3.600. Estimating link delays from route latencies. We consider a communications network with \( m \) links that connect \( p \) nodes. There are \( n \) routes in the network, and each route is a path from some source node, going along one or more links in the network, to a destination node. The routes are determined and known.

We associate a delay \( d_i > 0 \) with each link \( i \), representing the time needed to travel the link. We use \( d = (d_1, \ldots, d_m) \) to denote the vector of link delays in the network. We have a latency \( l_j > 0 \) associated with the route \( j \), which corresponds to the total time needed to travel from the source node to the destination node of the route. We use \( l = (l_1, \ldots, l_n) \) to denote the vector of route latencies in the network.

We say that the latency vector \( l \) is consistent with the underlying link delays if there exist a set of link delays which give that latency vector. In this problem we assume that all measured latency vectors are consistent.

Before we get to the questions, we define a matrix that might be useful. The route-link incidence matrix \( R \) specifies which routes are using which links and its \((i,j)\)th entry is defined as
\[
R_{ij} = \begin{cases} 
1 & \text{route } j \text{ utilizes link } i \\
0 & \text{otherwise}
\end{cases}
\]

\( a) \) When can we perfectly and without ambiguity recover all the link delays in the network given the route latencies? (Express your answer using defined terms such as \( l, d, \) and \( R \).)

\( b) \) True or False: If \( Ry = 0 \) for some \( y \in \mathbb{R}^n \) and \( l^Ty \neq 0 \), then \( l \) is not a consistent latency vector. State if this is a true or false statement and explain your reasoning.

\( c) \) A route latency vector \( l \) and a route-link matrix \( R \) are given in the file \texttt{route_latency_data.json}. If possible, find the link delays in the network from the latency data, otherwise state that this is not possible and give two different link delays both producing the same given latency.

\( d) \) Mr. Johnson (our favorite engineer) proposes the following method to compute link delays in the network from the latency data. Here is his proposal to the Boss.

We define the count matrix \( F \in \mathbb{R}^{m \times m} \) as follows: \( F_{ij} \) is the number of routes that utilize both the link \( i \) and \( j \). Therefore, \( F_{ii} \) is the number of routes utilizing the link \( i \).

For each link \( i \), we define \( g_i \) as the sum of all latencies \( l_j \), where \( j \) is over routes that contain link \( i \).

Then we compute the link delays as \( d = F^{-1}g \), where we require that \( F \) is invertible.

Choose one of the following:

- Boss rewards Johnson since the method works whenever the delays can be perfectly recovered from the latencies.
By ‘works’ we mean that $F$ is invertible, and that $F^{-1}g$ is the unique $d$ that gives the route latencies $l$. If you believe this is the case, explain why.

- **Boss fires Johnson since the method can fail, even when the delays can be perfectly recovered from the latencies.**

To justify the firing, give a specific example, where the delays can be perfectly recovered from the latency measurements, but the method above fails, i.e., either $F$ is singular, or $F^{-1}g$ does not have the required latency totals. (Please try to give as simplest example as you can think of.)

### 4.600. Sensor integrity monitor

A suite of $m$ sensors yields measurement $y \in \mathbb{R}^m$ of some vector of parameters $x \in \mathbb{R}^n$. When the system is operating normally (which we hope is almost always the case) we have $y = Ax$, where $m > n$. If the system or sensors fail, or become faulty, then we no longer have the relation $y = Ax$. We can exploit the redundancy in our measurements to help us identify whether such a fault has occurred. We’ll call a measurement $y$ consistent if it has the form $Ax$ for some $x \in \mathbb{R}^n$. If the system is operating normally then our measurement will, of course, be consistent. If the system becomes faulty, we hope that the resulting measurement $y$ will become inconsistent, i.e., not consistent. (If we are really unlucky, the system will fail in such a way that $y$ is still consistent. Then we’re out of luck.)

A matrix $B \in \mathbb{R}^{k \times m}$ is called an integrity monitor if the following holds:

- $By = 0$ for any $y$ which is consistent.
- $By \neq 0$ for any $y$ which is inconsistent.

If we find such a matrix $B$, we can quickly check whether $y$ is consistent; we can send an alarm if $By \neq 0$. Note that the first requirement says that every consistent $y$ does not trip the alarm; the second requirement states that every inconsistent $y$ does trip the alarm. Finally, the problem. Find an integrity monitor $B$ for the matrix

$$A = \begin{bmatrix}
1 & 2 & 1 \\
1 & -1 & -2 \\
-2 & 1 & 3 \\
1 & -1 & -2 \\
1 & 1 & 0
\end{bmatrix}.$$  

Your $B$ should have the smallest $k$ (i.e., number of rows) as possible. As usual, you have to explain what you’re doing, as well as giving us your explicit matrix $B$. You must also verify that the matrix you choose satisfies the requirements. **Hints:**

- You might find one or more of the Julia functions `nullspace` or `qr` useful. Then again, you might not; there are many ways to find such a $B$.
- When checking that your $B$ works, don’t expect to have $By$ exactly zero for a consistent $y$; because of roundoff errors in computer arithmetic, it will be really, really small. That’s OK.
- Be very careful typing in the matrix $A$. It’s not just a random matrix.
5.680. **Least-squares residuals.** Suppose $A$ is skinny and full-rank. Let $x_{ls}$ be the least-squares approximate solution of $Ax = y$, and let $y_{ls} = Ax_{ls}$. Show that the residual vector $r = y - y_{ls}$ satisfies

$$
\|r\|^2 = \|y\|^2 - \|y_{ls}\|^2.
$$

Also, give a brief geometric interpretation of this equality (just a couple of sentences, and maybe a conceptual drawing).

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 $l_{ij}$, $i = 1, \ldots, n^2$, $j = 1, \ldots, N$, 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 ($N$),
- `npxels`, 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 $\theta_j$ 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, $A = \text{reshape}(v, m, n)$ 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)
  $$\text{heatmap}(A, yflip=true, aspect_ratio=:equal, color=:gist_gray, cbar=:none, framestyle=:none)$$

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

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

7.1060. Curve-smoothing. We are given a function $F : [0,1] \to \mathbb{R}$ (whose graph gives a curve in $\mathbb{R}^2$). Our goal is to find another function $G : [0,1] \to \mathbb{R}$, which is a smoothed version of $F$. We’ll judge the smoothed version $G$ of $F$ in two ways:

- *Mean-square deviation from $F$*, defined as
  $$D = \int_0^1 (F(t) - G(t))^2 \, dt.$$

- *Mean-square curvature*, defined as
  $$C = \int_0^1 G''(t)^2 \, dt.$$
We want both $D$ and $C$ to be small, so we have a problem with two objectives. In general there will be a trade-off between the two objectives. At one extreme, we can choose $G = F$, which makes $D = 0$; at the other extreme, we can choose $G$ to be an affine function (i.e., to have $G''(t) = 0$ for all $t \in [0,1]$), in which case $C = 0$. The problem is to identify the optimal trade-off curve between $C$ and $D$, and explain how to find smoothed functions $G$ on the optimal trade-off curve. To reduce the problem to a finite-dimensional one, we will represent the functions $F$ and $G$ (approximately) by vectors $f, g \in \mathbb{R}^n$, where

$$f_i = F(i/n), \quad g_i = G(i/n).$$

You can assume that $n$ is chosen large enough to represent the functions well. Using this representation we will use the following objectives, which approximate the ones defined for the functions above:

- **Mean-square deviation**, defined as
  $$d = \frac{1}{n} \sum_{i=1}^{n} (f_i - g_i)^2.$$

- **Mean-square curvature**, defined as
  $$c = \frac{1}{n-2} \sum_{i=2}^{n-1} \left( \frac{g_{i+1} - 2g_i + g_{i-1}}{1/n^2} \right)^2.$$

In our definition of $c$, note that

$$\frac{g_{i+1} - 2g_i + g_{i-1}}{1/n^2}$$

gives a simple approximation of $G''(i/n)$. You will only work with this approximate version of the problem, i.e., the vectors $f$ and $g$ and the objectives $c$ and $d$.

a) Explain how to find $g$ that minimizes $d + \mu c$, where $\mu \geq 0$ is a parameter that gives the relative weighting of sum-square curvature compared to sum-square deviation. Does your method always work? If there are some assumptions you need to make (say, on rank of some matrix, independence of some vectors, etc.), state them clearly. Explain how to obtain the two extreme cases: $\mu = 0$, which corresponds to minimizing $d$ without regard for $c$, and also the solution obtained as $\mu \to \infty$ (i.e., as we put more and more weight on minimizing curvature).

b) Get the file `curve_smoothing.json` from the course web site. This file defines a specific vector $f$ that you will use. Find and plot the optimal trade-off curve between $d$ and $c$. Be sure to identify any critical points (such as, for example, any intersection of the curve with an axis). Plot the optimal $g$ for the two extreme cases $\mu = 0$ and $\mu \to \infty$, and for three values of $\mu$ in between (chosen to show the trade-off nicely). On your plots of $g$, be sure to include also a plot of $f$, say with dotted line type, for reference.