DOLPHIn - Dictionary Learning for Phase Retrieval
Andreas Tillmann, Yonina Eldar, Julien Mairal

To cite this version:
Andreas Tillmann, Yonina Eldar, Julien Mairal. DOLPHIn - Dictionary Learning for Phase Retrieval. IEEE Transactions on Signal Processing, 2016, 64 (24), pp.6485-6500. 10.1109/TSP.2016.2607180. hal-01387428

HAL Id: hal-01387428
https://inria.hal.science/hal-01387428
Submitted on 25 Oct 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
DOLPHIn – Dictionary Learning for Phase Retrieval

Andreas M. Tillmann, Yonina C. Eldar, Fellow, IEEE, and Julien Mairal, Senior Member, IEEE

Abstract—We propose a new algorithm to learn a dictionary for reconstructing and sparsely encoding signals from measurements without phase. Specifically, we consider the task of estimating a two-dimensional image from squared-magnitude measurements of a complex-valued linear transformation of the original image. Several recent phase retrieval algorithms exploit underlying sparsity of the unknown signal in order to improve recovery performance. In this work, we consider such a sparse signal prior in the context of phase retrieval, when the sparsifying dictionary is not known in advance. Our algorithm jointly reconstructs the unknown signal—possibly corrupted by noise—and learns a dictionary such that each patch of the estimated image can be sparsely represented. Numerical experiments demonstrate that our approach can obtain significantly better reconstructions for phase retrieval problems with noise than methods that cannot exploit such “hidden” sparsity. Moreover, on the theoretical side, we provide a convergence result for our method.

Index Terms—(MLR-DICT, MLR-LEAR, OPT-NCVX, OPT-SOFT) Machine Learning, Signal Reconstruction, Image Reconstruction

I. INTRODUCTION

Phase retrieval has been an active research topic for decades [1], [2]. The underlying goal is to estimate an unknown signal from the modulus of a complex-valued linear transformation of the signal. With such nonlinear measurements, the phase information is lost (hence the name “phase retrieval”), rendering the recovery task ill-posed and, perhaps not surprisingly, NP-hard [3]. Traditional approaches consider cases where the solution is unique up to a global phase shift, which can never be uniquely resolved, and devise signal reconstruction algorithms for such settings. Uniqueness properties and the empirical success of recovery algorithms usually hinge on oversampling the signal, i.e., taking more measurements than the number of signal components.

The most popular techniques for phase retrieval are based on alternating projections, see [4], [5], [6] for overviews. These methods usually require precise prior information about the signal (such as knowledge of the support set) and often converge to erroneous results. More recent approaches include semidefinite programming relaxations [7], [8], [9], [10], [11] and gradient-based methods such as Wirtinger Flow [12], [13].

In recent years, new phase retrieval techniques were developed for recovering sparse signals, which are linear combinations of only a few atoms from a known dictionary [8], [14], [13], [15]. With a sparsity assumption, these algorithms obtained better recovery performance than traditional non-sparse approaches. The main idea is akin to compressed sensing, where one works with fewer (linear) measurements than signal components [16], [17], [18]. An important motivation for developing sparse recovery techniques was that many classes of signals admit a sparse approximation in some basis or overcomplete dictionary [19], [20], [21]. While sometimes such dictionaries are known explicitly, better results have been achieved by adapting the dictionary to the data, e.g., for image denoising [20]. Numerous algorithms have been developed for this task, see, e.g., [19], [22], [23]. In this traditional setting, the signal measurements are linear and a large database of training signals is used to train the dictionary.

In this work, we propose a dictionary learning formulation for simultaneously solving the signal reconstruction and sparse representation problems given nonlinear, phaseless and noisy measurements. To optimize the resulting (nonconvex) objective function, our algorithm—referred to as DOLPHIn (DictiOnary Learning for PHase retrieval)—alternates between several minimization steps, thus monotonically reducing the value of the objective until a stationary point is found (if step sizes are chosen appropriately). Specifically, we iterate between best fitting the data and sparsely representing the recovered signal. DOLPHIn combines projected gradient descent steps to update the signal, iterative shrinkage to obtain a sparse approximation [24], and block-coordinate descent for the dictionary update [23].

In various experiments on image reconstruction problems, we demonstrate the ability of DOLPHIn to achieve significantly improved results when the oversampling ratio is low and the noise level high, compared to the recent state-of-the-art Wirtinger Flow (WF) method [12], which cannot exploit sparsity if the dictionary is unknown. In this two-dimensional setting, we break an image down into small patches and train a dictionary such that each patch can be sparsely represented using this dictionary. The patch size as well as the amount of overlap between patches can be freely chosen, which allows us to control the trade-off between the amount of computation required to reconstruct the signal and the quality of the result.

The paper is organized as follows: In Sections II and III,
we introduce the DOLPHIn framework and algorithm. Then, in Section IV, we present numerical experiments and implementation details, along with discussions about (hyper-)parameter selection and variants of DOLPHIn. We conclude the paper in Section V. The appendix provides further details on the mathematical derivation of the DOLPHIn algorithm. A short preliminary version of this work appeared in the conference paper [25].

II. PHASE RETRIEVAL MEETS DICTIONARY LEARNING

In mathematical terms, the phase retrieval problem can be formulated as solving a nonlinear system of equations:

\[
\text{Find } x \in \mathcal{X} \subseteq \mathbb{C}^N \text{ s.t. } |f_i(x)|^2 = y_i \quad \forall i = 1, \ldots, M,
\]

(1)

where the functions \( f_i : \mathbb{C}^N \to \mathbb{C} \) are linear operators and the scalars \( y_i \) are nonlinear measurements of the unknown original signal \( \hat{x} \) in \( \mathcal{X} \), obtained by removing the phase information. The set \( \mathcal{X} \) represents constraints corresponding to additional prior information about \( x \). For instance, when dealing with real-valued bounded signals, this may typically be a box constraint \( \mathcal{X} = [0, 1]^N \). Other common constraints include information about the support set—that is, the set of nonzero coefficients of \( x \). Classical phase retrieval concerns the recovery of \( x \) given the (squared) modulus of the signal’s Fourier transform. Other commonly considered cases pertain to randomized measurements (\( f_i \) are random linear functions) or coded diffraction patterns, i.e., concatenations of random signal masks and Fourier transforms (see, e.g., [2], [12]).

A. Prior and Related Work

The most popular methods for classical phase retrieval—Fienup’s algorithm [5] and many related approaches [2], [4], [6], [26], [27]—are based on alternating projections onto the sets \( \mathcal{Y} := \{ x \in \mathbb{C}^N \text{ s.t. } |f_i(x)| = y_i \ \forall i \} \) or \( \{ x \in \mathbb{C}^N \text{ s.t. } |f_i(x)|^2 = y_i \ \forall i \} \) and onto the set \( \mathcal{X} \). However, the nonconvexity of \( \mathcal{Y} \) makes the projection not uniquely defined and possibly hard to compute. The success of such projection-based methods hinges critically on precise prior knowledge (which, in general, will not be available in practice) and on the choice of a projection operator onto \( \mathcal{Y} \). Ultimately, convergence to \( \hat{x} \) (up to global phase) is in general not guaranteed and these methods often fail in practice.

Further algorithmic techniques to tackle (1) include two different semidefinite relaxation approaches, PhaseLift [7] and PhaseCut [11]. PhaseCut “lifts” (1) into the space of (complex) positive semidefinite rank-1 matrices via the variable transformation \( \mathbf{X} := \text{xx}^T \). Then, the nonlinear constraints \( |f_i(x)|^2 = y_i \) are equivalent to linear constraints with respect to the matrix variable \( \mathbf{X} \). By suitably relaxing the immediate but intractable rank-minimization objective, one obtains a convex semidefinite program (SDP). Similarly, PhaseCut introduces a separate variable \( \mathbf{u} \) for the phase, allowing to eliminate \( x \), and then lifts \( \mathbf{u} \) to obtain an equivalent problem with a rank-1 constraint, which can be dropped to obtain a different SDP relaxation of (1). Despite some guarantees on when these relaxations are tight, i.e., allow for correctly recovering the solution to (1) (again up to a global phase factor), their practical applicability is limited due to the dimension of the SDP that grows quadratically with the problem dimension.

A recent method that works in the original variable space is the so-called Wirtinger Flow algorithm [12]. Here, (1) is recast as the optimization problem

\[
\min_{x \in \mathbb{C}^N} \frac{1}{M} \sum_{i=1}^{M} (|f_i(x)|^2 - y_i)^2,
\]

(2)

which is approximately solved by a gradient descent algorithm. Note that in the case of complex variables, the concept of a gradient is not well-defined, but as shown in [12], a strongly related expression termed the “Wirtinger derivative” can be used instead and indeed reduces to the actual gradient in the real case. For the case of i.i.d. Gaussian random measurements, local convergence with high probability can be proven for the method, and a certain spectral initialization provides sufficiently accurate signal estimates for these results to be applicable. Further variants of the Wirtinger Flow (WF) method that have been investigated are the Truncated WF [28], which involves improving search directions by a statistically motivated technique to filter out components that bear “too much” influence, and Thresholded WF [13], which allows for improved reconstruction of sparse signals (i.e., ones with only a few significant components or nonzero elements), in particular when the measurements are corrupted by noise.

The concept of sparsity has been successfully employed in the context of signal reconstruction from linear measurements, perhaps most prominently in the field of compressed sensing [16], [17], [18], [29] during the past decade. There, the task is to recover an unknown signal \( \hat{x} \in \mathbb{C}^N \) from \( M < N \) linear measurements—that is, finding the desired solution among the infinitely many solutions of an underdetermined system of linear equations. For signals that are (exactly or approximately) sparse with respect to some basis or dictionary, i.e., when \( \hat{x} \approx \mathbf{D} \hat{\mathbf{a}} \) for a matrix \( \mathbf{D} \) and a vector \( \hat{\mathbf{a}} \) that has few nonzero entries, such recovery problems have been shown to be solvable in a very broad variety of settings and applications, and with a host of different algorithms. Dictionaries enabling sparse signal representations are sometimes, but not generally, known in advance. The goal of dictionary learning is to improve upon the sparsity achievable with a given (analytical) dictionary, or to find a suitable dictionary in the first place. Given a set of training signals, the task consists of finding a dictionary such that every training signal can be well-approximated by linear combinations of just a few atoms. Again, many methods have been developed for this purpose (see, e.g., [19], [20], [21], [22], [23]) and demonstrated to work well in different practical applications.

Signal sparsity (or compressibility) can also be beneficially exploited in phase retrieval methods, cf. [8], [9], [13], [14], [15]. However, to the best of our knowledge, existing methods assume that the signal is sparse itself or sparse with respect to a fixed pre-defined dictionary. This motivates the development of new algorithms and formulations to jointly learn suitable dictionaries and reconstruct input signals from nonlinear measurements.
B. Dictionary Learning for Phase Retrieval

In this paper, we consider the problem of phase retrieval by focusing on image reconstruction applications. Therefore, we will work in a two-dimensional setting directly. However, it should be noted that all expressions and algorithms can also easily be formulated for one-dimensional signals like (1), as detailed in the appendix. We will also consider the case of noisy measurements, and will show that our approach based on dictionary learning is particularly robust to noise, which is an important feature in practice.

Concretely, we wish to recover an image \( \hat{X} \) in \([0, 1]^{N_1 \times N_2}\) from noise-corrupted phasorless nonlinear measurements

\[
Y := |F(\hat{X})|^2 + N, \tag{3}
\]

where \( F : \mathbb{C}^{N_1 \times N_2} \to \mathbb{C}^{M_1 \times M_2} \) is a linear operator, \( N \) is a real matrix whose entries represent noise, and the complex modulus and squares are taken component-wise. As mentioned earlier, signal sparsity is known to improve the performance of phase retrieval algorithms, but a sparsifying transform is not always known in advance, or a better choice than a predefined selection can sometimes be obtained by adapting the dictionary to the data. In the context of image reconstruction, this motivates learning a dictionary \( D \in \mathbb{R}^{p \times n} \) such that each \( s_1 \times s_2 \) patch \( x^j \) of \( X \), represented as a vector of size \( s = s_1 s_2 \), can be approximated by \( x^j \approx Da^i \) with a sparse vector \( a^i \) in \( \mathbb{R}^n \). Here, \( n \) is chosen a priori and the number of patches depends on whether the patches are overlapping or not. In general, \( D \) is chosen such that \( n \geq s \). With linear measurements, the paradigm would be similar to the successful image denoising technique of [20], but the problem (3) is significantly more difficult to solve due to the modulus operator.

Before detailing our algorithm for solving (3), we introduce the following notation. Because our approach is patch-based (as most dictionary learning formulations), we consider the linear operator \( E : \mathbb{C}^{N_1 \times N_2} \to \mathbb{C}^{s \times p} \) that extracts the \( p \) patches \( x^i \) (which may overlap or not) from an image \( X \) and forms the matrix \( E(X) = (x^1, \ldots, x^p) \). Similarly, we define the linear operator \( R : \mathbb{C}^{s \times p} \to \mathbb{C}^{N_1 \times N_2} \) that reverses this process, i.e., builds an image from a matrix containing vectorized patches as its columns. Thus, in particular, we have \( R(E(X)) = X \). When the patches do not overlap, the operator \( R \) simply places every small patch at its appropriate location in a larger \( N_1 \times N_2 \) image. When they overlap, the operator \( R \) averages the pixel values from the patches that fall into the same location. Further, let \( A := (a^1, \ldots, a^p) \in \mathbb{R}^{n \times p} \) be the matrix containing the patch representation coefficient vectors as columns. Then, our desired sparse-approximation relation “\( x^i \approx Da^i \) for all \( i \)” can be expressed as \( E(X) \approx DA \).

With this notation in hand, we may now introduce our method, called DOLPHIn (DictOrnary Learning for PHase retrieval). We consider an optimization problem which can be interpreted as a combination of an optimization-based approach to phase retrieval—minimizing the residual norm with respect to the set of nonlinear equations induced by the phaseless measurements, cf. (2)—and a (patch-based) dictionary learning model similar to that used for image denoising in [20]. The model contains three variables: The image, or phase retrieval solution \( X \), the dictionary \( D \) and the matrix \( A \) containing as columns the coefficient vectors of the representation \( \hat{X} \approx R(\hat{D}A) \). The phase retrieval task consists of estimating \( X \) and the dictionary learning or sparse coding task consists of estimating \( D \) and \( A \); a common objective function provides feedback between the two objectives, with the goal of improving the phase retrieval reconstruction procedure by encouraging the patches of \( X \) to admit a sparse approximation.

Formally, the DOLPHIn formulation consists of minimizing

\[
\min_{X,D,A} \frac{1}{2} \| Y - |F(X)|^2 \|_F^2 + \frac{\mu}{2} \| E(X) - DA \|_F^2 + \lambda \sum_{i=1}^p \| a^i \|_1 \tag{4}
\]

s.t. \( X \in [0, 1]^{N_1 \times N_2} \), \( D \in D \).

Here, \( \| X \|_F \) denotes the Frobenius matrix-norm, which generalizes the Euclidean norm to matrices. The parameters \( \mu, \lambda > 0 \) in the objective (4) provide a way to control the trade-off between the data fidelity term from the phase retrieval problem and the approximation sparsity of the image patches\(^1\). To that effect, we use the \( \ell_1 \)-norm, which is well-known to have a sparsity-inducing effect [30]. In order to avoid scaling ambiguities, we also restrict \( D \) to be in the subset \( D := \{ D \in \mathbb{R}^{s \times n} : \| d^j \|_2 \leq 1 \ \forall j = 1, \ldots, n \} \) of matrices with column \( \ell_2 \)-norms at most 1, and assume \( n < p \) (otherwise, each patch is trivially representable by a 1-sparse vector \( a^i \) by including \( x^j/\| x^j \|_2 \) as a column of \( D \)).

The model (4) could also easily be modified to include further side constraints, a different type of nonlinear measurements, or multiple images or measurements, respectively; we omit these extensions for simplicity.

III. ALGORITHMIC FRAMEWORK

Similar to classical dictionary learning [19], [22], [21], [31] and phase retrieval, problem (4) is nonconvex and difficult to solve. Therefore, we adopt an algorithm that provides monotonic decrease of the objective while converging to a stationary point (see Section III-D below).

The algorithmic framework we employ is that of alternating minimization: For each variable \( A, X \) and \( D \) in turn, we take one step towards solving (4) with respect to this variable alone, keeping the other ones fixed. Each of these subproblems is convex in the remaining unfixed optimization variable, and well-known efficient algorithms can be employed accordingly. We summarize our method in Algorithm 1, where the superscript \(^*\) denotes the adjoint operator (for a matrix \( Z \), \( Z^* \) is thus the conjugate transpose), \( R(\cdot) \) extracts the real part of a complex-valued argument, and \( \odot \) denotes the Hadamard (element-wise) product of two matrices. The algorithm also involves the classical soft-thresholding operator \( S_\tau(Z) := \max\{0, |Z| - \tau\} \odot \text{sign}(Z) \) and the Euclidean projection \( \mathcal{P}_X(Z) := \max\{0, \min\{1, |Z|\}\} \) onto \( X := [0, 1]^{N_1 \times N_2} \); here, all operations are meant component-wise.

\(^1\)We discuss suitable choices and sensitivity of the model to these parameters in detail in Section IV-D.
To avoid training the dictionary on potentially useless early estimates, the algorithm performs two phases—while the iteration counter \( \ell \) is smaller than \( K_1 \), the dictionary is not updated. Below, we explain the algorithmic steps in more detail.

Note that DOLPHIn actually produces two distinct reconstructions of the desired image, namely \( X \) (the per se “image variable”) and \( RF(DA) \) (the image assembled from the sparsely coded patches)\(^2\). Our numerical experiments in Section IV show that in many cases, \( RF(DA) \) is in fact slightly or even significantly better than \( X \) with respect to at least one quantitative quality measure and is therefore also considered a possible reconstruction output of Algorithm I (at least in the noisy setups we consider in this paper). Nevertheless, \( X \) is sometimes more visually appealing and can be used, for instance, to refine parameter settings (if it differs strongly from \( RF(DA) \)) or to assess the influence of the patch-based “regularization” on the pure non-sparse Wirtinger Flow method corresponding to the formulation where \( \lambda \) and \( \mu \) are set to zero.

### A. Updating the Patch Representation Vectors

Updating \( A \) (i.e., considering (4) with \( D \) and \( X \) fixed at their current values) consists of decreasing the objective

\[
\sum_{i=1}^{p} \left( \frac{1}{2} \| D_{(\ell)} a^i - x^i_{(\ell)} \|_2^2 + \frac{\lambda}{2\mu} \| a^i \|_1^1 \right),
\]

(5)

which is separable in the patches \( i = 1, \ldots, p \). Therefore, we can update all vectors \( a^i \) independently and/or in parallel. To do so, we choose to perform one step of the well-known algorithm ISTA (see, e.g., [24]), which is a gradient-based method that is able to take into account a non-smooth regularizer such as the \(\ell_1\)-norm. Concretely, the update formula for each \( i = 1, \ldots, p \):

\[
a^i_{(\ell+1)} = S_{\gamma^A\lambda/\mu} \left( a^i_{(\ell)} - \gamma^A \left( D_{(\ell)}^\top D_{(\ell)} a^i_{(\ell)} - x^i_{(\ell)} \right) \right).
\]

(6)

This update involves a gradient descent step (the gradient with respect to \( a^i \) of the smooth term in each summand of (5) is \( D_{(\ell)}^\top (D_{(\ell)} a^i_{(\ell)} - x^i_{(\ell)}) \)), respectively) followed by soft-thresholding. Constructing \( A_{(\ell+1)} \) from the \( a^i_{(\ell+1)} \) as specified above is equivalent to Step 2 of Algorithm 1.

The step size parameter \( \gamma^A \) can be chosen in \((0, 1/L_A)\), where \( L_A \) is an upper bound on the Lipschitz constant of the gradient; here, \( L_A = \| D_{(\ell)}^\top D_{(\ell)} \|_2 = \| D_{(\ell)} \|_2^2 \) would be appropriate, but a less computationally demanding strategy is to use a backtracking scheme to automatically update \( L_A \) [24].

A technical subtlety is noteworthy in this regard: We can either find one \( \gamma^A \) that works for the whole matrix-variable update problem—this is what is stated implicitly in Step 2—or we could find different values, say \( \gamma_{\ell A} \), for each column \( a^i \), \( i = 1, \ldots, p \), of \( A \) separately. Our implementation does the latter, since it employs a backtracking strategy for each column update independently.

### B. Updating the Image Estimate

With \( D = D_{(\ell)} \) and \( A = A_{(\ell+1)} \) fixed, updating \( X \) consists of decreasing the objective

\[
\frac{1}{4} \| Y - |F(X)|^2 \|_F^2 + \frac{\mu}{2} \| E(X) - DA \|_F^2
\]

(7)

with \( X \in X = [0, 1]^{N_1 \times N_2} \).

This problem can be seen as a regularized version of the phase retrieval problem (with regularization parameter \( \mu \)) that encourages the patches of \( X \) to be close to the sparse approximation \( DA \) obtained during the previous (inner) iterations.

Our approach to decrease the value of the objective (7) is by a projected gradient descent step. In fact, for \( \mu = 0 \), this step reduces to the Wirtinger flow method [12], but with necessary modifications to take into account the constraints on \( X \) (real-valuedness and variable bounds \([0, 1]\)).

The gradient of \( \varphi(X) := \frac{1}{4} \| Y - |F(X)|^2 \|_F^2 \) with respect to \( X \) can be computed as

\[
\nabla \varphi(X) = \Re \left( F^* (F(X) \odot (|F(X)|^2 - Y)) \right),
\]

by using the chain rule. For \( \psi(X) := \frac{\mu}{2} \| E(X) - DA \|_F^2 \), the gradient is given by

\[
\nabla \psi(X) = \mu E^*(E(X) - DA) = \mu R \odot R (E(X) - DA),
\]

where \( R \) is an \( N_1 \times N_2 \) matrix whose entries \( r_{ij} \) equal the number of patches the respective pixel \( x_{ij} \) is contained in. Note that if the whole image is divided into a complete set of nonoverlapping patches, \( R \) will just be the all-ones matrix; otherwise, the element-wise multiplication with \( R \) undoes the averaging of pixel values performed by \( R \) when assembling an image from overlapping patches.

Finally, the gradient w.r.t. \( X \) of the objective in (7) is

\[
\nabla \varphi(X) + \nabla \psi(X) \in X = \Re [0, 1]^{N_1 \times N_2},
\]

and the update in Step 3 of Algorithm 1 is indeed shown to be a projected gradient descent step. Typically, a backtracking (line search) strategy is used for choosing the step size \( \gamma^X \); see Theorem 2 in Section III-D for a selection rule that gives theoretical convergence, and also Section IV-B for a heuristic alternative.

### C. Updating the Dictionary

To update the dictionary, i.e., to approximately solve (4) w.r.t. \( D \) alone, keeping \( X \) and \( A \) fixed at their current values, we employ one pass of a block-coordinate descent (BCD) algorithm on the columns of the dictionary [23]. The objective to decrease may be written as

\[
\frac{1}{2} \sum_{i=1}^{p} \| D a^i_{(\ell+1)} - x^i_{(\ell+1)} \|_2^2 \text{ s.t. } D \in D,
\]

(8)

and the update rule given by Steps 4–13 corresponds\(^3\) to one iteration of [21, Algorithm 11] applied to (8).

\[^3\]In [21, Algo. 11], and in our implementation, we simply normalize the columns of \( D \); it is easily seen that any solution with \( \| D^j \|_2 < 1 \) for some \( j \) is suboptimal (w.r.t. (4)) since raising it to \( t \) allows to reduce coefficients in \( A \) and thus to improve the \( \ell_1 \)-term of the DOLPHIn objective (4). However, using the projection is more convenient for proving the convergence results without adding more technical subtleties w.r.t. this aspect.
Algorithm 1 Dictionary learning for phase retrieval (DOLPHIn)

Input: Initial image estimate \( X_{(0)} \in [0,1]^{N_1 \times N_2} \), initial dictionary \( D_{(0)} \in \mathbb{D} \subset \mathbb{R}^{s \times n} \), parameters \( \mu, \lambda > 0 \), maximum number of iterations \( K_1, K_2 \)

Output: Learned dictionary \( D = D_{(K)} \), patch representations \( A = A_{(K)} \), image reconstructions \( X = X_{(K)} \) and \( \mathcal{R}(DA) \)

1: for \( \ell = 0, 1, 2, \ldots, K_1 + K_2 := K \) do
2: choose step size \( \gamma^A_\ell \) as explained in Section III-A and update
   \[ A_{(\ell+1)} \leftarrow S_{\gamma^A_\ell \lambda / \mu} \left( A_\ell - \gamma^A_\ell D^T_\ell \left( D_\ell A_\ell - \mathcal{E}(X_\ell) \right) \right) \]
3: choose step size \( \gamma^X_\ell \) as explained in Section III-D or IV-B and update
   \[ X_{(\ell+1)} \leftarrow \mathcal{P}_X \left( X_\ell - \gamma^X_\ell \left( \mathcal{R} \left( \mathcal{F}^*(\mathcal{F}(X) \odot (|\mathcal{F}(X)|^2 - Y)) + \mu \mathcal{R} (\mathcal{E}(X) - DA) \right) \right) \],
   where \( \mathcal{R} \) is defined in Section III-B
4: if \( \ell < K_1 \) then
5: do not update the dictionary: \( D_{(\ell+1)} \leftarrow D_\ell \)
6: else
7: set \( B \leftarrow \mathcal{E}(X_\ell) A^T_\ell \) and \( C \leftarrow A_\ell A^T_\ell \)
8: for \( j = 1, \ldots, n \) do
9: if \( C_{jj} > 0 \) then
10: update \( j \)-th column: \( (D_{(\ell+1)})_j \leftarrow \frac{1}{C_{jj}} (B_j - D_\ell C_j) + (D_\ell)_j \)
11: else
12: reset \( j \)-th column: e.g., \( (D_{(\ell+1)})_j \leftarrow \text{random } \mathcal{N}(0,1) \) vector in \( \mathbb{R}^s \)
13: project \( (D_{(\ell+1)})_j \) to \( \max \{1, \| (D_{(\ell+1)})_j \|_2 \} (D_{(\ell+1)})_j \)

To see this, note that each column update problem has a closed-form solution:

\[
(d^j)_{(\ell+1)} = \mathcal{P}_{\|x\|_2 \leq 1} \left( \frac{1}{\sum_{i=1}^p (a^j_i)^2} \sum_{i=1}^p a^j_i (x^i - \sum_{k=1, k \neq j}^n a^k_i d^k) \right) \\
= \frac{1}{\max \{1, \| w^j \|_2 \}} \frac{1}{\max \{1, \| q^j \|_2 \}} \cdot \frac{1}{\max \{1, \| w^j \|_2 \}} \frac{1}{\max \{1, \| q^j \|_2 \}}
\]

with \( w_j := \sum_i (a^j_i)^2 \) and \( q^j := \sum_i a^j_i (x^i - \sum_{k=1, k \neq j}^n a^k_i d^k) \); here, we abbreviated \( a^j := a^j_{(\ell+1)} \), \( x^i := x^i_{(\ell+1)} \). If \( w_j = 0 \), and thus \( a^j_i = 0 \) for all \( i = 1, \ldots, p \), then column \( d^j \) is not used in any of the current patch representations; in that case, the column’s update problem has a constant objective and is therefore solved by any \( d \) with \( \|d\|_2 \leq 1 \), e.g., a normalized random vector as in Step 12 of Algorithm 1. The computations performed in Steps 8–13 of Algorithm 1 are equivalent to these solutions, expressed differently using the matrices \( B \) and \( C \) defined there. Note that the operations could be parallelized to speed up computation.

D. Convergence of the Algorithm

As mentioned at the beginning of this section, with appropriate step size choices, DOLPHIn (Algorithm 1) exhibits the property of monotonically decreasing the objective function value (4) at each iteration. In particular, many line-search type step size selection mechanisms aim precisely at reducing the objective; for simplicity, we will simply refer to such subroutines as “suitable backtracking schemes” below. Concrete examples are the ISTA backtracking from [24, Section 3] we can employ in the update of \( A \), or the rule given in Theorem 2 for the projected gradient descent update of \( X \) (a different choice is described in Section IV-B); further variants are discussed, e.g., in [32].

Proposition 1: Let \( (A_\ell, X_\ell, D_\ell) \) be the current iterates (after the \( \ell \)-th iteration) of Algorithm 1 with step sizes \( \gamma^X_\ell \) and \( \gamma^A_\ell \) determined by suitable backtracking schemes (or arbitrary \( 0 < \gamma^X_\ell < 1 / \|D_\ell^T D_\ell\|_2 \), resp.) and let \( f_{\ell,i,j;k} \) denote the objective function value of the DOLPHIn model (4) at \( (A_\ell, X_\ell, D_\ell) \). Then, DOLPHIn either terminates in the \( (\ell + 1) \)-th iteration, or it holds that \( f_{\ell+1,\ell+1,\ell+1} \leq f_{\ell,\ell,\ell} \).

Proof: Since we use ISTA to update \( A \), it follows from [24] that \( f_{\ell+1,\ell+1,\ell} \leq f_{\ell,\ell,\ell} \). Similarly, a suitable backtracking strategy is known to enforce descent in the projected gradient method when the gradient is locally Lipschitz-continuous, whence \( f_{\ell,\ell,\ell} \leq f_{\ell+1,\ell+1,\ell} \leq f_{\ell+1,\ell+1,\ell} \). Finally, \( f_{\ell+1,\ell+1,\ell+1} \leq f_{\ell+1,\ell+1,\ell} \) follows from standard results for BCD methods applied to convex problems, see, e.g., [33]. Combining these inequalities proves the claim.

The case of termination in Proposition 1 can occur when the backtracking scheme is combined with a maximal number of trial steps, which are often used as a safeguard against numerical stability problems or as a heuristic stopping condition to terminate the algorithm if no (sufficient) improvement can be reached even with tiny step sizes. Note also that the assertion of Proposition 1 trivially holds true if all step sizes are 0; naturally, a true descent of the objective requires a strictly positive step size in at least one update step. In our algorithm, step size positivity can always be guaranteed since all these updates involve objective functions whose gradient is Lipschitz continuous, and backtracking essentially finds step sizes inversely proportional to (local) Lipschitz constants. (Due
to non-expansiveness, the projection in the X-update poses no problem either).

Adopting a specific Armijo-type step size selection rule for the X-update allows us to infer a convergence result, stated in Theorem 2 below. To simplify the presentation, let \( f_\ell^X(X) \) be the Alg 1 with the following Armijo rule to be used in Step 3:

\[
\Gamma_\ell := \nabla f_\ell^X(X_{(\ell)}) \quad \text{and} \quad \Gamma_\ell^X := \nabla f_\ell^X(X_{(\ell)}) - \nabla f_\ell^X(X_{(\ell)}) \leq -\frac{1}{\gamma} \|
abla f_\ell^X(X_{(\ell)}) - \nabla f_\ell^X(X_{(\ell)})\|^2_2 \quad \text{and set} \quad X_{(\ell+1)} := \mathcal{P}_X(X_{(\ell)} - \gamma_\ell^X \Gamma_\ell^X).
\]

If \( \mu \geq 1 \) and, for some \( 0 < \gamma_{\ell+1} \leq \gamma_{\ell} \), it holds that \( \gamma_{\ell+1} \leq \gamma_{\ell+1} \), \( \gamma_{\ell+1} \), \( \gamma_{\ell+1} \), \( \gamma_{\ell+1} \) and \( \gamma_{\ell+1} \), \( \gamma_{\ell+1} \) and \( \gamma_{\ell+1} \), \( \gamma_{\ell+1} \) all satisfy these conditions. More specifically, the directions are generated using a (suitably parameterized) strictly convex quadratic approximation of the objective (built using the gradient). Essentially due to the strict convexity, it is then always possible to make a positive step along such a direction that decreases the objective, unless stationarity already holds. Using a certain Armijo line-search rule designed to find such a step size, the directional descent property is obtained, which still ensures monotonic objective descent (and hence is “suitable” in the context of Proposition 1), also enables strictly positive steps, and empirically performs equally well. Finally, we remark that the condition \( \mu \geq 1 \) in Theorem 2 can be dropped if the relevant objective parts of problem (4) are not rescaled for the A- and D-updates, respectively.

To conclude the discussion of convergence, we point out that one can obtain a linear rate of convergence for DOLPHIn with the Armijo rule from Theorem 2 by extending the results of [34, Theorem 2 (cf. Section 8)].

Our implementation uses a different backtracking scheme for the X-update (see Section IV-B) that can be viewed as a cheaper heuristic alternative to the stated Armijo-rule which still ensures monotonic objective descent (and hence is “suitable” in the context of Proposition 1), also enables strictly positive steps, and empirically performs equally well. Finally, we discuss sensitivity of DOLPHIn with regard to various (hyper-)parameter choices (Subsections IV-D, IV-E and IV-F).

A. Experimental Setup

We consider several linear operators \( \mathcal{F} \) corresponding to different types of measurements that are classical in the phase
operators and introduce two complex Gaussian matrices $G \in \mathbb{C}^{M_1 \times N_1}$, $H \in \mathbb{C}^{M_2 \times N_2}$, whose entries are i.i.d. samples from the distribution $\mathcal{N}(0, 1/2) + i \mathcal{N}(0, 1/2)$. Then, we experiment with the operators $F(X) = GX$, $F(X) = G\mathbf{X}^*$, $F(X) = GXH^*$, and the coded diffraction pattern model
\[
F(X) = \left( \begin{array}{c} F(M_1 \odot X) \\ \vdots \\ F(M_m \odot X) \end{array} \right), \quad F^*(Z) = \sum_{j=1}^{m} (M_j \odot F^*(Z_j)),
\]
where $Z_j := Z_{\{j-1\}} N_1 + 1, \ldots, jN_1 \}$, i.e., $Z^T = (Z_1^T, \ldots, Z_m^T)$ and the $M_j$’s are admissible coded diffraction patterns (CDPs), see for instance [12, Section 4.1]; in our experiments we used ternary CDPs, such that each $M_j$ is in $\{0, \pm 1\}^{N_1 \times N_2}$. (Later, we will also consider octanary CDPs with $M_j \in \{\pm \sqrt{1/2}, \pm i \sqrt{1/2}, \pm \sqrt{3/3}, \pm i \sqrt{3/3}\} \subset \mathbb{C}^{N_1 \times N_2}$.)

To reconstruct $X$, we choose an oversampling setting where $M_1 = 4N_1$, $M_2 = 4N_2$ and/or $m = 2$, respectively. Moreover, we corrupt our measurements with additive white Gaussian noise $N$ such that $\text{SNR}(Y, |F(X)|^2 + N) = 10 \text{dB}$ for the Gaussian-type, and $20 \text{dB}$ for CDP measurements, respectively. Note that these settings yield, in particular, a relatively heavy noise level for the Gaussian cases and a relatively low oversampling ratio for the CDPs.

B. Implementation Details

We choose to initialize our algorithm with a simple random image $X(0)$ in $\mathcal{X}$ to demonstrate the robustness of our approach with respect to its initialization. Nevertheless, other choices are possible. For instance, one may also initialize $X(0)$ with a power-method scheme similar to that proposed in [12], modified to account for the real-valuedness and box-constraints. The dictionary is initialized as $D(0) = (I, F_D)$ in $\mathbb{R}^{s \times 2s}$, where $F_D$ corresponds to the two-dimensional discrete cosine transform (see, e.g., [20]).

To update $A$, we use the ISTA implementation from the SPAMS package\footnote{http://spams-devel.gforge.inria.fr/} [23] with its integrated backtracking line search (for $L_A$). Regarding the step sizes $\gamma^X$, for the update of $X$ (Step 3 of Algorithm 1), we adopt the following simple strategy, which is similar to that from [24] and may be viewed as a heuristic to the Armijo rule from Theorem 2: Whenever the gradient step leads to a reduction in the objective function value, we accept it. Otherwise, we recompute the step with $\gamma^X$ halved until a reduction is achieved; here, as a safeguard against numerical issues, we implemented a limit of 100 trials (forcing termination in case all were unsuccessful), but this was never reached in any of our computational experiments. Regardless of whether $\gamma^X$ was reduced or not, we reset its value to $1.68\gamma^X$ for the next round; the initial step size is $\gamma_0^X = 10^{2}/f(0)$, where $f(0)$ is the objective function of the DOLPHIn model (4), evaluated at $X(0), D(0)$ and least-squares patch representations $\arg\min_A \|E(X(0)) - D(0)A\|_F^2$. (Note that, while this rule deviates from the theoretical convergence of Theorem 2, Proposition 1 and the remarks following it remain applicable.)

Finally, we consider nonoverlapping $8 \times 8$ patches and run DOLPHIn (Algorithm 1) with $K_1 = 25$ and $K_2 = 50$; the regularization/penalty parameter values can be read from Table I (there, $m_Y$ is the number of elements of $Y$). We remark that these parameter values were empirically benchmarked to work well for the measurement setups and instances considered here; a discussion about the stability of our approach with respect to these parameter choices is presented below in Section IV-D. Further experiments with a sparsity-constrained DOLPHIn variant and using overlapping patches are discussed in Section IV-G.

Our DOLPHIn code is available online on the first author’s webpage\footnote{http://www.mathematik.tu-darmstadt.de/~tillmann/}.

C. Computational Experiments

We test our method on a collection of typical (grayscale) test images used in the literature, see Figure 1. All experiments were carried out on a Linux 64-bit quad-core machine (2.8 GHz, 8 GB RAM) running Matlab R2016a (single-thread).

We evaluate our approach with the following question in mind: Can we improve upon the quality of reconstruction compared to standard phase retrieval algorithms? Standard methods cannot exploit sparsity if the underlying basis or dictionary is unknown; as we will see, the introduced (patch-) sparsity indeed allows for better recovery results (at least in the oversampling and noise regimes considered here).

To evaluate the achievable sparsity, we look at the average number of nonzeros in the columns of $A$ after running our algorithm. Generally, smaller values indicate an improved suitability of the learned dictionary for sparse patch coding (high values often occur if the regularization parameter $\lambda$ is too small and the dictionary is learning the noise, which is something we would like to avoid). To assess the quality of the image reconstructions, we consider two standard measures, namely the peak signal-to-noise ratio (PSNR) of a reconstruction as well as its structural similarity index (SSIM) [36]. For PSNR, larger values are better, and SSIM-values closer to 1 (always ranging between 0 and 1) indicate better visual quality.

Table I displays the CPU times, PSNR- and SSIM-values and mean patch representation vector sparsity levels obtained for the various measurement types, averaged over the instance groups of the same size. The concrete examples in Figures 2 and 3 show the results from DOLPHIn and plain Wirtinger Flow (WF; the real-valued, [0, 1]-box constrained variant, which corresponds to running Algorithm 1 with $\mu = 0$ and omitting the updates of $A$ and $D$). In all tests, we let the Wirtinger Flow method run for the same number of iterations (75) and use the same starting points as for the DOLPHIn runs. Note that instead of random $X(0)$, we could also use a spectral initialization similar to the one proposed for the (unconstrained) Wirtinger Flow algorithm, see [12]. Such initialization can improve WF reconstruction (at least in the noiseless case), and may also provide better initial estimates for DOLPHIn. We have experimented with such a...
IEEE TRANSACTIONS ON SIGNAL PROCESSING

Fig. 1. Test images. (a)–(c): cameraman, house and peppers (size 256 × 256). (d)–(h): lena, barbara, boat, fingerprint and mandrill (size 512 × 512).

TABLE I

Test results for Gaussian-type and coded diffraction pattern (CDP) measurements. We report mean values (geometric mean for CPU times) per measurement type, obtained from three instances with random \( X_{(0)} \) and random noise for each of the three 256 × 256 and five 512 × 512 images, w.r.t. the reconstructions from DOLPHIn (\( X_{\text{DOLPHIn}} \) and \( \mathcal{P}_X(R(DA)) \)) with parameters \((\mu, \lambda)\) and (real-valued, [0, 1]-constrained) Wirtinger Flow (\( X_{WF} \)), respectively. (CPU times in seconds, PSNR in decibels).

<table>
<thead>
<tr>
<th>( F ) type</th>
<th>reconstruction</th>
<th>( (\mu, \lambda)/mV )</th>
<th>time</th>
<th>PSNR</th>
<th>SSIM</th>
<th>( |a^t|_0 )</th>
<th>( (\mu, \lambda)/mV )</th>
<th>time</th>
<th>PSNR</th>
<th>SSIM</th>
<th>( |a^t|_0 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( G^X )</td>
<td>( X_{\text{DOLPHIn}} ) ( \mathcal{P}_X(R(DA)) )</td>
<td>(0.5,0.105)</td>
<td>12.69</td>
<td>24.69</td>
<td>0.5747</td>
<td>3.77</td>
<td>(0.5,0.105)</td>
<td>68.62</td>
<td>24.42</td>
<td>0.6547</td>
<td>6.30</td>
</tr>
<tr>
<td></td>
<td>( X_{WF} )</td>
<td>7.49</td>
<td>19.00</td>
<td>0.2898</td>
<td>–</td>
<td>–</td>
<td>49.23</td>
<td>18.83</td>
<td>0.3777</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>( G^XG^* )</td>
<td>( X_{\text{DOLPHIn}} ) ( \mathcal{P}_X(R(DA)) )</td>
<td>(0.5,0.210)</td>
<td>51.78</td>
<td>22.67</td>
<td>0.4135</td>
<td>7.45</td>
<td>(0.5,0.210)</td>
<td>357.49</td>
<td>22.59</td>
<td>0.5296</td>
<td>11.37</td>
</tr>
<tr>
<td></td>
<td>( X_{WF} )</td>
<td>47.76</td>
<td>22.66</td>
<td>0.4131</td>
<td>–</td>
<td>–</td>
<td>349.28</td>
<td>22.58</td>
<td>0.5290</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>( G^XH^* )</td>
<td>( X_{\text{DOLPHIn}} ) ( \mathcal{P}_X(R(DA)) )</td>
<td>(0.5,0.210)</td>
<td>52.18</td>
<td>22.67</td>
<td>0.4132</td>
<td>7.50</td>
<td>(0.5,0.210)</td>
<td>357.66</td>
<td>22.57</td>
<td>0.5286</td>
<td>11.38</td>
</tr>
<tr>
<td></td>
<td>( X_{WF} )</td>
<td>48.24</td>
<td>22.65</td>
<td>0.4127</td>
<td>–</td>
<td>–</td>
<td>348.54</td>
<td>22.55</td>
<td>0.5282</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>CDP (cf. (9))</td>
<td>( X_{\text{DOLPHIn}} ) ( \mathcal{P}_X(R(DA)) )</td>
<td>(0.05,0.003)</td>
<td>8.56</td>
<td>27.15</td>
<td>0.7416</td>
<td>7.85</td>
<td>(0.05,0.003)</td>
<td>36.72</td>
<td>27.33</td>
<td>0.7819</td>
<td>11.48</td>
</tr>
<tr>
<td></td>
<td>( X_{WF} )</td>
<td>2.83</td>
<td>13.10</td>
<td>0.1170</td>
<td>–</td>
<td>–</td>
<td>14.79</td>
<td>12.70</td>
<td>0.1447</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

spectral approach and found the results comparable to what is achievable with random \( X_{(0)} \), both for WF and DOLPHIn. Therefore, we do not report these experiments in the paper.

The DOLPHIn method consistently provides better image reconstructions than WF, which clearly shows that our approach successfully introduces sparsity into the phase retrieval problem and exploits it for estimating the solution. As can be seen from Table I, the obtained dictionaries allow for rather sparse representation vectors, with the effect of making better use of the information provided by the measurements, and also denoising the image along the way. The latter fact can be seen in the examples (Fig. 2 and 3, see also Fig. 4) and also inferred from the significantly higher PSNR and SSIM values for the estimates \( X_{\text{DOLPHIn}} \) and \( R(DA) \) (or \( \mathcal{P}_X(R(DA)) \), resp.) obtained from DOLPHIn compared to the reconstruction \( X_{WF} \) of the WF algorithm (which cannot make use of hidden sparsity). The gain in reconstruction quality is more visible in the example of Fig. 3 (cf. Fig. 4) than for that in Fig. 2, though both cases assert higher quantitative measures. Furthermore, note that DOLPHIn naturally has higher running times than WF, since it performs more work per iteration (also, different types of measurement operators require different amounts of time to evaluate). Note also that storing \( A \) and \( D \) instead of an actual image \( X \) (such as the WF reconstruction) requires saving only about half as many numbers (including integer index pairs for the nonzero entries in \( A \)).
As indicated earlier, the reconstruction $\mathcal{R}(\mathbf{DA})$ is quite often better than $\mathbf{X}_{\text{DOLPHIn}}$ w.r.t. at least one of either PSNR or SSIM value. Nonetheless, $\mathbf{X}_{\text{DOLPHIn}}$ may be visually more appealing than $\mathcal{R}(\mathbf{DA})$ even if the latter exhibits a higher quantitative quality measure (as is the case, for instance, in the example of Figures 3 and 4); Furthermore, occasionally $\mathbf{X}_{\text{DOLPHIn}}$ achieves notably better (quantitative) measures than $\mathcal{R}(\mathbf{DA})$; an intuitive explanation may be that if, while the sparse coding of patches served well to eliminate the noise and—by means of the patch-fit objective term—to successfully “push” the $\mathbf{X}$-update steps toward a solution of good quality, that solution eventually becomes “so good”, then the fact that $\mathcal{R}(\mathbf{DA})$ is designed to be only an approximation (of $\mathbf{X}$) predominates.

On the other hand, $\mathbf{X}_{\text{DOLPHIn}}$ is sometimes very close to $\mathbf{X}_{\text{WF}}$, which indicates a suboptimal setting of the parameters $\mu$ and $\lambda$ that control how much “feedback” the patch-fitting objective term introduces into the Wirtinger-Flow-like $\mathbf{X}$-update in the DOLPHIn algorithm. We discuss parameter choices in more detail in the following subsection.

D. Hyperparameter Choices and Sensitivity

The DOLPHIn algorithm requires several parameters to be specified a priori. Most can be referred to as design
parameters; the most prominent ones are the size of image patches \((s_1 \times s_2)\), whether patches should overlap or not (not given a name here), and the number \(n\) of dictionary atoms to learn. Furthermore, there are certain algorithmic parameters (in a broad sense) that need to be fixed, e.g., the iteration limits \(K_1\) and \(K_2\) or the initial dictionary \(\mathbf{D}_0\) and image estimate \(\hat{\mathbf{x}}_0\). The arguably most important parameters, however, are the model or regularization parameters \(\mu\) and \(\lambda\). For any fixed combination of design and algorithmic parameters in a certain measurement setup (fixed measurement type/model and (assumed) noise level), it is conceivable that one can find some values for \(\mu\) and \(\lambda\) that work well for most instances, while the converse—choosing, say, iteration limits for fixed \(\mu\), \(\lambda\) and other parameters—is clearly not a very practical approach.

As is common for regularization parameters, a good “general-purpose” way to choose \(\mu\) and \(\lambda\) a priori is unfortunately not known. To obtain the specific choices used in our experiments, we fixed all the other parameters (including noise SNR and oversampling ratios), then (for each measurement model) ran preliminary tests to identify values \(\mu\) for which good results could be produced with some \(\lambda\), and finally fixed \(\mu\) at such values and ran extensive benchmark tests to find \(\lambda\) values that give the best results.

For DOLPHin, \(\mu\) offers some control over how much “feedback” from the current sparse approximation of the current image estimate is taken into account in the update step to produce the next image iterate—overly large values hinder the progress made by the Wirtinger-Flow-like part of the \(X\)-update, while too small values marginalize the influence of the approximation \(R(DA)\), with one consequence being that the automatic denoising feature is essentially lost. Nevertheless, in our experiments we found that DOLPHin is not strongly sensitive to the specific choice of \(\mu\) once a certain regime has been identified in which one is able to produce meaningful results (for some choice of \(\lambda\)). Hence, \(\lambda\) may be considered the most important parameter; note that this intuitively makes sense, as \(\lambda\) controls how strongly sparsity of the patch representations is actually promoted, the exploitation of which to obtain improved reconstructions being the driving purpose behind our development of DOLPHin.

Figure 5 illustrates the sensitivity of DOLPHin with respect to \(\lambda\), in terms of reconstruction quality and achievable patch-sparsity, for different noise levels, and exemplary measurement types and problem sizes. (In this figure, image quality is...
measured by SSIM values alone; the plots using PSNR instead look very similar and were thus omitted. Note, however, that the parameters $\lambda$ yielding the best SSIM and PSNR values, respectively, need not be the same.) As shown by (a) and (d), there is a clear correlation between the best reconstruction quality that can be achieved (in noisy settings) and the average sparsity of the patch representation vectors $a^i$. For larger noise, clearly a larger $\lambda$ is needed to achieve good results—see (b) and (e)—which shows that a stronger promotion of patch-sparsity is an adequate response to increased noise, as is known for linear sparsity-based denoising as well. Similarly, increasing the number of measurements allows to pick a smaller $\lambda$ whatever the noise level actually is, as can be seen in (c) and (f), respectively. The dependence of the best $\lambda$ on the noise level appears to follow an exponential curve (w.r.t. the reciprocal SNR) which is “dampened” by the sampling ratio, i.e., becoming less steep and pronounced the more measurements are available, cf. (b) and (e). Indeed, again referring to the subplots (c) and (f), at a fixed noise level the best $\lambda$ values seem to decrease exponentially with growing number of measurements. It remains subject of future research to investigate these dependencies in more detail, e.g., to come up with more or less general (functional) rules for choosing $\lambda$.

E. Impact of Increased Inner Iteration Counts

It is worth considering whether more inner iterations—i.e., consecutive update steps for the different variable blocks—lead to further improvements of the results and/or faster convergence. In general, this is an open question for block-coordinate descent algorithms, so the choices are typically made empirically. Our default choices of $a = 1$ ISTA iterations for the $A$-update (Step 2 in Algorithm 1), $x = 1$ projected gradient descent steps for the $X$-update (Step 3) and $d = 1$ iterations of the BCD scheme for the $D$-update (Steps 4–13) primarily reflect the desire to keep the overall iteration cost low. To assess whether another choice might yield significant improvements, we evaluated the DOLPHIn performance for all combinations of $a \in \{1, 3, 5\}$ and $d \in \{1, 3, 5\}$, keeping all other parameters equal to the settings from the experiments reported on above. (We also tried these combinations together with an increased iteration count for the $X$-update, but already for $x = 2$ or $x = 3$ the results were far worse than with just 1 projected gradient descent step; the reason can likely be found in the fact that without adapting $A$ and $D$ to a modified $X$-iterate, the patch-fitting term of the objective tries to keep $X$ close to a then-unsuitable estimate $R(DA)$ based on older $X$-iterates, which apparently has a quite notable negative influence on the achievable progress in the $X$-update loop.)

The results are summarized in condensed format in Table II, from which we can read off the spread of the best and worst results (among the best ones achievable with either $X$ or $P_X(R(DA)))$ for each measurement-instance combination among all combinations $(a, d) \in \{1, 3, 5\}^2$. (Full tables for each test run can be found alongside our DOLPHIn code on the first author’s webpage.) As the table shows, the results are all quite close; while some settings lead to sparser patch representations, the overall quality of the best reconstructions for the various combinations usually differ only slightly, and no particular combination stands out clearly as being better than all others. In particular, comparing the results with those in Table I, we find that our default settings provide consistently good results; they may be improved upon with some other combination of $(a, d)$, but at least with the same total iteration horizon ($K_1 + K_2 = 75$), the improvements are often only marginal. Since the overall runtime reductions (if any) obtained with other choices for $(a, d)$ are also very small, there does not seem to be a clear advantage to using more than a single iteration for either update problem.

F. Influence of the First DOLPHIn Phase

Our algorithm keeps the dictionary fixed at its initialization for the first $K_1$ iterations in order to prevent the dictionary from training on relatively useless first reconstruction iterates. Indeed, if all variables including $D$ are updated right from the beginning (i.e., $K_1 = 0$, $K_2 = 75$), then we end up with inferior results (keeping all other parameters unchanged): The obtained patch representations are much less sparse, the quality of the image estimate $R(DA)$ decreases drastically, and also the quality of the reconstruction $X$ becomes notably worse. This demonstrates that the dictionary apparently “learns too much noise” when updated from the beginning, and the positive effect of filtering out quite a lot of noise in the first phase by regularizing with sparsely coded patches using a fixed initial dictionary is almost completely lost. To give an example, for the CDP testset on $256 \times 256$ images, the average values obtained by DOLPHIn when updating also the dictionary from the first iteration onward are: 9.24 seconds runtime (versus 8.56 for default DOLPHIn, cf. Table I), mean patch sparsity $\mathcal{O}(|a^i|)_0 \approx 20.09$ (vs. 7.85), PSNR 26.80 dB and 8.88 dB (vs. 27.15 and 26.58) and SSIM-values 0.6931 and 0.0098 (vs. 0.7416 and 0.7654) for the reconstructions $X$ and $P_X(R(DA))$, respectively.

On the other hand, one might argue that if the first iterates are relatively worthless, the effort of updating $A$ in the first DOLPHIn phase (i.e., the first $K_1$ iterations) could be saved as well. However, the influence of the objective terms involving $D$ and $A$ should then be completely removed from the algorithm for the first $K_1$ iterations; otherwise, the patch-fitting term will certainly hinder progress made by the $X$-update because it then amounts to trying to keep $X$ close to the initial estimate $R(DA)$, which obviously needs not bear any resemblance to the sought solution at all. Thus, if both $D$ and $A$ are to be unused in the first phase, then one should temporarily set $\mu = 0$, with the consequence that the first phase reduces to pure projected gradient descent for $X$ with respect to the phase retrieval objective—i.e., essentially, Wirtinger Flow. Therefore, proceeding like this simply amounts to a different initialization of $X$. Experiments with this DOLPHIn variant ($K_1 = 25$ initial WF iterations followed by $K_2 = 50$ full DOLPHIn iterations including $A$ and $D$; all other parameters again left unchanged) showed that the achievable patch-sparsities remain about the same for the Gaussian measurement types but become much worse for
TABLE II

Test results for DOLPHIn variants with different combinations of inner iteration numbers $a$ and $d$ for the $A$- and $D$-updates, resp. Reported are the best mean values achievable (via either $X_{DOLPHIn}$ or $P_D(R(DA))$) with any of the considered combinations $(a, d) \in \{1, 3, 5\} \times \{1, 3, 5\}$ (first rows for each measurement type) and the worst among the best values for each combination (second rows), along with the respective combinations yielding the stated values. All other parameters are identical to those used in the experiments for default DOLPHIn ($a = d = 1$), cf. Table I.

$$
\begin{array}{|c|c|c|c|c|}
\hline
\text{F type} & \text{time} & \text{PSNR} & \text{SSIM} & \|a^i\|_0 \\
\hline
\text{GX} & 12.61 \,(1,3) & 24.72 \,(1,5) & 0.6680 \,(1,5) & 3.69 \,(1,5) \\
 & 15.30 \,(5,5) & 24.48 \,(3,3) & 0.6485 \,(3,3) & 4.94 \,(3,1) \\
\hline
\text{G}X^G & 51.02 \,(1,3) & 23.71 \,(1,5) & 0.7330 \,(1,5) & 6.71 \,(5,5) \\
 & 54.98 \,(3,1) & 23.57 \,(5,5) & 0.7188 \,(1,3) & 7.85 \,(1,3) \\
\hline
\text{G}X^H & 50.95 \,(1,3) & 23.68 \,(1,1) & 0.7315 \,(1,5) & 6.64 \,(5,1) \\
 & 56.05 \,(3,1) & 23.56 \,(5,3) & 0.7143 \,(5,3) & 7.95 \,(1,3) \\
\hline
\text{CDP} \,(9) & 8.56 \,(1,1) & 27.19 \,(3,1) & 0.7962 \,(3,1) & 7.71 \,(1,3) \\
 & 11.40 \,(5,5) & 27.04 \,(3,1) & 0.7654 \,(1,1) & 9.49 \,(3,1) \\
\hline
\end{array}
$$

Full tables containing results for all te Stranger reported on the CDP setup, and that the average PSNR and SSIM values become (often clearly) worse in virtually all cases. In the example of measurements $GX$ of the $512 \times 512$ test images, the above-described DOLPHIn variant runs $62.34$ seconds on average (as less effort is spent in the first phase, this is naturally lower than the $68.62$ seconds default DOLPHIn takes), produces slightly lower average patch-sparsity ($5.88$ vs. $6.30$ for default DOLPHIn), but for both $X$ and $P_D(R(DA))$, the PSNR and SSIM values are notably worse ($22.32$ dB and $20.55$ dB vs. $24.42$ dB and $22.66$ dB, and $0.6165$ and $0.5921$ vs. $0.6547$ and $0.6807$, resp.). The reason for the observed behavior can be found in the inferior performance of WF without exploiting patch-sparsity (cf. also Table I); note that the results also further demonstrate DOLPHIn’s robustness w.r.t. the initial point—apparently, the initial point obtained from some WF iterations is not more helpful for DOLPHIn than a random first guess.

Finally, it is also worth considering what happens if the dictionary updates are turned off completely, i.e., $K_2 = 0$. Then, DOLPHIn reduces to patch-sparsity regularized Wirtinger Flow, a WF variant that apparently was not considered previously. Additional experiments with $K_1 = 75$, $K_2 = 0$ and $D = D_{(0)} = (I, F_D)$ (other parameters left unchanged) showed that this variant consistently produces higher sparsity (i.e., smaller average number of nonzero entries) of the patch representation coefficient vectors, but that the best reconstruction ($X$ or $P_D(R(DA))$) is always significantly inferior to the best one produced by our default version of DOLPHIn. The first observation is explained by the fact that with $D$ fixed throughout, the patch coding ($A$-update) only needs to adapt w.r.t. new $X$-iterates but not a new $D$; at least if the new $X$ is not too different from the previous one; the former representation coefficient vectors still yield an acceptable approximation, which no longer holds true if the dictionary was also modified. While it should also be mentioned that the patch-sparsity regularized version performs much better than plain WF already, the second observation clearly indicates the additional benefit of working with trained dictionaries, i.e., superiority of DOLPHIn also over the regularized WF variant.

Full tables containing results for all test runs reported on this subsection are again available online along with our DOLPHIn code on the first author’s website.

### G. Sparsity-Constrained DOLPHIn Variant

From Table I and Figure 5, (a) and (d), it becomes apparent that a sparsity level of $8 \pm 4$ accompanies the good reconstructions by DOLPHIn. This suggests use in a DOLPHIn variant we already briefly hinted at: Instead of using $\ell_1$-norm regularization, we could incorporate explicit sparsity constraints on the $a^i$. The corresponding DOLPHIn model then reads

$$
\min_{X,D,A} \frac{1}{2} \|Y - F(X)\|_2^2 + \frac{\mu}{2} \|X - DA\|_F^2
$$

s.t. $X \in [0, 1]^{N_1 \times N_2}$, $D \in D$, $\|a^i\|_0 \leq k \forall i = 1, \ldots, p$.

(10)

where $k$ is the target sparsity level. We no longer need to tune the $\lambda$ parameter, and can let our previous experimental results guide the choice of $k$. Note that the only modification to Algorithm 1 concerns the update of $A$ (Step 2), which now requires solving or approximating

$$
\min_{X,D,A} \frac{1}{2} \|X - DA\|_F^2
$$

s.t. $\|a^i\|_0 \leq k \forall i = 1, \ldots, p$.

In our implementation, we do so by running Orthogonal Matching Pursuit (OMP) [37] for each column $a^i$ of $A$ separately until either the sparsity bound is reached or $\|x^i - D a^i\|_2 \leq \varepsilon$, where we set a default of $\varepsilon := 0.1$. (The value of $\varepsilon$ is again a parameter that might need thorough benchmarking; usually, it is related to the amount of noise one wants to filter out by sparsely representing the patches—higher noise levels will require larger $\varepsilon$ values. The $0.1$ default worked quite well in our experiments, but could probably be improved by benchmarking as well.) The OMP code we used is also part of the SPAMS package.

The effect of the parameter $\mu$ is more pronounced in the sparsity-constrained DOLPHIn than in Algorithm 1; however, it appears its choice is less dependent on the measurement model used. By just a few experimental runs, we found that good results in all our test cases can be achieved using $K_1 = K_2 = 25$ iterations, where in the first $K_1$ (with fixed
TABLE III
TEST RESULTS FOR SPARSITY-CONSTRAINED DOLPHIn, USING OVERLAPPING PATCHES, FOR mY GAUSSIAN-TYPE AND CODED DIFFRACTION PATTERN (CDP) MEASUREMENTS. REPORTED ARE MEAN VALUES (GEOMETRIC MEAN FOR CPU TIMES) PER MEASUREMENT TYPE, OBTAINED FROM THREE INSTANCES WITH RANDOM X,(0) AND RANDOM NOISE FOR EACH OF THE THREE 256 × 256 AND FIVE 512 × 512 IMAGES, W.R.T. THE RECONSTRUCTIONS FROM DOLPHIn (X,DOLPHIn AND P,x((R(DA)) WITH PARAMETERS (µ, µ2) AND (REAL-VALUED, [0, 1]-CONSTRAINED) WIRTINGER FLOW (X,WF), RESPECTIVELY. (CPU TIMES IN SECONDS, PSNR IN DECIBELS).}

<table>
<thead>
<tr>
<th>F type</th>
<th>reconstruction</th>
<th>256 × 256 instances</th>
<th>512 × 512 instances</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>(µ, µ2)/mY</td>
<td>time</td>
</tr>
<tr>
<td>GX</td>
<td>X,DOLPHIn</td>
<td>(0.005,0.0084)</td>
<td>23.58</td>
</tr>
<tr>
<td></td>
<td>P,x((R(DA))</td>
<td>27.98</td>
<td>0.7721</td>
</tr>
<tr>
<td></td>
<td>X,WF</td>
<td>5.04</td>
<td>19.00</td>
</tr>
<tr>
<td>GXG^*</td>
<td>X,DOLPHIn</td>
<td>(0.005,0.0084)</td>
<td>57.52</td>
</tr>
<tr>
<td></td>
<td>P,x((R(DA))</td>
<td>27.29</td>
<td>0.6129</td>
</tr>
<tr>
<td></td>
<td>X,WF</td>
<td>32.44</td>
<td>22.71</td>
</tr>
<tr>
<td>GXH^*</td>
<td>X,DOLPHIn</td>
<td>(0.005,0.0084)</td>
<td>57.67</td>
</tr>
<tr>
<td></td>
<td>P,x((R(DA))</td>
<td>27.14</td>
<td>0.6059</td>
</tr>
<tr>
<td></td>
<td>X,WF</td>
<td>32.32</td>
<td>22.56</td>
</tr>
<tr>
<td>CDP (cf. (9))</td>
<td>X,DOLPHIn</td>
<td>(0.005,0.0084)</td>
<td>22.10</td>
</tr>
<tr>
<td></td>
<td>P,x((R(DA))</td>
<td>26.87</td>
<td>0.7789</td>
</tr>
<tr>
<td></td>
<td>X,WF</td>
<td>2.70</td>
<td>21.95</td>
</tr>
</tbody>
</table>

dictionary), we use a value of µ = µ1 = 0.005mY along with a sparsity bound k = k1 = 4, and in the final K2 iterations (in which the dictionary is updated), µ = µ2 = 1.68µ1 = 0.0084mY and k = k2 = 8. Results on the same instances considered before (using the same D,(0)) are presented in Table III, with the exception that for the CDP case, we used 2 complex-valued octavary masks (cf. [12]) here; the initial image estimates and measurement noise were again chosen randomly. Note also that for these tests, we used complete sets of overlapping patches. This greatly increases p and hence the number of subproblems to be solved in the A-update step, which is the main reason for the increased running times compared to Table I for the standard DOLPHIn method. It should however also be mentioned that OMP requires up to k iterations per subproblem, while in Algorithm 1 we explicitly restricted the number of ISTA iterations in the A-update to just a single one.

A concrete example is given in Figure 6; here, we consider the color “mandrill” image, for which the reconstruction algorithms (given just two quite heavily noise-corrupted octavary CDP measurements) were run on each of the three RGB channels separately. The sparsity-constrained DOLPHIn reconstructions appear superior to the plain WF solution both visually and in terms of the quality measures PSNR and SSIM.

V. DISCUSSION AND CONCLUSION

In this paper, we introduced a new method, called DOLPHIn, for dictionary learning from noisy nonlinear measurements without phase information. In the context of image reconstruction, the algorithm fuses a variant of the recent Wirtinger Flow method for phase retrieval with a patch-based dictionary learning model to obtain sparse representations of image patches, and yields monotonic objective decrease or (with appropriate step size selection) convergence to a stationary point for the nonconvex combined DOLPHIn model.

Our experiments demonstrate that dictionary learning for phase retrieval with a patch-based sparsity is a promising direction, especially for cases in which the original Wirtinger Flow approach fails (due to high noise levels and/or relatively low sampling rates).

Several aspects remain open for future research. For instance, regarding the generally difficult task of parameter tuning, additional benchmarking for to-be-identified instance settings of special interest could give further insights on how to choose, e.g., the regularization parameters in relation to varying noise levels.

It may also be worth developing further variants of our algorithm; the successful use of ℓ_0-constraints instead of the ℓ_1-penalty, combining OMP with our framework, is just one example. Perhaps most importantly, future research will be directed towards the “classic” phase retrieval problem in which one is given the (squared) magnitudes of Fourier measurements, see, e.g., [2], [1], [5]. Here, the WF method fails, and existing other (projection-based) methods are not always reliable either. The hope is that introducing sparsity via a learned dictionary will also enable improved reconstructions in the Fourier setting.

To evaluate the quality of the learned dictionary, one might also ask how DOLPHIn compares to the straightforward approach to first run (standard) phase retrieval and then learn dictionary and sparse patch representations from the result. Some preliminary experiments (see also those in Section IV-F pertaining to keeping both A and D fixed in the first DOLPHIn phase) indicate that both approaches produce comparable results in the noiseless setting, while our numerical results demonstrate a denoising feature of our algorithm that the simple approach obviously lacks.

Similarly, it will be of interest to see if the dictionaries learned by DOLPHIn can be used successfully within sparsity-aware methods (e.g., the Thresholded WF proposed in [13], if that were modified to handle local (patch-based) sparsity instead of global priors). In particular, learning dictionaries for patch representations of images from a whole class of images would then be an interesting point to consider. To that
Thus, if \( g \) gives the number of patches this component is contained in. Diagonal entry is associated to a specific vector component and \( R \) matrix \( P \) will be case, to \( F \) a \( X \) loss of phase (or, more generally, measurement nonlinearity), be extended to multiple input images whose patches are all formulas for the 2D-case can be obtained by applying the ones in the one-dimensional setting (1). In particular, the (gradient)

\[
\nabla \text{loss of phase (or, more generally, measurement nonlinearity), be extended to multiple input images whose patches are all formulas for the 2D-case can be obtained by applying the ones in the one-dimensional setting (1). In particular, the (gradient)
\]

Another interesting aspect to evaluate is by how much re-
construction quality and achievable sparsity degrade due to the loss of phase (or, more generally, measurement nonlinearity), compared to the linear measurement case.

### APPENDIX

In the following, we derive the DOLPHIn algorithm for the one-dimensional setting (1). In particular, the (gradient) formulas for the 2D-case can be obtained by applying the ones given below to the vectorized image \( x = \text{vec}(X) \) (stacking columns of \( X \) on top of each other to form the vector \( x \)), the vectorized matrix \( a = \text{vec}(A) \), and interpreting the matrix \( F \in \mathbb{C}^{M \times N} \) as describing the linear transformation corresponding to \( F \) in terms of the vectorized variables.

We now have a patch-extraction matrix \( P_c \in \mathbb{R}^{ps \times N} \) which gives us \( P_c x = ((x^1)^T, \ldots, (x^p)^T)^T \) (in the vectorized 2D-case, \( x^i \) then is the vectorized \( i \)-th patch of \( X \), i.e., \( P_c \) corresponds to \( E \)). Similarly, we have a patch-reassembly matrix \( P_a \in \mathbb{R}^{N \times ps} \); then, the reassembled signal vector will be \( P_a ((a_1)^T, \ldots, (a_p)^T)^T \). (so \( P_a \) corresponds to \( R \)). Note that \( P_a = P_c^T = ((P_c^T P_c)^{-1})^T \); in particular, \( x = P_a P_c x \) and \( P_a P_c \) is a diagonal matrix for which each diagonal entry is associated to a specific vector component and gives the number of patches this component is contained in. (Thus, if \( x = \text{vec}(X) \) is a vectorized 2D-image, \( P_c^T P_c x = \text{vec}(R \odot X) \) with \( R \) as defined in Section III-B.)

Note that for nonoverlapping patches, \( P_c \) is simply a permutation matrix, and \( P_a = P_c^T \) (so \( P_a P_c = I \)). Also, applying just \( P_c^T \) actually reassembles a signal from patches by simply adding the component’s values without averaging.

We wish to represent each patch as \( x^i \approx D a^i \) with sparse coefficient vectors \( a^i \); with \( a := ((a_1)^T, \ldots, (a_p)^T)^T \) and \( D := I_p \otimes D \), this sparse-approximation relation can be written as \( P_c x \approx \hat{D} a \). Our model to tackle the 1D-problem (1) reads

\[
\min_{x, D, a} \frac{1}{4} \| y - |F x|^2 \|_2^2 + \frac{\mu}{2} \| P_c x - \hat{D} a \|_2^2 + \lambda \| a \|_1
\]

s.t. \( x \in \mathcal{X} := [0, 1]^N \), \( D \in \mathcal{D} \);

\[
\text{(11)}
\]

here, \( y := \|F x\|_2^2 + n \), with \( \hat{x} \) the original signal we wish to recover and \( n \) a noise vector.

The update formulas for \( a \) (separately for \( a^1, \ldots, a^p \)) and \( D \) remain the same as described before, see Sections III-A and III-C, respectively. However, the update problem for the phase retrieval solution—now derived from (11), with \( D \) and \( a \) fixed at their current values—becomes decreasing the objective

\[
\min_{x, D, a} \frac{1}{4} \| y - |F x|^2 \|_2^2 + \frac{\mu}{2} \| P_c x - \hat{D} a \|_2^2 \quad \text{with} \quad x \in \mathcal{X}.
\]

\[
\text{(12)}
\]

We approach this by means of a projected gradient descent step; since we consider real-valued \( x \)-variables, this essentially amounts to (one iteration of) the Wirtinger Flow method [12], accommodating the \([0, 1]\)-box constraints via projection onto them after the (Wirtinger) gradient step. The step size will be chosen to achieve a reduction of the objective w.r.t. its value at the previous \( x \).
The objective function (12) can be rewritten as
\[
\xi(x) := \phi(x) + \psi(x) = \frac{1}{2} \sum_{j=1}^{N} (y_j - x^\top F^*_j F_j x)^2 + \frac{\mu}{2} \| P_e x - Da \|^2.
\]

The gradient of \(\psi(x)\) is straightforwardly found to be
\[
\nabla \psi(x) = \mu P^\top_e (P_e x - Da).
\]

Regarding \(\nabla \phi(x)\) is \[
= \frac{1}{2} \sum_{j=1}^{N} (y_j - x^\top M^\top j M^j x) \cdot \nabla (y_j - x^\top M^\top j M^j x),
\]
where \(M^j = F^*_j F_j\) (\(F_j\) is the j-th row of \(F\)), we first note that \((M^j)^* = M^j\) and hence, in particular, \(M_{ik}^j = M_{ik}\) (i.e., \(\Re(M_{ik}^j) = \Re(M_{ik})\) and \(\Im(M_{ik}^j) = -\Im(M_{ik})\)). Thus, it is easily seen that for each \(i = 1, \ldots, N\), the terms in the double-sum \(x^\top M^j x = \sum_{\ell=1}^{N} \sum_{k=1}^{N} M_{ik}^\ell x_{\ell} x_k\) that contain \(x_i\) are precisely
\[
M_{ii}^j x_i^2 + x_i \sum_{k=1, k \neq i}^{N} M_{ik}^\ell x_{\ell} + x_i \sum_{\ell=1, \ell \neq i}^{N} M_{i\ell}^\ell x_i\]
\[
= M_{ii}^j x_i^2 + \left(2 \sum_{k=1, k \neq i}^{N} \Re(M_{ik}^j) x_k\right) x_i.
\]

Hence, \(\frac{\partial}{\partial x_i} (y_j - x^\top M^j x) = -2 \sum_{k=1}^{N} \Re(M_{ik}^j) x_k = -2\Re(M_{ik}^j) x\), and therefore \(\nabla \phi(x)\) is equal to
\[
- \frac{1}{2} \sum_{j=1}^{N} (y_j - x^\top M^j x) \cdot \left(2\Re(M_{i1}^j) x, \ldots, 2\Re(M_{iN}^j) x\right)^\top = - \sum_{j=1}^{N} (y_j - x^\top F^*_j F_j x) \Re(F^*_j F_j) x.
\]

Consequently,
\[
\nabla \xi(x) = \nabla \phi(x) + \nabla \psi(x) = \mu P^\top_e (P_e x - Da) - \sum_{j=1}^{N} (y_j - x^\top F^*_j F_j x) \Re(F^*_j F_j) x.
\]

Thus, with the projection \(P_X(x) = \max\{0, \min\{1, x\}\}\) (component-wise) and a step size \(\gamma^{X}_{(l)} > 0\), the update of the phase retrieval solution estimate in the \(l\)-th DOLPHin iteration for the 1D-case sets \(x^{(l+1)}\) to the value
\[
P_X \left( x^{(l)} - \gamma^{X}_{(l)} \left( \sum_{j=1}^{N} \left( (x^{(l)})^\top F^*_j F_j x^{(l)} - y_j \right) \Re(F^*_j F_j) x^{(l)} + \mu P^\top_e (P_e x^{(l)} - \hat{D}^{(l)} a^{(l)}) \right) \right) = \nabla^{X}_{(l)} \nabla \phi(x) = - \sum_{j=1}^{N} (y_j - x^\top F^*_j F_j x) \Re(F^*_j F_j) x.
\]

Further, using \(F^*_j = (F_j)^* = (F^*)^j\) and rearranging terms, this becomes
\[
\nabla \phi(x) = \Re \left( \sum_{j=1}^{N} (F^*)^j (|Fx|^2 - y_j) F_j x \right) = \Re \left( F^* \left( (|Fx|^2 - y) \odot Fx \right) \right).
\]

From this last form, it is particularly easy to obtain the gradient matrix \(\nabla \phi(x)\) in the 2D-case, which corresponds precisely to the gradient \(\nabla \phi(x)\) with \(x\) the vectorized matrix variable \(X\) and \(F\) representing the linear operator \(F(X)\) (in terms of \(x\), reshaped to match the size of \(X\) (i.e., reversing the vectorization process afterwards). Similarly, the expression for \(\nabla \psi(x)\) can be derived from \(\nabla \psi(x)\) w.r.t. the vectorized variable; here, the product with \(P^\top_e\) then needs to be replaced by an application of the adjoint \(\xi^*\), which can be recognized as \(\xi^*(\cdot) = R \circ \Re(\cdot)\) by translating the effect of multiplying by \(P^\top_e\) to the vectorized variable back to matrix form. (Analogously, one obtains \(R^* (Z) = \xi^*(1/R) \odot Z\), where \(1/R\) has entries \(1/R_{ij}\), i.e., is the entry-wise reciprocal of \(R\).)

**ACKNOWLEDGEMENTS**

We thank the anonymous referees for their comments, which helped to improve the paper.

**REFERENCES**
