PARAMETER-FREE PLUG-AND-PLAY ADMM FOR IMAGE RESTORATION

Xiran Wang and Stanley H. Chan

School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907.

ABSTRACT

Plug-and-Play ADMM is a recently developed variation of the classical ADMM algorithm that replaces one of the subproblems using an off-the-shelf image denoiser. Despite its apparently ad-hoc nature, Plug-and-Play ADMM produces surprisingly good image recovery results. However, since in Plug-and-Play ADMM the denoiser is treated as a black-box, behavior of the overall algorithm is largely unknown. In particular, the internal parameter that controls the rate of convergence of the algorithm has to be adjusted by the user, and a bad choice of the parameter can lead to severe degradation of the result. In this paper, we present a parameter-free Plug-and-Play ADMM where internal parameters are updated as part of the optimization. Our algorithm is derived from the generalized approximate message passing, with several essential modifications. Experimentally, we find that the new algorithm produces solutions along a reliable and fast converging path.

***Index Terms*—** Plug-and-Play, ADMM, image restoration, parameter-free, generalized approximate message passing

1. INTRODUCTION

1.1. Plug-and-Play ADMM

With the astonishing number of applications of the alternating direction method of multiplier (ADMM, [1]), it is reasonably safe to say that ADMM is almost *the* workhorse of most, if not all, image restoration algorithms we use nowadays [2–4]. ADMM is a generic algorithm that solves optimization problems in the form

$$\underset{\boldsymbol{x}}{\text{minimize}} \quad f(\boldsymbol{x}) + \lambda g(\boldsymbol{x}) \tag{1}$$

for some cost function f and regularization function g. In model-based image processing, f is called the data fidelity term, and g is called the prior term [5].

ADMM has many attractive features. Apart from its simple implementation and broad applicability, the variable splitting nature of the algorithm offers additional degrees of freedom in designing the steps of the algorithm. In particular, if we use ADMM to solve (1), the algorithm proceeds by solving a sequence of subproblems in the following *modules*:

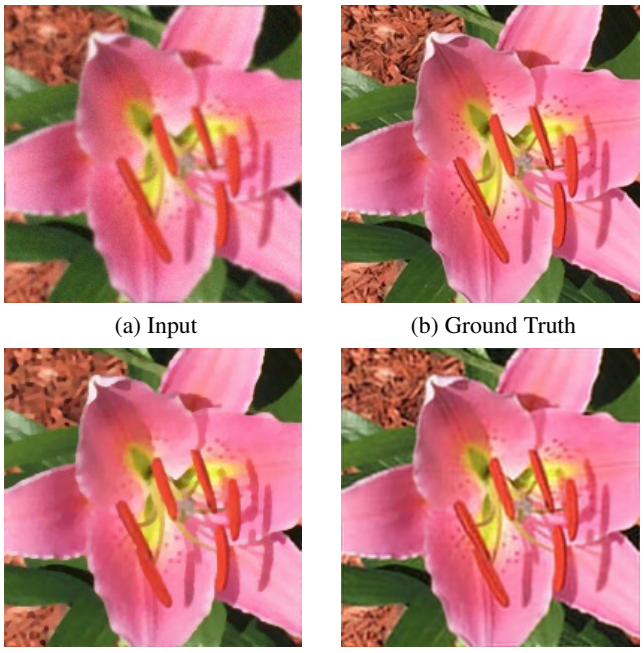
$$\boldsymbol{x}^{(k+1)} = \underset{\boldsymbol{x}\in\mathbb{R}^n}{\text{argmin}} \quad f(\boldsymbol{x}) + \frac{\rho}{2}\|\boldsymbol{x} - \widetilde{\boldsymbol{x}}^{(k)}\|^2, \tag{2}$$

$$\boldsymbol{v}^{(k+1)} = \underset{\boldsymbol{v}\in\mathbb{R}^n}{\text{argmin}} \quad \lambda g(\boldsymbol{v}) + \frac{\rho}{2}\|\boldsymbol{v} - \widetilde{\boldsymbol{v}}^{(k)}\|^2, \tag{3}$$

$$\boldsymbol{u}^{(k+1)} = \boldsymbol{u}^{(k)} + \rho(\boldsymbol{x}^{(k+1)} - \boldsymbol{v}^{(k+1)}), \tag{4}$$

where $\widetilde{\boldsymbol{x}}^{(k)} \overset{\text{def}}{=} \boldsymbol{v}^{(k)} - (1/\rho)\boldsymbol{u}^{(k)}$, $\widetilde{\boldsymbol{v}}^{(k)} \overset{\text{def}}{=} \boldsymbol{x}^{(k+1)} + (1/\rho)\boldsymbol{u}^{(k)}$, and $\boldsymbol{u}^{(k)}$ is called the Lagrange multiplier. In this set of equations, subproblem (2) is an inversion module that minimizes f using a quadratic regularization, whereas subproblem (3) is a denoising module that denoises $\widetilde{\boldsymbol{v}}^{(k)}$ using a regularization function g.

E-Mails: {wang470, stanleychan}@purdue.edu.



(a) Input (b) Ground Truth

(c) Total Variation [4] 29.97dB (d) Plug-Play [6] 31.29dB

Fig. 1: Image deblurring using Plug-and-Play ADMM with BM3D denoiser compared to conventional ADMM with total variation (TV) prior. In this example, the regularization parameter is optimally tuned to $\lambda = 10^{-4}$ for Plug-Play, and $\lambda = 5 \times 10^{-3}$ for TV. The internal parameter of Plug-Play is $\rho = 1$. The blur in this example is Gaussian of size 9×9 with radius $\sigma = 1$. The noise level is $5/255$.

Recognizing the inversion-denoising modules of the ADMM allows us to re-design the steps. One possibility is to replace the denoising module by an off-the-shelf image denoising algorithm, i.e.,

$$\boldsymbol{v}^{(k+1)} = \mathcal{D}_\sigma\left(\widetilde{\boldsymbol{v}}^{(k)}\right), \tag{5}$$

for some denoiser $\mathcal{D}_\sigma$ with a noise level $\sigma \overset{\text{def}}{=} \sqrt{\lambda/\rho}$. The resulting algorithm is called the Plug-and-Play ADMM, with the name attributed to Venkatakrishnan et al. [7].

Since the introduction of Plug-and-Play ADMM, many applications have been developed, e.g., X-ray computed tomography [8], image interpolation [9], super-resolution [6, 10], Poisson denoising [11], and single photon imaging [6]. In [12], the same framework was used in camera processing, with a named coined flexible ISP.

To provide readers a quick comparison between Plug-and-Play ADMM and conventional ADMM algorithms, we show in Figure 1 a deblurring result using Plug-and-Play ADMM with BM3D [13] as the denoiser and the same result using conventional ADMM with the total variation regularization. As can be seen in the figure, when both algorithms are tuned to their best parameter λ, Plug-and-Play demonstrates more than 1dB improvement over the total variation.

1.2. Problem of Plug-and-Play

While Plug-and-Play ADMM offers promising image restoration results, it has a significant problem due to the parameter ρ. ρ controls the strength of the intermediate regularization, and is typically assigned by the user. If ρ is set too large, the quadratic regularization in (3) dominates and so the denoiser is weak. If ρ is set too small, the denoiser is strong but the subproblem (2) becomes ill-conditioned. Therefore, finding an optimal ρ is essential to the convergence of the algorithm.

If f is convex and if the denoiser $\mathcal{D}_\sigma$ has a doubly stochastic gradient, Sreehari et al. [8] showed that $\mathcal{D}_\sigma$ is a non-expansive proximal operator and so ρ does not affect the final solution. As a rule-of-thumb, our experience shows that $\rho = 1$ is often a reliable choice. However, for the broader class of bounded denoisers, $\mathcal{D}_\sigma$ could be expansive and so the convergence depends on ρ. In this case, one possible solution is to define a sequence of increasing ρ's such that $\rho_{k+1} = \gamma\rho_k$ for some constant $\gamma > 1$ [6]. This will ensure that the iterates $\boldsymbol{x}^{(k)}$ and $\boldsymbol{v}^{(k)}$ can converge to a fixed point. However, we now have to choose the initial value ρ_0 and the update constant γ. Therefore, if we like to choose ρ properly, an automatic update scheme within the algorithm would be desirable.

1.3. Related Work and Contributions

The contribution of this paper is a *parameter-free* Plug-and-Play ADMM. Here, by parameter-free we mean that the internal parameter ρ is updated as part of the ADMM algorithm, thus is free of tuning. It does not, however, mean that the regularization parameter λ is automatically tuned. Should λ be tuned automatically, a few existing methods can be considered, e.g., SURE [14] and cross validation [15], etc.

Our key idea behind the parameter-free Plug-and-Play ADMM is the Generalized Approximate Message Passing (GAMP) in the compressive sensing literature [16–20]. GAMP is a generic algorithm that solves problems in the form of (1), typically for $f(\boldsymbol{x}) = \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|^2$ with a random matrix $\boldsymbol{A}$ and a regularization function $g(\boldsymbol{x}) = \|\boldsymbol{x}\|_1$. In GAMP, the internal parameters are self-updated, which is a feature this paper attempts to obtain. Another piece of related work is the Denoiser-AMP by Metzler et al. [21], where a denoiser was used to replace the shrinkage step in the classical AMP. Convergence of Denoiser-AMP is known for i.i.d. Gaussian matrix $\boldsymbol{A}$ but not for general matrices.

The goal of this paper is to derive a parameter-free Plug-and-Play ADMM from GAMP. In Section II we provide a brief introduction to the GAMP algorithm. Then in Section III, we show how the GAMP algorithm can be modified into a parameter-free Plug-and-Play ADMM. Experimental results are shown in Section IV.

2. GENERALIZED APPROXIMATE MESSAGE PASSING

In this section we provide a brief introduction to the generalized approximate message passing (GAMP) algorithm. For full discussions of GAMP, we refer the readers to [18]. Among all image restoration problems, we are particularly interested in the deblurring problem with f in the form

$$f(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|^2,$$

where $\boldsymbol{x} \in \mathbb{R}^n$ is the unknown variable, $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ a convolution matrix, and $\boldsymbol{y} \in \mathbb{R}^n$ is the observed signal. The regularization function g is unimportant for Plug-and-Play ADMM, because we will later replace it by a denoiser. However, since the conventional GAMP requires an explicit g, we will keep the function g in this section with the assumption that it is separable. We will remove g in Section III.

2.1. GAMP

The GAMP algorithm begins by considering the following problem

$$\begin{array}{ll} \underset{\boldsymbol{x},\boldsymbol{v}}{\text{minimize}} & \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|^2 + \lambda g(\boldsymbol{v}) \\ \text{subject to} & \boldsymbol{x} = \boldsymbol{v}. \end{array} \tag{6}$$

Clearly, at the optimal point, (6) has the same solution as the original unconstrained problem.

The next step GAMP does is to separately consider $f(\boldsymbol{x})$ and $g(\boldsymbol{v})$ as the *output node* and the *input node* of a bipartite graph, respectively. Then the algorithm seeks the equilibrium of the two sides, by passing intermediate variables (called *messages*) forward and backward between the nodes. Specifically, by initializing two vectors $\boldsymbol{\tau}_x^{(0)} = \mathbf{1}_{n\times 1}$ and $\boldsymbol{u}^{(0)} = \mathbf{0}_{n\times 1}$, the computation on the **output node** involves:

$$\widetilde{\boldsymbol{x}}^{(k+1)} = \boldsymbol{x}^{(k)} - \boldsymbol{\tau}_x^{(k)} \cdot \boldsymbol{u}^{(k)}, \tag{7}$$

$$\boldsymbol{x}^{(k+1)} = \mathrm{prox}_{\boldsymbol{\tau}_x^{(k)} f}(\widetilde{\boldsymbol{x}}^{(k)}), \tag{8}$$

$$\boldsymbol{\pi}_x^{(k+1)} = \tau_x^{(k)} \cdot \frac{\partial}{\partial \widetilde{\boldsymbol{x}}} \mathrm{prox}_{\boldsymbol{\tau}_x^{(k)} f}(\widetilde{\boldsymbol{x}})\Big|_{\widetilde{\boldsymbol{x}}=\widetilde{\boldsymbol{x}}^{(k)}}, \tag{9}$$

$$\boldsymbol{\tau}_v^{(k+1)} = (\boldsymbol{\tau}_x^{(k)} - \boldsymbol{\pi}_x^{(k+1)})./(\boldsymbol{\tau}_x^{(k)})^2, \tag{10}$$

$$\boldsymbol{u}^{(k+1)} = (\boldsymbol{x}^{(k+1)} - \widetilde{\boldsymbol{x}}^{(k+1)})./\boldsymbol{\tau}_x^{(k)}. \tag{11}$$

In this set of equations, the function $\mathrm{prox}_{\boldsymbol{\tau} f}$ is the proximal operator, defined as

$$\mathrm{prox}_{\boldsymbol{\tau} f}(\widetilde{\boldsymbol{x}}) = \underset{\boldsymbol{x}}{\mathrm{argmin}} \ \ \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_{\boldsymbol{\tau}}^2 + \frac{1}{2}\|\boldsymbol{x} - \widetilde{\boldsymbol{x}}\|^2, \tag{12}$$

where the norm $\|\cdot\|_{\boldsymbol{\tau}}^2$ is a weighted norm given by $\|\boldsymbol{x}\|_{\boldsymbol{\tau}}^2 = \sum_{i=1}^n \tau_i x_i^2$. The variable $\boldsymbol{\tau}_x$ can be regarded as a vector version of the internal parameter ρ in ADMM, and $\boldsymbol{\pi}_x$ can be regarded as a measure of the variance $\boldsymbol{x}$ conditioned on $\boldsymbol{u}$.

The computation on the **input node** involves

$$\widetilde{\boldsymbol{v}}^{(k+1)} = \boldsymbol{v}^{(k)} + \boldsymbol{\tau}_v^{(k+1)} \cdot \boldsymbol{u}^{(k+1)}, \tag{13}$$

$$\boldsymbol{v}^{(k+1)} = \mathrm{prox}_{\boldsymbol{\tau}_v^{(k+1)}\lambda g}(\widetilde{\boldsymbol{v}}^{(k)}), \tag{14}$$

$$\boldsymbol{\pi}_v^{(k+1)} = \boldsymbol{\tau}_v^{(k+1)} \cdot \frac{\partial}{\partial \widetilde{\boldsymbol{v}}} \mathrm{prox}_{\boldsymbol{\tau}_v^{(k)}\lambda g}(\widetilde{\boldsymbol{v}})\Big|_{\widetilde{\boldsymbol{v}}=\widetilde{\boldsymbol{v}}^{(k)}}, \tag{15}$$

$$\boldsymbol{\tau}_x^{(k+1)} = \boldsymbol{\pi}_v^{(k+1)}. \tag{16}$$

For separable $g(\boldsymbol{v}) = \sum_{i=1}^n g_i(v_i)$, the proximal operator $\mathrm{prox}_{\boldsymbol{\tau} g}(\widetilde{\boldsymbol{v}})$ reads as

$$\mathrm{prox}_{\tau_i \lambda g}(\widetilde{v}_i) = \underset{v}{\mathrm{argmin}} \ \ \tau_i \lambda g_i(v) + \frac{1}{2}(v - \widetilde{v}_i)^2. \tag{17}$$

2.2. Equivalence between GAMP and ADMM

In the above input and output computation, if we ignore Equations (9)–(10) and (15)–(16), we will arrive at an ADMM algorithm with *vector-valued* parameters $\boldsymbol{\tau}_x$ and $\boldsymbol{\tau}_v$, instead of a common scalar parameter ρ. More specifically, GAMP and ADMM are related according to the following theorem [17]:

Theorem 1 *The iterates of the GAMP satisfy*

$$\begin{aligned}
\boldsymbol{v}^{(k+1)} &= \underset{\boldsymbol{v}}{\textit{argmin}} \;\; L(\boldsymbol{x}^{(k)}, \boldsymbol{v}, \boldsymbol{u}^{(k)}) + \frac{1}{2}\|\boldsymbol{v} - \boldsymbol{v}^{(k)}\|^2_{\boldsymbol{\tau}_v^{(k)}}, \\
\boldsymbol{x}^{(k+1)} &= \underset{\boldsymbol{x}}{\textit{argmin}} \;\; L(\boldsymbol{x}, \boldsymbol{v}^{(k+1)}, \boldsymbol{u}^{(k)}) + \frac{1}{2}\|\boldsymbol{x} - \boldsymbol{v}^{(k+1)}\|^2_{\boldsymbol{\tau}_x^{(k)}}, \\
\boldsymbol{u}^{(k+1)} &= \boldsymbol{u}^{(k)} + \frac{1}{\boldsymbol{\tau}_x^{(k)}}(\boldsymbol{x}^{(k+1)} - \boldsymbol{v}^{(k+1)}),
\end{aligned}$$

where $L(\boldsymbol{x}, \boldsymbol{v}, \boldsymbol{u}) = f(\boldsymbol{x}) + \lambda g(\boldsymbol{v}) + \boldsymbol{u}^T(\boldsymbol{x} - \boldsymbol{v})$ is the Lagrangian function.

Therefore, the key difference between GAMP and ADMM is the parameters $\boldsymbol{\tau}_v^{(k)}$ and $\boldsymbol{\tau}_x^{(k)}$. This suggests that if we want to derive a Plug-and-Play ADMM from GAMP, we must first define the parameters $\boldsymbol{\tau}_v^{(k)}$ and $\boldsymbol{\tau}_x^{(k)}$.

3. PARAMETER-FREE PLUG-AND-PLAY

We now derive the parameter-free Plug-and-Play using the GAMP formulation above. There are two major modifications we need for the derivation.

- The vector-valued parameters $\boldsymbol{\tau}_x$ and $\boldsymbol{\tau}_v$ should become scalars τ_x and τ_v. This would allow us to consider arbitrary denoisers which are not-necessarily separable.
- The proximal operator in (14) is replaced by an off-the-shelf denoiser as defined in (5) so that it fits the Plug-and-Play framework.

With these two modifications we can consider the output and the input nodes. We also note that among the equations (7)-(16), the biggest challenges are the proximal operators. The following two subsections will address these operators.

3.1. Output Node

For a convolution matrix $\boldsymbol{A}$, the proximal operator can be shown as

$$\begin{aligned}
\text{prox}_{\tau_x f}(\widetilde{\boldsymbol{x}}) &= \underset{\boldsymbol{x}}{\text{argmin}} \;\; \frac{\tau_x}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|^2 + \frac{1}{2}\|\boldsymbol{x} - \widetilde{\boldsymbol{x}}\|^2 \\
&= \left(\tau_x \boldsymbol{A}^T\boldsymbol{A} + \boldsymbol{I}\right)^{-1}\left(\tau_x \boldsymbol{A}^T\boldsymbol{y} + \widetilde{\boldsymbol{x}}\right). \qquad (18)
\end{aligned}$$

Taking derivative with respect to $\widetilde{\boldsymbol{x}}$ yields

$$\frac{\partial}{\partial \widetilde{\boldsymbol{x}}}\text{prox}_{\tau_x f}(\widetilde{\boldsymbol{x}}) = \left(\tau_x \boldsymbol{A}^T\boldsymbol{A} + \boldsymbol{I}\right)^{-1}\mathbf{1} \qquad (19)$$

Note that this is a vector of gradients. Since we are looking for a scalar, one option is to consider the divergence. This yields

$$\begin{aligned}
\text{div}\left\{\text{prox}_{\tau_x f}(\widetilde{\boldsymbol{x}})\right\} &= \frac{1}{n}\mathbf{1}^T\left(\tau_x \boldsymbol{A}^T\boldsymbol{A} + \boldsymbol{I}\right)^{-1}\mathbf{1} \qquad (20) \\
&\overset{(a)}{=} \frac{1}{n}\mathbf{1}^T\boldsymbol{F}^T\left(\tau_x|\boldsymbol{\Lambda}|^2 + \boldsymbol{I}\right)^{-1}\boldsymbol{F}\mathbf{1} \\
&\overset{(b)}{=} (\tau_x|\lambda_{1,1}|^2 + 1)^{-1} = (\tau_x + 1)^{-1},
\end{aligned}$$

where in (a) we used the fact that a convolution matrix $\boldsymbol{A}$ is diagonalizable by the Fourier transform matrix $\boldsymbol{F}$ to yield $\boldsymbol{A} = \boldsymbol{F}^T\boldsymbol{\Lambda}\boldsymbol{F}$, and in (b) we observe that $\boldsymbol{F}\mathbf{1} = \sqrt{n}[1, 0, \ldots, 0]^T$. The scalar $\lambda_{1,1}$ is the first entry of the eigenvalue matrix $\boldsymbol{\Lambda}$, which is 1 for convolutional matrices. Substituting this result into (9), we have

$$\pi_x^{(k+1)} = \tau_x^{(k)}/(\tau_x^{(k)} + 1). \qquad (21)$$

3.2. Input Node

On the input node, we have to replace the proximal operator by a denoiser $\mathcal{D}_\sigma$. Here, the noise level of the denoiser, σ, should be defined as $\sigma = \sqrt{\tau_x\lambda}$. This explains (14).

For the derivative in (15), we note that since τ_v is now a scalar, we have to replace the derivative by its divergence (which is the sum of gradients). This gives

$$\pi_v^{(k+1)} = \tau_v^{(k+1)}\text{div}\mathcal{D}_\sigma(\widetilde{\boldsymbol{v}}^{(k)}). \qquad (22)$$

Calculating the divergence can be performed numerically using a Monte Carlo scheme. More specifically, the divergence of the denoiser at $\widetilde{\boldsymbol{v}}$ can be approximated by

$$\begin{aligned}
\text{div}\mathcal{D}_\sigma(\widetilde{\boldsymbol{v}}) &= \lim_{\epsilon\to 0}\mathbb{E}_{\boldsymbol{b}}\left\{\boldsymbol{b}^T\left(\frac{\mathcal{D}_\sigma(\widetilde{\boldsymbol{v}} + \epsilon\boldsymbol{b}) - \mathcal{D}_\sigma(\widetilde{\boldsymbol{v}})}{\epsilon}\right)\right\} \\
&\approx \boldsymbol{b}^T\left(\frac{\mathcal{D}_\sigma(\widetilde{\boldsymbol{v}} + \epsilon\boldsymbol{b}) - \mathcal{D}_\sigma(\widetilde{\boldsymbol{v}})}{n\epsilon}\right), \qquad (23)
\end{aligned}$$

where $\boldsymbol{b} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ is a random vector, and $\epsilon \ll 1$ is a small constant (typically $\epsilon = 10^{-3}$). The approximation of the expectation generally holds for large n due to concentration of measure [14].

Numerically, computing (23) only requires evaluating the denoiser twice: once for $\mathcal{D}_\sigma(\widetilde{\boldsymbol{v}})$, and the other time for $\mathcal{D}_\sigma(\widetilde{\boldsymbol{v}} + \epsilon\boldsymbol{b})$.

3.3. Final Algorithm

The final algorithm can be derived by substituting (18) into (8), (21) into (9) for the output nodes, and (5) into (14), (22) into (15) for the input nodes. Moreover, we can simplify steps by defining

$$\rho_x = 1/\tau_x, \qquad \rho_v = 1/\tau_v. \qquad (24)$$

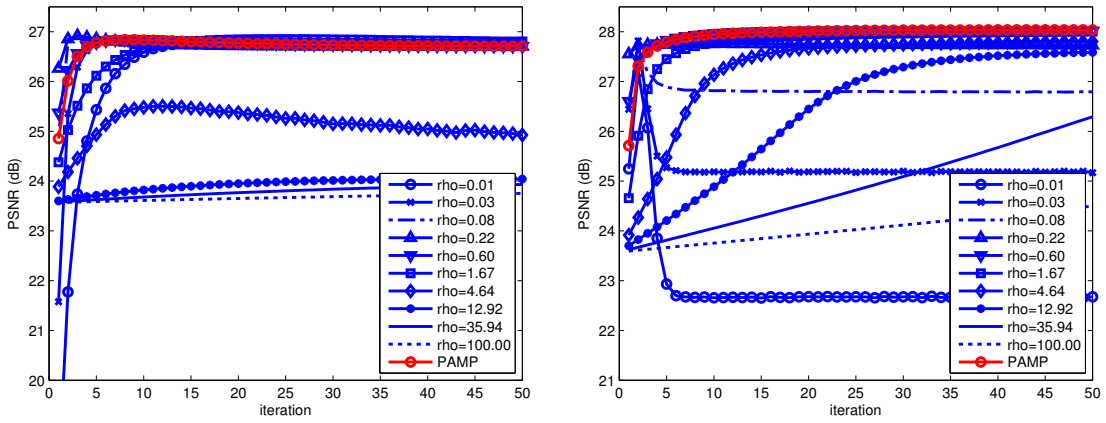
Then we can show that the GAMP algorithm can be simplified to Algorithm 1. We call the resulting algorithm Plug-and-Play generalized approximate message passing (PAMP).

As shown in Algorithm 1, the difference between PAMP and Plug-and-Play ADMM is the parameters ρ_x and ρ_v. In Plug-and-Play, the parameters share the same value ρ and is fixed throughout the iterations. In PAMP, ρ_x and ρ_v are automatically updated as part of the algorithm. Therefore, PAMP is a *parameter-free* algorithm.

4. EXPERIMENTAL RESULTS

In this section we present experimental results to evaluate the performance of the proposed PAMP algorithm. We focus on the image deblurring problem, although the method can easily be extended to image interpolation and image super-resolution.

We test the algorithm using 10 standard gray-scale images. Each image is blurred by a spatially invariant Gaussian blur kernel of size 9×9 and standard deviation 1. Additive i.i.d. Gaussian noise of zero mean and standard deviation $\sigma = 5/255$ is added to the blurred image. Two denoisers $\mathcal{D}_\sigma$ are considered in this experiment: total variation denoising [22], and BM3D [13]. For total variation, we



(a) Total Variation Denoiser (b) BM3D Denoiser

Fig. 2: PSNR of PAMP compared to Plug-and-Play ADMM using a fixed ρ. (a) using total variation denoising as the denoiser; (b) using BM3D as the denoiser. In this figure, all PSNR values are averaged over 10 testing images.

Algorithm 1 Proposed Algorithm: PAMP

Initialize $\boldsymbol{u}^{(0)} = \boldsymbol{x}^{(0)} = \boldsymbol{0}$, $\rho_v^{(0)} = 1$.
for $k = 0, 1, \ldots, k_{\max}$ **do**
  % ($\boldsymbol{v}$-subproblem)
  $\widetilde{\boldsymbol{v}} = \boldsymbol{x}^{(k)} + (1/\rho_v^{(k)})\boldsymbol{u}^t$
  $\boldsymbol{v}^{(k+1)} = \mathcal{D}_\sigma(\widetilde{\boldsymbol{v}})$, where $\sigma = \sqrt{\lambda/\rho_v^{(k)}}$
  $\rho_x^{(k+1)} = \rho_v^{(k)}/\mathrm{div}\mathcal{D}_\sigma(\widetilde{\boldsymbol{v}})$

  % ($\boldsymbol{x}$-subproblem)
  $\widetilde{\boldsymbol{x}} = \boldsymbol{v}^{(k+1)} - (1/\rho_x^{(k+1)})\boldsymbol{u}^{(k)}$
  $\boldsymbol{x}^{(k+1)} = (\boldsymbol{A}^T\boldsymbol{A} + \rho_x^{(k+1)}\boldsymbol{I})^{-1}(\boldsymbol{A}^T\boldsymbol{y} + \rho_x^{(k+1)}\widetilde{\boldsymbol{x}})$
  $\rho_v^{(k+1)} = \rho_x^{(k+1)}/(\rho_x^{(k+1)} + 1)$

  % (Multiplier update)
  $\boldsymbol{u}^{(k+1)} = \boldsymbol{u}^{(k)} + \rho_x^{(k+1)}(\boldsymbol{x}^{(k+1)} - \boldsymbol{v}^{(k+1)})$
end for

use the MATLAB implementation in [4], whereas for BM3D, we use the code available on author's website. When total variation is used, we set the regularization parameter $\lambda = 10^{-2}$. When BM3D is used, we set $\lambda = 10^{-3}$. These values are selected as they produce the best overall result for the entire dataset.

Since the objective of PAMP is to automatically select ρ, we compare PAMP with Plug-and-Play ADMM which uses a fixed ρ. We select 10 values of ρ from 10^{-2} to 10^2 in the logarithmic scale. For each ρ, we run Plug-and-Play ADMM for 50 iterations and record the PSNR values. For PAMP, we initialize the algorithm with $\rho_v^{(0)} = 1$ and let the algorithm to update ρ_x and ρ_v internally.

The results are shown in Figure 2. In this figure, the PSNR values are averaged over the 10 testing images. As can be observed, the parameter ρ has important influence to the Plug-and-Play ADMM algorithm where large ρ tends to converge slower and approaches a solution with low PSNR. If ρ is too small, e.g., $\rho = 0.01$ in the BM3D case, the PSNR actually drops rapid after the first iteration. As for PAMP, we observe that in both denoisers the solution picks a rapid convergence path with almost the highest PSNR.

Additional results, including other types of blur and other noise levels, can be found at https://engineering.purdue.edu/ChanGroup/

5. DISCUSSION AND CONCLUSION

Why it works? Line 6 and Line 11 of Algorithm 1 reveals that there are two opposite forces in updating ρ_v and ρ_x:

$$\rho_x \leftarrow \rho_v/\mathrm{div}\mathcal{D}_\sigma(\widetilde{\boldsymbol{v}}), \tag{25}$$

$$\rho_v \leftarrow \rho_x/(\rho_x + 1) = \rho_x \ \mathrm{div}\left\{\mathrm{prox}_{(1/\rho_x)f}(\widetilde{\boldsymbol{x}})\right\}. \tag{26}$$

Divergence of a function is an indicator of the sensitivity with respect to the input. When $\mathrm{div}\mathcal{D}_\sigma$ is large, the denoiser $\mathcal{D}_\sigma$ behaves sensitively at $\widetilde{\boldsymbol{v}}$ and so the denoised output is less reliable. Thus, PAMP makes ρ_x small to attenuate the influence of the denoiser when solving the inversion. Now, since ρ_x becomes small, the inversion is weak and so in the next iteration a strong denoiser is needed. This is achieved by decreasing ρ_v in (26). These two opposite forces form a trajectory of the pair (ρ_x, ρ_v). As $k \to \infty$, one can show that (ρ_x, ρ_v) approaches a steady state where the two divergence terms coincides: $\mathrm{div}\mathcal{D}_\sigma(\widetilde{\boldsymbol{v}}) = \mathrm{div}\left\{\mathrm{prox}_{(1/\rho_x)f}(\widetilde{\boldsymbol{x}})\right\}$.

Convergence? Convergence of GAMP is an open problem. The best result we know so far is that with appropriately designed damping strategies, GAMP converges for strictly convex functions f and g [23]. However, when damping is not used, there are examples where GAMP diverges [24]. Moving from an explicit g to an implicit denoiser $\mathcal{D}_\sigma$ will cause additional challenges yet to be studied.

Conclusion. Plug-and-Play generalized approximate message passing (PAMP) is a new algorithm that automatically updates the internal parameters of a conventional Plug-and-Play ADMM. The update rules are based on the measure of the divergence of the subproblems, and are derived from the generalized approximate message passing (GAMP). At the current stage, numerical results show that PAMP is a promising algorithm as it generates solutions through a rapid convergence path. Intuitive arguments of the algorithm are made. Future work should focus on the convergence analysis.

6. REFERENCES

[1] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Found. Trends Mach. Learn.*, vol. 3, no. 1, pp. 1–122, Jan. 2011.

[2] J. Yang, Y. Zhang, and W. Yin, "An efficient TVL1 algorithm for deblurring multichannel images corrupted by impulsive noise," *SIAM J. on Sci. Comput.*, vol. 31, no. 4, pp. 2842–2865, Jul. 2009.

[3] M. Afonso, J. Bioucas-Dias, and M. Figueiredo, "Fast image recovery using variable splitting and constrained optimization," *IEEE Trans. Image Process.*, vol. 19, no. 9, pp. 2345–2356, Apr. 2010.

[4] S. H. Chan, R. Khoshabeh, K. B. Gibson, P. E. Gill, and T. Q. Nguyen, "An augmented Lagrangian method for total variation video restoration," *IEEE Trans. Image Process.*, vol. 20, no. 11, pp. 3097–3111, May 2011.

[5] C. A. Bouman, "Model-based image processing," Available online at `https://engineering.purdue.edu/~bouman/publications/pdf/MBIP-book.pdf`, 2015.

[6] S. H. Chan, X. Wang, and O. A. Elgendy, "Plug-and-Play ADMM for image restoration: Fixed point convergence and applications," *IEEE Trans. Computational Imaging*, In Press. Available online at `http://arxiv.org/abs/1605.01710`.

[7] S. Venkatakrishnan, C. Bouman, and B. Wohlberg, "Plug-and-play priors for model based reconstruction," in *Proc. IEEE Global Conference on Signal and Information Processing*, 2013, pp. 945–948.

[8] S. Sreehari, S. V. Venkatakrishnan, B. Wohlberg, G. T. Buzzard, L. F. Drummy, J. P. Simmons, and C. A. Bouman, "Plug-and-play priors for bright field electron tomography and sparse interpolation," *IEEE Trans. Computational Imaging*, vol. 2, no. 4, pp. 408 – 423, Dec. 2016.

[9] S. H. Chan, "Algorithm-induced prior for image restoration," Available online at `http://arxiv.org/abs/1602.00715`, Feb. 2016.

[10] A. Brifman, Y. Romano, and M. Elad, "Turning a denoiser into a super-resolver using plug and play priors," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, Sep. 2016, pp. 1404–1408.

[11] A. Rond, R. Giryes, and M. Elad, "Poisson inverse problems by the plug-and-play scheme," *Journal of Visual Communication and Image Representation*, vol. 41, pp. 96–108, Nov. 2015.

[12] F. Heide, M. Steinberger, Y.-T. Tsai, M. Rouf, D. Pajak, D. Reddy, O. Gallo, J. Liu abd W. Heidrich, K. Egiazarian, J. Kautz, and K. Pulli, "FlexISP: A flexible camera image processing framework," *ACM Transactions on Graphics (Proceedings SIGGRAPH Asia 2014)*, vol. 33, no. 6, Dec. 2014.

[13] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian, "Image denoising by sparse 3D transform-domain collaborative filtering," *IEEE Trans. Image Process.*, vol. 16, no. 8, pp. 2080–2095, Aug. 2007.

[14] S. Ramani, T. Blu, and M. Unser, "Monte-Carlo SURE: A black-box optimization of regularization parameters for general denoising algorithms," *IEEE Trans. Image Process.*, vol. 17, no. 9, pp. 1540–1554, 2008.

[15] N. Nguyen, P. Milanfar, and G. Golub, "Efficient generalized cross-validation with applications to parametric image restoration and resolution enhancement," *IEEE Trans. Image Process.*, vol. 10, no. 9, pp. 1299–1308, Sep. 2001.

[16] M. Borgerding and P. Schniter, "Generalized approximate message passing for the cosparse analysis model," in *Proc. IEEE Int. Conf. Acoustics, Speech, Signal Process. (ICASSP)*, 2015, pp. 3756–3760.

[17] S. Rangan, P. Schniter, E. Riegler, A. Fletcher, and V. Cevher, "Fixed points of generalized approximate message passing with arbitrary matrices," in *Proc. IEEE Int. Symp. Information Theory*, 2013, pp. 664–668.

[18] S. Rangan, "Generalized approximate message passing for estimation with random linear mixing," in *Proc. IEEE Int. Symp. Information Theory*, 2011, pp. 2168–2172.

[19] D. L. Donoho, A. Maleki, and A. Montanari, "Message passing algorithms for compressed sensing," *Proc. Nat. Acad. Sci.*, vol. 106, no. 45, pp. 18914–18919, 2009.

[20] D. L. Donoho, A. Maleki, and A. Montanari, "How to design message passing algorithms for compressed sensing," Tech. Rep., Rice University, 2011, Available online at http://www.ece.rice.edu/ mam15/bpist.pdf.

[21] C. A. Metzler, A. Maleki, and R. G. Baraniuk, "From denoising to compressed sensing," *IEEE Trans. Information Theory*, vol. 62, no. 9, pp. 5117–5144, Sep. 2016.

[22] L. Rudin, S. Osher, and E. Fatemi, "Nonlinear total variation based noise removal algorithms," *Physica D*, vol. 60, pp. 259–268, 1992.

[23] J. Vila, P. Schniter, S. Rangan, F. Krzakala, and L. Zdeborova, "Adaptive damping and mean removal for the generalized approximate message passing algorithm," in *Proc. IEEE Int. Conf. Acoustics, Speech, Signal Process. (ICASSP)*, 2015, pp. 2021–2025.

[24] S. Rangan, P. Schniter, and A. K. Fletcher, "On the convergence of approximate message passing with arbitrary matrices," in *Proc. IEEE Int. Symp. Information Theory*, 2014, pp. 236–240.