Region of Interest Image Reconstructions using IR Tools

by

Sherry X. Huang

Dr. James Nagy

Advisor

Department of Mathematics

2019

Region of Interest Image Reconstructions using IR Tools

by

Sherry X. Huang

Dr. James Nagy

Advisor

Abstract of

A thesis submitted to the Faculty of Emory College of Arts and Science

of Emory University in partial fulfillment

of the requirements of the degree of

Bachelor of Sciences with Honors

Department of Mathematics

2019

Abstract

Region of Interest Image Reconstructions using IR Tools

By Sherry X. Huang

In this thesis, we consider large-scale, ill-posed inverse problems that arise in image processing applications. These problems can be modeled as linear systems, where the matrix that models the forward problem is extremely ill-conditioned. In addition, the observed data is contaminated by noise. Due to the ill-conditioning of the matrix and the presence of noise in the observed data, it is necessary to employ regularization to compute a meaningful approximation of the solution.

Although there has been some very effective but expensive image reconstruction algorithm, those algorithms cannot be applied to large images because of their cost. This thesis focuses on using inexpensive, fast methods to obtain an initial image reconstruction first, combined with more expensive methods to improve specific subimages, called region of interest (ROI) areas in the image.

Region of Interest Image Reconstructions using IR Tools

by

Sherry X. Huang

Dr. James Nagy
Advisor

A thesis submitted to the Faculty of Emory College of Arts and Science
of Emory University in partial fulfillment
of the requirements of the degree of
Bachelors of Science with Honors

Department of Mathematics

2019

Contents

List of Figures

Chapter 1

Introduction

A digital image, commonly obtained from cameras, scanners, or printers, is a rectangular grid of pixels. In its numeric representation, the image is a matrix, where each entry of the matrix contains a specific pixel value. Those pixel values are then scaled according to the colormap to represent different colors visually.

In real applications, it is very likely that a measured image is blurred, such as blur caused by defocus aberration or motion blur. The blur is typically modeled as the convolution of a *point spread function* (PSF). We assume the point spread function can be modeled mathematically. Given the degraded image and the blurring function, we want to find the true unknown image. This is an example of an *inverse* problem. In most cases, there is insufficient information in the blurred image to uniquely determine a plausible original image, making it an *ill-posed* problem. In addition, the blurred image often contains additional noise which complicates the task of determining the original image. This is generally solved by the use of a *regularization* method to attempt to eliminate implausible solutions.

1.1 Mathematical Background

In this section we provide some mathematical background material that will be needed throughout the thesis.

1.1.1 Inverse Problem

In image processing applications, when an unknown image is recorded, it is often degraded by blur. A mathematical model of this problem can be expressed in the continuous setting as an integral equation [8] [15]:

$$b(s) = \int_c^d a(s,t)x(t)dt$$

where the function $b(s)$ is the observed (blurred) image, $x(t)$ is the original image, and the kernel $a(s,t)$ is a function that specifies how the points in the image are distorted, and is therefore called the *point spread function* (PSF).

In a realistic setting, images are collected only at discrete points (pixels) and are also only available in a finite region. Therefore, in real cases, one normally work directly with the discrete model of this linear system obtained from *discretization* of the continuous model.

1.1.2 Discretization Issues

Recall that in numerical analysis, a quadrature rule gives an approximation of the definite integral of a function, usually stated as a weighted sum of function values at specified points within the domain of integration

$$R(x) \equiv \sum_{i=0}^{N} w_i x(t_i)$$

given **points** $t_0 < t_1 < ...t_N$ and **weights** $w_0, w_1, ...w_N$.

Therefore, the integral equation can be approximated as follows

$$b(s_i) = \int_c^d a(s_i, t)x(t)dt \approx \sum_{j=0}^{N} w_j a(s_i, t_j)x(t_j)$$

For $i = 0, 1, ...N$, this will give separate linear equations. Hence the above discretization method leads to a matrix equation of the form

$$Ax = b$$

where A is a matrix (called the PSF or blurring matrix) that models the blurring operation:

$$A = \begin{bmatrix} w_1 a(s_1, t_1) & w_2 a(s_1, t_2) & \dots & w_N a(s_1, t_N) \\ w_1 a(s_2, t_1) & w_2 a(s_2, t_2) & \dots & w_N a(s_2, t_N) \\ \vdots & & & \\ w_1 a(s_N, t_1) & w_2 a(s_N, t_2) & \dots & w_N a(s_N, t_N) \end{bmatrix}$$

x and b are vectors that represent the true and blurred images:

$$x = \begin{bmatrix} x(t_1) \\ x(t_2) \\ \vdots \\ x(t_N) \end{bmatrix} \qquad b = \begin{bmatrix} b(s_1) \\ b(s_2) \\ \vdots \\ b(s_N) \end{bmatrix}$$

1.2 Conditioning of the Problem

In the mathematical model, we might assume that the PSF matrix A and b are known. However, in almost all cases, there is insufficient information in the blurred image, thus two problems exist:

- There may not not be a unique solution, x. Hence there may not be a unique image corresponding to the given blurred image and point spread function.

- A small change in the observed image, b, can lead to large changes in the solution, x.

Image restoration is thus called an *ill-posed problem.*

In the equivalent matrix-vector equation obtained from the ill-posed integral equation, we can obtain the *Singular Value Decomposition* (SVD) of the PSF matrix A. The SVD is defined as follows:

- **Singular Value Decomposition.** If A is an $n \times n$ matrix, then there exist $n \times n$ orthogonal matrices U and V, and a diagonal matrix $\Sigma = diag(\sigma_1, \sigma_2, ...\sigma_n)$ such that

  $$A = U\Sigma V^T$$

  where $\sigma_1 \geq \sigma_2 \geq \cdots \sigma_n \geq 0$ are called *singular values* of A, and columns of U and V are called *singular vectors* of A.

By definition, if the *condition number* of A, $\kappa_2(A) = \sigma_1/\sigma_n$ is large, then the matrix A is said to be *ill-conditioned*.

It turns out that if the matrix A comes from discretizing an ill-posed problem, as in the case in image restoration, then the SVD of A typically has the following properties:

- A is very ill-conditioned. That is, $\kappa_2(A) = \sigma_1/\sigma_n$ is large.
- The singular values decay smoothly to zero. So it is difficult to determine a cutoff between large and small singular values.
- The singular vectors u and v tend to oscillate more and more as i increases, causing the true image to be hidden by the noise - discussed in Chapter 2.

To summarize: The image restoration problem is an ill-posed problem, and the PSF matrix A is also ill-conditioned in the sense that the singular value of A gradually decay and cluster at zero.

1.3 Computational Difficulties

We are concerned with large-scale problems, where the PSF matrix A is normally represented by a sparse, ill-conditioned matrix. Also, in most cases, even the observed blurred image $b = Ax$ is not known exactly. Rather, the observed data is typically of the form

$$b_\eta = b + \eta = Ax + \eta$$

where η is a vector representing noise or measurement errors.

Recall the singular value decomposition (SVD) for A exists:

$$A = U\Sigma V^T$$

where U and V are orthogonal matrices. By definition of orthogonal matrix, there are *orthonormal bases*:

$$\{u_1, u_2, ...u_n\} \qquad \{v_1, v_2, ...v_n\}$$

for $\mathbb{R}^n$, in which we can always find scalar coefficients x_i and η_i such that

$$\sum_{i=1}^{n} x_i v_i = x \qquad \sum_{i=1}^{n} \eta_i u_i = \eta$$

Then, b_η can be written as

$$b_\eta = Ax + \eta = A\sum_{i=1}^{n} x_i v_i + \sum_{i=1}^{n} \eta_i u_i$$

Since from $A = U\Sigma V^T$, the property $Av_i = \sigma_i u_i$ holds. Hence

$$b_\eta = \sum_{i=1}^{n} \sigma_i u_i x_i + \sum_{i=1}^{n} \eta_i u_i = \sum_{i=1}^{n} (\sigma_i x_i + \eta_i) u_i$$

In idealistic cases, the observed image b does not contain any noise in it, such that $Ax_e = b$. We can reconstruct the true image x_e using

$$x_e = A^{-1}b = V\Sigma^{-1}U^T b = \sum_{i=1}^{n} \frac{u_i^T b}{\sigma_i} v_i$$

In real applications, however, only b_η can be obtained. If we use what we really have, namely A and observed image with noise b_η to compute x, the computed solution will be of the form:

$$x = A^{-1}b_\eta = \sum_{i=1}^{n} \frac{u_i^T b_\eta}{\sigma_i} v_i = \sum_{i=1}^{n} \frac{u_i^T(\sigma_i x_i + \eta_i)u_i}{\sigma_i} v_i = \sum_{i=1}^{n} \frac{\sigma_i x_i + \eta_i}{\sigma_i} \boldsymbol{v}_i$$

$$x = \sum_{i=1}^{n} (x_i + \frac{\eta_i}{\sigma_i}) v_i$$

Since A is very ill-conditioned, its singular values, σ_i, decay smoothly to zero. When i is large, σ_i is very small, and the noise contributions η_i for large indices i are highly magnified. Thus the computed solution is dominated by noise.

The computed solution x also has the variant form:

$$x = A^{-1}b_\eta = A^{-1}(Ax_e + \eta) = x_e + A^{-1}\eta = x_e + \sum_{i=1}^{n} \frac{u_i^T \eta}{\sigma_i} v_i$$

Since the singular vectors u_i and v_i tend to oscillate more as i increases, the exact true image x_e is also hidden by such oscillation.

As discussed above, a potential problem of image deblurring is that the computed x is very sensitive to noise η. *Regularization* is therefore needed to produce stable solutions to this problem.

So far, we have covered some background on inverse problem, some basics of numerical linear algebra, and the need to use regularization. We introduce two basic regularization methods in Chapter 2: Truncated Singular Value Decomposition and Classical Tikhonov regularization. In Chapter 3, we provide some background materials on image deblurring problems, examples are also presented. Then some iterative methods that can be used to solve those image deblurring problems are discussed in Chapter 4. After that, we introduce the new idea of image reconstruction: region of interest (ROI) computations. This idea is discussed in detail in Chapter 5, where new approaches and tools are described. These tools will be added to the IR Tools software package [6]. Finally, concluding remarks are given in Chapter 6.

Chapter 2

Regularization

Several approaches can be applied to reduce noise contributions in the reconstruction of x. Discrete regularization is a scheme to “filter out” the influence of noise in those noise dominated terms. In particular, a desirable, regularized solution x_{reg} may have the following form:

$$x_{reg} = \sum_{i=1}^{n} \phi_i \frac{\sigma_i x_i + \eta_i}{\sigma_i} v_i$$

where ϕ_i is a scalar called the “filter factor”. It has value between 0 and 1, and satisfies the property that as i increases, the value of ϕ_i tends to zero in such a way that the increasing noise contribution in $\dfrac{\sigma_i x_i + \eta_i}{\sigma_i} v_i$ is filtered out accordingly.

There are various regularization schemes, depending on how the filter factors ϕ_i are defined.

2.1 Truncated SVD

One simple regularization method is implemented by replacing all but the first k "large" singular values of A by zero. Therefore, in a truncated SVD (TSVD) of A, only the first k columns of U and V are included in the solution:

$$x_{reg} = \sum_{i=1}^{k} \frac{b^T u_i}{\sigma_i} v_i$$

where $k \leq n$ and b is the observed blurred image with noise.

Suppose we truncate all singular values of A that fall below a certain tolerance, τ, that is, $\sigma_k \geq \tau \geq \sigma_{k+1}$. The vector of filter factors, $\boldsymbol{\phi}$, is then defined as that the first k values of the vector are 1, and the rest are zero.

2.2 Tikhonov Regularization

More generally, regularization is achieved by solving a penalized lease-squares problem of the form

$$\min_x \{\|Ax - b\|_2^2 + \lambda^2 \Omega(x)\}$$

where λ is the regularization parameter that controls the "smoothness" of the regularized solution, and penalty term $\Omega(x)$ is chosen to reflect a specific type of regularization that is suited for the problem.

In the case where $\Omega(x) = \|x\|_2^2$, the filtering method is called the classical

Tikhonov Regularization. It can be shown that solving the least squares (LS) problem

$$\min_x \{\|Ax - b\|_2^2 + \lambda^2 \|x\|_2^2\}$$

is equivalent to solve the following LS problem

$$\min_x \left\| \begin{bmatrix} A \\ \lambda I \end{bmatrix} x - \begin{bmatrix} b \\ 0 \end{bmatrix} \right\|_2^2$$

Recall solving the *normal equation* gives the solution to a LS problem. The normal equation for the problem above is given by

$$(A^T A + \lambda^2 I) x_{reg} = A^T b$$

According to the SVD of A, the normal equation can be transformed as follows:

$$\begin{aligned} (V\Sigma^T \Sigma V^T + \lambda^2 V V^T) x_{reg} &= V\Sigma^T U^T b \\ V(\Sigma^T \Sigma + \lambda^2) V^T x_{reg} &= V\Sigma^T U^T b \\ diag(\sigma_i^2 + \lambda^2) V^T x_{reg} &= diag(\sigma_i) U^T b \end{aligned}$$

Multiply the inverse of the diagonal matrix on both sides:

$$\begin{aligned} V^T x_{reg} &= diag\left(\frac{1}{\sigma_i^2 + \lambda^2}\right) diag(\sigma_i) U^T b \\ x_{reg} &= V diag\left(\frac{\sigma_i}{\sigma_i^2 + \lambda^2}\right) U^T b \\ x_{reg} &= \sum_{i=1}^{n} \frac{\sigma_i u_i^T b}{\sigma_i^2 + \lambda^2} v_i \end{aligned}$$

Hence the Tikhonov regularized solution is

$$x_{reg} = \sum_{i=1}^{n} \phi_i \frac{u_i^T b}{\sigma_i} v_i$$

where the filter factors ϕ_i for this case are $\phi_i = \dfrac{\sigma_i^2}{\sigma_i^2 + \lambda^2}$.

To summarize: The TSVD method corresponds to sharp filter that simply cuts off the last $(n - k)$ components which are dominated by noise beyond some tolerance. In Tikhonov regularization, the filter factor is smooth that damps the components corresponding to $\sigma_i < \lambda$.

Chapter 3

Image Deblurring

Image deblurring, which is sometimes referred to as image restoration or deconvolution, is the process of removing blurs and noise from degraded images to recover the original true images. This field of image processing technology becomes increasingly significant in many scientific applications such as astronomy [1] [3] [20], medical imaging [1] [18], and microscopy [21].

For example, viewing distant star fields, space vehicles, and satellites using ground based telescopes and obtaining pictures from it may be bothered by the distortion of the real view caused by the random interfering light rays coming from various sources, or the turbulent mixing in the atmosphere of Earth which causes variations of the optical refractive index. Due to such factors, various types of blurs may be incurred. We will use the MATLAB package - IR tools [6], to generate a series of large-scale image deblurring test problems [9].

3.1 Spacially Invariant Blur

The simpler type of blurs are *spacially invariant* blurs, which are independent of the position of the point source. In this case, one point spread function can completely describe the blurring operation.

We can use IR tools to generate the following spacially invariant blurring operations:

– `PRblurdefocus` - the blur caused by being out of focus

– `PRblurgauss` - result of blurring an image by a Gaussian function

– `PRblurshake` - the blur caused by random shaking of a camera

– `PRblurspeckle` - the blur caused by atmospheric turbulence

To start, call the test problems in the IR tools package using the following MATLAB statement:

```
[A, b, x, ProbInfo] = PRblur___(n, options);
```

Fill the blank indicated by ___ with the name corresponding to one of the blurs listed above. The two inputs in the function are optional. `n` is the size of the image and can be either a scalar `n` (so that the size of the image is $n \times n$), or a vector of the form `[nrow ncol]` (so that the size of image is $nrow \times ncol$). If `n` is not specified, the default value is 256. `option` is a structure that can be used to set the optional fields such as:

– `trueImage` - image to be used in this test problem; the package contains `'hst'`(default, image of Hubble space telescope), `'satellite'`...

- `BlurLevel` - severity of the blur; may have value `‘mild’`, `‘medium’` (default), or `‘severe’`
- `BC` - specify boundary condition *
- `CommitCrime` - whether to get the exact system $Ax = b$ *

For the last two, recall that each pixel in the blurred image is formed by integrating a PSF with pixel values of the true image. However, pixels of the blurred image near the boundary of the “viewable” region are affected by information outside the region we can observe. Therefore, in constructing matrix A, one needs to incorporate boundary conditions to model how the image scene extends beyond the boundaries. When `CommitCrime` option is set to `‘On’`, it means the specified boundary condition used to construct A exactly model how x behaves outside the viewable region. However, this situation is unrealistic.

For simplicity, we will only consider various values of `trueImage` and `BlurLevel` to generate our test problems.

3.1.1 Example: Satellite

We can generate a speckle blur test problem, choose the non-default image of satellite of default size, and set the blur level to severe:

```
options = PRset(‘trueImage’, ‘satellite’, ‘BlurLevel’, ‘severe’);
[A, b, x, ProbInfo] = PRblurspeckle(options);
```

Since it is a spacially invariant blur, this function returns a PSF matrix

A, the vector representation of the blurred image b, and the true image x. Information of this test problem is stored as well. Then the vectors b and x produced by this test problem can be displayed using the following command:

```
PRshowb(b, ProbInfo);
PRshowx(x, ProbInfo);
```

We can also display the visual effect of the PSF using either of the functions:

```
PRshowx(ProbInfo.psf, ProbInfo);
mesh(ProbInfo.psf);
```

where `ProbInfo.psf` is the point spread function.

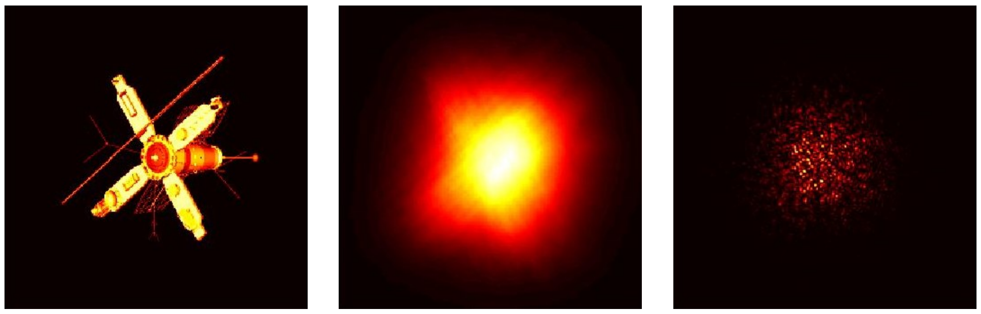


Figure 3.1: Example of an image deblurring problem, where the blur is caused by atmospheric turbulence with a severe blurring level. The PSF which defines the matrix A is shown on the right.

3.1.2 Example: Dots with Noise

We now generate another test problem, using the non-default image of small Gaussian shaped dots with default blurring level. Show the corresponding blurred image from a simulation for camera shaking motion:

```
options = PRset('trueImage', 'dotk');
[A, b, x, ProbInfo] = PRblurshake (options);
```

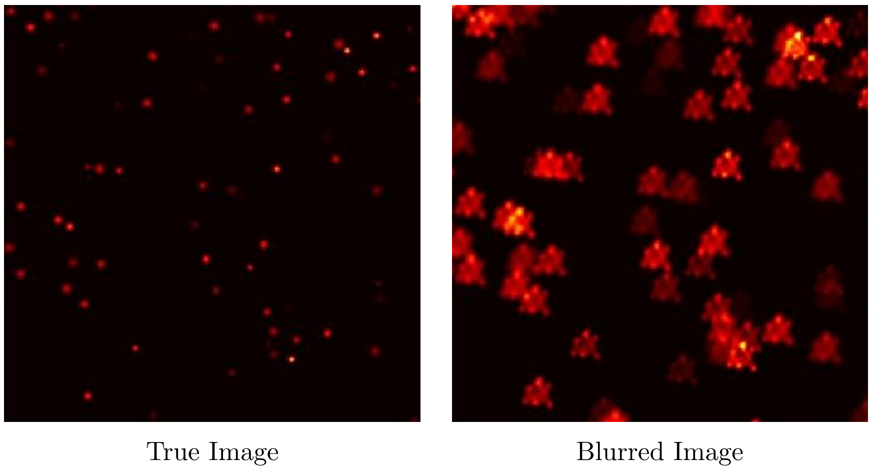


Figure 3.2: Example of an image deblurring problem, where the blur is caused by random shaking of a camera with a medium blurring level.

Adding noise to the data

Use built-in IR toolds function `PRnoise` to add noise to the blurred image data b:

```
[bn, NoiseInfo] = PRnoise(b, NoiseLevel, kind);
```

where the output `bn = b + noise` is the blurred image contaminated by noise. The information about the noise is stored in structure `NoiseInfo`, such as the vector of perturbations `noise`.

The inputs `NoiseLevel` and `kind` are optional. The `NoiseLevel` defines the relative level of noise, with default value `NoiseLevel = 0.01`. Given the noise level, the noise vector `noise` is scaled in such a way that

$$\|\texttt{noise}\|_2/\|\texttt{b}\|_2 = \texttt{NoiseLevel}$$

Three kinds of noise that can be added to the data b are listed as follows:

– `‘gauss’` (default) - Gaussian noise, created by means of MATLAB’s `randn` function.

– `‘laplace’l` - Laplacian noise, generated as follows [6]:

```
r = rand(numel(b),1);
r = sign(0.5-r).*(1/sqrt(2)).*log(2*min(r,1-r));
noise = ((NoiseLevel*norm(b(:)))/norm(r))*r;
```

– `‘multiplicative’` - Multiplicative noise, where each element of `bn` equals the corresponding element of b times a random variable following a Gamma distribution with mean 1, scaled to approximately the desired noise level.

For example, we can then add Gauss and Laplacian noise to the dots image and specify `NoiseLevel = 0.01`:

```
NoiseLevel = 0.01
[bn1, NoiseInfo] = PRnoise(b, NoiseLevel, ‘gauss’);
[bn2, NoiseInfo] = PRnoise(b, NoiseLevel, ‘laplace’);
```

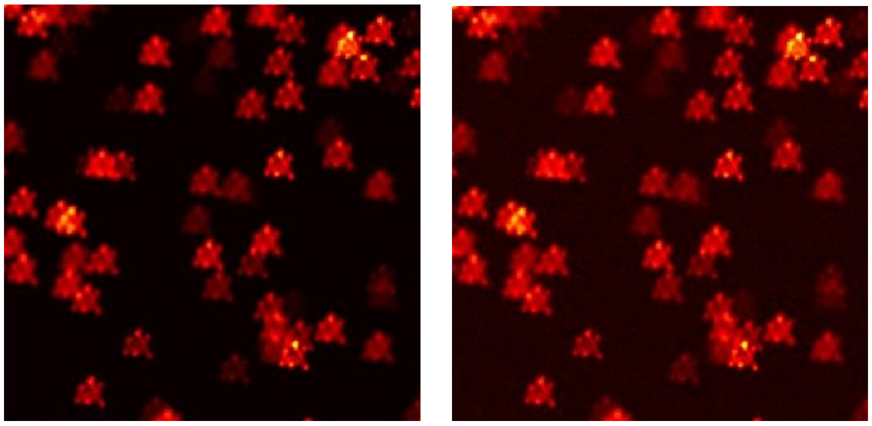


Figure 3.3: The blurred dots image caused by random shaking of a camera, compared to the blurred image with Gaussian noise.

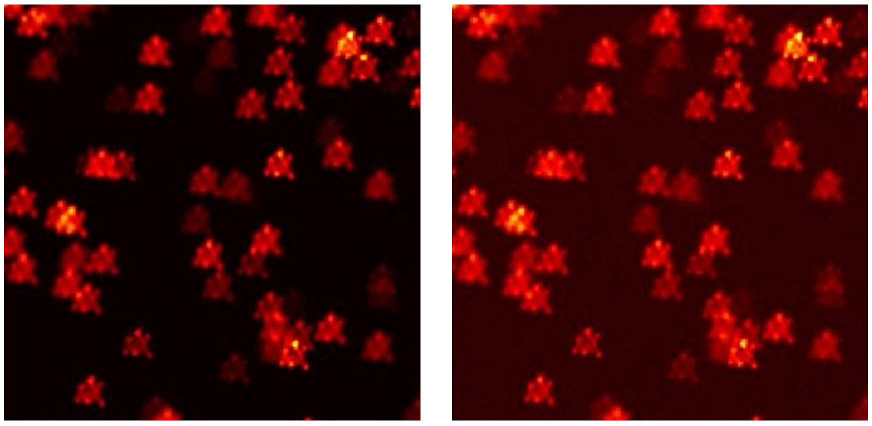


Figure 3.4: The blurred dots image caused by random shaking of a camera, compared to the blurred image with Laplacian noise.

3.2 Spacially Variant Blur

The situation is more difficult if the blurs are *spacially variant*, where the blurs depend on the positions. In this case, a single PSF does not completely describe the blurring operation.

3.2.1 Example: Dots with Rotation Blur

For example, we generate a test problem of spacially variant blur, using the following blurring operation:

– `PRblurrotation` - a spacially variant rotational motion blur around the center of the image

Set the blurring level to severe and obtain the corresponding blurred image:

```
options = PRset(‘trueImage’, ‘dotk’, ‘Blurlevel’, ‘severe’);
[A, b, x, ProbInfo] = PRblurrotation (options);
```

Notice in this case, a single PSF matrix cannot completely describe the blurring operation, and the output A is a sparse matrix. Also notice it takes much longer to generate this test problem.

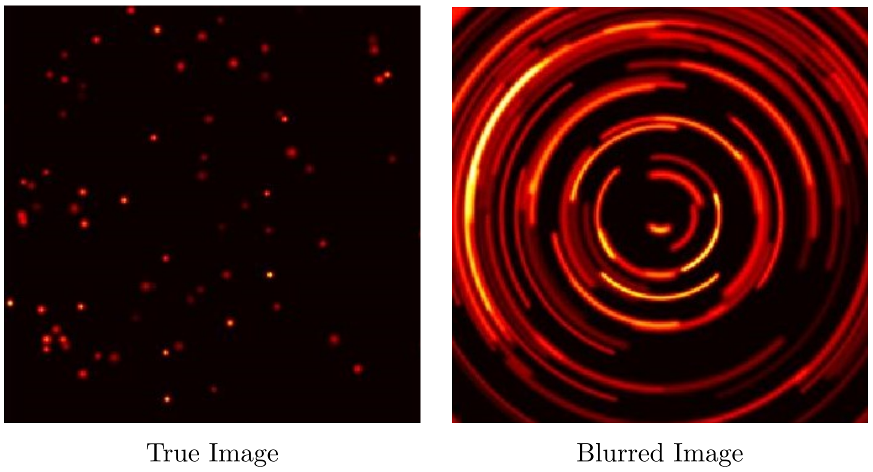


Figure 3.5: Example of an image deblurring problem, where the blur is caused by rotation around the center of the image, with a severe blurring level.

Chapter 4

Iterative Methods for Image Deblurring

We have introduced some basic regularization techniques that help to filter out noise contributions in the computed solution, such as truncated SVD and Tikhonov regularization. A different way to achieve regularization is to apply an iterative method on the fit-to-data term

$$\min_x \|Ax - b\|_2^2$$

and terminate the iteration when a desired approximation is obtained, before noise starts to show up in the solution. This method is often referred to as *iterative regularization.* Iterative regularization allows for the incorporation of various types of prior knowledge about the class of feasible solutions, and it is shown that noise effects can be minimized by terminating the algorithms after a finite number of iterations.

IR tools package contains several iterative solvers to the image deblurring problem [12]. We will use some of them in our region of interest (ROI) computations, as discussed in Chapter 5.

4.1 Iterative Solvers in IR Tools

Methods relying on semi-convergence

In methods relying on semi-convergence mechanism, regularization can be enforced by terminating the iteration before its convergence to the unregularized and undesired least squares solution. Some examples of iterative methods in IR tools based on semi-convergence are listed below:

– `IRart` - The algebraic reconstruction technique also known as Kaczmarz's method.

– `IRcgls` - The conjugate gradient least squares (CGLS) algorithm applied implicitly to the normal equations.

– `IRnnfcgls` - The implementation of the flexible CGLS algorithm that ensures convergence to a non-negative solution.

Methods solving penalized LS problems

Some iterative methods relying on semi-convergence can also be used to solve the penalized least squares problems

$$\min_x \{\|Ax - b\|_2^2 + \lambda^2 \Omega(x)\}$$

with $\Omega(x) = \|Lx\|_2^2$ (i.e. Tikhonov Regularization). In this case, such solvers ignore the semi-convergence mechanism and instead, rely on the convergence to the penalized solution [7]. For example,

- `IRcgls` - When solving penalized least squares problems, the matrix L in $\|Lx\|_2^2$ is not required to be the identity matrix.
- `IRfista` - The first-order optimal method that solves the Tikhonov problem with box and/or energy constraints. L can only be the identity matrix [2].

Hybrid Krylov subspace methods

Penalized least squares problems can also be solved in hybrid Krylov subspace methods [5] [4]. In hybrid Krylov subspace methods, the penalization is moved from the "original problem" as shown above to the "projected problem" - the least squares problem restricted to the Krylov subspace.

There is one main advantage of doing so: the search for a good regularization parameter, λ, that controls the "smoothness" of the regularized solution, is done on the projected subproblem. This means the cost of computation is relatively small since the projected problem has relatively small dimensions and is less computationally demanding than the original large-scale problem. The regularization parameter in hybrid Krylov subspace problem is iteration dependent, and is adjusted as the Krylov subspace grows. When the discrepancy principle is ensured, the iteration stops. Following hybrid Krylov subspace methods are included in IR tools:

– `IRhybrid_lsqr` - Hybrid version of built-in `lsqr` function that applies a 2-norm penalty term to the projected problem based on the Krylov subspace [19].

The underlying `lsqr` method explicitly builds an orthonormal basis for this space, which allows us to easily formilate and solve the penalized projected problem. We will use this hybrid Krylov subspace method as our “inexpensive”, fast method to obtain an initial image reconstruction in Chapter 5.

4.2 Solve 2D Image Deblurring Problems

In IR tools, the iterative solvers are called as follows:

```
[X, Info] = IR___(A, b, K, options);
```

where `A` is the discrete forward operator and is of one of the forms: (1) a full or sparse matrix; (2) a user-defined function handle; (3) a matrix object that performs the matrix $\times$ vector operation. `b` is the measured data, probably with noise in it.

`K` and `options` are optional inputs. `K` is an integer vector that specifies which iterations during the process are returned and stored as columns in `X`. The maximum number of iterations is assumed to be the largest element in `K`. `options` is a structure that defines the algorithm parameters, including:

- `x0` - initial guess for the iterations, with a default value zero vector.
- `x_true` - true solution to the problem, which allows the function to return error norms (relative) with respect to the provided true solution at each iteration.
- `NoiseLevel` - the information that can be used along with the discrepancy rule to determine a good stopping iteration, without knowing the true image.
- `RegParam` - regularization parameter λ, to be employed if CGLS is used to solve the regularized problem.
- `NoStop` - specifies whether the iterations should proceed after a stopping criterion has been satisfied.

There are other parameters in the `options` structure, but we will not cover them in this thesis.

For the outputs, `X` is the matrix of computed solutions, stored columnwise, corresponding to the k^{th} iterations listed in `K`. Information about the behavior of the solver is stored in `Info`, such as:

- `its` - number of the last computed iteration.
- `StopFlag` - a string that describes the stopping conditions and is of one of the following: (1) reached max number of iterations; (2) residual tolerance (discrepancy principle) satisfied; (3) normal equation residual tolerance satisfied.
- `Enrm` - relative error norms at each iteration (requires `x_true`)
- `StopReg` - struct containing the information about the solution that satisfies the stopping criterion: (1) `X` - the solution satisfying the stop criterion; (2) `It` - iteration where the stopping criterion is satisfied; (3) `Enrm` - the best relative error.
- `BestReg` - struct containing the information about the solution that minimizes the relative error among all iterations (requires `x_true`): (1) `X` - the best solution; (2) `It` - iteration where the best solution is attained; (3) `Enrm` - best relative error that can be achieved.

Again, these are just some of the parameters stored in `Info`. For details, see [6].

If optional inputs are not provided, default values are used for maximum number of iterations (100), regularization parameters, and stopping criteria, depending on different solvers being called. In this case, `X` only contains the approximate solution at the final iteration. If additional information about the test problem is provided in `options`, additional information about the behavior of the iterative solver can be stored in the output structure `Info`. For instance, if the true solution to the problem, `x_true` is provided, then the relative errors can be computed at each iteration and returned in `Info.Enrm`, and the best regularized solution is also saved in `Info.BestReg.X`.

4.2.1 Example: Hubble Space Telescope Deblurring by CGLS Method

First, generate a speckle blur test problem, choose the default image of Hubble space telescope, set the blur level to mild, and then add noise to the data:

```
options = PRset('BlurLevel', 'mild');
[A, b, x, ProbInfo] = PRblurspeckle (options);
NoiseLevel = 0.01;
[bn, NoiseInfo] = PRnoise (b, NoiseLevel, 'gauss');
```

We begin the deblurring process by running `IRcgls` without specifying any optional parameters except the true image. So the maximum number of iterations is the default value (100), iteration continues until the stopping condition for `IRcgls` is reached, and the output `X` only contains the approximate solution at the final iteration.

```
options = IRset('x_true', x);
```

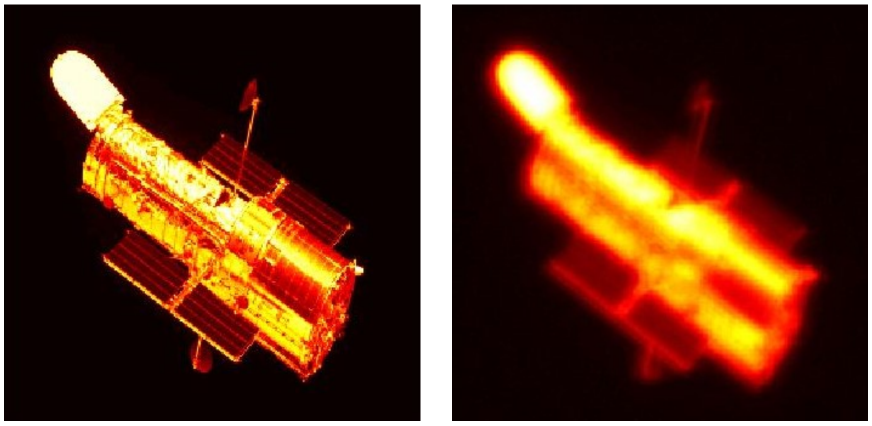


Figure 4.1: Example of an image deblurring problem, where the blur is caused by atmospheric turbulence. The blurred image with a mild blurring level and noise is shown on the right.

```
[xvec, infocgls] = IRcgls(A, bn, options);
```

We obtain the information about the behavior of the solver:

```
infocgls.its = 100

infocgls.BestReg.It = 23
```

It shows that the convergence criteria for `IRcgls` is not satisfied, so the solver reached the maximum number of iterations and thus `xvec` is a vector representation of the solution only at iteration 100. However, among all the 100 iterations, the solution with the minimum relative error to the true solution is obtained at iteration 23, which is saved in `Info.BestReg.X`.

A plot of the relative errors can be displayed, with the iteration where the optimal solution is reached being marked:

```
plot(infocgls.Enrm);
plot(infocgls.BestReg.It, min(infocgls.Enrm), 'mo');
```

In figure 4.2, the semi-convergence behavior of CGLS is clearly shown. We can observe that the smallest relative error is attained at iteration 23.

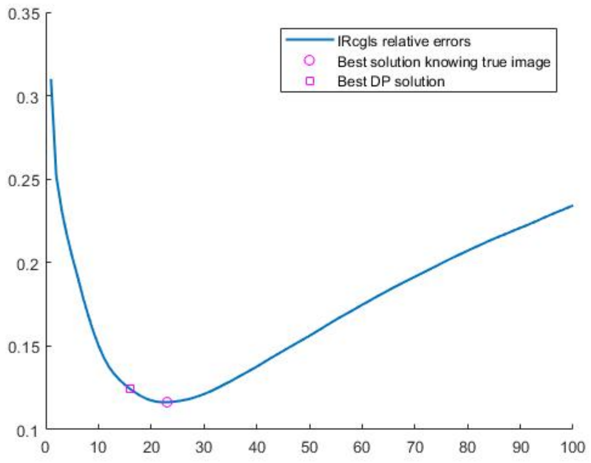


Figure 4.2: Relative error plot for image deblurring problem from the Hubble space telescope image with a mild speckle blur. The blue line shows the relative errors of `IRcgls` at each iteration, the magenta circle marks the iteration 23 where the best solution is achieved, and the magenta square marks the iteration 16 where the best solution based on discrepancy principle (true image is not provided) is obtained.

In the case when true image is not known, we are unable to compare the relative errors and find a best solution. However, we can use other schemes such as noise level to determine a good stopping iteration. Basically, the knowledge of noise level can be used along with the discrepancy principle to help determine a good stopping point for the iterations. Hence in this case, the iterative solver will not reach to the maximum number of iterations, but will stop at where the 'best solution' chosen by the `NoiseLevel` along with the discrepancy rule is obtained.

Compared to previous set of the `options`, we need to add information of the noise level:

```
options = IRset(options, ‘NoiseLevel’, 0.01);
[Xdp, infocgls_dp] = IRcgls(A, bn, options);
```

We obtain the information about the behavior of the solver:

```
infocgls_dp.its = 16
```

Notice this time, since `NoiseLevel` is applied to help determine a stopping iteration, the solver has only run for total 16 iterations. The output `Xdp` thus stores the solution at iteration 16.

Display the best solution with minimum relative error and the optimal solution based on discrepancy rule when true image is not provided:

```
PRshowx(infocgls.BestReg.X, ProbInfo);
PRshowx(xdp, ProbInfo);
```

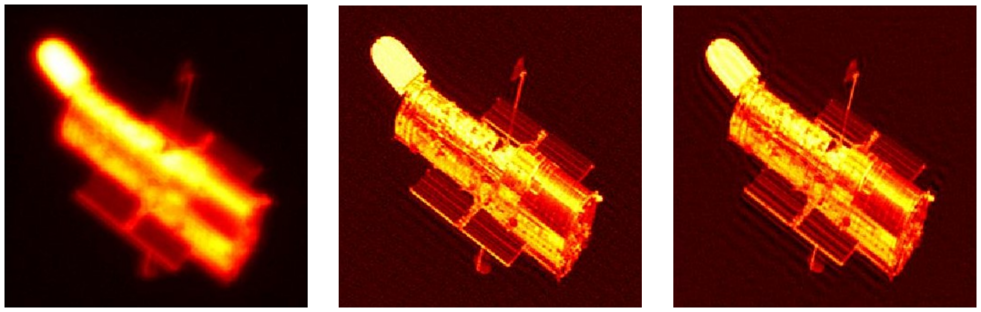


Figure 4.3: Blurred Hubble space telescope image, compared with a restored image using 23 iteration of `IRcgls`, and a restored image using 16 iteration of `IRcgls`.

4.2.2 Example: Hubble Space Telescope Deblurring by Hybrid Method

We can also restore the blurred Hubble space telescope image with the hybrid method. The hybrid method can recommend a regularization parameter, which is not provided by those non-hybrid methods such as `IRcgls` and `IRfista`. Hence the hybrid method is more efficient in the case when we need to have the method recommend a regularization parameter. In Chapter 5, we use a hybrid method in our initial reconstruction of the image.

Here, we restore the same Hubble space telescope image generated in section 4.2.1 using the hybrid method in IR tools, `IRhybrid_lsqr`, without changing the value of `options` set before:

```
[Xhybrid, infohybrid] = IRhybrid_lsqr(A, bn, options);
```

The method terminates at iteration 33, and the solution obtained at the recommended stopping point is thus saved in both `Xhybrid` and `infohybrid.StopReg.X`. However, from `x_true` we know that the optimal solution with the minimum relative error is attained at iteration 30, as indicated by `infohybrid.BestReg.It`.

Since this scheme enforces regularization at each iteration, it avoids the semi-convergence behavior as seen in `IRcgls`. Rather, iterations keep converging to a regularized solution and stop when the stopping criteria is

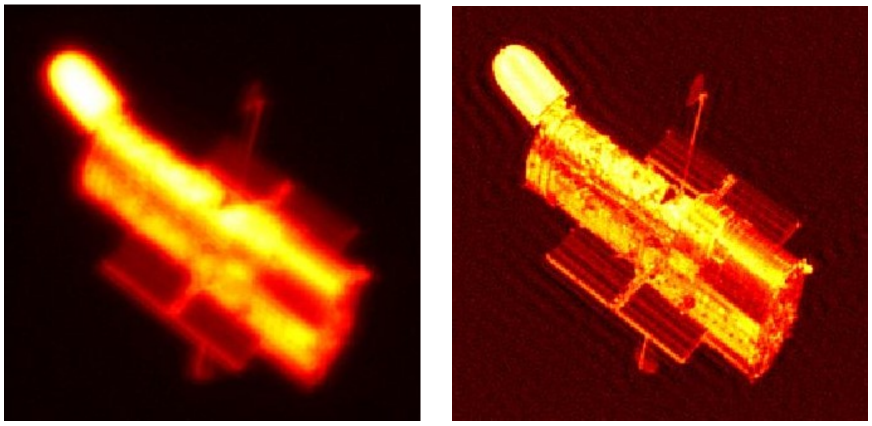


Figure 4.4: Blurred Hubble space telescope image, compared with a restored image using 33 iteration of `IRhybrid_lsqr` (the solution at the recommended stopping iteration).

satisfied (discrepancy rule). To show this method indeed avoids the semi-convergence behavior, we modify the `options` structure to make it run more iterations after the stopping condition is reached:

```
options = IRset(options, 'NoStop', 'on');
[Xhybrid2, infohybrid2] = IRhybrid_lsqr(A, bn, options);
```

With the `NoStop` option turned on, the iterations continue to default maximum number of 100, as indicated by `infohybrid2.its`. In this case, the vector `Xhybrid2` stores the solution at the final iteration 100. However, the solution where the discrepancy principle is satisfied occurs at the recommended stopping iteration 33, as indicated by `infohybrid2.StopReg.It`, and the solution is stored in `infohybrid2.StopReg.X`. What's more, because `x_true` is specified among the input options, the method is able to compute and compare the relative error norms and thus find the optimal solution with the minimum error to the true solution is attained at iteration 30, as indicated by `infohybrid2.BestReg.It` and stored in `infohybrid2.BestReg.X`.

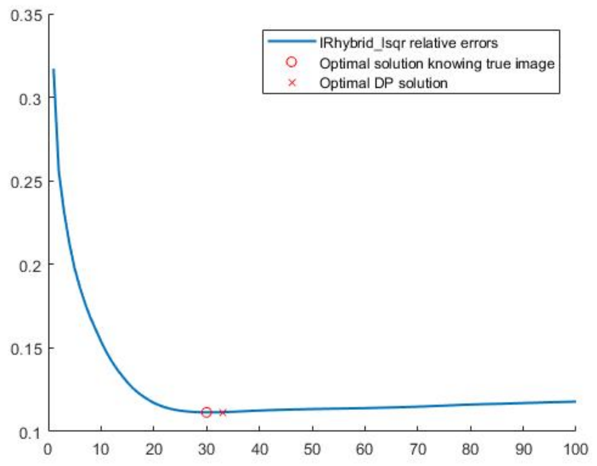


Figure 4.5: Relative error plot for image deblurring problem from the Hubble space telescope image with a mild speckle blur. The blue line shows the relative errors of `IRhybrid_lsqr` at each iteration, the red circle marks the iteration 30 where the best solution is achieved, and the red cross marks the iteration 33 where the solution at recommended stopping iteration (based on discrepancy principle) is obtained.

Chapter 5

Region of Interest Computations

In chapter 4, we have discussed some iterative solvers in IR tools that can be used for image deblurring problems. The methods we have illustrated before are relatively inexpensive, and can be applied to the whole test image. However, in real applications, sometimes there may be the need to recover more details from the observed, blurred image, and thus further improvement with more effective methods is required. Although IR tools indeed include more effective restoration algorithms which can provide better reconstruction quality, it may be unrealistic to apply those methods to the entire image because of their cost, especially when the image is very large.

In this chapter, we claim that we can first obtain an initial restored image using those inexpensive methods. After a general reconstruction of the image, we can further improve the quality of a small part of the image, called the *region of interest* (ROI) area, in which we want more detailed reconstruction [17]. We develop the techniques to (1) extract ROI subimages, that is, to generate the corresponding PSF submatrices and subvectors; (2) apply more expensive solvers to those subimages; (3) put together multiple improved regions back to the initial restored image.

5.1 Extract ROI Subimages

Notice in order to extract a an ROI subimage which can be restored using a more expensive solver, we need to construct a subproblem of the form

$$A_s x_s = b_s$$

where A_s is the PSF submatrix corresponding to the subimage, b_s is the observed subimage (probably with some noise). We need to obtain A_s and b_s first from the original problem so that they can be provided as inputs when we call the iterative solver.

Notice that for large scale problems, the matrix A is not formed explicitly. What's more, since the matrix plays an effect on the entire true image, extracting a submatrix from it which works only on the subimage is not a trivial operation.

5.1.1 Construction of the Subproblem

Assume the original image has dimension $n \times n$. Consider the problem:

$$Ax = b$$

- A - PSF matrix, has size $N \times N$, where $N = n^2$
- b - vector representation of measured image, has size $N \times 1$
- x - unknown true image, has size $N \times 1$

Also, assume the subimage has dimension $r \times c$, and thus

- b_s - vector representation of measured subimage, has size $S \times 1$, where $S = rc$
- x_s - unknown true subimage, has size $S \times 1$

Let E be a matrix with size $N \times S$. Each column of E is a unit vector (with only one nonzero entry equal to 1) corresponding to a pixel in the true subimage, that is:

$$E^T x = x_s$$

where each row of E^T extracts one entry from x to form one entry in x_s.

Similarly, let $\bar{E}$ with size $N \times (N - S)$ have columns corresponding to the pixels not in the subimage:

$$\bar{E}^T x = x_t$$

It can be shown that

$$\begin{bmatrix} E & \bar{E} \end{bmatrix} \begin{bmatrix} E^T \\ \bar{E}^T \end{bmatrix} = I$$

so we can transform the original problem into:

$$Ax = (A \begin{bmatrix} E & \bar{E} \end{bmatrix})(\begin{bmatrix} E^T \\ \bar{E}^T \end{bmatrix} x) = b$$

$$Ax = \begin{bmatrix} \hat{A}_s & \hat{A}_t \end{bmatrix} \begin{bmatrix} x_s \\ x_t \end{bmatrix} = b$$

where $\hat{A}_s = AE$, and $\hat{A}_t = A\bar{E}$. Therefore, the subproblem we want to construct is as follows:

$$\hat{A}_s x_s = b - \hat{A}_t x_t \tag{1}$$

where $b - \hat{A}_t x_t$ on the right hand side is the b_s.

Constructing a smaller subproblem

Now, suppose we can partition the original problem $Ax = b$ as follows:

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

If x_2 is our region of interest, based on the subproblem equation we have discussed before, the problem can be written as:

$$\begin{bmatrix} A_{12} \\ A_{22} \\ A_{32} \end{bmatrix} x_2 = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} - \begin{bmatrix} A_{11} & A_{13} \\ A_{21} & A_{23} \\ A_{31} & A_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_3 \end{bmatrix}$$

However, based on the property of partitioning of a matrix, we can continue to construct a smaller subproblem of the form:

$$A_{22}x_2 = b_2 - A_{21}x_1 - A_{23}x_3$$

Suppose a good initial approximation of the unknown true image can be computed, and x_1 and x_3 have the values $\hat{x}_1$ and $\hat{x}_3$ respectively, we can rewrite the above equation as:

$$A_{22}x_2 = b_2 - A_{21}\hat{x}_1 - A_{23}\hat{x}_3 \tag{2}$$

Therefore, $A_s = A_{22}$, $x_s = x_2$, and $b_s = b_2 - A_{21}\hat{x}_1 - A_{23}\hat{x}_3$.

5.1.2 Compute the Subvector

We begin by constructing the subvector, b_s, corresponding to the observed subimage. Let $\hat{E}$ be a matrix, similarly defined to the matrix E, but each column is a unit vector corresponding to a row of A_{22}. That is:

$$\hat{E}^T \hat{A}_s = A_{22}$$

where each row of $\hat{E}^T$ extracts a row from $\hat{A}_s$ to form A_{22}.

Then, after an initial reconstruction of the image, where $\hat{x}$ is the computed solution, we can obtain the b_s in this way:

(1) Map the computed solution $\hat{x}$ to a vector z in such a way that:

– $z(i) = 0$ if in the subimage

– $z(i) = \hat{x}(i)$ if outside the subimage

This can be achieved through the following operation:

$$z = \begin{bmatrix} E & \bar{E} \end{bmatrix} \begin{bmatrix} 0 \\ \bar{E}^T \end{bmatrix} \hat{x} = \begin{bmatrix} E & \bar{E} \end{bmatrix} \begin{bmatrix} 0 \\ \hat{x}_t \end{bmatrix}$$

Notice $[E \quad \bar{E}]$ works as a permutation matrix that makes entries in z have a correct order.

(2) Compute $\hat{b} = b - Az$, that is:

$$\hat{b} = b - A \begin{bmatrix} E & \bar{E} \end{bmatrix} \begin{bmatrix} 0 \\ \bar{E}^T \end{bmatrix} \hat{x} = b - \begin{bmatrix} \hat{A}_s & \hat{A}_t \end{bmatrix} \begin{bmatrix} 0 \\ \bar{E}^T \end{bmatrix} \hat{x} = b - \hat{A}_t x_t$$

Recall in equation (1), we have $\hat{A}_s x_s = b - \hat{A}_t x_t$. Hence $\hat{b} = \hat{A}_s x_s$, and we have obtained the right hand side of the equation as follows:

$$\begin{bmatrix} A_{12} \\ A_{22} \\ A_{32} \end{bmatrix} x_2 = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} - \begin{bmatrix} A_{11} & A_{13} \\ A_{21} & A_{23} \\ A_{31} & A_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_3 \end{bmatrix}$$

(3) Compute $b_s = \hat{E}^T \hat{b}$.

Recall $\hat{E}^T$ is defined in such a way that $\hat{E}^T \hat{A}_s = A_{22}$. Hence we have $\hat{E}^T \hat{b} = \hat{E}^T \hat{A}_s x_s = A_{22} x_s$, which is exactly the left hand side of the equation (2):

$$A_{22} x_2 = b_2 - A_{21} \hat{x}_1 - A_{23} \hat{x}_3$$

Therefore, $b_s = b_2 - A_{21} \hat{x}_1 - A_{23} \hat{x}_3$ is obtained.

5.1.3 Compute the Submatrix

Now we turn to construct the PSF submatrix, A_s, which works on the subimage. According to equation (2) which describes the subproblem we want to solve, $A_s = A_{22}$. Hence A_{22} is what we try to compute. Recall the definition of $\hat{E}$:

$$\hat{E}^T \hat{A}_s = A_{22}$$

Since $\hat{A}_s = AE$, we can get

$$A_{22} = \hat{E}^T \hat{A}_s = \hat{E}^T A E$$

Claim: It can be shown that A, E, and $\hat{E}^T$ can all be decomposed into a *Kronecker product* [13] [14].

- **Kronecker product.** If C is an $m \times n$ matrix, and D is a $p \times q$ matrix, then the *Kronecker product* of C and D is the $mp \times nq$ matrix of the form:

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

(1) For spacially invariant blurs, it is given that the $N \times N$ matrix A can be decomposed into [11]:

$$A = \sum_{i=1}^{k} C_i \otimes D_i$$

where $k = rank(P)$, P is a $p \times p$ matrix containing the coefficients of the PSF. C_i and D_i are banded $n \times n$ Toeplitz matrices.

(2) Recall E is an $N \times S$ matrix which extracts x_s from x. When it comes to the two-dimensional image X where $x = \texttt{vec}(X)$, we expect E of another form to extract the submatrix X_s from X. Therefore, E has a Kronecker product decomposition [10] [16]:

$$E = E_p \otimes E_q$$

where E_p is an $n \times r$ matrix with each column a unit vector corresponding to a row of X_s, and E_q is an $n \times c$ matrix with each column a unit vector corresponding to a column of X_s.

The decomposition above can be illustrated by a simple example. Recall the Kronecker product has the following properties:

- $(A \otimes B)^T = A^T \otimes B^T$
- $(A \otimes B)(C \otimes D) = AC \otimes BD$
- $\boldsymbol{y} = (A \otimes B)\boldsymbol{x} \Leftrightarrow Y = BXA^T$ where $\boldsymbol{x} = \texttt{vec}(X)$;

Since $x_s = E^T x$, replacing E with its Kronecker product form yields:

$$x_s = (E_p \otimes E_q)^T x$$
$$x_s = (E_p^T \otimes E_q^T) x$$
$$X_s = E_q^T X E_p$$

Suppose $n = 4$, $r = 2$, $c = 2$:

$$X = \begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} \\ x_{21} & x_{22} & x_{23} & x_{24} \\ x_{31} & x_{32} & x_{33} & x_{34} \\ x_{41} & x_{42} & x_{43} & x_{44} \end{bmatrix} \qquad X_s = \begin{bmatrix} x_{22} & x_{23} \\ x_{32} & x_{33} \end{bmatrix}$$

Then E_p is a 4×2 matrix of the form:

$$E_p = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}$$

where each column corresponds to one row of X_s, and E_q^T is a 2×4 matrix of the form:

$$E_q^T = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

where each row corresponds to a column of X_s.

(3) Analogous to the Kronecker product decomposition of E, the matrix $\hat{E}$ also has a Kronecker product decomposition:

$$\hat{E} = \hat{E}_p \otimes \hat{E}_q$$

Therefore, here is how we compute the submatrix A_s

$$\begin{aligned} A_s &= \hat{E}^T A E \\ &= (\hat{E}_p \otimes \hat{E}_q)^T (\sum_{i=1}^{k} C_i \otimes D_i)(E_p \otimes E_q) \\ &= \sum_{i=1}^{k} (\hat{E}_p^T C_i E_p) \otimes (\hat{E}_q^T D_i E_q) \end{aligned}$$

5.2 ROI Experiments

5.2.1 Simple Square ROI Computation

We begin by a generating a test problem using IR tools. For simplicity, we specify the size of the image to be relatively small (256×256):

```
options = PRset('trueImage', 'satellite');
[A, b, x, ProbInfo] = PRblurspeckle(256, options);
[bn, NoiseInfo] = PRnoise(b);
```

Next, we obtain an initial restored image using an inexpensive method, such as the hybrid method discussed in Chapter 4:

```
[xinitial, IterInfo] = IRhybrid_lsqr(A, bn);
```

Other than `PRshowx` and `PRshowb`, we can also display the image by converting the vector back to the two-dimensional image with the built-in MATLAB function `reshape`, where the size vector of the matrix `sz` is provided in

`ProbInfo.xSize` and `ProbInfo.bSize`, and display the data using `imagesc`. Such as:

```
xImage = reshape(xinitial, ProbInfo.xSize);
imagesc(xImage);
```

We can also change the colormap of the displayed image with function `colormap`.

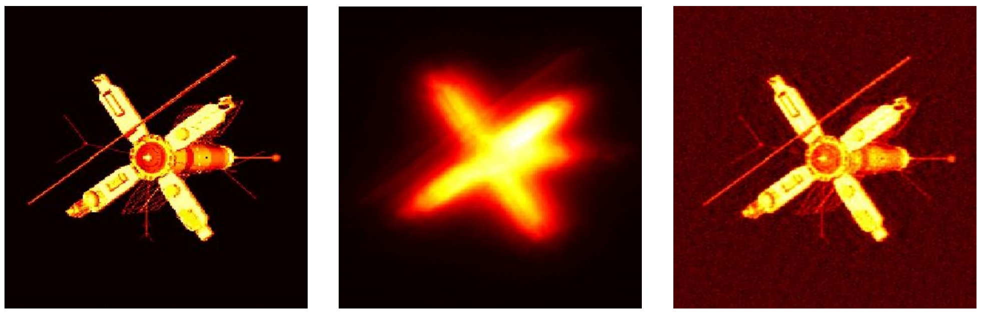


Figure 5.1: The satellite image and its intial reconstruction using inexpensive method.

Next, we specify a square as our region of interest area and want to further improve the restoration quality of that region. We draw one arbitrary point on the image as the center of the square using MATLAB function `ginput`. Desired side length of subimage is also provided. In the function, we check if the square is within the boundary. The coordinates of four corners of the subimage will be returned to the vector `region` only if the square is in the image. For example, we draw a 100×100 square centered at $(84, 172)$ in the image.

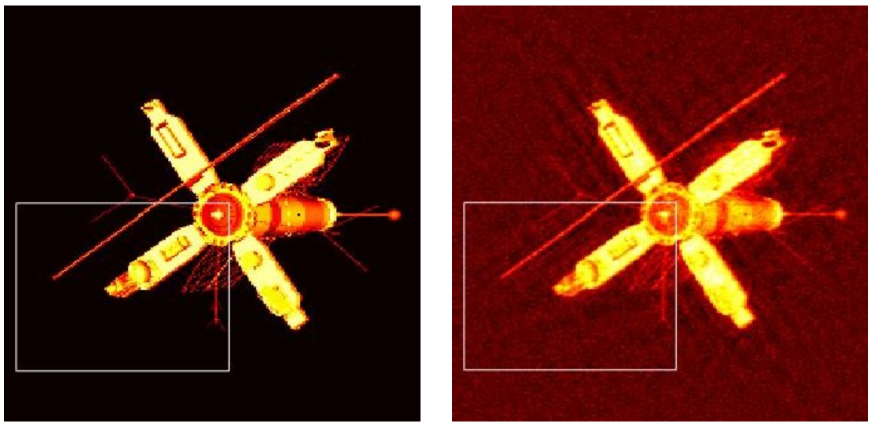


Figure 5.2: A square in the original image as the region of interest area. We will further improve the reconstruction of this subimage with another more expensive and effective method.

Afterwards, we use the coordinates stored in `region` to obtain the two-dimensional subimage, and according to the subimage, we compute the subvector `bs` and submatrix `As` following the steps discussed in Section 5.1.2 and 5.1.3.

After we compute the subvector and submatrix, we can then apply more expensive solvers to the subimage. For instance, we can use the modified flexible CGLS for nonnegativity constrained LS problems - `IRnncgls` in this step:

```
xSubImage = xImage(region(1):region(2),region(3):region(4));

[xs, IterInfo2] = IRnnfcgls(As, bs);

SubImage = reshape(xs, size(xSubImage));
```

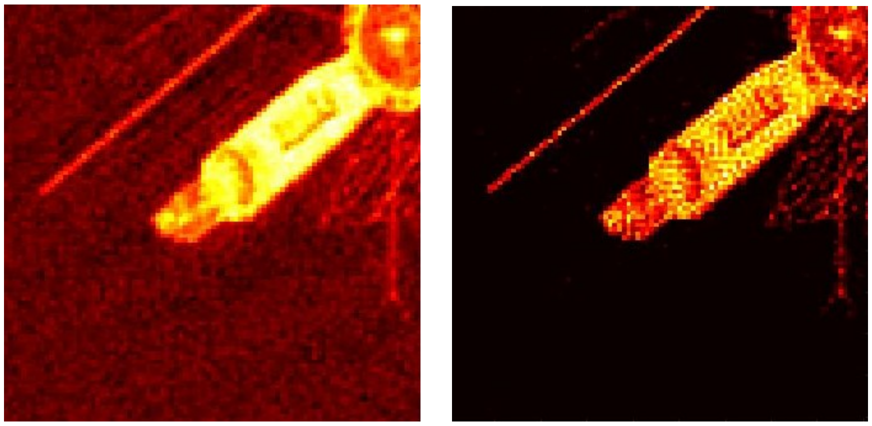


Figure 5.3: The subimage after the initial reconstruction, compared with the subimage after a further restoration using `IRnncgls` shown on the right.

We can "embed" the improved subimage back to the initial restored image (rescaling of the colormap is required), which will be covered in another example in Section 5.2.3.

5.2.2 Singular Value Analysis at Different Locations

Recall the PSF matrix is ill-conditioned, and the singular values of the matrix smoothly "decay" to zero, making it hard to get the real matrix. Generally, the faster the decay of the singular value, the harder to get the real image. However, this is only for the "decay" case. If singular values of the matrix suddenly "drop" very close to zero (that is, there is a clear "cutoff" between large and small singular values), then the problem is much easier to solve.

In this experiment, we analyze how the singular values of the submatrix change as the location and size of the subimage varies. Using the same test problem generated in Section 5.2.1, we draw three arbitrary points on the image as three different centers of the ROI squares with MATLAB function `gintput`. At each center, we set the initial size of the square as 11×11. Again, in the function obtaining the subimages, we check if the initial squares are within the boundary. If any of the squares is outside the image, we draw the three points again. The coordinates of four corners of each of the three ROI squares will be stored in `region`, where each row represents coordinates of one square. For instance, the three centers of the subimages are $(90, 162)$, $(135, 123)$, $(164, 190)$ respectively.

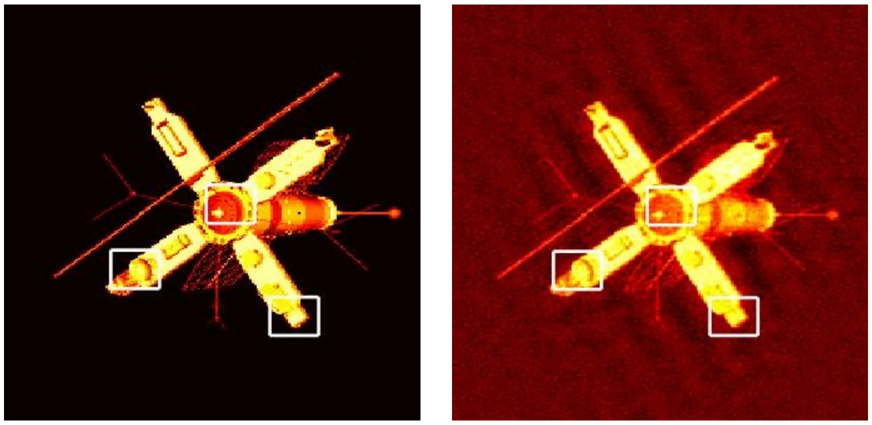


Figure 5.4: The three initial 11×11 ROI squares centered at $(90, 162)$, $(135, 123)$, $(164, 190)$ respectively.

Next, for each of the three locations, we increases the size of the square from 11×11 to 21×21 with an increasing rate of the side length equal to

2. At different sizes, we first check if the current square is within the image. If any of them is outside the image, we will stop the iteration. Otherwise, we continue to compute the submatrix from the current subimage, input the submatrix to the MATLAB function `full` to convert it to a full storage organization, conduct singular value decomposition on this matrix with `svd`, and plot its singular values. Here is what we obtain:

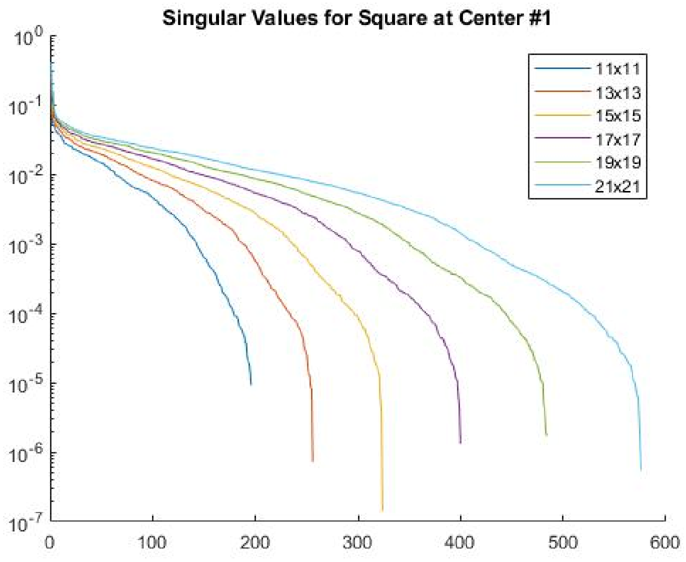
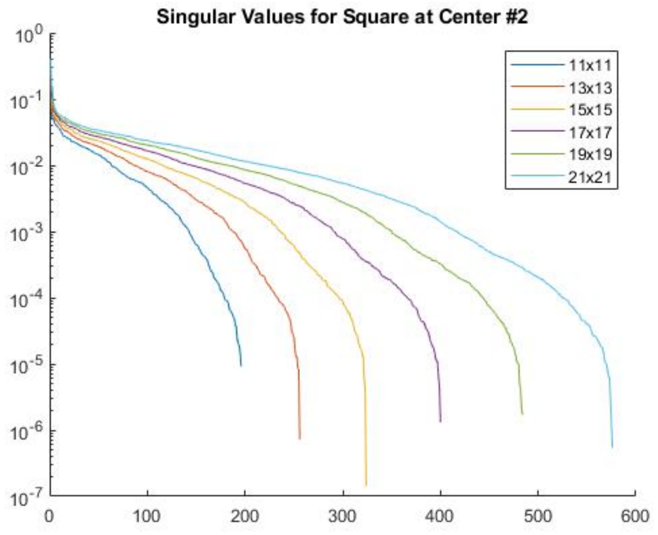


Figure 5.5: The singular values of the submatrices corresponding to subimages centered at $(90, 162)$. As the size of the subimage increases, compared to the smaller one, the singular values of the submatrix of larger subimage generally tend to decay flatter and slower. However, after some specific point, their singular values suddenly decay much faster to very close to zero.

As it shows in Figure 5.5, we can see a general trend of the decay of singular values of submatrix, corresponding to different size of the subimage. In all cases, it seems there is no clear “cutoff” between large and small singular values. The singular values of submatrices corresponding to larger subimages tend to decay slower at first, but after some specific point, they suddenly decay much faster to very close to zero. This change in decay pattern makes the comparison more complicated. Hence smaller subimage does not necessarily imply that the real image is easier to be restored.

When we compare the singular values at different locations, however, there is no obvious difference. In fact, singular values of submatrices corresponding to the same size of subimage at different locations seem to have nearly the same decay. What’s more, they all present the similar trend as size of the subimage increases, shown in Figure 5.6.



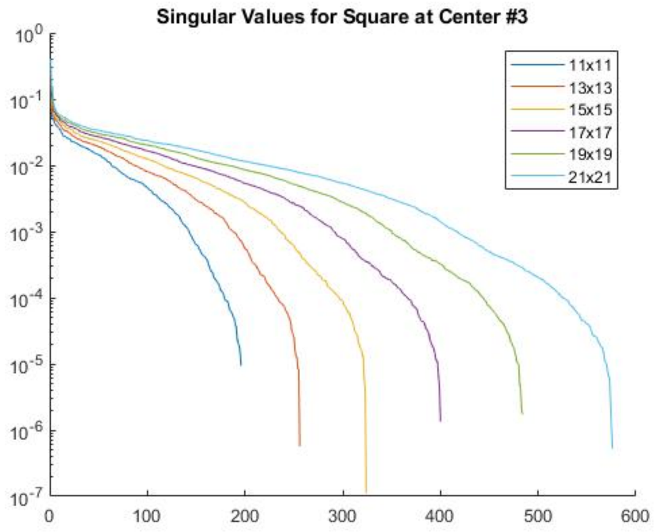


Figure 5.6: The singular values decay of submatrices corresponding to subimages centered at $(135, 123)$, $(164, 190)$ respectively.

5.2.3 Multiple ROI Computation

Notice that the ROI does not necessarily need to be the square shape. Based on previous experiments, we can design more complicated shaped ROIs and obtain the restored subimage. For instance, we can design a "Z" shaped region of interest area, which consists of three separate rectangle regions combined together:

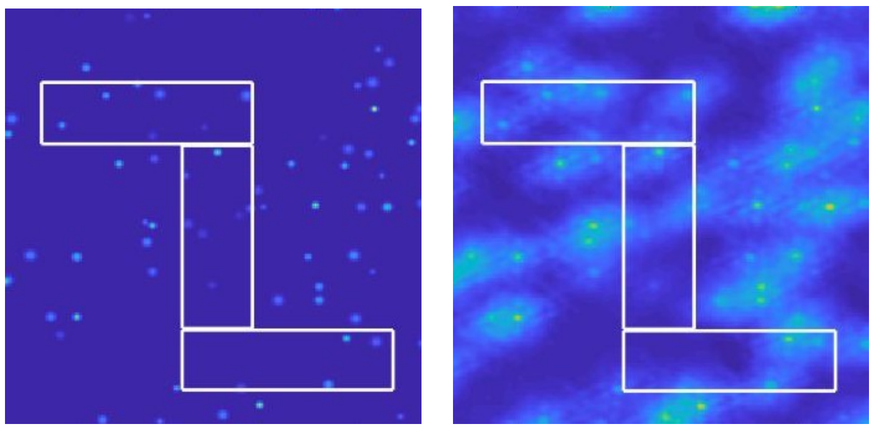


Figure 5.7: The true image with Z-shape ROI box on it, compared to the initially reconstructed image shown on the right.

In the function we designed, we draw two points on the image with `ginput` to specify the upper left corner and lower right corner of the "Z". Then the subimage is automatically generated according to some prescribed scale. For each of the three subimages, we again apply a more expensive solver such as `IRnncgls` to them to obtain an improved subimage:

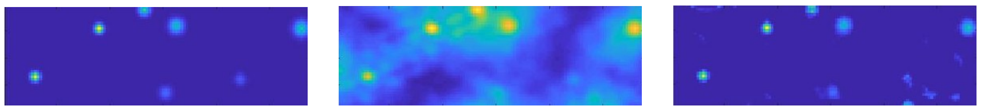


Figure 5.8: The true image of the first rectangle of the Z-shape ROI, the initially reconstructed subregion, compared to the improved subimage shown on the right.

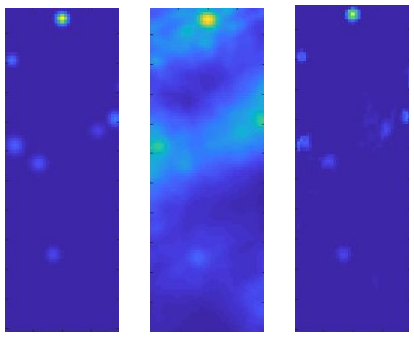


Figure 5.9: The true image of the second rectangle of the Z-shape ROI, the initially reconstructed subregion, compared to the improved subimage shown on the right.

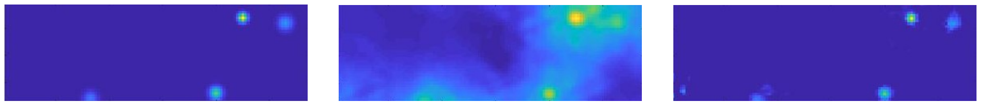


Figure 5.10: The true image of the third rectangle of the Z-shape ROI, the initially reconstructed subregion, compared to the improved subimage shown on the right.

After that, the new task is to "embed" the improved subregion back to the bigger initially reconstructed image. However, since the initially restored image and those subregions may not have the same color scale, in order to have a better visual effects, we need to rescale the color scales of the initially reconstructed image and all improved subimages to the true image before we put multiple regions all together. To achieve this, we scale the max value of pixels in each region to the max value of pixels in the original true image:

```
maxtrue = max(x);
maxintial = max(xImage(:));
xImage = maxtrue * xImage/maxinitial;
```

Suppose `xlsSubImage` is the improved subregion of one of the three rectangles that constitute the “Z”,

```
maxsub = max(xlsSubImage(:));
xlsSubImage = maxtrue * xlsSubImage/maxsub;
```

Once the rescaling is done, suppose the coordinates of four corners of the rectangle is stored in `reg = [rs, rf, cs, cf]`, we can then put `xlsSubImage` back into the initially restored image `xImage` by the following operation:

```
xImage2 = xImage;
xImage2(reg(1):reg(2), reg(3):reg(4)) = xlsSubImage;
```

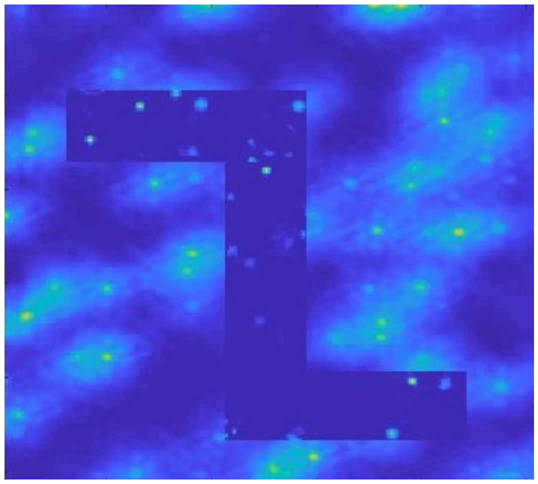


Figure 5.11: The resulting restored image of Gaussian dots after the initial reconstruction with inexpensive solver, combined with a Z-shape improved subregion solved by more expensive method.

In real applications, we can modify the shape of the region of interest area at our own ease, tailored to different needs. This will be very efficient, since the only work to do is the modification of the function in which we obtain the subimage(s) and return a vector `region` that stores coordinates of the subimage. Also, notice that since we do not apply the expensive method to the entire image but only apply it to specific subregion we are interested in, the reconstruction process is much more effective and efficient than those general image deblurring process, especially when the problem size is very large.

Chapter 6

Concluding Remarks

We have demonstrated how we can use the MATLAB package IR tools to generate image deblurring test problems and reconstruct the blurred image using various iterative solvers. After that, we proposed the idea that when we want a more detailed reconstruction of a blurred image, instead of applying the expensive method to the whole image after the initial restoration, which has a high cost and low efficiency, we can specify a region of interest (ROI) area. The ROI area is basically the part of the image we want more details, and we can then apply the more expensive solver only to this part. We have found efficient ways to construct the submatrix and subvector according to the subimage. Numerical experiments were conducted to demonstrate and compare the efficiency and quality of the image reconstruction.

In the experiments, we also analyzed how the singular values of the PSF matrix (submatrix) change as the subimage varies in sizes and locations. It shows that although the pattern of singular values decay does not present a significant difference when the same size of subimage is chosen at different locations, the pattern of decay does varies when the size of the subimage changes. As the size of the subimage increases, the decay generally tends to be flatter and slower. However, for most test sizes of the subimage, after some point, the singular values all suddenly decay to very close to zero. A smaller subimage does not necessarily imply the real image is easier to be restored, though.

We have extended the subimage restoration to multi-subregion reconstruction with more complicated shapes. We demonstrated how we "embedded" the improved image back to the initially restored image to make it a whole one. We also claimed that we can modify the shape of the ROI tailored to different needs, and this process is very efficient since we only need to modify the function in which we obtain the coordinates of the subregion(s). To conclude, ROI computation is much more effective and efficient than the general image reconstruction using one method for all, especially in large-scale image deblurring problems.

Bibliography

[1] M. R. Banham and A. K. Katsaggelos. Digital image restoration. *IEEE Signal Processing Magazine*, pages 24–41, 1997.

[2] A. Beck and M. Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM J. Imaging Sci.*, 2:183–202, 2009.

[3] R. Berry and J. Burnell. *The Handbook of Astronomical Image Processing*. Willmann-Bell Inc., Richmond, VA, 2000.

[4] D. Calvetti, L. Reichel, and A. Shuibi. Enriched krylov subspace methods for ill-posed problems. *Linear Algebra Appl.*, 362:257–273, 2003.

[5] J. Chung, J. G. Nagy, and D. P. O'Leary. A weighted GCV method for Lanczos hybrid regularization. *Elec. Trans. Numer. Anal.*, 28:149–167, 2008.

[6] S. Gazzola, P. C. Hansen, and J. G. Nagy. IR Tools: A MATLAB package of iterative regularization methods and large-scale test prob-

lems. *Numerical Algorithms*, https://doi.org/10.1007/s11075-018-0570-7, 2008.

[7] S. Gazzola and J. G. Nagy. Generalized Arnoldi-Tikhonov method for sparse reconstruction. *SIAM J. Sci. Comput.*, 36:B225–B247, 2014.

[8] P. C. Hansen. *Discrete Inverse Problems: Insight and Algorithms*, volume 7. SIAM, Philadelphia, PA, 2010.

[9] P. C. Hansen, J. G. Nagy, and D. P. O'Leary. *Deblurring Images: Matrices, Spectra, and Filtering.* SIAM: Fundamentals of Algorithms, 2006.

[10] J. Kamm and J. G. Nagy. Kronecker product and SVD approximations in image restoration. *Linear Algebra Appl.*, 284:177–192, 1998.

[11] J. Kamm and J. G. Nagy. Optimal Kronecker product approximation of block Toeplitz matrices. *SIAM J. Matrix Anal. Appl.*, 22:155–172, 2000.

[12] R. L. Lagendijk and J. Biemond. *Iterative Identification and Restoration of Images*. Kluwer Academic Publishers, Boston/Dordrecht/London, 1991.

[13] C. F. Van Loan. The ubiquitous Kronecker product. *J. Comp. Appl. Math.*, 123:85–100, 2000.

[14] C. F. Van Loan and N. P. Pitsianis. Approximation with Kronecker products. In M. S. Moonen and G. H. Golub, editors, *Linear Alge-*

bra for Large Scale and Real Time Applications, pages 293–314. Kluwer Publications, 1993.

[15] J. L. Mueller and S. Siltanen. *Linear and Nonlinear Inverse Problems with Practical Applications.* SIAM, Philadelphia, PA, 2012.

[16] J. G. Nagy, M. K. Ng, and L. Perrone. Kronecker product approximation for image restoration with reflexive boundary conditions. *SIAM J. Matrix Anal. Appl.*, 25:829–841, 2004.

[17] J. G. Nagy and D. P. O'Leary. Image restoration through subimages and confidence images. *Elec. Trans. Num. Anal.*, 13:22–37, 2002.

[18] National Research Council Institute of Medicine. *Mathematics and Physics of Emerging Biomedical Imaging.* National Academy Press, Washington D.C., 1996.

[19] C. C. Paige and M. A. Saunders. Lsqr: An algorithm for sparse linear equations and sparse least squares. *ACM: Transactions on Mathematical Software*, 8:43–71, 1982.

[20] Jon Van. Bringing fuzzy field into focus: Image help for the hubble. *Chicago Tribune*, 1991.

[21] Jon Van. Novel imaging systems rely on focus-free optics. *SIAM News*, 36, 2003.