SDS 385 Exercise Set

Kevin Song

November 3, 2016

The Laplacian matrix has the following form (where $d(i)$ is the degree of vertex i):

$$\mathcal{L}_{i,j} = \begin{cases} d(i) & \text{for } i = j \\ -1 & \text{for } i \text{ adjacent to } j \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

If we let D_i denote the ith column of D, then the i, jth element of $D^T D$ is equal to $D_i \cdot D_j$. We now seek to show that $D_i \cdot D_j$ is equivalent to the i, jth element of $\mathcal{L}$ as described in Equation (1).

Proof. From the description in the text, D_i has nnz (number of nonzero elements) equal to the degree of vertex i. Let $nzi(D_i)$ be the nonzero indicies of D_i, that is, if $j \in nzi(D_i)$, then D_{ij} (the j-th element of D_i) is equal to either plus or minus one. Then

$$D_i \cdot D_i = \sum_{j \in nzi(D_i)} D_{ij}^2 = \sum_{j \in nzi(D_i)} 1 = \text{nnz}(D_i) = d(i)$$

So $D^T D_{i,j} = d(i)$ when $i = j$, which matches Equation (1).

Now suppose $i \neq j$. If vertex i and vertex j do not share any edges, then there does not exist a k such that $D_{ik} \neq 0$ and $D_{jk} \neq 0$. Then $D_i \cdot D_j$ is necessarily zero, in agreement with Equation (1)

Now suppose that the vertices share one (and exactly one) edge. Suppose this edge is edge k. Then if $i < j$, $D_{ik} = 1$ and $D_{ji} = -1$, or vice versa if $i > k$. In either case, $D_i^T D_j = -1$. □

Since they are elementwise equal, $\mathcal{L} = D^T D$, and so $x^T \mathcal{L} x = x^T D^T D x = \|Dx\|_2^2$.

This minimization problem can now be expressed as

$$\min_{x \in \mathbb{R}^n} \frac{1}{2}(y - x)^T (y - x) + \frac{\lambda}{2}(Dx)^T (Dx)$$

After some algebraic manipulation, we find that this is equivalent to

$$\min_{x \in \mathbb{R}^n} \frac{1}{2} x^T (\lambda D^T D + I^T I) x - y^T x + \frac{y^T y}{2}$$

Since this is a quadratic form, the minimum is found by solving

$$(\lambda D^T D + I)\hat{x} = y \tag{2}$$

so we choose $b = y$ and $C = (\lambda D^T D + I)$

Comparisons!

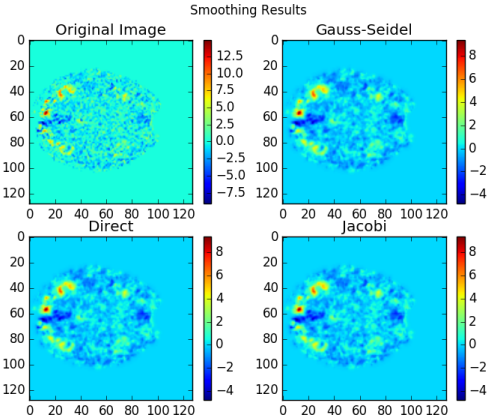
I chose to solve this with a direct solver, Gauss-Seidel, and Jacobi Iteration. Running `solvers.py` in the code directory will run all three solvers, and generate plot comparisons with the original.

In terms of running speed, Gauss-Seidel was hideously slow. Direct solves with SuperLU were decently fast, at just under a second apiece, and Jacobi iteration blew everything else away at about 0.1s per solve. I attribute the issues with Gauss-Seidel to the lack of a good sparse triangular solver in SciPy: without a solver that can take advantage of a triangular matrix to do forward solves, the fastest way to create a solution is to invert the lower triangular matrix, which takes eons because it is does not account for the triangular structure and instead does an LU factorization. The running speeds are summarized in Table 1.

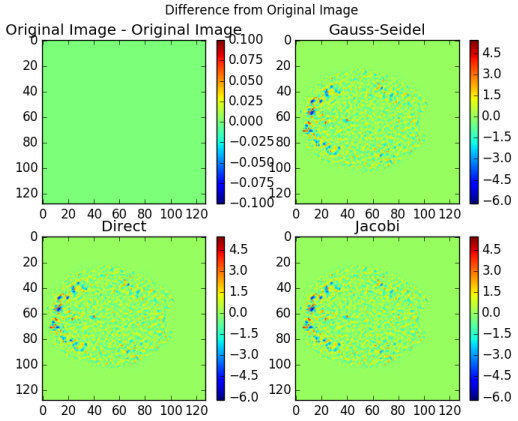
Direct Solve	0.777 seconds
Gauss-Seidel	43.99 seconds
Jacobi Iteration	0.08 seconds

Table 1: Time to solve a single system, averaged over 3 runs

The solutions show a residual of about 10^{-14} from the problem defined in Equation (2), and no greater difference than 10^{-15} from each other elementwise (i.e. the inf norm of the difference of any two solutions is less than 10^{-15}), which suggests that the solutions are mostly identical. A plot of the solutions, shown in Figure 1(a), shows that they are qualitatively the same. The plot in Figure 1(b), of the differences between the smoothed and actual, shows that none of the techniques have exactly the same answer.



(a) A plot of the spatially-smoothed results. Qualitatively, all methods arrive at the same result.



(b) Differences of the spatially-smoothed data from the actual data. Minor differences are present, but they are on the order of 10^{-15}. Crucially, each method gets a different result.

1 Graph fused lasso

Since I spent entirely too much time on the above section, I chose to implement the slow ADMM.

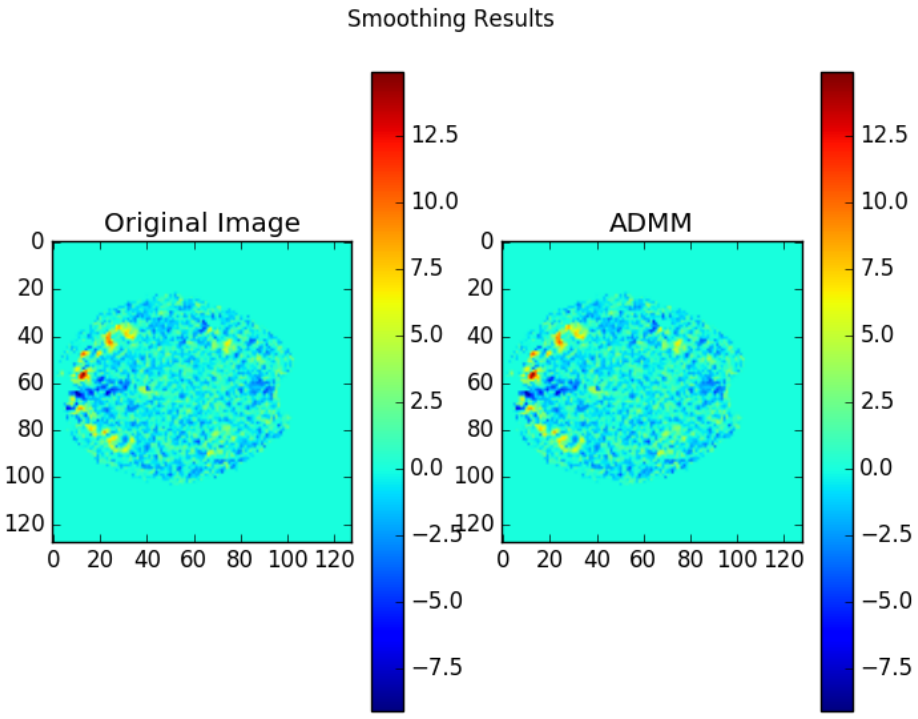


Figure 2: ADMM versus original data