EE263 Homework 4

Summer 2024

4.570. 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 $(UV)^\mathsf{T}(UV) = I$ given $U^\mathsf{T}U = I$ and $V^\mathsf{T}V = I$. We have

$$\begin{aligned}(UV)^\mathsf{T}(UV) &= V^\mathsf{T}U^\mathsf{T}UV \\ &= V^\mathsf{T}V && \text{(since } U^\mathsf{T}U = I) \\ &= I && \text{(since } V^\mathsf{T}V = I)\end{aligned}$$

and we are done.

b) Since U is square and orthogonal we have $U^{-1} = U^\mathsf{T}$ and therefore by taking inverses of both sides $U = (U^\mathsf{T})^{-1}$ or equivalently $U = (U^{-1})^\mathsf{T}$ (the inverse and transpose operations commute.) But $U^\mathsf{T}U = I$ and by substitution $U^{-1}(U^{-1})^\mathsf{T} = I$. Since U^{-1} is square this also implies that $(U^{-1})^\mathsf{T}U^{-1} = I$ 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 α 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 π. 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.

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

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

c) $AB = 0$.

d) $BA = 0$.

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

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.

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

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

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

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

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

This shows that statements d and h are equivalent.

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

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 (3) 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 (2) and (1) are not equivalent. Groups (2) and (4) are not either.

When $A = B \neq 0$, $\text{null}(A) = \text{null}(B)$ but $AB = BA = A^2 \neq 0$. Hence groups (2) and (3) are not equivalent. Group (2) 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 (1) and (4) are not equivalent. Furthermore since $BA \neq 0$ groups (1) and (3) are not equivalent. Since $AB \neq 0$, group (1) 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 (3) and (4) are not equivalent and that statement c and group (4) aren't either.

Thus, the final answer is

a, e, i

b, g, k

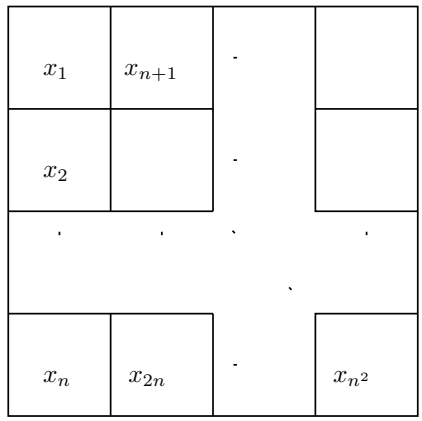
c

d, h

f, j.

6.741. Image reconstruction from line integrals. In this problem we explore a simple version of a tomography problem. We consider a square region, which we divide into an $n \times n$ array

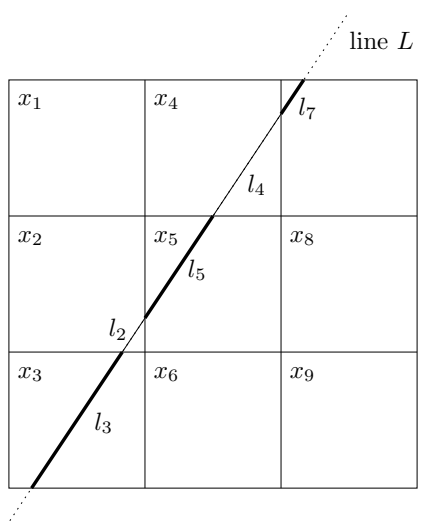
of square pixels, as shown below.



The pixels are indexed column first, by a single index i ranging from 1 to n^2, as shown above. We are interested in some physical property such as density (say) which varies over the region. To simplify things, we'll assume that the density is constant inside each pixel, and we denote by x_i the density in pixel i, $i = 1, \ldots, n^2$. Thus, $x \in \mathbb{R}^{n^2}$ is a vector that describes the density across the rectangular array of pixels. The problem is to estimate the vector of densities x, from a set of sensor measurements that we now describe. Each sensor measurement is a *line integral* of the density over a line L. In addition, each measurement is corrupted by a (small) noise term. In other words, the sensor measurement for line L is given by

$$\sum_{i=1}^{n^2} l_i x_i + v,$$

where l_i is the length of the intersection of line L with pixel i (or zero if they don't intersect), and v is a (small) measurement noise. This is illustrated below for a problem with $n = 3$. In this example, we have $l_1 = l_6 = l_8 = l_9 = 0$.



Now suppose we have N line integral measurements, associated with lines $L_1, \ldots, L_N$. From these measurements, we want to estimate the vector of densities x. The lines are characterized by the intersection lengths

$$l_{ij}, \quad i = 1, \ldots, n^2, \quad j = 1, \ldots, N,$$

where l_{ij} gives the length of the intersection of line L_j with pixel i. Then, the whole set of measurements forms a vector $y \in \mathbb{R}^N$ whose elements are given by

$$y_j = \sum_{i=1}^{n^2} l_{ij} x_i + v_j, \quad j = 1, \ldots, N.$$

And now the problem: you will reconstruct the pixel densities x from the line integral measurements y. The class webpage contains the file `tomo_data.json`, which contains the following variables:

- `N`, the number of measurements (N),
- `npixels`, the side length in pixels of the square region (n),
- `y`, a vector with the line integrals $y_j,\ j = 1, \ldots, N$,
- `line_pixel_lengths`, an $n^2 \times N$ matrix containing the intersection lengths l_{ij} of each pixel $i = 1, \ldots, n^2$ (ordered column-first as in the above diagram) and each line $j = 1, \ldots, N$,
- `lines_d`, a vector containing the displacement (distance from the center of the region in pixel lengths) d_j of each line $j = 1, \ldots, N$, and
- `lines_theta`, a vector containing the angles θ_j of each line $j = 1, \ldots, N$.

(You shouldn't need `lines_d` or `lines_theta`, but we're providing them to give you some idea of how the data was generated. Similarly, the file `tmeasure.jl` shows how we computed the measurements, but you don't need it or anything in it to solve the problem. The variable `line_pixel_lengths` was computed using the function in this file.)

Use this information to find x, and display it as an image (of n by n pixels). You'll know you have it right.

Julia hints:

- The `reshape` function might help with converting between vectors and matrices, for example, `A = reshape(v, m, n)` will convert a vector with $v = mn$ elements into an $m \times n$ matrix.
- To display a matrix `A` as a grayscale image, you can use: (or any method that works for you)
  ```
  heatmap(A, yflip=true, aspect_ratio=:equal, color=:gist_gray,
          cbar=:none, framestyle=:none)
  ```
  You'll need to have loaded the JuliaPlots package with `using Plots` to access the `heatmap` function. (The `yflip` argument gets it to plot the origin in the top-left rather than the bottom-left.)

Note: While irrelevant to your solution, this is actually a simple version of *tomography*, best known for its application in medical imaging as the CAT scan. If an *x-ray* gets attenuated at rate x_i in pixel i (a little piece of a cross-section of your body), the j-th measurement is

$$z_j = \prod_{i=1}^{n^2} e^{-x_i l_{ij}},$$

with the l_{ij} as before. Now define $y_j = -\log z_j$, and we get

$$y_j = \sum_{i=1}^{n^2} x_i l_{ij}.$$

Solution. The first thing to do is to restate the problem in the familiar form $y = Ax + v$. Here, $y \in \mathbb{R}^N$ is the measurement (given), $x \in \mathbb{R}^{n^2}$ is the physical quantity we are interested in, $A \in \mathbb{R}^{N \times n^2}$ is the relation between them, and $v \in \mathbb{R}^N$ is the noise, or measurement error (unknown, and we'll not worry about it). So we need to find the elements of A ... how do we do that?

$$\text{Comparing} \quad y = Ax, \quad i.e., \quad y_j = \sum_{i=1}^{n^2} A_{ji} x_i \quad \text{with our model} \quad y_j = \sum_{i=1}^{n^2} l_{ij} x_i + v_j$$

it is clear that $A_{ji} = l_{ij}, \quad j = 1 \ldots N, \quad i = 1 \ldots n^2$. We have thus determined a standard linear model that we want to "invert" to find x. On running `tomo_data.json`, we find that $n = 30$ and $N = 1225$, so $N > n^2$, *i.e.*, we have more rows than columns – a skinny matrix. If A is full-rank the problem is overdetermined. We can find a unique (but not exact) solution x_{ls} – the least-squares solution that minimizes $\|Ax - y\|$ – which, due to its noise-reducing properties, provides a good estimate of x (it is the *best linear unbiased estimate*). So here's what we do: we construct A element for element by finding the length of the intersection of each line with each pixel. That's done using `line_pixel_length.jl` provided. A turns out to be full-rank (`rank(A)` returns 64), so we can compute a unique x_{ls}. We go ahead and solve the least-squares problem, and then display the result.

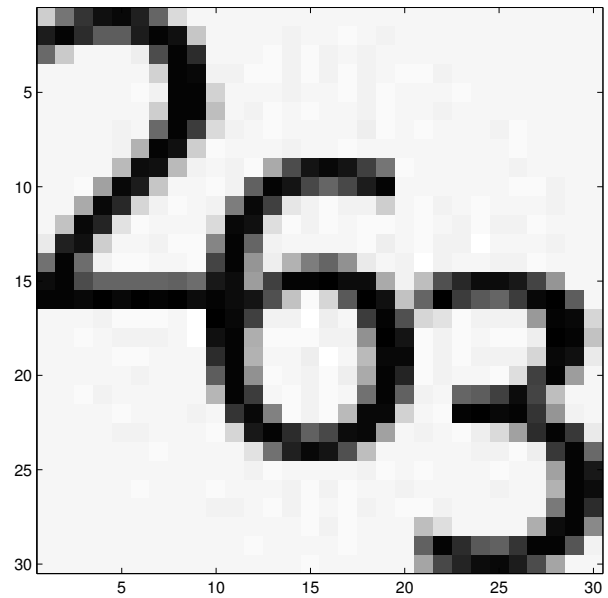
Here comes a translation of the above paragraph into Julia code:

```
using LinearAlgebra
using Plots

include("readclassjson.jl")
data = readclassjson("tomo_data.json")
N = data["N"]
L = data["line_pixel_lengths"]
npixels = data["npixels"]
y = data["y"];

x = (L*L') \ L * y
image = reshape(x, npixels, npixels)
heatmap(image, yflip=true, aspect_ratio=:equal, \
    color=:gist_gray, cbar=:none, framestyle=:none)
```

And here's the end result, the reconstructed image



6.810. Estimating a signal with interference. This problem concerns three proposed methods for estimating a signal, based on a measurement that is corrupted by a small noise and also by an interference, that need not be small. We have

$$y = Ax + Bv + w,$$

where $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{m\times p}$ are known. Here $y \in \mathbb{R}^m$ is the measurement (which is known), $x \in \mathbb{R}^n$ is the signal that we want to estimate, $v \in \mathbb{R}^p$ is the interference, and w is a noise. The noise is unknown, and can be assumed to be small. The interference is unknown, but cannot be assumed to be small. You can assume that the matrices A and B are skinny and full rank (*i.e.*, $m > n$, $m > p$), and that the ranges of A and B intersect only at 0. (If this last condition does not hold, then there is no hope of finding x, even when $w = 0$, since a nonzero interference can masquerade as a signal.) Each of the EE263 TAs proposes a method for estimating x. These methods, along with some informal justification from their proposers, are given below. Nikola proposes the **ignore and estimate method.** He describes it as follows:

> We don't know the interference, so we might as well treat it as noise, and just ignore it during the estimation process. We can use the usual least-squares method, for the model $y = Ax + z$ (with z a noise) to estimate x. (Here we have $z = Bv + w$, but that doesn't matter.)

Almir proposes the **estimate and ignore method.** He describes it as follows:

> We should simultaneously estimate both the signal x *and* the interference v, based on y, using a standard least-squares method to estimate $[x^\mathsf{T}\ v^\mathsf{T}]^\mathsf{T}$ given y. Once we've estimated x and v, we simply ignore our estimate of v, and use our estimate of x.

Miki proposes the **estimate and cancel method.** He describes it as follows:

> Almir's method makes sense to me, but I can improve it. We should simultaneously estimate both the signal x and the interference v, based on y, using a standard least-squares method, exactly as in Almir's method. In Almir's method, we then throw away $\hat{v}$, our estimate of the interference, but I think we should use it. We can form the "pseudo-measurement" $\tilde{y} = y - B\hat{v}$, which is our measurement, with the effect of the estimated interference subtracted off. Then, we use standard least-squares to estimate x from $\tilde{y}$, from the simple model $\tilde{y} = Ax + z$. (This is exactly as in Nikola's method, but here we have subtracted off or cancelled the effect of the estimated interference.)

These descriptions are a little vague; part of the problem is to translate their descriptions into more precise algorithms.

a) Give an explicit formula for each of the three estimates. (That is, for each method give a formula for the estimate $\hat{x}$ in terms of A, B, y, and the dimensions n, m, p.)

b) Are the methods really different? Identify any pairs of the methods that coincide (*i.e.*, always give exactly the same results). If they are all three the same, or all three different, say so. Justify your answer. To show two methods are the same, show that the formulas given in part (a) are equal (even if they don't appear to be at first). To show two methods are different, give a specific numerical example in which the estimates differ.

c) Which method or methods do you think work best? Give a very brief explanation. (If your answer to part (b) is "The methods are all the same" then you can simply repeat here, "The methods are all the same".)

Solution. Let's analyze the three methods. For Nikola's method, we have

$$\hat{x}_{\text{ie}} = (A^\mathsf{T} A)^{-1} A^\mathsf{T} y.$$

(The subscript 'ie' stands for 'ignore and estimate'.) Almir's method is more complicated. We start with

$$y = \begin{bmatrix} A & B \end{bmatrix} \begin{bmatrix} x \\ v \end{bmatrix} + w.$$

We need to say something about the rank of the matrix $\begin{bmatrix} A & B \end{bmatrix}$. Each is full rank, *i.e.*, has independent columns. We are also told that the ranges of the matrices A and B only intersect at 0. This means that the range of $\begin{bmatrix} A & B \end{bmatrix}$ has dimension $n+p$, which must be no more than m. Thus, the matrix $\begin{bmatrix} A & B \end{bmatrix}$ is also skinny (or square) and full rank. Forming a least-squares estimate of x and v we get

$$\begin{aligned} \begin{bmatrix} \hat{x}_{\text{ei}} \\ \hat{v}_{\text{ei}} \end{bmatrix} &= \left(\begin{bmatrix} A & B \end{bmatrix}^\mathsf{T} \begin{bmatrix} A & B \end{bmatrix} \right)^{-1} \begin{bmatrix} A & B \end{bmatrix}^\mathsf{T} y \\ &= \begin{bmatrix} A^\mathsf{T} A & A^\mathsf{T} B \\ B^\mathsf{T} A & B^\mathsf{T} B \end{bmatrix}^{-1} \begin{bmatrix} A & B \end{bmatrix}^\mathsf{T} y. \end{aligned}$$

(The subscript 'ei' stands for 'estimate and ignore'.) Finally, let's work out Miki's method. The pseudo-measurement is given by $\tilde{y} = y - B\hat{v}$, and then we have

$$\hat{x}_{\text{ec}} = (A^\mathsf{T} A)^{-1} A^\mathsf{T}(y - B\hat{v}) = (A^\mathsf{T} A)^{-1}(A^\mathsf{T} y - A^\mathsf{T} B\hat{v}),$$

where $\hat{v} = \hat{v}_{\text{ei}}$. (The subscript 'ec' stands for 'estimate and cancel'.) So far it's not clear that any of these are the same. But it turns out that Almir and Miki's method are identical. To see this, we note that

$$\begin{bmatrix} A^\mathsf{T} A & A^\mathsf{T} B \\ B^\mathsf{T} A & B^\mathsf{T} B \end{bmatrix} \begin{bmatrix} \hat{x}_{\text{ei}} \\ \hat{v}_{\text{ei}} \end{bmatrix} = \begin{bmatrix} A & B \end{bmatrix}^\mathsf{T} y.$$

Expanding this we get

$$A^\mathsf{T} A\hat{x}_{\text{ei}} + A^\mathsf{T} B\hat{v}_{\text{ei}} = A^\mathsf{T} y, \qquad B^\mathsf{T} A\hat{x}_{\text{ei}} + B^\mathsf{T} B\hat{v}_{\text{ei}} = B^\mathsf{T} y.$$

This means that

$$A^\mathsf{T} y - A^\mathsf{T} B\hat{v}_{\text{ei}} = A^\mathsf{T} A\hat{x}_{\text{ei}},$$

and plugging this into our formula above for $\hat{x}_{\text{ec}}$, we get

$$\hat{x}_{\text{ec}} = (A^\mathsf{T} A)^{-1}(A^\mathsf{T} y - A^\mathsf{T} B\hat{v}_{\text{ei}}) = (A^\mathsf{T} A)^{-1} A^\mathsf{T} A\hat{x}_{\text{ei}} = \hat{x}_{\text{ei}}.$$

Thus, we have shown that Almir's and Miki's methods are the same. In other words, the extra step of cancelling off the effect of v does *nothing*. Finally, we need to show that Nikola's is different from the other two. Here is a simple example that shows it's different:

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

We then have $\hat{x}_{\text{ie}} = [1\ 0]y$, and

$$\begin{bmatrix} \hat{x}_{\text{ei}} \\ \hat{v}_{\text{ei}} \end{bmatrix} = \begin{bmatrix} A^\mathsf{T} A & A^\mathsf{T} B \\ B^\mathsf{T} A & B^\mathsf{T} B \end{bmatrix}^{-1} \begin{bmatrix} A & B \end{bmatrix}^\mathsf{T} y = \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix} y,$$

so $\hat{x}_{\text{ei}} = [-1\ 1]y$. Clearly the estimates differ; for $y = [1\ 1]$ we have $\hat{x}_{\text{ie}} = 1$, but $\hat{x}_{\text{ie}} = 0$.

6.970. 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 α, β, γ, $\delta \in \mathbb{R}$ 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 β, γ, and δ 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 α, β, γ, and δ for which our model (approximately) fits our measurements. We define the fitting cost as

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

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 α, β, γ, δ 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 α, β, γ, and δ you find, and the corresponding value of J.

Solution. We can write

$$\begin{aligned} J &= \frac{1}{N} \sum_{i=1}^{N} \left(\log \frac{\hat{T}_i}{T_i} \right)^2 \\ &= \frac{1}{N} \sum_{i=1}^{N} (\log \alpha + \beta \log k_i + \gamma \log m_i + \delta \log n_i - \log T_i)^2 \\ &= \frac{1}{N} \|Ax - b\|^2, \end{aligned}$$

where

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

and $x = \begin{bmatrix} \log \alpha & \beta & \gamma & \delta \end{bmatrix}^\mathsf{T}$. The solution (by least squares) is $x^\star = (A^\mathsf{T} A)^{-1} A^\mathsf{T} b$.

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

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

```
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;
```