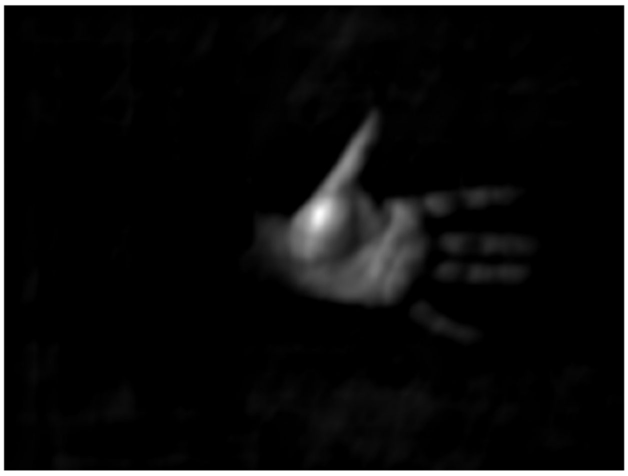
CMSC498D Final Project Report

William Lin

May 14, 2025

Tutorial: ADMM with Total Variation

After reading through the jupyter notebook and running the reconstruction, I got the following:



I understood that the general form of the ADMM algorithm is given by:

$$(\hat{x}, \hat{v}) = \arg\min_{x,v} \; f(x) + \lambda g(v), \quad \text{subject to } x = v$$

So at first, I was confused by the objective function given in the jupyter notebook which was given by

$$\hat{\mathbf{v}} = \arg \min_{w \geq 0,\, u,\, x} \frac{1}{2} \left\| \mathbf{b} - \mathbf{C}x \right\|_2^2 + \tau \left\| u \right\|_1 \quad \text{s.t. } x = \mathbf{M}\mathbf{v},\; w = \mathbf{v}$$

This doesn't really make that clear what the u meant. I then read through more of the resources given on ADMM and saw the more detailed objective function given by

$$\hat{\mathbf{v}} = \arg \min_{w \geq 0,\, u,\, x} \frac{1}{2} \left\| \mathbf{b} - \mathbf{C}x \right\|_2^2 + \tau \left\| u \right\|_1 \quad \text{s.t. } x = \mathbf{M}\mathbf{v},\; u = \Psi\mathbf{v},\; w = \mathbf{v}$$

With this, we see that $f(x) = \frac{1}{2} \left\| \mathbf{b} - \mathbf{C}x \right\|_2^2$ and $g(\mathbf{v}) = \tau \left\| \Psi\mathbf{v} \right\|_1$.

In this implementation of ADMM, we use total variation for regularization. The anisotropic total variation is given by

$$\text{TV}(v) = \|\nabla_x \mathbf{v}\|_1 + \|\nabla_y \mathbf{v}\|_1 = \left\| \begin{bmatrix} \nabla_x \\ \nabla_y \end{bmatrix} \mathbf{v} \right\|_1$$

But notice that $(\nabla_x \mathbf{v})_{i,j} = \mathbf{v}_{i+1,j} - \mathbf{v}_{i,j}$ and $(\nabla_y \mathbf{v})_{i,j} = \mathbf{v}_{i,j+1} - \mathbf{v}_{i,j}$.
We're also given that,

$$\Psi v_{ij} = \begin{bmatrix} \mathbf{v}_{i+1,j} - \mathbf{v}_{i,j} \\ \mathbf{v}_{i,j+1} - \mathbf{v}_{i,j} \end{bmatrix}$$

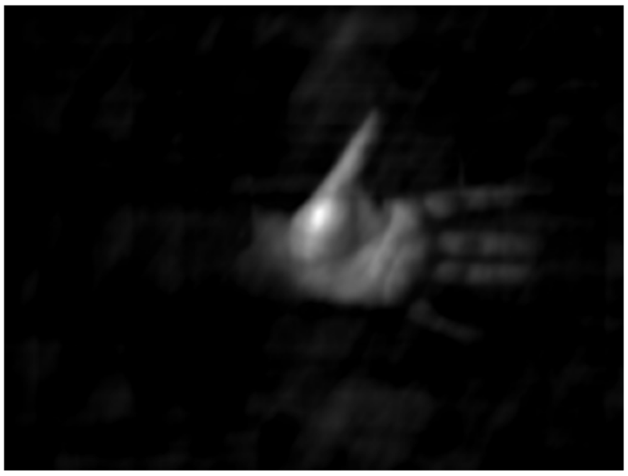
So to put this together,

$$\text{TV}(\mathbf{v}) = \|\Psi\mathbf{v}\|_1 = \sum_{i,j} \|\Psi\mathbf{v}_{i,j}\|_1 = \sum_{i,j} \left(|\mathbf{v}_{i+1,j} - \mathbf{v}_{i,j}| + |\mathbf{v}_{i,j+1} - \mathbf{v}_{i,j}| \right)$$

After following the jupyter notebook's link to Convex Optimization by Boyd, Stephen, and Lieven Vandenberghe, I learned about total variation. Total variation is similar to quadratic smoothing which we've seen in that it removes unwanted noise from images. We can tune the weight of the regularizer by changing τ. The image becomes flat when the weight of the total variation regularizer is increased. Decreased weights for the total variation regularizer leads to less noise being removed. However, total variation is different from quadratic smoothing in that it preserves edges. Much of the "sharp transitions in the signal" are preserved.

ADMM wtih L1 Regularization

The reconstruction result is very similar to the above,



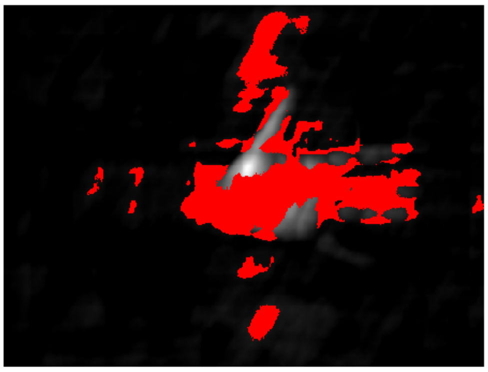
The optimization objective of the ADMM reconstruction using the L1 variant is given by the following:

$$\hat{\mathbf{v}} = \arg \min_{v, w \geq 0,\, u,\, x} \frac{1}{2} \|\mathbf{b} - \mathbf{C}x\|_2^2 + \tau \|u\|_1 \quad \text{s.t. } x = \mathbf{M}\mathbf{v},\ u = \mathbf{v},\ w = \mathbf{v}$$

The update rules are given by the following:

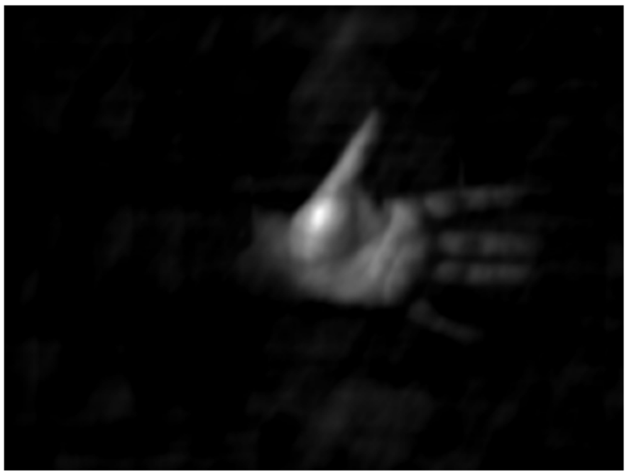
$$\begin{aligned}
u_{k+1} &\leftarrow \mathcal{T}_{\frac{\tau}{\mu_2}}\left(\mathbf{z}_k + \frac{\eta_k}{\mu_2}\right) \\
x_{k+1} &\leftarrow \left(\mathbf{C}^H\mathbf{C} + \mu_1 I\right)^{-1}\left(\xi_k + \mu_1 \mathbf{M}\mathbf{z}_k + \mathbf{C}^H\mathbf{y}\right) \\
w_{k+1} &\leftarrow \max(\rho_k/\mu_3 + \mathbf{z}_k, 0) \\
\mathbf{z}_{k+1} &\leftarrow (\mu_1 \mathbf{M}^H\mathbf{M} + (\mu_2 + \mu_3)I)^{-1} r_k, \\
\xi_{k+1} &\leftarrow \xi_k + \mu_1(\mathbf{A}\mathbf{z}_k - x_{k+1}) \\
\eta_{k+1} &\leftarrow \eta_k + \mu_2(\mathbf{z}_{k+1} - u_{k+1}) \\
\rho_{k+1} &\leftarrow \rho_k + \mu_3(\mathbf{z}_{k+1} - w_{k+1})
\end{aligned}$$

where $r_k = (\mu_3 w_{k+1} - \rho_k) + (\mu_2 u_{k+1} - \eta_k) + \mathbf{M}^H(\mu_1 x_{k+1} - \xi_k)$.
L1 regularization applies a soft threshold over pixels which tends to brings out stronger signals in the image. Although almost the same as using the TV reconstruction, we see in our case that L1 does let through some uncessary light around the hand compared to before. Some differences between the TV reconstruction and L2 reconstruction are highlighted in red below:



ADMM wtih L2 Regularization

The reconstruction result is again very similar,



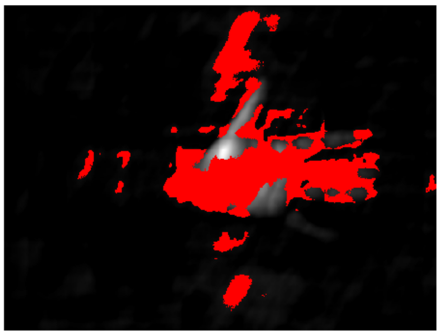
The optimization objective of the ADMM reconstruction using the L2 variant is given by the following:

$$\hat{\mathbf{v}} = \arg \min_{v, w \geq 0,\, u,\, x} \frac{1}{2} \|\mathbf{b} - \mathbf{C}x\|_2^2 + \tau \|u\|_2^2 \quad \text{s.t. } x = \mathbf{M}\mathbf{v},\ u = \mathbf{v},\ w = \mathbf{v}$$

With the following update rules:

$$\begin{aligned}
u_{k+1} &\leftarrow \frac{\mu_2}{\mu_2 + \tau} \left(\mathbf{z}_k + \frac{\eta_k}{\mu_2} \right) \\
x_{k+1} &\leftarrow \left(\mathbf{C}^H \mathbf{C} + \mu_1 I \right)^{-1} \left(\xi_k + \mu_1 \mathbf{M} \mathbf{z}_k + \mathbf{C}^H \mathbf{y} \right) \\
w_{k+1} &\leftarrow \max(\rho_k / \mu_3 + \mathbf{z}_k, 0) \\
\mathbf{z}_{k+1} &\leftarrow (\mu_1 \mathbf{M}^H \mathbf{M} + (\mu_2 + \mu_3) I)^{-1} r_k, \\
\xi_{k+1} &\leftarrow \xi_k + \mu_1 (\mathbf{A} \mathbf{z}_k - x_{k+1}) \\
\eta_{k+1} &\leftarrow \eta_k + \mu_2 (\mathbf{z}_{k+1} - u_{k+1}) \\
\rho_{k+1} &\leftarrow \rho_k + \mu_3 (\mathbf{z}_{k+1} - w_{k+1})
\end{aligned}$$

where $r_k = (\mu_3 w_{k+1} - \rho_k) + (\mu_2 u_{k+1} - \eta_k) + \mathbf{M}^H (\mu_1 x_{k+1} - \xi_k)$.
Unlike L1 regularization, L2 regularization proportionally reduces values and supresses most noises. However as we've seen in class, this regularization variant is not good at preserving sharp edges. In our case, there were that many strong transformations between edges so it wasn't a problem. Like the L1 reconstruction, the L2 construction also let some uncessary light through. The differences between the TV reconstruction and L2 reconstruction are highlighted in red below:

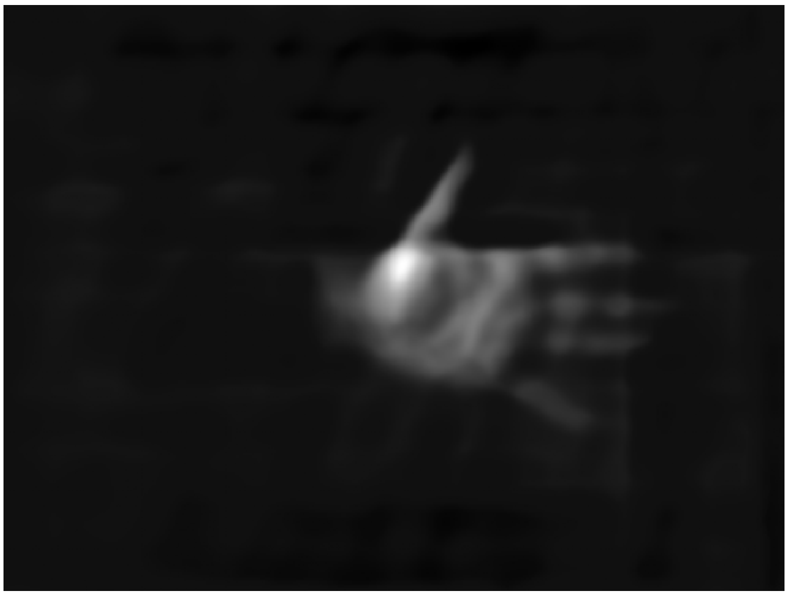


Closed Formula Update

For this section, I had read the paper on the Plug and Play ADMM closely. Mostly trying to understand the different update rules and general algorithm which was given in both pseudocode and MatLab so it was fairly simple to implement.
I found the following update rules to be useful:

$$\begin{aligned}
\mathbf{x}^{(k+1)} &= \arg\min_{\mathbf{x}} f(\mathbf{x}) + (\rho_k/2) \left\| \mathbf{x} - \left(\mathbf{v}^{(k)} - \mathbf{u}^{(k)} \right) \right\|^2, \\
\mathbf{v}^{(k+1)} &= \mathcal{D}_{\sigma_k} \left(\mathbf{x}^{(k+1)} + \mathbf{u}^{(k)} \right), \\
\mathbf{u}^{(k+1)} &= \mathbf{u}^{(k)} + \left(\mathbf{x}^{(k+1)} - \mathbf{v}^{(k+1)} \right), \\
\rho_{k+1} &= \gamma_k \rho_k,
\end{aligned}$$

I got the following results:



Conjugate Gradient Update

For this section, some outside research was require for me to implement the code. I understood that only the updating for x was necessary since we were going to use Conjugate Gradient process. However, I didn't really remember learning about this concept in class. So I read up on Some history on the conjugate gradient and Lanczoz algorithm and Methods of Conjugate Gradients for Solving Linear Systems which helped me get the following update rules:

$$\begin{aligned}
p_0 &= r_0 = k - Ax_0 \quad (x_0 \text{ arbitrary}) \\
a_i &= \frac{|r_i|^2}{(p_i \cdot Ap_i)} \\
x_{i+1} &= x_i + a_i p_i \\
r_{i+1} &= r_i - a_i A p_i \\
b_i &= \frac{|r_{i+1}|^2}{|r_i|^2} \\
p_{i+1} &= r_{i+1} + b_i p_i
\end{aligned}$$

After updating the code from the previous section, I got the following reconstruction result: