Exercise 8 Spatial Smoothing at scale **Cenying(Tracy) Yang**
SDS 385 Due Date: 11/9/2016
Prof. James Scott

Laplacian Smoothing

(A) **Solution**
(B) **Solution**
Denote objective function as $f(x, \lambda) = \frac{1}{2}\|y - x\|_2^2 + \frac{}{2} x^T Lx$. Taking derivative with respect to x and setting it to zero give

$$\frac{\partial F}{\partial x} = -(y - x) + \lambda Lx = 0$$
$$\Rightarrow (I + \lambda L)\hat{x} = y$$

Therefore, in terms of a linear system, $C = I + \lambda L$, or $C = I + \lambda L = I + \lambda D^T D$, and $b = y$.
(C) **Solution**
A nice reference for Gauss-Seidel method and Jacobi iterative method is ¡here¿. The difference between Gauss-Seidel and Jacobi is that GS uses updated x_i^k to update x_j^k, where $j > i$, while Jacobi uses the update in last iteration for all x update in the current iteration. The comparison of three methods, i.e., sparse matrix factorization, GS, and Jacobi, and the raw data is plotted in Figure 1.

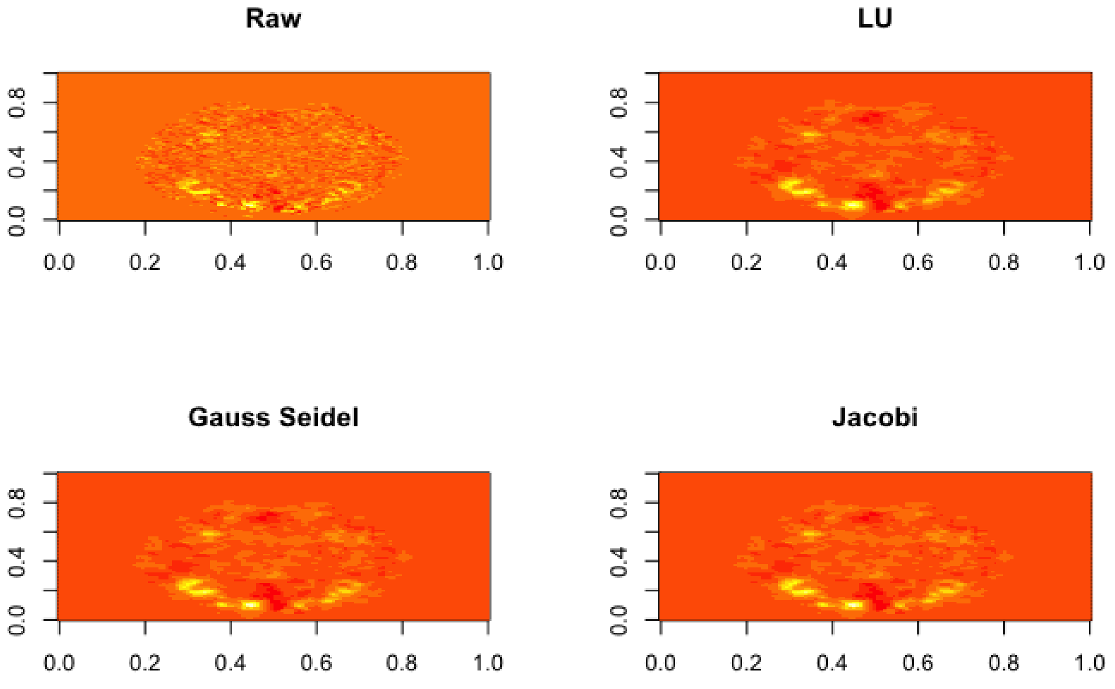


Figure 1: Raw vs LU vs GS vs Jacobi

Graph Fused Lasso

With the ***ordinary version of ADMM*** that we derived in Exercise 7, the problem is

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \frac{1}{2}\|y-x\|_2^2 + \gamma\|z\|_1 \\ \text{subject to} \quad & Dx - z = 0 \end{aligned} \tag{1}$$

The augmented Lagrangian form of problem (1) is

$$L(x,z,\lambda) = \frac{1}{2}\|y-x\|_2^2 + \gamma\|z\|_1 + \frac{\rho}{2}\|Dx - z + \lambda/\rho\|_2^2$$

The algorithm runs as follows

$$\begin{aligned} x^{k+1} &= \arg\min_x L(x, z^k, \lambda^k) \\ z^{k+1} &= \arg\min_z L(x^{k+1}, z, \lambda^{k+1}) \\ \lambda^{k+1} &= \lambda^k + \rho(x^{k+1} - z^{k+1}) \end{aligned}$$

For a ***more efficient version of ADMM***, the problem is

$$\underset{x,z}{\text{minimize}} \quad \frac{1}{2}\|y-x\|_2^2 + \gamma\|z\|_1 + I_C(x,z)$$

where $I_C(x,z)$ takes the value 0 whenever $(x,z) \in C$ and inf otherwise. C is the convex set $C = \{x, z : Dx = z\}$. Introducing two sets of slcack variables (r for x, s for z), we get a new problem

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \frac{1}{2}\|y-x\|_2^2 + \gamma\|z\|_1 + I_C(r,s) \\ \text{subject to} \quad & x - r = 0 \\ & z - s = 0 \end{aligned} \tag{2}$$

The augmented Lagrangian form of the problem (2) in **scaled** form is

$$L(x,z,r,s,t,u) = \frac{1}{2}\|y-x\|_2^2 + \gamma\|z\|_1 + I_C(r,s) + \frac{\rho}{2}\|x - r + u\|_2^2 + \frac{\rho}{2}\|z - s + t\|_2^2$$

The algorithm runs as follows

$$\begin{aligned} x^{k+1} &= \arg\min_x L(x, z^k, r^k, s^k, t^k, u^k) = (I + \rho I)^{-1}(y + \rho r) \\ z^{k+1} &= \arg\min_z L(x^{k+1}, z, r^k, s^k, t^k, u^k) = S_{\gamma/\rho}(s^k - t^k) \\ (r^{k+1}, s^{k+1}) &= \arg\min_{r,s} L(x^{k+1}, z^{k+1}, r, s, t^k, u^k) \\ u^{k+1} &= u^k + x^{k+1} - r^{k+1} \\ t^{k+1} &= t^k + z^{k+1} - s^{k+1} \end{aligned}$$

The joint update (r,s) is the only computationally demanding step of the algorithm. It is equivalent to

$$\begin{aligned} \underset{r,s}{\text{minimize}} \quad & \|r - w\|_2^2 + \|Dr - v\|_2^2 \\ \text{subject to} \quad & s - Dr = 0 \end{aligned} \tag{3}$$

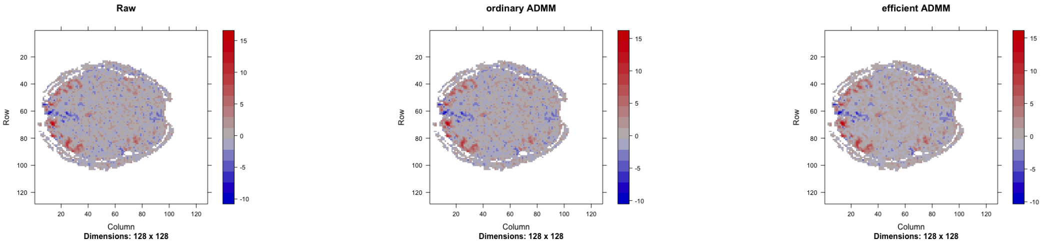
The ordinary first order optimal condition for r in problem (3) is

$$(I + D^T D)r^{k+1} = w + D^T v$$

where $w = x^{k+1} + u^k$ and $v = z^{k+1} + t^k$. After updating r, s can be updated as $s^{k+1} = Dr^{k+1}$. I implemented both ordinary ADMM used in exercise 7, and efficient ADMM described in Tansey's paper. However, the efficient version of ADMM is actually much slower. It takes effect when the inversion involves changing values.

```
Unit: seconds
             expr       min       lq      mean    median       uq       max neval
      admm(y, D)  2.655938  3.05288  3.461831  3.444066  3.80798  4.365099    10
 fast_admm(y, D) 31.455013 32.83340 34.319511 34.070225 34.61189 40.235039    10
```

The results of two algorithms are depicted in Figure 2



(a) raw

(b) ordinary ADMM

(c) raw

Figure 2: ordinary ADMM v.s. efficient ADMM

Class Notes

Denote degree of freedom of predictor $\hat{y}$ as $df(\hat{y})$, where $\hat{y_i} = g_i(y_i)$ and y is the actual data. Define df as

$$df(\hat{y}) = \frac{1}{\sigma^2} \sum_i cov(\hat{y_i}, y_i)$$

where $var(y_i) = \sigma^2$. The intuition is that the bigger the covariance is, the bigger the degree of freedom is. Consider several cases of $g_i(*)$.

- $\hat{y_i} = y_i, \qquad df = n$
- $\hat{y_i} = \bar{y}, \qquad df = 1$
- $\hat{y} = Sy, \qquad df = trace(S)$
- linear squares $y = X\beta + \epsilon$

For $\hat{y} = Sy$, consider the example of linear lease squares.

$$\begin{aligned} & y = X\beta + \epsilon \\ & \hat{\beta} = (X^T X)^{-1} X^T y \\ & \hat{y} = X\hat{beta} = X(X^T X)^{-1} X^T y = P_x y \end{aligned}$$

Then, the degree of freedom can be calculated as $df(\hat{y} = trace(P_x) = dim(range(X)) = p$. What is usually used in practice getting degree of freedom of predictors (estimates) is the **Stein's Lemma**. It is used in lasso fit because it is more easy to compute degree of freedom. In lasso, you have higher degree of freedom (i.e., higher covariance) and pick up the features that matter. While OLS is like a shrinkage version of lasso with lower degree of freedom.

Laplacian method words for nice smooth and continuous objective function, but not for non smooth, discontinuous objective function. Graph fused lasso works better for non smooth case, where the objective function has jumps. That is also the difference between ℓ_2 penalty and ℓ_1 penalty.
To solve fused lasso, the ordinary version of ADMM is inefficient since it has to do matrix factorization every iteration. That gives to the idea of more efficient ADMM.