ADMM for snapshot foreground-background separation

1 Problem formulation

Let $X \in \mathbb{R}^{M\times N\times F}$ be a video sequence, of which we observe

$$Y = \mathcal{H}(X)\,, \tag{1}$$

where $\mathcal{H} : \mathbb{R}^{M\times N\times F} \to \mathbb{R}^{M\times Q}$ is a linear operator that models snapshot compressive imaging. When $Q = N$, we obtain the traditional setup of CACTI [1]. When $Q = N + F - 1$, we obtain the one-pixel (horizontal) shift scenario of [2].

Assumption 1.1. $X \in \mathbb{R}^{M\times N\times F}$ can be decomposed as

$$X = B + S\,, \tag{2}$$

where

- $B \in \mathbb{R}^{M\times N\times F}$ represents a *low-rank* background, in the sense that the matrix $\overline{B} \in \mathbb{R}^{MN\times F}$ is low-rank. Each column f of $\overline{B}$ is obtained by vectorizing the fth frame of B: $\overline{B}_{:,f} = \mathrm{vec}(B_{:,:,f})$, for $f = 1,\dots,F$.
- $S \in \mathbb{R}^{M\times N\times F}$ represents a sparse foreground, satisfying patch similarity. That is, many of the entries of S are zero, and the matrix $\widetilde{S} \in \mathbb{R}^{P\times L}$ is low-rank. The L columns of the matrix $\widetilde{S}$ are obtained by vectorizing L patches of size $\sqrt{P}\times\sqrt{P}$ from the frames S_f, $f = 1,\dots,F$.

Notice that because S is sparse, the matrix $\widetilde{S}$ only needs to contain patches whose pixels are nonzero. This will reduce the number of patches L. Throughout, given a 3D tensor $M \in \mathbb{R}^{d_1\times d_2\times d_3}$, the notation $\overline{M} \in \mathbb{R}^{d_1 d_2\times d_3}$ represents the matrix whose columns are obtained by vectorizing each slice $M_{:,:,i}$, along the third dimension of M. And $\widetilde{M} \in \mathbb{R}^{P\times L}$ refers to the matrix obtained by extracting L patches for size $\sqrt{P}\times\sqrt{P}$ from slices of $M_{:,:,i}$, for any i, and vectorizing them into the columns of $\widetilde{M}$.

Optimization problem. Given the above assumption, we can formulate the reconstruction problem as

$$\underset{B,S}{\text{minimize}}\ \frac{\lambda_0}{2}\left\|Y - \mathcal{H}(B+S)\right\|_F^2 + \lambda_1\left\|S\right\|_1 + \lambda_2\left\|\overline{B}\right\|_\star + \lambda_3\left\|\widetilde{S}\right\|_\star\,, \tag{3}$$

where

- $\|\cdot\|_F$ is the Frobenius norm. It is the sum of squared entries, whether applied to a matrix or a tensor.
- $\|\cdot\|_1$ is the ℓ_1-norm (sum of absolute values). When applied to a matrix or tensor, it is the same as the ℓ_1-norm applied to their vectorization.
- $\|\cdot\|_\star$ is the nuclear-norm (Schatten 1-norm), which is the ℓ_1-norm applied to the singular values of the matrix.
- $\lambda_0, \lambda_1, \lambda_2, \lambda_3 \geq 0$ are nonnegative hyperparameters.

2 Applying ADMM

To apply ADMM to (3), we need to rewrite it in a way that the variables become decoupled. To do so, we introduce two copies of S, denoted U and V, and one copy of B, denoted L. Problem (3) is thus equivalent to

$$\begin{array}{ll} \underset{X,S,U,V,B,L}{\text{minimize}} & \frac{\lambda_0}{2}\left\|Y - \mathcal{H}(X)\right\|_F^2 + \lambda_1\left\|U\right\|_1 + \lambda_2\left\|\overline{L}\right\|_\star + \lambda_3\left\|\widetilde{V}\right\|_\star \\ \text{subject to} & S = U \\ & S = V \\ & B = L \\ & X = B + V\,. \end{array} \tag{4}$$

The augmented Lagrangian of (4) is

$$\begin{aligned} L_\rho(X,S,U,V,B,L\,;\,\Theta,\Gamma,\Delta,\Lambda) &= \frac{\lambda_0}{2}\left\|Y - \mathcal{H}(X)\right\|_F^2 + \lambda_1\left\|U\right\|_1 + \lambda_2\left\|\overline{L}\right\|_\star + \lambda_3\left\|\widetilde{V}\right\|_\star \\ &+ \langle\Theta, S-U\rangle + \langle\Gamma, S-V\rangle + \langle\Delta, B-L\rangle + \langle\Lambda, X-B-V\rangle + \frac{\rho}{2}\|S-U\|_F^2 + \frac{\rho}{2}\|S-V\|_F^2 + \frac{\rho}{2}\|B-L\|_F^2 \\ &+ \frac{\rho}{2}\|X-B-V\|_F^2\,, \end{aligned}$$

where we associated the dual variables $\Theta, \Gamma, \Delta, \Lambda \in \mathbb{R}^{M\times N\times F}$ with the 1st, 2nd, 3rd, and 4th constraints of (4).

We group the primal variables into (X, S, L) and (U, V, B). ADMM iteratively minimizes L_ρ w.r.t. the first set of variables and, then, with these variables fixed at the new value, minimizes L_ρ w.r.t. the second set of variables. Finally, it updates all the Lagrange multipliers.

More concretely, starting with arbitrary $\left(U^0, V^0, B^0\right)$ and $\left(\Theta^0, \Gamma^0, \Delta^0, \Lambda^0\right)$, ADMM iterates for $k = 0, 1, \ldots,$

$$\begin{aligned} (X^{k+1}, S^{k+1}, L^{k+1}) = \underset{X,S,L}{\arg\min}\ &\left[\frac{\lambda_0}{2}\left\|Y - \mathcal{H}(X)\right\|_F^2 + \langle\Lambda^k, X - B^k - V^k\rangle + \frac{\rho}{2}\|X - B^k - V^k\|_F^2\right] \\ &+ \left[\langle\Theta^k, S - U^k\rangle + \langle\Gamma^k, S - V^k\rangle + \frac{\rho}{2}\|S - U^k\|_F^2 + \frac{\rho}{2}\|S - V^k\|_F^2\right] \\ &+ \left[\lambda_2\left\|\overline{L}\right\|_\star + \langle\Delta^k, B^k - L\rangle + \frac{\rho}{2}\|B^k - L\|_F^2\right] \end{aligned} \tag{5a}$$

$$(U^{k+1}, V^{k+1}, B^{k+1}) = \underset{U,V,B}{\arg\min}\ \left[\lambda_1\left\|U\right\|_1 + \langle\Theta^k, S^{k+1} - U\rangle + \frac{\rho}{2}\|S^{k+1} - U\|_F^2\right]$$

$$+\left[\lambda_3\left\|\widetilde{V}\right\|_\star + \langle \Gamma^k, S^{k+1} - V\rangle + \langle \Delta^k, B - L^{k+1}\rangle + \langle \Lambda^k, X^{k+1} - B - V\rangle \right.$$
$$\left. + \frac{\rho}{2}\|S^{k+1} - V\|_F^2 + \frac{\rho}{2}\|B - L^{k+1}\|_F^2 + \frac{\rho}{2}\|X^{k+1} - B - V\|_F^2\right] \tag{5b}$$

$$\Theta^{k+1} = \Theta^k + \rho(S^{k+1} - U^{k+1}) \tag{5c}$$
$$\Gamma^{k+1} = \Gamma^k + \rho(S^{k+1} - V^{k+1}) \tag{5d}$$
$$\Delta^{k+1} = \Delta^k + \rho(B^{k+1} - L^{k+1})\,. \tag{5e}$$
$$\Lambda^{k+1} = \Lambda^k + \rho(X^{k+1} - B^{k+1} - V^{k+1})\,. \tag{5f}$$

We can see that (5a) decomposes into three independent problems, in X, S, and L, that can be solved in parallel. Also, (5b) decomposes into two independent problems, in U, and in (V, B), that can be solved in parallel. We now elaborate on how to solve each of these problems.

Updating X. The problem in X in (5a) can be rewritten as

$$\begin{aligned} & \underset{X}{\text{minimize}} \ \frac{\lambda_0}{2}\left\|Y - \mathcal{H}(X)\right\|_F^2 + \langle \Lambda^k, X - B^k - V^k\rangle + \frac{\rho}{2}\left\|X - B^k - V^k\right\|_F^2 \\ \iff \quad & \underset{X}{\text{minimize}} \ \frac{\lambda_0}{2}\left\|Y - \mathcal{H}(X)\right\|_F^2 + \frac{\rho}{2}\left\|X - \left(B^k + V^k - \frac{1}{\rho}\Lambda^k\right)\right\|_F^2 . \end{aligned} \tag{6}$$

Defining $y := \text{vec}(Y) \in \mathbb{R}^{MQ}$ and

$$X_a := B^k + V^k - \frac{1}{\rho}\Lambda^k \in \mathbb{R}^{M\times N\times F}$$
$$x_a := \text{vec}(X_a) \in \mathbb{R}^{MNF}\,,$$

where the vectorization of a tensor follows a specific order (described below), problem (6) can be written in a vector format:

$$\underset{x\in\mathbb{R}^{MNF}}{\text{minimize}} \ \frac{\lambda_0}{2}\|y - Hx\|_2^2 + \frac{\rho}{2}\|x - x_a\|_2^2\,,$$

where $H \in \mathbb{R}^{MQ\times MNF}$ is a matricial representation of $\mathcal{H}$. Taking the gradient w.r.t. x and equating it to zero, we obtain a closed-form solution for updating X:

$$x^{k+1} = \left(H^\top H + \frac{\rho}{\lambda_0} I_{MNF\times MNF}\right)^{-1}\left[H^\top y + \frac{\rho}{\lambda_0}x_a\right]\,, \tag{7}$$

where $I_{MNF\times MNF}$ is the identity matrix.

In practice, we should not compute the matrices in (7) explicitly. To see how to perform the operations in (7), let us consider the simple case of conventional snapshot compressive imaging, i.e., no pixel-shifts. In this case, $Y \in \mathbb{R}^{M\times N}$. Let $Z \in \mathbb{R}^{M\times N\times F}$ be a generic tensor and let $\mathcal{H}$ apply mask $M_f \in \mathbb{R}^{M\times N}$ to frame f. In other words,

$$Y = \sum_{f=1}^{F} M_f \odot Z_{:,:,f}$$

$$\iff \qquad \operatorname{vec}(Y) = \sum_{f=1}^{F} \operatorname{vec}(M_f) \odot \operatorname{vec}(Z_{:,:,f})$$

$$\iff \qquad \underbrace{\operatorname{vec}(Y)}_{y} = \underbrace{\begin{bmatrix}\operatorname{Diag}(m_1) & \cdots & \operatorname{Diag}(m_F)\end{bmatrix}}_{H} \underbrace{\begin{bmatrix}\operatorname{vec}(Z_{:,:,1}) \\ \vdots \\ \operatorname{vec}(Z_{:,:,F})\end{bmatrix}}_{z},$$

where $\odot$ represents the pointwise (Hadamard) product, $m_f := \operatorname{vec}(M_f)$, and $\operatorname{Diag}(\cdot)$ represents a diagonal matrix with a specified vector in the diagonal. The matrix H is indicated as well as the vectorization of the tensor X. The matrix $H^\top H$ in (7) is a block matrix with diagonal blocks. However, using the matrix inversion lemma [3], we can work instead with $HH^\top$, which is diagonal:

$$HH^\top = \sum_{f=1}^{F} \operatorname{Diag}(m_f^2)\,,$$

where $m_f^2 := m_f \odot m_f$ denotes the vector m_f with squared entries. Note that the diagonal of $HH^\top$ can be computed before the execution of the algorithm. The matrix inversion lemma will give us

$$\left(H^\top H + \frac{\rho}{\lambda_0} I_{MNF\times MNF}\right)^{-1} = \frac{\lambda_0}{\rho}\left[I - \frac{\lambda_0}{\rho} H^\top \left(I + \frac{\lambda_0}{\rho} HH^\top\right)^{-1} H\right], \tag{8}$$

where we omitted the dimensions of the identity matrices for simplicity. Note that the first one in the righthand side of (8) is $MNF \times MNF$, while the second one is $MN \times MN$. Plugging (8) into (7) yields

$$x^{k+1} = \frac{\lambda_0}{\rho}\left[I - \frac{\lambda_0}{\rho} H^\top \left(I + \frac{\lambda_0}{\rho} HH^\top\right)^{-1} H\right]\left(H^\top y + \frac{\rho}{\lambda_0} x_a\right). \tag{9}$$

Undoing the vectorization operations, we can compute the result of (9) from right to left according to Algorithm 1. In step 5, $\oslash$ denotes the elementwise division, and $1_{M\times N}$ the matrix of all-ones in $\mathbb{R}^{M\times N}$.

Algorithm 1 Update of X in (9) for conventional SCI

Before the algorithm: Compute diagonal of $HH^\top$: $D_{HH} = \sum_{f=1}^{F} M_f^2 \in \mathbb{R}^{M\times N}$

During the algorithm:

1: $C_1 = \frac{\rho}{\lambda_0}(B^k + V^k - \frac{1}{\rho}\Lambda^k) \in \mathbb{R}^{M\times N\times F}$
2: $C_2 \in \mathbb{R}^{M\times N\times F}$ such that each slice $(C_2)_{:,:,f} = M_f \odot Y$, $f = 1, \ldots, F$
3: $C_3 = C_1 + C_2 \in \mathbb{R}^{M\times N\times F}$
4: $C_4 = \sum_{f=1}^{F} M_f \odot (C_3)_{:,:,f} \in \mathbb{R}^{M\times N}$
5: $C_5 = C_4 \oslash (1_{M\times N} + \frac{\lambda_0}{\rho} D_{HH}) \in \mathbb{R}^{M\times N}$
6: $C_6 \in \mathbb{R}^{M\times N\times F}$ such that each slice $(C_6)_{:,:,f} = M_f \odot C_5$, $f = 1, \ldots, F$
7: $X^{k+1} = \frac{\lambda_0}{\rho}(C_3 - \frac{\lambda_0}{\rho} C_6) \in \mathbb{R}^{M\times N\times F}$

This code can run fast if

- Memory to store $C_1, \ldots, C_6, X^{k+1}$ is preallocated
- Operations in each step are parallelized, e.g., in a GPU.

The total operation count is roughly $10MNF + 3MN$ FLOPs.

Updating S. The problem in S in (5a) is

$$\begin{aligned} S^{k+1} &= \arg\min_S \ \langle \Theta^k, S - U^k \rangle + \langle \Gamma^k, S - V^k \rangle + \frac{\rho}{2}\|S - U^k\|_F^2 + \frac{\rho}{2}\|S - V^k\|_F^2 \\ &= \frac{1}{2}\left(U^k + V^k - \frac{1}{\rho}\Theta^k - \frac{1}{\rho}\Gamma^k\right), \end{aligned}$$

where we simply equated the derivative w.r.t. S to zero.

Updating L. The problem in L in (5a) can be rewritten as

$$\begin{aligned} & \underset{L}{\text{minimize}} \ \ \lambda_2 \left\|\overline{L}\right\|_\star + \langle \Delta^k, B^k - L \rangle + \frac{\rho}{2}\|B^k - L\|_F^2 \\ \iff \quad & \underset{L}{\text{minimize}} \ \ \frac{\lambda_2}{\rho}\left\|\overline{L}\right\|_\star + \frac{1}{2}\left\|L - \left(B^k + \frac{1}{\rho}\Delta^k\right)\right\|_F^2 . \end{aligned} \tag{10}$$

In other words,

$$L^{k+1} = \operatorname{prox}_{\frac{\lambda_2}{\rho}\|\overline{\cdot}\|_\star}\left(B^k + \frac{1}{\rho}\Delta^k\right), \tag{11}$$

where the *proximal operator* of a function f at point y is defined as

$$\operatorname{prox}_f(y) := \arg\min_x \ f(x) + \frac{1}{2}\|x - y\|_2^2 .$$

If we use a simple nuclear norm (unweighted), the solution to (11) requires

1. Forming the tensor $L_a := B^k + \frac{1}{\rho}\Delta^k \in \mathbb{R}^{M\times N\times F}$ and reshaping it to a matrix $\overline{L}_a \in \mathbb{R}^{MN\times F}$.
2. Computing the singular value decomposition of $\overline{L}_a$:

   $$\overline{L}_a = U\Sigma V^\top ,$$

   where $U \in \mathbb{R}^{MN\times k}$ and $V \in \mathbb{R}^{F\times k}$ contain orthonormal columns, and $\Sigma \in \mathbb{R}^{k\times k}$ is a diagonal matrix with singular values. The integer k is the rank of $\overline{L}_a$ and is usually not known in advance.
3. Constructing the solution $\overline{L}^{k+1} = U\mathcal{S}_{\frac{\lambda_2}{\rho}}(\Sigma)V^\top$, where $\mathcal{S}_c(x) := \max\{x - c, 0\}$ is a soft-thresholding operation applied componentwise to a matrix. See, e.g., [4, (1)-(3)]. Then, we need to reshape $\overline{L}^{k+1}$ back to a tensor L^{k+1}.

If we use a weighted nuclear norm [4], only step 3 will be different (see below). To compute the SVD decomposition in step 2, we can use the randomized algorithms described in [5].

In Python, the following packages implement algorithms from [5]:

- PyTorch

- Scikit-learn

According to [5, §4.6, §5.1], the cost of computing the SVD decomposition in step 2 should be $O(MNF(\log(\ell)+k))$, where k is the rank of $\overline{L}_a$ and ℓ is the oversampling parameter, which is typically a small number (between 2 and 10). This assumes the computation is done using the FFT rather than Gaussian matrices.

Remark 2.1. An important aspect in terms of implementation is that if the above software packages accept implicit access to $\overline{L}_a$, i.e., rather than the full matrix, only functions performing matrix vector products $\overline{L}_a x$ and $\overline{L}_a^\top y$, we may be able to compute the SVD decomposition without reshaping L_a into $\overline{L}_a$.

According to [4], [6], we can improve the performance of the algorithm by using a weighted (or reweighted) nuclear norm. That is, instead of (10), we solve

$$\underset{\overline{L}}{\text{minimize}} \;\; \frac{\lambda_2}{\rho}\left\|\overline{L}\right\|_{w,\star} + \frac{1}{2}\left\|\overline{L}-\overline{L}_a\right\|_F^2, \tag{12}$$

where $\overline{L}_a$ is defined in step 1 above,

$$\|X\|_{w,\star} := \sum_i w_i \sigma_i(X)\,, \tag{13}$$

is the *weighted nuclear norm*, and $\sigma_i(X)$ represents the ith singular value of X. We assume $w_i \geq 0$, for all i. Theorem 1 of [4] applies the von Neumann trace inequality to show that the solution of (12) is given by

$$\overline{L}^{k+1} = UDV^\top,$$

where $\overline{L}_a = U\Sigma V^\top$ is the SVD decomposition of $\overline{L}_a$, and $D = \text{Diag}\Big(d_1,\ldots,d_{\min\{MN,F\}}\Big)$ is a diagonal matrix whose diagonal values are the solution of

$$\begin{array}{ll} \underset{d_1,\ldots,d_{\min\{MN,F\}}}{\text{minimize}} & \sum_{i=1}^{\min\{MN,F\}} \frac{1}{2}(\sigma_i - d_i)^2 + \frac{\lambda_2 w_i}{\rho} d_i \\ \text{subject to} & d_1 \geq d_2 \geq \cdots \geq d_n \geq 0\,, \end{array} \tag{14}$$

where σ_i is the ith diagonal entry of Σ. As in Corollary 1 of [4], if the weights satisfy $0 \leq w_1 \leq w_2 \leq \cdots \leq w_{\min\{MN,F\}}$, then the constraint of (14) is automatically satisfied, and its solution is given by

$$d_i = \max\left\{\sigma_i - \frac{\lambda_2}{\rho} w_i\,, 0\right\}, \qquad i = 1,\ldots,\min\{MN,F\}\,. \tag{15}$$

The paper suggests a reweighted solution based on reweighted ℓ_1-norm minimization [7], where weights are iteratively updated as

$$w_i^{t+1} = \frac{1}{d_i^t + \epsilon}\,, \qquad i = 1,\ldots,\min\{MN,F\}\,, \tag{16}$$

where t denotes the iteration number, d_i^t is the estimate of the ith singular value of $\overline{L}^{k+1}$ at iteration t, and $\epsilon > 0$ is a small number for stability. The weights in (16) automatically satisfy the ordering required for (15). The authors in [4] compute the limit point of these iterations in

Algorithm 2 Update of L in (12)

Initialization: Stopping tolerance $\delta > 0$ (e.g., 10^{-3}), stability number $\epsilon > 0$ (e.g., 10^{-3}), maximum number of reweighting iterations $T > 0$ (e.g., 50), parameters to compute the randomized SVD (ℓ and a small tolerance).

Compute $L_a := B^k + \frac{1}{\rho}\Delta^k \in \mathbb{R}^{M\times N\times F}$

Reshape L_a into $\overline{L}_a \in \mathbb{R}^{MN\times F}$

Compute SVD $\overline{L}_a = U\Sigma V^\top$ with randomized algorithm ▷ Do not confuse U, V with U^k, V^k

Initialize $d_i = \sigma_i$, for $i = 1, \dots, \min\{MN, F\}$, where σ_i is the ith diagonal entry of Σ.

for $t = 1, \dots, T$ **do**

  Compute, for all $i = 1, \dots, \min\{MN, F\}$, ▷ Use vector form, not a for loop

$$d_i^{t+1} = \max\left\{\sigma_i - \frac{\lambda_2}{\rho}\frac{1}{d_i^t + \epsilon}, 0\right\}$$

  if $\max_i |d_i^{t+1} - d_i^t| \le \delta$ **then**

    Break

  end if

end for

Compute matricial solution $\overline{L}^{k+1} = U\mathrm{Diag}(d_1^{t+1}, \dots, d_{\min\{MN,F\}}^{t+1})V^\top$

Reshape $\overline{L}^{k+1} \in \mathbb{R}^{MN\times F}$ into $L^{k+1} \in \mathbb{R}^{M\times N\times F}$

closed-form, but the analysis looks complicated and the solution seems to have one more free parameter than necessary.

My suggestion is to implement the iterations in (16) recursively using, for example $\epsilon = 10^{-3}$. In my experience, these requires from 3 to 40 iterations to converge. The resulting algorithm is described in Algorithm 2. Note that Algorithms 1 and 2 can be run in parallel.

Updating U. The problem in U in (5b) can be written as

$$\begin{aligned} U^{k+1} &= \arg\min_U \ \lambda_1\left\|U\right\|_1 + \langle \Theta^k, S^{k+1} - U\rangle + \frac{\rho}{2}\|S^{k+1} - U\|_F^2 \\ &= \arg\min_U \ \frac{\lambda_1}{\rho}\left\|U\right\|_1 + \frac{1}{2}\left\|U - \left(S^{k+1} + \frac{1}{\rho}\Theta^k\right)\right\|_F^2 \\ &= \mathrm{prox}_{\frac{\lambda_1}{\rho}\|\cdot\|_1}\left(S^{k+1} + \frac{1}{\rho}\Theta^k\right). \end{aligned}$$

The procedure to update U is then quite simple: soft-thresholding all the components of $U_a := S^{k+1} + \frac{1}{\rho}\Theta^k$ in parallel. This is done in Algorithm 3.

Updating (V, B). The terms in (5b) that depend on V and B are not decoupled, so they must be optimized simultaneously. We notice, however, that minimization w.r.t. B has a closed-form solution, so we can first minimize w.r.t. B, then w.r.t. V:

$$\begin{aligned} (V^{k+1}, B^{k+1}) = \arg\min_{V,B} \ & \lambda_3\left\|\widetilde{V}\right\|_\star + \langle \Gamma^k, S^{k+1} - V\rangle + \langle \Delta^k, B - L^{k+1}\rangle + \langle \Lambda^k, X^{k+1} - B - V\rangle \\ & + \frac{\rho}{2}\|S^{k+1} - V\|_F^2 + \frac{\rho}{2}\|B - L^{k+1}\|_F^2 + \frac{\rho}{2}\|X^{k+1} - B - V\|_F^2 \qquad (17) \\ = \arg\min_V \ & \lambda_3\left\|\widetilde{V}\right\|_\star + \frac{\rho}{4}\|V\|_F^2 + \langle V, -\frac{1}{2}\Delta^k - \frac{1}{2}\Lambda^k + 2L^{k+1} - 2X^{k+1}\rangle \end{aligned}$$

Algorithm 3 Update of U in (5b)

1: Compute $U_a = S^{k+1} + \frac{1}{\rho}\Theta^k$
2: **for** i, j, l in parallel **do**
3:

$$U_{ijk}^{k+1} = \begin{cases} (U_a)_{ijl} + \frac{\lambda}{\rho} & \text{, if } (U_a)_{ijl} < -\frac{\lambda}{\rho} \\ 0 & \text{, if } \left|(U_a)_{ijl}\right| \leq \frac{\lambda}{\rho} \\ (U_a)_{ijl} - \frac{\lambda}{\rho} & \text{, if } (U_a)_{ijl} > \frac{\lambda}{\rho} \end{cases}$$

4: **end for**

$$+ \langle \Gamma^k, S^{k+1} - V \rangle + \frac{\rho}{2}\|S^{k+1} - V\|_F^2 \tag{18}$$

$$= \arg\min_V \ \frac{2\lambda_3}{3\rho}\left\|\widetilde{V}\right\|_\star + \frac{1}{2}\left\|V - V_a\right\|_F^2. \tag{19}$$

From (17) to (18), we differentiated w.r.t. B and equated it zero, giving a closed-form expression for updating it:

$$B^{k+1} = \frac{1}{2}\left[L^{k+1} + X^{k+1} - V^{k+1} + \frac{1}{\rho}\Lambda^k - \frac{1}{\rho}\Delta^k\right].$$

We then replaced the expression for B in the objective function. From (18) to (19), we developed the squares, ignored terms that do not depend on V, rescaled the objective function, and defined

$$V_a := \frac{1}{3}\left[X^{k+1} - L^{k+1} + 2S^{k+1} + \frac{1}{\rho}(\Delta^k + \Lambda^k + 2\Gamma^k)\right].$$

Notice (19) has exactly the same format as the update of L in (10). The reshaping of the tensors, however, is different. Apart from this, the procedure is exactly like the update of L in Algorithm 2 and is described in Algorithm 4.

This update is likely to be the most time-consuming step of the entire algorithm, due to the potentially very large number of patches. One key point of step 2, however, is that S^{k+1} is likely to be sparse. This means V_a is also likely to be sparse. So the patch extraction in step 2 ***should consider patches containing only nonzero entries***. Note:

- We need to keep track of the location of the patches to be able to execute step 12.
- Algorithms 2 and 4 have similar code, so make sure to reuse it.

Updating Θ, Γ, Δ, and Λ. The updates of the dual variables in (5c)-(5f) are trivial. We can, however, make a small simplification: notice how all the dual variables in the above problems appear always multiplied by $1/\rho$. We can therefore define

$$\Theta'^k = \frac{1}{\rho}\Theta^k$$
$$\Gamma'^k = \frac{1}{\rho}\Gamma^k$$

Algorithm 4 Update of V and B in (5b)

Initialization: Stopping tolerance $\delta > 0$ (e.g., 10^{-3}), stability number $\epsilon > 0$ (e.g., 10^{-3}), maximum number of reweighting iterations $T > 0$ (e.g., 50), parameters to compute the randomized SVD (ℓ and a small tolerance), patch size P.

Compute $V_a = \frac{1}{3}\left[X^{k+1} - L^{k+1} + 2S^{k+1} + \frac{1}{\rho}(\Delta^k + \Lambda^k + 2\Gamma^k)\right] \in \mathbb{R}^{M\times N\times F}$

Reshape V_a into $\widetilde{V}_a \in \mathbb{R}^{P\times L}$ by extracting L patches of size $\sqrt{P}\times\sqrt{P}$ from V_a and vectorizing them into the columns of $\widetilde{V}_a$ ▷ See comment about sparsity

Compute SVD $\widetilde{V}_a = U\Sigma V^\top$ with randomized algorithm ▷ Do not confuse U, V with U^k, V^k

Initialize $d_i = \sigma_i$, for $i = 1, \ldots, \min\{P, L\}$, where σ_i is the ith diagonal entry of Σ.

for $t = 1, \ldots, T$ **do**

  Compute, for all $i = 1, \ldots, \min\{P, L\}$, ▷ Use vector form, not a for loop

$$d_i^{t+1} = \max\left\{\sigma_i - \frac{2\lambda_3}{3\rho}\frac{1}{d_i^t + \epsilon}\,,\, 0\right\}$$

  if $\max_i |d_i^{t+1} - d_i^t| \le \delta$ **then**

    Break

  end if

end for

Compute matricial solution $\widetilde{V}^{k+1} = U\mathrm{Diag}(d_1^{t+1}, \ldots, d_{\min\{P,L\}}^{t+1})V^\top$

Reshape $\widetilde{V}^{k+1} \in \mathbb{R}^{P\times L}$ into $V^{k+1} \in \mathbb{R}^{M\times N\times F}$

Compute $B^{k+1} = \frac{1}{2}\left[L^{k+1} + X^{k+1} - V^{k+1} + \frac{1}{\rho}\Lambda^k - \frac{1}{\rho}\Delta^k\right]$

$$\Delta'^k = \frac{1}{\rho}\Delta^k$$
$$\Lambda'^k = \frac{1}{\rho}\Lambda^k\,.$$

Dividing both sides of equations (5c)-(5e) by ρ, we obtain the updates in terms of these new variables:

$$\begin{aligned}
\Theta'^{k+1} &= \Theta'^k + S^{k+1} - U^{k+1} \\
\Gamma'^{k+1} &= \Gamma'^k + S^{k+1} - V^{k+1} \\
\Delta'^{k+1} &= \Delta'^k + B^{k+1} - L^{k+1} \\
\Lambda'^{k+1} &= \Lambda'^k + X^{k+1} - B^{k+1} - V^{k+1}\,.
\end{aligned}$$

This is equivalent to ADMM in scaled form [8, §3.1.1].

Full algorithm implementation. An implementation of the full algorithm should consider a stopping criterion in terms of the primal and dual residuals [8, §3.3] and adjust ρ at every iteration [8, §3.4.1]. The primal residual at iteration k should be

$$r^{k+1} = \begin{bmatrix} S^{k+1} - U^{k+1} \\ S^{k+1} - V^{k+1} \\ B^{k+1} - L^{k+1} \\ X^{k+1} - B^{k+1} - V^{k+1} \end{bmatrix},$$

and the dual residual should be

$$s^{k+1} = -\rho \begin{bmatrix} V^{k+1} - V^k + B^{k+1} - B^k \\ U^{k+1} - U^k + V^{k+1} - V^k \\ B^{k+1} - B^k \end{bmatrix}.$$

See Github for an example of the implementation of ADMM to solve a simpler problem.

Convergence. ADMM applies to problems with the format [8]

$$\begin{aligned} \underset{x,z}{\text{minimize}} \quad & f(x) + g(z) \\ \text{subject to} \quad & Ax + Bz = 0\,. \end{aligned} \tag{20}$$

It converges when f and g are convex and real valued, problem (20) is solvable, and the matrices A and B have full column rank [9]. Problem (4) can be written in the same format as (20):

$$\begin{aligned} \underset{(X,S,L),(U,V,B)}{\text{minimize}} \quad & \left(\tfrac{1}{2}\left\|Y - \mathcal{H}(X)\right\|_F^2 + \lambda_2\left\|\overline{L}\right\|_\star\right) + \left(\lambda_1\left\|U\right\|_1 + \lambda_3\left\|\widetilde{V}\right\|_\star\right) \\ \text{subject to} \quad & \begin{bmatrix} 0 & I & 0 \\ 0 & I & 0 \\ 0 & 0 & -I \\ I & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ s \\ l \end{bmatrix} + \begin{bmatrix} -I & 0 & 0 \\ 0 & -I & 0 \\ 0 & 0 & I \\ 0 & -I & -I \end{bmatrix} \begin{bmatrix} u \\ v \\ b \end{bmatrix} = 0\,, \end{aligned} \tag{21}$$

where lowercase variables are vectorizations of the corresponding uppercase variables, and I and 0 are identity and zero matrices of appropriate dimensions. It is clear that both matrices in (21) have full column rank. Also, the functions in the objective function are all convex and real-valued.

References

[1] P. Llull, X. Liao, X. Yuan, *et al.*, "Coded aperture compressive temporal imaging," *Optics Express*, vol. 21, no. 9, pp. 10 526–10 545, 2013. DOI: 10.1364/OE.21.010526.

[2] A. Matin and X. Wang, "Compressive coded rotating mirror camera for high-speed imaging," *Photonics*, vol. 8, no. 34, pp. 1–12, 2021. DOI: 10.3390/photonics8020034.

[3] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004. [Online]. Available: https://web.stanford.edu/~boyd/cvxbook/.

[4] S. Gu, Q. Xie, D. Meng, W. Zuo, X. Feng, and L. Zhang, "Weighted nuclear norm minimization and its applications to low level vision," *Int. J. Comput. Vis.*, vol. 121, no. 1, pp. 183–208, 2017. DOI: 10.1007/s11263-016-0930-5.

[5] N. Halko, P. G. Martinsson, and J. A. Tropp, "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions," *SIAM Review*, vol. 53, no. 2, pp. 217–288, 2011.

[6] Y. Liu, X. Yuan, J. Suo, D. J. Brady, and Q. Dai, "Rank minimization for snapshot compressive imaging," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 41, no. 12, pp. 2990–3006, 2019. DOI: 10.1109/TPAMI.2018.2873587.

[7] E. Candès, M. Wakin, and S. Boyd, "Enhancing sparsity by reweighted ℓ_1 minimization," *J. Fourier Anal. Appl.*, vol. 14, pp. 877–905, 2008.

[8] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating method of multipliers," *Foundations and Trends in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2011.

[9] J. F. C. Mota, J. M. F. Xavier, P. M. Q. Aguiar, and M. Püschel. "A proof of convergence for the alternating direction method of multipliers applied to polyhedral-constrained functions." arXiv: 1112.2295. (2011), unpublished.