A closed-form linear algebraic solution to the 2-D phase retrieval problem

Andrew E. Yagle
Dept. of EECS, The University of Michigan, Ann Arbor, MI 48109-2122

Abstract— **The 2-D discrete phase retrieval problem is to reconstruct an unknown image having known finite spatial extent from the magnitude of its discrete Fourier transform. Most methods for solving this problem are iterative but not POCS, and they tend to stagnate. Here we solve this problem by computing the null vectors of two lower-triangular-Toeplitz-block-lower-triangular-Toeplitz (LTBLT) matrices. The only assumption required (other than compact support) is that one corner pixel of the image have value zero and the opposite corner pixel be nonzero. We also demonstrate how to transform an arbitrary 2-D phase retrieval problem into one of this form, by computing a single root of one small polynomial. Several small and large examples illustrate the procedure.**

Keywords— **Phase Retrieval. EDICS: 2-REST**
Phone: 734-763-9810. Fax: 734-763-1503
Email: aey@eecs.umich.edu.

I. Introduction

A. Phase Retrieval Overview

The problem of reconstructing an image known to have compact support from its Fourier transform magnitudes arises in several disciplines in optics [1]. The image is reconstructed if the missing Fourier phase is recovered; hence the term "phase retrieval." For details of the history and applications of this problem see [1].

Since the image is assumed to have compact support, its Fourier transform may be sampled in wavenumber. Most images are approximately bandlimited to the extent that they may also be sampled spatially as well. This leads to the discrete version of this problem, in which a discrete-space image known to have finite spatial extent is to be reconstructed from the magnitude of its 2-D discrete Fourier transform (DFT). For details on discrete phase retrieval problems see [2]-[3]. Note that this precludes methods based on oversampling of continuous images. This is the problem solved in this paper.

The most common approach for phase retrieval problems is to use an iterative transform algorithm [4], which alternates between the spatial and wavenumber domains. However, these algorithms usually stagnate, failing to converge to a solution. Other approaches require the computationally expensive and extremely unstable numerical operation of tracking zero curves of algebraic functions [5]-[7]. A recursive procedure for images with triangular support given in [8]. We will not attempt to list all approaches here.

B. Contributions of This Paper

This paper proposes a novel approach to discrete 2-D phase retrieval. We assume that one corner pixel of the image is zero, and the opposite-corner pixel is nonzero (but otherwise unknown); we call this the "notched" phase retrieval problem. Note that this is *not* an assumption of triangular support, for which a recursive solution is known [8]; only *one* pixel need be zero. This can also be viewed as a way to avoid the image reversal ambiguity of 2-D phase retrieval (see below).

We formulate the 2-D phase retrieval problem in terms of lower triangular Toeplitz (LTT) polynomial matrices, and show that the image can be computed as the solution of a linear system of equations constructed recursively from the data. We then show that the polynomial representation can be replaced with a lower-triangular-Toeplitz-block-lower-triangular-Toeplitz (LTBLT) matrix one, in which the null vector of a matrix constructed from the data forms a linear system of equations whose solution is the image. This is similar to many recent proposed multichannel blind deconvolution algorithms.

We also give a procedure for transforming a non-notched 2-D phase retrieval problem into a notched 2-D phase retrieval problem, by computing a single root of a small polynomial. It should be noted that we have proposed a similar-looking algorithm for blind deconvolution [9], but this algorithm only applies to problems in which the unknown image and point-spread function have different aspect ratios, hence it cannot be applied to phase retrieval.

This paper is organized as follows. Section 2 formulates and discusses ambiguities of the 2-D discrete phase retrieval. Section 3 reviews lower triangular Toeplitz (LTT) matrices; for more details see [10]. Section 4 presents the matrix polynomial form of the algorithm. Section 5 presents the LTBLT matrix formulation of the algorithm. Section 6 shows how to transform a non-notched problem into a notched

problem. Several numerical examples illustrate the procedures in all three of these sections. Section 7 concludes with a summary and suggestions for future research.

II. Problem Formulation

A. Problem Statement

The 2-D discrete phase retrieval problem is as follows [2]-[3]. We observe

$$y(i_1, i_2) = u(i_1, i_2) * * u(M - i_1, M - i_2) \quad (1)$$

where $**$ denotes convolutions in i_1 and i_2. The 1-D convolution $*$ is defined here as

$$h(n) * u(n) = \sum_{i=0}^{n} h(n-i)u(i) = \sum_{i=0}^{n} h(i)u(n-i). \quad (2)$$

We will often omit commas between pairs of subscripts.

We make the following assumptions:

1. Image $u(i_1, i_2) = 0$ unless $0 \leq i_1, i_2 \leq M$;
2. Autocorrelation $y(i_1, i_2) = 0$ unless $0 \leq i_1, i_2 \leq 2M$;
3. Notch: $u(0,0) = 0; \quad u(M, M) \neq 0$;
4. All quantities are real functions.

The goal is to reconstruct the image $u(i_1, i_2)$ from its autocorrelation $y(i_1, i_2)$. Taking the discrete Fourier transform (DFT) of (1) shows that this is equivalent to computing the DFT $U(k_1, k_2)$ from its squared magnitude $Y(k_1, k_2) = |U(k_1, k_2)|^2$; hence the term "phase retrieval." We will not be using the DFT in this paper. No stochastic assumptions are made about $u(i_1, i_2)$, precluding methods based on cumulants, ARMA or Poisson image models, or stochastic equalization.

B. Problem Ambiguities

There are four ambiguities in 2-D phase retrieval:

1. *Scale factor:* If $u(i_1, i_2)$ is a solution, then the scaled $-u(i_1, i_2)$ is also a solution. For the complex-valued problem, $cu(i_1, i_2)$ is also a solution for any constant such that $|c| = 1$. If the image is known to be non-negative, clearly this is not a problem;
2. *Translation:* If $u(i_1, i_2)$ is a solution, then the shifted $u(i_1 + d_1, i_2 + d_2)$ is also a solution for any integers (d_1, d_2). We eliminate this ambiguity by specifying the support $0 \leq i_1, i_2 \leq M$;
3. *Reversal:* Either $u(i_1, i_2)$ or its reversal $u(M - i_1, M - i_2)$ can regarded as the "true" image. The notch assumption eliminates this by taking $u(0,0) = 0$ and $u(M, M) \neq 0$ (our method requires it regardless);
4. *Irreducibility:* The 2-D z-transform

$$U(x, y) = \sum_{i_1=0}^{M} \sum_{i_2=0}^{M} u(i_1, i_2) x^{i_1} y^{i_2} \quad (3)$$

is irreducible ($U(x, y)$ cannot be factored). This is almost surely true [11].

One way to see this last fact quickly is to note that (1) can be viewed as $\frac{1}{2}(2M+1)^2$ simultaneous quadratic equations in $(M+1)^2$ unknowns. Note that about half of the $(2M+1)^2$ equations (1) are redundant since $y(i_1, i_2) = y(2M - i_1, 2M - i_2)$. The problem is overdetermined, so by Bezout's theorem there is almost surely at most one solution. Indeed, generically there are *no* solutions; only from (1) do we know a solution exists.

The 1-D phase retrieval problem has an additional, non-trivial ambiguity. There are almost surely $2^{M/2}$ solutions to an $(M+1)$-point 1-D phase retrieval problem if M is even. This can be seen by noting that the zeros of the 1-D z-transform of the autocorrelation occur in reciprocal conjugate quadruples (if z_o is a zero, then $z_o^*, 1/z_o, 1/z_o^*$ are also zeros). Either z_o and z_o^*, or their reciprocals, can be assigned to the 1-D signal; since there are $M/2$ quadruples, this assignment can be made in $2^{M/2}$ ways (fewer if there are zeros on the unit circle or real axis).

III. Lower Triangular Toeplitz (LTT) Matrices

We quickly review some basic properties of LTT matrices for later use. An $(M+1) \times (M+1)$ LTT matrix A and its associated polynomial $a(z)$ have the forms

$$A = \begin{bmatrix} a_0 & 0 & \cdots & 0 \\ a_1 & a_0 & \ddots & 0 \\ \vdots & \ddots & \ddots & 0 \\ a_M & a_{M-1} & \cdots & a_0 \end{bmatrix} \quad (4a)$$

$$a(z) = a_0 + a_1 z + \ldots + a_M z^M. \quad (4b)$$

A. Basic Properties

If B is another LTT with associated polynomial $b(z)$, then it is easy to see that the following matrices are all LTT with associated polynomials

$$C = A + B; \quad c(z) = a(z) + b(z); \quad (5a)$$

$$D = AB = BA; \quad d(z) \equiv a(z)b(z) \, mod(z^{M+1}); \quad (5b)$$

$$E = A^{-1}; \quad e(z)a(z) \equiv 1 \, mod(z^{M+1}); \quad (5c)$$

$$e(z)a(z) + f(z)z^{M+1} = 1 \quad (5d)$$

for some (irrelevant) polynomial $f(z)$.

These imply that the set of LTT matrices with the usual matrix operations form a *commutative ring with identity whose units are the LTT matrices with nonzero diagonal.* That is, LTT matrices are closed and commute under matrix addition and multiplication, and they are invertible if the diagonal is nonzero.

Note that the following are equivalent:
- Equation (5d) has a solution;
- $a(z)$ and z^{M+1} are relatively prime;
- Constant $a_0 \neq 0$;
- LTT matrix A is invertible.

It is also worth noting that (5) also holds for circulant matrices, provided z^{M+1} is replaced with $z^{M+1}-1$. That is, for LTT matrices overflow is discarded, while for circulant matrices overflow is aliased. For both of these matrices, the matrix product $C = AB$ is equivalent to the matrix-vector product of A and the first column of B yielding the first column of C.

B. LTT Matrix Inversion

Recall from high school algebra the formula

$$A^{-1} = adj[A]/det[A] = adj[A]/a_0^{M+1}, \qquad (6)$$

where $adj[A]$ is the transpose of the matrix of cofactors; (6) shows that $adj[A]$ is LTT when A is LTT. This also shows that the first column of $adj[A]$ is a null vector for A when A is singular.

From (5d), the Euclidian algorithm can be used to compute the inverse of an LTT matrix. However, a faster approach which is $O(M \log M)$ is to note that

$$\begin{bmatrix} A_1 & 0 \\ A_2 & A_1 \end{bmatrix}^{-1} = \begin{bmatrix} A_1^{-1} & 0 \\ -A_1^{-1}A_2A_1^{-1} & A_1^{-1} \end{bmatrix}. \qquad (7)$$

Hence the inversion of a LTT matrix is equivalent to the inversion of one LTT matrix of half the size and two LTT matrix multiplications, which can be computed using the FFT on (5b).

All of these results are well known. We found [10] to be a useful reference.

IV. Matrix Polynomial Algorithm

A. Problem Formulation

To avoid confusing $u(i_1, i_2)$ and $u(M - i_1, M - i_2)$, we solve not (1) but the 2-D blind deconvolution problem

$$y(i_1, i_2) = u(i_1, i_2) ** h(i_1, i_2); h(i_1, i_2) = u(M-i_1, M-i_2) \qquad (8)$$

The procedure actually solves the problem for arbitrary $(M+1)\times(M+1)$ $h(i_1, i_2)$ such that $h(0,0) \neq 0$.

This can be written using $(N+1)\times(N+1)$ LTT polynomial matrices as

$$Y(z) = U(z)H(z); \qquad (9a)$$

$$Y(z) = \begin{bmatrix} Y_0(z) & 0 & \cdots & 0 \\ Y_1(z) & Y_0(z) & \ddots & 0 \\ \vdots & \ddots & \ddots & 0 \\ Y_N(z) & Y_{N-1}(z) & \cdots & Y_0(z) \end{bmatrix} \qquad (9b)$$

$$Y_n(z) = y(0,n) + y(1,n)z + \ldots + y(2M,n)z^{2M} \qquad (9c)$$

where
1. $N > 2M$ is an integer to be determined;
2. $Y_n(z) = 0$ for $2M < n \leq N+1$;
3. $U(z)$ and $H(z)$ are defined similarly;
4. One 1-D convolution in (8) is written as a product of z-transforms;
5. The other 1-D convolution in (8) is written as a product of LTT matrices (see (5b)).

We can rewrite (9a) as

$$Y(z)^{-1}U(z) = H(z)^{-1} \qquad (10)$$

and using (6) transforms (10) into

$$adj[Y(z)]U(z) = adj[H(z)]det[U(z)] \qquad (11)$$

since $det[U(z)] = det[Y(z)]/det[H(z)]$.

B. Problem Solution

Now we make use of the "notched" assumption $u(0,0) = 0$. We have

$$u(0,0) = 0 \rightarrow U_0(z) = u(1,0)z + \ldots + u(M,0)z^M$$

$$\rightarrow det[U_0(z)] = U_0^{N+1}(z) = z^{N+1}p(z) \qquad (12)$$

for some (irrelevant) polynomial $p(z)$. Equating coefficients of z in (11) and using (12) yields $N+1$ equations (all equal to zero) in $(M+1)^2 - 1$ unknowns $u(i_1, i_2)$. If $N \geq (M+1)^2 - 2$ these linear equations can be solved for the unknown image $u(i_1, i_2)$!

Defining the LTT polynomial matrix

$$W(z) = adj[Y(z)] \qquad (13)$$

the linear equations are the coefficients of (using the notation of (9b))

$$W_N(z)U_0(z) + \ldots + W_{N-M}(z)U_M(z) = O(z^{N+1}) \qquad (14)$$

where $O(z^{N+1})$ is a polynomial in which the coefficients of $\{1, z \ldots z^N\}$ are all zero. Note the following:
1. The system matrix formed from (14) has $M+1$ Toeplitz block columns, so it is Toeplitz-block-Toeplitz (TBT) in $u(i_1, i_2)$;

2. $W_n(z)$ can be computed recursively from $Y_n(z)$ using back substitution, with each polynomial multiplication implemented using the FFT and then reduced $mod(z^{N+1})$;
3. $u(i_1, i_2)$ is only determined to a scale factor, since it is computed as the null vector of the system matrix formed from (14). This is as expected from the scale factor ambiguity;
4. The notch assumption $u(0,0) = 0$ and $u(M,M) \neq 0$ eliminates the reversal ambiguity.

Note that (14) is actually the last row of (11). It may seem that using all the rows of (11) would enable use of a much smaller value of N. However, each row of (11) contributes only one new (independent of previous ones) equation, so the last row includes the contributions of all rows above it.

C. Small Illustrative Example

We present a small example to illustrate the procedure. We wish to solve the phase retrieval problem

$$\begin{bmatrix} 0 & a \\ b & c \end{bmatrix} ** \begin{bmatrix} c & b \\ a & 0 \end{bmatrix} = \begin{bmatrix} 0 & 8 & 6 \\ 12 & 29 & 12 \\ 6 & 8 & 0 \end{bmatrix}. \qquad (15)$$

Of course, for a 2×2 problem the notched support is equivalent to triangular support, so the algorithm of [8] could be used here. But this is not true for larger problems.

We have $M = 1$ (recall the image is $(M+1) \times (M+1)$) so that

$$N \geq (M+1)^2 - 2 \rightarrow N \geq 2. \qquad (16)$$

Following the above procedure, we have the following equations:

$$Y(z) = \begin{bmatrix} 0 + 12z + 6z^2 & 0 & 0 \\ 8 + 29z + 8z^2 & 0 + 12z + 6z^2 & 0 \\ 6 + 12z + 0z^2 & 8 + 29z + 8z^2 & 12z + 6z^2 \end{bmatrix}; \qquad (17a)$$

$$W(z) = \begin{bmatrix} Y_0^2(z) & 0 & 0 \\ -Y_0(z)Y_1(z) & Y_0^2(z) & 0 \\ Y_1^2(z) - Y_0(z)Y_2(z) & -Y_0(z)Y_1(z) & Y_0^2(z) \end{bmatrix}, \qquad (17b)$$

$$W(z) = \begin{bmatrix} 144z^2 + 144z^3 + 36z^4 & 0 & 0 \\ -(96z + 396z^2 + O(z^3)) & \ddots & 0 \\ 64 + 392z + O(z^2) & \ddots & \ddots \end{bmatrix}; \qquad (17c)$$

$$W(z) \begin{bmatrix} (0 + bz) \\ (a + cz) \\ 0 \end{bmatrix} = O(z^3); \qquad (17d)$$

$$\begin{bmatrix} 144 - 96 & 0 & \\ -96 & 64 & 0 \\ 396 & -392 & 96 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}; \qquad (17e)$$

$$[a, b, c] = [2, 3, 4]; \quad \text{(to scale factor)} \qquad (17f);$$

$$\begin{bmatrix} 0 & a \\ b & c \end{bmatrix} = \begin{bmatrix} 0 & 2 \\ 3 & 4 \end{bmatrix} \qquad (17g)$$

where the scale factor is found (to a sign) by inserting (17f) back into (16).

D. Larger Numerical Example

Next, consider the reconstruction of the 17×17 notched "eye" image in Figure 1a. Its 33×33 autocorrelation is shown in Figure 1b. The reconstructed image is shown in Figure 1c, and matches the original image perfectly. Since the image pixels are known to be integers, all computations were performed $mod(65537)$; the 289×289 linear system was solved using Gaussian elimination $mod(65537)$. This is extremely inefficient, since it does not exploit the TBT structure of the linear system, but it works.

The Matlab program used is also shown. To avoid the scaling issue, the opposite corner from the notch is set to unity; this could easily be avoided. The program will work on an integer-valued image of arbitrary square size, provided the prime modulus is large enough to avoid zero-valued pivots.

V. LTBLT Matrix Algorithm

A. Problem Formulation

The large numbers (cf. "396") that arise in even the small example above suggest that numerical overflow problems will arise in the computation of $W(z)$. This suggests the need for an alternative formulation of the procedure.

Since (14) has on its right side an unknown polynomial $O(z^{N+1})$, there is no reason not to perform all polyomial computations $mod(z^{N+1})$, i.e., discard all terms of degree $N+1$ or greater. From (5), this is equivalent to adding and multiplying LTT matrices. Hence the polynomial operations above can be replaced with LTT matrix operations, and the procedure can be implemented by nesting LTT matrices in LTT matrices, which result in lower triangular-Toeplitz-block-lower-triangular-Toeplitz (LTBLT) matrices.

The significance of this reformulation is that (13) and (14) can now be rewritten as the $(N+1)^2 \times (N+1)^2$ matrix equations

$$YW = [0]; \quad WU = [0] \qquad (18)$$

where W is the $(N+1)^2 \times (N+1)^2$ LTBLT matrix

$$W = \begin{bmatrix} W_0 & 0 & \cdots & 0 \\ W_1 & W_0 & \ddots & 0 \\ \vdots & \ddots & \ddots & 0 \\ W_N & W_{N-1} & \cdots & W_0 \end{bmatrix} \qquad (19)$$

with block LTT matrices (compare to (9))

$$W_n = \begin{bmatrix} w(0,n) & 0 & \cdots & 0 \\ w(1,n) & w(0,n) & \ddots & 0 \\ \vdots & \ddots & \ddots & 0 \\ w(N,n) & w(N-1,n) & \cdots & w(0,n) \end{bmatrix} \tag{20}$$

and U and Y are defined similarly.

B. Problem Solution

The point of this reformulation is that the computation (13) of the adjoint of the polynomial matrix $Y(z)$, which takes much computation and results in numerical overflows, is really nothing but a very inefficient computation of the null vector of the LTBLT matrix Y, as noted below (6)!

The original problem can be reformulated as a $(N+1)^2 \times (N+1)^2$ system $Y = HU$ where Y, H, U are all LTBLT matrices. We then have

1. $u(0,0) = 0 \rightarrow y(0,0) = 0$;
2. $u(0,0) = 0 \rightarrow det[U] = 0$;
3. $y(0,0) = 0 \rightarrow det[Y] = 0$;
4. $u(M,M) \neq 0 \rightarrow det[H] \neq 0$.

since $u(0,0)$ is the diagonal element of U and $y(0,0)$ the one for Y. Since Y and U are singular, they have a common null vector w since H is nonsingular. That is,

$$Y = HU \rightarrow 0 = Yw = H(Uw)$$

$$\rightarrow 0 = H^{-1}0 = Uw \rightarrow [0] = UW \rightarrow 0 = Wu \tag{21}$$

since matrix multiplication of LTBLT matrices is equivalent to matrix times first column equals first column multiplication. Here 0 is a zero vector and [0] is a zero matrix, and similarly for w.

This lead to the following alternative procedure:

1. Form the LTBLT matrix Y from $y(i_1, i_2)$;
2. Compute a null vector w of Y;
3. Form the LTBLT matrix W with first column w;
4. Compute the null vector u of W with zeros in all but the appropriate $(M+1)^2$ places;
5. Read off $u(i_1, i_2)$ from u.

Note the following:

1. While the matrix adjoint (6) could be used to compute w, almost any other method would be faster and lead to smaller numbers;
2. There are in fact $N+1$ null vectors of Y; any or all of them can be used to form the second LTBLT system;
3. Although Y is $(N+1)^2 \times (N+1)^2$, its LTBLT structure means that matrix manipulations can be performed using the 2-D FFT and discarding half of the results. So the computational load is actually no greater than the matrix polynomial method.
4. It might seem from (21) that N need not be as large as $(M+1)^2 - 2$ for this to work. But running the procedure on a notched 3×3 image (8 unknowns) with $N = 4$ results in a 25×8 system using any one null vector, and a 125×8 system using all five null vectors. Remarkably, the 125×8 system matrix has rank four! The minimum value of N is seven, as expected.

C. Small Illustrative Example

We apply this procedure to the 2-D phase retrieval problem (15). Following the above procedure, we have the following equations:

$$Y = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 12 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 6 & 12 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 29 & 8 & 0 & 12 & 0 & 0 & 0 & 0 & 0 \\ 8 & 29 & 8 & 6 & 12 & 0 & 0 & 0 & 0 \\ 6 & 0 & 0 & 8 & 0 & 0 & 0 & 0 & 0 \\ 12 & 6 & 0 & 29 & 8 & 0 & 12 & 0 & 0 \\ 0 & 12 & 6 & 8 & 29 & 8 & 6 & 12 & 0 \end{bmatrix} \tag{22}$$

which has as expected a (right) nullspace of dimension three spanned by

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 9 & 9 & 0 \\ 0 & 0 & 0 \\ -6 & -6 & 0 \\ 3.7992 & -1146 & 0 \\ 4 & 4 & 0 \\ 5.4672 & 771.8 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{23}$$

where we have scaled the three column vectors to show their similarity.

Omitting $0 = 0$ rows and the columns of the system matrix multiplied by zero, the second system of (21) is

$$\begin{bmatrix} 9 & -6 & 0 \\ -6 & 4 & 0 \\ 3.7992 & 5.4672 & -6 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}; \tag{24a}$$

$$[a, b, c] = [2, 3, 4]; \quad \text{(to scale factor)} \tag{24b}$$

$$\begin{bmatrix} 0 & a \\ b & c \end{bmatrix} = \begin{bmatrix} 0 & 2 \\ 3 & 4 \end{bmatrix} \tag{24c}$$

where the scale factor is found (to a sign) as before.

Note how much smaller the entries in the system matrix are compared to (17f). Of course, the systems are equivalent to each other, but numbers generated by finding the nullspace are smaller than those generated using the adjoint.

VI. Non-Notched to Notched Problem Transformation

A. Problem Solution

Now consider the general discrete 2-D phase retrieval problem for which $u(0,0) \neq 0$. In order to apply either of the above procedures, we must transform this problem into another problem for which $u(0,0) = 0$. This can be accomplished as follows.

The first row of (9a) is

$$Y_0(z) = U_0(z)H_0(z) \Leftrightarrow y(n,0) = u(n,0) * h(n,0). \tag{25}$$

$Y_0(z)$ is known, and this 1-D blind deconvolution problem has multiple non-trivial solutions, similarly to the 1-D phase retrieval problem. So there is no way to recover $U_0(z)$ and $H_0(z)$ directly from $Y_0(z)$.

Let z_0 be any non-multiple zero of $Y_0(z)$ such that $|z_0| \neq 1$. Then z_0 is a zero of either $U_0(z)$ or $H_0(z)$; without loss of generality let $U_0(z_0) = 0$ and $H_0(z_0) \neq 0$. Note that this eliminates the reversal ambiguity of distinguishing $u(i_1, i_2)$ and $h(i_1, i_2)$.

Now change variables from z to $y = z - z_0$. Then (25) becomes

$$y\tilde{Y}_0(y) = y\tilde{U}_0(y)\tilde{H}_0(y) \tag{26}$$

and similar transformations yield the other $\tilde{U}_i(y)$ and $\tilde{H}_j(y)$. The transformed problem is a notched 2-D blind deconvolution problem, which can be solved using either of the algorithms given above.

B. Small Illustrative Example

Consider the 2-D phase retrieval problem

$$\begin{bmatrix} 3 & 4 \\ 1 & 2 \end{bmatrix} ** \begin{bmatrix} 2 & 1 \\ 4 & 3 \end{bmatrix} = \begin{bmatrix} 6 & 11 & 4 \\ 14 & 30 & 14 \\ 4 & 11 & 6 \end{bmatrix}. \tag{27}$$

We now also supply the image to illustrate the transformation. We have

$$Y_0(z) = 6 + 14z + 4z^2 = 4(z+3)(z+\frac{1}{2}). \tag{28}$$

Without loss of generality, associate the zero at -3 with $U_0(z)$ and let $y = z + 3$. We then have

$$y\tilde{Y}_0(y) = 6 + 14(y-3) + 4(y-3)^2 = 0 - 10y + 4y^2 \tag{29a}$$

$$\tilde{Y}_1(y) = 11 + 30(y-3) + 11(y-3)^2 = 20 - 36y + 11y^2$$

$$\tilde{Y}_2(y) = 4 + 14(y-3) + 6(y-3)^2 = 16 - 22y + 6y^2 \tag{29c}$$

as well as

$$y\tilde{U}_0(y) = 3+1(y-3) = y; \quad \tilde{U}_1(y) = 4+2(y-3) = 2y-2 \tag{30a}$$

$$\tilde{H}_0(y) = 2+4(y-3) = 4y-10; \tilde{H}_1(y) = 1+3(y-3) = 3y-8. \tag{30b}$$

This results in the transformed and now notched problem

$$\begin{bmatrix} 0 & -2 \\ 1 & 2 \end{bmatrix} ** \begin{bmatrix} -10 & -8 \\ 4 & 3 \end{bmatrix} = \begin{bmatrix} 0 & 20 & 16 \\ -10 & -36 & -22 \\ 4 & 11 & 6 \end{bmatrix} \tag{31}$$

which can be solved using the methods described above. Then the transformation $z = y - 3$ recovers the solution to the original problem. Horner's method [10] is the most efficient procedure for performing these polynomial transformations.

VII. Conclusion

We have presented a new algorithm for the 2-D discrete phase retrieval problem. Unlike previous "exact" algorithms, no tracking of zero curves or surfaces is required. The only operations required are computations of the null vectors of two heavily structured and sparse LTBLT matrices. The 2-D FFT can be used to compute the matrix-vector product in an iterative algorithm. However, when the image pixels are known to be bounded integers, so that the autocorrelation pixels are also bounded integers, finite field computations (modulo a large prime) can be used to reconstruct the image perfectly. Our experiments indicate that the prime modulus should be several times the number of pixels in the image, to avoid zero-valued pivots.

The performance of these algorithms in noise still requires extensive numerical testing. The computation of null vectors and their subsequent use to compute the image is the same as that in many so-called "direct" algorithms for multichannel blind deconvolution. Hence the many results that have been applied to that problem in the 1990s are applicable to the present problem as well, with similar expected results.

VIII. Acknowledgment

The work of the author was supported in part by DARPA and ARO under grant #DAAD-19-02-1-0262.

References

[1] J.C. Dainty and J.R. Fienup, "Phase retrieval and image reconstruction for astronomy," in *Image Recovery: Theory and Application*, ed. by H. Stark, Academic Press, San Diego, 1987, pp. 231-275.

[2] M.H. Hayes, "The reconstruction of a multidimensional sequence from the phase or magnitude of its Fourier transform," *IEEE Trans. ASSP* 30(2), 140-154 (1982).

[3] M.H. Hayes, J.S. Lim, and A.V. Oppenheim, "Signal reconstruction from phase or magnitude," *IEEE Trans. ASSP* 28(6), 672-680 (1980).

[4] J.R. Fienup, "Phase retrieval algorithms: a comparison," *Applied Optics* 21, 2758-2769, 1982.
[5] H.V. Deighton, M.S. Scivier, and M.A. Fiddy, "Solution of the two-dimensional phase retrieval problem," *Opt. Lett.* 10, 250-251 (1985).
[6] D. Izraelevitz and J.S. Lim, "A new direct algorithm for image reconstruction from Fourier transform magnitude," *IEEE Trans. ASSP* 35(4), 511-519 (1987).
[7] R.G. Lane, W.R. Fright and R.H.T. Bates, "Direct phase retrieval," *IEEE Trans. ASSP* 35(4), 520-526 (1987).
[8] M.H. Hayes and T.F. Quatieri, "Recursive phase retrieval using boundary conditions," *J. Opt. Soc. Am* 73, 1427-1433 (1983).
[9] A.E. Yagle, "A simple closed-form linear algebraic solution to the single-blur 2-D blind deconvolution problem for compactly-supported images," submitted to *Linear Algebra and its Applications*, March 2003.
[10] V. Pan and D. Bini, *Polynomial and Matrix Computations*, Birkhaeuser, Boston, 1994.
[11] J.L.C. Sanz and T.S. Huang, "Polynomial systems of equations and its application to the study of the effect on noise of multidimensional Fourier transform phase retrieval from magnitude," *IEEE Trans. ASSP* 33(8), 997-1004 (1985).

FIG. 1a: ORIGINAL

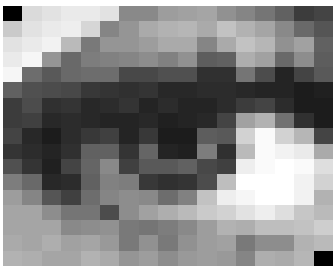


FIG. 1b: AUTOCORRELATION

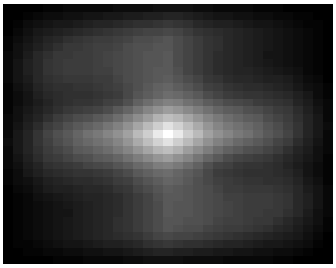
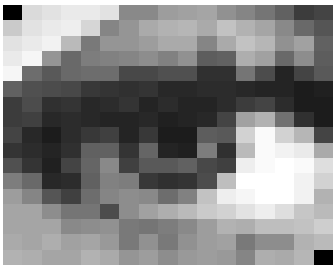


FIG. 1c: RECONSTRUCTED



MATLAB PROGRAM USED TO GENERATE FIGURES 1

```
Should combine these FOR loops! But easier to follow.
Need: Upper left corner=0;Lower right corner=1
(for scale; can easily get around that at the end).
load lena.mat;U=xx(125:141,125:141);U(1,1)=0;U(17,17)=1;
Y=conv2(U,fliplr(flipud(U)));M=length(U);N=65537;All mod(N)
Z(M*M,2*M-1)=0;Z(1,1)=1;Z(1:2*M-1,2)=Y(:,2);
W(M*M,2*M-1)=0;W(1,1)=1;W(1:2*M-1,2)=-Y(:,2);
Multiply each Yi(z) by Y1exp(i-2)(z), as in example above:
for I=3:2*M-1;ZZ=Y(:,I);for J=1:I-2;ZZ=rem(conv(ZZ,Y(:,1)),N)
if(length(ZZ)¿M*M);Z(:,I)=ZZ(1:M*M);else
Z(1:length(ZZ),I)=ZZ;end;end
Implement back substitution recursion:
for J=1:I-1;WW=rem(conv(Z(:,I-J+1),W(:,J)),N);
if(length(WW)¿M*M);W(:,I)=W(:,I)-WW(1:M*M);else;
W(1:length(WW),I)=W(1:length(WW),I)-WW;end;end;end
Use only most recent (2M-1) 1/Yn:
for I=2*M:M*M;for J=2:2*M-1;W(:,J-1)=W(:,J);end;
W(:,2*M-1)=zeros(M*M,1);
for J=I-2*M+2:I-1;
WW=rem(conv(Z(:,I-J+1),W(:,J+2*M-1-I)),N);
if(length(WW)¿M*M);W(:,2*M-1)=W(:,2*M-1)-WW(1:M*M);else
W(1:length(WW),2*M-1)=W(1:length(WW),2*M-1)-WW;end;end;end
Multiply each Xi(z) by Y1exp(N-i)(z):
for I=1:2*M-2;ZZ=W(:,I);for J=I:2*M-2;ZZ=rem(conv(ZZ,Y(:,1)),N);end
if(length(ZZ)¿M*M);W(:,I)=ZZ(1:M*M);else;
W(1:length(ZZ),I)=ZZ;end;end
Assemble Toeplitz matrix and find its null vector=solution:
T(M*M*M*M,1)=0;for I=1:M;TT=toeplitz(W(:,2*M-I),zeros(1,M));
for J=1:M*M;
T(M*M*(J-1)+1+(I-1)*M:M*M*(J-1)+M+(I-1)*M)=TT(J,:)';
end;end; Use Gaussian elimination to find nullvector of T:
for I=1:M*M;[G,TI,B]=gcd(T(I+M*M*(I-1)),N);
II=M*M*(I-1)+1:M*M*I;
for I=1:M*M;[G,TI,B]=gcd(T(I,I),N);T(I,:)=rem(T(I,:)*TI,N);
T(II)=rem(T(II)*TI,N);for J=I+1:M*M;JJ=M*M*(J-1)+1:M*M*J;
T(JJ)=rem(T(JJ)-T(II)*T(I+M*M*(J-1)),N);end;end;X(M*M,1)=1;
for I=1:M*M-1;
X(M*M-I)=rem(-T(M*M*(M*M-I-1)+1:M*M*(M*M-I))'*X,N);
if(X(M*M-I)¡0);X(M*M-I)=X(M*M-I)+N;end;end;
UHAT=reshape(X,M,M);
subplot(331),imagesc(U),colormap(gray)
subplot(334),imagesc(Y),colormap(gray)
subplot(337),imagesc(UHAT),colormap(gray)
```