Homework 2 Solutions

EE 263 Stanford University

Summer 2017

1. 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:

$$L_{\text{cone}} = \sum_{i=1}^{20} l_i p_i, \qquad M_{\text{cone}} = \sum_{i=1}^{20} m_i p_i, \qquad S_{\text{cone}} = \sum_{i=1}^{20} s_i p_i,$$

where p_i is the incident power in the ith 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 the 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_{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

$$\begin{bmatrix} m_1 \\ m_2 \\ m_3 \end{bmatrix} = Ap_{\text{test}} = A \begin{bmatrix} u & v & w \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}.$$

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 *visually independent*. If B is invertible, a_1, a_2, and a_3 exist that satisfy

$$Ap_{\text{test}} = A \begin{bmatrix} u & v & w \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}.$$

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_{test}, a match is always possible if B is invertible. In this case, if $a_i < 0$, the ith light should be mixed with p_{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.

```
# 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

$$A \underbrace{\begin{bmatrix} r_1 & 0 & \cdots & 0 \\ 0 & r_2 & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & r_{20} \end{bmatrix}}_{R} p = A \underbrace{\begin{bmatrix} \tilde{r}_1 & 0 & \cdots & 0 \\ 0 & \tilde{r}_2 & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & \tilde{r}_{20} \end{bmatrix}}_{\tilde{R}} p.$$

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 $(R - \tilde{R})\tilde{p} \notin \text{null}(A)$. 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; 0.8; 0.2; 0.9; 0.8; 0.2; 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
```

2. Geometric analysis of navigation. Let $(x, y) \in \mathbb{R}^2$ be the unknown coordinates of a point in the plane, let $(p_i, q_i) \in \mathbb{R}^2$ be the known coordinates of a beacon for $i = 1, \ldots, n$, and let ρ_i be the measured distance between (x, y) and the ith beacon. The linearized navigation equations near a point $(x_0, y_0) \in \mathbb{R}^2$ are

$$\delta\rho \approx A \begin{bmatrix} \delta x \\ \delta y \end{bmatrix},$$

where we define the matrix $A \in \mathbb{R}^{n\times 2}$ such that

$$A_{i1} = \frac{x_0 - p_i}{\|(x_0, y_0) - (p_i, q_i)\|} \qquad \text{and} \qquad A_{i2} = \frac{y_0 - q_i}{\|(x_0, y_0) - (p_i, q_i)\|}.$$

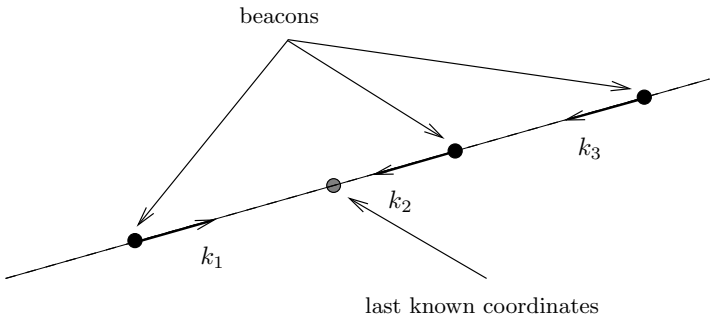
Find the conditions under which A has full rank. Describe the conditions geometrically (*i.e.*, in terms of the relative positions of the unknown coordinates and the beacons).

Solution. Suppose we have n beacons in $\mathbb{R}^2$ and the linearized equation is $y = Ax$ so $y \in \mathbb{R}^n$, $A \in \mathbb{R}^{n\times 2}$ and $x \in \mathbb{R}^2$, where

$$A = \begin{bmatrix} k_1^\mathsf{T} \\ k_2^\mathsf{T} \\ \vdots \\ k_n^\mathsf{T} \end{bmatrix}$$

and where k_i is the unit vector pointing from the ith beacon to the last known position. Since $A \in \mathbb{R}^{n\times 2}$, A is full rank if and only if at least two rows of A, or equivalently, two of the unit vectors k_i are independent. Independence of two vectors means that one should not be a scalar multiple of the other. Therefore, the only case for which A is *not* full rank is when all vectors k_i are parallel. Geometrically, in terms of the relative positions of the last known

coordinates and the beacons, this means that the beacons and the last known coordinates all lie on the same line (see figure). If the beacons and the last known coordinates do not lie on the same line then A is full rank.



3. Projection matrices. A matrix $P \in \mathbb{R}^{n\times n}$ is called a *projection matrix* if $P = P^\mathsf{T}$ and $P^2 = P$.

a) Show that if P is a projection matrix then so is $I - P$.

b) Suppose that the columns of $U \in \mathbb{R}^{n\times k}$ are orthonormal. Show that UU^T is a projection matrix. (Later we will show that the converse is true: every projection matrix can be expressed as UU^T for some U with orthonormal columns.)

c) Suppose $A \in \mathbb{R}^{n\times k}$ is full rank, with $k \leq n$. Show that $A(A^\mathsf{T}A)^{-1}A^\mathsf{T}$ is a projection matrix.

d) If $S \subseteq \mathbb{R}^n$ and $x \in \mathbb{R}^n$, the point y in S closest to x is called the *projection of x on S*. Show that if P is a projection matrix, then $y = Px$ is the projection of x on range(P). (Which is why such matrices are called projection matrices ...)

Solution.

a) To show that $I - P$ is a projection matrix we need to check two properties:

i. $I - P = (I - P)^\mathsf{T}$

ii. $(I - P)^2 = I - P$.

The first one is easy: $(I - P)^\mathsf{T} = I - P^\mathsf{T} = I - P$ because $P = P^\mathsf{T}$ (P is a projection matrix.) The show the second property we have

$$\begin{aligned}(I - P)^2 &= I - 2P + P^2 \\ &= I - 2P + P \qquad (\text{since } P = P^2) \\ &= I - P\end{aligned}$$

and we are done.

b) Since the columns of U are orthonormal we have $U^\mathsf{T}U = I$. Using this fact it is easy to prove that UU^T is a projection matrix, *i.e.*, $(UU^\mathsf{T})^\mathsf{T} = UU^\mathsf{T}$ and $(UU^\mathsf{T})^2 = UU^\mathsf{T}$. Clearly, $(UU^\mathsf{T})^\mathsf{T} = (U^\mathsf{T})^\mathsf{T}U^\mathsf{T} = UU^\mathsf{T}$ and

$$\begin{aligned}(UU^\mathsf{T})^2 &= (UU^\mathsf{T})(UU^\mathsf{T}) \\ &= U(U^\mathsf{T}U)U^\mathsf{T} \\ &= UU^\mathsf{T} \qquad \text{(since } U^\mathsf{T}U = I\text{)}.\end{aligned}$$

c) First note that $\left(A(A^\mathsf{T}A)^{-1}A^\mathsf{T}\right)^\mathsf{T} = A(A^\mathsf{T}A)^{-1}A^\mathsf{T}$ because

$$\begin{aligned}\left(A(A^\mathsf{T}A)^{-1}A^\mathsf{T}\right)^\mathsf{T} &= (A^\mathsf{T})^\mathsf{T}\left((A^\mathsf{T}A)^{-1}\right)^\mathsf{T}A^\mathsf{T} \\ &= A\left((A^\mathsf{T}A)^\mathsf{T}\right)^{-1}A^\mathsf{T} \\ &= A(A^\mathsf{T}A)^{-1}A^\mathsf{T}.\end{aligned}$$

Also $\left(A(A^\mathsf{T}A)^{-1}A^\mathsf{T}\right)^2 = A(A^\mathsf{T}A)^{-1}A^\mathsf{T}$ because

$$\begin{aligned}\left(A(A^\mathsf{T}A)^{-1}A^\mathsf{T}\right)^2 &= \left(A(A^\mathsf{T}A)^{-1}A^\mathsf{T}\right)\left(A(A^\mathsf{T}A)^{-1}A^\mathsf{T}\right) \\ &= A\left((A^\mathsf{T}A)^{-1}A^\mathsf{T}A\right)(A^\mathsf{T}A)^{-1}A^\mathsf{T} \\ &= A(A^\mathsf{T}A)^{-1}A^\mathsf{T} \qquad \text{(since } (A^\mathsf{T}A)^{-1}A^\mathsf{T}A = I\text{)}.\end{aligned}$$

d) To show that Px is the projection of x on range(P) we verify that the "error" $x - Px$ is orthogonal to *any* vector in range(P). Since range(P) is nothing but the span of the columns of P we only need to show that $x - Px$ is orthogonal to the columns of P, or in other words, $P^\mathsf{T}(x - Px) = 0$. But

$$\begin{aligned}P^\mathsf{T}(x - Px) &= P(x - Px) && \text{(since } P = P^\mathsf{T}\text{)} \\ &= Px - P^2x \\ &= 0 && \text{(since } P^2 = P\text{)}\end{aligned}$$

and we are done.

4. Groups of equivalent statements. In the list below there are 11 statements about two square matrices A and B in $\mathbb{R}^{n\times n}$.

a) range$(B) \subseteq$ range(A).

b) there exists a matrix $Y \in \mathbb{R}^{n\times n}$ such that $B = YA$.

c) $AB = 0$.

d) $BA = 0$.

e) rank$(\begin{bmatrix} A & B \end{bmatrix}) =$ rank(A).

f) $\text{range}(A) \perp \text{null}(B^{\mathsf{T}})$.

g) $\text{rank}(\begin{bmatrix} A \\ B \end{bmatrix}) = \text{rank}(A)$.

h) $\text{range}(A) \subseteq \text{null}(B)$.

i) there exists a matrix $Z \in \mathbb{R}^{n \times n}$ such that $B = AZ$.

j) $\text{rank}(\begin{bmatrix} A & B \end{bmatrix}) = \text{rank}(B)$.

k) $\text{null}(A) \subseteq \text{null}(B)$.

Your job is to collect them into (the largest possible) groups of equivalent statements. Two statements are equivalent if each one implies the other. For example, the statement 'A is onto' is equivalent to '$\text{null}(A) = \{0\}$' (when A is square, which we assume here), because every square matrix that is onto has zero nullspace, and vice versa. Two statements are not equivalent if there exist (real) square matrices A and B for which one holds, but the other does not. A group of statements is equivalent if any pair of statements in the group is equivalent.

We want *just* your answer, which will consist of lists of mutually equivalent statements; we do not need any justification.

Put your answer in the following specific form. List each group of equivalent statements on a line, in (alphabetic) order. Each new line should start with the first letter not listed above. For example, you might give your answer as

a, c, d, h
b, i
e
f, g, j, k.

This means you believe that statements a, c, d, and h are equivalent; statements b and i are equivalent; and statements f, g, j, and k are equivalent. You also believe that the first group of statements is not equivalent to the second, or the third, and so on.

Solution. Let b_i be the ith column of B.

$$\begin{aligned} \text{range}(B) \subseteq \text{range}(A) &\Leftrightarrow \text{every column of } B \text{ is in the range of } A \\ &\Leftrightarrow \text{there exists a vector } z_i \text{ such that } b_i = Az_i \\ &\Leftrightarrow \text{there exists a matrix } Z \in \mathbb{R}^{n \times n} \text{ such that } B = AZ \\ &\Leftrightarrow \text{rank}(\begin{bmatrix} A & B \end{bmatrix}) = \text{rank}(A). \end{aligned} \tag{1}$$

This shows that statements a, e and i are equivalent.

$$\begin{aligned}\text{null}(A) \subseteq \text{null}(B) &\Leftrightarrow \text{null}(A)^\perp \supseteq \text{null}(B)^\perp \\ &\Leftrightarrow \text{range}(B^\mathsf{T}) \subseteq \text{range}(A^\mathsf{T}) \\ &\Leftrightarrow \text{there exists a matrix } \tilde{Y} \in \mathbb{R}^{n\times n} \text{ such that } B^\mathsf{T} = A^\mathsf{T}\tilde{Y} \\ &\Leftrightarrow \text{there exists a matrix } Y \in \mathbb{R}^{n\times n} \text{ such that } B = YA \\ &\Leftrightarrow \text{rank}(\begin{bmatrix} A^\mathsf{T} & B^\mathsf{T} \end{bmatrix}) = \text{rank}(A^\mathsf{T}) \\ &\Leftrightarrow \text{rank}(\begin{bmatrix} A \\ B \end{bmatrix}) = \text{rank}(A). \end{aligned} \tag{2}$$

This shows that statements b, g and k are equivalent.

$$\begin{aligned}\text{range}(A) \subseteq \text{null}(B) &\Leftrightarrow \text{for all } z \in \mathbb{R}^n,\ B(Az) = 0 \\ &\Leftrightarrow BA = 0. \end{aligned} \tag{3}$$

This shows that statements d and h are equivalent.

$$\begin{aligned}\text{range}(A) \perp \text{null}(B^\mathsf{T}) &\Leftrightarrow \text{range}(A) \subseteq \text{null}(B^\mathsf{T})^\perp \\ &\Leftrightarrow \text{range}(A) \subseteq \text{range}(B) \\ &\Leftrightarrow \text{rank}(\begin{bmatrix} A & B \end{bmatrix}) = \text{rank}(B). \end{aligned} \tag{4}$$

This shows that statements f and j are equivalent.

None of these groups of statements is equivalent to any other, or to c. This is demonstrated by the following counterexamples.

Take

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

Since $AB = 0$ but $BA \neq 0$, then group (**??**) and statement c are not equivalent. Furthermore since

$$\text{rank}(\begin{bmatrix} A \\ B \end{bmatrix}) = \text{rank}(A) = \text{rank}(B) = 1$$

but $\text{rank}([A \quad B]) = 2$, groups (**??**) and (**??**) are not equivalent. Groups (**??**) and (**??**) are not either.

When $A = B \neq 0$, $\text{null}(A) = \text{null}(B)$ but $AB = BA = A^2 \neq 0$. Hence groups (**??**) and (**??**) are not equivalent. Group (**??**) and statement c are not equivalent either.

Take

$$A = I, \quad B = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}.$$

Since $\text{rank}([AB]) = \text{rank}(A) = 2$ but $\text{rank}(B) = 1$, groups (**??**) and (**??**) are not equivalent. Furthermore since $BA \neq 0$ groups (**??**) and (**??**) are not equivalent. Since $AB \neq 0$, group (**??**) and statement c aren't either.

In a similar fashion, taking

$$A = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}, \quad B = I,$$

shows that groups (??) and (??) are not equivalent and that statement c and group (??) aren't either.

Thus, the final answer is

a, e, i

b, g, k

c

d, h

f, j.

5. Fitting a quadratic form to data. A quadratic form is a function $f : \mathbb{R}^n \to \mathbb{R}$ of the form

$$f(x) = \sum_{i=1}^{n} \sum_{j=1}^{n} A_{ij} x_i x_j = x^\mathsf{T} A x.$$

We can assume without loss of generality that A is symmetric because

$$x^\mathsf{T} A x = x^\mathsf{T} \left(\tfrac{1}{2}(A + A^\mathsf{T}) \right) x,$$

where $\frac{1}{2}(A + A^\mathsf{T})$ is a symmetric matrix (called the symmetric part of A). This observation follows from the fact that

$$x^\mathsf{T} A x = (x^\mathsf{T} A x)^\mathsf{T} = x^\mathsf{T} A^\mathsf{T} x,$$

where the first step follows from the fact that any scalar is equal to its own transpose, and the second step is an application of the identity that the transpose of a product is equal to the product of the transposes in the reverse order. Suppose you are given data points $(x_1, y_1), \ldots, (x_N, y_N) \in \mathbb{R}^2$, where y_i is a noisy measurement of $f(x_i)$.

a) Explain how to find a symmetric matrix $A \in \mathbb{R}^{n \times n}$ that minimizes the mean squared fitting error:

$$J = \sum_{i=1}^{N} (x_i^\mathsf{T} A x_i - y_i)^2.$$

b) Apply your method to the data given in `fit_quadratic_form_data.m`. Report your matrix A, and the corresponding relative fitting error:

$$e_{\text{rel}} = \frac{\sqrt{\sum_{i=1}^{N} (x_i^\mathsf{T} A x_i - y_i)^2}}{\sqrt{\sum_{i=1}^{N} y_i^2}}.$$

Solution.

a) We can write the mean squared error as

$$\begin{aligned} J &= \sum_{i=1}^{N} (x_i^\mathsf{T} A x_i - y_i)^2 \\ &= \left\| \begin{bmatrix} x_1^\mathsf{T} A x_1 - y_1 \\ \vdots \\ x_N^\mathsf{T} A x_N - y_N \end{bmatrix} \right\|^2 \\ &= \left\| \begin{bmatrix} \sum_{i=1}^n \sum_{j=1}^n A_{ij} (x_1)_i (x_1)_j - y_1 \\ \vdots \\ \sum_{i=1}^n \sum_{j=1}^n A_{ij} (x_N)_i (x_N)_j - y_N \end{bmatrix} \right\|^2. \end{aligned}$$

We can use the symmetry constraints $A_{ij} = A_{ji}$ to eliminate the entries of A below the main diagonal (that is, A_{ij} for $i > j$). In particular, we can write the mean squared error as

$$J = \left\| \begin{bmatrix} \sum_{i=1}^n \sum_{j=i}^n (2 - \delta_{ij}) A_{ij} (x_1)_i (x_1)_j - y_1 \\ \vdots \\ \sum_{i=1}^n \sum_{j=i}^n (2 - \delta_{ij}) A_{ij} (x_N)_i (x_N)_j - y_N \end{bmatrix} \right\|^2,$$

where the factor of $2 - \delta_{ij}$ accounts for the fact that we get two copies of each term with $i \neq j$, but only one copy of each term with $i = j$. Define the vector of unknowns $z \in \mathbb{R}^{\frac{1}{2}n(n+1)}$ by stacking the the entries of A on or above the main diagonal into a vector:

$$z = \begin{bmatrix} A_{11} \\ A_{12} \\ \vdots \\ A_{1n} \\ A_{22} \\ \vdots \\ A_{2n} \\ \vdots \\ A_{nn} \end{bmatrix}.$$

We can write J as

$$J = \|Cz - y\|^2,$$

where we define the vector $y \in \mathbb{R}^N$ such that

$$y = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix},$$

and we define the matrix $C \in \mathbb{R}^{N \times \frac{1}{2}n(n+1)}$ such that

$$C_{pk} = (2 - \delta_{i(k)j(k)})(x_p)_{i(k)} (x_p)_{j(k)},$$

where we define the indexing functions $i(k)$ and $j(k)$ such that

$$z_k = A_{i(k)j(k)}.$$

We can compute the vector z by solving a least-squares problem; a solution is

$$z = C^\dagger y.$$

(This is the unique solution if C is skinny and full rank.) Then, we can recover the matrix A by setting

$$A_{i(k)j(k)} = A_{j(k)i(k)} = z_k, \qquad k = 1, \ldots, \tfrac{1}{2}n(n+1).$$

b) Our estimate of A is

$$A = \begin{bmatrix} 1.1702 & 0.4871 & 0.0640 \\ 0.4871 & -0.6981 & 1.7454 \\ 0.0640 & 1.7454 & 0.2629 \end{bmatrix},$$

and the corresponding relative error is 0.0208.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% clean up the workspace, and load the data
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all ; close all ; clc
fit_quadratic_form_data;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% fit a matrix to the quadratic-form data
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
C = nan(N , (1/2)*n*(n+1));
k = 0;
for i = 1:n
    for j = i:n
        k = k+1;
        if i == j
            C(:,k) = x(i,:) .* x(j,:);
        else
            C(:,k) = 2 * x(i,:) .* x(j,:);
        end
    end
end
z = C\y;

k = 0;
A = nan(n,n);
for i = 1:n
    for j = i:n
        k = k+1;
```

```
        A(i,j) = z(k);
        A(j,i) = z(k);
    end
end
A
erel = norm(C*z-y) / norm(y)
```

6. Least-squares model fitting. In this problem you will use least-squares to fit several different types of models to a given set of input/output data. The data consist of a scalar input sequence u, and a scalar output sequence y, for $t = 1, \ldots, N$. You will develop several different models that relate the signals u and y.

- *Memoryless models.* In a memoryless model, the output at time t, *i.e.*, $y(t)$, depends only the input at time t, *i.e.*, $u(t)$. Another common term for such a model is *static*.

$$\begin{aligned} \text{constant model:} &\quad y(t) = c_0 \\ \text{static linear:} &\quad y(t) = c_1 u(t) \\ \text{static affine:} &\quad y(t) = c_0 + c_1 u(t) \\ \text{static quadratic:} &\quad y(t) = c_0 + c_1 u(t) + c_2 u(t)^2 \end{aligned}$$

- *Dynamic models.* In a dynamic model, $y(t)$ depends on $u(s)$ for some $s \neq t$. We consider some simple time-series models (see problem 2 in the reader), which are linear dynamic models.

$$\begin{aligned} \text{moving average (MA):} &\quad y(t) = a_0 u(t) + a_1 u(t-1) + a_2 u(t-2) \\ \text{autoregressive (AR):} &\quad y(t) = a_0 u(t) + b_1 y(t-1) + b_2 y(t-2) \\ \text{autoregressive moving average (ARMA):} &\quad y(t) = a_0 u(t) + a_1 u(t-1) + b_1 y(t-1) \end{aligned}$$

  Note that in the AR and ARMA models, $y(t)$ depends indirectly on all previous inputs, $u(s)$ for $s < t$, due to the recursive dependence on $y(t-1)$. For this reason, the AR and ARMA models are said to have *infinite memory*. The MA model, on the other hand, has a *finite memory*: $y(t)$ depends only on the current and two previous inputs. (Another term for this MA model is 3-tap system, where taps refer to taps on a delay line.)

Each of these models is specified by its parameters, *i.e.*, the scalars c_i, a_i, b_i. For each of these models, find the least-squares fit to the given data. In other words, find parameter values that minimize the sum-of-squares of the residuals. For example, for the ARMA model, pick a_0, a_1, and b_1 that minimize

$$\sum_{t=2}^{N} \left(y(t) - a_0 u(t) - a_1 u(t-1) - b_1 y(t-1)\right)^2 .$$

(Note that we start the sum at $t = 2$ which ensures that $u(t-1)$ and $y(t-1)$ are defined.) For each model, give the root-mean-square (RMS) residual, *i.e.*, the squareroot of the mean of the optimal residual squared. Plot the output $\hat{y}$ predicted by your model, and plot the residual (which is $y - \hat{y}$). The data for this problem are available from the class web page in the file `uy_data.json`. This file contains the vectors u and y and the scalar N (the length of the vectors). Now you can plot u, y, etc. *Note:* the dataset u, y is *not* generated by any of the models above. It is generated by a nonlinear recursion, which has infinite memory.

Solution. For each of the given models, we get a linear relationship between the outputs and the unknown parameters. For example, for the constant model we have

$$\begin{bmatrix} y(1) \\ y(2) \\ \vdots \\ y(N) \end{bmatrix} = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} c_0$$

Or for the static quadratic model

$$\begin{bmatrix} y(1) \\ y(2) \\ \vdots \\ y(N) \end{bmatrix} = \begin{bmatrix} 1 & u(1) & u(1)^2 \\ 1 & u(2) & u(2)^2 \\ \vdots & \vdots & \vdots \\ 1 & u(N) & u(N)^2 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix},$$

Similarly, for the autoregressive moving average model we get

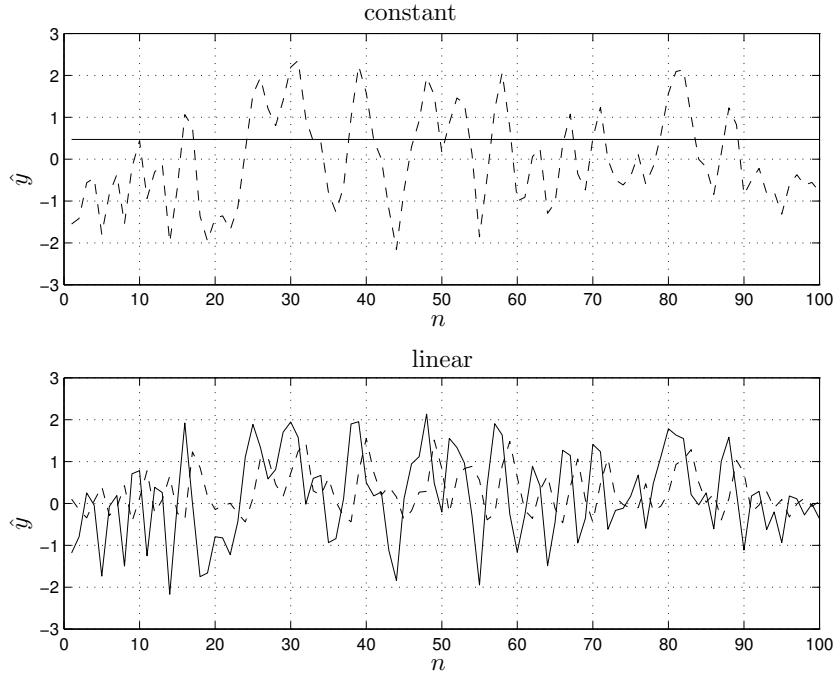
$$\begin{bmatrix} y(2) \\ y(3) \\ \vdots \\ y(N) \end{bmatrix} = \begin{bmatrix} u(2) & u(1) & y(1) \\ u(3) & u(2) & y(2) \\ \vdots & \vdots & \vdots \\ u(N) & u(N-1) & y(N-1) \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ b_1 \end{bmatrix},$$

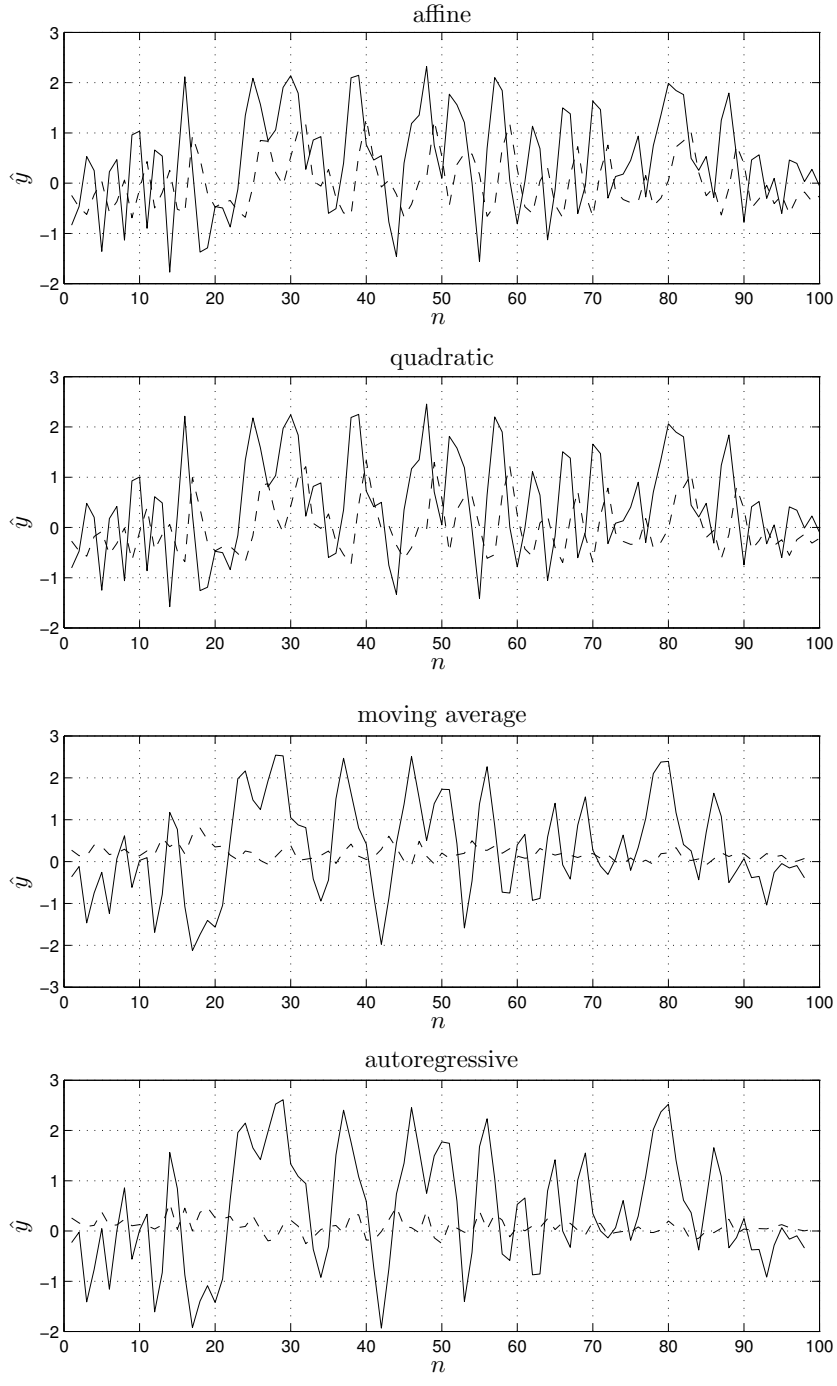
(Note that for this model we start from $y(2)$, since $u(0)$ and $y(0)$ are undefined). All of the above are in the form of $y = Ax$, where y is the output sequence, x is the vector of corresponding unknown coefficients. The goal is to find the coefficients that minimize the sum-of-squares of the residuals. This is nothing but the least-squares solution of $y = Ax$, given by $x_{ls} = (A^\mathsf{T}A)^{-1}A^\mathsf{T}y$. Then using x_{ls}, the model output can be computed as $\hat{y} = Ax_{ls}$. This can be done easily in matlab:

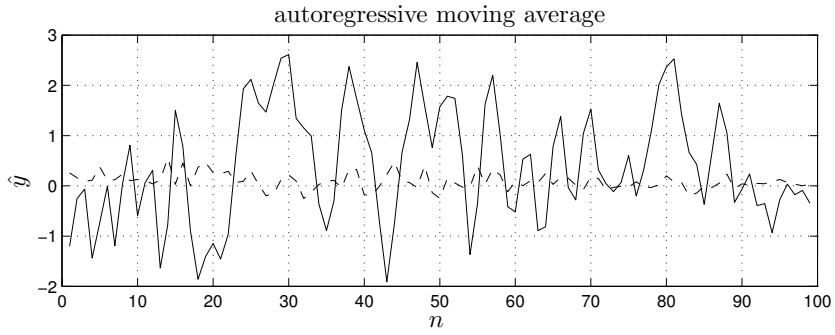
```
uy_data; % read u,y,N
A1=ones(N,1); A2=u; A3=[ones(N,1), u]; A4=[ones(N,1), u, u.^2];
x1=A1\y; y1_hat=A1*x1; r1=y-y1_hat; rms1=sqrt(mean(r1.^2))
x2=A2\y; y2_hat=A2*x2; r2=y-y2_hat; rms2=sqrt(mean(r2.^2))
x3=A3\y; y3_hat=A3*x3; r3=y-y3_hat; rms3=sqrt(mean(r3.^2))
x4=A4\y; y4_hat=A4*x4; r4=y-y4_hat; rms4=sqrt(mean(r4.^2))
A5=[u(3:N), u(2:N-1), u(1:N-2)]; y5=y(3:N); A6=[u(3:N), y(2:N-1),
y(1:N-2)]; y6=y(3:N); A7=[u(2:N), u(1:N-1), y(1:N-1)]; y7=y(2:N);
x5=A5\y5; y5_hat=A5*x5; r5=y5-y5_hat; rms5=sqrt(mean(r5.^2))
x6=A6\y6; y6_hat=A6*x6; r6=y6-y6_hat; rms6=sqrt(mean(r6.^2))
x7=A7\y7; y7_hat=A7*x7; r7=y7-y7_hat; rms7=sqrt(mean(r7.^2))
figure(1); subplot(211); plot(y1_hat,'b'); grid on; hold on;
plot(r1,'--r'); hold off; title('constant'); xlabel('n');
ylabel('y_{hat}'); subplot(212); plot(y2_hat,'b'); grid on; hold
on; plot(r2,'--r'); hold off; title('linear'); xlabel('n');
ylabel('y_{hat}');
figure(2); subplot(211); plot(y3_hat,'b'); grid on; hold on;
plot(r3,'--r'); hold off; title('affine'); xlabel('n');
```

```
ylabel('y_{hat}'); subplot(212); plot(y4_hat,'b'); grid on; hold
on; plot(r4,'--r'); hold off; title('quadratic'); xlabel('n');
ylabel('y_{hat}');
figure(3); subplot(211); plot(y5_hat,'b'); grid on; hold on;
plot(r5,'--r'); hold off; title('MA'); xlabel('n');
ylabel('y_{hat}'); subplot(212); plot(y6_hat,'b'); grid on; hold
on; plot(r7,'--r'); hold off; title('AR'); xlabel('n');
ylabel('y_{hat}');
figure(4); subplot(211); plot(y7_hat,'b'); grid on; hold on;
plot(r7,'--r'); hold off; title('ARMA'); xlabel('n');
ylabel('y_{hat}');
figure(1); print uy_1 figure(2); print uy_2 figure(3); print uy_3
figure(4); print uy_4
```

And the following RMS values for the residuals are obtained: Constant: 1.1181, linear: 0.5940, affine: 0.5210, quadratic: 0.5179; MA : 0.2504, AR : 0.1783, ARMA : 0.1853 For the memoryless models, the error decreases as the model becomes more complicated. However, the models with memory perform significantly better. Among these the error decreases with the introduction of autoregressive terms. Among the memoryless models, the affine model would be a good choice, since the more complicated quadratic model yields only slightly smaller residuals. Overall, the autoregressive model seems to do a good job. Of course, to choose a model, we should really validate on another batch of data. Note that in this problem we were only concerned with model fitting to the data, and not in 'validating' the model, *i.e.*, how well this model will work for inputs other than the ones used for fitting the model. The following plots show $\hat{y}$ (solid) and the residuals $y - \hat{y}$ (dashed):




affine
quadratic
moving average
autoregressive
$\hat{y}$
n



7. Identifying a system from input/output data. We consider the standard setup:

$$y = Ax + v,$$

where $A \in \mathbb{R}^{m\times n}$, $x \in \mathbb{R}^n$ is the input vector, $y \in \mathbb{R}^m$ is the output vector, and $v \in \mathbb{R}^m$ is the noise or disturbance. We consider here the problem of estimating the matrix A, given some input/output data. Specifically, we are given the following:

$$x^{(1)}, \ldots, x^{(N)} \in \mathbb{R}^n, \qquad y^{(1)}, \ldots, y^{(N)} \in \mathbb{R}^m.$$

These represent N samples or observations of the input and output, respectively, possibly corrupted by noise. In other words, we have

$$y^{(k)} = Ax^{(k)} + v^{(k)}, \quad k = 1, \ldots, N,$$

where $v^{(k)}$ are assumed to be small. The problem is to estimate the (coefficients of the) matrix A, based on the given input/output data. You will use a least-squares criterion to form an estimate $\hat{A}$ of A. Specifically, you will choose as your estimate $\hat{A}$ the matrix that minimizes the quantity

$$J = \sum_{k=1}^{N} \|Ax^{(k)} - y^{(k)}\|^2$$

over A.

a) Explain how to do this. If you need to make an assumption about the input/output data to make your method work, state it clearly. You may want to use the matrices $X \in \mathbb{R}^{n\times N}$ and $Y \in \mathbb{R}^{m\times N}$ given by

$$X = \begin{bmatrix} x^{(1)} & \cdots & x^{(N)} \end{bmatrix}, \qquad Y = \begin{bmatrix} y^{(1)} & \cdots & y^{(N)} \end{bmatrix}$$

in your solution.

b) On the course web site you will find some input/output data for an instance of this problem in the file `sysid_data.json`. Executing this Julia file will assign values to m, n, and N, and create two matrices that contain the input and output data, respectively. The $n \times N$ matrix variable `X` contains the input data $x^{(1)}, \ldots, x^{(N)}$ (*i.e.*, the first column of `X` contains $x^{(1)}$, etc.). Similarly, the $m \times N$ matrix `Y` contains the output data $y^{(1)}, \ldots, y^{(N)}$. You must give your final estimate $\hat{A}$, your source code, and also give an explanation of what you did.

Solution.

a) We start by expressing the objective function J as

$$\begin{aligned} J &= \sum_{k=1}^{N} \|Ax^{(k)} - y^{(k)}\|^2 \\ &= \sum_{k=1}^{N} \sum_{i=1}^{m} (Ax^{(k)} - y^{(k)})_i^2 \\ &= \sum_{k=1}^{N} \sum_{i=1}^{m} (a_i^\mathsf{T} x^{(k)} - y_i^{(k)})^2 \\ &= \sum_{i=1}^{m} \left(\sum_{k=1}^{N} (a_i^\mathsf{T} x^{(k)} - y_i^{(k)})^2 \right), \end{aligned}$$

where a_i^T is the ith row of A. The last expression shows that J is a sum of expressions J_i (shown in parentheses), each of which only depends on a_i. This means that to minimize J, we can minimize each of these expressions separately. That makes sense: we can estimate the rows of A separately. Now let's see how to minimize

$$J_i = \sum_{k=1}^{N} (a_i^\mathsf{T} x^{(k)} - y_i^{(k)})^2,$$

which is the contribution to J from the ith row of A. First we write it as

$$J_i = \begin{bmatrix} x^{(1)T} \\ \vdots \\ x^{(N)T} \end{bmatrix} a_i - \begin{bmatrix} y_i^{(1)} \\ \vdots \\ y_i^{(N)} \end{bmatrix}^2.$$

Now that we have the problem in the standard least-squares format, we're pretty much done. Using the matrix $X \in \mathbb{R}^{n \times N}$ given by

$$X = \begin{bmatrix} x^{(1)} & \cdots & x^{(N)} \end{bmatrix},$$

we can express the estimate as

$$\hat{a}_i = (XX^\mathsf{T})^{-1} X \begin{bmatrix} y_i^{(1)} \\ \vdots \\ y_i^{(N)} \end{bmatrix}.$$

Using the matrix $Y \in \mathbb{R}^{m \times N}$ given by

$$Y = \begin{bmatrix} y^{(1)} & \cdots & y^{(N)} \end{bmatrix},$$

we can express the estimate of A as

$$\hat{A}^\mathsf{T} = (XX^\mathsf{T})^{-1} XY^\mathsf{T}.$$

Transposing this gives the final answer:

$$\hat{A} = YX^\mathsf{T}(XX^\mathsf{T})^{-1}.$$

b) Once you have the neat formula found above, it's easy to get matlab to compute the estimate. It's a little inefficient, but perfectly correct, to simply use

```
Ahat = Y*X'*inv(X*X');
```

This yields the estimate

$$\hat{A} = \begin{bmatrix} 2.03 & 5.02 & 5.01 \\ 0.01 & 7 & 1.01 \\ 7.04 & 0 & 6.94 \\ 7 & 3.98 & 4 \\ 9.01 & 1.04 & 7 \\ 4.01 & 3.96 & 9.03 \\ 4.99 & 6.97 & 8.03 \\ 7.94 & 6.09 & 3.02 \\ 0.01 & 8.97 & -0.04 \\ 1.06 & 8.02 & 7.03 \end{bmatrix}.$$

Once you've got $\hat{A}$, it's a good idea to check the residuals, just to make sure it's reasonable, by comparing it to

$$\sum_{k=1}^{N} \|y^{(k)}\|^2.$$

Here we get $(64.5)^2$, around 4.08%. There are several other ways to compute $\hat{A}$ in matlab. You can calculate the rows of $\hat{A}$ one at a time, using

```
a1hat = (X'\(Y(i,:)'))';
```

In fact, the backslash operator in matlab solves multiple least-squares problems at once, so you can use

```
AhatT = X' \ (Y');
Ahat = AhatT';
```

or

```
Ahat = (X'\(Y'))';
```

In any case, it's not exactly a long matlab program ...