Exercises 8: Spatial smoothing at scale

In this set of exercises, we'll look at smoothing spatial data.

Kernel smoothing

In spatial smoothing, the underlying statistical model is assumed to look something like this:

$$y(s_i) = x(s_i) + \epsilon(s_i)\,, \quad i = 1, \ldots, n,$$

where s_i is the spatial location of the ith data point, x is the "true" de-noised signal, y is data, and ϵ is mean-zero error. Our goal is to estimate x in a way that leverages the assumption of spatial smoothness.

The simplest technique for denoising a spatial signal is called *kernel smoothing*. Suppose we want to estimate $x(s)$ at some target location s. The kernel-smoothing estimate takes the form of a weighted average of all the points in the data set.

$$\hat{x}(s) = \frac{\sum_{i=1}^n w_i(s, s_i) y(s_i)}{\sum_{i=1}^n w_i(s, s_i)}\,,$$

where $w(s, s_i)$ is a weighting function that (presumably) gets smaller as s and s_i get further apart. The question is, how we do choose the weights?

A very common approach is called "K nearest neighbors," where the weight function $w(s, s_i)$ takes the value $1/K$ if s_i is one of the K closest points to s, and 0 otherwise. Another common approach is to define the weights in terms of a kernel function. One example is the Gaussian kernel:

$$w_i(s, s_i; b) = \frac{1}{b} \exp\left\{ -\frac{(s - s_i)^2}{2b^2} \right\}\,,$$

which depends upon a bandwidth parameter b. Another example is the quadratic or "Epanechnikov" kernel:

$$w_i(s, s_i; b) = \max\left(0, \frac{3}{4b}\left[1 - \frac{(s - s_i)^2}{b^2}\right]\right)\,,$$

which has the advantage that it decays to zero beyond a certain distance. You can read more about kernels on Wikipedia: `https://en.wikipedia.org/wiki/Kernel_(statistics)`.

The goal of this first section is simple:

1. Download the data on atmospheric CO2 concentration (co2.csv) from the class website, which has latitude, longitude, and CO2.

2. Visualize it over the surface of the earth.

3. Use kernel smoothing to estimate a denoised spatial surface, and visualize that surface over the earth.

Note: because of the size of the data, you'll want to choose a weighting function that eventually decays to zero. Beyond that, you have considerable freedom to choose how to proceed here.

Try to write a kernel smoother that is as computationally efficient as possible.

Laplacian smoothing

Kernel smoothing is a "local" smoother, in that the estimate at some point only depends upon the immediately surrounding points. We'll now consider a "global" smoother, where the answer at any one point depends on all the data. Specifically, we'll look at a global smoother collected for discrete areal units. In discrete spatial-smoothing problems, "space" is defined not by distance (like in Euclidean space), but by a neighborhood structure that says which areal units are close to which others.

Say that we have observations y_i, each associated with a vertex $s_i \in \mathcal{S}$ in an undirected graph $\mathcal{G}$ with edge set $\mathcal{E}$. The edge set tells you which sites are neighbors on the graph. For example, in this set of exercises, we'll be looking at data from an fMRI experiment. In this case, s_i is the specific brain region ("voxel"), each y_i is a measurement of the brain activity at that voxel, and $\mathcal{E}$ is the edge structure corresponding to a regular grid. The underlying statistical model is the same: $y(s_i) = x(s_i) + \epsilon(s_i)\,, \quad i = 1, \ldots, n$.

Laplacian smoothing involves the following matrices defined with respect to the graph $\mathcal{G} = (\mathcal{S}, \mathcal{E})$:

- A, the graph adjacency matrix, which is a square $(0,1)$-valued matrix with zeros on its diagonal, and a 1 in entry (i,j) if vertices i and j have an edge between them.

- W, the graph degree matrix, which is the diagonal matrix where w_{ii} is the number of neighbors of vertex i.

- L, the Laplacian matrix, defined as $L = W - A$.

- D, the oriented edge matrix of the graph. Letting $m = |\mathcal{E}|$ be the size of the edge set, D is the $m \times n$ matrix defined as follows. If $(j,k), j < k$ is the ith edge in $\mathcal{E}$, then the ith row of D has a 1 in position j, a -1 in position k, and a 0 everywhere else. Thus the vector Dx encodes the set of pairwise first differences across the edges of the graph.

All of these are sparse matrices—although the graph determines how sparse.

(A) Consider the *Laplacian smoothing* problem:

$$\underset{x \in \mathbb{R}^n}{\text{minimize}} \quad \frac{1}{2}\|y - x\|_2^2 + \frac{\lambda}{2} x^T L x \,, \tag{1}$$

where L is the graph Laplacian. Show that the Laplacian matrix has the alternate representation $L = D^T D$, and therefore that penalty term $x^T L x$ can be rewritten as

$$x^T L x = \|Dx\|_2^2 \,.$$

(B) Show that the solution $\hat{x}$ of the Laplacian-smoothing objective solves a linear system

$$C\hat{x} = b \,,$$

where C and b are matrices you name.

(C) Obtain the data "fmri-z.csv" from the class website. This is a 128x128 cross-sectional slice of data from an fMRI experiment on spatial memory, from the laboratory of Russ Poldrack at Stanford University. The data matrix reflects the underlying grid structure of the graph, i.e. entry (i, j) in the matrix corresponds to site (i, j) on the grid. Make a nice heatmap of the data so you can see the spatial structure.

Now consider the following four algorithms for solving the above linear system:

1. a direct solver that uses a sparse matrix factorization (e.g. sparse Cholesky or sparse LU) of the coefficient matrix C.
2. the Gauss-Seidel method.
3. the Jacobi iterative method.
4. the conjugate gradient method. (Specifically, the conjugate gradient method for solving linear systems. There is also a nonlinear conjugate gradient method, which is not what you want.)

You obviously known about the direct solver, but read up on the other three (all of which are iterative methods). Nocedal and

Wright have an extensive discussion of the conjugate-gradient method in their *Numerical Optimization* book, while the other two are simple enough to be understood from their Wikipedia pages.

Pick the direct solver (method 1) and any *one* of the other three methods—whichever you think makes the most sense for the problem, in light of having read about them all. Implement both these methods *as efficiently as you can*, and use them both to solve the Laplacian smoothing problem for our fMRI data. Make sure you get the same results from both methods, up to small numerical errors. Compare them, both in terms of their speed and their ease of implementation. For now, pick λ by eye. Show a nice picture of the raw data side by side with the denoised $\hat{x}$.

I have given you an R function that constructs the oriented edge matrix D for a two-dimensional grid of size $n_x \times n_y$. You can find it in the file `makeD2_sparse.R`.

Also, you can implement all three iterative methods if you want! Just do one iterative method at a minimum.

Graph fused lasso

Now consider a version of the spatial smoothing problem where we change the ℓ_2 penalty to an ℓ_1 penalty:

$$\underset{x \in \mathbb{R}^n}{\text{minimize}} \quad \frac{1}{2}\|y - x\|_2^2 + \lambda \|Dx\|_1 \,, \tag{2}$$

where we recalled that D is the oriented edge matrix of the graph defined earlier. The penalty term rewards the solution for having small absolute first differences across the edges in the graph. This is called the graph fused lasso, or graph-based total-variation denoising.

Since this doesn't have a closed-form solution, we'll solve this problem via ADMM. There is a fairly obvious ADMM approach, based on rewriting the problem as

$$\begin{aligned} &\underset{x \in \mathbb{R}^n}{\text{minimize}} \quad \frac{1}{2}\|y - x\|_2^2 + \lambda \|r\|_1 \\ &\text{subject to} \quad Dx = r \,. \end{aligned} \tag{3}$$

However, this simple approach turns out to be a lot less efficient than the ADMM described in this paper. (Although I'm biased here, I think Wes Tansey's description in Section 5 of this paper is much clearer.) The more complex ADMM described in these references leads to more efficient updates that avoid the most expensive matrix operations of the "simple" approach.

Feel free to pick either the "simple and slow" ADMM based on equation (3), or the better one described in either of these links. Implement it, and compare the answer you get on the fMRI data to Laplacian smoothing. Again, feel free to do something more clever here like cross-validation, but picking λ by eye is fine here.

Note: if you want to get *really* clever, you can optionally use the "proximal-stacking" ADMM algorithm described in this paper. This will probably require a big investment of time on your part for understanding the approach, but it does lead to a state-of-the-art method for this problem that is *much* faster than either of the algorithms described above. (A related reference is this technical report; in the case of a grid graph like we have here, the approach described here reduces to the Barbera/Sra technique. I'm again biased here, but I think this paper is easier to understand.)