1) p_i(1) + G(2,3) p_i(3)); G(3,3) p_i(3) / (\sigma + G(3,1) p_i(1) + G(3,2) p_i(2)) \];
p = p_i; % matrix to store the powers versus time
for i = 1:num_iterations
p_i = A*p_i+b;
p = [p p_i]; % Find the new powers and save
SINR_current = \[ G(1,1) p_i(1) / (\sigma + G(1,2) p_i(2) + G(1,3) p_i(3)); G(2,2) p_i(2) / (\sigma + G(2,1) p_i(1) + G(2,3) p_i(3)); G(3,3) p_i(3) / (\sigma + G(3,1) p_i(1) + G(3,2) p_i(2)) \];
S = [S SINR_current];
end
figure(1); temp = 0:num_iterations; subplot(2,1,1);
plot(temp,p(:,1),"--", temp,p(:,2),"-",temp,p(:,3),"-.");
xlabel('Iteration Number'); ylabel('Transmitter Power');
title('\gamma = 3, intial powers = [.1; .1; .1]');
legend('Transmitter 1', 'Transmitter 2', 'Transmitter 3',0); grid;
subplot(2,1,2); plot(temp,S(:,1),"--", temp,S(:,2),"-",temp,S(:,3),"-." );
xlabel('Iteration Number'); ylabe('SINR'); title('\gamma = 3, intial powers = [.1; .1; .1]');
legend('Transmitter 1', 'Transmitter 2', 'Transmitter 3',0); grid;

The figure below shows the SINR and transmitter power as a function of iteration num-
Similar matlab code can be used to try other initial transmitter powers. For example, the simulation shown below used initial transmitter powers of .1, .01, and .02 for the first, second, and third transmitter respectively. In both cases, the final transmitter powers approach .083, .041, and .047. The SINR approaches $3.6 = \alpha \gamma$. The algorithm
appears to work.

\[ \gamma = 3, \text{ initial powers } = [0.1; 0.01; 0.02] \]

Testing the system for \( \gamma = 5 \) and the same initial conditions (see graphs below) shows that the algorithm does not always succeed. For both initial conditions tried, the trans-
mitter powers grow exponentially. Also, the SINR approaches $5.92 < \alpha \gamma = 6$.

2. **State equations for a linear mechanical system.** The equations of motion of a lumped mechanical system undergoing small motions can be expressed as

$$M\ddot{q} + D\dot{q} + Kq = f$$
where \( q(t) \in \mathbb{R}^k \) is the vector of deflections, \( M, D, \) and \( K \) are the mass, damping, and stiffness matrices, respectively, and \( f(t) \in \mathbb{R}^k \) is the vector of externally applied forces. Assuming \( M \) is invertible, write linear system equations for the mechanical system, with state

\[
x = \begin{bmatrix} q \\ \dot{q} \end{bmatrix},
\]

input \( u = f \), and output \( y = q \).

**Solution.** We need to express the output \( q \) and the state derivative, \( \dot{q} \) and \( \ddot{q} \), as a linear function of the state variables \( q, \dot{q} \) and the input \( f \). In other words, we should find matrices \( A, B, C \) and \( D \) such that

\[
\begin{bmatrix} \dot{q} \\ \ddot{q} \end{bmatrix} = A \begin{bmatrix} q \\ \dot{q} \end{bmatrix} + Bf, \quad q = C \begin{bmatrix} q \\ \dot{q} \end{bmatrix} + Df.
\]

Matrices \( C \) and \( D \) are easy to find: simply, for the second equation to hold we should have \( C = [ I \quad 0 ] \), \( D = 0 \).

\( A \) and \( B \) are a bit harder to find. We will use the differential equation to express \( \ddot{q} \) in terms of \( q, \dot{q} \) and \( f \). From the given dynamics equation \( M\ddot{q} + D\dot{q} + Kq = f \), and assuming \( M \) is invertible, we get

\[
\ddot{q} = -M^{-1}Kq - M^{-1}D\dot{q} + M^{-1}f,
\]

which expresses \( \ddot{q} \) in terms of \( q, \dot{q} \), and \( f \). Now we can write the linear dynamical system equations for the system. In block matrix notation we have

\[
\frac{d}{dt} \begin{bmatrix} q \\ \dot{q} \end{bmatrix} = \begin{bmatrix} 0 & I \\ -M^{-1}K & -M^{-1}D \end{bmatrix} \begin{bmatrix} q \\ \dot{q} \end{bmatrix} + \begin{bmatrix} 0 \\ M^{-1} \end{bmatrix} u, \quad y = \begin{bmatrix} I & 0 \end{bmatrix} \begin{bmatrix} q \\ \dot{q} \end{bmatrix},
\]

so the matrices in linear dynamical system description are:

\[
A = \begin{bmatrix} 0 & I \\ -M^{-1}K & -M^{-1}D \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ M^{-1} \end{bmatrix}, \quad C = [ I \quad 0 ], \quad D = 0.
\]

**3. Representing linear functions as matrix multiplication.** Suppose that \( f : \mathbb{R}^n \rightarrow \mathbb{R}^m \) is linear. Show that there is a matrix \( A \in \mathbb{R}^{m \times n} \) such that for all \( x \in \mathbb{R}^n, f(x) = Ax \). (Explicitly describe how you get the coefficients \( A_{ij} \) from \( f \), and then verify that \( f(x) = Ax \) for any \( x \in \mathbb{R}^n \).) Is the matrix \( A \) that represents \( f \) unique? In other words, if \( \tilde{A} \in \mathbb{R}^{m \times n} \) is another matrix such that \( f(x) = \tilde{A}x \) for all \( x \in \mathbb{R}^n \), then do we have \( \tilde{A} = A \)? Either show that this is so, or give an explicit counterexample.

**Solution.** Any \( x \in \mathbb{R}^n \) can be written as

\[
x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = x_1 e_1 + x_2 e_2 + \cdots + x_n e_n,
\]

where \( e_i \) are the standard basis vectors.
where $e_i$ is the $i$th standard unit vector in $\mathbb{R}^n$. From linearity of $f$ we have

$$f(x) = x_1 f(e_1) + x_2 f(e_2) + \cdots + x_n f(e_n),$$

or in (block) matrix form

$$f(x) = \begin{bmatrix} f(e_1) & f(e_2) & \cdots & f(e_n) \end{bmatrix} x.$$

Therefore, we simply take

$$A := \begin{bmatrix} f(e_1) & f(e_2) & \cdots & f(e_n) \end{bmatrix}.$$

So to determine $A$ we only need to find $f(e_i)$, for $i = 1, \ldots, n$.

Suppose the matrix $A$ is not unique, which means there is another $\tilde{A} \in \mathbb{R}^{m \times n}$ such that $f(x) = \tilde{A} x$. Then $Ax = \tilde{A} x$ or $(A - \tilde{A}) x = 0$ for all $x \in \mathbb{R}^n$. When $x = e_i$, $(A - \tilde{A}) e_i = 0$ implies that the $i$th column of $(A - \tilde{A})$ is zero. Repeating this argument for $i = 1, 2, \ldots, n$ proves that all columns of $(A - \tilde{A})$ are zero and hence $A = \tilde{A}$. Therefore the choice of $A$ is unique.

4. A mass subject to applied forces. Consider a unit mass subject to a time-varying force $f(t)$ for $0 \leq t \leq n$. Let the initial position and velocity of the mass both be zero. Suppose that the force has the form $f(t) = x_j$ for $j - 1 \leq t < j$ and $j = 1, \ldots, n$. Let $y_1$ and $y_2$ denote, respectively, the position and velocity of the mass at time $t = n$.

a) Find the matrix $A \in \mathbb{R}^{2 \times n}$ such that $y = Ax$.

b) For $n = 4$, find a sequence of input forces $x_1, \ldots, x_n$ that moves the mass to position 1 with velocity 0 at time $n$.

Solution. Let $p(t)$ and $v(t)$ denote, respectively, the position and velocity of the mass at time $t$.

a) The velocity is the integral of the applied force:

$$v(t) = v(0) + \int_0^t f(\tau) d\tau$$

$$= v(0) + \sum_{j=1}^{[t]} \int_{j-1}^j f(\tau) d\tau + \int_{[t]}^t f(\tau) d\tau$$

$$= v(0) + \sum_{j=1}^{[t]} \int_{j-1}^j x_j d\tau + \int_{[t]}^t x_{[t]+1} d\tau$$

$$= v(0) + \sum_{j=1}^{[t]} (\tau x_j)\big|_{\tau=j-1}^{\tau=j} + (\tau x_{[t]+1})\big|_{\tau=[t]}^{\tau=t}$$

$$= v(0) + \sum_{j=1}^{[t]} x_j + (t - [t])x_{[t]+1}.$$
In particular, because the mass is initially at rest (that is, \( v(0) = 0 \)), the final velocity is

\[
y_2 = v(n) = \sum_{j=1}^{n} x_j.
\]

Similarly, the position is the integral of the velocity:

\[
p(t) = p(0) + \int_0^t v(\tau) d\tau
\]

\[
= p(0) + \int_0^t (v(0) + (v(\tau) - v(0))) d\tau
\]

\[
= p(0) + v(0)t + \int_0^t (v(\tau) - v(0)) d\tau
\]

\[
= p(0) + v(0)t + \sum_{j=1}^{\lfloor t \rfloor} \int_{j-1}^{j} (v(\tau) - v(0)) d\tau + \int_{\lfloor t \rfloor}^{t} (v(\tau) - v(0)) d\tau
\]

\[
= p(0) + v(0)t + \sum_{j=1}^{\lfloor t \rfloor} \int_{j-1}^{j} \left( \sum_{k=1}^{\lfloor \tau \rfloor} x_k + (\tau - \lfloor \tau \rfloor)x_{\lfloor \tau \rfloor + 1} \right) d\tau
\]

\[
+ \int_{\lfloor t \rfloor}^{t} \left( \sum_{k=1}^{\lfloor \tau \rfloor} x_k + (\tau - \lfloor \tau \rfloor)x_{\lfloor \tau \rfloor + 1} + 1 \right) d\tau
\]

\[
= p(0) + v(0)t + \sum_{j=1}^{\lfloor t \rfloor} \left( \sum_{k=1}^{j-1} x_k + (\tau - (j-1))x_j \right) d\tau
\]

\[
+ \int_{\lfloor t \rfloor}^{t} \left( \sum_{k=1}^{\lfloor \tau \rfloor} x_k + (\tau - \lfloor \tau \rfloor)x_{\lfloor \tau \rfloor + 1} \right) d\tau
\]

\[
= p(0) + v(0)t + \left. \left( \sum_{k=1}^{\lfloor \tau \rfloor} \tau x_k + \frac{1}{2}(\tau - (j-1))^2 x_j \right) \right|_{\tau=j-1}^{\tau=j}
\]

\[
+ \left. \left( \sum_{k=1}^{\lfloor \tau \rfloor} \tau x_k + \frac{1}{2}(\tau - \lfloor \tau \rfloor)^2 x_{\lfloor \tau \rfloor + 1} \right) \right|_{\tau=\lfloor t \rfloor}^{\tau=t}
\]

\[
= p(0) + v(0)t + \sum_{j=1}^{\lfloor t \rfloor} \left( \sum_{k=1}^{j-1} x_k + \frac{1}{2}x_j \right) + \left( \sum_{k=1}^{\lfloor \tau \rfloor} x_k + \frac{1}{2}(\tau - \lfloor \tau \rfloor)^2 x_{\lfloor \tau \rfloor + 1} \right)
\]

\[
= p(0) + v(0)t + \sum_{j=1}^{\lfloor t \rfloor} \sum_{k=1}^{j-1} x_k + \frac{1}{2}x_j + \sum_{k=1}^{\lfloor \tau \rfloor} x_k + \frac{1}{2}(\tau - \lfloor \tau \rfloor)^2 x_{\lfloor \tau \rfloor + 1}
\]

\[
= p(0) + v(0)t + \sum_{k=1}^{\lfloor t \rfloor} \sum_{j=k+1}^{\lfloor t \rfloor} x_k + \frac{1}{2}x_j + \sum_{k=1}^{\lfloor \tau \rfloor} x_k + \frac{1}{2}(\tau - \lfloor \tau \rfloor)^2 x_{\lfloor \tau \rfloor + 1}
\]
\[ p(0) + v(0)t + \sum_{k=1}^{[t]} ([t] - k)x_k + \sum_{k=1}^{[t]} \frac{1}{2} x_{k+1} + \sum_{k=1}^{[t]} x_k + \frac{1}{2} (t - [t])^2 x_{[t]+1} \]

\[ = p(0) + v(0)t + \sum_{k=1}^{[t]} \left( ([t] - k) + \frac{1}{2} + (t - [t]) \right) x_k + \frac{1}{2} (t - [t])^2 x_{[t]+1} \]

\[ = p(0) + v(0)t + \sum_{k=1}^{[t]} \left( t - k + \frac{1}{2} \right) x_k + \frac{1}{2} (t - [t])^2 x_{[t]+1}. \]

In particular, because the mass is initially at rest at the origin (that is, \( p(0) = 0 \) and \( v(0) = 0 \)), the final position is

\[ y_1 = p(n) = \sum_{j=1}^{n} (n - k + \frac{1}{2}) x_j. \]

Thus, we obtain the following system of linear equations:

\[ y_1 = \sum_{j=1}^{n} (n - j + \frac{1}{2}) x_j, \]

\[ y_2 = \sum_{j=1}^{n} x_j. \]

Since \( A_{ij} \) gives the coefficient of \( x_j \) in our expression for \( y_i \), we have that

\[ A_{1j} = n - j + \frac{1}{2} \quad \text{and} \quad A_{2j} = 1, \quad j = 1, \ldots, n. \]

More concretely, we have that

\[ A = \begin{bmatrix}
    n - \frac{1}{2} & n - \frac{3}{2} & \cdots & \frac{3}{2} & \frac{1}{2} \\
    1 & 1 & \cdots & 1 & 1
\end{bmatrix}. \]

b) We want to solve the following system of linear equations:

\[
\begin{bmatrix}
    7 & 5 & 3 & 1 \\
    1 & 1 & 1 & 1
\end{bmatrix}
\begin{bmatrix}
    x_1 \\
    x_2 \\
    x_3 \\
    x_4
\end{bmatrix} =
\begin{bmatrix}
    1 \\
    0
\end{bmatrix}.
\]

This system is underdetermined, and has infinitely many solutions. Suppose we choose \( x_2 = x_3 = 0 \). Then, we are left with the system

\[
\begin{bmatrix}
    \frac{7}{2} & \frac{1}{2} \\
    1 & 1
\end{bmatrix}
\begin{bmatrix}
    x_1 \\
    x_4
\end{bmatrix} =
\begin{bmatrix}
    1 \\
    0
\end{bmatrix}.
\]

The second equation implies that \( x_4 = -x_1 \). Then, the first equation becomes

\[ \frac{7}{2} x_1 + \frac{1}{2} x_4 = \frac{7}{2} x_1 - \frac{1}{2} x_1 = 3 x_1 = 1. \]
Solving this equation, we find that $x_1 = \frac{1}{3}$. Substituting this value into our expression for $x_4$ gives $x_4 = -x_1 = -\frac{1}{3}$. Thus, one sequence of input forces that moves the mass to position 1 with velocity 0 at time $n$ is

$$x = \begin{bmatrix} \frac{1}{3} \\ 0 \\ 0 \\ -\frac{1}{3} \end{bmatrix}.$$  

### 5. Paths and cycles in a directed graph.

We consider a directed graph with $n$ nodes. The graph is specified by its *node adjacency matrix* $A \in \mathbb{R}^{n \times n}$, defined as

$$A_{ij} = \begin{cases} 1 & \text{if there is an edge from node } j \text{ to node } i \\ 0 & \text{otherwise.} \end{cases}$$

Note that the edges are oriented, i.e., $A_{34} = 1$ means there is an edge from node 4 to node 3. For simplicity we do not allow self-loops, i.e., $A_{ii} = 0$ for all $i$, $1 \leq i \leq n$. A simple example illustrating this notation is shown below.

![Directed Graph Example](image)

The node adjacency matrix for this example is

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

In this example, nodes 2 and 3 are connected in both directions, i.e., there is an edge from 2 to 3 and also an edge from 3 to 2. A *path* of length $l > 0$ from node $j$ to node $i$ is a sequence $s_0 = j, s_1, \ldots, s_l = i$ of nodes, with $A_{s_k,s_{k+1}} = 1$ for $k = 0, 1, \ldots, l - 1$. For example, in the graph shown above, 1, 2, 3, 2 is a path of length 3. A *cycle* of length $l$ is a path of length $l$, with the same starting and ending node, with no repeated nodes other than the endpoints. In other words, a cycle is a sequence of nodes of the form $s_0, s_1, \ldots, s_{l-1}, s_0$, with

$$A_{s_i,s_0} = 1, \quad A_{s_0,s_1} = 1, \quad \ldots \quad A_{s_{l-1},s_0} = 1,$$

and

$$s_i \neq s_j \text{ for } i \neq j, \quad i, j = 0, \ldots, l - 1.$$  

For example, in the graph shown above, 1, 2, 3, 4, 1 is a cycle of length 4. The rest of this problem concerns a specific graph, given in the file `directed_graph.m` on the course web site. For each of the following questions, you must give the answer explicitly (for example, enclosed in a box). You must also explain clearly how you arrived at your answer.

a) What is the length of a shortest cycle? (Shortest means minimum length.)
b) What is the length of a shortest path from node 13 to node 17? (If there are no paths from node 13 to node 17, you can give the answer as ‘infinity’.)

c) What is the length of a shortest path from node 13 to node 17, that does not pass through node 3?

d) What is the length of a shortest path from node 13 to node 17, that does pass through node 9?

e) Among all paths of length 10 that start at node 5, find the most common ending node.

f) Among all paths of length 10 that end at node 8, find the most common starting node.

g) Among all paths of length 10, find the most common pair of starting and ending nodes. In other words, find \( i, j \) which maximize the number of paths of length 10 from \( i \) to \( j \).

Solution.

a) Recall that \( (A^k)_{ij} \) gives the number of paths of length \( k \) from node \( j \) to node \( i \). Thus, \( (A^k)_{ii} \) is the number of paths of length \( k \) from node \( i \) to itself. Now imagine increasing \( k \) from \( k = 1 \) to \( k = 2 \), \( k = 3 \), and so on. We find the smallest \( k \) for which \( (A^k)_{ii} > 0 \). This \( k \) is the length of the smallest path from \( i \) to itself. This path is in fact also a cycle, since it cannot repeat nodes. (If it repeated nodes, there would have been a shorter path from \( i \) to itself.) Now let’s solve the problem. To find the length of a shortest cycle, find the smallest \( k \) such that \( (A^k)_{ii} > 0 \) for some \( i \). Note \( k \leq n \), because if a cycle exists then it is at most of length \( n \), where \( n \) is the number of nodes in the graph.

```matlab
clear all
directed_graph;
sz = size(A);
n = sz(1); % number of nodes
cycle_found = 0;
for k = 1:n
    if max(diag(A^k)) > 0
        cycle_found = 1;
        break;
    end
end
if cycle_found == 1
    length = k
else
    fprintf('Graph contains no cycle.');
end
>>smallest_cycle_length
length = 6
```

The smallest cycle is of length 6.
b) To find the length of a shortest path from node 13 to node 17, find the smallest $k$ such that $(A^k)_{17,13} > 0$.

```matlab
path_found = 0;
for k = 1:n
    Ak = A^k;
    if Ak(17,13) > 0
        path_found = 1;
        break;
    end
end
if path_found == 1
    length = k
else
    fprintf('No path from 13 to 17.\n')
end
>> shortest_path_13to17
length = 4
```

The shortest path from node 13 to node 17 is of length 4.

c) To find the shortest path from node 13 to node 17, that does not pass though node 3, remove node 3 from the graph and then find the shortest path from node 13 to node 17. The new adjacency matrix $B$ for the graph is obtained by removing the 3rd row and column of the matrix $A$. Then find the smallest $k$ such that $(B^k)_{17,13} > 0$.

```matlab
B = [A(1:2, 1:2) A(1:2, 4:20);
     A(4:20,1:2) A(4:20, 4:20)];
prefound = 0;
for k = 1:n-1
    Bk = B^k;
    if Bk(16,12)>0
        path_found = 1;
        break;
    end
end
if path_found == 1
    length = k
else
    fprintf('No path exists.\n')
end
>> shortest_path_13to17not3
length = 5
```

The shortest path from node 13 to node 17, that does not pass through node 3 is of length 5.
d) To find the smallest path from node 13 to node 17, that does pass through node 9, find the shortest path from node 13 to node 9 and the shortest path from node 9 to node 17.

```
path_found = 0;
for k = 1:n
    Ak = A^k;
    if Ak(9,13) > 0
        path_found = 1;
        break;
    end
end
if path_found == 1
    length13to9 = k
else
    fprintf('No path exists')
end
if path_found == 1
    path_found = 0;
    for k = 1:n
        Ak = A^k;
        if Ak(17,9) > 0
            path_found = 1;
            break;
        end
    end
    if path_found == 1
        length9to17 = k
    else
        fprintf('No path exists')
    end
    end
    if path_found == 1
        length = length13to9+length9to17
    end
>> shortest_path_13to17thru9
length13to9 = 6
length9to17 = 4
length = 10
```

The shortest path from node 13 to node 17, that does pass through node 9 is of length 10.

e) The matrix $A^{10}$ gives the number of paths of length 10, i.e., $(A^{10})_{ij}$ is the number of paths of length 10 that start at node $j$ and end at node $i$. The 5th column of the matrix $A^{10}$ gives the number of paths of length 10 that start at node 5 and end at nodes 1, 2, \ldots, 20 respectively. The index of the maximum entry of this column gives the most
common ending node for paths of length 10 starting at node 5.

\[ A10 = A^{10}; \]
\[ \text{start5} = A10(1:20,5); \]
\[ \text{[number, mostcommonendnode]} = \max(\text{start5}); \]
\[ \text{mostcommonendnode} >> \text{endnode} \]
\[ \text{mostcommonendnode} = 5 \]

The most common ending node for paths of length 10 starting at node 5, is 5.

f) The 8th row of the matrix \( A^{10} \) gives the number of paths that end at node 8. The index of the maximum entry of this row gives the most common starting node for paths of length 10 ending at node 8.

\[ \text{end8} = A10(8,1:20); \]
\[ \text{[number, mostcommonstartnode]} = \max(\text{end8}); \]
\[ \text{mostcommonstartnode} >> \text{startnode} \]
\[ \text{mostcommonstartnode} = 8 \]

The most common starting node for paths of length 10 ending at node 8, is 8.

g) The most common source/destination pair for paths of length 10 is the index of the maximum entry of \( A^{10} \), i.e., if \((i,j)\) is the most common source/destination pair then no other number in the matrix \( A^{10} \) is greater than \( (A^{10})_{ji} \).

\[ \text{[max_row, dests]} = \max(A10); \]
\[ \text{[max_e, mostcommonsource]} = \max(\text{max_row}); \]
\[ \text{mostcommonsource} >> \text{sdpair} \]
\[ \text{mostcommonsource} = 8 \]
\[ \text{mostcommondest} = 17 \]

The most common source/destination pair for paths of length 10 is \((8, 17)\).

6. Color perception. Human color perception is based on the responses of three different types of color light receptors, called cones. The three types of cones have different spectral-response characteristics, and are called L, M, and, S because they respond mainly to long, medium, and short wavelengths, respectively. In this problem we will divide the visible spectrum into 20 bands, and model the cones’ responses as follows:

\[ L_{\text{cone}} = \sum_{i=1}^{20} l_i p_i; \quad M_{\text{cone}} = \sum_{i=1}^{20} m_i p_i; \quad S_{\text{cone}} = \sum_{i=1}^{20} s_i p_i; \]

where \( p_i \) is the incident power in the \( i \)th wavelength band, and \( l_i, m_i \) and \( s_i \) are nonnegative constants that describe the spectral responses of the different cones. The perceived color
is a complex function of the three cone responses, i.e., the vector \((L_{\text{cone}}, M_{\text{cone}}, S_{\text{cone}})\), with different cone response vectors perceived as different colors. (Actual color perception is a bit more complicated than this, but the basic idea is right.)

a) **Metamers.** When are two light spectra, \(p\) and \(\tilde{p}\), visually indistinguishable? (Visually identical lights with different spectral power compositions are called metamers.)

b) **Visual color matching.** In a color matching problem, an observer is shown a test light, and is asked to change the intensities of three primary lights until the sum of the primary lights looks like the test light. In other words, the observer is asked to find a spectrum of the form

\[
p_{\text{match}} = a_1 u + a_2 v + a_3 w,
\]

where \(u, v, w\) are the spectra of the primary lights, and \(a_i\) are the intensities to be found, that is visually indistinguishable from a given test light spectrum \(p_{\text{test}}\). Can this always be done? Discuss briefly.

c) **Visual matching with phosphors.** A computer monitor has three phosphors, \(R\), \(G\), and \(B\). It is desired to adjust the phosphor intensities to create a color that looks like a reference test light. Find weights that achieve the match or explain why no such weights exist. The data for this problem is in `color_perception_data.json`, which contains the vectors `wavelength`, `B_phosphor`, `G_phosphor`, `R_phosphor`, `L_coefficients`, `M_coefficients`, `S_coefficients`, and `test_light`.

d) **Effects of illumination.** An object’s surface can be characterized by its reflectance (i.e., the fraction of light it reflects) for each band of wavelengths. If the object is illuminated with a light spectrum characterized by \(I_i\), and the reflectance of the object is \(r_i\) (which is between 0 and 1), then the reflected light spectrum is given by \(I_i r_i\), where \(i = 1, \ldots, 20\) denotes the wavelength band. Now consider two objects illuminated (at different times) by two different light sources, say an incandescent bulb and sunlight. Sally argues that if the two objects look identical when illuminated by a tungsten bulb, then they will look identical when illuminated by sunlight. Beth disagrees: she says that two objects can appear identical when illuminated by a tungsten bulb, but look different when lit by sunlight. Who is right? If Sally is right, explain why. If Beth is right give an example of two objects that appear identical under one light source and different under another. You can use the vectors `sunlight` and `tungsten` defined in the data file as the light sources.

**Remark.** Spectra, intensities, and reflectances are all nonnegative quantities, which the material of EE263 doesn’t address. So just ignore this while doing this problem. These issues can be handled using the material of EE364a, however.

**Solution.**

a) Let

\[
A = \begin{bmatrix}
l_1 & l_2 & l_3 & \cdots & l_{20} \\
m_1 & m_2 & m_3 & \cdots & m_{20} \\
s_1 & s_2 & s_3 & \cdots & s_{20}
\end{bmatrix}.
\]
Now suppose that \( c = Ap \) is the cone response to the spectrum \( p \) and \( \tilde{c} = A\tilde{p} \) is the cone response to spectrum \( \tilde{p} \). If the spectra are indistinguishable, then \( c = \tilde{c} \) and \( Ap = A\tilde{p} \). Solving the last expression for zero gives \( A(p - \tilde{p}) = 0 \). In other words, \( p \) and \( \tilde{p} \) are metamers if \( (p - \tilde{p}) \in \text{null}(A) \).

b) In symbols, the problem asks if it is always possible to find nonnegative \( a_1, a_2, \) and \( a_3 \) such that

\[
\begin{bmatrix}
m_1 \\
m_2 \\
m_3
\end{bmatrix} = A_p \text{test} = A \begin{bmatrix} u \\ v \\ w \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}.
\]

Let \( P = \begin{bmatrix} u & v & w \end{bmatrix} \) and let \( B = AP \). If \( B \) is invertible, then

\[
\begin{bmatrix}
a_1 \\
a_2 \\
a_3
\end{bmatrix} = B^{-1} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \end{bmatrix}.
\]

However, \( B \) is not necessarily invertible. For example, if \( \text{rank}(A) < 3 \) or \( \text{rank}(P) < 3 \) then \( B \) will be singular. Physically, \( A \) is full rank if the L, M, and S cone responses are linearly independent, which they are. The matrix \( P \) is full rank if and only if the spectra of the primary lights are independent. Even if both \( A \) and \( P \) are full rank, \( B \) could still be singular. Primary lights that generate an invertible \( B \) are called \textit{visually independent}. If \( B \) is invertible, \( a_1, a_2, \) and \( a_3 \) exist that satisfy

\[
A p_{\text{test}} = A \begin{bmatrix} u & v & w \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}.
\]

but one or more of the \( a_i \) may be negative in which case in the experimental setup described, no match would be possible. However, in a more complicated experimental setup that allows the primary lights to be combined either with each other or with \( p_{\text{test}} \), a match is always possible if \( B \) is invertible. In this case, if \( a_i < 0 \), the \( i \)th light should be mixed with \( p_{\text{test}} \) instead of the other primary lights. For example, suppose \( a_1 < 0, a_2, a_3 \geq 0 \) and \( b_1 = -a_1 \), then

\[
A(b_1 u + p_{\text{test}}) = A(a_2 v + a_3 w),
\]

and each spectrum has a nonnegative weight.

c) Weights can be found as described above. The R, G, and B phosphors should be weighted by 0.4226, 0.0987, and 0.5286 respectively.

The following Julia code illustrates the steps.

```julia
# Extraction of the data
include("readJSON263.jl");
mydata = readJSON263("color_perception.json");
```
L_coefficients = mydata["L_coefficients"]["data"];
M_coefficients = mydata["M_coefficients"]["data"];
S_coefficients = mydata["S_coefficients"]["data"];  
R_phosphor = mydata["R_phosphor"]["data"];  
G_phosphor = mydata["G_phosphor"]["data"];  
B_phosphor = mydata["B_phosphor"]["data"];  
test_light = mydata["test_light"]["data"];  
tungsten = mydata["tungsten"]["data"];  
sunlight = mydata["sunlight"]["data"];  

A = [L_coefficients; M_coefficients; S_coefficients];  
B = A*[R_phosphor' G_phosphor' B_phosphor'];  
weights = B\A*test_light

Equivalently, the following matlab code illustrates the steps.

close all; clear all;
color_perception;
A = [L_coefficients; M_coefficients; S_coefficients];  
B = A*[R_phosphor' G_phosphor' B_phosphor'];  
weights = inv(B)*A*test_light;

d) Beth is right. Let \( r \) and \( \tilde{r} \) be the reflectances of two objects and let \( p \) and \( \tilde{p} \) be two spectra. Let \( A \) be defined as before. Then, the objects will look identical under \( p \) if

\[
A \begin{bmatrix} r_1 & 0 & \ldots & 0 \\
0 & r_2 & \ldots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \ldots & r_{20} \end{bmatrix} p = \begin{bmatrix} \tilde{r}_1 & 0 & \ldots & 0 \\
0 & \tilde{r}_2 & \ldots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \ldots & \tilde{r}_{20} \end{bmatrix} p.
\]

This is equivalent to saying \((R - \tilde{R})p \in \text{null}(A)\). The objects will look different under \( \tilde{p} \) if, additionally, \( AR\tilde{p} \neq A\tilde{R}\tilde{p} \) which means that \((R - \tilde{R})\tilde{p} \not\in \text{null}(A)\). The following code shows how to find reflectances \( r_1 \) and \( r_2 \) for two objects such that the objects will have the same color under tungsten light and will have different colors under sunlight.

n = N[:,1];
n = n*10;

for i in 1:20
    n[i] = n[i]/tungsten[i];
end

r1 = [0; 0.2; 0.3; 0.7; 0.7; 0.8; 0.2; 0.9; 0.8; 0.2; 0.8; 0.9; 0.2; 0.8; 0.3; 0.8; 0.9; 0.2; 0.8; 0.3; 0.8];
r2 = r1 - n;
t1 = zeros(20);
t2 = zeros(20);

for i in 1:20
t1[i] = r1[i]*tungsten[i];
t2[i] = r2[i]*tungsten[i];
end

color1_tungsten = A*t1'
color2_tungsten = A*t2'

for i in 1:20
s1[i] = r1[i]*sunlight[i];
s2[i] = r2[i]*sunlight[i];
end

color1_sunlight = A*s1'
color2_sunlight = A*s2'

Or, in Matlab:

close all; clear all;
color_perception;
A = [L_coefficients; M_coefficients; S_coefficients]; N = null(A);
n = N(:,1);
n= n*10;
for i = 1:20
n(i) = n(i)/tungsten(i);
end
r1 = [0; .2; .3; .7; .7; .8; .8; .2; .9; .8; .2; .8; .9; .2; .8; .3; .8; .7; .2; .4];
r2 = r1-n;
for i = 1:20
t1(i) = r1(i)*tungsten(i);
t2(i) = r2(i)*tungsten(i);
end color1_tungsten = A*t1'; color2_tungsten = A*t2';
for i = 1:20
s1(i) = r1(i)*sunlight(i);
s2(i) = r2(i)*sunlight(i);
end color1_sun = A*s1'; color2_sun = A*s2';
253.5187