Homework 3 Solution

EE263 Stanford University, Fall 2018

Due: Thursday 10/18/18 11:59pm

1. Orthogonal complement of a subspace. If \( V \) is a subspace of \( \mathbb{R}^n \) we define \( V^\perp \) as the set of vectors orthogonal to every element in \( V \), i.e.,

\[
V^\perp = \{ x \mid \langle x, y \rangle = 0, \forall y \in V \}.
\]

a) Verify that \( V^\perp \) is a subspace of \( \mathbb{R}^n \).

b) Suppose \( V \) is described as the span of some vectors \( v_1, v_2, \ldots, v_r \). Express \( V \) and \( V^\perp \) in terms of the matrix \( V = [v_1 \ v_2 \ \cdots \ v_r] \in \mathbb{R}^{n \times r} \) using common terms (range, nullspace, transpose, etc.)

c) Show that every \( x \in \mathbb{R}^n \) can be expressed uniquely as \( x = v + v^\perp \) where \( v \in V \), \( v^\perp \in V^\perp \).

\textit{Hint:} let \( v \) be the projection of \( x \) on \( V \).

d) Show that \( \dim V^\perp + \dim V = n \).

e) Show that \( V \subseteq U \) implies \( U^\perp \subseteq V^\perp \).

Solution.

a) We do not need to check all properties of a vector space to hold for \( V^\perp \), since many of them hold only because \( V^\perp \subseteq \mathbb{R}^n \) and the vector sum and scalar product definitions over \( V^\perp \) and \( \mathbb{R}^n \) are the same. We only need to verify the following properties:

\begin{itemize}
  \item 0 \in V^\perp.
  \item \forall x_1, x_2 \in V^\perp : x_1 + x_2 \in V^\perp.
  \item \forall \alpha \in \mathbb{R}, \forall x \in V^\perp : \alpha x \in V^\perp.
\end{itemize}

The first property comes from the fact that \( \langle 0, y \rangle = 0 \) for all \( y \in V \) and therefore \( 0 \in V^\perp \). To verify the second property, we pick two arbitrary elements \( x_1 \) and \( x_2 \) in \( V^\perp \) and show that \( x_1 + x_2 \in V^\perp \). Let \( y \) be any vector in \( V \). We have

\[
\langle x_1 + x_2, y \rangle = \langle x_1, y \rangle + \langle x_2, y \rangle
= 0 + 0 \quad \text{(since } x_1 \in V^\perp \text{ and } x_2 \in V^\perp \text{)}
= 0,
\]
and therefore \( x_1 + x_2 \in \mathcal{V}^\perp \). Finally, we show that if \( \alpha \in \mathbb{R} \) and \( x \in \mathcal{V}^\perp \) then \( \alpha x \in \mathcal{V}^\perp \).

If \( y \in \mathcal{V} \) is arbitrary

\[
\langle \alpha x, y \rangle = \alpha \langle x, y \rangle \\
= \alpha \cdot 0 \quad \text{(since } x \in \mathcal{V}^\perp) \\
= 0,
\]

which by definition of \( \mathcal{V}^\perp \), proves that \( \alpha x \in \mathcal{V}^\perp \) and we are done.

b) Expressing \( \mathcal{V} \) in terms of the matrix \( V \) is easy. The span of vectors \( v_1, v_2, \ldots, v_r \) is simply all linear combinations of the columns of \( V \) and therefore \( \mathcal{V} = \text{range}(V) \). To express \( \mathcal{V}^\perp \) in terms of \( V \) we use the trivial fact that \( x \in \mathcal{V}^\perp \) if and only if \( x \perp v_i \) for \( i = 1, \ldots, r \). (If \( x \perp v_i \) then \( x \) is orthogonal to any linear combination of the \( v_i \)’s and hence any element in \( \mathcal{V}^\perp \). If \( x \in \mathcal{V}^\perp \) then \( x \) is specially orthogonal to the vectors \( v_i \in \mathcal{V}^\perp \) for \( i = 1, \ldots, r \).) Therefore \( x \in \mathcal{V}^\perp \) if and only if \( v_i^\top x = 0 \) for \( i = 1, \ldots, r \). In other words, using matrix notation, \( x \in \mathcal{V}^\perp \) if and only if

\[
\begin{bmatrix}
v_1^\top \\
v_2^\top \\
\vdots \\
v_r^\top
\end{bmatrix} x = 0
\]

or \( V^\top x = 0 \). Therefore \( \mathcal{V}^\perp = \text{null}(V^\top) \).

c) Suppose that \( w_1, w_2, \ldots, w_k \) is an orthonormal basis for \( \mathcal{V} \). Consider the projection of \( x \) on \( \mathcal{V} \), i.e.,

\[
v := (w_1^\top x)w_1 + (w_2^\top x)w_2 + \cdots + (w_k^\top x)w_k.
\]

Clearly, \( v \in \mathcal{V} \) because it is a linear combination of the basis vectors \( w_i \). Now we show that \( x - v \) (projection error) is an element in \( \mathcal{V}^\perp \). To do this we only have to verify that \( x - v \perp w_i \) or \( w_i^\top (x - v) = 0 \) for \( i = 1, \ldots, k \). This is easy because

\[
w_i^\top (x - v) = w_i^\top x - w_i^\top v \\
= w_i^\top x - (w_i^\top x)w_i^\top w_i \quad \text{since } w_i^\top w_j = 0 \text{ for } i \neq j \\
= 0 \quad \text{since } w_i^\top w_i = 1
\]

Now that \( x - v \in \mathcal{V}^\perp \), define \( v^\perp \in \mathcal{V}^\perp \) as \( v^\perp = x - v \) so \( x = v + v^\perp \) with \( v \in \mathcal{V} \) and \( v^\perp \in \mathcal{V}^\perp \). Now we show that the decomposition \( x = v + v^\perp \) is unique. Suppose that there are two ways to express \( x \) as the sum of elements in \( \mathcal{V} \) and \( \mathcal{V}^\perp \), i.e., \( x = v_1 + v_1^\perp \) and \( x = v_2 + v_2^\perp \) where \( v_1, v_2 \in \mathcal{V} \) and \( v_1^\perp, v_2^\perp \in \mathcal{V}^\perp \). Therefore \( v_1 + v_1^\perp = v_2 + v_2^\perp \) or \( v_1 - v_2 = v_1^\perp - v_2^\perp \). But \( v_1 - v_2 \in \mathcal{V} \) (because \( v_1, v_2 \in \mathcal{V} \)) and \( v_1^\perp - v_2^\perp \in \mathcal{V}^\perp \) (because \( v_1^\perp, v_2^\perp \in \mathcal{V}^\perp \)), and by definition of \( \mathcal{V}^\perp \) we should have \( (v_1 - v_2) \perp (v_1^\perp - v_2^\perp) \) or \( (v_1 - v_2)^\top (v_1^\perp - v_2^\perp) = 0 \). Now since \( v_1 - v_2 = v_1^\perp - v_2^\perp \) this implies that

\[
(v_1 - v_2)^\top (v_1 - v_2) = \|v_1 - v_2\|^2 = 0
\]

and

\[
(v_1^\perp - v_2^\perp)^\top (v_1^\perp - v_2^\perp) = \|v_1^\perp - v_2^\perp\|^2 = 0
\]

so \( v_1 = v_2 \) and \( v_1^\perp = v_2^\perp \) or the decomposition is unique.
d) This follows from the previous part. In part (c) we showed that any vector in \( \mathbb{R}^n \) can be expressed as the sum of two elements in \( \mathcal{V} \) and \( \mathcal{V}^\perp \). Therefore, if \( \{w_i\}_{i=1}^k \) is a basis for \( \mathcal{V} \) and \( \{u_i\}_{i=1}^l \) is a basis for \( \mathcal{V}^\perp \), for arbitrary \( x \in \mathbb{R}^n \) the scalars \( \alpha_i \) and \( \beta_i \) exist such that

\[
x = \sum_{i=1}^k \alpha_i w_i + \sum_{i=1}^l \beta_i u_i
\]

or the set of vectors \( \{w_1, \ldots, w_k, u_1, \ldots, u_l\} \) span \( \mathbb{R}^n \). In fact, the vectors \( w_i \) for \( i = 1, \ldots, k \) are orthogonal to the vectors \( u_i \) for \( i = 1, \ldots, l \) by the definition of \( \mathcal{V}^\perp \) and are therefore independent. Since the set of vectors \( \{w_1, \ldots, w_k, u_1, \ldots, u_l\} \) span \( \mathbb{R}^n \) and \( w_1, \ldots, w_k, u_1, \ldots, u_l \) are independent we get

\[
\dim \mathcal{V} + \dim \mathcal{V}^\perp = k + l = n.
\]

e) To show that \( \mathcal{U}^\perp \subseteq \mathcal{V}^\perp \) we take an arbitrary element \( x \in \mathcal{U}^\perp \) and prove that \( x \in \mathcal{V}^\perp \).

Since \( x \in \mathcal{U}^\perp \) then \( x \perp y \) for all \( y \in \mathcal{U} \). But \( \mathcal{V} \subseteq \mathcal{U} \) so we also have \( x \perp y \) for all \( y \in \mathcal{V} \).

By definition of \( \mathcal{V}^\perp \), this is nothing but to state that \( x \in \mathcal{V}^\perp \) and we are done.

2. Reverse engineering a smoothing filter. A smoothing filter takes an input vector \( u \in \mathbb{R}^n \) and produces an output vector \( y \in \mathbb{R}^n \). (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 \( \lambda, \mu, \) and \( \kappa \) 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 \( \lambda, \mu, \) and \( \kappa \). In other words, you will reverse engineer the smoothing filter, working from an input-output pair.

a) Explain how to find \( \lambda, \mu, \) and \( \kappa \). (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 \texttt{rev_eng_smooth_data.m}. Give the values of the weights.
**Solution.** We first define matrices \( D_1 \in \mathbb{R}^{(n-1)\times n} \) and \( D_2 \in \mathbb{R}^{(n-2)\times n} \) as

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

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 \( \lambda \), \( \mu \), and \( \kappa \), so we re-write these equations as a set of linear equations involving \( \lambda \), \( \mu \), and \( \kappa \):

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

This is a set of \( n \) equations in the three unknowns \( \lambda \), \( \mu \), and \( \kappa \). 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 \( \lambda \), \( \mu \), and \( \kappa \) exactly). It is full rank precisely when \( y \) does not satisfy a two-term recursion, i.e., \( y_{i+1} = \alpha y_i + \beta y_{i-1} \) for some \( \alpha, \beta \in \mathbb{R} \). 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 \( \lambda \), \( \mu \), \( \kappa \) 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 \( (\lambda, \mu, \kappa) \) 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.)

3. Interpolation with rational functions. Consider a function \( f: \mathbb{R} \to \mathbb{R} \) of the form

\[
    f(x) = \frac{a_0 + a_1 x + \cdots + a_m x^m}{1 + b_1 x + \cdots + b_m x^m},
\]

where \( a_0, \ldots, a_m \) and \( b_1, \ldots, b_m \) are parameters, with either \( a_m \neq 0 \) or \( b_m \neq 0 \). Such a function is called a rational function of degree \( m \). We are given data points \( x_1, \ldots, x_N \in \mathbb{R} \), and \( y_1, \ldots, y_N \in \mathbb{R} \), where \( y_i = f(x_i) \).

a) Explain how to find a rational function of smallest degree that is consistent with the data: that is, explain how to find the smallest value of \( m \), and corresponding values of \( a_0, \ldots, a_m \), and \( b_1, \ldots, b_m \) such that \( f(x_i) = y_i \) for \( i = 1, \ldots, N \).

b) Carry out your method on the data in rational_interpolation_data.m. Report your value of \( m \), and the corresponding coefficients \( a_0, \ldots, a_m \), and \( b_1, \ldots, b_m \). Plot the data and the rational function \( f(x) \). Verify that \( y_i = f(x_i) \) for \( i = 1, \ldots, N \) (possibly with small numerical errors).

Solution.

a) Suppose we know \( m \). Then, we want to find coefficients \( a_0, \ldots, a_m \) and \( b_1, \ldots, b_m \) such that

\[
y_i = \frac{a_0 + a_1 x_i + \cdots + a_m x_i^m}{1 + b_1 x_i + \cdots + b_m x_i^m}, \quad i = 1, \ldots, N.
\]
We can rewrite this condition as
\[ a_0 + a_1 x_i + \cdots + a_m x_i^m - b_1 x_i y_i - \cdots - b_m x_i^m y_i = y_i, \quad i = 1, \ldots, N. \]

We can collect these equations into a single matrix equation:
\[
\begin{bmatrix}
1 & x_1 & \cdots & x_1^m & -x_1 y_1 & \cdots & -x_1^m y_1 \\
\vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
1 & x_N & \cdots & x_N^m & -x_N y_N & \cdots & -x_N^m y_N
\end{bmatrix}
\begin{bmatrix}
a_0 \\
a_1 \\
\vdots \\
a_m \\
b_1 \\
\vdots \\
b_m
\end{bmatrix}
= \begin{bmatrix}
y_1 \\
\vdots \\
y_N
\end{bmatrix}.
\]

There is a rational function of degree \( m \) that is consistent with the data if and only if the system above has a solution. In order to find the smallest value of \( m \), we simply try \( m = 1, 2, \ldots \), and take the smallest value of \( m \) that is consistent with the data.

b) The smallest value of \( m \) that is consistent with the data is \( m = 5 \); corresponding coefficients are
\[ a_0 = 0.2742, \quad a_1 = 1.0291, \quad a_2 = 1.2906, \]
\[ a_3 = -5.8763, \quad a_4 = -2.6738, \quad a_5 = 6.6845, \]
\[ b_1 = -1.2513, \quad b_2 = -6.5107, \quad b_3 = 3.2754, \quad b_4 = 17.3797, \quad b_5 = 6.6845. \]

The sum of squared errors for the interpolating function is 1.92e-13, which verifies that \( y_i = f(x_i) \) except for small numerical errors. A plot of the data and the interpolating function is given in figure 1.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% clean up the workspace, and load the data
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all ; close all ; clc
rational_interpolation_data;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% find the smallest value of m consistent with the data
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for m = 1:N
    Z = ones(N,2*m+1);
    for j = 1:m
        Z(:,j+1) = x.^j;
        Z(:,j+m+1) = -x.^j .* y;
    end
    if rank([Z y]) == rank(Z);
        break;
    end
end
4. Orthogonal matrices.

a) Show that if $U$ and $V$ are orthogonal, then so is $UV$.

b) Show that if $U$ is orthogonal, then so is $U^{-1}$.

c) Suppose that $U \in \mathbb{R}^{2 \times 2}$ is orthogonal. Show that $U$ is either a rotation or a reflection. Make clear how you decide whether a given orthogonal $U$ is a rotation or reflection.

Solution.

a) To prove that $UV$ is orthogonal we have to show that $(UV)^T(UV) = I$ given $U^TU = I$ and $V^TV = I$. We have

$$(UV)^T(UV) = V^TU^TU^TUV$$

$= V^TV$  \hspace{1cm} \text{(since $U^TU = I$)}

$= I$  \hspace{1cm} \text{(since $V^TV = I$)}

and we are done.
Figure 1: the data and interpolating rational function
b) Since $U$ is square and orthogonal we have $U^{-1} = U^T$ and therefore by taking inverses of both sides $U = (U^T)^{-1}$ or equivalently $U = (U^{-1})^T$ (the inverse and transpose operations commute.) But $U^T U = I$ and by substitution $U^{-1} (U^{-1})^T = I$. Since $U^{-1}$ is square this also implies that $(U^{-1})^T U^{-1} = I$ so $U^{-1}$ is orthogonal.

c) Suppose that $U = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \in \mathbb{R}^{2 \times 2}$ is orthogonal. This is true if and only if

- columns of $U$ are of unit length, i.e., $a^2 + c^2 = 1$ and $b^2 + d^2 = 1$,
- columns of $U$ are orthogonal, i.e., $ab + cd = 0$.

Since $a^2 + c^2 = 1$ we can take $a$ and $c$ as the cosine and sine of an angle $\alpha$ respectively, i.e., $a = \cos \alpha$ and $c = \sin \alpha$. For a similar reason, we can take $b = \sin \beta$ and $d = \cos \beta$. Now $ab + cd = 0$ becomes

$$\cos \alpha \sin \beta + \sin \alpha \cos \beta = 0$$

or

$$\sin(\alpha + \beta) = 0.$$ 

The sine of an angle is zero if and only if the angle is an integer multiple of $\pi$. So $\alpha + \beta = k\pi$ or $\beta = k\pi - \alpha$ with $k \in \mathbb{Z}$. Therefore

$$U = \begin{bmatrix} \cos \alpha & \sin(k\pi - \alpha) \\ \sin \alpha & \cos(k\pi - \alpha) \end{bmatrix}.$$ 

Now two things can happen:

- $k$ is even so $\sin(k\pi - \alpha) = -\sin \alpha$ and $\cos(k\pi - \alpha) = \cos \alpha$, and therefore

$$U = \begin{bmatrix} \cos \alpha & -\sin \alpha \\ \sin \alpha & \cos \alpha \end{bmatrix}.$$ 

Clearly, from the lecture notes, this represents a rotation. Note that in this case $\det U = \cos^2 \alpha + \sin^2 \alpha = 1$.

- $k$ is odd so $\sin(k\pi - \alpha) = \sin \alpha$ and $\cos(k\pi - \alpha) = -\cos \alpha$, and therefore

$$U = \begin{bmatrix} \cos \alpha & \sin \alpha \\ \sin \alpha & -\cos \alpha \end{bmatrix}.$$ 

From the lecture notes, this represents a reflection. The determinant in this case is $\det U = -\cos^2 \alpha - \sin^2 \alpha = -1$.

Therefore we have shown that any orthogonal matrix in $\mathbb{R}^{2 \times 2}$ is either a rotation or reflection whether its determinant is $+1$ or $-1$ respectively.
5. Zeroing out the board. Malavika and Reza are playing a game. This game is played on a $6 \times 6$ board as follows. First, Malavika fills the board with 36 arbitrary real numbers and then Reza performs a sequence of actions:

At each step, Reza is allowed to choose one cell, and an arbitrary real number $x$. Then he can add $x$ to the selected cell and subtract $x$ from all adjacent cells. (Some cells have four adjacent cells, some have three, and some have two.) Reza’s goal to perform a sequence of allowed actions to derive a table which consists of 36 zeros. If Reza can derive the table consisting of zeros, he wins the game, otherwise Malavika is the winner.

a) If Malavika writes 1 in a corner cell (and 0 elsewhere), can Reza win the game? If you believe the answer is positive, you should specify the sequence of actions Reza should take. If your answer is negative, you should prove that there is no possible sequence of actions that Reza can take to zero out the table.

b) Can Malavika fill in the table so that Reza has no possible way of winning the game? If your answer is positive, you should prove that there exists an initial table that Reza cannot turn into zero. If your answer is negative, you should prove that Reza can turn any initial table into zero with a sequence of allowed actions.

c) Solve part b for a $9 \times 9$ table.

Solution.

a) We represent every $6 \times 6$ table by a vector in $s \in \mathbb{R}^{36}$. Let $s^{(0)} \in \mathbb{R}^{36}$ represent the initial configuration. The action associated with the real number $x$ and the $i$th cell is equivalent to adding a vector $xa_i \in \mathbb{R}^{36}$ to $s$ where

$$(a_i)_j = \begin{cases} 
1 & j \text{ denotes the selected cell } i \\
-1 & j \text{ denotes an adjacent cell to the selected cell } i \\
0 & \text{Otherwise.}
\end{cases}$$

Thus, the table derived after a sequence of actions will be $s^{(0)} + \sum_i x_i a_i = s^{(0)} + Ax$ where $A$ is a matrix whose columns are $a_i$’s. So we are looking for an input $x$ such that $s^{(0)} + Ax = 0$. The following code solves the problem:

```matlab
>> d=6
A=[];
for i=1:d
    for j=1:d
        a=zeros(d^2,1);
        a(d*(i-1)+j)=1;
        if i>1
            a(d*(i-2)+j)=-1;
        end
        if i<d
            a(d*(i)+j)=-1;
        end
        A=[A a];
    end
end
```

10
if j>1
    a(d*(i-1)+j-1)=-1;
end
if j<d
    a(d*(i-1)+j+1)=-1;
end
A=[A,a];
end
end

s = zeros(d*d,1);
s(1) = -1;
if rank([A,s]) == rank(A)
    reshape(A\s,d,d)
end

We see that the answer will be

\[
M = -\frac{1}{13} \begin{bmatrix}
17 & 2 & -9 & -6 & 1 & 3 \\
2 & -6 & -5 & 2 & 4 & 2 \\
-9 & -5 & 8 & 9 & -1 & -5 \\
-6 & 2 & 9 & 0 & -9 & -6 \\
1 & 4 & -1 & -9 & -2 & 8 \\
3 & 2 & -5 & -6 & 8 & 16
\end{bmatrix}
\]

b) The problem is equivalent to verifying that the linear transformation defined by \(Ax\) is onto. Since \(A\) is square, it suffices to see if \(A\) has zero nullspace. The following code solves the problem.

clc
for d=[6,9]
    A=[];
    for i=1:d
        for j=1:d
            a=zeros(d^2,1);
            a(d*(i-1)+j)=1;
            if i>1
                a(d*(i-2)+j)=-1;
            end
            if i<d
                a(d*(i)+j)=-1;
            end
            if j>1
                a(d*(i-1)+j-1)=-1;
            end
    end
end

11
if $j<d$
    $a(d\times(i-1)+j+1)=-1$;
end
$A=[A,a]$;
end
fprintf('for d=%d, the dimension of nullspace is %d
',d,d^2-rank($A$))
end

We see that Reza can always win, since the transformation is onto.

c) The code from part b shows that the transformation is not onto, so Malavika can select a good initialization to win.

6. 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:

\[ x^{(1)}, \ldots, x^{(N)} \in \mathbb{R}^n, \quad y^{(1)}, \ldots, y^{(N)} \in \mathbb{R}^m. \]

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

\[ X = \begin{bmatrix} x^{(1)} & \cdots & x^{(N)} \end{bmatrix}, \quad Y = \begin{bmatrix} y^{(1)} & \cdots & y^{(N)} \end{bmatrix} \]

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

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

$$= \sum_{k=1}^{N} \sum_{i=1}^{m} (Ax^{(k)} - y^{(k)})^2_i$$

$$= \sum_{k=1}^{N} \sum_{i=1}^{m} (a_i^T x^{(k)} - y_i^{(k)})^2$$

$$= \sum_{i=1}^{m} \left( \sum_{k=1}^{N} (a_i^T x^{(k)} - y_i^{(k)})^2 \right),$$

where $a_i^T$ is the $i$th 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

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

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

$$J_i = \left\| \begin{bmatrix} x^{(1)} & \cdots & x^{(N)} \end{bmatrix} a_i - \begin{bmatrix} y_i^{(1)} \\ \vdots \\ y_i^{(N)} \end{bmatrix} \right\|^2.$$

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

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

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

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

Transposing this gives the final answer:

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

$$\hat{A} = YX'\text{inv}(X'X');$$

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

$$\text{aihat} = (X'\backslash(Y(i,:'))');$$

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

$$\text{AhatT} = X' \backslash (Y');$$
$$\text{Ahat} = \text{AhatT}';$$

or

$$\text{Ahat} = (X'\backslash(Y'))';$$

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

7. Fitting a model for hourly temperature. You are given a set of temperature measurements (in degrees C), \(y_t \in \mathbb{R}, t = 1,\ldots,N\), taken hourly over one week (so \(N = 168\)). An expert says that over this week, an appropriate model for the hourly temperature is a trend \((i.e.,\ a\ linear\ function\ of\ t)\) plus a diurnal component \((i.e.,\ a\ 24-periodic\ component):\)

$$\hat{y}_t = at + pt,$$
where \( a \in \mathbb{R} \) and \( p \in \mathbb{R}^N \) satisfies \( p_{t+24} = p_t \), for \( t = 1, \ldots, N - 24 \). We can interpret \( a \) (which has units of degrees C per hour) as the warming or cooling trend (for \( a > 0 \) or \( a < 0 \), respectively) over the week.

a) Explain how to find \( a \in \mathbb{R} \) and \( p \in \mathbb{R}^N \) (which is 24-periodic) that minimize the RMS value of \( y - \hat{y} \).

b) Carry out the procedure described in part (a) on the data set found in `tempfit_data.m`. Give the value of the trend parameter \( a \) that you find. Plot the model \( \hat{y} \) and the measured temperatures \( y \) on the same plot. (The matlab code to do this is in the data file, but commented out.)

c) Temperature prediction. Use the model found in part (b) to predict the temperature for the next 24-hour period (i.e., from \( t = 169 \) to \( t = 192 \)). The file `tempdata.m` also contains a 24 long vector \( y_{tom} \) with tomorrow’s temperatures. Plot tomorrow’s temperature and your prediction of it, based on the model found in part (b), on the same plot. What is the RMS value of your prediction error for tomorrow’s temperatures?

**Solution.**

a) Since \( p \) is 24-periodic, we only need to specify its values for \( t = 1, \ldots, 24 \). We can express the vector of model temperatures as

\[
\hat{y} = Ax,
\]

where

\[
x = \begin{bmatrix}
a \\
p_1 \\
\vdots \\
p_{24}
\end{bmatrix} \in \mathbb{R}^{25}, \quad A = \begin{bmatrix}
1 & I_{24 \times 24} \\
2 & I_{24 \times 24} \\
\vdots & \vdots \\
168 & I_{24 \times 24}
\end{bmatrix} \in \mathbb{R}^{168 \times 25},
\]

where the righthand part of \( A \) consists of 7 \( 24 \times 24 \) identity matrices stacked on top of each other.

The solution is given by

\[
x = (A^T A)^{-1} A^T y.
\]

The associated model is given by \( \hat{y} = Ax \).

This is a perfectly acceptable answer. But in this case we can work out a more explicit solution. We did not expect any of you to do this, and we don’t even consider this a better solution. But we mention it anyway.

First, we compute the value of \( a \). Using the periodicity of \( p \) we have \( y_t - at = y_{t+24k} - a(t + 24k), k = 0, \ldots, 6 \). We solve for \( a \) and we have that \( a = (y_{t+24k} - y_t)/(24k) \). We compute the value of \( a \) for \( k = 0, \ldots, 6 \) and \( t = 1, \ldots, 24 \), and we get the mean as the estimate of \( a \). Now, let’s find the values of \( p \). We know that \( p_t = p_{t+24k} \), for \( k = 0, \ldots, 6 \) and \( t = 1, \ldots, 24 \). By taking the average value of \( p_t \) over the period of seven days we find an estimate of \( p_t \), \( t = 1, \ldots, 24 \),

\[
p_t = \frac{1}{7} \sum_{k=0}^{6} (y_{24k+t} - a(24k + t)),
\]

15
b) The following matlab script solves part (b) and part (c).

```matlab
% loading the data from the files given
tempdata
% matrix A of the system y=Ax
e24=eye(24);
A = [ e24; e24; e24; e24; e24; e24; e24];
A = [(1:N)' A];
% estimate coeffs
x = A\y ;
% the fit
yhat = (A*x)';
% plot the fit and the data on the same figure
plot([1:168],y,'--r',[1:168],yhat,'-.');
% now we solve part (c)
% prediction of tomorrow's temperature using the fit
ytomhat = ([(N+1:N+24)' e24]*x)';
% plot the fit and the data for next day figure;
plot([1:24],ytom,'r',[1:24],ytomhat,'-.');
% RMS error
RMS = sqrt(norm(ytomhat-ytom)^2/24)
```

By running the above script we get \( a = -0.0121 \). The fit, \( \hat{y} \), along with the data, \( y \), are presented in the following figure.

![Figure](image)

By running the above script we get \( a = -0.0121 \). The fit, \( \hat{y} \), along with the data, \( y \), are presented in the following figure.

---

c) In part (b) we estimated the values of \( a \) and \( p_i \), \( i = 1, \ldots, 24 \), collected together as one vector \( x \in \mathbb{R}^{25} \). To estimate \( y_{\text{tom}} \), the 24-vector of tomorrow’s temperatures, we use

\[
y_{\text{tom}} = A_{\text{tom}}x,
\]
where $x$ is the value found in part (b), and

$$A_{\text{tom}} = \begin{bmatrix}
169 \\
170 \\
\vdots \\
192 \\
I_{24 \times 24}
\end{bmatrix}.$$ 

The prediction of the temperature along with the data for the next day are presented in the following figure.

![Graph showing temperature prediction and data](image)

The RMS value of the difference between the prediction and the data for the next day is 0.6522.

8. **Empirical algorithm complexity.** The runtime $T$ of an algorithm depends on its input data, which is characterized by three key parameters: $k$, $m$, and $n$. (These are typically integers that give the dimensions of the problem data.) A simple and standard model that shows how $T$ scales with $k$, $m$, and $n$ has the form

$$\hat{T} = \alpha k^\beta m^\gamma n^\delta,$$

where $\alpha$, $\beta$, $\gamma$, $\delta \in \mathbb{R}$ are constants that characterize the approximate runtime model. If, for example, $\delta \approx 3$, we say that the algorithm has (approximately) cubic complexity in $n$. (In general, the exponents $\beta$, $\gamma$, and $\delta$ need not be integers, or close to integers.)

Now suppose you are given measured runtimes for $N$ executions of the algorithm, with different sets of input data. For each data record, you are given $T_i$ (the runtime), and the parameters $k_i$, $m_i$, and $n_i$. It’s possible (and often occurs) that two data records have identical values of $k$, $m$, and $n$, but different values of $T$. This means the algorithm was run on two different data sets that had the same dimensions; the corresponding runtimes can be (and often are) a little different.
We wish to find values of $\alpha$, $\beta$, $\gamma$, and $\delta$ for which our model (approximately) fits our measurements. We define the fitting cost as

$$J = \frac{1}{N} \sum_{i=1}^{N} \left( \log \left( \frac{\hat{T}_i}{T_i} \right) \right)^2,$$

where $\hat{T}_i = \alpha k_i^\beta m_i^\gamma n_i^\delta$ is the runtime predicted by our model, using the given parameter values. This fitting cost can be (loosely) interpreted in terms of relative or percentage fit. If $(\log(\hat{T}_i/T_i))^2 \leq \epsilon$, then $\hat{T}_i$ lies between $T_i/\exp\sqrt{\epsilon}$ and $T_i\exp\sqrt{\epsilon}$.

Your task is to find constants $\alpha$, $\beta$, $\gamma$, $\delta$ that minimize $J$.

a) Explain how to do this. If your method always finds the values that give the true global minimum value of $J$, say so. If your algorithm cannot guarantee finding the true global minimum, say so. If your method requires some matrix (or matrices) to be full rank, say so.

b) Carry out your method on the data found in `empac_data.m`. Give the values of $\alpha$, $\beta$, $\gamma$, and $\delta$ you find, and the corresponding value of $J$.

Solution. We can write

$$J = \frac{1}{N} \sum_{i=1}^{N} \left( \log \left( \frac{\hat{T}_i}{T_i} \right) \right)^2$$

$$= \frac{1}{N} \sum_{i=1}^{N} (\log \alpha + \beta \log k_i + \gamma \log m_i + \delta \log n_i - \log T_i)^2$$

$$= \frac{1}{N} ||Ax - b||^2,$$

where

$$A = \begin{bmatrix} 1 & \log k_1 & \log m_1 & \log n_1 \\ 1 & \log k_2 & \log m_2 & \log n_2 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & \log k_N & \log m_N & \log n_N \end{bmatrix}, \quad b = \begin{bmatrix} \log(T_1) \\ \log(T_2) \\ \vdots \\ \log(T_N) \end{bmatrix},$$

and $x = [\log \alpha \quad \beta \quad \gamma \quad \delta]^T$. The solution (by least squares) is $x^* = (A^TA)^{-1}A^Tb$.

Running the following matlab script on our dataset gives $\alpha = \exp(x_1^*) = 9.3364 \times 10^{-15}$, $\beta = x_2^* = 3.1062$, $\gamma = x_3^* = 1.0943$, and $\delta = x_4^* = 2.1078$. The cost in this case is $J^* = 0.0113$.

For comparison, the values we used to generate the data are $\alpha = 1 \times 10^{-14}$, $\beta = 3.1$, $\gamma = 1.1$, and $\delta = 2.1$.

```matlab
empac_data;
% form the matrices
A = [ones(N,1),log(k),log(m),log(n)]; b = log(T);
% solve by least-squares
xstar = A\b;
% extract the estimated constants
```
alpha = exp(xstar(1)); beta = xstar(2); gamma = xstar(3); delta = xstar(4);
% compute J
Jstar = (1/N)*norm(A*xstar-b)^2;