2-D blind deconvolution for compactly-supported images by finding a null vector of a Toeplitz-block-Toeplitz linear system of equations

Andrew E. Yagle
Dept. of EECS, The University of Michigan, Ann Arbor, MI 48109-2122
aey@eecs.umich.edu

April 9, 2021

Abstract

The 2-D blind deconvolution problem is to reconstruct an unknown image and an unknown point-spread function (PSF) from their known 2-D convolution. The unknown image and PSF both have compact support (known finite spatial extents). Their known 2-D convolution is represented as a Toeplitz-block-Toeplitz (TBT) matrix, which is the product of two unknown TBT matrices whose elements are the unknown pixel values of the image and PSF. The image is reconstructed directly in the null vector of the TBT matrix of the known 2-D convolution. No additional processing, such as deconvolution of the now-known PSF, is needed. The scale factor ambiguity of the reconstructed image is manifested in the unknown scale factor of the null vector. The only requirements are: (1) the support of the unknown image is non-square; and (2) the TBT matrices of the unknown PSF and image have full ranks.

1 INTRODUCTION

1.1 Blind Deconvolution

The problem of reconstructing an image with known compact support from its 2-D convolution with an unknown blurring or point-spread function (PSF) arises in several disciplines, including image restoration from an unknown blurring agent, remote sensing through the atmosphere, and medical imaging.

Since both the image and point-spread function (PSF) can be assumed to have finite spatial extent (i.e., finite support), their Fourier transforms 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 image with finite extent is to be reconstructed from its 2-D discrete convolution with an unknown discrete PSF with finite spatial extent. This precludes methods based on oversampling of continuous images.

1.2 Previous Work

A common approach for blind deconvolution problems is to use an iterative transform algorithm [1], which alternate between the spatial and wavenumber domains. However, these algorithms often stagnate, failing to converge to a solution. Other approaches require the computationally expensive and extremely unstable numerical operation of tracking zero sheets of algebraic functions, or statistical estimation algorithms that also may not converge. We will not attempt to review or even list all approaches here.

The problem addressed in this paper should be distinguished from the problem of *multichannel* blind deconvolution, addressed in many papers. In the latter problem, a single unknown signal or image is filtered with several unknown blurring functions, resulting in several known outputs. This is much simpler than *single-blur* blind deconvolution.

1.3 Problem Statement

The 2-D discrete blind deconvolution problem is as follows. We observe y_{i_1,i_2} where

$$y_{i_1,i_2} = h_{i_1,i_2} * * x_{i_1,i_2} \tag{1}$$

where $**$ denotes 2-D convolution in i_1 and i_2. 1-D and 2-D convolution are defined in the next section.

We make the following support assumptions:

1. Image $x_{i_1,i_2} = 0$ unless $\begin{smallmatrix} 0 \le i_1 \le M_1 - 1 \\ 0 \le i_2 \le M_2 - 1 \end{smallmatrix}$; $M_1 > M_2$;
2. PSF $h_{i_1,i_2} = 0$ unless $0 \le i_1, i_2 \le L - 1$;
3. Data $y_{i_1,i_2} = 0$ unless $\begin{smallmatrix} 0 \le i_1 \le L + M_1 - 2 \\ 0 \le i_2 \le L + M_2 - 2 \end{smallmatrix}$;
4. $M_1 > M_2$: the image is "tall." If $M_1 < M_2$, the problem can be rotated by 90^o.
5. All variables are real functions.

The goal is to reconstruct the image x_{i_1,i_2} from knowledge of only the data y_{i_1,i_2}; hence the term "blind deconvolution." No stochastic assumptions are made about either the image or the point-spread function. This precludes use of methods based on cumulants, ARMA or Poisson image models or stochastic equalization. Neither the image nor the PSF need be nonzero for all i_1, i_2 in the above ranges; the support constraints prevent translational ambiguity.

1.4 Problem Ambiguities

The ambiguities in 2-D blind deconvolution are:

1. *Scale factor:* If $\{h_{i_1,i_2}, x_{i_1,i_2}\}$ is a solution, then for any real constant c, $\{ch_{i_1,i_2}, \frac{1}{c}x_{i_1,i_2}\}$ is also a solution. If c cannot be determined from the known image energy, it usually is irrelevant. The problem is considered solved when the image is determined to a scale factor c, as done here;
2. *Translation:* If $\{h_{i_1,i_2}, x_{i_1,i_2}\}$ is a solution, then $\{h_{i_1+d_1,i_2+d_2}, x_{i_1-d_1,i_2-d_2}\}$ is also a solution for any constants (d_1, d_2). This ambiguity is eliminated by specifying supports of h_{i_1,i_2} and x_{i_1,i_2};
3. *Exchange:* We need to be able to distinguish h_{i_1,i_2} from x_{i_1,i_2}. This ambiguity is eliminated since x_{i_1,i_2} has non-square $(M_1 \times M_2)$ support, while h_{i_1,i_2} has square $(L \times L)$ support.

2 1-D and 2-D Convolutions

2.1 1-D Convolution

2.1.1 Definition of 1-D Convolution

The 1-D convolution of a sequence h_n of length L with a sequence x_n of length M results in a sequence y_n of length $L + M - 1$, where (assuming $L \le M$)

$$y_n = \sum_{i=0}^{L-1} h_i x_{n-i}, \quad 0 \le n \le L + M - 2. \tag{2}$$

For example,

$$\{1, 2, 3\} * \{4, 5, 6\} = \{4, 13, 28, 27, 18\}. \tag{3}$$

2.1.2 Toeplitz Matrix times Vector Forms

This can be written in matrix form as the "tall" form

$$\begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 2 & 1 \\ 0 & 3 & 2 \\ 0 & 0 & 3 \end{bmatrix} \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix} = \begin{bmatrix} 4 \\ 13 \\ 28 \\ 27 \\ 18 \end{bmatrix} \tag{4}$$

or as the "reclining" form

$$[4, 13, 28, 27, 18] = [4, 5, 6] \begin{bmatrix} 1 & 2 & 3 & 0 & 0 \\ 0 & 1 & 2 & 3 & 0 \\ 0 & 0 & 1 & 2 & 3 \end{bmatrix}. \tag{5}$$

The "reclining" is the transpose of the "tall" form.

2.1.3 Product of Toeplitz Matrices Forms

The "tall" form can be extended to the product of Toeplitz matrices

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 & 0 \\ 3 & 2 & 1 & 0 & 0 \\ 0 & 3 & 2 & 1 & 0 \\ 0 & 0 & 3 & 2 & 1 \\ 0 & 0 & 0 & 3 & 2 \\ 0 & 0 & 0 & 0 & 3 \end{bmatrix} \begin{bmatrix} 4 & 0 & 0 \\ 5 & 4 & 0 \\ 6 & 5 & 4 \\ 0 & 6 & 5 \\ 0 & 0 & 6 \end{bmatrix} = \begin{bmatrix} 4 & 0 & 0 \\ 13 & 4 & 0 \\ 28 & 13 & 4 \\ 27 & 28 & 13 \\ 18 & 27 & 28 \\ 0 & 18 & 27 \\ 0 & 0 & 18 \end{bmatrix}. \tag{6}$$

The "reclining" form can be extended similarly to the transpose of the "tall" form.

2.1.4 z-transforms

The z-transform $X(z)$ of x_n is defined as

$$X(z) = \sum_{n=0}^{M-1} x_n z^{-n}. \tag{7}$$

The 1-D convolution (2) becomes the product

$$Y(z) = H(z)X(z). \tag{8}$$

2.2 2-D Convolution

2.2.1 Definition of 2-D Convolution

The 2-D convolution y_{i_1,i_2}=$h_{i_1,i_2} * * x_{i_1,i_2}$ of $(L \times L)$ PSF $\{h_{i_1,i_2}, 0 \le i_1, i_2 \le L-1\}$ with $(M_1 \times M_2)$ image $\{x_{i_1,i_2}, 0 \le i_1 \le M_1 - 1, 0 \le i_2 \le M_2 - 1\}$ is $((L+M_1-1) \times (L+M_2-1))$ $\{y_{i_1,i_2}, \begin{smallmatrix} 0 \le i_1 \le L+M_1-2 \\ 0 \le i_2 \le L+M_2-2 \end{smallmatrix}\}$, where (assuming $L \le M_1, M_2$)

$$y_{i_1,i_2} = \sum_{n_1=0}^{L-1} \sum_{n_2=0}^{L-1} h_{n_1,n_2} x_{i_1-n_1,i_2-n_2}. \tag{9}$$

A 2-D convolution can be viewed as nested 1-D convolutions. For example,

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} * * \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 5 & 16 & 12 \\ 22 & 60 & 40 \\ 21 & 52 & 32 \end{bmatrix}. \tag{10}$$

2.2.2 2-D z-transforms

Defining the 2-D z-transform

$$X(z_1, z_2) = \sum_{i_1=0}^{M_1-1} \sum_{i_2=0}^{M_2-1} x_{i_1,i_2} z_1^{-i_1} z_2^{-i_2}, \tag{11}$$

the 2-D convolution (9) becomes

$$Y(z_1, z_2) = H(z_1, z_2)X(z_1, z_2). \tag{12}$$

Setting $z_1 = z_2^{L+M_2-2}$ maps the 2-D convolution (9) to a 1-D convolution. For (10),

$$[1,2,0,3,4] * [5,6,0,7,8] = \tag{13}$$

$$[5,16,12,22,60,40,21,52,32].$$

This explains the Toeplitz-block-Toeplitz form below.

For completeness, there is a fourth ambiguity of 2-D blind deconvolution. We must assume

$$\begin{aligned} H(z_1, z_2) &= \sum_{i_1=0}^{L-1} \sum_{i_2=0}^{L-1} h_{i_1,i_2} z_1^{-i_1} z_2^{-i_2} \\ X(z_1, z_2) &= \sum_{i_1=0}^{M_1-1} \sum_{i_2=0}^{M_2-1} x_{i_1,i_2} z_1^{-i_1} z_2^{-i_2} \end{aligned} \tag{14}$$

are irreducible (i.e., they cannot be factored). This is almost surely true in practice. To see this, (9) is

$(L+M_1-1)(L+M_2-1)$ simultaneous quadratic equations

in $L^2 + M_1 M_2 < (L+M_1-1)(L+M_2-1)$ unknowns.

Since the problem is overdetermined, by Bezout's theorem there is almost surely at most one solution. In fact, there are generically *no* solutions; only from (9) do we know that even one solution exists.

2.2.3 TBT Matrix times Vector Forms

(10) can be written in matrix form as the "tall" form

$$\left[\begin{array}{ccc|cc} 1 & 0 & & 0 & 0 \\ 2 & 1 & & 0 & 0 \\ 0 & 2 & & 0 & 0 \\ - & - & - & - & - \\ 3 & 0 & & 1 & 0 \\ 4 & 3 & & 2 & 1 \\ 0 & 4 & & 0 & 2 \\ - & - & - & - & - \\ 0 & 0 & & 3 & 0 \\ 0 & 0 & & 4 & 3 \\ 0 & 0 & & 0 & 4 \end{array}\right] \begin{bmatrix} 5 \\ 6 \\ 7 \\ 8 \end{bmatrix} = \begin{bmatrix} 5 \\ 16 \\ 12 \\ 22 \\ 60 \\ 40 \\ 21 \\ 52 \\ 32 \end{bmatrix} \tag{15}$$

or as "reclining" form (transpose of the "tall" form)

$$[5,16,12,22,60,40,21,52,32] = \tag{16}$$

$$[5,6,7,8] \left[\begin{array}{ccc|ccc|ccc} 1 & 2 & 0 & 3 & 4 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 0 & 3 & 4 & 0 & 0 & 0 \\ - & - & - & - & - & - & - & - & - \\ 0 & 0 & 0 & 1 & 2 & 0 & 3 & 4 & 0 \\ 0 & 0 & 0 & 0 & 1 & 2 & 0 & 3 & 4 \end{array}\right].$$

In this paper, it will be useful to use the "reclining" form for individual blocks and "tall" form for the arrangement of the blocks. For example,

$$\begin{bmatrix} 5 & 16 & 12 & 0 & 0 & 0 \\ 22 & 60 & 40 & 5 & 16 & 12 \\ 21 & 52 & 32 & 22 & 60 & 40 \\ 0 & 0 & 0 & 21 & 52 & 32 \end{bmatrix} = \tag{17}$$

$$\begin{bmatrix} 5 & 6 & 0 & 0 & 0 & 0 \\ 7 & 8 & 5 & 6 & 0 & 0 \\ 0 & 0 & 7 & 8 & 5 & 6 \\ 0 & 0 & 0 & 0 & 7 & 8 \end{bmatrix} \begin{bmatrix} 1 & 2 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 0 & 0 & 0 \\ 3 & 4 & 0 & 1 & 2 & 0 \\ 0 & 3 & 4 & 0 & 1 & 2 \\ 0 & 0 & 0 & 3 & 4 & 0 \\ 0 & 0 & 0 & 0 & 3 & 4 \end{bmatrix}.$$

3 TBT Matrix Formulation for 2-D Blind Deconvolution

3.1 Introduction

We can write the 2-D blind deconvolution problem as

$$\mathbf{Y} = \mathbf{X}\mathbf{H} \tag{18}$$

where $\mathbf{Y}$, $\mathbf{X}$ and $\mathbf{H}$ are TBT matrices. We define

$$\mathbf{G} = \mathbf{H}^{-1}. \tag{19}$$

$\mathbf{H}$ is a square matrix by construction (see below) and we assume $\mathbf{H}$ is invertible and $\mathbf{X}$ has full rank.

Next, we combine (18) and (19) into

$$[\mathbf{Y} \quad -\tilde{\mathbf{I}}] \begin{bmatrix} \mathbf{G_n} \\ \mathbf{X_n} \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix}. \tag{20}$$

$\mathbf{G_n}$ and $\mathbf{X_n}$ are $(M_2+1)^{th}$ columns of $\mathbf{G}$ and $\mathbf{X}$ and

$$\tilde{\mathbf{I}} = \begin{bmatrix} 0 & 0 & 0 & 0 & I_{M_2} \\ 0 & 0 & I_{M_2} & 0 & 0 \\ I_{M_2} & 0 & 0 & 0 & 0 \end{bmatrix}^{\mathbf{T}} \tag{21}$$

picks off the nonzero values of $\mathbf{X_n}$.

3.2 Consistency of Block Sizes

The sizes of each block, in reclining form:

$$(k_2+1)\{\underbrace{\boxed{\mathbf{Y}}}_{L+k_1} = (k_2+1)\{\underbrace{\boxed{\mathbf{X}}}_{M_2+k_2} (k_1+1)\{\underbrace{\boxed{\mathbf{H}}}_{L+k_1} \tag{22}$$

for k_1 and k_2 so that #columns of $\mathbf{X}$=#rows of $\mathbf{H}$:

$$k_1 + 1 = M_2 + k_2. \tag{23}$$

Note $\mathbf{Y}$ is $(k_2+1) \times (k_2+M_2+L-1)$ as expected.

The sizes of the blocks, in tall form, for k_3 and k_4:

$$(M_1+k_4)\{\underbrace{\boxed{\mathbf{Y}}}_{k_3+1} = (M_1+k_4)\{\underbrace{\boxed{\mathbf{X}}}_{k_4+1} (L+k_3)\{\underbrace{\boxed{\mathbf{H}}}_{k_3+1} \tag{24}$$

so #block columns of $\mathbf{X}$=#block rows of $\mathbf{H}$:

$$k_4 + 1 = L + k_3. \tag{25}$$

Note $\mathbf{Y}$ is $(k_3+M_1+L-1) \times (k_3+1)$ as expected.

We also need $\mathbf{H}$ to be square, so that $\mathbf{G} = \mathbf{H}^{-1}$ is defined. This requires

$$(L+k_3)(k_1+1) = (L+k_1)(k_3+1) \tag{26}$$

which simplifies to

$$k_3 = k_1. \tag{27}$$

We need $\mathbf{Y}$ to be tall, with vertical size exceeding horizontal size by (M_1M_2) so $[\mathbf{X} \quad -\tilde{\mathbf{I}}]$ is square:

$$(M_1+k_4)(k_2+1) = (L+k_1)(k_3+1)+(M_1M_2). \tag{28}$$

Substituting (23), (25), and (27) into (28) gives

$$(M_1+L-1+k_1)(2+k_1-M_2) = (L+k_1)(k_1+1)+(M_1M_2). \tag{29}$$

Solving (29) for k_1 gives

$$k_1 = \frac{(2M_1+L-1)(M_2-1)+1}{M_1-M_2} \tag{30}$$

and k_2, k_3, k_4 from (23), (25), and (27). The denominator of (30) shows why we require $M_1 > M_2$.

4 Revised Procedure

We can reduce the size of the matrix $[\mathbf{Y} \quad -\mathbf{I}]$ by $M_1 M_2$ as follows. Instead of augmenting $[\mathbf{Y}]$ by $[-\tilde{\mathbf{I}}]$ to account for the unknown image pixels in $\mathbf{X_n}$, we may delete the rows of $[\mathbf{Y}]$ corresponding to those values, and then reconstruct $\mathbf{X_n}$ by multiplying the deleted rows of $[\mathbf{Y}]$ and $\mathbf{G_n}$. This loses the advantage of reconstructing $\mathbf{X_n}$ directly, but requires solution of a smaller system of equations $[\tilde{\mathbf{Y}}]\mathbf{G_n} = \mathbf{0}$ where $[\tilde{\mathbf{Y}}]$ is $[\mathbf{Y}]$ with the rows corresponding to nonzero elements of $\mathbf{X_n}$ deleted.

A much greater improvement in required computation is possible by allowing the PSF to be nonsquare. Such a PSF can arise from a combination of motion blur and a square PSF. This also allows the image to be square. The same program can be used, with the roles of the image and the PSF exhanged.

5 Matlab Programs

Appendix A is an illustrative Matlab program that generates $\mathbf{Y}$, $\mathbf{X}$, and $\mathbf{H}$ in (18) and checks $\mathbf{Y}$=$\mathbf{XH}$.

Appendix B is an illustrative Matlab program that finds the null vector of $[\mathbf{Y} \quad -\tilde{\mathbf{I}}]$ in (20) and scales the null vector to match the original image.

In order to use a conjugate gradient type algorithm, instead of the SVD, a fast way of computing the TBT matrix $[\mathbf{Y} \quad -\tilde{\mathbf{I}}]\times$vector product is needed. This has the advantage that the TBT matrix need not be constructed and stored.

Appendix D computes this product using zero-padding and the 2-D FFT to compute a 2-D convoluton. We were unable to get a conjugate-gradient-type algorithm working, perhaps due to the conditioning of the TBT matrix $[\mathbf{Y} \quad -\tilde{\mathbf{I}}]$, but the reader may succeed with a preconditioner.

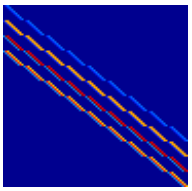
If the "reclining form" Toeplitz matrix is postmul tiplied by a column vector, the result is a "valid' convolution of length $(M - L + 1)$. The valid convo lution is the usual (linear) convolution with the firs and last $(L - 1)$ elements deleted.

Appendices C and E implement two examples.

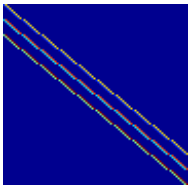
6 Tiny Example of Matrices

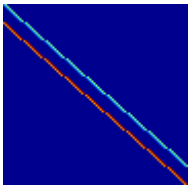
For a (3×2) image and (2×2) PSF the sizes of the matrices: $\mathbf{Y}$: (96×90); $\mathbf{X}$: (96×90); $\mathbf{H}$: (90×90).

Examination of the matrix $\mathbf{X}$ in the example below shows that the $(M_2 + 1)^{th}$ column of $\mathbf{X}$ has zeros to its left and right, and the values of $x_{i,j}$ contained in it are bounded by zeros above and below. So this column with zero constraints uniquely specifies $x_{i,j}$.



=

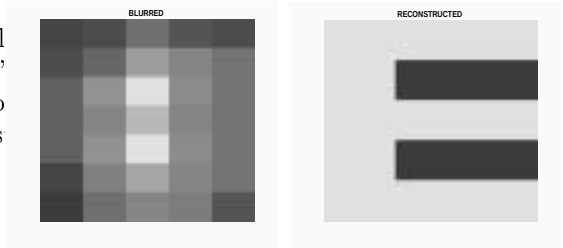




7 Tiny Example

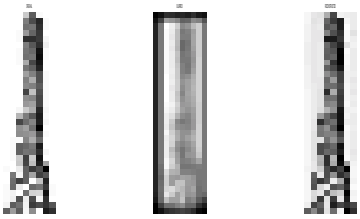
We applied the method to blind deconvolution of a 5×3 letter 'E' blurred by a 3×3 PSF consisting of the digits of π, resulting in a 7×5 blurred image. The matrix $[\mathbf{Y} \quad -\tilde{\mathbf{I}}]$ was (240×239) despite the tiny size of this problem.

The blurred and reconstructed images are shown. The reconstruction has maximum error of .0002%. The Matlab program is given in Appendix B.



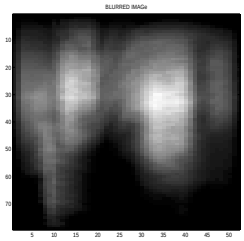
8 Larger Example

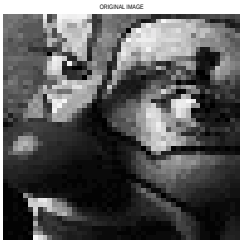
We applied the method to blind deconvolution of a (34×8) thumbnail image of the top of the Eiffel tower in Paris blurred by a (3×3) PSF consisting of the digits of π, resulting in a (36×10) blurred image. The matrix $[\mathbf{Y} \quad -\tilde{\mathbf{I}}]$ was (715×712). The reconstruction has maximum error of .0001%.

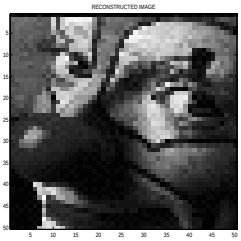


9 Still Larger Example

We apply the revised procedure to blind deconvolution of an unknown (50×50) image and an unknown (30×3) PSF of random values. While the problem is much larger than the previous examples, the matrix $\tilde{Y}$ is only (614×590), much smaller.







10 Appendix A

```
%Demo of YY=XX*HH.
  clear;%Need M1>M2 (tall X) and H square.
  X=[1 2 3;4 5 6]';H=[1 2;3 4];
  Y=conv2(X,H);[M1,M2]=size(X);L=size(H,1);[N1,N2]=size(Y);
  K1=ceil(((M1+M1+L-1)*(M2-1)+1)/(M1-M2));
  K2=K1+1-M2;K4=L-1+K1;
  YY1size=(M1+K4)*(K2+1);YY2size=(L+K1)*(K1+1);
  for I=1:N1;TTY(:,:,I)=toeplitz([Y(I,1);zeros(K2,1)],...
  [Y(I,:)  zeros(1,K2)]);end;
  for I=1:M1;TTX(:,:,I)=toeplitz([X(I,1);zeros(K2,1)],...
  [X(I,:)  zeros(1,K2)]);end;
  for I=1:L;TTH(:,:,I)=toeplitz([H(I,1);zeros(K1,1)],...
  [H(I,:)  zeros(1,K1)]);end;
  for J=0:N1-1;for I=0:M1+K4-N1;
  IY1=(K2+1)*I+1+J*(K2+1):(K2+1)*(I+1)+J*(K2+1);
  IY2=(N2+K2)*I+1:(N2+K2)*(I+1);YY(IY1,IY2)=TTY(:,:,J+1);
  end;end;for J=0:M1-1;for I=0:M1+K4-M1;
  IX1=(K2+1)*I+1+J*(K2+1):(K2+1)*(I+1)+J*(K2+1);
  IX2=(M2+K2)*I+1:(M2+K2)*(I+1);XX(IX1,IX2)=TTX(:,:,J+1);
  end;end;for J=0:L-1;for I=0:L+K1-L;
  IH1=(K1+1)*I+1+J*(K1+1):(K1+1)*(I+1)+J*(K1+1);
  IH2=(L+K1)*I+1:(L+K1)*(I+1);HH(IH1,IH2)=TTH(:,:,J+1);
  end;end;max(max(YY-XX*HH))
```

11 Appendix B

```
%2-D Blind deconvolution by finding the null vector
  %of a Toeplitz-Block-Toeplitz matrix
  %Need M1>M2 (tall X) and H square.
  %Assume upper left corner of image is unity
  clear;%to fix scale factor.  A tiny example:
  X(1,1:3)=1;X(3,1:3)=1;X(5,1:3)=1;X(1:5,1)=1;%"E"
  H=[3 1 4;1 5 9;2 6 5];%[YY Z] is 240X239.
  Y=conv2(X,H);[M1,M2]=size(X);L=size(H,1);[N1,N2]=size(Y);
  K1=ceil(((M1+M1+L-1)*(M2-1)+1)/(M1-M2));%round up.
  K2=K1+1-M2;K4=L-1+K1;
  YY1size=(M1+K4)*(K2+1);YY2size=(L+K1)*(K1+1);
  for I=1:N1;TT(:,:,I)=toeplitz([Y(I,1);zeros(K2,1)],...
  [Y(I,:)  zeros(1,K2)]);end;
  %Delete rows & columns of Toeplitz to make TBT:
  for J=0:N1-1;for I=0:M1+K4-N1;
  IY1=(K2+1)*I+1+J*(K2+1):(K2+1)*(I+1)+J*(K2+1);
  IY2=(N2+K2)*I+1:(N2+K2)*(I+1);
  YY(IY1,IY2)=TT(:,:,J+1);end;end;
  %Use (M2+1)th column of YY*GG=XX (GG = HH^-1)
  Z=zeros(YY1size,M1*M2);for I=0:M1-1;
  Z([1:M2]+(K2+1)*I+(M2+1),[1:M2]+M2*I)=-eye(M2);end;
  %Compute null vector of matrix using QR algorithm:
  ZZ=[YY Z];[Q R]=qr(ZZ');K=size(Q,1);
  XHAT1=Q(K-M1*M2+1:K,K);XHAT1=XHAT1/XHAT1(M2);
  XHAT=reshape(XHAT1,M2,M1);XHAT=(flipud(XHAT))';[XHAT,X]
  figure,imagesc(Y),title('BLURRED'),axis off
  figure,imagesc(XHAT),title('RECONSTRUCTED'),axis off
```

12 Appendix C

```
clear;%Need M1>M2 (tall X) and H square.
  X1=imread('eiffel.jpg');X2=X1(70:400,260:330);
  X=X2(1:10:331,1:10:71);%Downsample by 10X10 w/o LPF
  H=[3 1 4;1 5 9;2 6 5];Y=conv2(X,H);
  figure,imagesc(X),colormap(gray),title('ORIGINAL')
  figure,imagesc(Y),colormap(gray),title('BLURRED')
  [M1,M2]=size(X);L=size(H,1);[N1,N2]=size(Y);
  K1=ceil(((M1+M1+L-1)*(M2-1)+1)/(M1-M2));
  K2=K1+1-M2;K4=L-1+K1;
  YY1size=(M1+K4)*(K2+1);YY2size=(L+K1)*(K1+1);
  for I=1:N1;TT(:,:,I)=toeplitz([Y(I,1);zeros(K2,1)],...
  [Y(I,:)  zeros(1,K2)]);end;
  %Delete rows & columns of Toeplitz to make TBT:
  for J=0:N1-1;for I=0:M1+K4-N1;
  IY1=(K2+1)*I+1+J*(K2+1):(K2+1)*(I+1)+J*(K2+1);
  IY2=(N2+K2)*I+1:(N2+K2)*(I+1);
  YY(IY1,IY2)=TT(:,:,J+1);end;end;
  Z=zeros(YY1size,M1*M2);
  for I=0:M1-1;Z([1:M2]+(K2+1)*I+2,[1:M2]+M2*I)=-eye(M2);
  end;ZZ=[YY Z];%Use SVD to compute null vector:
  [UZ,SZ,VZ]=svd(ZZ);SZ1=diag(SZ);LZ=length(SZ1);
  XHAT1=VZ(LZ-M1*M2+1:LZ,LZ);
  XHAT1=XHAT1/XHAT1(M2)*X(1);
  XHAT=reshape(XHAT1,M2,M1);XHAT=(flipud(XHAT))';
  figure,imagesc(XHAT),colormap(gray),title('RECONSTRUCTED')
```

13 Appendix D

```
clear;X=[1 2 3;4 5 6]';H=[1 2;3 4];
  %set up TBT matrix to confirm the 2-D FFT result.
  Y=conv2(X,H);[M1,M2]=size(X);L=size(H,1);[N1,N2]=size(Y);
  K1=ceil(((M1+M1+L-1)*(M2-1)+1)/(M1-M2));
  K2=K1+1-M2;K4=L-1+K1;
  YY1size=(M1+K4)*(K2+1);YY2size=(L+K1)*(K1+1);
  for I=1:N1;TT(:,:,I)=toeplitz([Y(I,1);zeros(K2,1)],...
  [Y(I,:)  zeros(1,K2)]);end;
  %Delete rows & columns of Toeplitz to make TBT:
  for J=0:N1-1;for I=0:M1+K4-N1;
  IY1=(K2+1)*I+1+J*(K2+1):(K2+1)*(I+1)+J*(K2+1);
  IY2=(N2+K2)*I+1:(N2+K2)*(I+1);YY(IY1,IY2)=TT(:,:,J+1);
  end;end;Z=zeros(YY1size,M1*M2);
  for I=0:M1-1;Z([1:M2]+(K2+1)*I+1,[1:M2]+M2*I)=-eye(M2);end;
  ZZ=[YY Z];%Now use the 2-F FFT and compare results:
  W1=[1:(L+K1)*(K1+1)]';W2=reshape(W1,L+K1,K1+1);W2=W2';
  D1=(L+K1)-(K2+1);D2=N2+L+K1-1;D3=N1+K1;n=nextpow2(max(D2,D3));
  FY=fft2(fliplr(Y),2^n,2^n);FW=fft2(W2,2^n,2^n);
  FQ=FY.*FW;Q=round(real(ifft2(FQ)));Q=Q';
  Q4=Q((D1+1):(D2-D1),1:D3);Q5=Q4(:);X1=X(:);
  for I=0:M1-1;Q5([1:M2]+(K2+1)*I+1)=Q5([1:M2]+(K2+1)*I+1)-...
  X1([1:M2]+(M2*I));end;[ZZ*[W1;X(:)],Q5]%[YY*W1,Q5]
```

14 Appendix E

```
%First 4 and last 5 lines for example.
  %X is PSF and H is image (for example).
  clear;
  load clown.mat;%200X200. Downsample to 50X50.
  FX=fft2(X);FXD=FX;FXD(25:202-25,25:202-25)=0;
  XD=real(ifft2(FXD));H=XD(1:4:end,1:4:end);
  X=rand(30,3);Y=conv2(X,H);%Y is 79X52.
  [M1,M2]=size(X);L=size(H,1);
  K1=ceil(((M1+M1+L-1)*(M2-1)+1)/(M1-M2));
  K2=K1+1-M2;K4=L-1+K1;
  YY1size=(M1+K4)*(K2+1);YY2size=(L+K1)*(K1+1);
  for I=1:(M1+L-1);TT(:,:,I)=toeplitz([Y(I,1);zeros(K2,1)],...
  [Y(I,:) zeros(1,K2)]);end;
  YY=zeros(YY1size,YY2size);
  %Delete rows and columns of Toeplitz to make TBT:
  for J=0:M1+L-2;for I=0:K4-L+1;
  IY1=(K2+1)*I+1+J*(K2+1):(K2+1)*(I+1)+J*(K2+1);
  IY2=(K2+M2+L-1)*I+1:(K2+M2+L-1)*(I+1);
  YY(IY1,IY2)=TT(:,:,J+1);end;end;
  %Delete rows of YY corresponding to nonzero rows of XX.
  ZZ=YY;ZZ1=[];I1=[];
  for I=0:M1-1;I1=[I1 ([1:M2]+(K2+1)*I+2)];end;
  ZZ(I1,:)=[];ZZ1(1:length(I1),:)=YY(I1,:);
  [U,S,V]=svd(ZZ);S1=diag(S);L1=length(S1);
  XHAT1=ZZ1*V(:,L1);
  XHAT1=XHAT1/XHAT1(M2)*X(1);%fix scale factor
  XHAT=reshape(XHAT1,M2,M1);XHAT=(flipud(XHAT))';
  %XHAT,X,size(ZZ),S1(L1-1),S1(L1)
  %Deconvolve image from known Y ans estimated PSF XHAT.
  FXHAT=fft2(XHAT,80,54);FY=fft2(Y,80,54);FHHAT=FY./FXHAT;
  HHAT=real(ifft2(FHHAT));HHAT=HHAT(1:50,1:50);
  figure,imagesc(H),colormap(gray),title('ORIGINAL IMAGe')
  figure,imagesc(Y),colormap(gray),title('BLURRED IMAGe')
  figure,imagesc(HHAT),colormap(gray),...
  title('RECONSTRUCTED IMAGE')
```