ADMM Tutorial [1]

赖泽强

2020 年 12 月 28 日

[1]Draft version. Working in progress.

目录

Chapter 1

Introduction

ADMM 是针对优化问题的一种解法。优化问题则是说，我们希望在给定一些约束的情况下，去寻找一个解来最小化一个我们定义的目标函数。这个过程可以形式化地定义为：

$$P : \min_{x \in D} f(x)$$

其中 f 是目标函数，D 是由约束条件划定的 x 的取值范围。

ADMM 尝试解决的则是一种特殊的优化问题。在这个问题中，我们的目标函数是一个均方误差，约束条件则是一个 L1 范数。我们使用朗格朗日法将有约束问题转化为无约束问题，就变成了公式1.1所示的形式。

$$\min_{x} \frac{1}{2} \sum_{i=1}^{n} (y_i - x_i)^2 + \lambda \sum_{(i,j) \in E} |x_i - x_j| \tag{1.1}$$

通常我们将这个问题称为 **2d fused lasso** 或 **2d total variation denoising** 问题

我们可以使用各种各样的优化算法来解这个特殊的问题，例如 Proximal gradient descent，Coordinate descent，但 ADMM 是这些算法中收敛最快的（即需要迭代次数少）[1]。

1.1 Vanilla ADMM

那么 ADMM 是怎么做的呢？ADMM 先是引入了一个新的变量 v，并约束 $x = v$，然后解公式1.1所示的无约束问题，就变成了解公式1.2所示的有约束问题。

$$(\widehat{\boldsymbol{x}}, \widehat{\boldsymbol{v}}) = \underset{\boldsymbol{x}, \boldsymbol{v}}{\operatorname{argmin}} \quad f(\boldsymbol{x}) + \lambda g(\boldsymbol{v}), \quad \text{subject to } \boldsymbol{x} = \boldsymbol{v} \tag{1.2}$$

然后我们用增广朗格朗日法再将其变成无约束问题，变成公式3.3的形式[2]。

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{v}, \boldsymbol{u}) = f(\boldsymbol{x}) + \lambda g(\boldsymbol{v}) + \boldsymbol{u}^T(\boldsymbol{x} - \boldsymbol{v}) + \frac{\rho}{2}\|\boldsymbol{x} - \boldsymbol{v}\|^2 \tag{1.3}$$

优化这个方程可以使用分步优化的方法，即先选取一个优化变量，然后固定其它变量，对刚刚选取的变量进行优化，依次选取所有需要优化的变量重复进行。这个过程可以用公式1.4描述。

[1]需要注意的是，ADMM 在这个问题上快，不代表它在其它问题上也快。

[2]为什么要变成这种形式？一个直观的解释是新的函数更凸，而凸函数在优化是具有很好的性质，如收敛快，更容易获得更优解等。参见：交替方向乘子法（ADMM）算法的流程和原理是怎样的？- 大大大的 v 的回答 - 知乎 https://www.zhihu.com/question/36566112/answer/118715721

$$
\begin{aligned}
\boldsymbol{x}^{(k+1)} &= \underset{\boldsymbol{x}\in\mathbb{R}^n}{\operatorname{argmin}} \quad f(\boldsymbol{x}) + \tfrac{\rho}{2}\left\|\boldsymbol{x} - \tilde{\boldsymbol{x}}^{(k)}\right\|^2 \\
\boldsymbol{v}^{(k+1)} &= \underset{\boldsymbol{v}\in\mathbb{R}^n}{\operatorname{argmin}} \quad \lambda g(\boldsymbol{v}) + \tfrac{\rho}{2}\left\|\boldsymbol{v} - \tilde{\boldsymbol{v}}^{(k)}\right\|^2 \\
\overline{\boldsymbol{u}}^{(k+1)} &= \overline{\boldsymbol{u}}^{(k)} + \left(\boldsymbol{x}^{(k+1)} - \boldsymbol{v}^{(k+1)}\right)
\end{aligned}
\tag{1.4}
$$

其中第三个优化 u 的式子，我们是要让 u 最大，用这种方式强迫 x 和 v 更接近。然后因为我们求导有解析解，我们可以直接使用梯度上升法。

至于 ADMM 这种做法为什么会获得更快的收敛速度，我还没有深入研究。

1.2 Plug-and-Play ADMM

对于公式1.4里的第二个式子，定义 $\sigma = \sqrt{\lambda\rho}$，我们可以将其改写成：

$$
\boldsymbol{v}^{(k+1)} = \underset{\boldsymbol{v}\in\mathbb{R}^n}{\operatorname{argmin}} \quad g(\boldsymbol{v}) + \frac{1}{2\sigma^2}\left\|\boldsymbol{v} - \tilde{\boldsymbol{v}}^{(k)}\right\|^2 \tag{1.5}
$$

直观的，我们可以把这个优化过程看出一个降噪的过程，其中 σ 是高斯噪声的强度 (我们假设噪声是高斯噪声)。我们可以把 v 当成降噪后的图像，v^k 看出带噪声的图像。$g(v)$ 是说我们降噪后的图像要是一个图像（满足先验 g)，后面一项则是说降噪后的图像和原图像要接近。

因此，我们可以使用一个降噪器去替代这个优化过程。每一步优化，我们都将 v^k 输入一个降噪器获得 v^{k+1}。

具体为什么说这个形式很像降噪，我们需要先回顾一下降噪的优化算法是什么样的。

对于一个降噪问题，我们形式化为如下的优化问题：

$$
\widehat{\boldsymbol{x}}_{\text{map}} = \arg\max_{\boldsymbol{x}} p(\boldsymbol{x} \mid \boldsymbol{y}) \tag{1.6}
$$

其中 y 是输入的噪声图像，x 是去噪图像。

使用贝叶斯公式，加负号，我们可以将其转换成如下的优化问题：

$$
\widehat{\boldsymbol{x}}_{\text{map}} = \arg\min_{\boldsymbol{x}}\{-\ln p(\boldsymbol{y} \mid \boldsymbol{x}) - \ln p(\boldsymbol{x})\} \tag{1.7}
$$

如果我们假设噪声是高斯噪声，那么 $e = y - x$ 应该服从正态分布, 即 $\boldsymbol{e} \sim \mathcal{N}\left(\boldsymbol{0}, \sigma_e^2\boldsymbol{I}\right)$。

因为 $p(y|x)$ 是给定原始图像，噪声图像出现的概率，既然我们知道噪声的概率分布，那 $p(y|x)$ 事实上应该就是噪声出现的概率。因此我们可以将正态分布的公式代入，求解出 $-lnp(y|x)$：

$$
-logP(y|x) = -log(\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{(y-x)^2}{2\sigma}}) = \frac{(y-x)^2}{2\sigma} + log(\sigma\sqrt{2\pi}) \propto \frac{(y-x)^2}{2\sigma}
$$

到这里，就不难看出为什么我们说 ADMM 优化 v 的步骤可以看成一个降噪过程了，对比公式1.5和公式1.7，前者的 g(v) 就相当于后者的 $-lnp(x)$，前者的后一个平方差项和后者则是完全一致的。

1.3 Tuning Free PnP ADMM

在 PnP ADMM 中，观察公式1.4，我们知道这个算法存在两个超参数 ρ 和 σ。Tuning Free PnP ADMM[1] 这个算法就是使用强化学习的方法去自动寻找**每一步迭代**最适合的参数。

因为每一步迭代都可以使用不同的超参数，因此有时候可以获得比人工调参，甚至穷举[3]更优的结果。

当然，这个算法最大的好处还是不用自动化了调参的过程。

[3]穷举是指在最开始穷举得到一个最优参数，但每一步迭代参数相同，因为每一步迭代都穷举并不现实。

Chapter 2

Background

为了能够读懂后面的 Examples, 你可能需要补充一点背景知识。如果你对这些知识很熟悉, 你可以选择跳过这一章节。如果你不了解这些概念，那也没有关系，你可以选择先看 Examples，遇到不懂的时候再回来翻阅这部分内容。

2.1 Least square problem

2.2 Fourier Transform

2.3 Circular Matrix

2.4 Convolution Theorem

Chapter 3

Examples

3.1 ADMM 1D TV Denosing

3.1.1 公式推导

1D TV Denosing 问题描述如下，其中 D 是一个 Difference matrix，主对角线全是 1，主对角线上方元素是-1。

$$\text{minimize} \quad \frac{1}{2}\|x-y\|_2^2 + \lambda\|Dx\|_1 \tag{3.1}$$

ADMM 形式：

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}\|x-y\|_2^2 + \lambda\|z\|_1 \\ \text{subject to} & Dx - z = 0 \end{array} \tag{3.2}$$

增广朗格朗日形式：

$$L_\rho(x,z,\nu) = \frac{1}{2}\|x-y\|_2^2 + \lambda\|z\|_1 + \nu^T(Dx-z) + \frac{\rho}{2}\|Dx-z\|_2^2 \tag{3.3}$$

令 $\mu = \nu/\rho$，可以验证：

$$\nu^T(Dx-z) + \frac{\rho}{2}\|Dx-z\|_2^2 = \frac{\rho}{2}\|Dx-z+\mu\|_2^2 - \frac{\rho}{2}\|\mu\|_2^2 \tag{3.4}$$

因此，新的增广朗格朗日形式可以写成：

$$L_\rho(x,z,\nu) = \frac{1}{2}\|x-y\|_2^2 + \lambda\|z\|_1 + \frac{\rho}{2}\|Dx-z+\mu\|_2^2 - \frac{\rho}{2}\|\mu\|_2^2 \tag{3.5}$$

因此，ADMM 的分布优化步骤为：

$$\begin{aligned} x^{(k+1)} &= \arg\min_x \left(\frac{1}{2}\|x^{(k)}-y\|_2^2 + \frac{\rho}{2}\left\|Dx^{(k)} - z^{(k)} + \mu^{(k)}\right\|_2^2\right) \\ z^{(k+1)} &= \arg\min_z \left(\lambda\|z^{(k)}\|_1 + \frac{\rho}{2}\left\|Dx^{(k+1)} - z^{(k)} + \mu^{(k)}\right\|_2^2\right) \\ \nu^{(k+1)} &= \nu^{(k)} + Dx^{(k+1)} - z^{(k+1)} \end{aligned} \tag{3.6}$$

这三个优化步骤都有解析解，如下 (牢记：$\mu = \nu/\rho$)：

$$\begin{aligned} x^{k+1} &:= \left(I + \rho D^T D\right)^{-1}\left(y + \rho D^T\left(z^k - \mu^k\right)\right) \\ z^{k+1} &:= T_{\lambda/\rho}\left(Dx^{k+1} + \mu^k\right) \\ \nu^{k+1} &:= \nu^k + Dx^{k+1} - z^{k+1} \end{aligned} \tag{3.7}$$

具体含义和推导见下：

求解 x: 优化 x 等价于求解一个最小二乘问题。推导如下：

最小二乘法的目标函数为：

$$L(\vec{\beta}) = \|X\vec{\beta} - Y\|^2 \tag{3.8}$$

有解析解：

$$(X^T X)^{-1} X^T Y \tag{3.9}$$

改写公式3.6中 x 的目标函数：

$$x^{(k+1)} = \arg\min_x \left(\|x - y\|_2^2 + \left\| \sqrt{\rho} Dx - \sqrt{\rho}(z^{(k)} - \mu^{(k)}) \right\|_2^2 \right) \tag{3.10}$$

我们把这两个二范数合起来写成矩阵形式：

$$\min_x \left\| \begin{bmatrix} I \\ \sqrt{\rho} D \end{bmatrix} x - \begin{bmatrix} y \\ \sqrt{\rho}\left(z^{(k)} - \mu^{(k)}\right) \end{bmatrix} \right\|_2^2 \tag{3.11}$$

把 x 当成 β, 把式子前面的矩阵当成 X，把后面的矩阵当成 Y，可知，优化 x 等价于一个最小二乘问题。

代入最小二乘法的解析解公式：

$$\begin{aligned} x^{(k+1)} &= \left(I + \rho D^T D\right)^{-1} \left[I, \sqrt{\rho} D^T\right] \begin{bmatrix} y \\ \sqrt{\rho}\left(z^{(k)} - \mu^{(k)}\right) \end{bmatrix} \\ &= \left(I + \rho D^T D\right)^{-1} \left(y + \rho D^T \left(z^{(k)} - \mu^{(k)}\right)\right) \end{aligned} \tag{3.12}$$

求解 z: z 是一个一维向量，将 z 的目标函数展开，令 $v = Dx + \mu$, 我们可以得到：

$$\begin{aligned} &\underset{z}{\text{minimize}} \ \lambda \sum_{n=1}^{N} |z[n]| + \frac{\rho}{2} \sum_{n=1}^{N} (z[n] - v[n])^2 \\ =& \text{minimize}_z \ \sum_{n=1}^{N} \left(\lambda |z[n]| + \frac{\rho}{2} (z[n] - v[n])^2 \right) \end{aligned} \tag{3.13}$$

因为 z 的每一个分量没有关系，我们可以单独优化 z 的每一个分量：

$$\underset{z \in R}{\text{minimize}} \ \lambda |z| + \frac{\rho}{2} (z - v)^2 \tag{3.14}$$

这个函数除了 0 处处可导，且是个凸函数，导数为

$$\frac{df}{dz} = \begin{cases} \lambda + z - \rho v, & z > 0 \\ -\lambda + z - \rho v, & z < 0 \end{cases} \tag{3.15}$$

凸函数的极值点就是最值点，因此 z 的最优解为导数为 0 的地方。当 $|v| > \lambda/\rho$ 时，导数可以取到 0；反之，z 等于 0 时取到导数绝对值最小的位置，即最优解。

$$z^\star = \begin{cases} \rho v - \lambda, & v > \lambda/\rho \\ 0, & |v| \le \lambda/\rho \\ \rho v + \lambda, & v < -\lambda/\rho \end{cases} \tag{3.16}$$

用 $T_\lambda(\cdot)$ 表示这个函数，则

$$z^{(k+1)} = T_{\lambda/\rho}(v) = T_{\lambda/\rho}(Dx + \mu) \tag{3.17}$$

$T_\lambda(\cdot)$ 被称为 **soft thresholding** 或 **shrinkage** operator。

求解 ν: 使用公式3.3求导即可：

$$\frac{d\nu}{dL} = (Dx - z)/\rho \tag{3.18}$$

ν 要最大化，使用梯度上升法。

3.1.2 DFT 优化

在每一步优化 x 的时候，我们需要求 $I + \rho D^T D$ 的逆，假设输入 y 是一个 n 维向量，则 $I + \rho D^T D$ 将是一个 $n * n$ 的矩阵，当 n 很大的时候，求它的逆是很慢的。

$$x^{(k+1)} = \left(I + \rho D^T D\right)^{-1} \left(y + \rho D^T \left(z^{(k)} - \mu^{(k)}\right)\right) \tag{3.19}$$

对这个式子做一个简单的变换，将 $\left(I + \rho D^T D\right)^{-1}$ 移到等式左边。

$$\left(I + \rho D^T D\right) x^{(k+1)} = \left(y + \rho D^T \left(z^{(k)} - \mu^{(k)}\right)\right) \tag{3.20}$$

假设 $\mathcal{F}$ 是 Fourier matrix，$\mathcal{F}^{-1}$ 是 inverse Fourier matrix，对等式两边同时做傅里叶变换[1]：

$$\left(\mathcal{F}I + \rho \mathcal{F} D^T D\right) x^{(k+1)} = \left(\mathcal{F}y + \rho \mathcal{F} D^T \left(z^{(k)} - \mu^{(k)}\right)\right) \tag{3.21}$$

对 Fourier matrix，有 $\mathcal{F}^{-1}F = \mathcal{F}^H\mathcal{F} = I$。使用这个性质对上个式子再做一次变换, 我们有：

$$\begin{aligned} \left(\mathcal{F}I + \rho \mathcal{F} D^T D\right) \mathcal{F}^H \mathcal{F} x^{(k+1)} &= LHS \\ => \left(\mathcal{F}I\mathcal{F}^H + \rho \mathcal{F} D^T D \mathcal{F}^H\right) \mathcal{F} x^{(k+1)} &= LHS \end{aligned} \tag{3.22}$$

这里需要用到一个性质，Fourier matrix 可以对角化所有的 circulant matrix，并且 F^H 的列向量就是这些 circulant matrix 的特征向量，特征值则是生成 circulant matrix 的信号的 DFT 结果。

$$\begin{aligned} \mathcal{F} D \mathcal{F}^H &= \Lambda \\ \mathcal{F} D^T \mathcal{F}^H &= \Lambda^H \\ \mathcal{F} D^T D \mathcal{F}^H &= \mathcal{F} D^T \mathcal{F}^H \mathcal{F} D \mathcal{F}^H = \Lambda^H \Lambda \end{aligned} \tag{3.23}$$

使用上述性质，结合 $\mathcal{F}I\mathcal{F}^H = I$，我们有：

$$\left(I + \rho \Lambda^H \Lambda\right) \mathcal{F} x^{(k+1)} = \left(\mathcal{F}y + \rho \mathcal{F} D^T \left(z^{(k)} - \mu^{(k)}\right)\right) \tag{3.24}$$

左右两边左乘 $\left(I + \rho \Lambda^H \Lambda\right)^{-1}$，再同时做逆傅里叶变换，有：

$$x^{(k+1)} = \mathcal{F}^{-1} \left(I + \rho \Lambda^H \Lambda\right)^{-1} \left(\mathcal{F}y + \rho \mathcal{F} D^T \left(z^{(k)} - \mu^{(k)}\right)\right) \tag{3.25}$$

因为 $I + \rho \Lambda^H \Lambda$ 是对角矩阵，它的逆就是对角线各元素取逆。并且一个对角矩阵和向量相乘等价于对角线元素和向量各元素的 element-wise 的乘法。因此，上式也可以写成：

[1]这里用到了傅里叶变换的矩阵乘表示，以及傅里叶变换的线性性质。

$$x^{(k+1)} = \mathcal{F}^{-1} \frac{\left(\mathcal{F}y + \rho \mathcal{F} D^T \left(z^{(k)} - \mu^{(k)}\right)\right)}{vec\left(I + \rho \Lambda^H \Lambda\right)} \tag{3.26}$$

其中 vec 表示将对角线元素组成一个向量，除法则是 element-wise 除法。

至此，DFT 优化的推导就完成了。通过 DFT，我们将时域上十分耗时的矩阵逆操作转换成了频率域上的除法操作。当然，我们还需要额外的时间来求解 $D^T D$ 的特征值，但因为这个值在迭代过程中是不变的，我们只需要求一次，而且这个值也可以通过傅立叶变换快速求解，因此额外开销其实很小。

3.1.3 代码实现

代码实现的难点和容易产生困惑的地方主要在 x 的优化上。

$D^T D$ 特征值的计算 ：

回顾 Fourier matrix 可以对角化所有的 circulant matrix，并且 F^H 的列向量就是这些 circulant matrix 的特征向量，特征值则是生成 circulant matrix 的信号的 DFT 结果。

假设我们的卷积核 $d = [1, -1]$，则 D 的特征值 Λ 可以使用下述 Python 代码进行计算：

```
eigD = np.fft.fft([1, -1], length)
```

fft 在计算的时候先对 d 进行了一个 padding，padding 到和 y 一个长度再进行傅立叶变换，变换完之后得到一个一维向量，这个向量就是 $vec(\Lambda)$[2]。

因为 $eig(D^T D) = \Lambda^H \Lambda$, 且对一个复数，我们有 $a^H a = ||a||_2^2$，因此我们可以使用下述代码计算 $eig(D^T D)$：

```
eigDtD = abs(np.fft.fft([1, -1], length)) ** 2
```

$\left(I + \rho \Lambda^H \Lambda\right)$ 的计算 ：

这个公式本来是一个单位阵加上一个特征值方阵，但因为我们代码求的特征值其实是放在一个向量里，所以我们可以直接用 element-wise 的加和乘来计算这个式子：

```
lhs = 1 + rho * eigFtF
```

$FD^T(z - \mu)$ 的计算 ：

普通的 FDx 就是用 D 对应的卷积核 h 对 x 做一个 circular convolution，但 D^T 对应的卷积核是什么呢?

在这个回答里https://dsp.stackexchange.com/a/64587提到了 D^T 对应的卷积核是原卷积核的 flipped version，但经过实验，事实上并不完全是这样，除了 flip 卷积核，我们还需要 circular shift 一下 x 才可以。伪代码如下：

```
H*x = ifft(fft(x).*fft(kernel, row, col));
Ht*x = ifft(fft(circshift(x,[0,-1])).*fft(rot90(kernel,2), row, col));
```

对于一维向量，如果 kernel 是行向量，则 shift 的大小是 [0,-1]，如果是列向量，则是 [-1,0]。

3.1.4 参数设置

ADMM 1D TV Denosing 就两个超参数 λ 和 ρ ，参数设置基本不会影响大致结果，但是要注意的是 ρ 不要小于 1，否则可能会造成 x,z 更新之后过大，出现 INF。

[2]没有证明，如果觉得不可思议，可以看下附带的 Jupyter Notebook 确认一下。

3.2 ADMM 2D TV Denosing

对于 2D TV Denosing 问题, 我们有如下优化目标,

$$\text{minimize} \quad \frac{1}{2}\|x-y\|_2^2 + \lambda\|D_r x\|_1 + \lambda\|D_c x\|_1 \tag{3.27}$$

其中 x 是向量化[3]后的优化变量, y 是向量化后的原始带噪声图像。2 维的 TV 约束是横向梯度和纵向梯度的 1 范数，我们可以用两个卷积来求梯度，而 D_r 和 D_c 则是将二维卷积用矩阵乘法表示的 doubly block circulant matrices.

用 z_* 替换 $D_* x$，并添加两个新的约束，可以得到 ADMM 形式：

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}\|x-y\|_2^2 + \lambda\|z_r\|_1 + \lambda\|z_c\|_1 \\ \text{subject to} & D_r x - z_r = 0 \\ \text{subject to} & D_c x - z_c = 0 \end{array} \tag{3.28}$$

用增广朗格朗日法去掉约束：

$$\begin{aligned} L_\rho(x, z_r, \nu_r, z_c, \nu_c) = \frac{1}{2}\|x-y\|_2^2 + \lambda\|z_r\|_1 + \nu_r^T(D_r x - z_r) + \frac{\rho}{2}\|D_r x - z_r\|_2^2 \\ + \lambda\|z_c\|_1 + \nu_c^T(D_c x - z_c) + \frac{\rho}{2}\|D_c x - z_c\|_2^2 \end{aligned} \tag{3.29}$$

与 1D TV 类似 (见公式 3.4)，令 $\mu_r = \nu_r/\rho$，$\mu_c = \nu_c/\rho$，我们可以将上述公式改写成下面的形式,。

$$\begin{aligned} L_\rho(x, z_r, \nu_r, z_c, \nu_c) = \frac{1}{2}\|x-y\|_2^2 + \lambda\|z_r\|_1 + \frac{\rho}{2}\|D_r x - z_r + \mu_r\|_2^2 - \frac{\rho}{2}\|\mu_r\|_2^2 \\ + \lambda\|z_c\|_1 + \frac{\rho}{2}\|D_c x - z_c + \mu_c\|_2^2 - \frac{\rho}{2}\|\mu_c\|_2^2 \end{aligned} \tag{3.30}$$

因此，ADMM 的分布优化步骤为：

$$\begin{aligned} x^{(k+1)} &= \arg\min_x \left(\frac{1}{2}\|x^{(k)} - y\|_2^2 + \frac{\rho}{2}\left\|D_r x^{(k)} - z_r^{(k)} + \mu_r^{(k)}\right\|_2^2 + \frac{\rho}{2}\left\|D_c x^{(k)} - z_c^{(k)} + \mu_c^{(k)}\right\|_2^2 \right) \\ z_r^{(k+1)} &= \arg\min_z \left(\lambda\|z_r^{(k)}\|_1 + \frac{\rho}{2}\left\|D_r x^{(k+1)} - z_r^{(k)} + \mu_r^{(k)}\right\|_2^2 \right) \\ z_c^{(k+1)} &= \arg\min_z \left(\lambda\|z_c^{(k)}\|_1 + \frac{\rho}{2}\left\|D_c x^{(k+1)} - z_c^{(k)} + \mu_c^{(k)}\right\|_2^2 \right) \\ \nu_r^{(k+1)} &= \nu_r^{(k)} + D_r x^{(k+1)} - z_r^{(k+1)} \\ \nu_c^{(k+1)} &= \nu_c^{(k)} + D_c x^{(k+1)} - z_c^{(k+1)} \end{aligned} \tag{3.31}$$

3.2.1 x subproblem

与 1D TV 一样，优化 x 可以看成是一个最小二乘问题。改写公式3.31中 x 的目标函数，我们有：

$$\begin{aligned} x^{(k+1)} = \arg\min_x \ \|x^{(k)} - y\|_2^2 + \left\|\sqrt{\rho}D_r x^{(k)} - \sqrt{\rho}(z_r^{(k)} - \mu_r^{(k)})\right\|_2^2 \\ + \left\|\sqrt{\rho}D_c x^{(k)} - \sqrt{\rho}(z_c^{(k)} - \mu_c^{(k)})\right\|_2^2 \end{aligned} \tag{3.32}$$

写成矩阵形式：

[3]向量化即按 row major order 将二维矩阵化为一维向量。

$$\min_x \left\| \begin{bmatrix} I \\ \sqrt{\rho}D_r \\ \sqrt{\rho}D_c \end{bmatrix} x^{(k)} - \begin{bmatrix} y \\ \sqrt{\rho}\left(z_r^{(k)} - \mu_r^{(k)}\right) \\ \sqrt{\rho}\left(z_c^{(k)} - \mu_c^{(k)}\right) \end{bmatrix} \right\|_2^2 \tag{3.33}$$

代入 $(X^TX)^{-1}X^TY$，得解析解：

$$\begin{aligned} x^{(k+1)} &= \left(I + \rho(D_r^TD_r + D_c^TD_c)\right)^{-1} \left[I, \sqrt{\rho}D_r^T, \sqrt{\rho}D_c^T\right] \begin{bmatrix} y \\ \sqrt{\rho}\left(z_r^{(k)} - \mu_r^{(k)}\right) \\ \sqrt{\rho}\left(z_c^{(k)} - \mu_c^{(k)}\right) \end{bmatrix} \\ &= \left(I + \rho(D_r^TD_r + D_c^TD_c)\right)^{-1} \left(y + \rho\left[D_r^T\left(z_r^{(k)} - \mu_r^{(k)}\right) + D_c^T\left(z_c^{(k)} - \mu_c^{(k)}\right)\right]\right) \end{aligned} \tag{3.34}$$

3.2.2 DFT Speedup

与一维类似，我们可以使用傅立叶变换进行优化，这里只列推导结果：

$$\begin{aligned} \left(\mathcal{F}I + \rho(\mathcal{F}D_r^TD_r + \mathcal{F}D_c^TF_c)\right)\mathcal{F}^H\mathcal{F}x^{(k+1)} &= LHS \\ => \left(\mathcal{F}I\mathcal{F}^H + \rho(\mathcal{F}D_r^TD_r\mathcal{F}^H + \mathcal{F}D_c^TF_c\mathcal{F}^H)\right)\mathcal{F}x^{(k+1)} &= LHS \end{aligned} \tag{3.35}$$

$$\left(I + \rho(\Lambda_r + \Lambda_c)\right)\mathcal{F}x^{(k+1)} = \left(\mathcal{F}y + \rho\left[\mathcal{F}D_r^T\left(z_r^{(k)} - \mu_r^{(k)}\right) + \mathcal{F}D_c^T\left(z_c^{(k)} - \mu_c^{(k)}\right)\right]\right) \tag{3.36}$$

、

$$x^{(k+1)} = \mathcal{F}^{-1}\frac{\left(\mathcal{F}y + \rho\left[\mathcal{F}D_r^T\left(z_r^{(k)} - \mu_r^{(k)}\right) + \mathcal{F}D_c^T\left(z_c^{(k)} - \mu_c^{(k)}\right)\right]\right)}{vec\left(I + \rho(\Lambda_r + \Lambda_c)\right)} \tag{3.37}$$

3.2.3 Implementation

2 维的 ADMM TV Denosing 实现和一维类似，但需要注意的是，虽然公式里是将二维矩阵向量化进行推导，但我们在计算的时候并不需要这么做，我们可以直接在原二维矩阵上做运算。

$eig(D_r^TD_r + D_c^TD_c)$ 的计算 ：

同样的，我们使用 DFT 来加速 $\left(I + \rho(D_r^TD_r + D_c^TD_c)\right)^{-1}$ 的计算：

```
eigDtD = np.abs(np.fft.fft2(np.array([[1, -1]]), (row, col))) ** 2 + \
         np.abs(np.fft.fft2(np.array([[1, -1]]).transpose(), (row, col))) ** 2
```

在这里，我们首先使用 fft2 求特征值。在原公式里，我们其实是要求一个 $[row * col, row * col]$ 方阵，方阵的对角线上有 row*col 个特征值，然后我们需要将这个方阵取逆后与一个 row*col 长度的向量[4]相乘。这个过程其实就是将原二维矩阵中的每个元素都除以一个它对应的特征值。

fft2 经过 padding 之后可以求出一个和原二维矩阵同样大小的矩阵，这个矩阵中的每个元素就是原二维矩阵元素对应的特征值[5]。

$\mathcal{F}D_r^T\left(z_r^{(k)} - \mu_r^{(k)}\right) + \mathcal{F}D_c^T\left(z_c^{(k)} - \mu_c^{(k)}\right)$ 的计算 ：

参考一维情况，$FD^T(z - \mu)$ 是一个二维卷积操作，D^T 对应的卷积核是 D 对应的卷积核的 flipped version，输入同样需要 circular shift，shift 的大小是 [-1,-1]。

[4]即 $\left(\mathcal{F}y + \rho\left[\mathcal{F}D_r^T\left(z_r^{(k)} - \mu_r^{(k)}\right) + \mathcal{F}D_c^T\left(z_c^{(k)} - \mu_c^{(k)}\right)\right]\right)$

[5]这是一个结论，证明我再找找，可以看下 Jupyter Notebook 的验证。

3.3 ADMM 2D TV Deblur

2D TV Deblur 和 Denosie 基本一样，唯一区别是在 $\frac{1}{2}\|Hx-y\|_2^2$ 中多了一个模糊的卷积操作。

$$\text{minimize} \quad \frac{1}{2}\|Hx-y\|_2^2+\lambda\|D_r x\|_1+\lambda\|D_c x\|_1 \tag{3.38}$$

Deblur 只有 x 部分的优化与 Denoise 不同。

$$x^{(k+1)}=\arg\min_x\left(\frac{1}{2}\|Hx^{(k)}-y\|_2^2+\frac{\rho}{2}\left\|D_r x^{(k)}-z_r^{(k)}+\mu_r^{(k)}\right\|_2^2+\frac{\rho}{2}\left\|D_c x^{(k)}-z_c^{(k)}+\mu_c^{(k)}\right\|_2^2\right)$$

求解 ，写成矩阵形式：

$$\min_x\left\|\begin{bmatrix}H\\ \sqrt{\rho}D_r\\ \sqrt{\rho}D_c\end{bmatrix}x^{(k)}-\begin{bmatrix}y\\ \sqrt{\rho}\left(z_r^{(k)}-\mu_r^{(k)}\right)\\ \sqrt{\rho}\left(z_c^{(k)}-\mu_c^{(k)}\right)\end{bmatrix}\right\|_2^2 \tag{3.39}$$

代入 $(X^TX)^{-1}X^TY$：

$$\begin{aligned}x^{(k+1)}&=\left(H^TH+\rho(D_r^TD_r+D_c^TD_c)\right)^{-1}\left[H^T,\sqrt{\rho}D_r^T,\sqrt{\rho}D_c^T\right]\begin{bmatrix}y\\ \sqrt{\rho}\left(z_r^{(k)}-\mu_r^{(k)}\right)\\ \sqrt{\rho}\left(z_c^{(k)}-\mu_c^{(k)}\right)\end{bmatrix}\\ &=\left(H^TH+\rho(D_r^TD_r+D_c^TD_c)\right)^{-1}\left(H^Ty+\rho\left[D_r^T\left(z_r^{(k)}-\mu_r^{(k)}\right)+D_c^T\left(z_c^{(k)}-\mu_c^{(k)}\right)\right]\right)\end{aligned} \tag{3.40}$$

优化 ，使用 DFT：

$$\left(\Lambda_H+\rho(\Lambda_r+\Lambda_c)\right)\mathcal{F}x^{(k+1)}=\left(\mathcal{F}H^Ty+\rho\left[\mathcal{F}D_r^T\left(z_r^{(k)}-\mu_r^{(k)}\right)+\mathcal{F}D_c^T\left(z_c^{(k)}-\mu_c^{(k)}\right)\right]\right) \tag{3.41}$$

Chapter 4

Resources

如果你看完这个 Tutorial，还是有不明白的地方，这里总结了本文撰写时参考的一些资料，包括论文，公开课的 slide，视频，开源代码等等。

4.1 Papers && Notes && Slides

有关 ADMM 的论文很多，但大部分对于初学者都不太友好，一些在论文作者看来显然的东西，对初学者可能并不显然，而且限于论文篇幅以及侧重，大部分论文并不会详细解释 ADMM 的细节。

初学者最好看下 CMU Convex Optimization 10-725 / 36-725 的这个 Introduction：`https://www.cs.cmu.edu/~ggordon/10725-F12/slides/01-intro-ryan.pdf`。这个 Slide 对于优化问题是什么，各种优化算法的作用，优点进行了介绍，而且还有图片作为例子。看完这个 Slide，应该可以对优化问题，以及 ADMM 有一点基本的感觉。

论文的话其实需要看的不多，应该看了也看不出什么名堂，主要还是课程的 slide 和 lecture note 比较有用。

- 看完 CMU 的那个 Slide，可以看下这篇 One Network to Solve Them All[2]，可以从 Introduction 开始，除了实验都看。
- 然后可以看下这篇的 Introduction，Plug-and-Play ADMM for Image Restoration[3]。
- 这篇 Decoupled Algorithm for MRI Reconstruction Using Nonlocal Block Matching Model: BM3D-MRI[4]，第三章 3 BM3D-MRI Formulation 中将傅立叶变换优化那段很有用。

有几个 Lecture Notes 特别有用 ：

- Stanford ee367 的一个 Notes，有讲傅立叶变换优化：`https://stanford.edu/class/ee367/reading/lecture6_notes.pdf`
- 强推 Gatech 的一个 Lecture Notes，详细讲了如何将 x subproblem 转换成 least square problem，以及 z subproblem 需要的 soft thresholding 是怎么回事：`http://mdav.ece.gatech.edu/ece-8823-spring2019/notes/16-admm.pdf`

4.2 Web

关于对角化 Circular Matrix ：

- Circular Convolution Matrix of $H^H H$： https://dsp.stackexchange.com/a/56721
- 关于 Diagonalization of circulant matrices 的证明： https://math.stackexchange.com/a/3207584

关于 Convolution ：

- Meaning of the Transpose of Convolution: https://dsp.stackexchange.com/a/64587
- 如果你不明白 2D Circular Convolution 怎么做的，看看这个：https://dsp.stackexchange.com/a/56021

4.3 Videos

只推荐一个视频：Lecture 19: Case Study: Generalized Lasso Problems (https://www.youtube.com/watch?v=A8qJDwO-bnE)

4.4 Code

ADMM TV 1D 的代码比较多，2D 的代码有一些，但通常都没有解释，找不到对应公式，可能比较难懂。

- 1D TV，用的求矩阵逆的方法： https://web.stanford.edu/~boyd/papers/admm/total_variation/total_variation.html
- 强推这个 Github 仓库，有很多算法，2D TV，1D TV 都有：https://github.com/tarmiziAdam2005/Image-Signal-Processing
- 这个 PnP ADMM 的 Matlab 包也很不错（论文 [3] 的源代码）：https://www.mathworks.com/matlabcentral/fileexchange/60641-plug-and-play-admm-for-image-restoration

Bibliography

[1] Kaixuan Wei et al. "Tuning-free Plug-and-Play Proximal Algorithm for Inverse Imaging Problems". In: *arXiv preprint arXiv:2002.09611* (2020) (cit. on p. 3).

[2] J. H. Rick Chang et al. *One Network to Solve Them All — Solving Linear Inverse Problems using Deep Projection Models*. 2017. arXiv: 1703.09912 [cs.CV] (cit. on p. 12).

[3] Stanley H. Chan, Xiran Wang, and Omar A. Elgendy. *Plug-and-Play ADMM for Image Restoration: Fixed Point Convergence and Applications*. 2016. arXiv: 1605.01710 [cs.CV] (cit. on pp. 12, 13).

[4] Ender Eksioglu. "Decoupled Algorithm for MRI Reconstruction Using Nonlocal Block Matching Model: BM3D-MRI". In: *Journal of Mathematical Imaging and Vision* 56 (Nov. 2016). DOI: 10.1007/s10851-016-0647-7 (cit. on p. 12).