Homework 2: Linear Systems

CS 205A: Mathematical Methods for Robotics, Vision, and Graphics (Spring 2017)
Stanford University
Due Thursday, April 27, 11:59pm

Textbook problems: 4.6(a) (10 points); 4.12(b,d) (20 points); 4.13 (15 points); 4.14(a-c) (20 points) (Omit 14.4(d))

Julia Programming Assignment (35 points): The coding portion of this problem set will be done in the provided Julia notebook. The following writeup will give you the background to understand and code up the problem. This problem was adapted from the EE103 course notes on Tomography.

In this problem you will be solving the problem of Image Tomography. Specifically, you will be reconstructing an image from sensor data. This problem will focus on the 2-D case, but it can be extended into 3-D. This technique is used in many fields including medicine, networking, and geography and one of its best known applications is the CAT (computer-aided tomography) scans.

To make the 2-D problem more concrete, suppose you wanted to reconstruct the 2-D cross-sectional image of an object sealed inside a wooden crate. You only have access to a sensor that shoots a laser through the crate and records a line integral measurement. In the discrete case of pixel values, we can visualize this process below:

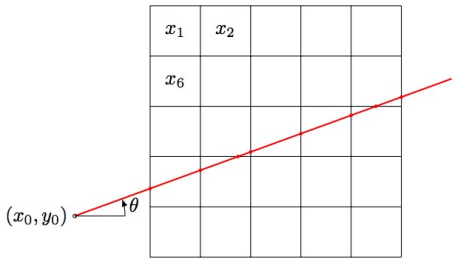

Figure 1: **A laser path:** $y = 1.06x_{16} + 0.80x_{17} + 0.27x_{12} + 1.06x_{13} + 1.06x_{14} + 0.53x_{15} + 0.54x_{10} + v$

The figure above shows a 5x5 pixel image, where each pixel is labeled from $1 \ldots 25$ and the pixel values (in gray scale) are $x_i \in [0, 1]$. The coefficient in front of each x_i denotes the length of the line segment that passes through that pixel. y, the value the sensor returns, is essentially a weighted average over the pixel values denoting the average "density" that sensor perceives. For each measurement, we model sensor noise by adding $v \sim N(0, \sigma^2)$, some small normally distributed noise.

In order to get enough measurements to reconstruct the image, we will be shooting our laser t times uniformly spaced over the image at an specific angle θ. We will then repeat this many times over r randomly chosen values of θ. In total we will have $r \cdot t$ total sensor measurements. We can see a sample of these sensor measurements on a 64x64 image in Figure 2:

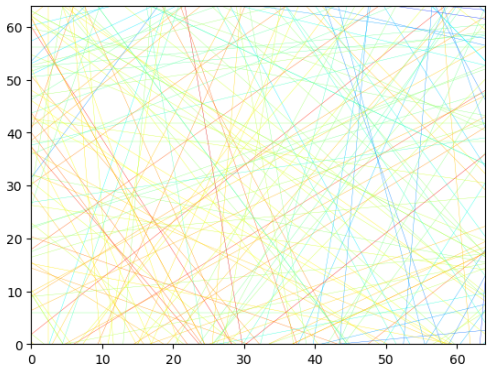

Figure 2: **Multiple laser paths:** The lines are colored by the value of the sensor measurement y. Blue lines show areas of low pixel values, and red lines show areas of high pixel values.

For the following problems, we will provide you with these sensor measurements. Your job is to reconstruct these images by solving the Least Squares problem using QR factorization. We can formulate this problem as follows.

Assume for an image of size $M \times N$, you are given m sensor measurements over an image with n total pixels. Each value $a_{i,j}$ is the length of measurement i's line segment through pixel x_j. The value y_j is the value the measurement returns.

$$a_{1,1}x_1 + a_{1,2}x_2 + \cdots + a_{1,n}x_n = y_1$$
$$a_{2,1}x_1 + a_{2,2}x_2 + \cdots + a_{2,n}x_n = y_2$$
$$\cdots$$
$$a_{m,1}x_1 + a_{m,2}x_2 + \cdots + a_{m,n}x_n = y_m$$

In short, our sensor measurements will provide you with all values of $a_{i,j}$ and y_j for $i \in \{1 \dots m\}$ and $j \in \{1 \dots n\}$. We can write the values of $a_{i,j}$ as a matrix A and the values of y_j as a vector y.

Note that the vector x is the vectorized version of the $M \times N$ image. This process is shown in Figure 1. We are essentially stacking the rows of the image, until we get a single long column vector of length $M \cdot N$. We can then formulate the entire problem as:

$$Ax = y$$

Using least squares, you can solve for the vector x, which is the value of the pixels $x_1 \dots x_n$. In the provided Julia notebook, we give you the code to reshape x and turn it into an image.

Question 1: Using the measurement values provided in `tomography_data.jld`, use QR factorization to solve the least squares problem for x. Then display the resulting image in your Julia notebook. You should find that the resulting image is incredibly grainy and choppy. Since each pixel is solved for individually, there is nothing in our least squares problem that tries to smooth out our image (neighboring pixels should be close in color). We will be using regularization to achieve this.

Question 2: Why does the usual application of Tikhonov regularization (i.e minimizing $||Ax-y||_2^2+\alpha||x||_2^2$) not help out with smoothing? What will it actually do? Solve this least squares problem using QR factorization, output the image for $\alpha = 1, 10$ and 100 and briefly discuss it in 2-3 sentences.

Alternative regularization schemes: Ideally we would want the difference between neighboring pixel values to be small. Let us denote the $M \times N$ image, the matrix of pixel values X. We can represent this mathematically as:

$$||D_h x||_2^2 + ||D_v x||_2^2 = \sum_{i=1}^{M}\sum_{j=1}^{N-1}(X_{i,j} - X_{i,j+1})^2 + \sum_{i=1}^{M-1}\sum_{j=1}^{N}(X_{i,j} - X_{i+1,j})^2$$

where the first summation is the difference between all horizontal neighbors and the second summation is the difference between all vertical neighbors. Since x is vectorized, this regularization matrix is not trivial to formulate. They are defined as:

Let D_h be a matrix of size $M(N-1) \times MN$:

$$D_{\mathrm{h}} = \begin{bmatrix} I & -I & 0 & \cdots & 0 & 0 & 0 \\ 0 & I & -I & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & I & -I & 0 \\ 0 & 0 & 0 & \cdots & 0 & I & -I \end{bmatrix}$$

where all blocks are of size $M \times M$, and I is the identity matrix.
Let D_v be a matrix of size $(M-1)N \times MN$:

$$D_{\mathrm{v}} = \begin{bmatrix} D & 0 & \cdots & 0 \\ 0 & D & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & D \end{bmatrix}$$

where each of the N diagonal blocks D is a $(M-1) \times M$ difference matrix:

$$D = \begin{bmatrix} 1 & -1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 1 & -1 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & -1 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 1 & -1 \end{bmatrix}$$

This method of regularization is commonly known as Laplacian regularization.

Question 3: Implement D_h and D_v and solve the regularized least squares problem below:

$$\textbf{minimize} \quad ||Ax - y||_2^2 + \alpha \left(||D_h x||_2^2 + ||D_v x||_2^2 \right).$$

using $\alpha = 0, 1, 10, 100$. Compare the resulting images and briefly explain in 2-4 sentences the effects of this regularization. What happens to image at the extremes (when α is 0 and 100)?

Question 4: Now we will see what happens when we reduce the amount of sensor data. In many applications like CT scans, the challenge is to get the best scan possible with as few sensor measurements as possible due to the increased exposure to radiation. Using `tomography_less_data.jld`, solve the least squares problem with and without Laplacian regularization and display both images. Roughly what α value gives you the clearest image?

Question 5: Suppose now our sensors are incredibly faulty and have increased noise. Previously, the noise was normally distributed with a variance of 0.001. In `tomography_noisy_data.jld`, we use a variance of 0.1. Solve the least squares problem with and without Laplacian regularization and display both images. Roughly what α value gives you the clearest image?

Question 6: In `tomography_secret_data.jld` we have a hidden message for you. Solve the least squares problem with Laplacian regularization and display the resulting image. What is the secret word?

Note: For Questions 4, 5 and 6, you may just re-use the code from Question 3.