
An Object Oriented Approach to Image Restoration in Matlab

James G. Nagy

Emory University
Atlanta, GA

www.mathcs.emory.edu/~nagy

Joint work with Katrina Lee and Lisa Perrone

Image Restoration (Deconvolution)

$$g = Hf + n$$

g = blurred, noisy image

f = true image

n = noise

H = blurring matrix (large, structured, ill-conditioned)

Computational Problem:

Given g , H (or PSF), find an approximation of f

Regularization

- Naive solution $\mathbf{f} = H^{-1}\mathbf{g}$
corrupted by noise if H is ill-conditioned.
 - Regularization \Rightarrow mathematical trick to stabilize numerical scheme.
 - Regularization by filtering
(e.g., Pseudo inverse filter, TSVD, Wiener filter, Tikhonov)
 - Regularization by iteration
(e.g., Lucy-Richardson, CG, EMLS, ...)
-

Outline

1. Regularization by filtering.

- Computational issues.
- Matlab tools.

2. Regularization by iteration.

- Computational issues.
- Matlab tools.

3. Summary

Regularization by Filtering

Suppose $H = U\Sigma V^*$, $U^*U = I$, $V^*V = I$
 Σ =diagonal matrix

Then the naive inverse solution is

$$\begin{aligned} \mathbf{f} = H^{-1}\mathbf{g} &= V\Sigma^{-1}U^*\mathbf{g} \\ &= \sum_i \frac{\mathbf{u}_i^*\mathbf{g}}{\sigma_i} \mathbf{v}_i \\ &= \sum_i \left(f_i + \frac{n_i}{\sigma_i} \right) \mathbf{v}_i \end{aligned}$$

Small singular values magnify high frequency \mathbf{v}_i .

Regularization by Filtering

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

That is, compute:

$$\mathbf{f}_{reg} = \sum_{i=1}^N \phi_i \frac{\mathbf{u}_i^* \mathbf{g}}{\sigma_i} \mathbf{v}_i$$

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

1. Truncated SVD (of pseudo inverse filter)

$$\mathbf{f}_{\text{tsvd}} = \sum_{i=1}^k \frac{\mathbf{u}_i^* \mathbf{g}}{\sigma_i} \mathbf{v}_i$$

2. Tikhonov

$$\mathbf{f}_{\text{tik}} = \sum_{i=1}^N \frac{\sigma_i^2}{\sigma_i^2 + \alpha^2} \frac{\mathbf{u}_i^* \mathbf{g}}{\sigma_i} \mathbf{v}_i$$

3. Wiener filter

$$\mathbf{f}_{\text{wien}} = \sum_{i=1}^N \frac{\delta_i \sigma_i^2}{\delta_i \sigma_i^2 + \lambda_i} \frac{\mathbf{u}_i^* \mathbf{g}}{\sigma_i} \mathbf{v}_i$$

Regularization by Filtering

One method to choose regularization parameters is:

Generalized Cross Validation

For example, in Tikhonov, choose α to minimize

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

Regularization by Filtering

Computational Issues:

1. Choose an appropriate basis for U and V
e.g., singular vectors, Fourier vectors, ...
2. Efficient algorithm (speed and storage) to compute

$$H = U\Sigma V^*$$

psfMatrix, H

- Assume spatially invariant blur.
- H is large, structured – defined by
 - PSF
 - boundary condition

psfMatrix, H

- Assume spatially invariant blur.
- H is large, structured – defined by
 - PSF
 - boundary condition

zero

psfMatrix, H

- Assume spatially invariant blur.
- H is large, structured – defined by
 - PSF
 - boundary condition
 - periodic
 - zero

psfMatrix, H

- Assume spatially invariant blur.
- H is large, structured – defined by
 - PSF
 - boundary condition

periodic

zero

reflexive

psfMatrix, H

Matlab matrix constructor:

```
>> H = psfMatrix(PSF);  
>> H = psfMatrix(PSF, boundary);  
>> H = psfMatrix(PSF, center, boundary);
```

Notes:

- Matrix is **not** explicitly constructed
- PSF = image array
- center = location of point source
- boundary = 'periodic', 'zero', 'reflexive'

psfMatrix, H

Overload Matlab functions and operators:

* \Rightarrow `>> g = H * f;`

svd \Rightarrow `>> [U, S, V] = svd(H);`

SVD computation depends on PSF and boundary condition.

Computing SVD

Easy cases for SVD:

- Periodic BC $\Rightarrow H = \mathcal{F}^* \Sigma \mathcal{F}$

$$U = V = \text{FFT}, \quad \Sigma = \text{eigenvalues (i.e., OTF)}$$

- Reflexive BC, symmetric PSF $\Rightarrow H = \mathcal{C}^* \Sigma \mathcal{C}$

$$U = V = \text{DCT}, \quad \Sigma = \text{eigenvalues}$$

Computing SVD

Other cases, check separability of PSF

- If PSF is separable, $\text{PSF} = \mathbf{a}\mathbf{b}^T$

$$H = A \otimes B$$

- If PSF is not separable, find best approximation among easy cases.

Kronecker Products

$$\begin{aligned} H &= A \otimes B \\ n^2 \times n^2 & \quad (n \times n) \otimes (n \times n) \\ &= \begin{bmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & & \vdots \\ a_{n1}B & \cdots & a_{nn}B \end{bmatrix} \end{aligned}$$

SVD and Kronecker Products

$$A \otimes B$$

SVD and Kronecker Products

$$\begin{array}{ccc} A & \otimes & B \\ (U_a \Sigma_a V_a^T) & \otimes & (U_b \Sigma_b V_b^T) \end{array}$$

SVD and Kronecker Products

$$\begin{array}{ccc} A & \otimes & B \\ (U_a \Sigma_a V_a^T) & \otimes & (U_b \Sigma_b V_b^T) \end{array}$$

$$(U_a \otimes U_b)(\Sigma_a \otimes \Sigma_b)(V_a \otimes V_b)^T$$

SVD and Kronecker Products

$$\begin{array}{ccc} A & \otimes & B \\ (U_a \Sigma_a V_a^T) & \otimes & (U_b \Sigma_b V_b^T) \end{array}$$

$$\begin{array}{ccc} (U_a \otimes U_b) & (\Sigma_a \otimes \Sigma_b) & (V_a \otimes V_b)^T \\ U & \Sigma & V^T \end{array}$$

SVD and Kronecker Products

$$\begin{array}{ccc} A & \otimes & B \\ (U_a \Sigma_a V_a^T) & \otimes & (U_b \Sigma_b V_b^T) \\ \\ (U_a \otimes U_b) & (\Sigma_a \otimes \Sigma_b) & (V_a \otimes V_b)^T \\ U & \Sigma & V^T \end{array}$$

computations done with small matrices

$$\text{e.g., } \hat{\mathbf{g}} = U^T \mathbf{g} \Leftrightarrow \hat{G} = U_b^T G U_a$$

SVD Summary

The Matlab command:

```
>> [U, S, V] = svd(H);
```

Computes either:

- FFT decomposition $O(n^2 \log n)$
 - DCT decomposition $O(n^2 \log n)$
 - SVD of Kronecker product $O(n^3)$
-

Iterative Regularization

We consider algorithms of the form

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

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

- \mathbf{f}_{j+1} = computations involving \mathbf{f}_j , H ,
preconditioner matrix, P
- determine if stopping criteria are satisfied

end

Iterative Regularization

Some examples:

1. Conjugate Gradient (CG)
2. (Constrained) Steepest Descent (EM-LS)
3. Lucy-Richardson (EM-ML)

Need to multiply with H , H^T
solve linear systems with P , P^T .

EM-LS Method (Kaufman, 93; N., Strakos, 00)

Construct iterative method

$$\mathbf{f}^{(j+1)} = \mathbf{f}^{(j)} + \alpha_j \mathbf{p}^{(j)}$$

where we choose $\mathbf{p}^{(j)}$ and α_j to **maintain nonnegativity** of the iterates, e.g.,

- restrict α_j
- incorporate constraints into $\mathbf{p}^{(k)}$

EM-LS Method

$$\text{minimize } \Gamma[\mathbf{f}] = \frac{1}{2} \|\mathbf{g} - H\mathbf{f}\|^2 \quad \text{subject to } \mathbf{x} \geq 0$$

- parameterize: $\mathbf{f}_i = e^{z_i}$

- chain rule \Rightarrow

$$\text{grad } \Gamma[\mathbf{x}] = D_x^T H^T (H\mathbf{f} - \mathbf{g}) \quad D_x = \text{diag}(\mathbf{f})$$

- construct iteration method

$$\mathbf{f}^{(j+1)} = \mathbf{f}^{(j)} + \alpha_j \mathbf{p}^{(j)}$$

such that:

$$\mathbf{p}^{(j)} = -\text{grad}\Gamma[\mathbf{f}]$$

α_j is chosen to minimize residual with bounded line search

EM-LS Algorithm

$$\mathbf{f} = \mathbf{f}^{(0)}$$

$$\mathbf{s} = H^T(H\mathbf{f} - \mathbf{g})$$

$$D = \text{diag}(\mathbf{f})$$

$$\gamma = \mathbf{s}^T D \mathbf{s}$$

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

$$\mathbf{p} = -D\mathbf{s}$$

$$\mathbf{u} = H\mathbf{p}$$

$$\alpha = \min(\gamma / \mathbf{u}^T \mathbf{u}, \min_{p_i < 0} (-f_i / p_i))$$

$$\mathbf{f} = \mathbf{f} + \alpha \mathbf{p}$$

$$D = \text{diag}(\mathbf{f})$$

$$\mathbf{z} = H^T \mathbf{u}$$

$$\mathbf{s} = \mathbf{s} + \alpha \mathbf{z}$$

$$\gamma = \mathbf{s}^T D \mathbf{s}$$

end

Multiply with H

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

Spatially Variant Blurs

- Assume blur is locally invariant
- Assume several PSFs are known
- Use the model (N., O'Leary; Faisal, et al.; Boden et al.)

$$H = \sum_{i=1}^p D_i H_i$$

where $\sum D_i = I$

Spatially Variant Blurs

Computational efficiency for multiplying

$$H = \sum_{i=1}^p D_i H_i$$

is based on:

- Domain decomposition idea
(similar to “overlap-add/save” convolution)
- Use FFTs on subregions

psfMatrix, H

Overload Matlab $*$ operator so that

- Boundary condition checked to determine appropriate padding
- Type of blur (invariant/variant) checked for appropriate algorithm
- Use FFTs for implementation

Note: Some overhead costs with object oriented approach

Preconditioning Ill-Posed Problems

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

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

where

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

That is, the preconditioner is:

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

Preconditioning Ill-Posed Problems

Notice that the preconditioned system is:

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

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.

Matlab Tools for Preconditioning

We construct preconditioner as:

```
>> P = svdPrec(H, g, tol);
```

- Uses our overloaded `svd` function
 - Uses a modified TSVD algorithm
 - Can use GCV to choose `tol`.
-

To-Do List

- Stopping criteria for iterative methods.
- Better understanding of preconditioning EM-LS.
- Include other regularization by filtering methods.
- Include other iterative regularization methods.
(e.g., Blind Image Restoration – Plemmons, Jefferies)
- Test codes using more real applications.

Summary

Build image restoration algorithms from:

- Basic objects `psfMatrix`, `svdPrec`
- Overloaded function `svd`
- Overloaded operators `*` `\`

Some algorithms included:

- TSVD, Tikhonov
- CGLS, EMLS

Matlab software: <http://www.mathcs.emory.edu/~nagy/RestoreTools/>

Related software for ill-posed problems (Hansen, Jacobsen)

<http://www.imm.dtu.dk/~pch/Regutools/>