EE787, Autumn 2019 Jong-Han Kim

Homework 2

1. *House price example.* This is an extension from the house price example that we have covered in class. Download `house_price_example.ipynb` and `house_price_data.csv` from the course webpage. Running the first cell, on which your kind instructor spent an hour for you, will download and pre-process the data so that it generates the data matrix $U \in \mathbf{R}^{1456\times 57}$ and $v \in \mathbf{R}^{1456}$ in `U` and `v`. For descriptions on each feature, see the corresponding lecture slide (`house_price_example.pdf`) from the course webpage.

   It further divides the data in 70-30 split, so the 70% train sets are contained in `U_train` and `v_train`, and the 30% test sets in `U_test` and `v_test`.

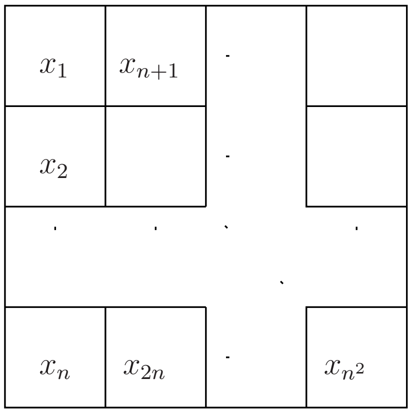
   (a) Standardize the train set and solve for a series of Tykhonov regularized regression problems, sweeping your regularization parameter λ over the range $[10^{-1}, 10^{3}]$ and plot the resulting training and test errors.

   (b) Choose an appropriate value for λ, *i.e.*, the largest value that achieves approximately minimum test error. Show your optimized model parameter θ^* and report the corresponding test error.

   (c) Download the description file `data_description.txt` which contains brief explanations on each field of the raw data. Choose whatever columns you can find from the data file, and use whatever feature engineering techniques you can do to improve your model and to repeat (b).

   (d) Report the test error that your best model achieves. Your homework will be evaluated by this number.

2. *Tomography.* 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.

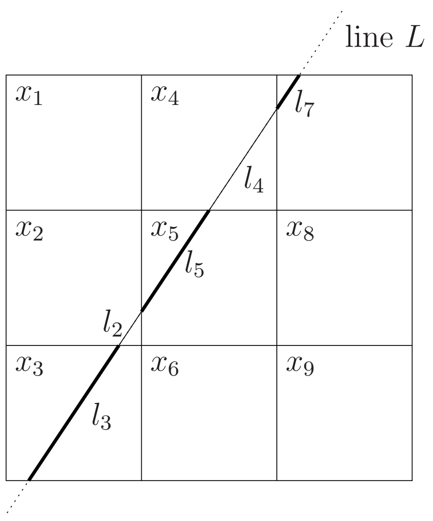


The pixels are indexed column first, by a single index i ranging from 1 to n^2, as shown above. We are interested in some physical property such as density (say) which varies over the region. To simplify things, we'll assume that the density is constant inside each pixel, and we denote by x_i the density in pixel i, $i = 1, \ldots, n^2$. Thus, $x \in \mathbf{R}^{n^2}$ 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 $l_1 = l_6 = l_8 = l_9 = 0$, and your sensor measurement will be something like

$$y_L = l_2x_2 + l_3x_3 + l_4x_4 + l_5x_5 + l_7x_7 + v_L$$



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 $y \in \mathbf{R}^N$ 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 `tomodata_fullysampled.json` and `tomodata_undersampled.json`, each of which contains the following variables:

- `N`, the number of measurements (N),
- `n_pixels`, the side length in pixels of the square region (n),
- `y`, a vector with the line integrals $y_j,\ j = 1, \ldots, N$,
- `lines_d`, a vector containing the displacement $d_j,\ j = 1, \ldots, N$, (distance from the center of the region in pixels lengths) of each line, and
- `lines_theta`, a vector containing the angles $\theta_j,\ j = 1, \ldots, N$, of each line.

We also provide the function `line_pixel_length.jl` on the webpage, which you do need to use in order to solve the problem. This function computes the pixel intersection lengths for a given line. That is, given d_j and θ_j (and the side length n), `line_pixel_length.jl` returns a $n \times n$ matrix, whose i, jth element corresponds to the intersection length for pixel i, j on the image. Use this information to find x, and display it as an image (of n by n pixels). You'll know you have it right when the image of x forms a familiar acronym...

You can also download `tomography.ipynb` if you want. This will load the `json` file and required variables for you.

(a) Use `tomodata_fullysampled.json` to reconstruct the pixel densities. Note that the file contains the measurement data obtained from 5184 line integrals on a 60×60 image, therefore providing more measurement data than the unknown variables. So you can simply do this job by setting up a problem like

$$\underset{x}{\text{minimize}} \quad \|Ax - y\|_2^2$$

where your feature matrix A can be constructed by using `line_pixel_length.jl`.

(b) Use `tomodata_undersampled.json` to reconstruct the pixel densities. Note that the file contains the measurement data obtained from 1296 line integrals on a 60×60 image, therefore providing way less measurement data than the unknown variables. This implies your A is not full column rank and not even tall, so the least squares solution is not uniquely determined. Try the least squares solution for this data, possibly by using the backslash operator in Julia.

(c) In order to handle the undersampled case, try the following regularizer defined by the 2-normed total variation (TV) function. The TV function on an $m \times n$ image X with its column-first vectorized realization $x = \text{vec}(X)$ is defined as:

$$\text{TV}_2(X) = \sum_{i=1}^{m-1} \sum_{j=1}^{n-1} \left(|X_{ij} - X_{i+1,j}|^2 + |X_{ij} - X_{i,j+1}|^2 \right) = \left\| \begin{bmatrix} D_x \\ D_y \end{bmatrix} x \right\|_2^2$$

with $D_x \in \mathbf{R}^{m(n-1)\times mn}$

$$D_x = \begin{bmatrix} -1 & & & 1 & & & \\ & -1 & & & 1 & & \\ & & \ddots & & & \ddots & \\ & & & & & & \\ & & & & & & \end{bmatrix}$$

returning the $m(n-1) \times mn$ column-wise differences and $D_y \in \mathbf{R}^{(m-1)n\times mn}$

$$D_y = \begin{bmatrix} -1 & 1 & & & & & & \\ & \ddots & \ddots & & & & & \\ & & -1 & 1 & & & & \\ & & & & -1 & 1 & & \\ & & & & & \ddots & \ddots & \\ & & & & & & & \\ & & & & & & & \end{bmatrix}$$

returning the $(m-1)n \times mn$ row-wise differences. So the TV function returns the square sum of the horizontal and vertical differences between adjacent pixels. In other words, you will have to set up and solve the following regularized regression problems.

$$\underset{x}{\text{minimize}} \quad \|Ax - y\|_2^2 + \lambda \left\| \begin{bmatrix} D_x \\ D_y \end{bmatrix} x \right\|_2^2$$

Sweep your regularization parameter λ for $\lambda = 10^{-6}, 10^{-4}, 10^{-2}, 10^{0}, 10^{2}, 10^{4}$ and compare the reconstructed images, X.

You are welcome to use `Dx` and `Dy` provided in `tomography.ipynb`, for which you will probably be grateful to your instructor and think he is a very nice person, really.

(d) You may also try the same regularized problem on the fully sampled data.

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

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