# Blind Deconvolution and Toeplitzation Using Iterative Null-Space and Rank-One Projections

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

*Abstract*— **Toeplitzation requires finding a singular Toeplitz matrix near to a given one; blind deconvolution requires finding unknown image and point-spread functions (PSF) whose convolution is known. Both problems are commonly solved using iterative algorithms that require solving a linear system of equations or computing a minimum singular vector at each iteration (Fourier transforms can only be used for blind deconvolution if the image has compact support). We present a new reformulation of these problems in which these are replaced by computing the rank-one matrix nearest to a given matrix; this is easily done using the power method. Numerical examples and Matlab codes illustrate the two new algorithms.**

*Keywords*— **Blind deconvolution, Toeplitzation**
**Phone: 734-763-9810. Fax: 734-763-1503.**
**Email: aey@eecs.umich.edu.** **EDICS: 2-REST.**

## I. INTRODUCTION

### A. Toeplitzation

The Toeplitzation problem is as follows: Given a non-Hermitian Toeplitz matrix (constant along its diagonals), find a non-Hermitian Toeplitz matrix near in Frobenius norm (sum of squares of elements) to the given matrix, and is singular (rank-deficient).

This problem has applications in:

- Target direction finding in noise;
- Spectral line enhancement in noise;
- Blind deconvolution of symmetric PSFs.

The Toeplitz matrix would be singular in the absence of noise, but noise makes it non-singular. Hence perturbing the elements to the nearest singular Toeplitz matrix greatly reduces the effects of noise.

The usual procedure is to apply an iterative algorithm that at each iteration alternately:

- Computes the nearest (in Frobenius norm) singular matrix to the given matrix by computing the singular vectors associated with the minimum singular value, and subtracting the outer product of these three;
- Computes the nearest (in Frobenius norm) Toeplitz matrix by averaging along its diagonals.

The first step can be computationally difficult. The inverse power method is commonly used, but this:

- Requires solution of a linear system of equations;
- more ill-conditioned as the method converges
- slowly if there are several small singular values.

The algorithm presented in this paper requires, at each iteration, a step analogous to averaging along diagonals (but completely different) and computation of the singular vectors associated with the maximum singular value. These latter can be computed using the (direct) power method, which:

- Requires only matrix-vector multiplication;
- Has no conditioning issues, and converges quickly
- since there is only one large singular value.

### B. Blind Deconvolution

The blind deconvolution problem is to solve

- $y_{i,j}$=$\sum_{m=0}^{L-1}\sum_{n=0}^{L-1} h_{m,n}x_{i-m,j-n}$
- $x_{i,j}$=unknown M×M image;
- $x_{i,j}$=unknown L×L PSF;
- $y_{i,j}$=**known** N×N blurred image.

It has many applications in optics and astronomy.

The usual procedure is to apply an iterative algorithm that at each iteration alternately:

- Deconvolves an estimated image $x_{i,j}$ from the given data $y_{i,j}$ and the present estimate of PSF $h_{i,j}$;
- Support constraint: set appropriate $x_{i,j}$=0;
- Deconvolves an estimated PSF $h_{i,j}$ from the given data $y_{i,j}$ and the present estimate of image $x_{i,j}$;
- Support constraint: set appropriate $h_{i,j}$=0.

If the image has known *compact* support ($x_{i,j}$=0 except in a known region) and all nonzero $y_{i,j}$ are known, then Fourier transforms can be used for the deconvolutions. But in many applications not all nonzero $y_{i,j}$ are known (only the *valid* convolution) in which case each deconvolution requires solution of a linear system of equations (if N<L+M–1 above).

The algorithm presented in this paper replaces the deconvolutions (which are generally ill-posed) with computation of maximum singular vectors using the power method, which usually converges quickly.

## II. REFORMULATION

### A. Toeplitzation

The basic idea is to recognize that the N×N Toeplitz matrix with $(i,j)^{th}$ element $x_{i,j}$=$x_{i-j}$ is singular if and only if there exists some nonzero $\{a_i, i=0\ldots N-1\}$ such that $\sum_{i=0}^{N-1} a_i x_{n-i}$=0, i.e.,

$$\begin{bmatrix} x_0 & x_1 & \cdots & x_{N-1} \\ x_{-1} & x_0 & \ddots & x_{N-2} \\ \ddots & \ddots & \ddots & \ddots \\ x_{1-N} & \cdots & \cdots & x_0 \end{bmatrix} \begin{bmatrix} a_{N-1} \\ a_{N-2} \\ \vdots \\ a_0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad (1)$$

This can now be rewritten as an N×(2N–1)N heavily underdetermined linear system of equations having unknowns the Kronecker product of $\vec{a}$ and $\vec{x}$:

$$\vec{a} \times \vec{x} = [a_0 x_{1-N} \ldots a_0 x_{N-1}, a_1 x_{1-N} \ldots a_{N-1} x_{N-1}]$$

For example, for N=3 we can rewrite the system

$$\begin{bmatrix} x_0 & x_1 & x_2 \\ x_{-1} & x_0 & x_1 \\ x_{-2} & x_{-1} & x_0 \end{bmatrix} \begin{bmatrix} a_2 \\ a_1 \\ a_0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad (2)$$

as the sparse 3×(2(3)–1)(3)=3×15 linear system

$$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \overrightarrow{ax}$$

$$\overrightarrow{ax} = [a_0 x_{-2} \ldots a_0 x_2, a_1 x_{-2} \ldots a_1 x_2, a_2 x_{-2} \ldots a_2 x_2]$$

This suggests the following iterative algorithm:

- **Initialize:** $\{a_n\}$=a random N-vector;
- $\{x_n\}$=the given (noisy) matrix entries.
- **At each iteration:**
- Project $\overrightarrow{ax}$ onto the nullspace of matrix T;
- Project $\overrightarrow{ax}$ onto the set of rank=1 matrices.

The first projection is given by

$$\overrightarrow{ax} \leftarrow (I - T^T(TT^T)^{-1}T)\overrightarrow{ax} \quad (3)$$

where $T$ is the N×(2N–1)N system matrix above. Since $TT^T$=$\frac{I}{N}$, it is easily seen that this projection subtracts from each element of $\overrightarrow{ax}$ accessed in a row of $T$ the average of those elements. Then $T\overrightarrow{ax}$=0.

The second projection is given by

$$u_1 \sigma_1 v_1^H \leftarrow AX = \sum_{i=1}^{N} u_i \sigma_i v_i^H \quad (4)$$

where $AX_{i,j}$=$\overrightarrow{ax}_{i+N(j-1)}$ is the rearrangement of $\overrightarrow{ax}$ into successive columns of an N×N matrix AX.

The maximum singular vectors $u_1$ and $v_1$ can be computed iteratively using the power method:

$$u_1 \leftarrow \frac{(AX)(AX)^H u_1}{||(AX)(AX)^H u_1||}; \quad v_1 = (AX)^H u_1 \quad (5)$$

This converges quickly if $\sigma_1 >> \sigma_2$, as will be the case after the first few iterations. Note each iteration requires only two matrix-vector multiplications.

### B. Blind Deconvolution

The reformulation for blind deconvolution follows the same plan as above. The main differences are:

- The underdetermined matrix T is harder to form;
- T depends on which values of the image $x_{i,j}$ are known to be zero (the support constraint geometry);
- But matrix T does **not** depend on the data $y_{i,j}$.

Once T is obtained, the only difference from the Toeplitzation algorithm is that the first projection now includes the data $-y_{i,j}$ in the values of $\overrightarrow{ax}$ to be averaged, so that now $T\overrightarrow{ax}$=y instead of zero.

Since the PSF is much smaller than the image, the matrix $(AX)^H$ is "tall" and "thin." Hence it may be faster to compute directly the maximum eigenvector of the small matrix $(AX)(AX)^H$, instead of using the power method to compute it.

## III. NUMERICAL RESULTS

### A. Toeplitzation

The Matlab program listed below implements the Toeplitzation algorithm described above. Note that:

- The maximum singular vectors are all computed directly, rather than using the power method;
- The singular Toeplitz matrix is generated from the sum of several sinusoids with random amplitudes, so the matrix is symmetric and non-circulant;
- Random Gaussian noises are added to the matrix elements, and to the initialization.

The base-10 logarithm of the minimum singular value is plotted vs. iteration number for a typical run in Figure #1. Note the following from Figure #1:

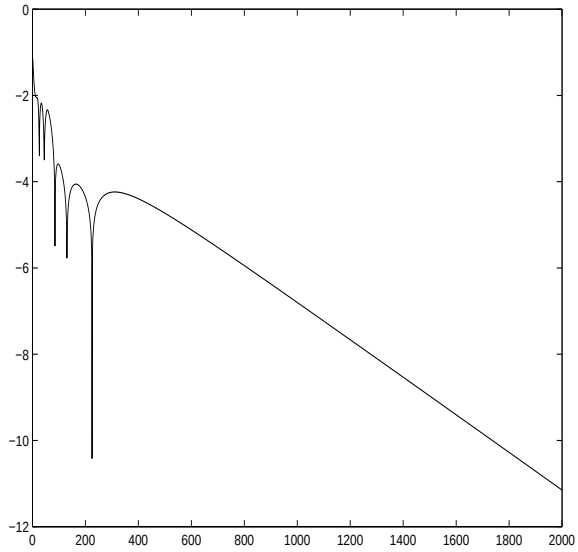
- The trend is the minimum singular value decreases exponentially with iteration number;
- Occasional sharp dips and rises occur in this plot;
- The singular values show that $\sigma_{32} << \sigma_{31}, \sigma_1$.

The singular values for this run were as follows:

| SINGULAR VALUE: | $\sigma_1$ | $\sigma_{31}$ | $\sigma_{32}$ |
|---|---|---|---|
| INITIALIZATION: | 32. | 1.1 | 0.94 |
| 2000 ITERATIONS: | 3.0 | $10^{-3}$ | $7 \times 10^{-12}$ |

Matlab code used to generate this example:

```
clear;N=32;%Must be an even number
C=ones(1,N);for J=1:N/2-1;
C=C+rand*cos(2*pi*J*[0:N-1]/(N+0.2));end
R=C;A=toeplitz(C,R);X=null(A);
X=X+0.2*randn(N,1);R=R+0.2*randn(1,N);C=R;
ANOISY=toeplitz(C,R);S=svd(ANOISY);
RC=[fliplr(C) R(2:end)];RCX=X*RC;Y=RCX(:);
for J=1:2000;for I=1:N;NI=N*(N-I);%J=iteration
II=[NI+1:N+1:NI+N*N];W(II)=sum(Y(II));
end;Y=Y-W'/N;YY=reshape(Y,N,2*N-1);
[U S V]=svd(YY);Y=U(:,1)*V(:,1)';Y=Y(:);
AEST=toeplitz(flipud(V(1:N,1)),V([N:2*N-1],1));
S=svd(AEST);SS(J)=S(N);end;plot(log10(SS))
```

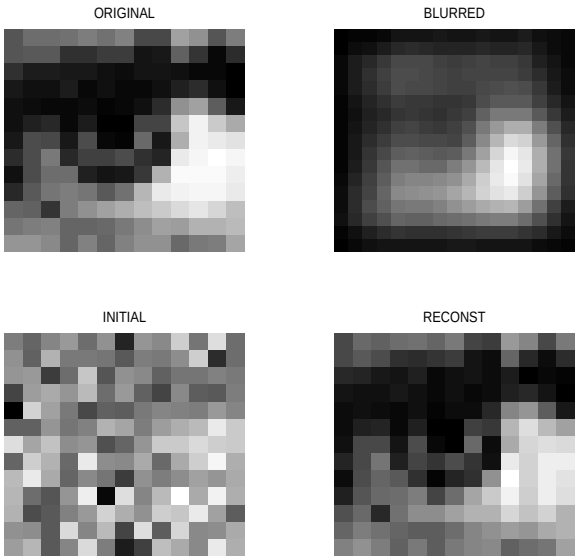



*B. Blind Deconvolution*

The Matlab program listed below implements the blind deconvolution algorithm described above, but:

- The maximum singular vectors are computed from eigenvectors of a small matrix, rather than using the power method, since this is much faster (see above);
- Initializations are the actual image and PSF with extremely large amounts of noise added to each one;
- A tiny image is used to make the runtime short.

Results of a typical run are shown in Figures #2-3:




- The unknown image is 20×20;
- The unknown PSF is 5×5;
- The complete 24×24 data is used;
- The program can be modified to incorporate non-compact support, but this is problem-dependent;
- Both the unknown image and PSF are successfully reconstructed, to an (unavoidable) scale factor.

Matlab code used to generate this example:

```
clear;load lena.mat;X=xx(128:140,128:140);%Eye image
H=[1 4 1 5 9;2 6 5 3 5;8 9 7 9 3;2 3 8 4 6;2 6 4 3 3];
Y=conv2(X,H);M=size(X,1);L=size(H,1);%Blur image
X1=X+100*randn(M,M);H1=H+3*randn(L,L);%Initial
XH=X1(:)*H1(:)';Z=XH(:);%Initial X*H Kronecker
%Index mapping for square suppot constraint:
for J=0:M-1;for K1=1:L;for K2=0:L-1;K3=K1+K2*L;
I((J+K2)*(L+M-1)+[1:M]+(K1-1),K3)...
=J*M+[1:M]'+(K3-1)*M*M;end;end;end;
%Iterative part of algorithm (K=iteration #):
for K=1:2000;for J=1:(L+M-1)*(L+M-1);II=I(J,I(J,:)>0);
Z(II)=Z(II)+(Y(J)-sum(Z(II)))/length(II);end;
ZZ=reshape(Z,M*M,L*L);[V S]=eig(ZZ'*ZZ);
[S1 J]=max(abs(diag(S)));V1=V(:,J);U1=ZZ*V1;
XH1=U1*V1';Z=XH1(:);end;
XHAT=reshape(U1,M,M);HHAT=reshape(V1,L,L);
subplot(441),imagesc(X),colormap(gray),title('ORIGINAL')
subplot(442),imagesc(Y),colormap(gray),title('BLURRED')
subplot(443),imagesc(X1),colormap(gray),title('INITIAL')
subplot(444),imagesc(XHAT),colormap(gray),title('RECONS')
subplot(445),imagesc(H),colormap(gray),title('ORIGINAL')
subplot(447),imagesc(H1),colormap(gray),title('INITIAL')
subplot(448),imagesc(HHAT),colormap(gray),title('RECONS')
```