
Kronecker Products

in

Image Restoration

James G. Nagy
Emory University
Atlanta, GA

Julie Kamm (Raytheon)
Lisa Perrone (Emory)
Michael Ng (HKU)
Misha Kilmer (Tufts)

Kronecker Product

$$C \otimes D = \begin{bmatrix} c_{11}D & c_{12}D & \cdots & c_{1n}D \\ c_{21}D & c_{22}D & \cdots & c_{2n}D \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1}D & c_{m2}D & \cdots & c_{mn}D \end{bmatrix}$$

- Properties:
- $(C \otimes D)^T = C^T \otimes D^T$
 - $(C \otimes D)^{-1} = C^{-1} \otimes D^{-1}$
 - $(CDE) \otimes (FGH) = (C \otimes F)(D \otimes G)(E \otimes H)$
 - $(C \otimes D)\mathbf{x} \Leftrightarrow DXC^T, \quad \mathbf{x} = \text{vec}(X)$

Image Restoration

Given distorted image, reconstruct picture of original scene

$$\mathbf{b} = A\mathbf{x} + \mathbf{n}$$

\mathbf{b} = distorted (blurred and noisy) image

\mathbf{x} = true image

\mathbf{n} = noise

A = matrix modeling distortion process

Computational Problem:

Given \mathbf{b} , A , find an approximation of \mathbf{x}

Regularization

- Problem: Given $\mathbf{b} = A\mathbf{x} + \mathbf{n}$, compute \mathbf{x}
- Ill-conditioned $A \Rightarrow \hat{\mathbf{x}} = A^{-1}\mathbf{b}$ is bad!
- Regularization \Rightarrow math trick to stabilize numerical scheme.
 - Regularization by filtering
e.g., using singular value decomposition (SVD)



Regularization

- Problem: Given $\mathbf{b} = A\mathbf{x} + \mathbf{e}$, compute \mathbf{x}
 - Ill-conditioned $A \Rightarrow \hat{\mathbf{x}} = A^{-1}\mathbf{b}$ is bad!
 - Regularization \Rightarrow math trick to stabilize numerical scheme.
 - Regularization by filtering
e.g., using singular value decomposition (SVD)
 - Regularization by iteration
e.g., using conjugate gradients (CG)
-

Outline

1. Regularization by filtering.
2. Constructing the matrix A .
(Kronecker products)
3. Regularization by iteration.
4. Summary.

Regularization by Filtering

Suppose $A = U\Sigma V^T$, $U^T U = I$, $V^T V = I$
 $\Sigma =$ diagonal matrix

Then the "inverse" solution is

$$\begin{aligned} \mathbf{x} &= A^{-1}\mathbf{b} \\ &= V\Sigma^{-1}U^T\mathbf{b} \\ &= \sum_{i=1}^n \frac{\mathbf{u}_i^T \mathbf{b}}{\sigma_i} \mathbf{v}_i \end{aligned}$$

Regularization by Filtering

Suppose $A = U\Sigma V^T$, $U^T U = I$, $V^T V = I$
 Σ = diagonal matrix

Then the "inverse" solution is

$$\begin{aligned}\hat{\mathbf{x}} &= A^{-1}(\mathbf{b} + \boldsymbol{\varepsilon}) \\ &= V\Sigma^{-1}U^T(\mathbf{b} + \boldsymbol{\varepsilon}) \\ &= \sum_{i=1}^n \frac{\mathbf{u}_i^T(\mathbf{b} + \boldsymbol{\varepsilon})}{\sigma_i} \mathbf{v}_i\end{aligned}$$

Regularization by Filtering

Suppose $A = U\Sigma V^T$, $U^T U = I$, $V^T V = I$
 $\Sigma =$ diagonal matrix

Then the "inverse" solution is

$$\begin{aligned}\hat{\mathbf{x}} &= A^{-1}(\mathbf{b} + \boldsymbol{\varepsilon}) \\ &= V\Sigma^{-1}U^T(\mathbf{b} + \boldsymbol{\varepsilon}) \\ &= \sum_{i=1}^n \frac{\mathbf{u}_i^T(\mathbf{b} + \boldsymbol{\varepsilon})}{\sigma_i} \mathbf{v}_i \\ &= \sum_{i=1}^n \frac{\mathbf{u}_i^T \mathbf{b}}{\sigma_i} \mathbf{v}_i + \sum_{i=1}^n \frac{\mathbf{u}_i^T \boldsymbol{\varepsilon}}{\sigma_i} \mathbf{v}_i \\ &= \mathbf{x} + \text{error}\end{aligned}$$

Regularization by Filtering

Basic Idea: Filter out problems caused by small singular values.

That is, compute:

$$\mathbf{x}_{reg} = \sum_{i=1}^N \phi_i \frac{\mathbf{u}_i^T \mathbf{b}}{\sigma_i} \mathbf{v}_i$$

where the “filter factors” satisfy:

$$\phi_i \approx \begin{cases} 1 & \text{if } \sigma_i \text{ is large} \\ 0 & \text{if } \sigma_i \text{ is small} \end{cases}$$

Regularization by Filtering

- Truncated SVD (of pseudo inverse filter)

$$\mathbf{x}_{\text{tsvd}} = \sum_{i=1}^k \frac{\mathbf{u}_i^T \mathbf{b}}{\sigma_i} \mathbf{v}_i$$

- Tikhonov

$$\mathbf{x}_{\text{tik}} = \sum_{i=1}^N \frac{\sigma_i^2}{\sigma_i^2 + \alpha^2} \frac{\mathbf{u}_i^T \mathbf{b}}{\sigma_i} \mathbf{v}_i$$

Regularization by Filtering

One method to choose regularization parameters is:

Generalized Cross Validation

For example, in Tikhonov, choose α to minimize

$$GCV(\alpha) = \frac{N \sum_{i=1}^N \left(\frac{\mathbf{u}_i^T \mathbf{b}}{\sigma_i^2 + \alpha^2} \right)^2}{\left(\sum_{i=1}^N \frac{1}{\sigma_i^2 + \alpha^2} \right)^2}$$

Regularization by Filtering

How to efficiently compute SVD of large matrix?

Exploit structure of A :

1. Circulant \Rightarrow use Fourier transform:

$$A = \mathcal{F}^* \Lambda \mathcal{F}$$

2. "Symmetric" Toeplitz + Hankel \Rightarrow use cosine transform:

$$A = \mathcal{C}^T \Lambda \mathcal{C}$$

3. Kronecker product \Rightarrow use SVD:

$$\begin{aligned} A = C \otimes D &= (U_c \Sigma_c V_c^T) \otimes (U_d \Sigma_d V_d^T) \\ &= (U_c \otimes U_d) (\Sigma_c \otimes \Sigma_d) (V_c \otimes V_d)^T \end{aligned}$$

Constructing matrix A

- Assume distortion operation is "spatially invariant"
- Choose an appropriate boundary condition
(e.g., zero, periodic, reflexive)
- Build matrix one column at a time:

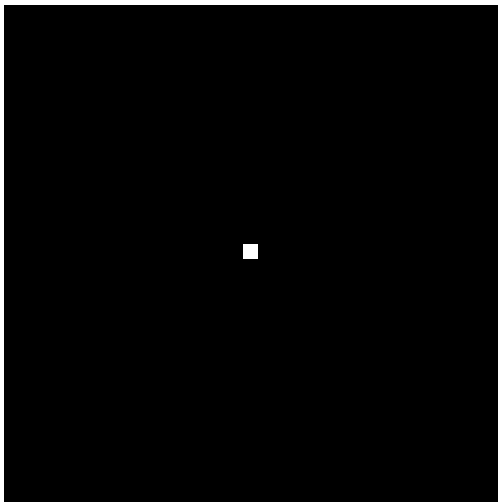
$$i\text{-th column} = A\mathbf{e}_i,$$

where $\mathbf{e}_i = i\text{-th unit vector}$

Constructing matrix A

Build matrix one column at a time, using imaging system:

- Suppose $e_i = i$ th unit vector

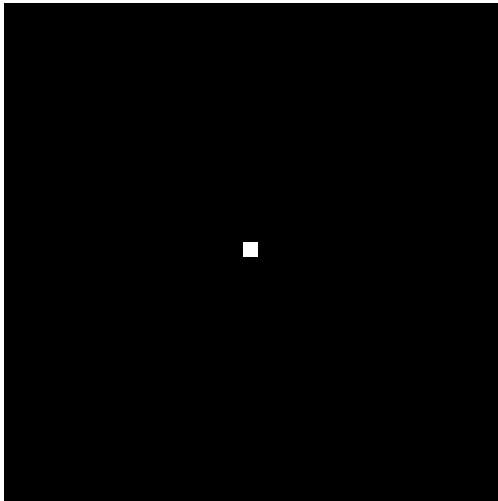


point source

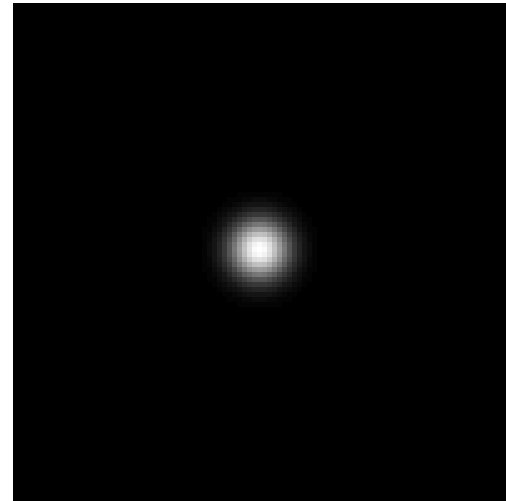
Constructing matrix A

Build matrix one column at a time, using imaging system:

- Suppose $e_i = i$ th unit vector
- Then $Ae_i = i$ th column of A



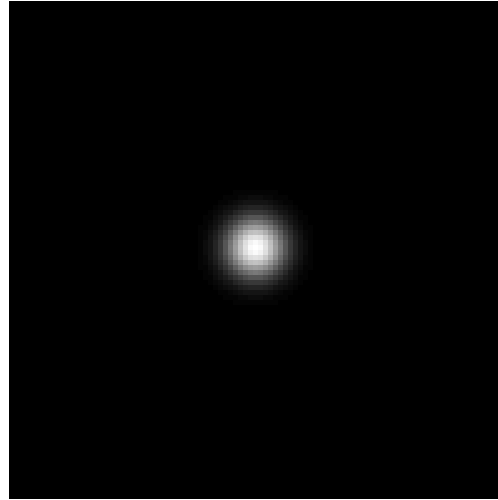
point source



point spread function (PSF)

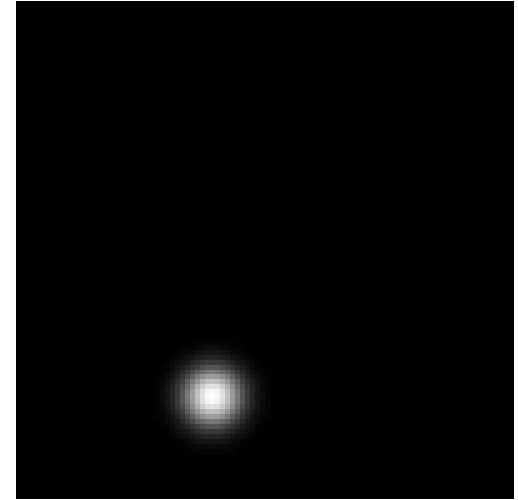
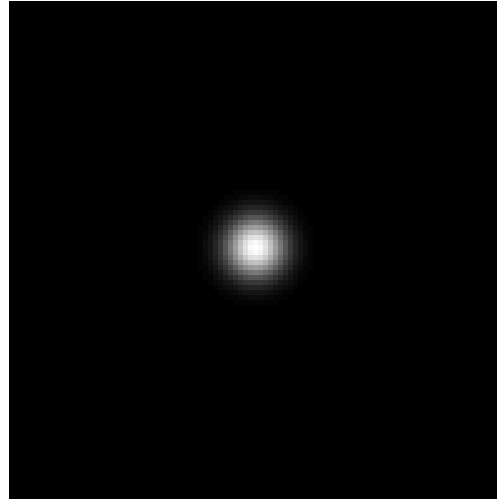
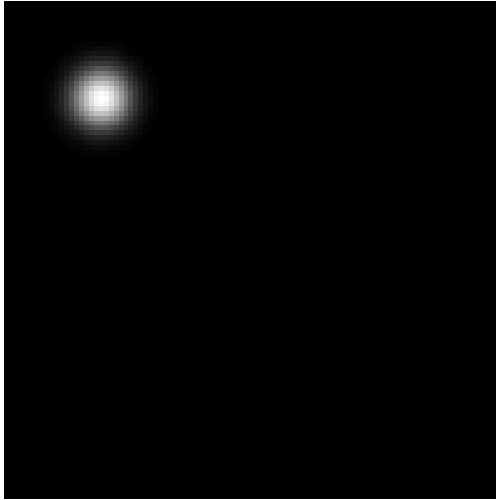
Constructing matrix A

Spatially invariant \Rightarrow distortion independent of position



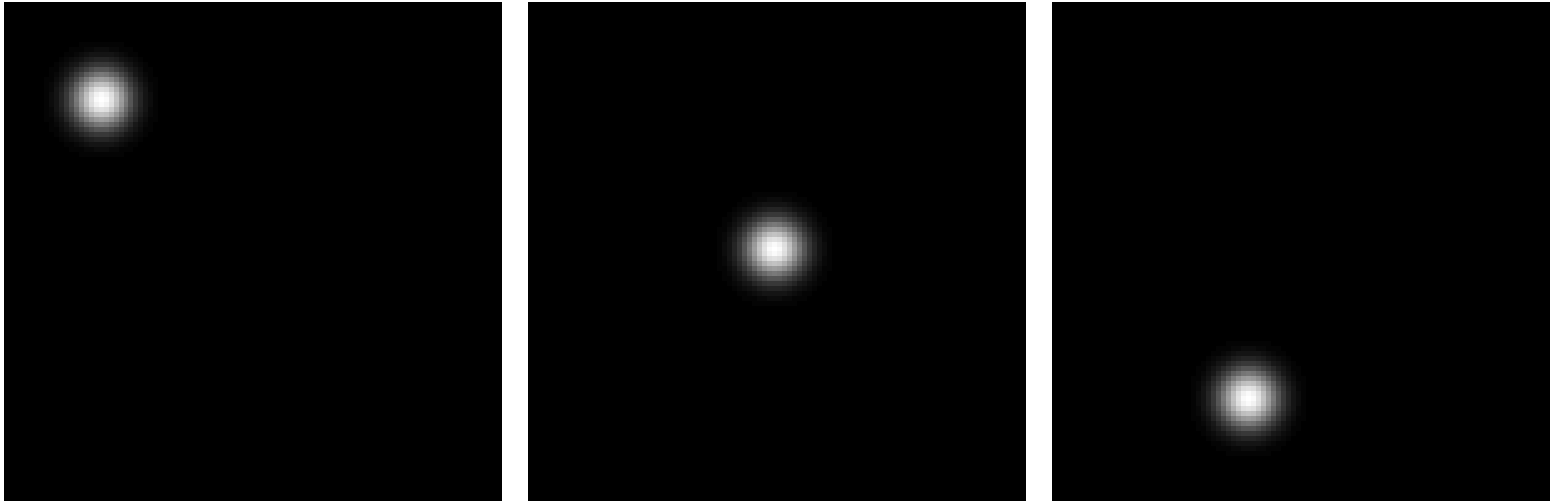
Constructing matrix A

Spatially invariant \Rightarrow distortion independent of position



Constructing matrix A

Spatially invariant \Rightarrow distortion independent of position



Each column of A is identical, modulo shift

\Rightarrow one PSF is enough to describe A

$\Rightarrow A$ has Toeplitz structure

Constructing matrix A : Example 1

$$\mathbf{e}_5 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \rightarrow \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \xrightarrow{\text{blur}} \begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{bmatrix} \rightarrow A\mathbf{e}_5 = \begin{bmatrix} p_{11} \\ p_{12} \\ p_{13} \\ p_{21} \\ p_{22} \\ p_{23} \\ p_{31} \\ p_{32} \\ p_{33} \end{bmatrix}$$

$$A = \begin{bmatrix} & p_{11} & \\ & p_{12} & \\ & p_{13} & \\ & p_{21} & \\ & p_{22} & \\ & p_{23} & \\ & p_{31} & \\ & p_{32} & \\ & p_{33} & \end{bmatrix}$$

Constructing matrix A : Example 1

$$\mathbf{e}_5 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \rightarrow \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \xrightarrow{\text{blur}} \begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{bmatrix} \rightarrow A\mathbf{e}_5 = \begin{bmatrix} p_{11} \\ p_{12} \\ p_{13} \\ p_{21} \\ p_{22} \\ p_{23} \\ p_{31} \\ p_{32} \\ p_{33} \end{bmatrix}$$

$$A = \begin{bmatrix} & p_{12} & p_{11} & & \\ & p_{13} & p_{12} & p_{11} & \\ & & p_{13} & p_{12} & \\ p_{22} & p_{21} & & & \\ p_{23} & p_{22} & p_{21} & & \\ & p_{23} & p_{22} & & \\ p_{32} & p_{31} & & & \\ p_{33} & p_{32} & p_{31} & & \\ & p_{33} & p_{32} & & \end{bmatrix}$$

Constructing matrix A : Example 2

$$\begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{bmatrix} = \mathbf{cd}^T = \begin{bmatrix} c_1 d_1 & c_1 d_2 & c_1 d_3 \\ c_2 d_1 & c_2 d_2 & c_2 d_3 \\ c_3 d_1 & c_3 d_2 & c_3 d_3 \end{bmatrix} \rightarrow A\mathbf{e}_5 = \begin{bmatrix} c_1 \begin{pmatrix} d_1 \\ d_2 \\ d_3 \end{pmatrix} \\ c_2 \begin{pmatrix} d_1 \\ d_2 \\ d_3 \end{pmatrix} \\ c_3 \begin{pmatrix} d_1 \\ d_2 \\ d_3 \end{pmatrix} \end{bmatrix}$$

Constructing matrix A : Example 2

$$\begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{bmatrix} = \mathbf{c}\mathbf{d}^T = \begin{bmatrix} c_1d_1 & c_1d_2 & c_1d_3 \\ c_2d_1 & c_2d_2 & c_2d_3 \\ c_3d_1 & c_3d_2 & c_3d_3 \end{bmatrix} \rightarrow A\mathbf{e}_5 = \begin{bmatrix} c_1 \begin{pmatrix} d_1 \\ d_2 \\ d_3 \end{pmatrix} \\ c_2 \begin{pmatrix} d_1 \\ d_2 \\ d_3 \end{pmatrix} \\ c_3 \begin{pmatrix} d_1 \\ d_2 \\ d_3 \end{pmatrix} \end{bmatrix}$$

$$\begin{bmatrix} & c_1 \begin{pmatrix} d_1 \\ d_2 \\ d_3 \end{pmatrix} & \\ & c_2 \begin{pmatrix} d_1 \\ d_2 \\ d_3 \end{pmatrix} & \\ & c_3 \begin{pmatrix} d_1 \\ d_2 \\ d_3 \end{pmatrix} & \end{bmatrix}$$

Constructing matrix A : Example 2

$$\begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{bmatrix} = \mathbf{cd}^T = \begin{bmatrix} c_1 d_1 & c_1 d_2 & c_1 d_3 \\ c_2 d_1 & c_2 d_2 & c_2 d_3 \\ c_3 d_1 & c_3 d_2 & c_3 d_3 \end{bmatrix} \rightarrow A\mathbf{e}_5 = \begin{bmatrix} c_1 \begin{pmatrix} d_1 \\ d_2 \\ d_3 \end{pmatrix} \\ c_2 \begin{pmatrix} d_1 \\ d_2 \\ d_3 \end{pmatrix} \\ c_3 \begin{pmatrix} d_1 \\ d_2 \\ d_3 \end{pmatrix} \end{bmatrix}$$

$$\left[\begin{array}{c|c|c} & c_1 \begin{pmatrix} d_2 & d_1 \\ d_3 & d_2 \\ & d_3 & d_2 \end{pmatrix} & \\ \hline & c_2 \begin{pmatrix} d_2 & d_1 \\ d_3 & d_2 \\ & d_3 & d_2 \end{pmatrix} & \\ \hline & c_3 \begin{pmatrix} d_2 & d_1 \\ d_3 & d_2 \\ & d_3 & d_2 \end{pmatrix} & \end{array} \right]$$

Constructing matrix A : Example 2

$$\begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{bmatrix} = \mathbf{c} \mathbf{d}^T = \begin{bmatrix} c_1 d_1 & c_1 d_2 & c_1 d_3 \\ c_2 d_1 & c_2 d_2 & c_2 d_3 \\ c_3 d_1 & c_3 d_2 & c_3 d_3 \end{bmatrix} \rightarrow A \mathbf{e}_5 = \begin{bmatrix} c_1 \begin{pmatrix} d_1 \\ d_2 \\ d_3 \end{pmatrix} \\ c_2 \begin{pmatrix} d_1 \\ d_2 \\ d_3 \end{pmatrix} \\ c_3 \begin{pmatrix} d_1 \\ d_2 \\ d_3 \end{pmatrix} \end{bmatrix}$$

$$\left[\begin{array}{c|c|c} c_2 \begin{pmatrix} d_2 & d_1 \\ d_3 & d_2 \\ & d_1 \\ & d_2 \end{pmatrix} & c_1 \begin{pmatrix} d_2 & d_1 \\ d_3 & d_2 \\ & d_1 \\ & d_2 \end{pmatrix} & \\ \hline c_3 \begin{pmatrix} d_2 & d_1 \\ d_3 & d_2 \\ & d_1 \\ & d_2 \end{pmatrix} & c_2 \begin{pmatrix} d_2 & d_1 \\ d_3 & d_2 \\ & d_1 \\ & d_2 \end{pmatrix} & c_1 \begin{pmatrix} d_2 & d_1 \\ d_3 & d_2 \\ & d_1 \\ & d_2 \end{pmatrix} \\ \hline & c_3 \begin{pmatrix} d_2 & d_1 \\ d_3 & d_2 \\ & d_1 \\ & d_2 \end{pmatrix} & c_2 \begin{pmatrix} d_2 & d_1 \\ d_3 & d_2 \\ & d_1 \\ & d_2 \end{pmatrix} \end{array} \right] = C \otimes D$$

Kronecker Product Summary

If point spread function satisfies: $P = \mathbf{c}\mathbf{d}^T$, then

- Zero BC $\Rightarrow A = C \otimes D$, where C and D are Toeplitz
- Periodic BC $\Rightarrow A = C \otimes D$, where C and D are circulant
- Reflexive BC $\Rightarrow A = C \otimes D$, where C and D are Toeplitz+Hankel

General Kronecker Product Decomposition

- If point spread function satisfies: $P = \sum_{k=1}^r \mathbf{c}_k \mathbf{d}_k^T$, then

$$A = \sum_{k=1}^r C_k \otimes D_k$$

- Optimal approximation using work by Pitsianis and Van Loan (Kamm, N., 2000; N., Ng, Perrone, 2003)
- For 3-dimensional problems, need tensor decompositions.

Kronecker Product Approximations

Optimal approximations (2-D problems):

- $P = \sum_{k=1}^r \sigma_k \mathbf{u}_k \mathbf{v}_k^T \Rightarrow \sigma_1 \mathbf{u}_1 \mathbf{v}_1^T = \text{best rank-1 approximation}$

- Setting $\mathbf{c}_1 = \sqrt{\sigma_1} \mathbf{u}_1$ and $\mathbf{d}_1 = \sqrt{\sigma_1} \mathbf{v}_1$, then

$$A \approx C_1 \otimes D_1 = \text{best Kronecker product approximation}$$

Kronecker Product Approximations

Optimal approximations (2-D problems):

- $P = \sum_{k=1}^r \sigma_k \mathbf{u}_k \mathbf{v}_k^T \Rightarrow \sigma_1 \mathbf{u}_1 \mathbf{v}_1^T = \text{best rank-1 approximation}$

- Setting $\mathbf{c}_1 = \sqrt{\sigma_1} \mathbf{u}_1$ and $\mathbf{d}_1 = \sqrt{\sigma_1} \mathbf{v}_1$, then

$A \approx C_1 \otimes D_1 = \text{best Kronecker product approximation}$

$$= (U_c \Sigma_c V_c^T) \otimes (U_d \Sigma_d V_d^T) = \text{SVD approximation}$$

Kronecker Product Approximations

Optimal approximations (2-D problems):

- $P = \sum_{k=1}^m \sigma_k \mathbf{u}_k \mathbf{v}_k^T \Rightarrow \sum_{k=1}^m \sigma_k \mathbf{u}_k \mathbf{v}_k^T = \text{rank-}m \text{ approximation}$

- Setting $\mathbf{c}_k = \sqrt{\sigma_k} \mathbf{u}_k$ and $\mathbf{d}_k = \sqrt{\sigma_k} \mathbf{v}_k$, then

$$A \approx \sum_{k=1}^m C_k \otimes D_k = \text{sum of Kronecker product approximation}$$

Kronecker Product Approximations

Optimal approximations (2-D problems):

$$A = \sum_{k=1}^r C_k \otimes D_k \approx (U_c \otimes U_d) \Sigma (V_c \otimes V_d)^T$$

where $C_1 = U_c \Sigma_c V_c^T$, $D_1 = U_d \Sigma_d V_d^T$

$$\min_{\Sigma} \|A - (U_c \otimes U_d) \Sigma (V_c \otimes V_d)^T\|_F$$

$$\begin{aligned} \Sigma &= \text{diag} \left((U_c \otimes U_d)^T A (V_c \otimes V_d) \right) \\ &= \text{diag} \left((U_c \otimes U_d)^T \left(\sum_{k=1}^r C_k \otimes D_k \right) (V_c \otimes V_d) \right) \end{aligned}$$

Kronecker Product Approximations

For 3-D problems:

- Need orthogonal tensor decompositions

SIMAX: de Lathauwer, de Moor, Vandewalle, '00;

Kolda, '01;

Zhang, Golub, '01

- We use HOSVD (de Lathauwer, de Moor, Vandewalle, '00):

$$P = \sum_{i=1}^{r_1} \sum_{j=1}^{r_2} \sum_{k=1}^{r_3} \sigma_{ijk} \mathbf{u}_i \circ \mathbf{v}_j \circ \mathbf{w}_k.$$

- These vectors define matrices C_i , D_j and F_k , with

$$A = \sum_{\sigma_{ijk} \neq 0} \sum \sum C_i \otimes D_j \otimes F_k,$$

SVD Summary

The PSF and boundary condition define A :

- Periodic BC \Rightarrow FFT decomposition $O(n^2 \log n)$
 - Reflexive BC, symmetric PSF \Rightarrow DCT decomposition $O(n^2 \log n)$
 - Otherwise Kronecker product approximation \Rightarrow SVD $O(n^3)$
-

SVD Summary

The PSF and boundary condition define A :

- Periodic BC \Rightarrow FFT decomposition $O(n^2 \log n)$
- Reflexive BC, symmetric PSF \Rightarrow DCT decomposition $O(n^2 \log n)$
- Otherwise Kronecker product approximation \Rightarrow SVD $O(n^3)$

In MATLAB:

```
>> A = psfMatrix(PSF, 'boundary condition');  
>> [U, S, V] = svd(A);  
>> x = TSVD(U, S, V, b);
```

Iterative Regularization

We consider algorithms (e.g., conjugate gradients) of the form

\mathbf{x}_0 = initial estimate of \mathbf{x}

for $j = 0, 1, 2, \dots$

- \mathbf{x}_{j+1} = computations involving \mathbf{x}_j , A ,
preconditioner matrix, M
- determine if stopping criteria are satisfied

end

Need to efficiently:

multiply with A , A^T

solve linear systems with M , M^T .

Multiply with A

- Use FFTs to form convolutions on padded arrays
- Padding depends on:
 - boundary condition
 - support (size) of PSF
- Can also do spatially variant blurs.

Preconditioning

Typical approach for $A\mathbf{x} = \mathbf{b}$

Find matrix M satisfying the following properties:

- Inexpensive to “construct” M .
- Inexpensive to solve $M\mathbf{z} = \mathbf{w}$.
- $M^{-1}A \approx I$ (singular values of $M^{-1}A$ cluster around 1).

Preconditioning

“Ideal” preconditioner: $M = A = U\Sigma V^T$

- Solving $M\mathbf{z} = \mathbf{w} \Rightarrow \mathbf{z} = V\Sigma^{-1}U^T\mathbf{w}$.
- For ill-posed problems, noise/errors in \mathbf{w} are amplified.

Preconditioning

“Ideal” preconditioner: $M = A = U\Sigma V^T$

- Solving $M\mathbf{z} = \mathbf{w} \Rightarrow \mathbf{z} = V\Sigma^{-1}U^T\mathbf{w}$.
- For ill-posed problems, noise/errors in \mathbf{w} are amplified.
- Remedy: Regularize using truncated SVD

$$\mathbf{z} = V\Sigma^\dagger U^T\mathbf{w}$$

where $\Sigma^\dagger = \text{diag}(1/\sigma_1, \dots, 1/\sigma_k, 0, \dots, 0)$

- Problem: The matrix M is singular.

Preconditioning Ill-Posed Problems

Modify TSVD idea (Hanke, N., Plemmons, '93)

$$\mathbf{z} = V \Sigma_{\tau}^{-1} U^T \mathbf{w}$$

where

- $\Sigma_{\tau}^{-1} = \text{diag}(1/\sigma_1, \dots, 1/\sigma_k, 1, \dots, 1)$
- $\sigma_k \geq \tau > \sigma_{k+1}$

That is, the preconditioner is:

$$M_{\tau} = U \Sigma_{\tau} V^T$$

Preconditioning Ill-Posed Problems

Notice that the preconditioned system is:

$$\begin{aligned}M_{\tau}^{-1}A &= (U\Sigma_{\tau}V^T)^{-1}(U\Sigma V^T) \\ &= V\Sigma_{\tau}^{-1}\Sigma V^T \\ &= V\Delta V^T\end{aligned}$$

where $\Delta = \text{diag}(1, \dots, 1, \sigma_{k+1}, \dots, \sigma_n)$

That is,

- Large (good) singular values clustered at 1.
- Small (bad) singular values not clustered.

Preconditioning Summary

- Construct preconditioner by approximating A with either

FFT, DCT or SVD

- Use modified TSVD to regularize preconditioner.

In MATLAB:

```
>> A = psfMatrix(PSF, 'boundary condition');  
>> M = svdPrec(A, b);
```

The End!

- Exploiting Kronecker product structure in image restoration
 - Can be effective in filtering methods that use SVD
 - Can be effective as preconditioners
- Matlab software:
<http://www.mathcs.emory.edu/~nagy/RestoreTools/>
- Related software for ill-posed problems (Hansen, Jacobsen)
<http://www.imm.dtu.dk/~pch/Regutools/>