Technical Report

TR-2012-013

# Iterative Methods for Image Restoration

by

S. Berisha, J. G. Nagy

Mathematics and Computer Science

EMORY UNIVERSITY

# Iterative Methods for Image Restoration

Sebastian Berisha and James G. Nagy
Department of Mathematics and Computer Science
Emory University
Atlanta, GA, USA
Email: `nagy@mathcs.emory.edu`

**Abstract**

Although image restoration methods based on spectral filtering techniques are very efficient, they can be applied only to problems with fairly simple spatially invariant blurring operators. Iterative methods, however, are much more flexible; they can be very efficient for spatially invariant as well as spatially variant blurs, they can incorporate a variety of regularization techniques and boundary conditions, and they can more easily incorporate additional constraints, such as nonnegativity. This chapter describes a variety of iterative methods used in image restoration, with a particular emphasis on efficiency, convergence behavior, and implementation. Discussion of MATLAB software implementing the methods is also provided.

# Contents

## 1 Introduction

Image restoration is the process of reconstructing an approximation of an image from blurred and noisy measurements. In this chapter, we use the standard linear image formation model,

$$g(x, y) = \int_{\mathcal{R}^2} k(x, y; \xi, \eta) f(\xi, \eta) d\xi d\eta + \eta(x, y) \,, \tag{1}$$

where $f$ is the true object, $g$ is the observed image, and $\eta$ is additive noise. The kernel function $k$ models the blurring operation, and is called the *point spread function* (PSF). In many applications the kernel satisfies $k(x, y; \xi, \eta) = k(x - \xi, y - \eta)$, and the blur is said to be spatially invariant, otherwise the blur is said to be spatially variant. In the case of spatially invariant blurs, the integration in equation (1) is a convolution operation, and thus the image restoration problem is often referred to as deconvolution.

Typically we do not have a precise function definition for $g$ because the observed image is recorded digitally, and thus is known only at discrete values. Moreover, in many cases it is necessary to estimate $k$ from measured data. Therefore, it is natural to consider the digital image restoration problem

$$\boldsymbol{g} = \boldsymbol{K} \boldsymbol{f}_{\text{true}} + \boldsymbol{\eta} \,, \tag{2}$$

which is obtained from equation (1) by discretizing the functions and approximating integration with a quadrature rule. If the images are assumed to have $m \times n$ pixels, then $\boldsymbol{K} \in \mathcal{R}^{mn \times mn}$ and $\boldsymbol{g}, \boldsymbol{f}, \boldsymbol{\eta} \in \mathcal{R}^{mn}$. Two aspects of the digital image restoration problem (2) make it computationally challenging:

- In most image restoration problems involving images with $m \times n$ pixels, $\boldsymbol{K}$ is an $N \times N$ matrix with $N = mn =$ number of pixels in the image[1]. For example, if $m = n = 10^3$, then $\boldsymbol{K}$ is a $10^6 \times 10^6$ matrix. Thus, the problem is large-scale. Fortunately the matrix $\boldsymbol{K}$ can usually be represented by a compact data structure, such as when the blur is spatially invariant. Approximation techniques for more complicated spatially variant blurs include geometrical coordinate transformations [62, 83, 87], sectioning [37, 97, 98], PSF interpolation [14, 33, 67, 68], and in some cases a sparse matrix data structure can be used for $\boldsymbol{K}$ [32, 80].
- The matrix $\boldsymbol{K}$ is severely ill-conditioned, with singular values decaying to, and clustering at 0. This means that *regularization* is needed to avoid computing solutions that are corrupted by noise. Regularization can be enforced through well-known techniques such as Wiener filtering and Tikhonov regularization, and/or by incorporating constraints such as nonnegativity [31, 47, 48, 102].

Efficient implementation of an image restoration algorithm is obtained by exploiting structure of the matrix $\boldsymbol{K}$. For example, if the blur is spatially invariant *and* we assume periodic boundary conditions, then $\boldsymbol{K}$ is a block circulant matrix with circulant blocks. In this case many image restoration algorithms, such as the Wiener filter, can be implemented in the Fourier domain, using fast Fourier transforms (FFT) [1, 49]. If spatial invariance is a poor approximation of the actual blur, or periodic boundary conditions are poor approximations to the actual true image scene, then the quality of reconstructions will be limited. Moreover, it is not possible to incorporate additional constraints, such as nonnegativity, into simple filtering methods.

[1]There are situations, such as multi-frame problems, where $\boldsymbol{K}$ is $M \times N$ with $M \neq N$. The algorithms discussed in this chapter are not restricted to square matrices, and can be applied to these situations as well.

Iterative image restoration algorithms have many advantages over simple filtering techniques [10, 59, 102]. Iterative methods can be very efficient for spatially invariant as well as spatially variant blurs, they can incorporate a variety of regularization techniques and boundary conditions, and can more easily incorporate additional constraints, such as nonnegativity [5, 70]. The cost of an iterative scheme depends on the amount of computation needed per iteration, as well as on the number of iterations needed to reach a good restoration of the image. Convergence can be accelerated using *preconditioning*, but if not done carefully, it can lead to erratic convergence behavior that results in fast convergence to a poor approximate solution. In this chapter, we describe a variety of iterative methods that can be used for image restoration, and also describe some preconditioning techniques that can be used to accelerate convergence. We show that many well-known iterative methods can be viewed as a basic method with a particular preconditioner. This viewpoint provides a natural mechanism to compare a variety of iterative methods.

## 2 Background

In this section, we present the essential background material needed to develop and implement iterative image restoration methods, focusing on approaches that have the following general form:

```
$\boldsymbol{f}_0$ = initial estimate of $\boldsymbol{f}_{\text{true}}$
for  $k = 0, 1, 2, \ldots$
    • $\boldsymbol{f}_{k+1}$ =  computations involving $\boldsymbol{f}_k$, $\boldsymbol{K}$
                  and other intermediate quantities
    • determine if stopping criteria are satisfied
end
```

Most well-known iterative methods have this basic form, including conjugate gradient type schemes, the expectation-maximization method (sometimes referred to as the Richardson-Lucy method), and many others. Since any one iterative method is not optimal for all image restoration problems, the study of iterative methods is an important and active area of research. The specific computational operations that are required to update $\boldsymbol{f}_{k+1}$ at each iteration depend on the particular iterative scheme being used, but the most intensive part of these computations usually involves matrix vector products with $\boldsymbol{K}$ and its transpose, $\boldsymbol{K}^\top$.

The rest of this section is outlined as follows. In Section 2.1, we provide a brief introduction to regularization, and in Section 2.2, we discuss efficient approaches to perform matrix-vector multiplications with $\boldsymbol{K}$ for various types of blurs, including spatially invariant and spatially variant. In Section 2.3, we describe a general preconditioning approach.

### 2.1 Regularization

The continuous image formation model (1) is useful for analysis purposes, since it is a classic example of an ill-posed inverse problem, and a substantial amount of research has been done to understand the properties of such problems; see, for example [10, 31, 48, 102]. We will not discuss these theoretical properties in detail, but we do mention that even in the continuous problem there may not be a unique function $f$ that solves equation (1) and small changes in the noise $\eta$ can cause arbitrarily large changes in $f$ [31, 42, 48, 102].

The discrete problem (2) inherits these properties, which may be summarized as follows. Define $\boldsymbol{g}_{\text{true}} = \boldsymbol{K}\boldsymbol{f}_{\text{true}}$ to be the noise-free blurred image, and let $\boldsymbol{K} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$ be the singular value decomposition of the $N \times N$ matrix $\boldsymbol{K}$. That is, $\boldsymbol{U}$ and $\boldsymbol{V}$ are orthogonal matrices and $\boldsymbol{\Sigma}$ is a diagonal matrix with entries satisfying $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_N \geq 0$. If $\boldsymbol{K}$ arises from discretizing the integral equation (1), then, assuming the problem is scaled so that $\sigma_1 = 1$, we have:

- The singular values, $\sigma_i$, decay to, and cluster at 0, without a significant gap to indicate numerical rank.
- The components $|\boldsymbol{u}_i^\top \boldsymbol{g}_{\text{true}}|$, where $\boldsymbol{u}_i$ is the $i$th column of $\boldsymbol{U}$, decay on average faster than the singular values $\sigma_i$. This is referred to as the *discrete Picard condition* [47].
- The singular vectors $\boldsymbol{v}_i$ (i.e., the columns of $\boldsymbol{V}$) corresponding to small singular values tend to have more oscillations than the singular vectors corresponding to large singular values.

With these properties we can easily see what effect the error term $\boldsymbol{\eta}$ has on the inverse solution:

$$\begin{aligned}
\boldsymbol{f}_{\text{inv}} &= \boldsymbol{K}^{-1}\boldsymbol{g} \\
&= \boldsymbol{K}^{-1}(\boldsymbol{g}_{\text{true}} + \boldsymbol{\eta}) \\
&= \boldsymbol{V}\boldsymbol{\Sigma}^{-1}\boldsymbol{U}^\top(\boldsymbol{g}_{\text{true}} + \boldsymbol{\eta}) \\
&= \sum_{i=1}^{N} \frac{\boldsymbol{u}_i^\top(\boldsymbol{g}_{\text{true}} + \boldsymbol{\eta})}{\sigma_i}\boldsymbol{v}_i \\
&= \sum_{i=1}^{N} \frac{\boldsymbol{u}_i^\top \boldsymbol{g}_{\text{true}}}{\sigma_i}\boldsymbol{v}_i + \sum_{i=1}^{N} \frac{\boldsymbol{u}_i^\top \boldsymbol{\eta}}{\sigma_i}\boldsymbol{v}_i \\
&= \boldsymbol{f}_{\text{true}} + \text{error}\,.
\end{aligned}$$

From this relation we can see that the high frequency components in the error are highly magnified by division of small singular values. The computed inverse solution is dominated by these high frequency components, and is in general a very poor approximation of the true solution, $\boldsymbol{f}_{\text{true}}$.

In order to compute an accurate approximation of $\boldsymbol{f}_{\text{true}}$, or at least one that is not horribly corrupted by noise, the solution process must be modified. This process is usually referred to as *regularization* [31, 42, 47, 102]. One class of regularization methods, called *filtering*, can be formulated as a modification of the inverse solution [47]. Specifically, a filtered solution is defined as

$$\boldsymbol{f}_{\text{filt}} = \sum_{i=1}^{N} \phi_i \frac{\boldsymbol{u}_i^\top \boldsymbol{g}}{\sigma_i}\boldsymbol{v}_i\,, \tag{3}$$

where the *filter factors*, $\phi_i$, satisfy $\phi_i \approx 1$ for large $\sigma_i$, and $\phi_i \approx 0$ for small $\sigma_i$. That is, the large singular value components of the solution are reconstructed, while the components corresponding to the small singular values are filtered out.

It is sometimes possible to replace the singular value decomposition with the alternative factorization,

$$\boldsymbol{K} = \boldsymbol{Q}^H \boldsymbol{\Lambda} \boldsymbol{Q}\,,$$

where $\boldsymbol{\Lambda}$ is a diagonal matrix, and $\boldsymbol{Q}^H$ is the Hermitian (complex conjugate) transpose of $\boldsymbol{Q}$, with $\boldsymbol{Q}^H\boldsymbol{Q} = \boldsymbol{I}$. We refer to this as a spectral value decomposition; the columns of $\boldsymbol{Q}$ are eigenvectors and the diagonal elements of $\boldsymbol{\Lambda}$ are the eigenvalues of $\boldsymbol{K}$. Although every matrix has an SVD, only *normal matrices* (i.e., matrices that satisfy $\boldsymbol{K}^\top \boldsymbol{K} = \boldsymbol{K}\boldsymbol{K}^\top$) have a spectral value decomposition. However, if $\boldsymbol{K}$ has a spectral factorization, then it can be used, in place of the singular value decomposition, to implement the filtering methods described above. The advantage is that it is sometimes more computationally convenient to compute a spectral value decomposition than it is a singular value decomposition; for example in the case of a spatially invariant blur with periodic boundary conditions, the eigenvalues are obtained by taking a Fourier transform of the PSF (this is often also referred to as the optical transfer function, or OTF), the eigenvectors of $\boldsymbol{K}$ are the Fourier vectors, and fast Fourier transforms (FFT) can be used for operations involving $\boldsymbol{Q}$ (forward Fourier transform) and $\boldsymbol{Q}^H$ (inverse Fourier transform) [1, 49].

Generally we will not distinguish between a singular or spectral value decomposition, and use the generic acronym SVD to mean methods based on one or the other. We remark that different choices of filter factors lead to different methods; popular choices are the pseudo-inverse (or truncated SVD), Tikhonov, and Wiener filters [47, 50, 102]. Although the focus of this chapter is on problems for which it is not computationally feasible to compute an SVD of $\boldsymbol{K}$, we do consider using SVD approximations to construct preconditioners; this will be discussed in more detail in Section 2.3.

If it is not computationally feasible to use an SVD based filtering method then it may be preferable to use a *variational* form of regularization, which amounts to computing a solution of

$$\min_{\boldsymbol{f}} \left\{ \|\boldsymbol{g} - \boldsymbol{K}\boldsymbol{f}\|_2^2 + \lambda^2 \mathcal{J}(\boldsymbol{f}) \right\} , \tag{4}$$

where the regularization operator $\mathcal{J}$ and the regularization parameter $\lambda$ must be chosen. The variational form provides a lot of flexibility. For example, one could include additional constraints on the solution, such as nonnegativity, or it may be preferable to replace the least squares criterion with the Poisson log likelihood function [3, 4, 6]. As with filtering, there are many choices for the regularization operator, $\mathcal{J}$, such as:

- Tikhonov regularization [42, 95, 94, 96]: $\mathcal{J}(\boldsymbol{f}) = \|\boldsymbol{L}\boldsymbol{f}\|_2^2$
- Total variation [24, 84, 102]: $\mathcal{J}(\boldsymbol{f}) = \left\| \sqrt{(\boldsymbol{D}_{\mathrm{h}}\boldsymbol{f})^2 + (\boldsymbol{D}_{\mathrm{v}}\boldsymbol{f})^2} \right\|_1$
- Sparsity constraints [21, 36, 99]: $\mathcal{J}(\boldsymbol{f}) = \|\boldsymbol{\Phi}\boldsymbol{f}\|_1$

Tikhonov regularization, which was first proposed and studied extensively in the1960's and 1970's [63, 78, 94, 95, 96], is perhaps the most well known approach to regularizing ill-posed problems. $\boldsymbol{L}$ is typically chosen to be the identity matrix, or a discrete approximation to a derivative operator, such as the Laplacian. If $\boldsymbol{L} = \boldsymbol{I}$, then it is not difficult to show that the resulting variational form of Tikhonov regularization, namely

$$\min_{\boldsymbol{f}} \left\{ \|\boldsymbol{g} - \boldsymbol{K}\boldsymbol{f}\|_2^2 + \lambda^2 \|\boldsymbol{f}\|_2^2 \right\} , \tag{5}$$

can be written in an equivalent filtering framework by replacing $\boldsymbol{K}$ with its SVD [47].

In the case of total variation regularization, $\boldsymbol{D}_{\mathrm{h}}$ and $\boldsymbol{D}_{\mathrm{v}}$ denote discrete approximations of horizontal and vertical derivatives of the 2D image $\boldsymbol{f}$, and the approach extends to 3D images in an obvious way. Efficient and stable implementation of total variation regularization is a nontrivial problem; see [24, 102] and the references therein for further details.

In the case of sparse reconstructions, the matrix $\boldsymbol{\Phi}$ represents a basis in which the image, $\boldsymbol{f}$, is sparse. For example, for astronomical images that contain a few bright objects surrounded by a significant amount of black background, an appropriate choice for $\boldsymbol{\Phi}$ might be the identity matrix. Clearly, the choice of $\boldsymbol{\Phi}$ is highly dependent on the structure of the image $\boldsymbol{f}$. The usage of sparsity constraints for regularization is currently a very active field of research, with many open problems [36].

Iterative methods applied directly to the least squares problem

$$\min_{\boldsymbol{f}} \|\boldsymbol{g} - \boldsymbol{K}\boldsymbol{f}\|_2$$

offer yet another choice of enforcing regularization. If the iterative method is applied directly to this least squares problem then the iteration converges to $\boldsymbol{f}_{\text{inv}} = \boldsymbol{K}^{-1}\boldsymbol{g}$, which, as was discussed above, is typically a poor approximation of $\boldsymbol{f}_{\text{true}}$. Fortunately, many iterative methods exhibit a *semi-convergence* behavior with respect to the relative error,

$$\frac{\|\boldsymbol{f}_k - \boldsymbol{f}_{\text{true}}\|_2}{\|\boldsymbol{f}_{\text{true}}\|_2}, \tag{6}$$

where $\boldsymbol{f}_k$ is the approximate solution at the $k$th iteration. Although this is discussed in more detail in Section 4, here it is worth mentioning that the term semi-convergence refers to the property that in the early iterations the relative error begins to decrease and, after some "optimal" iteration, the error then begins to increase. By stopping the iteration when the error is low, we obtain a regularized approximation of the solution. Because the true object, $\boldsymbol{f}_{\text{true}}$, is not known, the relative error (6) cannot be used as a practical measure for determining a stopping iteration, and alternative approaches, such as methods based on knowledge of the noise (e.g., discrepancy principle) must be used.

As was mentioned in the previous paragraph, in the case of iterative regularization, it is necessary to have a practical approach to determine an appropriate stopping iteration. In fact, each of the regularization methods discussed above require choosing a *regularization parameter*. For filtering methods, one has to decide what is meant by "large" and "small" singular values. In the variational form, one has to choose the scalar $\lambda$, where as for iterative regularization methods the index where the iteration is terminated plays the role of the regularization parameter. In each case, it is a nontrivial matter to choose an "optimal" regularization parameter. Although there are methods (e.g., generalized cross validation [38], discrepancy principle [31], L-curve [47], L-ribbon [17], and many others [31, 47, 102]) that may be used to estimate the regularization parameter, it may be necessary to solve the problem for a variety of parameters, and use knowledge of the application to help decide which solution is best.

We also mention that when the majority of the elements in the image $\boldsymbol{f}$ are zero or near zero, as may be the case for astronomical or medical images, it may be wise to enforce nonnegativity constraints on the solution [4, 6, 102]. This requires that each element of the computed solution $\boldsymbol{f}$ is not negative, which is often written as $\boldsymbol{f} \geq \boldsymbol{0}$. Though these constraints add a level of difficulty when solving, they can produce results that are more feasible than when nonnegativity is ignored. Some iterative methods that enforce nonnegativity will be discussed in Section 5.

## 2.2 Matrix Structures and Matrix-Vector Multiplications

The efficiency of iterative methods depends on the number of iterations needed to compute the desired solution, as well as on the cost required for each iteration. The computational cost of each iteration can depend on many factors, but all methods will require matrix-vector multiplication with $\boldsymbol{K}$, and often also with its transpose $\boldsymbol{K}^\top$. Because the size of $\boldsymbol{K}$ is extremely large, an efficient storage scheme, exploiting structure and sparsity, should be used. In this subsection we briefly discuss structures that arise most often in image restoration problems.

In most of our discussion, we assume images are represented as vectors, for example by storing the pixel values in lexicographical order. This is mathematically convenient when describing algorithms using linear algebra notation. However, we will sometimes need to think of images as 2-dimensional arrays (i.e., $m \times n$ matrices). It will therefore be helpful to have notation that describes the transformation from image array to vector, and from vector to image array. Specifically, if $F$ is an $m \times n$ image array of pixels, then we use the notation

$$\nu(F) = \boldsymbol{f} \in \mathcal{R}^{mn}$$

to transform the image array into a vector with $mn$ elements. Similarly, we will use $\mu(\cdot)$ to denote the inverse transformation from a vector to an array,

$$\mu(\boldsymbol{f}) = F \in \mathcal{R}^{m\times n}\,.$$

Of course, $\boldsymbol{f} = \nu(\mu(\boldsymbol{f}))$ and $F = \mu(\nu(F))$.

### 2.2.1 Boundary Conditions

Pixels near the edges of a blurred image are likely to have been affected by information outside the field of view. This situation often causes ringing artifacts in deblurred images. These ringing artifacts can be reduced by incorporating *boundary conditions*, which are are used to approximate the image scene outside the field of view. Boundary conditions can be incorporated explicitly in the matrix $\boldsymbol{K}$, but for iterative methods it is usually easier to pad images before performing the matrix-vector multiplication. Commonly used boundary conditions include:

- *Periodic* boundary conditions imply that the image repeats itself in all directions.
- *Zero* boundary conditions imply a black boundary, so that the pixels outside the borders of the image are zero.
- *Replicating* boundary conditions repeat the boarder elements outside the field of view.
- *Reflective* boundary conditions imply that the scene outside the image boundaries is a mirror image of the scene inside the image boundaries.

Fig. 1 illustrates these choices for boundary conditions with a particular image. There are many other choices for boundary conditions; see, for example, [34, 88]. We have found that reflective boundary conditions are often better than either periodic, zero and replicating, and they are relatively easy to implement. Thus, reflective boundary conditions are the default we use in our software. However, we also mention work by Reeves [81], which can be implemented very efficiently if the blur is spatially invariant and the support of the PSF is fairly narrow. Specifically, the approach decomposes the problem into a sum of two independent restorations, one uses a standard FFT-based filtering algorithm, while the other reconstructs values associated with the unknown pixels outside the boundary of the observed image.

### 2.2.2 Spatially Invariant Blurs

In the case of spatially invariant blurs, the matrix $\boldsymbol{K}$ generally has a Toeplitz structure, but this can depend on the imposed boundary condition [49]. However, the details are not extremely

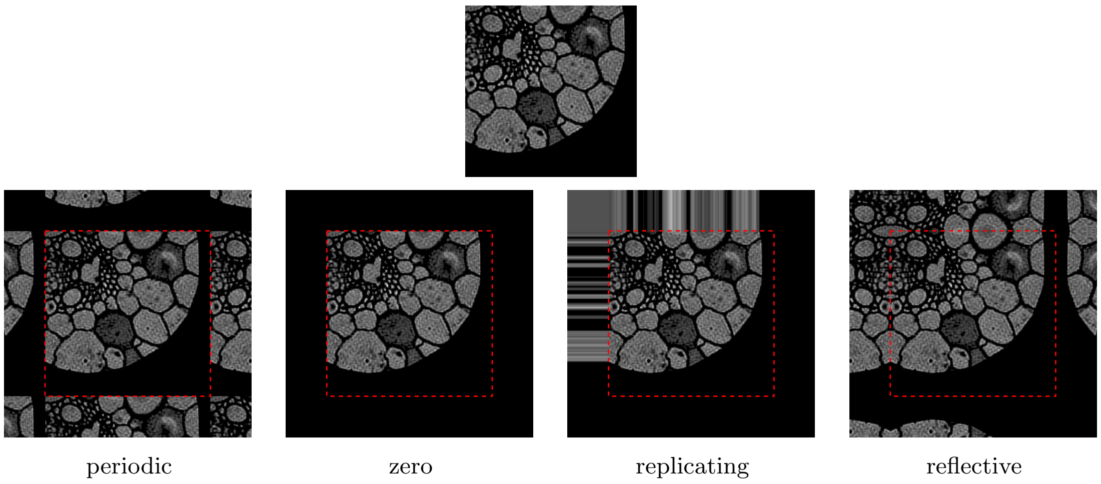


Figure 1: Examples of padding to incorporate boundary conditions. The top row is the image before padding, and the bottom row shows the result after padding. The red dotted line indicates the boundary of the image before padding.

important for iterative methods because we need only perform matrix-vector multiplications with $\boldsymbol{K}$ and $\boldsymbol{K}^\top$. Consider the multiplication

$$\boldsymbol{K}\boldsymbol{f}\,.$$

If the blur is spatially invariant and periodic boundary conditions are used, then this multiplication can be done by simply convolving the PSF (which defines $\boldsymbol{K}$) with $\boldsymbol{f}$. If we want to enforce a different boundary condition, then we simply pad the vector $\boldsymbol{f}$, as described in the previous subsection, convolve the PSF with this padded object, and then extract the portion of the result corresponding to the field of view. Note that this approach to compute the matrix-vector multiplication requires storing only the PSF, and does not require explicitly constructing the $mn \times mn$ matrix $\boldsymbol{K}$. The convolution of the PSF and the padded vector can be done very efficiently using FFTs; see, for example [70]. Matrix-vector multiplications with $\boldsymbol{K}^\top$ is done similarly.

### 2.2.3 Locally Spatially Invariant Blurs

In the case of spatially variant blurs, there is not a single PSF that can be used to represent the blurring operation; point sources in different locations of the image may result in different PSFs. In the most general case, there is a different PSF for each pixel in the image. It is not computationally feasible to construct all of these PSFs, so other approaches to representing $\boldsymbol{K}$ need to be considered. Here we consider situations where there is a continuous and smooth variation in the PSF with respect to the position of the point source, such as is shown in Fig. 2.

In this case it may be appropriate to assume that the blur is approximately locally spatially invariant. Exploiting local spatial invariance was considered in [98, 97]. Here we describe an approach where $\boldsymbol{K}$ is approximated with interpolation [14, 33, 67, 68, 70]. Specifically, by partitioning the image into regions, and assuming that a spatially invariant PSF can be obtained

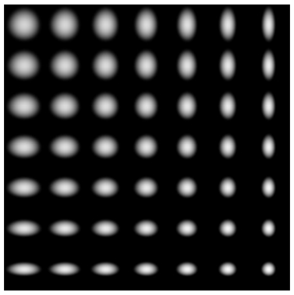


Figure 2: PSFs for a spatially variant blur that changes smoothly and continuously with respect to position of points sources.

for each region, the blurring matrix can be approximated by

$$\boldsymbol{K} \approx \sum_{i=1}^{r} \boldsymbol{K}_i \boldsymbol{D}_i \tag{7}$$

where $\boldsymbol{K}_i$ is defined by the PSF corresponding to the point source located at the center of the $i$-th region and $r$ is the overall number of regions. The *masking* matrix $\boldsymbol{D}_i$ is diagonal, with $\boldsymbol{D}_1 + \cdots + \boldsymbol{D}_r = \boldsymbol{I}$. In the case of piecewise constant interpolation, the diagonal entries of $\boldsymbol{D}_i$ are equal to one for pixels in region $i$, and zero otherwise. Higher order interpolation can also be used, but requires additional computational cost [67, 68].

### 2.2.4 Sparse Spatially Variant Blurs

If there is more severe spatial variance that cannot be approximated well by the local spatially invariant model, then it is important to exploit other structure in the problem. In some cases (we will see a specific example in Section 3.4) it is possible to use a sparse matrix data format, such as compressed row [29], to represent $\boldsymbol{K}$. This requires only keeping track of the nonzero entries and their locations in the matrix $\boldsymbol{K}$. Efficient multiplications can be done with such matrices; see, for example [29].

### 2.2.5 Separable Blurs

In some cases, the horizontal and vertical components of the blur can be separated. In this case, $\boldsymbol{K}$ can be decomposed as a Kronecker product,

$$\boldsymbol{K} = \boldsymbol{K}_{\mathrm{h}} \otimes \boldsymbol{K}_{\mathrm{v}}\,.$$

If the image has $m \times n$ pixels, then $\boldsymbol{K}_{\mathrm{v}} \in \mathcal{R}^{m\times m}$ represents the vertical component of the blurring, $\boldsymbol{K}_{\mathrm{h}} \in \mathcal{R}^{n\times n}$ represents the horizontal component of the blurring, and

$$\boldsymbol{K}\boldsymbol{f} = (\boldsymbol{K}_{\mathrm{h}} \otimes \boldsymbol{K}_{\mathrm{v}})\,\boldsymbol{f} = \nu\left(\boldsymbol{K}_{\mathrm{v}}\,\mu(\boldsymbol{f})\,\boldsymbol{K}_{\mathrm{h}}^{\top}\right)$$

where we recall that $\mu(\cdot)$ transforms a vector to an image array, and $\nu(\cdot)$ transforms an image array to a vector. Usually it is not computationally practical to explicitly construct the full matrix $\boldsymbol{K}$, however it is not so difficult to explicitly construct the much smaller matrices $\boldsymbol{K}_{\mathrm{h}}$ and

$\boldsymbol{K}_{\mathrm{v}}$, and by exploiting properties of Kronecker products, it is possible to efficiently implement SVD based filtering methods [49]. Although in most realistic situations the blur is unlikely to be exactly separable, it is possible to efficiently compute separable approximations of $\boldsymbol{K}$, which can be used to construct preconditioners.

## 2.3 Preconditioning

Preconditioning is a classical approach used in many areas of scientific computing to accelerate convergence of iterative methods. Much is known about constructing effective preconditioners for well-posed problems [8, 41, 86]. However, if not done carefully for ill-posed problems, such as image restoration, preconditioning can lead to erratic convergence behavior that results in fast convergence to a poor approximate solution. Some work has been done to overcome these difficulties for problems where the blurring operator is known [45, 46, 72], including for spatially variant blurs [67, 68, 71], and multi-frame problems [22]. Although there is an additional cost when using preconditioning, for typical iterative methods the number of iterations can be reduced dramatically, resulting in a substantial reduction in overall cost of the iterative scheme. We discuss some of these issues in this section, and describe a general approach to construct preconditioners for image restoration.

Speed of convergence of iterative methods is typically dictated by certain spectral properties of the matrix $\boldsymbol{K}$. Preconditioning refers to a process of modifying the spectral properties of the matrix to accelerate convergence, and is often presented in the context of solving linear systems $\boldsymbol{K}\boldsymbol{f} = \boldsymbol{g}$. The standard approach to preconditioning is to construct a matrix, $\boldsymbol{R}$, that satisfies the following properties:

- It should be relatively inexpensive to construct $\boldsymbol{R}$.
- It should be relatively inexpensive to solve linear systems of the form $\boldsymbol{R}\boldsymbol{z} = \boldsymbol{w}$ and $\boldsymbol{R}^\top \boldsymbol{z} = \boldsymbol{w}$.
- The preconditioned system should satisfy $\hat{\boldsymbol{K}} = \boldsymbol{K}\boldsymbol{R}^{-1} \approx \boldsymbol{I}$ (right preconditioning) or $\hat{\boldsymbol{K}} = \boldsymbol{R}^{-\top}\boldsymbol{K} \approx \boldsymbol{I}$ (left preconditioning).

Then instead of applying the iterative method to the linear system $\boldsymbol{K}\boldsymbol{f} = \boldsymbol{g}$, we apply it to a modified system $\hat{\boldsymbol{K}}\hat{\boldsymbol{f}} = \hat{\boldsymbol{g}}$, where

$$\begin{array}{llll} \text{Right preconditioning:} & \hat{\boldsymbol{K}} = \boldsymbol{K}\boldsymbol{R}^{-1}, & \hat{\boldsymbol{f}} = \boldsymbol{R}\boldsymbol{f}, & \hat{\boldsymbol{g}} = \boldsymbol{g} \\ \text{Left preconditioning:} & \hat{\boldsymbol{K}} = \boldsymbol{R}^{-\top}\boldsymbol{K}, & \hat{\boldsymbol{f}} = \boldsymbol{f}, & \hat{\boldsymbol{g}} = \boldsymbol{R}^{-\top}\boldsymbol{g} \end{array}$$

One can also use a combination of right and left preconditioning, but for this brief discussion we focus only on one sided preconditioning.

If the iterative method is applied to the preconditioned system, the most intensive part of the computation at each iteration is matrix-vector multiplications with $\hat{\boldsymbol{K}}$ and $\hat{\boldsymbol{K}}^\top$, or equivalently, matrix-vector multiplications with $\boldsymbol{K}$ and $\boldsymbol{K}^\top$ *and* linear system solves with $\boldsymbol{R}$ and $\boldsymbol{R}^\top$. Thus, the first two of the aforementioned requirements in constructing a preconditioner are related to the additional computational costs of preconditioning; constructing $\boldsymbol{R}$ is a one time cost, where as solving linear systems with matrices $\boldsymbol{R}$ and $\boldsymbol{R}^\top$ are required at each iteration. The last requirement determines the speed of convergence; better approximations $\hat{\boldsymbol{K}} \approx \boldsymbol{I}$, usually imply faster convergence.

Note that if we design a preconditioner such that the singular values of $\hat{\boldsymbol{K}}$ are clustered around 1, then $\hat{\boldsymbol{K}} \approx \boldsymbol{I}$. That is, more singular values clustered around one, as well as tighter clusters, usually implies faster convergence. Although this approach works well for well-posed problems, it does not work well for ill-posed problems such as image restoration, unless regularization has already been incorporated into the matrix. For example, if we apply the iterative method to the Tikhonov regularized least squares problem

$$\min_{\boldsymbol{f}} \left\| \begin{bmatrix} \boldsymbol{g} \\ \boldsymbol{0} \end{bmatrix} - \begin{bmatrix} \boldsymbol{K} \\ \lambda \boldsymbol{L} \end{bmatrix} \boldsymbol{f} \right\|_2$$

then clustering the singular values of

$$\begin{bmatrix} \boldsymbol{K} \\ \lambda \boldsymbol{L} \end{bmatrix} \boldsymbol{R}^{-1}$$

is the right idea.

However, if the iterative method is applied directly to

$$\min_{\boldsymbol{f}} \|\boldsymbol{g} - \boldsymbol{K}\boldsymbol{f}\|_2 \,,$$

as in the case of iterative regularization (recall the discussion in Section 2.1), then clustering all singular values around 1 will likely lead to very poor reconstructions. Indeed the SVD analysis outlined in Section 2.1 suggests that the large singular values correspond to signal information we want to reconstruct, while small singular values correspond to noise information we do not want to reconstruct. By clustering all singular values around one, the signal and noise information becomes mixed together, and it is impossible for the iterative method to distinguish between signal information and noise information. In this situation, we get fast convergence to the (noise corrupted) inverse solution.

Another difficulty, in the case of left preconditioning, is the risk of changing the statistical characteristics of the problem. In particular, with left preconditioning, the problem is modified as:

$$\boldsymbol{R}^{-\top}\boldsymbol{g} = \boldsymbol{R}^{-\top}\boldsymbol{K}\boldsymbol{f} + \boldsymbol{R}^{-\top}\boldsymbol{\eta} \,.$$

Thus, if an iterative method (such as maximum likelihood) implicitly assumes particular statistical characteristics of the data and noise, then the left preconditioned system may not continue to have these properties.

We will describe some specific preconditioning approaches when we begin to introduce particular iterative methods. As we will see, many well-known iterative methods are simply variations of the same basic scheme but with different preconditioners.

## 2.4 MATLAB Notes

Implementation of even the most basic iterative method for image restoration is not trivial, especially if we want to have the flexibility to use a variety of blurring operators and boundary conditions. We have developed a MATLAB toolbox to simplify the process of using iterative methods for image restoration. The software can be found at `www.mathcs.emory.edu/~nagy/RestoreTools`, with the original implementation described in [70]. A significant update to this software has been done while preparing this chapter.

The software uses an object oriented approach to hide the difficult implementation details from the user. Throughout this chapter we include some information on the important aspects of the software in case readers wish to test our iterative methods and preconditioning techniques on their own data. We also provide sample test data with the software; this test data is described in Section 3. We omit most of the implementation details, and instead just provide an overview of the more important tools in the software, and examples on how to use them.

**Spatially Invariant Blurs.**

For iterative methods, probably the most important object is the `psfMatrix`, which defines the matrix $\boldsymbol{K}$ implicitly using a compact data structure. In the case of spatially invariant blurs, we assume that there is an array containing a PSF and there is a vector containing the row and column indices of the location of the corresponding point source (i.e., the center of the PSF). For example, if a $256 \times 256$ PSF resulted from a point source located at row 128 and column 127, then we would define a vector

```
>> center = [128, 127];
```

With this data, the simple MATLAB statement

```
>> K = psfMatrix(PSF, center);
```

constructs an object containing information about the blurring operator. Some preprocessing is done to prepare `K` for efficient matrix-vector multiplications, and the "$*$" operator is overloaded so that an operation such as $\boldsymbol{g} = \boldsymbol{K}\boldsymbol{f}$ can be computed with the simple statement:

```
>> g = K*f;
```

Operator overloading allows for implementing iterative methods using standard MATLAB linear algebra operations, such as the matrix-vector multiplication `K*f`, just as if `K` was an explicitly defined matrix.

The above example assumes compatibility between $\boldsymbol{K}$ and $\boldsymbol{f}$. That is, `f` can be either an $m \times n$ image array, in which case after multiplication `g` is also an image of the same size, or `f` can be a vectorized form of the image, in which case after multiplication `g` is also a vector. We remark that the constructor routine `psfMatrix` can accept additional input parameters, including a boundary condition. The default boundary condition is `'reflective'`, which is typically much better than zero and periodic boundary conditions at reducing ringing effects if significant details are located near the edges of the observed image.

**Locally Spatially Invariant Blurs**

For locally spatially invariant blurs, we assume that the data `PSF` and `center` are cell arrays containing PSFs and corresponding point source locations for various regions of the image. The dimension of the cell arrays are assumed to be the same as the region partitioning of the image. That is, for example, if the image is partitioned into $6 \times 6$ regions of size $k \times k$ each, and a PSF is taken from each region, then we store all of the PSFs in a $6 \times 6$ cell array. Similarly we store the coordinates of each point source in a $6 \times 6$ cell array. If these cell arrays are denoted as `PSF` and `center`, then we can use any of the same statements used to construct the `psfMatrix`, and to perform matrix-vector multiplications as we used in the spatially invariant case described above.

We again emphasize that an advantage of using this object oriented approach, with operator overloading, is that iterative methods developed in the scientific computing community typically use matrix-vector notation. With our `psfMatrix`, these iterative methods can be easily used for image restoration problems.

**Sparse Spatially Variant Blurs**

In the case of sparse spatially variant blurs, we simply make use of MATLAB's very efficient built-in sparse matrix tools. An example of this is given in Section 3.4.

**Separable Blurs**

In the case of separable blurs, we have implemented a **`kronMatrix`** object that efficiently works with Kronecker products. In general, we do not expect a blur to be separable, so this object is mainly used to implement certain preconditioners; this is discussed in more detail in Section 2.3.

## 3 Model Problems

In this section, we describe several examples that can be used to test the iterative methods and preconditioning techniques described in this chapter. These examples include spatially invariant and spatially variant blurs of the types described in the previous section.

### 3.1 Spatially Invariant Gaussian Blur

Gaussian PSFs are often used to test image deblurring methods. A general Gaussian PSF is given by

$$k(s,t) = \frac{1}{2\pi\sqrt{\gamma}} \exp\left\{ -\frac{1}{2} \begin{bmatrix} s & t \end{bmatrix} \boldsymbol{C}^{-1} \begin{bmatrix} s \\ t \end{bmatrix} \right\} \tag{8}$$

where

$$\boldsymbol{C} = \begin{bmatrix} \alpha_1^2 & \rho^2 \\ \rho^2 & \alpha_2^2 \end{bmatrix}, \quad \text{and} \quad \gamma = \alpha_1^2\alpha_2^2 - \rho^4 > 0\,.$$

The shape of the Gaussian PSF depends on the parameters $\alpha_1, \alpha_2$ and $\rho$. If $\rho = 0$, then

$$k(s,t) = \frac{1}{2\pi\alpha_1\alpha_2} \exp\left\{ -\frac{1}{2}\left( \frac{s^2}{\alpha_1^2} + \frac{t^2}{\alpha_2^2} \right) \right\}$$

and if, additionally, $\alpha \equiv \alpha_1 = \alpha_2$, then

$$k(s,t) = \frac{1}{2\pi\alpha^2} \exp\left\{ -\frac{1}{2\alpha^2}\left( s^2 + t^2 \right) \right\}.$$

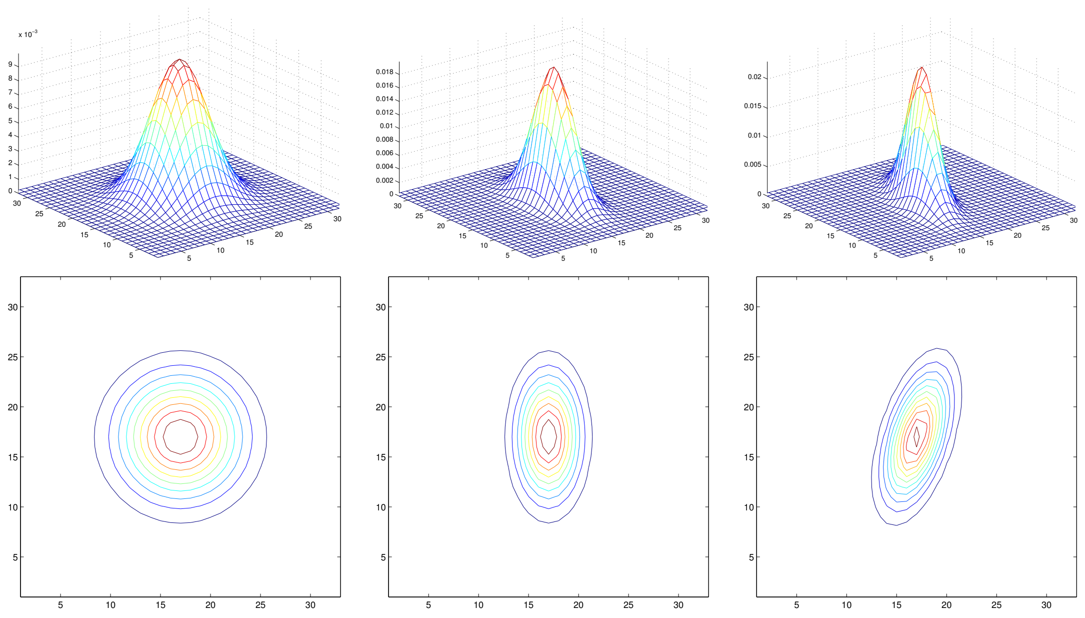
Notice that in the simplest case, where $\rho = 0$ and $\alpha_1 = \alpha_2$, the PSF is circularly symmetric and separable (that is, the blur can be decomposed into a product of two 1-dimensional blurs, one each for the horizontal and vertical directions). Fig. 3 shows three examples of Gaussian PSFs, and corresponding blurred images are shown in Fig. 4. The PSFs are displayed using a false colormap, rather than in grayscale, for better visualization.

We remark that in generating the simulated blurred images shown in Fig. 4, we actually used a true image with $618 \times 600$ pixels. After convolving a $256 \times 256$ Gaussian PSF with this image, we cropped the result down to size $256 \times 256$. This approach simulates obtaining a finite dimensional image of an infinite scene, and will allow us to illustrate how the choice of boundary condition can affect the quality of the restored image.

### 3.2 Spatially Invariant Atmospheric Turbulence Blur

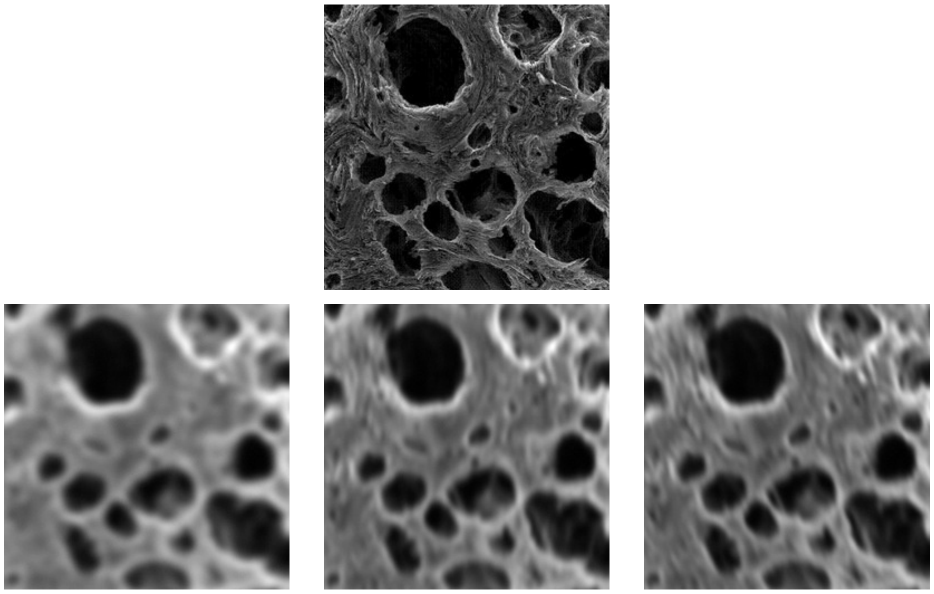
When imaging objects in space using ground based telescopes, the PSF depends on the wavefront of incoming light at the telescope's mirror; if the wavefront function is known, then $k$ is known. More specifically, $k(x,y;\xi,\eta) = k(x-\xi, y-\eta)$, with

$$k(s,t) = \left| \mathcal{F}^{-1}\left\{ P(s,t)e^{i(1-\omega(s,t))} \right\} \right|^2 = \left| \mathcal{F}^{-1}\left\{ P(s,t)e^{i\phi(s,t)} \right\} \right|^2, \tag{9}$$



$\alpha_1 = \alpha_2 = 4, \rho = 0$ $\quad$ $\alpha_1 = 4, \alpha_2 = 2, \rho = 0$ $\quad$ $\alpha_1 = 4, \alpha_2 = 2, \rho = 2$

Figure 3: Examples of Gaussian PSFs. The top row is a mesh plot of the PSF, and the bottom row shows how the contours differ for various values of $\alpha_1, \alpha_2$ and $\rho$ (a false colormap is used, rather than grayscale, for better visualization). Images blurred with these PSFs are shown in Fig. 4.



$\alpha_1 = \alpha_2 = 4, \rho = 0$ $\alpha_1 = 4, \alpha_2 = 2, \rho = 0$ $\alpha_1 = 4, \alpha_2 = 2, \rho = 2$

Figure 4: Examples of blurred images using the Gaussian PSFs shown in Fig. 3. The image in the top row is the true object, and the bottom row shows the blurred images.

where $\omega(s,t)$ is a function that models the shape of the wavefront of incoming light at the telescope, $i = \sqrt{-1}$, $P(s,t)$ is a characteristic function that models the shape of the telescope aperture (e.g., a circle or an annulus), $\mathcal{F}^{-1}$ is the 2-dimensional inverse Fourier transform, and $\phi(s,t) = 1 - \omega(s,t)$ is the phase error, or the deviation from planarity of the wavefront $\omega$.

In the ideal situation, where the atmosphere causes no distortion of the incoming wavefront, $\omega(s,t) = 1$ and $\phi(s,t) = 0$. In this *diffraction limited* case,

$$k_0(s,t) = \left|\mathcal{F}^{-1}\left\{P(s,t)\right\}\right|^2$$

where $P(s,t)$ is the pupil aperture function. Note that if $P(s,t) = 1$ for all $s$ and $t$, then $k_0(s,t)$ is a delta function, and (except for noise) there is no distortion in the observed image $g$. However, in a realistic situation, $P(s,t) = 1$ in at most a finite region (e.g., within a circle or annulus defined by the telescope aperture), and thus it is impossible to obtain a perfect image. The best result we can hope to obtain is the noise free, diffraction limited image

$$f_0(x,y) = \int_{\mathcal{R}^2} k_0(x-\xi, y-\eta) f(\xi,\eta)\,.$$

The distortion of the wavefront from atmospheric turbulence depends on many factors, including weather, temperature, wavelength, and the diameter of the telescope. For example, viewing objects directly above the telescope site on a clear night will have significantly better seeing conditions than looking during daylight hours at objects close to the horizon. Astronomers often quantify seeing conditions in terms of the ratio $d/r_0$, where $d$ is the diameter of the telescope and $r_0$ is called the *Fried parameter*, which is related to the wavelength, and provides a statistical description of the level of atmospheric turbulence at a particular site [51]. It is not essential to understand the precise definitions and characteristics of the Fried parameter, except that:

- Good seeing conditions correspond to "small" $d/r_0$, such as $d/r_0 \approx 10$.
- Poor seeing conditions correspond to "large" $d/r_0$, such as as $d/r_0 \approx 50$.

Fig. 5 shows examples of wavefronts and corresponding PSFs for $d/r_0 = 10, 30, 50$. The wavefronts and PSFs are displayed using a false colormap, rather than in grayscale, for better visualization. Note that the profile of the wavefronts looks similar, but the color bar shows that there is substantially more fluctuation in the wavefront during poor seeing conditions. Blurring caused by these PSFs is illustrated in Fig. 6.

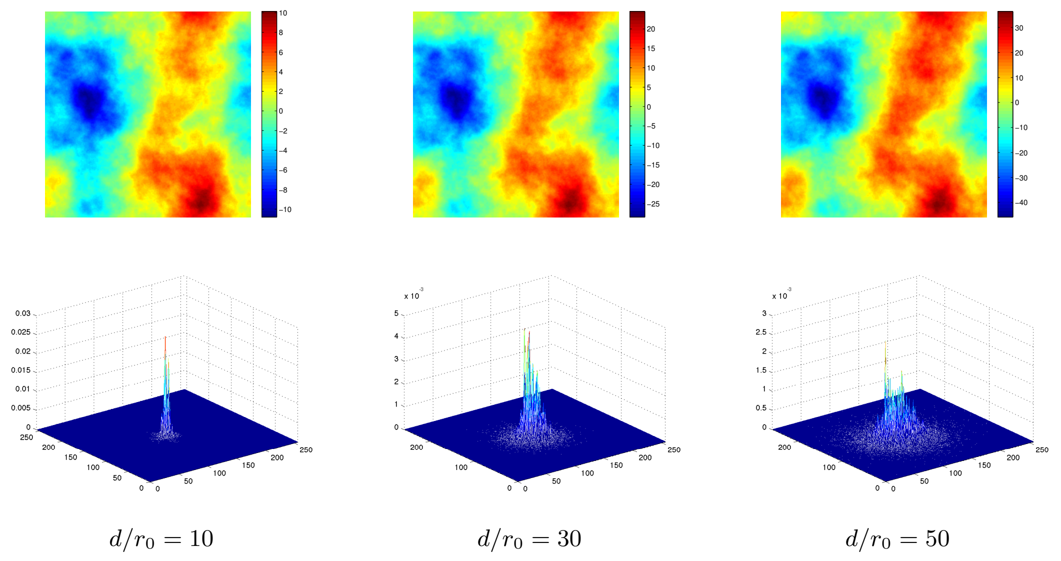


Figure 5: Examples of wavefronts (top row) and PSFs (bottom row) for $d/r_0 = 10, 30, 50$ (a false colormap is used, rather than grayscale, for better visualization). Blurred images for these PSFs are shown in Fig. 6.

## 3.3 Spatially Variant Gaussian Blur

It is more difficult to simulate a spatially variant blur, but one fairly simple example can be obtained by modifying the Gaussian PSF to include spatial variation as follows:

$$k(s,t) = \frac{1}{2\pi\alpha_1(s)\alpha_2(t)} \exp\left\{-\frac{1}{2}\left(\frac{s^2}{\alpha_1^2(s)} + \frac{t^2}{\alpha_2^2(t)}\right)\right\} \tag{10}$$

Spatially variant Gaussians have been used to test image restoration algorithms, see for example [2, 65]. Because the blur is spatially variant, one cannot use simple convolution to apply the blur. However, in this case the blur is separable,

$$k(s,t) = \left(\frac{1}{\sqrt{2\pi}\alpha_1(s)} \exp\left\{-\frac{1}{2}\left(\frac{s^2}{\alpha_1^2(s)}\right)\right\}\right)\left(\frac{1}{\sqrt{2\pi}\alpha_2(t)} \exp\left\{-\frac{1}{2}\left(\frac{t^2}{\alpha_2^2(t)}\right)\right\}\right),$$

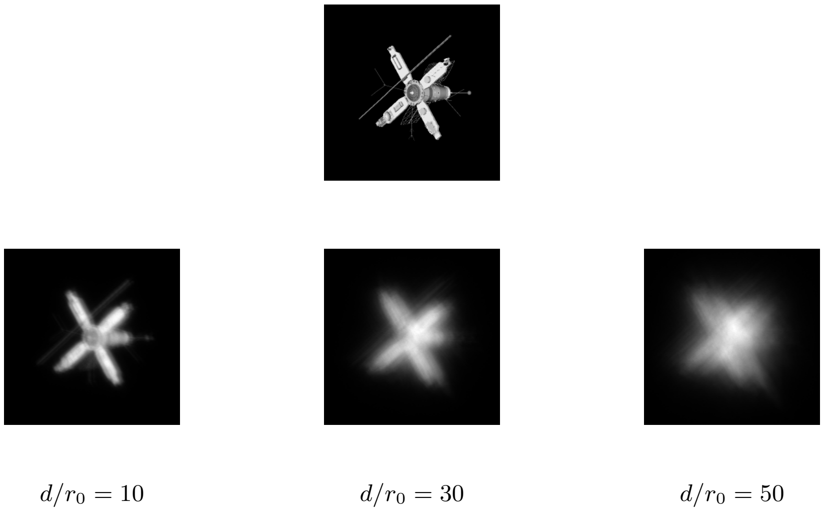


Figure 6: Examples of blurred images using PSFs shown in Fig. 5. The image in the top row is the true object, and the bottom row shows the blurred images.

and so the matrix $\boldsymbol{K}$ can be represented as a Kronecker product,

$$\boldsymbol{K} = \boldsymbol{K}_{\mathrm{h}} \otimes \boldsymbol{K}_{\mathrm{v}}\,,$$

where we refer readers to the discussion of separable blurs in Section 2.2.5. Usually it is not computationally practical to explicitly construct the full matrix $\boldsymbol{K}$, however it is not so difficult to explicitly construct the much smaller matrices $\boldsymbol{K}_{\mathrm{h}}$ and $\boldsymbol{K}_{\mathrm{v}}$.

The amount of variation in the blur is defined by $\alpha_1(s)$ and $\alpha_2(t)$. Note that if these are constants, then we get a spatially invariant Gaussian blur discussed in Section 3.1. We illustrate with three different examples:

- First, we consider a case in which the center of the image has only a mild distortion, and the blur becomes more severe as we move away from the center of the image. This can be simulated by using:

$$\alpha_1(s) = \tfrac{3}{4}(3|s|+1) \quad \text{and} \quad \alpha_2(t) = \tfrac{3}{4}(3|t|+1),$$

  where we assume $-1 \le s, t \le 1$.
- Next, we consider the alternative situation where the blur is more severe near the center of the image, and becomes less so near the edges. This can be simulated using:

$$\alpha_1(s) = \tfrac{3}{4}(-3|s|+4) \quad \text{and} \quad \alpha_2(t) = \tfrac{3}{4}(-3|t|+4),$$

  where we assume $-1 \le s, t \le 1$.
- Finally, we consider a situation where there is only a small amount of blurring in the lower right portion of the image, and it becomes more severe as we move toward the upper left corner of the image. This can be simulated using:

$$\alpha_1(s) = 3 - \tfrac{s}{2} \quad \text{and} \quad \alpha_2(t) = 3 - \tfrac{t}{2},$$

where we assume $-1 \leq s, t \leq 1$.

To visualize the variation in the blur, Fig. 7 shows blurred point sources in various locations in the image domain. Examples of blurred images with these spatially variant PSFs are shown in Fig. 8.

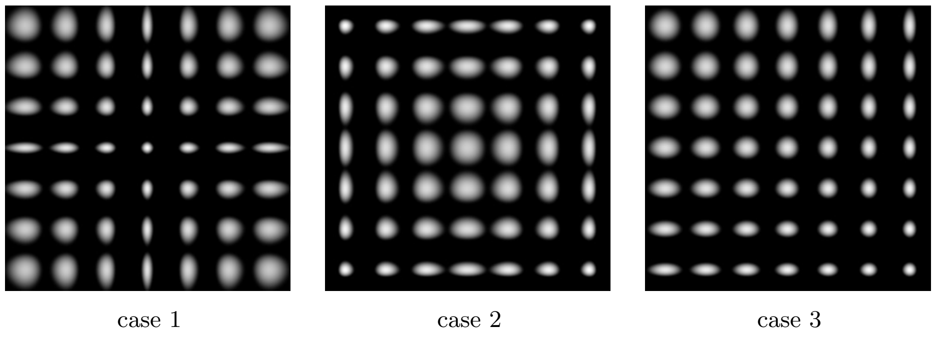


Figure 7: Spatially variant Gaussian Blurred images of point sources in various regions of the image domain. Blurred images for these PSFs are shown in Fig. 8.

The purpose of this example is to create data that can be used to test the approximation of spatially variant blur with a locally spatially invariant model, as described in Section 2.2.3. That is, because we are unlikely to have an explicit formula for the PSF, such as is given by equation (10), we approximate $\boldsymbol{K}$ using the interpolation model given in equation (7). Recall that to form such an approximation, all that is needed is a set of PSFs in different regions of the spatial domain.

## 3.4 Spatially Variant Motion Blur

In this section, we consider blur caused by motion of a rigid object, or equivalently motion blur caused by rigid movements of the imaging device. If the relative motion has constant speed and direction, then the blurring operation is spatially invariant. However, if the speed and direction varies during the image acquisition time, then the blur is spatially variant, and difficult to describe with a concise mathematical formula, as in the case of the spatially variant Gaussian blur. However, if it is possible to continuously measure the relative position of the object and the imaging device, then it is possible to construct a sparse approximation of the blurring matrix $\boldsymbol{K}$.

Although it may not be possible to know precisely and continuously the relative position of the object and detector, there are applications in which this information can be either approximated [97], or measured to high accuracy [32, 80]. The purpose of this subsection is not to describe motion estimation techniques, but to describe how to construct the matrix if the relative position information is known, so that we can use it as a test case for iterative image restoration algorithms.

To describe how to construct the matrix $\boldsymbol{K}$, and to simulate spatially variant motion blur, assume the observed image $\boldsymbol{g}$ is the (normalized) sum of images at incremental times during acquisition. Each of the individual images represents a snapshot of the object in a fixed position. To obtain a mathematical model, let $f(x, y)$ be a continuous function representing the object,

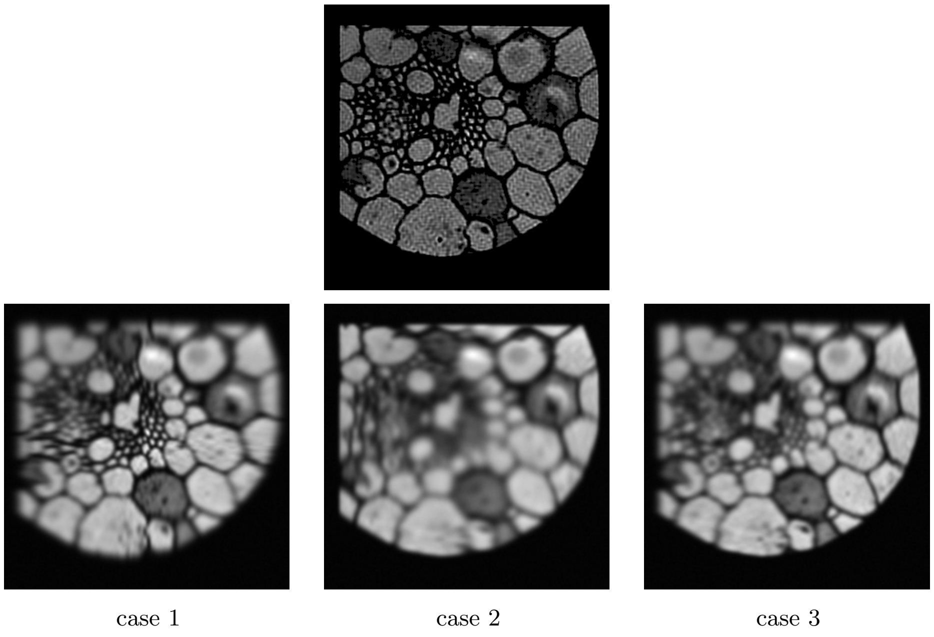


Figure 8: Examples of blurred images using spatially variant Gaussian PSFs shown in Fig. 7. The image in the top row is the true object, and the bottom row shows the blurred images.

and let $F$ be a discrete image, whose $(k, \ell)$ entry is given by

$$F(k, \ell) = f(x_k, y_\ell), \quad k = 1, 2, \ldots n, \quad \ell = 1, 2, \ldots n.$$

Now suppose $F_1$ is a discrete image obtained from the object $f$ after a rigid movement. Then there is an affine transformation $A \in \mathcal{R}^{3\times 3}$ such that

$$\begin{bmatrix} \hat{x}_k \\ \hat{y}_\ell \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_k \\ y_\ell \\ 1 \end{bmatrix}$$

and

$$F_1(k, \ell) = f(\hat{x}_k, \hat{y}_\ell).$$

Note that because the continuous image $f$ is not known at every point $(x, y)$ (all that is known is the discrete image $F$), it may not be possible to evaluate $f(\hat{x}_k, \hat{y}_\ell)$, unless $\hat{x}_k = x_{\hat{k}}$ and $\hat{y}_\ell = y_{\hat{\ell}}$ for some $1 \le \hat{k} \le n$ and $1 \le \hat{\ell} \le n$. However, an approximation of $f(\hat{x}_k, \hat{y}_\ell)$ can be computed by interpolating known values of $f$ near $f(\hat{x}_k, \hat{y}_\ell)$. Suppose (as illustrated in Fig. 9) that $f(x_{\hat{k}}, y_{\hat{\ell}})$, $f(x_{\hat{k}+1}, y_{\hat{\ell}})$, $f(x_{\hat{k}}, y_{\hat{\ell}+1})$ and $f(x_{\hat{k}+1}, y_{\hat{\ell}+1})$ are four known pixel values surrounding the unknown value $f(\hat{x}_k, \hat{y}_\ell)$. Nearest neighbor interpolation uses the known pixel value closest to $f(\hat{x}_k, \hat{y}_\ell)$; for example, in the illustration in Fig. 9 we have

$$F_1(k, \ell) = f(\hat{x}_k, \hat{y}_\ell) \approx f(x_{\hat{k}}, y_{\hat{\ell}+1}).$$

In the case of bilinear interpolation, a weighted average of the four pixels surrounding $f(\hat{x}_k, \hat{y}_\ell)$ is used for the approximation:

$$\begin{aligned} F_1(k,\ell) &= f(\hat{x}_k, \hat{y}_\ell) \\ &\approx (1-\Delta x_k)(1-\Delta y_\ell) f(x_{\hat{k}}, y_{\hat{\ell}}) + (1-\Delta x_k)\Delta y_\ell f(x_{\hat{k}}, y_{\hat{\ell}+1}) \\ &\quad + \Delta x_k (1-\Delta y_\ell) f(x_{\hat{k}+1}, y_{\hat{\ell}}) + \Delta x_k \Delta y_\ell f(x_{\hat{k}+1}, y_{\hat{\ell}+1}), \end{aligned}$$

where $\Delta x_k = \hat{x}_k - x_{\hat{k}}$ and $\Delta y_\ell = \hat{y}_\ell - y_{\hat{\ell}}$. The above discussion assumes that each pixel is a square with sides of length one; incorporating arbitrary size dimensions for pixels is not difficult.

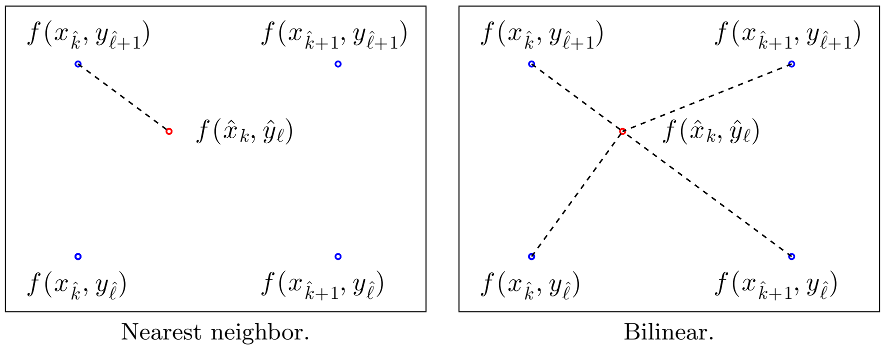

$f(x_{\hat{k}}, y_{\hat{\ell}+1})$
$f(x_{\hat{k}+1}, y_{\hat{\ell}+1})$
$f(\hat{x}_k, \hat{y}_\ell)$
$f(x_{\hat{k}}, y_{\hat{\ell}})$
$f(x_{\hat{k}+1}, y_{\hat{\ell}})$
Nearest neighbor.
$f(x_{\hat{k}}, y_{\hat{\ell}+1})$
$f(x_{\hat{k}+1}, y_{\hat{\ell}+1})$
$f(\hat{x}_k, \hat{y}_\ell)$
$f(x_{\hat{k}}, y_{\hat{\ell}})$
$f(x_{\hat{k}+1}, y_{\hat{\ell}})$
Bilinear.

Figure 9: Illustration of interpolation. The graphic on the left illustrates using the nearest (known pixel) neighbor to $f(\hat{x}_k, \hat{y}_\ell)$ to approximate its value. The graphic on the right illustrates bilinear interpolation (weighted average of the four known pixels) to approximate $f(\hat{x}_k, \hat{y}_\ell)$.

If we define vectors $\boldsymbol{f} = \nu(F)$ and $\boldsymbol{f}_1 = \nu(F_1)$ from the discrete image arrays (e.g., through lexicographical ordering), we can write

$$\boldsymbol{f}_1 = \boldsymbol{K}_1 \boldsymbol{f}$$

where $\boldsymbol{K}_1$ is a sparse matrix that contains the interpolation weights. Specifically, the $k$th row of $\boldsymbol{K}_1$ contains the weights for the pixel in the $k$th entry of $\boldsymbol{f}_1$. For example, in the case of bilinear interpolation, there are at most four nonzero entries per row, given by

$$(1-\Delta x_k)(1-\Delta y_\ell),\ (1-\Delta x_k)\Delta y_\ell,\ \Delta x_k(1-\Delta y_\ell),\ \Delta x_k \Delta y_\ell.$$

In the case of nearest neighbor interpolation, there is just one nonzero entry in each row. We emphasize that by using a sparse data format (e.g., compressed row [29]) to represent $\boldsymbol{K}$, we need only keep track of the nonzero entries and their locations in the matrix $\boldsymbol{K}$. Moreover, this discussion assumes the affine transformation $A$ is known, because this provides the necessary information to construct the interpolation weights.

As previously discussed, we assume the observed motion blurred image is the (normalized) sum of images at incremental times during the acquisition. That is, we assume

$$\boldsymbol{g} = \sum_{t=1}^{T} w_t \boldsymbol{f}_t + \boldsymbol{\eta}$$

where $\boldsymbol{f}_t = \boldsymbol{K}_t \boldsymbol{f}$ is a vector representing the discrete image at time $t$, $w_t$ is the normalization weight for the $t$-th image (for example, we could simply use $w_t = \frac{1}{T}$), and $\boldsymbol{\eta}$ is additive noise.

Thus, we obtain the deblurring problem given in equation (2), where the matrix modeling the motion blur is

$$\boldsymbol{K} = \sum_{t=1}^{T} w_t \boldsymbol{K}_t. \tag{11}$$

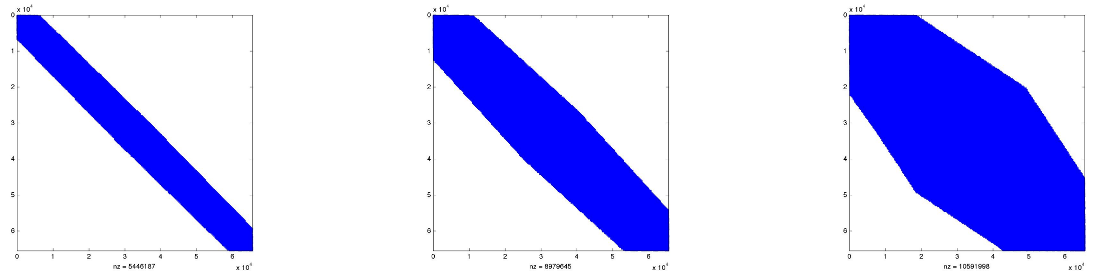
The sparsity structure of $\boldsymbol{K}$ depends on the severity of the motion blur, as well as on the number of known positions of the object. To illustrate, suppose that the motion is described by a shift and a rotation; specifically, suppose the object at the $t$-th position is shifted by $(\delta x_t, \delta y_t)$ and rotated (about the center) by $\theta_t$. An affine transformation for this, assuming the image is centered at $(x, y) = (0, 0)$ is

$$A_\ell = \begin{bmatrix} \cos(\theta_t) & \sin(\theta_t) & \delta x_t \\ -\sin(\theta_t) & \cos(\theta_t) & \delta y_t \\ 0 & 0 & 1 \end{bmatrix}$$

To generate test problems, we take random values for $\delta x_t$, $\delta y_t$ and $\theta_t$. We can control the severity of motion through how large we choose these values. Consider three examples, where in each case we use 50 positions (i.e., $t = 1, 2, \ldots, 50$):

- First we consider the case where the object moves only slightly during the acquisition time. In this case the values for $\delta x_t$ and $\delta y_t$ are chosen from a uniform distribution in the interval $(0, 2)$, and the values for $\theta_t$ are chosen from a normal distribution with mean 0 and standard deviation of $\pi/40$.
- In the second example we increase the severity of the motion by taking $\delta x_t$ and $\delta y_t$ from a uniform distribution in the interval $(0, 5)$, and the values for $\theta_t$ are chosen from a normal distribution with mean 0 and standard deviation of $\pi/20$.
- In the third example we further increase the severity of the motion by taking $\delta x_t$ and $\delta y_t$ from a uniform distribution in the interval $(0, 10)$, and $\theta_t$ from a normal distribution with mean 0 and standard deviation of $\pi/10$.

The sparsity pattern for the matrices $\boldsymbol{K}$ for these specific cases, using bilinear interpolation, is shown in Fig. 10. Notice that more severe motion results in a decrease in sparsity of $\boldsymbol{K}$. Blurred images for these cases are shown in Fig. 11.



small motion | medium motion | severe motion

Figure 10: Sparsity pattern of the matrix $\boldsymbol{K}$ for various amounts of object movement. Blurred images for these matrices are shown in Fig. 11.

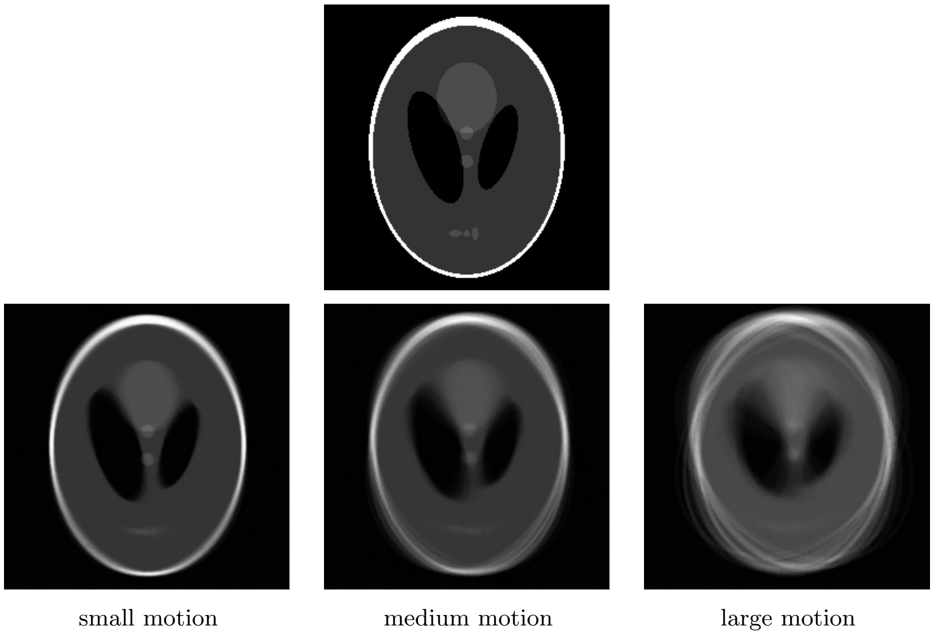


Figure 11: Examples of motion blurred images, using the blurring matrices shown in Fig. 10. The image in the top row is the true object, and the bottom row shows the blurred images.

### 3.5 MATLAB Notes

Test data for the examples described in this section are available at `www.mathcs.emory.edu/~nagy/RestoreTools`. The data are saved as `mat` files, and can be opened in MATLAB with the `load` command. Each of the `mat` files contains the true image $\boldsymbol{f}_{\text{true}}$ and the blurred, noisy image $\boldsymbol{g}$, as well as some statistical information about the noise, $\boldsymbol{\eta}$. The noise was generated using a combination of Poisson (background photon) and Gaussian white (readout) noise, with

$$\frac{\|\boldsymbol{\eta}\|_2}{\|\boldsymbol{K}\boldsymbol{f}_{\text{true}}\|_2} = 0.01.$$

The specific details of the noise model will be discussed in more detail in Section 5.2.2. For the data discussed in Sections 3.1–3.3 we include PSFs and their centers, while for the motion blur examples described in Section 3.4 we include the sparse matrices $\boldsymbol{K}$.

A summary of the data is as follows:

- Three examples of spatially invariant Gaussian blurs, using the PSF given in equation (8), and with the following parameters:

  `GaussianBlur440.mat` corresponds to the case $\alpha_1 = \alpha_2 = 4$ and $\rho = 0$.
  `GaussianBlur420.mat` corresponds to the case $\alpha_1 = 4$, $\alpha_2 = 2$ and $\rho = 0$.
  `GaussianBlur422.mat` corresponds to the case $\alpha_1 = 4$, $\alpha_2 = 2$ and $\rho = 2$.

- Three examples of blurring caused by atmospheric turbulence. The PSF is described by equation (9), and wavefronts were created with three different values of $d/r_0$. Specifically,

`AtmosphericBlur10` corresponds to $d/r_0 = 10$.
`AtmosphericBlur30` corresponds to $d/r_0 = 30$.
`AtmosphericBlur50` corresponds to $d/r_0 = 50$.

- Spatially variant Gaussian blur, using the PSF given by equation (10).

  `VariantGaussianBlur1.mat` uses $\alpha_1(s) = \frac{3}{4}(3|s|+1)$ and $\alpha_2(t) = \frac{3}{4}(3|t|+1)$.
  `VariantGaussianBlur2.mat` uses $\alpha_1(s) = \frac{3}{4}(-3|s|+1)$ and $\alpha_2(t) = \frac{3}{4}(-3|t|+1)$.
  `VariantGaussianBlur3.mat` uses $\alpha_1(s) = 3 - \frac{2}{2}$ and $\alpha_2(t) = 3 - \frac{t}{2}$.

  For each of these we generated 49 PSFs in a $7 \times 7$ region partitioning of the image domain. Thus, when this data is loaded into MATLAB, `PSF` and `center` will be cell arrays as described in Section 2.4.

- Spatially variant motion blur, using the examples described in Section 3.4. Specifically,

  `VariantMotionBlur_small.mat` contains data for the small motion simulation.
  `VariantMotionBlur_medium.mat` contains data for the medium motion simulation.
  `VariantMotionBlur_large.mat` contains data for the large motion simulation.

## 4 Iterative Methods for Unconstrained Problems

In this section, we describe some iterative methods that can be used to solve unconstrained image restoration problems. We focus on methods that can be applied directly to

$$\min_{\boldsymbol{f}} \|\boldsymbol{g} - \boldsymbol{K}\boldsymbol{f}\|_2$$

in an iterative regularization scheme, or to the Tikhonov least squares problem

$$\min_{\boldsymbol{f}} \left\| \begin{bmatrix} \boldsymbol{g} \\ \boldsymbol{0} \end{bmatrix} - \begin{bmatrix} \boldsymbol{K} \\ \lambda \boldsymbol{L} \end{bmatrix} \boldsymbol{f} \right\|_2 .$$

We note that if we define the quadratic function $\psi(\boldsymbol{f}) = \frac{1}{2}\boldsymbol{f}^\top \boldsymbol{K}^\top \boldsymbol{K}\boldsymbol{f} - \boldsymbol{f}^\top \boldsymbol{K}^\top \boldsymbol{g}$, then the following minimization problems are equivalent:

$$\min_{\boldsymbol{f}} \psi(\boldsymbol{f}) \quad \text{and} \quad \min_{\boldsymbol{f}} \|\boldsymbol{g} - \boldsymbol{K}\boldsymbol{f}\|_2 .$$

This equivalence is often used to describe and design iterative methods.

Iterative methods for unconstrained image restoration problems have the general form,

$$\boldsymbol{f}^{(k+1)} = \boldsymbol{f}^{(k)} + \tau_k \boldsymbol{d}^{(k)}$$

where $\boldsymbol{d}^{(k)}$ is a *step direction* and $\tau_k$ is a *step length.* Different choices for these terms leads to different iterative methods.

### 4.1 Richardson Iteration

The Richardson iteration, which is often called the Landweber method [31, 47, 48, 102], is generally used as an iterative regularization method. The basic iteration takes the form:

$$\boldsymbol{f}^{(k+1)} = \boldsymbol{f}^{(k)} + \tau \boldsymbol{K}^\top \left( \boldsymbol{g} - \boldsymbol{K}\boldsymbol{f}^{(k)} \right)$$

That is, the step

$$\boldsymbol{d}^{(k)} = \boldsymbol{K}^\top \left(\boldsymbol{g} - \boldsymbol{K}\boldsymbol{f}^{(k)}\right)$$

is the steepest descent direction for the quadratic function $\psi(\boldsymbol{f})$, and the step size $\tau$ remains constant for each iteration. It can be shown that to ensure convergence the step size must satisfy

$$0 < \tau < \frac{2}{\sigma_{\max}^2}$$

where $\sigma_{\max}$ is the largest singular value of $\boldsymbol{K}$. Although it might be difficult to compute $\sigma_{\max}^2 = \|\boldsymbol{K}^\top\boldsymbol{K}\|_2$, a default choice for $\tau$ can be found using the well-known bound [39]

$$\sigma_{\max}^2 = \|\boldsymbol{K}^\top\boldsymbol{K}\|_2 \leq \|\boldsymbol{K}\|_1\|\boldsymbol{K}\|_\infty .$$

In general, the matrix norms $\|\boldsymbol{K}\|_1$ and $\|\boldsymbol{K}\|_\infty$ are easy to compute. In fact, if $\boldsymbol{K}$ has no negative entries, as is the case in image restoration problems, and if we define $\mathbf{1}$ to be a vector with every entry equal to 1, then

$$\begin{aligned} \|\boldsymbol{K}\|_\infty &= \text{max element in the vector } \boldsymbol{K}\mathbf{1}, \text{ and} \\ \|\boldsymbol{K}\|_1 &= \text{max element in the vector } \boldsymbol{K}^\top\mathbf{1}. \end{aligned}$$

Thus, using this bound, a default choice for $\tau$ can be

$$\tau = \frac{1}{\|\boldsymbol{K}\|_1\|\boldsymbol{K}\|_\infty} \leq \frac{1}{\sigma_{\max}^2} < \frac{2}{\sigma_{\max}^2} .$$

It can be shown [47] that the Richardson iteration can be interpreted as an SVD filtering method as described in equation (3). In particular, if the initial guess is $\boldsymbol{f}^{(0)} = \mathbf{0}$, then the filter factors for the $k$th iteration are given by

$$\phi_i^{(k)} = 1 - (1 - \tau\sigma_i^2)^k , \quad i = 1, 2, \ldots, N ,$$

where $\sigma_i$ is the $i$-th largest singular (or spectral) value of $\boldsymbol{K}$. To visualize what this tells us about convergence properties of the Richardson iteration, assume that $\sigma_{\max} = \tau = 1$. In this case, we see that in the early iterations solution components corresponding to large singular values are reconstructed, while we need many iterations before components corresponding to small singular values are reconstructed. That is,

$$0 \approx \phi_N^{(k)} \leq \phi_{N-1}^{(k)} \leq \cdots \leq \phi_1^{(k)} \approx 1 ,$$

as should be the case for an SVD filtering method. However, it may take many iterations to reconstruct components corresponding to intermediate singular values. For example, using again $\sigma_1 = \tau = 1$, then with the intermediate singular value $\sigma_j = 0.01$, we obtain filter factors $\phi_j^{(k)} = 1 - (1 - \tau\sigma_j)^k = 1 - (0.99)^k$, and thus $k$ would need to be very large to obtain $\phi_j^{(k)} \approx 1$. Whether or not we want to reconstruct this component of the solution depends on the problem, but because we expect the singular values to decay to zero, a value of $\sigma_j = 0.01$ is, in general, not very small and is likely to correspond to signal subspace rather than noise subspace information.

From this analysis, we expect that the basic Richardson iteration converges very slowly. However, as will be seen below, the method can be accelerated with preconditioning. It also provides a good introduction to discussing other iterative methods, including ones that enforce a nonnegativity constraint. An algorithm for the Richardson iteration can be stated as follows:

**Algorithm: Richardson Iteration**

```
given:      $\boldsymbol{K}$, $\boldsymbol{g}$
choose:     initial guess for $\boldsymbol{f}$
            step length parameter $\tau$
compute:
            $\boldsymbol{r} = \boldsymbol{g} - \boldsymbol{K}\boldsymbol{f}$
            $\boldsymbol{d} = \boldsymbol{K}^\top \boldsymbol{r}$
            while (not stop) do
               $\boldsymbol{f} = \boldsymbol{f} + \tau \boldsymbol{d}$
               $\boldsymbol{r} = \boldsymbol{g} - \boldsymbol{K}\boldsymbol{f}$
               $\boldsymbol{d} = \boldsymbol{K}^\top \boldsymbol{r}$
            end while
```

As with any iterative regularization method, it is important to choose appropriate criteria for terminating the `while` loop. This is a nontrivial topic. About this topic we mention only an approach known as the *Morozov's discrepancy principle* [64], which terminates the iteration when

$$\|\boldsymbol{g} - \boldsymbol{K}\boldsymbol{f}^{(k)}\|_2 \leq \delta\varepsilon$$

where $\varepsilon$ is the expected value of the noise $\boldsymbol{\eta}$ and $\delta > 1$. The discrepancy principle is easy to implement, but it does require a good estimate of $\varepsilon$. Other approaches to estimating a stopping iteration are described in [48, 102].

## 4.2 Preconditioned Richardson Methods

We now consider preconditioning of the basic Richardson method. Recall that preconditioning is a technique used to modify the problem in such a way as to accelerate convergence. In the case of right preconditioning, we replace $\boldsymbol{K}$ with $\hat{\boldsymbol{K}} = \boldsymbol{K}\boldsymbol{R}^{-1}$ and $\boldsymbol{f}$ with $\hat{\boldsymbol{f}} = \boldsymbol{R}\boldsymbol{f}$. After some algebraic manipulation, the main part of the iteration can be written as:

$$\boldsymbol{f}^{(k+1)} = \boldsymbol{f}^{(k)} + \tau \boldsymbol{M}^{-1}\boldsymbol{K}^\top \left(\boldsymbol{g} - \boldsymbol{K}\boldsymbol{f}^{(k)}\right) \tag{12}$$

where $\boldsymbol{M} = \boldsymbol{R}^\top \boldsymbol{R}$.

In the case of left preconditioning, we replace $\boldsymbol{K}$ with $\hat{\boldsymbol{K}} = \boldsymbol{R}^{-\top}\boldsymbol{K}$ and $\boldsymbol{g}$ with $\hat{\boldsymbol{g}} = \boldsymbol{R}^{-\top}\boldsymbol{g}$. After some algebraic manipulation, the main part of the iteration can be written as:

$$\boldsymbol{f}^{(k+1)} = \boldsymbol{f}^{(k)} + \tau \boldsymbol{K}^\top \boldsymbol{M}^{-1} \left(\boldsymbol{g} - \boldsymbol{K}\boldsymbol{f}^{(k)}\right) \tag{13}$$

where $\boldsymbol{M} = \boldsymbol{R}^\top \boldsymbol{R}$.

Algorithms for these preconditioned Richardson iterative methods can be written as:

**Algorithm: Richardson Iteration Left Preconditioned**

```
given:    $K$, $g$, $M$
choose:   initial guess for $f$
          step length parameter $\tau$
compute:
          $r = g - Kf$
          $v = M^{-1}r$
          $d = K^\top v$
          while (not stop) do
             $f = f + \tau d$
             $r = g - Kf$
             $v = M^{-1}r$
             $d = K^\top v$
          end while
```

**Algorithm: Richardson Iteration Right Preconditioned**

```
given:    $K$, $g$, $M$
choose:   initial guess for $f$
          step length parameter $\tau$
compute:
          $r = g - Kf$
          $v = K^\top r$
          $d = M^{-1}v$
          while (not stop) do
             $f = f + \tau d$
             $r = g - Kf$
             $v = K^\top r$
             $d = M^{-1}v$
          end while
```

It is important to keep in mind that the convergence behavior, and the choice of $\tau$ depend on the singular values of the preconditioned system. For example, consider the right preconditioned system, $\hat{\boldsymbol{K}} = \boldsymbol{K}\boldsymbol{R}^{-1}$. Then the similarity transformation

$$\boldsymbol{R}^{-1}\hat{\boldsymbol{K}}^\top\hat{\boldsymbol{K}}\boldsymbol{R} = \left(\boldsymbol{R}^\top\boldsymbol{R}\right)^{-1}\boldsymbol{K}^\top\boldsymbol{K} = \boldsymbol{M}^{-1}\boldsymbol{K}^\top\boldsymbol{K}\,,$$

shows that $\hat{\boldsymbol{K}}^\top\hat{\boldsymbol{K}}$ and $\boldsymbol{M}^{-1}\boldsymbol{K}^\top\boldsymbol{K}$ have the same eigenvalues. Thus,

$$\hat{\sigma}^2_{\max} = \|\hat{\boldsymbol{K}}^\top\hat{\boldsymbol{K}}\|_2 = \|\boldsymbol{M}^{-1}\boldsymbol{K}^\top\boldsymbol{K}\|_2 \le \|\boldsymbol{M}^{-1}\|_2\|\boldsymbol{K}^\top\boldsymbol{K}\|_2\,.$$

Thus, as with the case of the unpreconditioned Richardson iteration, a default choice for $\tau$ can be

$$\tau = \frac{1}{\|\boldsymbol{M}^{-1}\|_2\|\boldsymbol{K}\|_1\|\boldsymbol{K}\|_\infty} \le \frac{2}{\|\boldsymbol{M}^{-1}\boldsymbol{K}^\top\boldsymbol{K}\|_2} = \frac{2}{\|\hat{\boldsymbol{K}}^T\hat{\boldsymbol{K}}\|_2}\,,$$

assuming that we can compute an estimate of $\|\boldsymbol{M}^{-1}\|_2$. A similar result holds for left preconditioning.

We now give a few examples of preconditioners, and show that some of them lead to other well-known iterative methods.

### 4.2.1 A Diagonal Matrix Preconditioner: Cimmino's Method

We begin with a very simple preconditioner that results in the *Cimmino* method. Specifically, we consider using the following diagonal matrix as a left preconditioner:

$$\boldsymbol{R} = \mathrm{diag}\left(\frac{\|\boldsymbol{k}_1\|_2}{\sqrt{c_1}}, \frac{\|\boldsymbol{k}_2\|_2}{\sqrt{c_2}}, \ldots, \frac{\|\boldsymbol{k}_N\|_2}{\sqrt{c_N}}\right)\,, \quad \boldsymbol{M} = \boldsymbol{R}^T\boldsymbol{R} = \mathrm{diag}\left(\frac{\|\boldsymbol{k}_1\|_2^2}{c_1}, \frac{\|\boldsymbol{k}_2\|_2^2}{c_2}, \ldots, \frac{\|\boldsymbol{k}_N\|_2^2}{c_N}\right)\,,$$

where $\boldsymbol{k}_i^\top$ is the $i$th row of $\boldsymbol{K}$ and $c_i$ are positive numbers. We could choose $\tau$ to be

$$\tau = \frac{1}{\|\boldsymbol{M}^{-1}\|_2 \|\boldsymbol{K}\|_1 \|\boldsymbol{K}\|_\infty}$$

where $\|\boldsymbol{M}^{-1}\|_2 = \max\limits_{1 \le i \le N} \left\{ c_i / \|\boldsymbol{k}_i\|_2^2 \right\}$. Another choice, proposed by Cimmino [9, 48], is to use

$$\tau = \sum_{i=1}^{N} c_i \, .$$

It is not difficult to show that with this value, $\tau < 2/\hat{\sigma}_{\max}^2$. Observe that if we choose $c_1 = \cdots = c_N = 1$, then we obtain

$$\boldsymbol{M}^{-1} = \operatorname{diag}\left( \frac{1}{\|\boldsymbol{k}_1\|_2^2}, \frac{1}{\|\boldsymbol{k}_2\|_2^2}, \ldots, \frac{1}{\|\boldsymbol{k}_N\|_2^2} \right) \quad \text{and} \quad \tau = 2/N \, ,$$

where $N$ is the number of pixels in the image. In this case, $\tau$ is less than one, resulting in small steps at each iteration. This may or may not lead to slow convergence; if the direction vector $\boldsymbol{d}$ is good, then a small step size might be appropriate.

In an implementation of Cimmino's method, inverting the diagonal matrix $\boldsymbol{M}$ is trivial, but construction of the preconditioner requires computing $\|\boldsymbol{k}_i\|_2^2$, the squared norms of rows of $\boldsymbol{K}$. If the blur is spatially invariant, then we can assume that these norms are approximately constant, equal to the sum of squares of the PSF pixel values; exact equality occurs in the case of periodic boundary conditions.

### 4.2.2 Triangular Matrix Preconditioners: SOR Methods

The methods of *successive over-relaxation* (SOR) have been well studied in the applied mathematics and scientific computing community, especially in the context of solving partial differential equations [12, 39, 41, 86]. They have recently been proposed for use in image restoration [93, 103], so we give a brief description of these methods here.

The SOR methods are based on matrix splittings. Specifically, since the matrix $\boldsymbol{K}^\top \boldsymbol{K}$ is symmetric, we can split it up as

$$\boldsymbol{K}^\top \boldsymbol{K} = \boldsymbol{T} + \boldsymbol{D} + \boldsymbol{T}^\top ,$$

where $\boldsymbol{D}$ is the diagonal part and $\boldsymbol{T}$ is the strictly lower triangular part of $\boldsymbol{K}^\top \boldsymbol{K}$. If we then set

$$\boldsymbol{M} = \boldsymbol{D} + \tau \boldsymbol{T} \, ,$$

we obtain the standard SOR method. A similar approach is the *symmetric successive over-relaxation* (SSOR) method [86], which uses the preconditioner

$$\boldsymbol{M} = \frac{1}{2-\tau} \left( \boldsymbol{D} + \tau \boldsymbol{T} \right) \boldsymbol{D}^{-1} \left( \boldsymbol{D} + \tau \boldsymbol{T} \right)^\top .$$

The term *over-relaxation* is used because $\tau$ is considered a relaxation parameter, and in well-posed problems arising, for example, from discretization of partial differential equations, an optimal value for $\tau$ is chosen greater than one (i.e., the method uses over-relaxation).

The SOR preconditioner requires less work to implement than SSOR, but it is not obvious how to provide a general analysis of the filtering properties of SOR for ill-posed problems. However, in the SSOR case $\boldsymbol{M}$ can be written as

$$\boldsymbol{M} = \boldsymbol{R}^\top \boldsymbol{R}, \quad \text{where} \quad \boldsymbol{R} = \sqrt{\frac{1}{2-\tau}} \boldsymbol{D}^{-1/2} (\boldsymbol{D} + \tau \boldsymbol{T})^\top .$$

Thus the filtering properties of SSOR are explained by examining the singular values of the preconditioned system $\hat{\boldsymbol{K}} = \boldsymbol{K}\boldsymbol{R}^{-1}$.

We mention that the properties of SOR for image restoration, in the simplified case of spatially invariant blurs with periodic boundary conditions, were studied in [93, 103]. They show that, contrary to well-posed problems arising in partial differential equations, the relaxation parameter should satisfy $\tau \ll 1$; that is, the method should use *severe under*-relaxation instead of *over*-relaxation. We can see that this makes sense by supposing that $\boldsymbol{K}^\top \boldsymbol{K}$ and $\boldsymbol{M}$ have the same eigenvectors; that is, suppose

$$\boldsymbol{K}^\top \boldsymbol{K} = \boldsymbol{Q}^T \boldsymbol{\Lambda}_{\rm k} \boldsymbol{Q} \quad \text{and} \quad \boldsymbol{M} = \boldsymbol{Q}^T \boldsymbol{\Lambda}_{\rm m} \boldsymbol{Q}\,,$$

where $\boldsymbol{\Lambda}_{\rm k}$ is a diagonal matrix containing the eigenvalues of $\boldsymbol{K}^\top \boldsymbol{K}$ and $\boldsymbol{\Lambda}_{\rm m}$ is a diagonal matrix containing the eigenvalues of $\boldsymbol{M}$. In this case,

$$\boldsymbol{M}^{-1} \boldsymbol{K}^\top \boldsymbol{K} = \boldsymbol{Q}^T \boldsymbol{\Lambda} \boldsymbol{Q}\,,$$

where the diagonal matrix $\boldsymbol{\Lambda}$ contains the ratio of eigenvalues of $\boldsymbol{K}^\top \boldsymbol{K}$ and $\boldsymbol{M}$. In the worst case, it could happen that one of the diagonal entries of $\boldsymbol{\Lambda}$ is the largest eigenvalue of $\boldsymbol{K}^\top \boldsymbol{K}$ divided by the smallest eigenvalue of $\boldsymbol{M}$. This value could be very large, and so the upper bound for $\tau$, $2/\|\boldsymbol{M}^{-1} \boldsymbol{K}^\top \boldsymbol{K}\|_2$, could be very small.

It is important to observe that by choosing $\tau \ll 1$ we obtain stability in the preconditioned iteration. However, because $\tau$ is also the step size in the algorithm, small $\tau$ means small steps, and thus convergence can be very slow. We say *can* be slow, because the initial direction vectors could be very good, and it therefore could be the case that we need only a few iterations to reconstruct a good approximation of $\boldsymbol{f}_{\rm true}$ before noise begins to corrupt the solution. It is difficult to provide a general analysis of this for the SOR and SSOR preconditioners. It should be mentioned that, in the context of set theoretic methods [26, 89, 90], Combettes [27] has proposed a scheme to actually *increase* the relaxation parameter to a value greater than 2, with the goal to improve the rate of convergence. It would be interesting to see if such an approach can be applied to the SOR and SSOR methods. Another disadvantage of the SOR and SSOR preconditioners is that solving systems with $\boldsymbol{M}$ can be relatively expensive compared to other preconditioners. We discuss these issues in more detail for filtering based preconditioners.

### 4.2.3 Filtering Based Preconditioning

In this subsection, we describe an approach to construct preconditioners for image restoration problems that was originally proposed in 1993 [46]. We first explain the basic idea in the ideal situation where we can compute a singular value decomposition (SVD) of $\boldsymbol{K}$. The SVD is defined as

$$\boldsymbol{K} = \boldsymbol{U} \boldsymbol{\Sigma} \boldsymbol{V}^\top\,,$$

where $\boldsymbol{U}$ and $\boldsymbol{V}$ are orthogonal matrices, and $\boldsymbol{\Sigma}$ =diag$(\sigma_1, \sigma_2, \ldots, \sigma_N)$ is a diagonal matrix containing the singular values of $\boldsymbol{K}$. If we define $\boldsymbol{R} = \boldsymbol{U} \boldsymbol{\Sigma}_r \boldsymbol{V}^\top$, where $\boldsymbol{\Sigma}_r$ =diag$(\sigma_1^{(r)}, \sigma_2^{(r)}, \ldots, \sigma_N^{(r)})$, then in the case of left preconditioning,

$$\hat{\boldsymbol{K}} = \boldsymbol{R}^{-1}\boldsymbol{K} = \boldsymbol{V}\hat{\boldsymbol{\Sigma}}V^\top \,,$$

where $\hat{\boldsymbol{\Sigma}} = \text{diag}(\hat{\sigma}_1, \hat{\sigma}_2, \ldots, \hat{\sigma}_N)$, $\hat{\sigma}_i = \sigma_i/\sigma_i^{(r)}$. Recalling the convergence analysis and filtering properties of the basic Richardson iteration, to obtain fast convergence of components corresponding to large singular values (i.e., signal information), we should choose $\sigma_i^{(r)}$ so that $\hat{\sigma}_i \approx 1$ for large $\sigma_i$. On the other hand, to make sure we do not have fast convergence of the noise subspace information (i.e., components of the solution corresponding to small singular values) we should choose $\sigma_i^{(r)}$ so that $\hat{\sigma}_i$ is small. There are many ways to do this. In [45, 46] a truncated SVD, or pseudo inverse like approach was proposed, where

$$\sigma_i^{(r)} \approx \begin{cases} \sigma_i & \sigma_i \geq \texttt{tol} \\ 1, & \sigma_i < \texttt{tol} \end{cases}$$

where `tol` is a specified truncation tolerance. Here we see that the singular values larger than `tol` of the preconditioned system are clustered at one, and are well separated from the remaining "small" singular values. Determining how to choose the truncation tolerance is related to determining regularization parameters. Various techniques can be used, including the discrete Picard condition, L-curve, and generalized cross validation (GCV); see [45, 46, 70] for more details.

Another approach for choosing $\sigma_i^{(r)}$ is to use a Tikhonov regularization type approach, where

$$\sigma_i^{(r)} = \sqrt{\sigma_i^2 + \lambda^2}\,,$$

where $\lambda$ is a specified regularization parameter for the preconditioner. As with the truncation tolerance, various techniques to choose $\lambda$, including the discrete Picard condition, L-curve, and GCV. A comparison of these and other choices for choosing $\sigma_i^{(r)}$ is given in [16].

Computing the SVD of $\boldsymbol{K}$ is typically too expensive for large scale problems, and in the cases where it is possible to compute, we might as well use direct filtering methods instead of iterative methods. However, this discussion suggests that filtering preconditioners can be constructed with SVD approximations. A general approach to computing an SVD approximation is to choose, *a priori*, orthogonal (or unitary, if we want to consider complex bases) matrices $\widetilde{\boldsymbol{U}}$ and $\widetilde{\boldsymbol{V}}$, and determine a diagonal matrix $\widetilde{\boldsymbol{\Sigma}}$ such that

$$\widetilde{\boldsymbol{\Sigma}} = \arg\min_{\boldsymbol{\Sigma}} \|\widetilde{\boldsymbol{U}}^\top \boldsymbol{K}\widetilde{\boldsymbol{V}} - \boldsymbol{\Sigma}\|_F$$

where $\|\cdot\|_F$ denotes the Frobenius norm, and where the minimization is done over all diagonal matrices, $\boldsymbol{\Sigma}$. We then use as a preconditioner $\boldsymbol{R} = \widetilde{\boldsymbol{U}}\boldsymbol{\Sigma}_r\widetilde{\boldsymbol{V}}^\top$, where we use one of the approaches discussed above for choosing the diagonal entries $\sigma_i^{(r)}$ from $\tilde{\sigma}_i$.

To make this approach efficient, the construction of, and computations with $\widetilde{\boldsymbol{U}}$ and $\widetilde{\boldsymbol{V}}$ should be inexpensive. Several approaches have been proposed in the literature:

- By choosing $\widetilde{\boldsymbol{U}} = \widetilde{\boldsymbol{V}} = \mathcal{F}$, the discrete Fourier transform matrix, computations with $\widetilde{\boldsymbol{U}}$ and $\widetilde{\boldsymbol{V}}$ can be done very efficiently with FFTs. The resulting approximation, $\widetilde{\boldsymbol{K}} = \widetilde{\boldsymbol{U}}\widetilde{\boldsymbol{\Sigma}}\widetilde{\boldsymbol{V}}^\top$ is a block circulant matrix with circulant blocks, and is the best such approximation of $\boldsymbol{K}$; see [23] for more details. The cost of constructing the approximation, and computations with the preconditioner are $O(N \log N)$.

- Instead of using the FFT basis to build $\widetilde{\boldsymbol{U}}$ and $\widetilde{\boldsymbol{V}}$, we could use another fast transform, such as the discrete cosine transform (DCT) [23]. As with FFTs, construction of this approximation, and computations with the resulting preconditioner are $O(N \log N)$.
- Another alternative is use a separable approximation of $\boldsymbol{U}$ and $\boldsymbol{V}$, so that $\widetilde{\boldsymbol{U}}$ and $\widetilde{\boldsymbol{V}}$ have the form
$$\widetilde{\boldsymbol{U}} = \boldsymbol{U}_{\mathrm{h}} \otimes \boldsymbol{U}_{\mathrm{v}} \quad \text{and} \quad \widetilde{\boldsymbol{V}} = \boldsymbol{V}_{\mathrm{h}} \otimes \boldsymbol{V}_{\mathrm{v}}$$
where $\otimes$ denotes Kronecker product. This approximation can be obtained by finding the best separable (i.e., $x$-$y$) approximation of the PSF. Construction of this approximation, and computations with it, are $O(N^{3/2})$. This is slightly more than the $O(N \log N)$ computations of the fast transforms, but it is still very efficient, and a separable basis may prove to be a much better approximation than the FFT or DCT for some problems [52, 53, 66].

Note that the cost of matrix vector multiplications with $\boldsymbol{K}$ can usually be done with FFTs (including the spatially variant case; see [67, 68]), so even if preconditioning is not used, the cost of each iteration is at least $O(N \log N)$. Thus, if convergence of the iterative method is much faster when using preconditioning, there can be a dramatic overall savings in computational cost compared to using no preconditioning.

We refer to the approach described in this subsection as a *filtering preconditioner* because the idea is very similar to applying an approximate pseudo-inverse filter at each iteration. These preconditioners can be constructed for both spatially invariant and spatially variant blurs [70]. It is not possible to say that one approach is better than the others; the optimal approach depends on the PSF as well as on the image data.

## 4.3 Steepest Descent Methods

In the steepest descent method, as in the case of the Richardson iteration, we take a step direction that is the negative gradient of $\psi(\boldsymbol{f})$, but we allow the step size to change at each iteration. In particular, the step size is chosen to minimize $\psi(\boldsymbol{f})$ in the direction of the negative gradient. That is,
$$\boldsymbol{f}^{(k+1)} = \boldsymbol{f}^{(k)} + \tau_k \boldsymbol{d}^{(k)} ,$$
where $\boldsymbol{d}^{(k)} = \boldsymbol{K}^\top (\boldsymbol{g} - \boldsymbol{K}\boldsymbol{f})$ and $\tau_k = \arg\min_{\tau} \psi(\boldsymbol{f}^{(k)} + \tau_k \boldsymbol{d}^{(k)})$. It is not difficult to show that
$$\tau_k = \arg\min_{\tau} \psi(\boldsymbol{f}^{(k)} + \tau \boldsymbol{d}^{(k)}) = \frac{\|\boldsymbol{d}^{(k)}\|_2^2}{\|\boldsymbol{K}\boldsymbol{d}^{(k)}\|_2^2} .$$
Thus, the basic iteration requires computing
$$\begin{aligned} \boldsymbol{r}^{(k)} &= \boldsymbol{g} - \boldsymbol{K}\boldsymbol{f}^{(k)} \\ \boldsymbol{d}^{(k)} &= \boldsymbol{K}^\top \boldsymbol{r}^{(k)} \\ \tau_k &= \|\boldsymbol{d}^{(k)}\|_2^2 / \|\boldsymbol{K}\boldsymbol{d}^{(k)}\|_2^2 \\ \boldsymbol{f}^{(k+1)} &= \boldsymbol{f}^{(k)} + \tau_k \boldsymbol{d}^{(k)} \end{aligned}$$
A direct implementation of these steps would require three matrix-vector multiplications – two with $\boldsymbol{K}$ and one with $\boldsymbol{K}^\top$. This can be reduced to two matrix-vector multiplications by observing that
$$\boldsymbol{r}^{(k+1)} = \boldsymbol{g} - \boldsymbol{K}\boldsymbol{f}^{(k+1)} = \boldsymbol{g} - \boldsymbol{K}(\boldsymbol{f}^{(k)} + \tau_k \boldsymbol{d}^{(k)}) = \boldsymbol{r}^{(k)} - \tau_k \boldsymbol{K}\boldsymbol{d}^{(k)} . \tag{14}$$

Using this recursion for $\boldsymbol{r}^{(k+1)}$, the steepest descent algorithm can be written as

**Algorithm: Steepest Descent**

```
given:      $\boldsymbol{K}$, $\boldsymbol{g}$
choose:     initial guess for $\boldsymbol{f}$
compute:
            $\boldsymbol{r} = \boldsymbol{g} - \boldsymbol{K}\boldsymbol{f}$
            while (not stop) do
               $\boldsymbol{d} = \boldsymbol{K}^\top \boldsymbol{r}$
               $\boldsymbol{w} = \boldsymbol{K}\boldsymbol{d}$
               $\tau = \|\boldsymbol{d}\|_2^2 / \|\boldsymbol{w}\|_2^2$
               $\boldsymbol{f} = \boldsymbol{f} + \tau \boldsymbol{d}$
               $\boldsymbol{r} = \boldsymbol{r} - \tau \boldsymbol{w}$
            end while
```

As with the Richardson iteration, the discrepancy principle can be used as a stopping criterion if steepest descent is used for image restoration problems. Because of the variable step size, it is difficult to provide a theoretical analysis of the filtering properties of steepest descent, but some experimental results investigating this topic, and application of filtering based preconditioners for steepest descent can be found in [69].

We mention that other criteria can be used for choosing the step length, such as the Brakhage $\nu$ methods [15], and Barzilai and Borwein's lagged steepest descent scheme [7].

## 4.4 Conjugate Gradient Methods

Conjugate gradient methods are usually much more efficient than gradient descent based methods, such as the Richardson iteration and steepest descent. There are two closely related conjugate gradient based methods for least squares problems: CGLS [12] and LSQR[2] [76, 77]. We focus our discussion on LSQR.

### 4.4.1 LSQR and Filtering

LSQR is based on the Golub-Kahan (sometimes referred to as Lanczos) bidiagonalization (GKB) process, which computes the factorization

$$\boldsymbol{K} = \boldsymbol{W}\boldsymbol{B}\boldsymbol{Y}^\top \tag{15}$$

[2]Note that while the acronym CGLS is obviously obtained from the phrase "conjugate gradient method for least squares", the precise meaning of the acronym LSQR is less clear (it was not explicitly defined in the original papers by Paige and Saunders [76, 77]). LSQR is an iterative method to solve "least squares" problems, and it uses an efficient "$QR$" factorization updating scheme at each iteration.

where $\boldsymbol{W}$ and $\boldsymbol{Y}$ are orthogonal matrices, and $\boldsymbol{B}$ is bidiagonal,

$$\boldsymbol{B} = \begin{bmatrix} \gamma_1 & & & \\ \beta_2 & \gamma_2 & & \\ & \beta_3 & \gamma_3 & \\ & & \ddots & \ddots \end{bmatrix} .$$

Because $\boldsymbol{W}$ and $\boldsymbol{Y}$ are orthogonal matrices, we can rewrite equation (15) as

$$\boldsymbol{K}\boldsymbol{Y} = \boldsymbol{W}\boldsymbol{B} \quad \text{and} \quad \boldsymbol{K}^\top \boldsymbol{W} = \boldsymbol{Y}\boldsymbol{B}^\top .$$

From these relationships, we obtain

$$\boldsymbol{K}\boldsymbol{y}_k = \gamma_k \boldsymbol{w}_k + \beta_{k+1}\boldsymbol{w}_{k+1} \quad \text{and} \quad \boldsymbol{K}^\top \boldsymbol{w}_k = \beta_k \boldsymbol{y}_{k-1} + \gamma_k \boldsymbol{y}_k , \tag{16}$$

where $\boldsymbol{y}_k$ is the $k$-th column of $\boldsymbol{Y}$ and $\boldsymbol{w}_k$ is the $k$-th column of $\boldsymbol{W}$. The recursions in equation (16) can be used to iteratively compute the columns of $\boldsymbol{W}$ and $\boldsymbol{Y}$, as well as the coefficients $\gamma_k$ and $\beta_k$ that define $\boldsymbol{B}$. Specifically, given $\boldsymbol{K}$ and $\boldsymbol{g}$, we compute:

**GKB Iteration**

```
$\beta_1 = \|\boldsymbol{g}\|_2$
$\boldsymbol{w}_1 = \boldsymbol{g}/\beta_1$
$\hat{\boldsymbol{y}}_1 = \boldsymbol{K}^\top \boldsymbol{w}_1$
$\gamma_1 = \|\hat{\boldsymbol{y}}_1\|_2$
$\boldsymbol{y}_1 = \hat{\boldsymbol{y}}_1/\gamma_1$
for $k = 2, 3, \dots$
    $\hat{\boldsymbol{w}}_k = \boldsymbol{K}\boldsymbol{y}_{k-1} - \gamma_{k-1}\boldsymbol{w}_{k-1}$
    $\beta_k = \|\hat{\boldsymbol{w}}_k\|_2$
    $\boldsymbol{w}_k = \hat{\boldsymbol{w}}_k/\beta_k$
    $\hat{\boldsymbol{y}}_k = \boldsymbol{K}^\top \boldsymbol{w}_k - \beta_k \boldsymbol{y}_{k-1}$
    $\gamma_k = \|\hat{\boldsymbol{y}}_k\|_2$
    $\boldsymbol{y}_k = \hat{\boldsymbol{y}}_k/\gamma_k$
```

If we define $\boldsymbol{W}_{k+1} = \begin{bmatrix} \boldsymbol{w}_1 & \cdots & \boldsymbol{w}_k & \boldsymbol{w}_{k+1} \end{bmatrix}$, $\boldsymbol{Y}_k = \begin{bmatrix} \boldsymbol{y}_1 & \cdots & \boldsymbol{y}_k \end{bmatrix}$, and

$$\boldsymbol{B}_k = \begin{bmatrix} \gamma_1 & & & \\ \beta_2 & \gamma_2 & & \\ & \ddots & \ddots & \\ & & \beta_k & \gamma_k \\ & & & \beta_{k+1} \end{bmatrix} . \tag{17}$$

then it is not difficult to show that the GKB iteration results in the matrix relations

$$\boldsymbol{K}^\top \boldsymbol{W}_{k+1} = \boldsymbol{Y}_k \boldsymbol{B}_k^\top + \gamma_{k+1}\boldsymbol{y}_{k+1}\boldsymbol{e}_{k+1}^\top \tag{18}$$

$$\boldsymbol{K}\boldsymbol{Y}_k = \boldsymbol{W}_{k+1}\boldsymbol{B}_k, \tag{19}$$

where $\boldsymbol{e}_{k+1}$ denotes the $(k+1)$st standard unit vector.

Given these relations, and recalling that the first column of $\boldsymbol{W}_k$ is $\boldsymbol{g}/\|\boldsymbol{g}\|_2$, an approximate solution $\boldsymbol{f}_k$ of $\boldsymbol{f}_{\text{true}}$ can be computed from the *projected* least squares problem

$$\min_{\boldsymbol{f}\in R(\boldsymbol{Y}_k)} \|\boldsymbol{K}\boldsymbol{f}-\boldsymbol{g}\|_2^2 = \min_{\hat{\boldsymbol{f}}} \|\boldsymbol{B}_k\hat{\boldsymbol{f}} - \beta_1\boldsymbol{e}_1\|_2^2 \tag{20}$$

where $\beta_1 = \|\boldsymbol{g}\|_2$, and $\boldsymbol{f}_k = \boldsymbol{Y}_k\hat{\boldsymbol{f}}$. It can be shown that $\boldsymbol{f}_k$ converges to the solution of the least squares problem $\min\|\boldsymbol{g}-\boldsymbol{K}\boldsymbol{f}\|_2$. We omit specific implementation details, and refer to [76, 77]. However, we do mention the following important points:

- When $k$ is small, solving the projected problem in equation (20) is very simple because the matrix $\boldsymbol{B}_k$ is a sparse $(k+1)\times k$ matrix. In fact, using plane rotations [39], it is possible to efficiently update the solution of the projected least squares problem from iteration $k-1$ to iteration $k$.
- If we only want to compute $\boldsymbol{f}$, then it is not necessary to save all vectors in the matrices $\boldsymbol{W}_k$ and $\boldsymbol{Y}_k$; updating vectors and coefficients in the $k$-th iteration only requires information from iteration $k-1$.

An important property of GKB is that for small values of $k$ the singular values of the matrix $\boldsymbol{B}_k$ approximate very well certain singular values of $\boldsymbol{K}$, with the quality of the approximation depending on the relative distance between the singular values of $\boldsymbol{K}$; specifically, the larger the relative spread, the better the approximation [12, 40, 85]. For ill-posed inverse problems, the singular values of $\boldsymbol{K}$ decay to and cluster at zero, such as $\sigma_i = O(i^{-c})$ where $c > 1$, or $\sigma_i = O(c^i)$, where we use the big-$O$ notation to mean "on the order of", $0 < c < 1$ and $i = 1, 2, \ldots, n$ [100, 101]. Thus the relative gap between large singular values of $\boldsymbol{K}$ is generally much larger than the relative gap between small singular values. Therefore, if the GKB iteration is applied to a linear system arising from discretization of an ill-posed inverse problem, such as in image restoration, then the singular values of $\boldsymbol{B}_k$ converge very quickly to the largest singular values of $\boldsymbol{K}$.

This property implies that if LSQR is applied to the least squares problem $\min\limits_{\boldsymbol{f}} \|\boldsymbol{K}\boldsymbol{f}-\boldsymbol{g}\|_2$, then at early iterations the approximate solutions $\boldsymbol{f}_k$ will be in a subspace that approximates a subspace spanned by the large singular components of $\boldsymbol{K}$. Thus for $k \ll n$, $\boldsymbol{f}_k$ is a regularized solution. However, eventually $\boldsymbol{f}_k$ should converge to the inverse solution, which is corrupted with noise (recall the discussion in Section 2.1). This means that the iteration index $k$ plays the role of a regularization parameter; if $k$ is too small, then the computed approximation $\boldsymbol{f}_k$ is an over smoothed solution, while if $k$ is too large, $\boldsymbol{f}_k$ is corrupted with noise. More extensive theoretical arguments of this semi-convergence behavior of conjugate gradient methods can be found elsewhere; see [43] and the references therein.

Finally, we mention that the filtering based preconditioners described in Section 4.2.3 can be used with LSQR.

### 4.4.2 A Hybrid Method

One of the main disadvantages of iterative regularization methods is that it can be very difficult to determine appropriate stopping criteria. To address this problem, work has been done to develop hybrid methods that combine variational approaches with iterative methods. That is, an iterative method, such as the LSQR implementation of the conjugate gradient method, is applied to the least squares problem $\min\limits_{\boldsymbol{f}} \|\boldsymbol{K}\boldsymbol{f}-\boldsymbol{g}\|_2^2$, and variational regularization is incorporated within the iteration process.

Instead of early termination of the iteration, hybrid approaches enforce regularization at each iteration of the GKB method. Hybrid methods were first proposed by O'Leary and Simmons in 1981 [75], and later by Björck in 1988 [11]. The basic idea is to regularize the projected least squares problem (20) involving $\boldsymbol{B}_k$, which can be done very cheaply because of the smaller size of $\boldsymbol{B}_k$. More specifically, because the singular values of $\boldsymbol{B}_k$ approximate those of $\boldsymbol{K}$, as the GKB iteration proceeds, the matrix $\boldsymbol{B}_k$ becomes more ill-conditioned. The iteration can be stabilized by including Tikhonov regularization in the projected least squares problem (20), to obtain

$$\min_{\hat{\boldsymbol{f}}} \left\{ \|\boldsymbol{B}_k \hat{\boldsymbol{f}} - \beta_1 \boldsymbol{e}_1\|_2^2 + \lambda^2 \|\hat{\boldsymbol{f}}\|_2^2 \right\} \tag{21}$$

where again $\beta_1 = \|\boldsymbol{g}\|_2$ and $\boldsymbol{f}_k = \boldsymbol{Y}_k \hat{\boldsymbol{f}}$. The regularization parameter $\lambda$ needs to be chosen, either a priori using knowledge of the data, or through regularization parameter choice methods. Thus at each iteration it is necessary to solve a regularized least squares problem involving the $(k+1) \times k$ bidiagonal matrix $\boldsymbol{B}_k$. Notice that since the dimension of $\boldsymbol{B}_k$ is very small compared to $\boldsymbol{K}$, it is much easier to solve for $\hat{\boldsymbol{f}}$ in equation (21) than it is to solve for $\boldsymbol{f}$ in the full Tikhonov regularized problem (5). More importantly, when solving equation (21) one can use sophisticated parameter choice methods [25, 57]to find a suitable $\lambda$ at each iteration.

To summarize, hybrid methods have the following benefits:

- Powerful regularization parameter choice methods can be implemented efficiently on the projected problem.
- Semi-convergence behavior of the relative errors observed in LSQR is avoided, so an imprecise (over) estimate of the stopping iteration does not have a deleterious effect on the computed solution.

Realizing these benefits in practice, though, is nontrivial. Thus, various authors have considered computational and implementation issues, such as robust approaches to choose regularization parameters and stopping iterations; see for example, [13, 19, 25, 44, 56, 57, 60, 75]. The biggest disadvantage of the hybrid approach is that all $\boldsymbol{y}_k$ vectors must be kept throughout the iteration process, and thus the storage requirements grow as the iteration proceeds.

## 4.5 MATLAB Notes

In describing the algorithms, we refer to some of our MATLAB implementations. Most of the methods have the form:

`[f, info] = IR`*method*`(K, g, options)`

where `IR` denotes "Iterative Restoration", *method* will refer to the particular method being described; for example, `IRlsqr` uses LSQR. In the case of hybrid methods, we use the HyBR (hybrid bidiagonal regularization) implementation described in [25], and refer to it as `IRhybr`.

The input and output variables are as follows:

- `K` (required input) is a matrix that can be a MATLAB full or sparse matrix, or a psfMatrix object. Other objects can be used as well, provided the appropriate operators (e.g., `mtimes`) and functions (e.g., `norm`) are overloaded with appropriate methods.
- `g` (required input) is a blurred, noisy image.
- `options` (optional) is a structure, built using `IRset.m`, that can be used to set certain optional input values:

  - Initial guess for $\boldsymbol{f}$; default is $\boldsymbol{f}_0 = \boldsymbol{K}^\top \boldsymbol{g}$.
  - Maximum number of iterations; if $\boldsymbol{K}$ is an $m \times n$ matrix, then the default is $\min(m, n, 100)$.
  - Stopping tolerance for the relative residual $\|\boldsymbol{g} - \boldsymbol{K}\boldsymbol{f}\|_2 / \|\boldsymbol{g}\|_2$; default is $10^{-6}$.
  - Stopping tolerance for the normal equations residual $\|\boldsymbol{K}^\top(\boldsymbol{g} - \boldsymbol{K}\boldsymbol{f})\|_2 / \|\boldsymbol{K}^\top \boldsymbol{g}\|_2$; default is $10^{-6}$.
  - Flag to indicate whether or not an error bar should be displayed on the screen to indicate progress relative to the maximum number of iterations; default action is to show the error bar.
  - The true object, $\boldsymbol{f}_{\text{true}}$, which can be used to return relative error information about the iterations, $\|\boldsymbol{f}_k - \boldsymbol{f}_{\text{true}}\|_2 / \|\boldsymbol{f}_{\text{true}}\|_2$.

  For example, the statement

  ```
  options = IRset('MaxIter', 27, 'Rtol', 1e-4);
  ```

  sets the maximum number of iterations to 27, and the relative residual tolerance to $10^{-4}$.

- `f` is the computed restoration after the iteration completes.
- `info` (optional) is a structure that contains information about the iteration, including:
  - The number of iterations performed.
  - A vector containing the residual norm $\|\boldsymbol{g} - \boldsymbol{K}\boldsymbol{f}_k\|_2$ at each iteration. The first component of this vector is the residual norm corresponding to the initial guess, so the length of the vector is one more than the number of iterations performed by the method.
  - A vector containing the normal equations residual norm $\|\boldsymbol{K}^\top(\boldsymbol{g} - \boldsymbol{K}\boldsymbol{f}_k)\|_2$ at each iteration. The first component of this vector is the normal equations residual norm corresponding to the initial guess, so the length of the vector is one more than the number of iterations performed by the method.
  - A vector containing the norm of the solution $\|\boldsymbol{f}_k\|$ at each iteration. The first component of this vector corresponds to the initial guess, so the length of the vector is one more than the number of iterations performed by the method.
  - If the true solution $\boldsymbol{f}_{\text{true}}$ was set in `options`, then this is a vector containing the relative error norm $\|\boldsymbol{f}_k - \boldsymbol{f}_{\text{true}}\|_2 / \|\boldsymbol{f}_{\text{true}}\|_2$ at each iteration. The first component of this vector corresponds to the initial guess, so the length of the vector is one more than the number of iterations performed by the method.
  - Flag that provides information about why the iteration terminated:

    $1 \Rightarrow$ residual tolerance was satisfied

    $2 \Rightarrow$ normal equations residual tolerance was satisfied

    $3 \Rightarrow$ maximum number of iterations was reached.

For more details, see the MATLAB `help` or `doc` information on `IRset` as well as for any `IR`*method*.

## 5 Iterative Methods with Nonnegativity Constraints

Since pixels in an image generally represent measured intensity values, it is natural to include a nonnegativity constraint in the iterative algorithm. For example, in the case of iterative regularization methods, we may want to develop algorithms that solve

$$\min_{\boldsymbol{f} \geq \boldsymbol{0}} \|\boldsymbol{g} - \boldsymbol{K}\boldsymbol{f}\|_2^2 \,,$$

or in the case of variational regularization, we modify the minimization problem given in equation (4) to include the constraint $\boldsymbol{f} \geq \mathbf{0}$:

$$\min_{\boldsymbol{f}\geq\mathbf{0}} \left\{ \|\boldsymbol{g} - \boldsymbol{K}\boldsymbol{f}\|_2^2 + \lambda^2 \mathcal{J}(\boldsymbol{f}) \right\} .$$

As in the case of the unconstrained problem, the regularization operator $\mathcal{J}$ and the regularization parameter $\lambda$ must be chosen.

In this chapter we focus only on iterative regularization methods that can be obtained as natural extensions of the unconstrained iterative algorithms discussed in previous sections. In the case of variational regularization and nonnegativity constraints in the context of image restoration, we refer to [6, 102]. More general optimization methods can also be used for the variational case; see for example [55, 74].

## 5.1 Projection Methods

In the case of Richardson and steepest descent methods, a nonnegativity constraint can be easily incorporated into the iteration by projecting the iteration $\boldsymbol{f}^{(k)}$ onto the nonnegative orthant. That is, the $i$th entry of a projected vector $\boldsymbol{f}$ is given by

$$[\mathcal{P}(\boldsymbol{f})]_i = \begin{cases} f_i & \text{if} \quad f_i \geq 0 \\ 0 & \text{if} \quad f_i < 0. \end{cases}$$

Thus, to incorporate nonnegativity in each of the algorithms discussed in Sections 4.1, 4.2 and 4.3 all that is needed is to choose an initial guess $\boldsymbol{f} \geq 0$ and to replace the update step

$$\boldsymbol{f} = \boldsymbol{f} + \tau \boldsymbol{d}$$

with

$$\boldsymbol{f} = \mathcal{P}(\boldsymbol{f} + \tau \boldsymbol{d}) .$$

Properties of the projected Richardson method, including preconditioning have been studied; see, for example [79, 16]. The specific case of projected SOR for well posed problems was first studied by Cryer [28], while its use for ill-posed problems in image restoration has been considered more recently in [93, 103]. Properties of a general projected steepest (gradient) descent method can be found in one of many references on numerical optimization methods, such as [55, 74], and a discussion in the context of image restoration can be found in [102]. In the case of the Barzilai and Borwein's lagged steepest descent method with nonnegativity constraints see [58]. Incorporating nonnegativity into conjugate gradient methods is not as simple; for some suggestions at how this might be done see [18, 54], or for more general set theoretic approaches, see [26, 89, 90].

## 5.2 Statistically Motivated Methods

In this subsection we again consider the linear image formation model

$$\boldsymbol{g} = \boldsymbol{K}\boldsymbol{f}_{\text{true}} + \boldsymbol{\eta}$$

and develop iterative methods to approximate $\boldsymbol{f}_{\text{true}}$ that incorporate statistical information.

### 5.2.1 A General Iterative Scheme

We begin by assuming the noise vector, $\boldsymbol{\eta}$, is a random vector with mean $E(\boldsymbol{\eta}) = \mathbf{0}$, and we denote the covariance matrix as $\boldsymbol{C_\eta}$, where $[\boldsymbol{C_\eta}]_{i,j} = E(\eta_i \eta_j)$. The Gauss-Markov theorem [30, 35] states that, if $\boldsymbol{K}$ has full rank, then the *best unbiased linear estimator* (BLUE) for $\boldsymbol{f}_{\text{true}}$ can be found by solving

$$\boldsymbol{K}^\top \boldsymbol{C}_{\boldsymbol{\eta}}^{-1} (\boldsymbol{g} - \boldsymbol{K}\boldsymbol{f}) = \mathbf{0}\,.$$

To incorporate nonnegativity constraints into this model, define a diagonal matrix $\boldsymbol{D}$ with diagonal entries

$$[\boldsymbol{D}]_{i,i} = \begin{cases} 1 & \text{if } [\boldsymbol{f}_{\text{true}}]_i > 0 \\ 0 & \text{if } [\boldsymbol{f}_{\text{true}}]_i = 0 \end{cases}$$

so that $\boldsymbol{K}\boldsymbol{f}_{\text{true}} = \boldsymbol{K}\boldsymbol{D}\boldsymbol{f}_{\text{true}}$. With this notation, it can be shown (see [5]) that the BLUE approximation for $\boldsymbol{f}_{\text{true}}$ can be found by solving

$$\boldsymbol{D}\boldsymbol{K}^\top \boldsymbol{C}_{\boldsymbol{\eta}}^{-1} (\boldsymbol{g} - \boldsymbol{K}\boldsymbol{D}\boldsymbol{f}) = \mathbf{0}\,,$$

or equivalently by solving

$$\boldsymbol{f} \odot \boldsymbol{D}\boldsymbol{K}^\top \boldsymbol{C}_{\boldsymbol{\eta}}^{-1} (\boldsymbol{g} - \boldsymbol{K}\boldsymbol{D}\boldsymbol{f}) = \mathbf{0}\,, \tag{22}$$

where $\odot$ denotes component wise multiplication. Unfortunately it is not possible to directly use the relationship given in equation (22) because $\boldsymbol{D}$ is defined by the unknown true object, $\boldsymbol{f}_{\text{true}}$. However, observe that if $\boldsymbol{f}$ is a best unbiased linear estimator, then $\boldsymbol{f}$ is zero whenever $\boldsymbol{f}_{\text{true}}$ is zero. Furthermore, for any vector $\boldsymbol{v}$, we have $\boldsymbol{f} \odot \boldsymbol{D}\boldsymbol{v} = \boldsymbol{f} \odot \boldsymbol{v}$. It therefore follows that the best unbiased linear estimator is a solution of the nonlinear equation

$$\boldsymbol{f} \odot \boldsymbol{K}^\top \boldsymbol{C}_{\boldsymbol{\eta}}^{-1} (\boldsymbol{g} - \boldsymbol{K}\boldsymbol{f}) = \mathbf{0}\,.$$

One simple approach to solving this nonlinear equation is to use the fixed point iteration

$$\boldsymbol{f}^{(k+1)} = \boldsymbol{f}^{(k)} + \tau_k \boldsymbol{d}^{(k)}\,,$$

where the direction vector is $\boldsymbol{d}^{(k)} = \boldsymbol{f}^{(k)} \odot \boldsymbol{K}^\top \boldsymbol{C}_{\boldsymbol{\eta}}^{-1} \left(\boldsymbol{g} - \boldsymbol{K}\boldsymbol{f}^{(k)}\right)$. Notice the similarity of this basic iteration with that used in the unconstrained steepest descent method. In this case, since we want to maintain nonnegative values in the updated $\boldsymbol{f}^{(k+1)}$, care is needed in choosing the step length $\tau_k$. We do this through a *bounded line search*. Specifically, we first find the step length $\tau_{\text{uc}}$ corresponding to the unconstrained minimization problem

$$\tau_{\text{uc}} = \arg\min_{\tau} \left\| \boldsymbol{C}_{\boldsymbol{\eta}}^{-1/2} \boldsymbol{g} - \boldsymbol{C}_{\boldsymbol{\eta}}^{-1/2} \boldsymbol{K} (\boldsymbol{f}^{(k)} + \tau \boldsymbol{d}^{(k)}) \right\|_2^2 = \frac{\boldsymbol{d}^{(k)\top} \boldsymbol{v}^{(k)}}{\| \boldsymbol{C}_{\boldsymbol{\eta}}^{-1/2} \boldsymbol{K} \boldsymbol{d}^{(k)} \|_2^2}\,, \tag{23}$$

where $\boldsymbol{v}^{(k)} = \boldsymbol{K}^\top \boldsymbol{C}_{\boldsymbol{\eta}}^{-1} (\boldsymbol{g} - \boldsymbol{K}\boldsymbol{f}^{(k)}) = \boldsymbol{K}^\top \boldsymbol{C}_{\boldsymbol{\eta}}^{-1} \boldsymbol{r}^{(k)}$. Next we find the step length that reaches the boundary of the set of nonnegative solutions,

$$\tau_{\text{bd}} = \min \left\{ -\frac{[\boldsymbol{f}^{(k)}]_i}{[\boldsymbol{d}^{(k)}]_i} \;\middle|\; [\boldsymbol{d}^{(k)}]_i < 0 \right\}\,, \tag{24}$$

and use $\tau_k = \min\{\tau_{\mathrm{uc}}, \tau_{\mathrm{bd}}\}$. Thus, if we set $\boldsymbol{r}^{(0)} = \boldsymbol{g} - \boldsymbol{K}\boldsymbol{f}^{(0)}$, a basic iteration to compute a nonnegative approximation of $\boldsymbol{f}_{\mathrm{true}}$ is

$$\begin{aligned}
\boldsymbol{v}^{(k)} &= \boldsymbol{K}^\top \boldsymbol{C}_{\boldsymbol{\eta}}^{-1}\boldsymbol{r}^{(k)} \\
\boldsymbol{d}^{(k)} &= \boldsymbol{f}^{(k)} \odot \boldsymbol{v}^{(k)} \\
\boldsymbol{w}^{(k)} &= \boldsymbol{K}\boldsymbol{d}^{(k)} \\
\tau_k &= \min\{\tau_{\mathrm{uc}}, \tau_{\mathrm{bd}}\} \quad \text{(see equations (23) and (24))} \\
\boldsymbol{f}^{(k+1)} &= \boldsymbol{f}^{(k)} + \tau_k \boldsymbol{d}^{(k)} \\
\boldsymbol{r}^{(k+1)} &= \boldsymbol{r}^{(k)} - \tau_k \boldsymbol{w}^{(k)}
\end{aligned}$$

where the recursion for computing $\boldsymbol{r}^{(k+1)}$ is the same as in the unconstrained steepest descent algorithm; see equation (14).

### 5.2.2 A General Noise Model

We now discuss what to use for $\boldsymbol{C}_\eta^{-1}$ by considering a specific statistical model. In particular, we use the model described in [91, 92] (see also [6, 102]) for CCD arrays, in which each pixel is assumed to be the sum of realizations from three random variables

$$[\boldsymbol{g}]_i = \mathrm{Poisson}\left([\boldsymbol{K}\boldsymbol{f}_{\mathrm{true}}]_i\right) + \mathrm{Poisson}(\beta) + \mathrm{Normal}(0, \sigma^2)$$

where

- $[\boldsymbol{g}]_i$ is the $i$-th pixel in the observed image.
- $\mathrm{Poisson}\left([\boldsymbol{K}\boldsymbol{f}_{\mathrm{true}}]_i\right)$ is a Poisson random variable with parameter (i.e., mean and variance) equal to the $i$th pixel value of the noise-free blurred image, $[\boldsymbol{K}\boldsymbol{f}_{\mathrm{true}}]_i$. This term represents the number of object dependent photoelectrons measured at the $i$-th pixel of the CCD array.
- $\mathrm{Poisson}(\beta)$ is a Poisson random variable with parameter (i.e., mean and variance) equal to $\beta$. This term represents the number of background photoelectrons, arising from both natural and artificial sources, measured at the $i$-th pixel of the CCD array.
- $\mathrm{Normal}(0, \sigma^2)$ is a Gaussian random variable with mean zero and variance $\sigma^2$. This term represents random errors caused by CCD electronics and the analog-to-digital conversion measured in voltages. It is often referred to as *readout noise*.

These random variables are assumed to be independent of each other, as well as independent for each pixel. Thus, we can write the observed image as a realization of a random vector:

$$\boldsymbol{g} = \mathrm{Poisson}(\boldsymbol{K}\boldsymbol{f}_{\mathrm{true}}) + \mathrm{Poisson}(\beta\mathbf{1}) + \mathrm{Normal}(\mathbf{0}, \sigma^2\boldsymbol{I})$$

where $\mathbf{1}$ is a vector of all ones, and $\mathbf{0}$ is a vector of all zeros.

We use the central limit theorem [30, 35] to approximate this model so that it involves a single distribution. First note that in image restoration problems the entries in $\boldsymbol{K}$ are nonnegative, as are the pixel values of $\boldsymbol{f}_{\text{true}}$. Thus, $[\boldsymbol{K}\boldsymbol{f}_{\text{true}}]_i + \beta + \sigma^2 > \sigma^2$, and so

$$\begin{aligned}
\boldsymbol{K}\boldsymbol{f}_{\text{true}} + \beta\mathbf{1} + \sigma^2\mathbf{1} &+ \text{Normal}(\mathbf{0}, \text{diag}(\boldsymbol{K}\boldsymbol{f}_{\text{true}} + \beta\mathbf{1} + \sigma^2\mathbf{1})) \\
&= \text{Normal}(\boldsymbol{K}\boldsymbol{f}_{\text{true}} + \beta\mathbf{1} + \sigma^2\mathbf{1}, \text{diag}(\boldsymbol{K}\boldsymbol{f}_{\text{true}} + \beta\mathbf{1} + \sigma^2\mathbf{1})) \\
&\approx \text{Poisson}(\boldsymbol{K}\boldsymbol{f}_{\text{true}} + \beta\mathbf{1} + \sigma^2\mathbf{1}) \\
&= \text{Poisson}(\boldsymbol{K}\boldsymbol{f}_{\text{true}}) + \text{Poisson}(\beta\mathbf{1}) + \text{Poisson}(\sigma^2\mathbf{1}) \\
&\approx \text{Poisson}(\boldsymbol{K}\boldsymbol{f}_{\text{true}}) + \text{Poisson}(\beta\mathbf{1}) + \text{Normal}(\sigma^2\mathbf{1}, \sigma^2\boldsymbol{I}) \\
&= \text{Poisson}(\boldsymbol{K}\boldsymbol{f}_{\text{true}}) + \text{Poisson}(\beta\mathbf{1}) + \text{Normal}(\mathbf{0}, \sigma^2\boldsymbol{I}) + \sigma^2\mathbf{1} \\
&= \boldsymbol{g} + \sigma^2\mathbf{1}\,,
\end{aligned}$$

where the approximations are due to the central limit theorem, and where the notation diag($\boldsymbol{z}$) is used to denote a diagonal matrix whose $i$-th diagonal entry is given by the $i$-th entry in the vector $\boldsymbol{z}$. We therefore obtain the approximate image formation model

$$\boldsymbol{g} = \boldsymbol{K}\boldsymbol{f}_{\text{true}} + \beta\mathbf{1} + \text{Normal}(\mathbf{0}, \text{diag}(\boldsymbol{K}\boldsymbol{f}_{\text{true}} + \beta\mathbf{1} + \sigma^2\mathbf{1})\,,$$

or

$$\boldsymbol{g} - \beta\mathbf{1} = \boldsymbol{K}\boldsymbol{f}_{\text{true}} + \text{Normal}(\mathbf{0}, \text{diag}(\boldsymbol{K}\boldsymbol{f}_{\text{true}} + \beta\mathbf{1} + \sigma^2\mathbf{1})\,. \tag{25}$$

Observe that equation (25) is in the form discussed in Section 5.2.1, where

$$\boldsymbol{C}_{\boldsymbol{\eta}} = \text{diag}(\boldsymbol{K}\boldsymbol{f}_{\text{true}} + \beta\mathbf{1} + \sigma^2\mathbf{1})\,,$$

and where we use $\boldsymbol{g} - \beta\mathbf{1}$ in place of $\boldsymbol{g}$. Since we do not know $\boldsymbol{f}_{\text{true}}$ we cannot explicitly construct $\boldsymbol{C}_{\boldsymbol{\eta}}$, but we can consider some ways to approximate it.

### 5.2.3 Algorithms

First we remark that often $\beta$ and $\sigma^2$ are known, or can be estimated, and that incorporating this information into $\boldsymbol{C}_{\boldsymbol{\eta}}$ is not difficult. The real issue is how to approximate $\boldsymbol{K}\boldsymbol{f}_{\text{true}}$. Three possibilities are to use $\mathbf{0}$, or $\boldsymbol{g} - \beta\mathbf{1}$, or an iteration dependent approximation $\boldsymbol{K}\boldsymbol{f}^{(k)}$.

If we replace $\boldsymbol{K}\boldsymbol{f}_{\text{true}}$ with $\mathbf{0}$ when constructing an approximation of $\boldsymbol{C}_{\boldsymbol{\eta}}$, and if we assume that $\beta = 0$ (i.e., there is only Gaussian white noise), then the basic iteration described in Section 5.2.1 results in the *modified residual norm steepest descent* (MRNSD) algorithm [5, 54, 73].

**Algorithm: MRNSD**

```
given:     $\boldsymbol{K}$, $\boldsymbol{g}$
choose:    initial guess for $\boldsymbol{f} \geq \boldsymbol{0}$
compute:
           $\boldsymbol{r} = \boldsymbol{g} - \boldsymbol{K}\boldsymbol{f}$
           while (not stop) do
              $\boldsymbol{v} = \boldsymbol{K}^\top \boldsymbol{r}$
              $\boldsymbol{d} = \boldsymbol{f} \odot \boldsymbol{v}$
              $\boldsymbol{w} = \boldsymbol{K}\boldsymbol{d}$
              $\tau_{\text{uc}} = \boldsymbol{d}^\top \boldsymbol{v} / \|\boldsymbol{w}\|_2^2$
              $\tau_{\text{bd}} = \min(-\boldsymbol{f}(\boldsymbol{d} < 0) \oslash \boldsymbol{d}(\boldsymbol{d} < 0))$
              $\tau = \min(\tau_{\text{uc}}, \tau_{\text{bd}})$
              $\boldsymbol{f} = \boldsymbol{f} + \tau \boldsymbol{d}$
              $\boldsymbol{r} = \boldsymbol{r} - \tau \boldsymbol{w}$
           end while
```

In the above algorithm, $\odot$ denotes component wise multiplication and $\oslash$ denotes component wise division. Furthermore, the computation $\boldsymbol{f}(\boldsymbol{d} < 0) \oslash \boldsymbol{d}(\boldsymbol{d} < 0)$ means only perform element wise division when the components of $\boldsymbol{d}$ are negative.

In the case where we replace $\boldsymbol{K}\boldsymbol{f}_{\text{true}}$ with $\boldsymbol{g} - \beta\boldsymbol{1}$ when constructing an approximation of $\boldsymbol{C}_{\boldsymbol{\eta}}$, we obtain the *weighted modified residual norm steepest descent* (WMRNSD) algorithm [5].

**Algorithm: Weighted MRNSD**

```
given:     $\boldsymbol{K}$, $\boldsymbol{g}$, $\sigma^2$, $\beta$
choose:    initial guess for $\boldsymbol{f} \geq \boldsymbol{0}$
compute:
           $\boldsymbol{c} = \boldsymbol{g} + \sigma^2 \boldsymbol{1}$
           $\boldsymbol{g} = \boldsymbol{g} - \beta \boldsymbol{1}$
           $\boldsymbol{r} = \boldsymbol{g} - \boldsymbol{K}\boldsymbol{f}$
           while (not stop) do
              $\boldsymbol{v} = \boldsymbol{K}^\top (\boldsymbol{r} \oslash \boldsymbol{c})$
              $\boldsymbol{d} = \boldsymbol{f} \odot \boldsymbol{v}$
              $\boldsymbol{w} = \boldsymbol{K}\boldsymbol{d}$
              $\tau_{\rm uc} = \boldsymbol{d}^\top \boldsymbol{v} / \|\boldsymbol{w} \oslash \boldsymbol{c}^{1/2}\|_2^2$
              $\tau_{\rm bd} = \min(-\boldsymbol{f}(\boldsymbol{d} < 0) \oslash \boldsymbol{d}(\boldsymbol{d} < 0))$
              $\tau = \min(\tau_{\rm uc}, \tau_{\rm bd})$
              $\boldsymbol{f} = \boldsymbol{f} + \tau \boldsymbol{d}$
              $\boldsymbol{r} = \boldsymbol{r} - \tau \boldsymbol{w}$
           end while
```

The WMRNSD algorithm can be interpreted as a preconditioned version of MRNSD; see [73] for further details. In the algorithm we store the diagonal entries of $\boldsymbol{C}_{\boldsymbol{\eta}}$ in a vector $\boldsymbol{c}$, the square root $\boldsymbol{c}^{1/2}$ is done component wise, and component wise division with $\boldsymbol{c}$ and $\boldsymbol{c}^{1/2}$ is used in place of computing matrix inverses $\boldsymbol{C}_{\boldsymbol{\eta}}^{-1}$ and $\boldsymbol{C}_{\boldsymbol{\eta}}^{-1/2}$.

Finally we consider the case where we replace $\boldsymbol{K}\boldsymbol{f}_{\rm true}$ with the iteration dependent approximation $\boldsymbol{K}\boldsymbol{f}^{(k)}$ when constructing an approximation of $\boldsymbol{C}_{\boldsymbol{\eta}}$.

**Algorithm: $k$-weighted MRNSD**

```
given:      $\boldsymbol{K}$, $\boldsymbol{g}$, $\sigma^2$, $\beta$
choose:     initial guess for $\boldsymbol{f} \geq \boldsymbol{0}$
compute:
            $\boldsymbol{c} = \boldsymbol{K}\boldsymbol{f} + \beta\boldsymbol{1} + \sigma^2\boldsymbol{1}$
            $\boldsymbol{g} = \boldsymbol{g} - \beta\boldsymbol{1}$
            $\boldsymbol{r} = \boldsymbol{g} - \boldsymbol{K}\boldsymbol{f}$
            while (not stop) do
               $\boldsymbol{v} = \boldsymbol{K}^\top(\boldsymbol{r} \oslash \boldsymbol{c})$
               $\boldsymbol{d} = \boldsymbol{f} \odot \boldsymbol{v}$
               $\boldsymbol{w} = \boldsymbol{K}\boldsymbol{d}$
               $\tau_{\mathrm{uc}} = \boldsymbol{d}^\top\boldsymbol{v}/\|\boldsymbol{w} \oslash \boldsymbol{c}^{1/2}\|_2^2$
               $\tau_{\mathrm{bd}} = \min(-\boldsymbol{f}(\boldsymbol{d} < 0) \oslash \boldsymbol{d}(\boldsymbol{d} < 0))$
               $\tau = \min(\tau_{\mathrm{uc}}, \tau_{\mathrm{bd}})$
               $\boldsymbol{f} = \boldsymbol{f} + \tau\boldsymbol{d}$
               $\boldsymbol{r} = \boldsymbol{r} - \tau\boldsymbol{w}$
               $\boldsymbol{c} = \boldsymbol{K}\boldsymbol{f} + \beta\boldsymbol{1} + \sigma^2\boldsymbol{1}$
            end while
```

An interesting observation about using $\boldsymbol{K}\boldsymbol{f}^{(k)}$ when constructing an approximation of $\boldsymbol{C_\eta}$ was made in [5]. Note that if the blur is spatially invariant and we use periodic boundary conditions, then $\boldsymbol{K}^\top\boldsymbol{1} = \boldsymbol{1}$ (in fact, this relation is approximately true for most blurs and boundary conditions). Then, if we take $\tau_k = 1$ for all $k$, the iteration for $\boldsymbol{f}^{(k+1)}$ can be written as

$$\begin{aligned}
\boldsymbol{f}^{(k+1)} &= \boldsymbol{f}^{(k)} + \tau_k\boldsymbol{d}^{(k)} \\
&= \boldsymbol{f}^{(k)} + \boldsymbol{f}^{(k)} \odot \boldsymbol{K}^\top(\boldsymbol{g} - \beta\boldsymbol{1} - \boldsymbol{K}\boldsymbol{f}^{(k)}) \oslash (\boldsymbol{K}\boldsymbol{f}^{(k)} + \beta\boldsymbol{1} + \sigma^2\boldsymbol{1}) \\
&= \boldsymbol{f}^{(k)} + \boldsymbol{f}^{(k)} \odot \boldsymbol{K}^\top\left((\boldsymbol{g} + \sigma^2\boldsymbol{1}) \oslash (\boldsymbol{K}\boldsymbol{f}^{(k)} + \beta\boldsymbol{1} + \sigma^2\boldsymbol{1}) - \boldsymbol{1}\right) \\
&= \boldsymbol{f}^{(k)} + \boldsymbol{f}^{(k)} \odot \left(\boldsymbol{K}^\top\left((\boldsymbol{g} + \sigma^2\boldsymbol{1}) \oslash (\boldsymbol{K}\boldsymbol{f}^{(k)} + \beta\boldsymbol{1} + \sigma^2\boldsymbol{1})\right) - \boldsymbol{1}\right) \\
&= \boldsymbol{f}^{(k)} + \boldsymbol{f}^{(k)} \odot \boldsymbol{K}^\top\left((\boldsymbol{g} + \sigma^2\boldsymbol{1}) \oslash (\boldsymbol{K}\boldsymbol{f}^{(k)} + \beta\boldsymbol{1} + \sigma^2\boldsymbol{1})\right) - \boldsymbol{f}^{(k)} \\
&= \boldsymbol{f}^{(k)} \odot \boldsymbol{K}^\top\left((\boldsymbol{g} + \sigma^2\boldsymbol{1}) \oslash (\boldsymbol{K}\boldsymbol{f}^{(k)} + \beta\boldsymbol{1} + \sigma^2\boldsymbol{1})\right)
\end{aligned}$$

Summarizing this, we obtain the Richardson-Lucy algorithm [61, 82, 102]:

```
Algorithm: Richardson-Lucy

given:    $\boldsymbol{K}$, $\boldsymbol{g}$, $\sigma^2$, $\beta$
choose:   initial guess for $\boldsymbol{f} \geq \boldsymbol{0}$
compute:
          $\boldsymbol{c} = \boldsymbol{K}\boldsymbol{f} + \beta\boldsymbol{1} + \sigma^2\boldsymbol{1}$
          $\boldsymbol{g} = \boldsymbol{g} + \sigma^2\boldsymbol{1}$
          $\boldsymbol{r} = \boldsymbol{g} - \boldsymbol{K}\boldsymbol{f}$
          while (not stop) do
             $\boldsymbol{v} = \boldsymbol{K}^\top(\boldsymbol{g} \oslash \boldsymbol{c})$
             $\boldsymbol{f} = \boldsymbol{f} \odot \boldsymbol{v}$
             $\boldsymbol{r} = \boldsymbol{g} - \boldsymbol{K}\boldsymbol{f}$
             $\boldsymbol{c} = \boldsymbol{K}\boldsymbol{f} + \beta\boldsymbol{1} + \sigma^2\boldsymbol{1}$
          end while
```

Although the residual is not needed in the above algorithm, we compute it anyway since it is often used in the stopping criteria.

It was observed in [5] that the weighted MRNSD method (i.e., the algorithm that uses $\boldsymbol{C}_{\boldsymbol{\eta}}^{-1} = \text{diag}(\boldsymbol{g} + \sigma^2\boldsymbol{1})$ often preforms much better (in terms of speed of convergence as well as in quality of solution) than the other approaches discussed in this section. Some of these issues will be discussed in more detail in the next section.

### 5.3 MATLAB Notes

The algorithms described in this section can be implemented very much like those for the unconstrained image restoration problem. In the case of the statistically motivated problems, it is necessary to include additional information about the noise, such as $\beta$ and $\sigma^2$, if it is available. In all of the data described in Section 3.5, and which is available at `www.mathcs.emory.edu/~nagy/RestoreTools`, we include the noise information in a structure `NoiseInfo`:

- `NoiseInfo.PoissonParam` is the Poisson parameter, $\beta$.
- `NoiseInfo.GaussianStdev` is the standard deviation, $\sigma$, for the white Gaussian noise.

## 6 Examples

In this section we present some numerical results illustrating the effectiveness of the iterative methods described in this chapter. We use the test data described in Section 3 to see how the methods perform on a variety of images and blurring models. We use the relative restoration error

$$\frac{\|\boldsymbol{f}^{(k)} - \boldsymbol{f}_{\text{true}}\|_2}{\|\boldsymbol{f}_{\text{true}}\|_2}$$

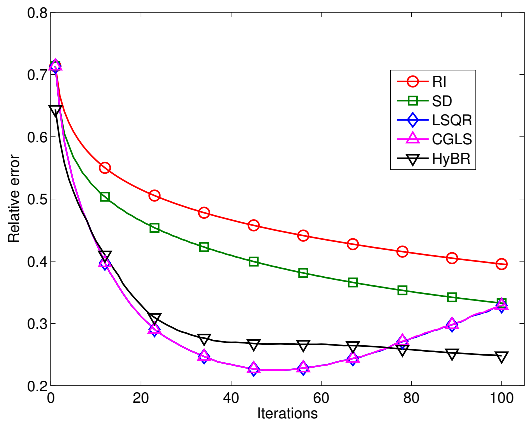
as a measure for the accuracy of the reconstructions. We present the relative errors achieved within the first 100 iterations. The blurred and noisy images $\boldsymbol{g}$ contain both Poisson and

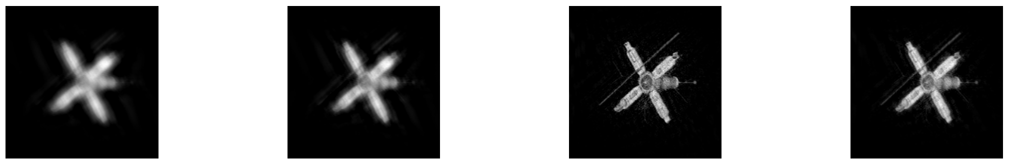
Gaussian components, as described in Section 5.2.2, with Poisson parameter $\beta = 10$ for the background noise, and standard deviation $\sigma = 5$ for the Gaussian readout noise. $\boldsymbol{K}^T\boldsymbol{g}$ is used as an initial guess. All computations were done in MATLAB 7.10.

## 6.1 Unconstrained Iterative Methods

In this subsection we show results obtained for the unconstrained iterative methods. In our implementation of the Richardson iteration we use $\tau = \frac{1}{\|\boldsymbol{K}\|_1 \|\boldsymbol{K}\|_\infty}$ as a default choice for the step length.

Our first example is a test problem that is widely used for testing algorithms in astronomical image restoration. The true object and blurred image ($d/r_0 = 30$), both of size $256 \times 256$, are shown in Fig. 6. In Fig. 12 we compare the performance of the unconstrained iterative methods for this example. The results show that LSQR, CGLS, and HyBR (the hybrid method) significantly outperform the Richardson iteration and steepest descent in achieving better relative errors at the same number of iterations. Note that in Fig. 12, the convergence history of CGLS and LSQR are indistinguishable for this example. Although this makes sense since the two algorithms are mathematically equivalent, it is often stated in the literature that LSQR has slightly better numerical stability properties than CGLS.





RI SD LSQR HYBR

Figure 12: Convergence history of the unconstrained methods and restorations computed at 47th iteration, using the spatially invariant atmospheric turbulence blur data.

The semi-convergence behavior of LSQR and CGLS, which was discussed in Section 4.4, is clearly seen; after achieving a minimal value at iteration $k = 47$, the errors for these two methods increase if the iterations are allowed to proceed. Notice that HyBR avoids this semi-convergence behavior. We note that HyBR has a default stopping criteria, which is based on generalized cross validation [25]. For this test data it suggests terminating at iteration $k = 39$, but we force it run for 100 iterations so that we can observe how it avoids the semi-convergence behavior that occurs with LSQR and CGLS. The solution obtained by CGLS is not shown since its restoration error is the same as that of LSQR. The LSQR solution has the smallest restoration error and is visually the best; see Fig. 12.

For the second experiment, we choose the spatially invariant Gaussian blur example. The $256 \times 256$ true image and the blurred image ($\alpha_1 = 4, \alpha_2 = 2, \rho = 2$) with noise are shown in Fig. 4. Fig. 13 shows the restoration errors and the restored images for different unconstrained iterative methods. We observe small differences in the convergence history of this test problem compared with the previous one. LSQR and CGLS, which again are indistinguishable, reveal the semi-convergence behavior earlier when the methods are run to high iteration counts. Richardson iteration and steepest descent are again the slowest. However the restoration errors corresponding to these methods tend to approach more quickly the minimum restoration error achieved by LSQR and CGLS. The hybrid method shows a similar convergence behavior as in the previous example but it achieves the minimum error of LSQR and CGLS within 100 iterations. We note that HyBR suggests to stop at iteration 16 for this test problem, but we run it for 100 iterations to observe the stable convergence behavior. LSQR and CGLS achieve the minimum error at iteration 29.

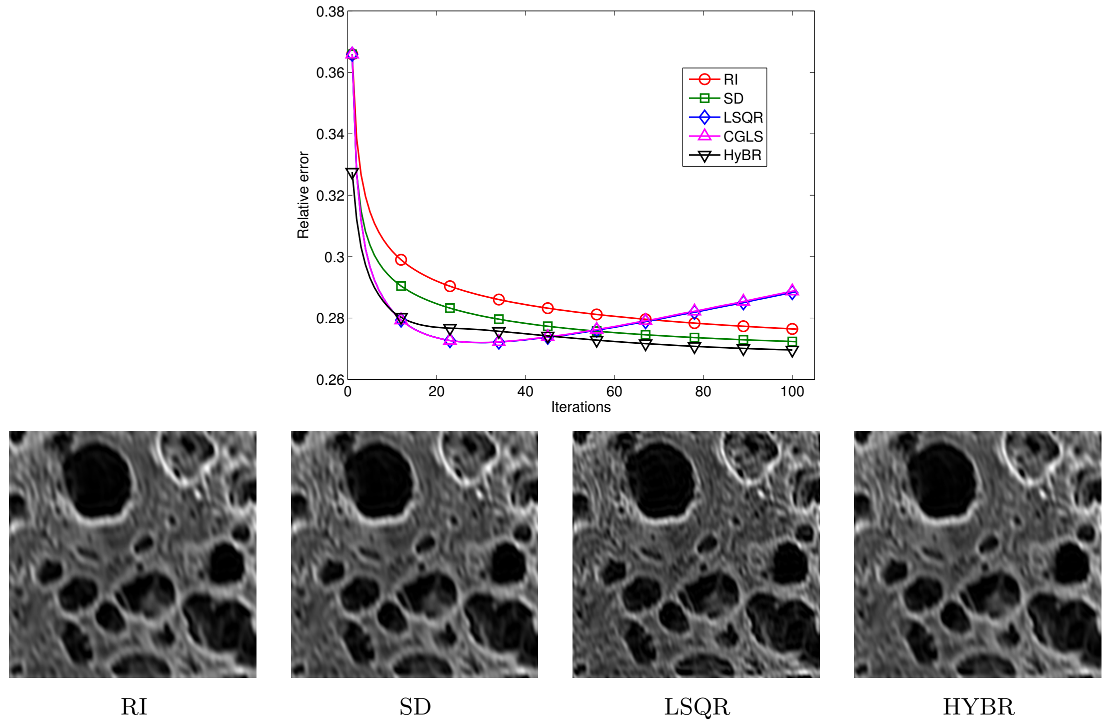


Figure 13: Convergence history of the unconstrained methods and restorations computed at 29th iteration, using the spatially invariant Gaussian blur data.

In the third experiment, we address the performance of the unconstrained iterative methods in the spatially variant Gaussian blur example. The true image is $316 \times 316$ and we use the blurred noisy image corresponding to case 2 in Fig. 8. The minimum restoration error for LSQR and CGLS is achieved at iteration 20 (Fig. 14). It can be seen that HyBR has the lowest relative errors for this test problem. HyBR suggests to stop at iteration 16 for this test problem, but, as with previous examples, we run it for 100 iterations.

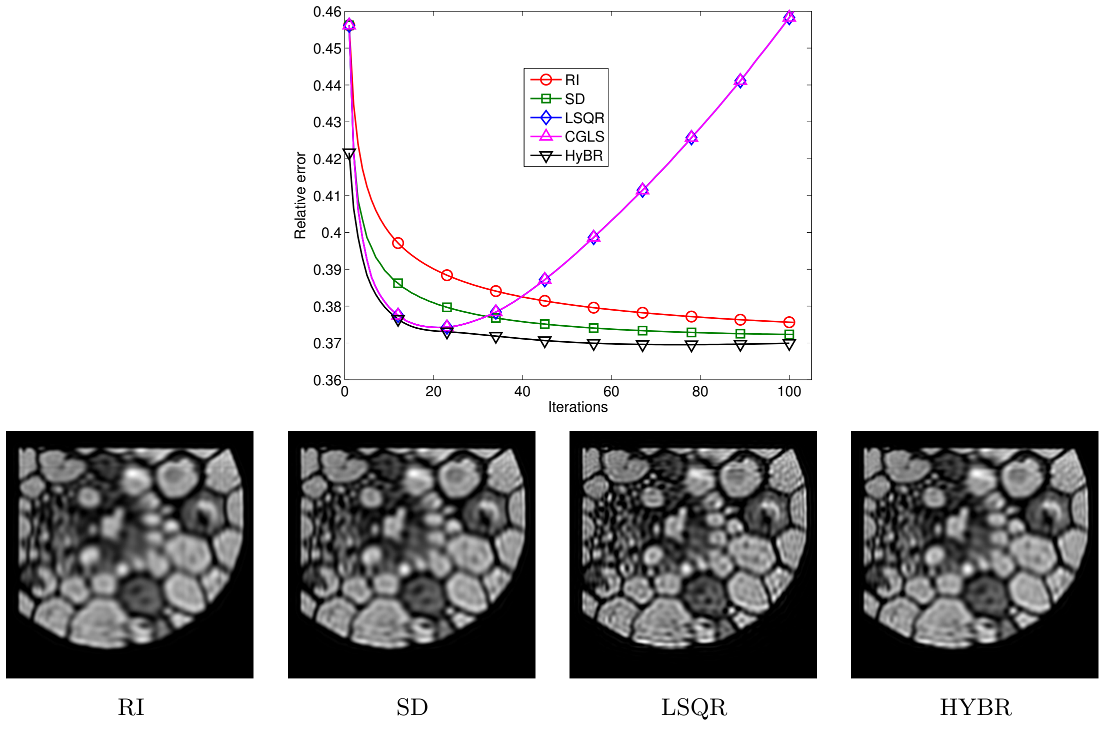
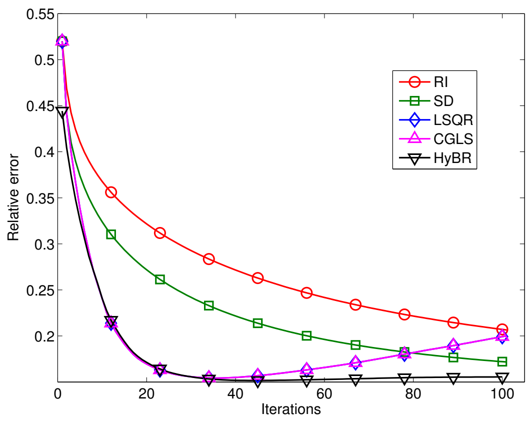
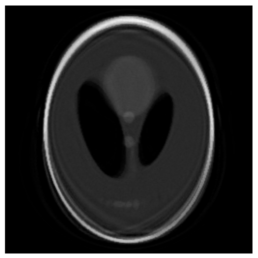


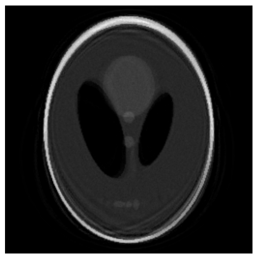
Figure 14: Convergence history of the unconstrained methods and restorations computed at 20th iteration, using the spatially variant Gaussian blur data.

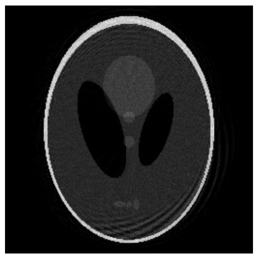
As a last experiment we address the spatially variant motion blur example. The true object and its blurred and noisy image (medium motion) are shown in Fig. 11. The image size is $256 \times 256$. The convergence history and the restored images for this test problem are shown in Fig. 15. The solution with minimum error for LSQR and CGLS is reached at iteration count $k = 35$. We again note that the hybrid method would normally stop at iteration 95 for this test problem (with a stopping condition corresponding to flat GCV curve) but we let it perform the maximum number of iterations.

In general, we can observe that as expected Richardson iteration is the slowest unconstrained iterative method. Steepest descent shows lower relative errors than Richardson iteration but it has a similar slow convergence rate. LSQR and CGLS outperform both Richardson iteration and steepest descent. However both of these methods show the semi-convergence behavior which is common among conjugate gradient type methods. The hybrid method avoids this semi-convergence behavior and in some of the test problems reaches an overall lower relative error compared to LSQR and CGLS.









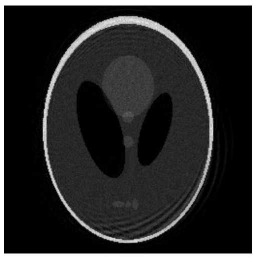


Figure 15: Convergence history of the unconstrained methods and restorations computed at 35th iteration, using the variant motion blur data.

## 6.2 Nonnegativity Constrained Iterative Methods

In this section we use methods that enforce nonnegativity at each iteration. For the spatially invariant atmospheric turbulence blur example we compare the convergence history of all the nonnegatively constrained iterative methods explained in Section 5. Figure 16 shows the restoration errors obtained for the first test problem. Observe that, as with the unconstrained version, the nonnegatively constrained Richardson iteration has the slowest convergence rate. We see that KWMRNSD is the most effective algorithm for the atmospheric blur data. Also, we observed that the rest of our test problems produced similar results, in that WMRNSD and KWMRNSD (k-weighted MRNSD) outperform all other constrained algorithms. Thus for the rest of this section, we show only the convergence history of WMRNSD and KWMRNSD compared to the convergence history of the unconstrained LSQR. This comparison for the atmospheric turbulence blur is shown in Fig. 17.

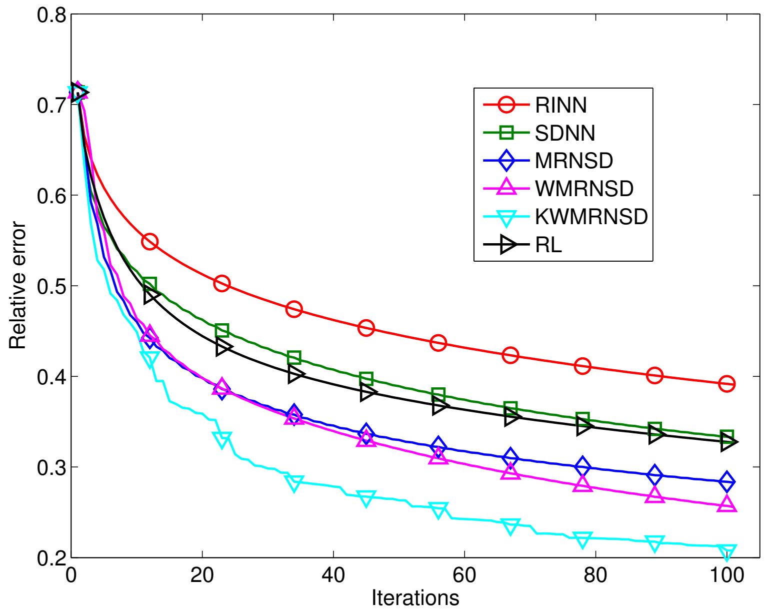
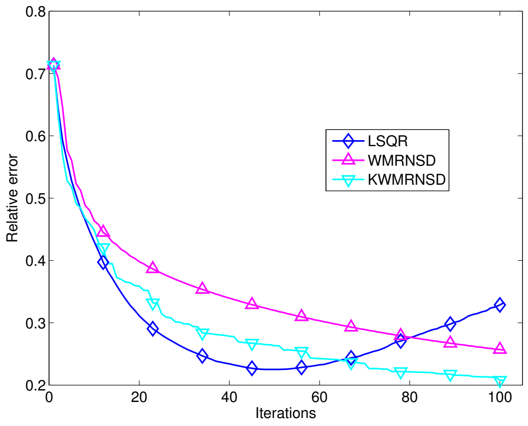


Figure 16: Convergence history of the nonnegatively constrained methods, using the spatially invariant atmospheric turbulence blur data.

We also compare WMRNSD and KWMRNSD with LSQR for the spatially invariant Gaussian blur test problem. The convergence history and restored images are shown in Fig. 18. LSQR reaches its minimum relative error at iteration $k = 29$. Compared with LSQR, the convergence of the KWMRNSD relative errors is very similar at early iterations. After LSQR reaches its minimum error, the relative errors for KWMRNSD keep decreasing. Visually, the restorations look similar.

We now examine the performance of WMRNSD, KWMRNSD, and LSQR when applied to the spatially variant Gaussian blur example. The convergence history of the relative errors and the restored images are shown in Fig. 19. The restored images correspond to the turning point in the convergence-history curve of LSQR. It can be seen that in this test problem the



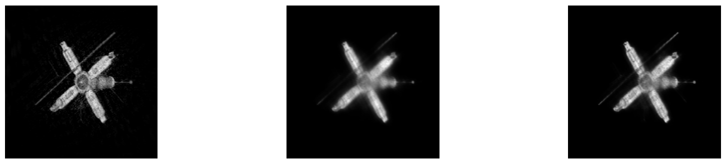
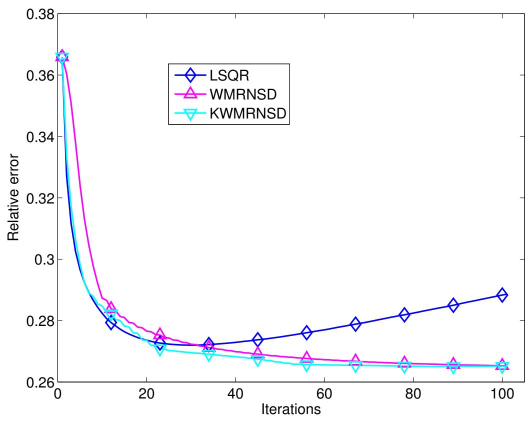


Figure 17: Convergence history of WMRNSD, KWMRNSD, the unconstrained LSQR, and restorations computed at 47th iteration, using the spatially invariant atmospheric turbulence blur data.



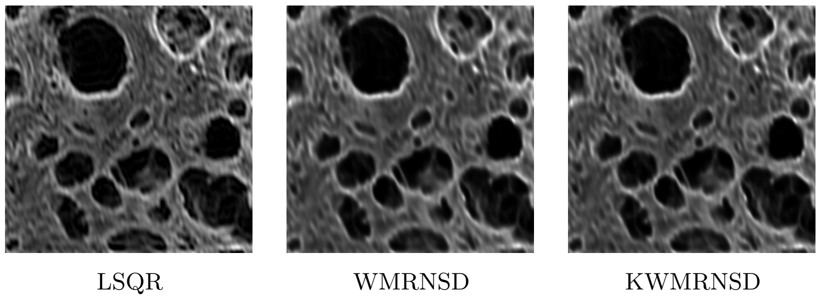
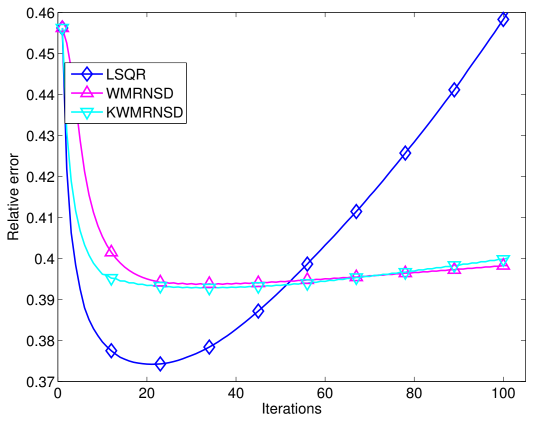
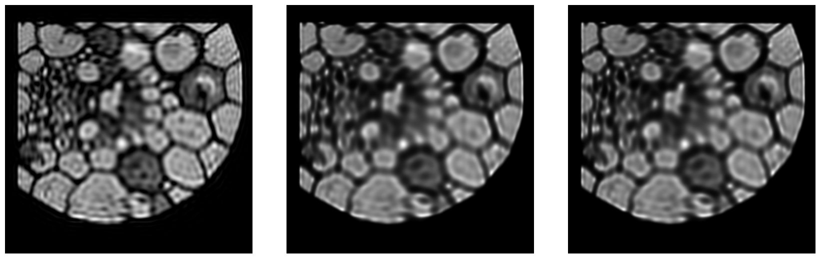


Figure 18: Convergence history of WMRNSD, KWMRNSD, the unconstrained LSQR, and restorations computed at 29th iteration, using the spatially invariant Gaussian blur data.

unconstrained LSQR outperforms the other two constrained methods. Note that one difficulty with the constrained algorithms presented in this chapter is that if a pixel value becomes zero in the iteration, it usually remains zero for subsequent iterations. By looking at the center of the image, such a situation appears to occur with this test problem, and some of the fine details are lost.

We also observe from the convergence plot for this example that the system is highly ill-conditioned. Specifically, inversion of noise corrupts the approximations after just a few iterations; relative errors corresponding to LSQR quickly increase after iteration $k = 20$. We can also observe that the relative errors corresponding to WMRNSD and KWMRNSD show a tendency of increasing at higher iterations. In such cases it is important to have a reliable stopping criteria. Recalling the convergence history (Fig. 14) for the unconstrained problem, HyBR seems to be the best choice for this test problem since it can control the semi-convergence and estimate an appropriate stopping iteration.





LSQR WMRNSD KWMRNSD

Figure 19: Convergence history of WMRNSD, KWMRNSD, the unconstrained LSQR, and restorations computed at 20th iteration, using the spatially variant Gaussian blur data.

The results for the spatially variant motion blur example are shown in Fig. 20. We can observe that KWMRNSD has superior convergence properties compared to WMRNSD at early iterations. Also, both nonnegatively constrained methods perform better than the unconstrained LSQR at higher iterations. The WMRNSD and KWMRNSD restorations show less artifacts compared to LSQR.

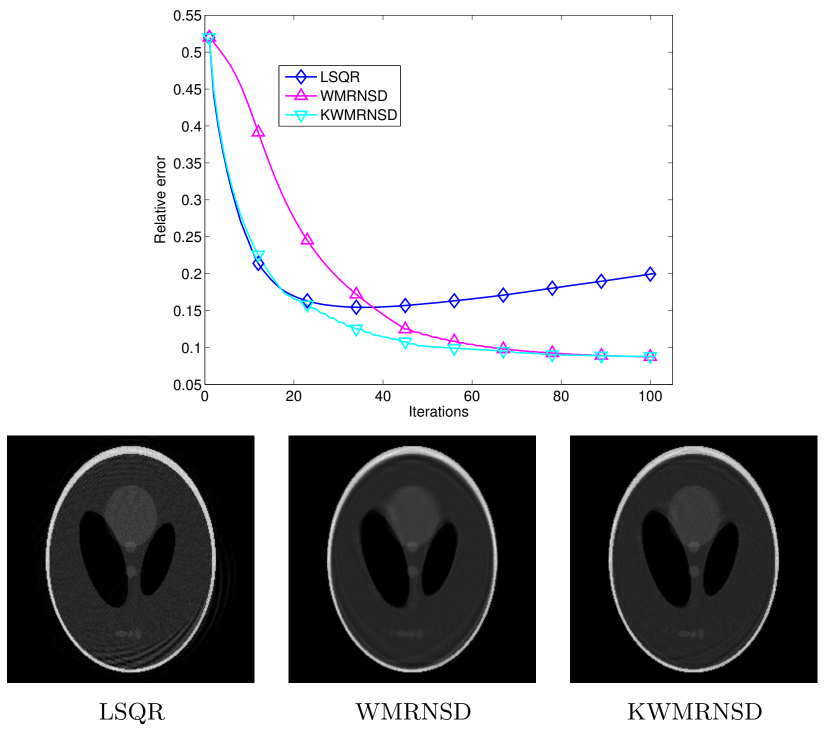


Figure 20: Convergence history of WMRNSD, KWMRNSD, the unconstrained LSQR, and restorations computed at 35th iteration, using the variant motion blur data.

## 7 Concluding Remarks and Open Questions

In this chapter we have described a variety of iterative methods that can be used for image restoration, and compared their performance on a variety of test problems. The data and MATLAB codes used in this chapter are available at www.mathcs.emory.edu/~nagy/RestoreTools. The numerical experiments illustrate that there is not one best method for all problems, which explains the vast literature on this topic. It is important to emphasize that our presentation is far from being a complete survey. For example, we did not discuss Bayesian methods, such as those described in [20], or more sophisticated constrained algorithms such as those described in [3, 6]. Moreover, we only briefly mentioned techniques for choosing regularization parameters and stopping criteria; for some additional work on these topics, see, for example, [4, 57]. One other class of methods that we did not discuss, but is worth further study in the context of constrained iterative algorithms, is projections onto convex sets (POCS) [26, 89, 90]. The POCS approach provides a powerful mechanism, with rigorous mathematical justification, to incorporate a variety of constraints on the object and PSF.

## Acknowledgements

The research of J. Nagy is supported by the United States National Science Foundation (NSF) under grant DMS-0811031, and by the United States Air Force Office of Scientific Research (AFOSR) under grant FA9550-09-1-0487.

## References

[1] H. C. Andrews and B. R. Hunt. *Digital Image Restoration*. Prentice-Hall, Englewood Cliffs, NJ, 1977.

[2] E. S. Angel and A. K. Jain. Restoration of images degraded by spatially varying pointspread functions by a conjugate gradient method. *Applied Optics*, 17:2186–2190, 1978.

[3] J. M. Bardsley. An efficient computational method for total variation-penalized Poisson likelihood estimation. *Inverse Problems and Imaging*, 2(2):167–185, 2008.

[4] J. M. Bardsley. Stopping rules for a nonnegatively constrained iterative method for ill-posed Poisson imaging problems. *BIT*, 48(4):651–664, 2008.

[5] J. M. Bardsley and J. G. Nagy. Covariance-preconditioned iterative methods for nonnegatively constrained astronomical imaging. *SIAM J. Matrix Anal. Appl.*, 27:1184–1197, 2006.

[6] J. M. Bardsley and C. R. Vogel. A nonnegatively constrained convex programming method for image reconstruction. *SIAM J. Sci. Comput.*, 25(4):1326–1343, 2003.

[7] J. Barzilai and J. M. Borwein. Two-point step size gradient methods. *IMA Journal of Numerical Analysis*, 8(1):141–148, 1988.

[8] M. Benzi. Preconditioning techniques for large linear systems: A survey. *Journal of Computational Physics*, 182:418–477, 2002.

[9] M. Benzi. Gianfranco Cimmino's contributions to numerical mathematics. In *Atti del Seminario di Analisi Matematica*, volume speciale: Ciclo di Conferenze in Ricordo di

Gianfranco Cimmino, Marzo-Maggio 2004, pages 87–109. Dipartimento di Matematica dell`Universitá di Bologna, 2005.

[10] M. Bertero and P. Boccacci. *Introduction to Inverse Problems in Imaging.* Institute of Physics Publishing, London, 1998.

[11] Å. Björck. A bidiagonalization algorithm for solving large and sparse ill-posed systems of linear equations. *BIT*, 28(3):659–670, 1988.

[12] Å. Björck. *Numerical Methods for Least Squares Problems.* SIAM, Philadelphia, PA, 1996.

[13] Å. Björck, E. Grimme, and P. van Dooren. An implicit shift bidiagonalization algorithm for ill-posed systems. *BIT*, 34(4):510–534, 1994.

[14] A. F. Boden, D. C. Redding, R. J. Hanisch, and J. Mo. Massively parallel spatially-variant maximum likelihood restoration of Hubble Space Telescope imagery. *J. Opt. Soc. Am. A*, 13:1537–1545, 1996.

[15] H. Brakhage. On ill-posed problems and the method of conjugate gradients. In H. W. Engl and C. W. Groetsch, editors, *Inverse and Ill-Posed Problems*, pages 165–175. Academic Press, Boston, 1987.

[16] P. Brianzi, F. Di Bendetto, and C. Estatico. Improvement of space-invariant image deblurring by preconditioned Landweber iterations. *SIAM J. Sci. Comput.*, 30:1430–1458, 2008.

[17] C. Calvetti, P. C. Hansen, and L. Reichel. L-curve curvature bounds via Lanczos bidiagonalization. *Electron. Trans. Numer. Anal.*, 14:20–35, 2002.

[18] D. Calvetti, G. Landi, L. Reichel, and F. Sgallari. Non-negativity and iterative methods for ill-posed problems. *Inverse Problems*, 20:1747–1758, 2004.

[19] D. Calvetti and L. Reichel. Tikhonov regularization of large linear problems. *BIT*, 43(2):263–283, 2003.

[20] D. Calvetti and E. Somersalo. *Introduction to Bayesian Scientific Computing.* Springer-Verlag, New York, 2007.

[21] E. J. Candès, J. K. Romberg, and T. Tao. Stable signal recovery from incomplete and inaccurate measurements. *Comm. Pure Appl. Math.*, 59(8):1207–1223, 2006.

[22] R. H. Chan, J. G. Nagy, and R. J. Plemmons. FFT-based preconditioners for Toeplitz-block least squares. *SIAM J. Numer. Anal.*, 30:1740–1768, 1993.

[23] R. H. Chan and M. K. Ng. Conjugate gradient methods for Toeplitz systems. *SIAM Review*, 38:427–482, 1996.

[24] T. F. Chan and J. Shen. *Image Processing and Analysis: Variational, PDE, Wavelet, and Stochastic Methods.* SIAM, Philadelphia, PA, 2005.

[25] 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.

[26] P. L. Combettes. The foundations of set theoretic estimation. *Proceedings of the IEEE*, 91:182–208, 1993.

[27] P. L. Combettes. Convex set theoretic image recovery by extrapolated iterations of parallel subgradient projections. *IEEE Trans. Image Proc.*, 6:493–506, 1997.

[28] C. W. Cryer. The solution of a quadratic programming problem using systematic overrelaxation. *SIAM J. Control*, 9(3):385–392, 1971.

[29] T. A. Davis. *Direct Methods for Sparse Linear Systems.* SIAM, Philadelphia, PA, 2006.

[30] M. H. DeGroot. *Probability and Statistics.* Addison-Wesley, Reading, MA, 1989.

[31] H. W. Engl, M. Hanke, and A. Neubauer. *Regularization of Inverse Problems.* Kluwer Academic Publishers, Dordrecht, 2000.

[32] T. L. Faber, N. Raghunath, D. Tudorascu, and J. R. Votaw. Motion correction of PET brain images through deconvolution: I. Theoretical development and analysis in software simulations. *Phys. Med. Biol.*, 54(3):797–811, 2009.

[33] M. Faisal, A. D. Lanterman, D. L. Snyder, and R. L. White. Implementation of a modified Richardson-Lucy method for image restoration on a massively parallel computer to compensate for space-variant point spread function of a charge-coupled device camera. *J. Opt. Soc. Am. A*, 12:2593–2603, 1995.

[34] Y.-W. Fan and J. G. Nagy. Synthetic boundary conditions for image deblurring. *Linear Alg. Applic.,*, 434:2244–2268, 2010.

[35] W. Feller. *An Introduction to Probability Theory and Its Applications.* Wiley, New York, 1971.

[36] M. A. T. Figueiredo, R. D. Nowak, and S. J. Wright. Gradient projection for sparse reconstruction: Application to compressed sensing and other inverse problems. *IEEE J. Selected Topics in Signal Processing*, 1(4):586–597, 2007.

[37] D. A. Fish, J. Grochmalicki, and E. R. Pike. Scanning singular-value-decomposition method for restoration of images with space-variant blur. *J. Opt. Soc. Am. A*, 13:1–6, 1996.

[38] G. H. Golub, M. Heath, and G. Wahba. Generalized cross-validation as a method for choosing a good ridge parameter. *Technometrics*, 21(2):215–223, 1979.

[39] G. H. Golub and C. Van Loan. *Matrix Computations, third edition.* Johns Hopkins Press, 1996.

[40] G. H. Golub, F. T. Luk, and M. L. Overton. A block Lanczos method for computing the singular values and corresponding singular vectors of a matrix. *ACM Trans. Math Software*, 7(2):149–169, 1981.

[41] A. Greenbaum. *Iterative Methods for Solving Linear Systems.* SIAM, Philadelphia, 1997.

[42] C. W. Groetsch. *The Theory of Tikhonov Regularization for Fredholm Integral Equations of the First Kind.* Pitman, Boston, 1984.

[43] M. Hanke. *Conjugate Gradient Type Methods for Ill-Posed Problems.* Pitman Research Notes in Mathematics, Longman Scientific & Technical, Harlow, Essex, 1995.

[44] M. Hanke. On Lanczos based methods for the regularization of discrete ill-posed problems. *BIT*, 41(5):1008–1018, 2001.

[45] M. Hanke and J. G. Nagy. Restoration of atmospherically blurred images by symmetric indefinite conjugate gradient techniques. *Inverse Problems*, 12:157–173, 1996.

[46] M. Hanke, J. G. Nagy, and R. J. Plemmons. Preconditioned iterative regularization for ill-posed problems. In L. Reichel, A. Ruttan, and R. S. Varga, editors, *Numerical Linear Algebra*, pages 141–163. de Gruyter, Berlin, 1993.

[47] P. C. Hansen. *Rank-deficient and discrete ill-posed problems.* SIAM, Philadelphia, PA, 1997.

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

[49] P. C. Hansen, J. G. Nagy, and D. P. O'Leary. *Deblurring Images: Matrices, Spectra and Filtering*. SIAM, Philadelphia, PA, 2006.

[50] A. K. Jain. *Fundamentals of Digital Image Processing*. Prentice-Hall, Englewood Cliffs, NJ, 1989.

[51] S. M. Jefferies and M. Hart. Deconvolution from wave front sensing using the frozen flow hypothesis. *Optics Express*, 19:1975–1984, 2011.

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

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

[54] L. Kaufman. Maximum likelihood, least squares, and penalized least squares for PET. *IEEE Trans. Med. Imag.*, 12:200–214, 1993.

[55] C. T. Kelley. *Iterative Methods for Optimization*. SIAM, Philadelphia, 1999.

[56] M. E. Kilmer, P. C. Hansen, and M. I. Español. A projection-based approach to general-form Tikhonov regularization. *SIAM J. Sci. Comput.*, 29(1):315–330, 2007.

[57] M. E. Kilmer and D. P. O'Leary. Choosing regularization parameters in iterative methods for ill-posed problems. *SIAM J. Matrix Anal. Appl.*, 22(4):1204–1221, 2001.

[58] D. Kim, S. Sra, and I. S. Dhillon. A non-monotonic method for large-scale nonnegative least squares. http://www.optimization-online.org/DB_FILE/2010/05/2629.pdf, 2011.

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

[60] R. M. Larsen. *Lanczos Bidiagonalization with Partial Reorthogonalization*. PhD thesis, Dept. of Computer Science, University of Aarhus, Denmark, 1998.

[61] B. Lucy. An iterative method for the rectification of observed distributions. *Astronomical Journal*, 79:745–754, 1974.

[62] S. R. McNown and B. R. Hunt. Approximate shift-invariance by warping shift-variant systems. In R. J. Hanisch and R. L. White, editors, *The Restoration of HST Images and Spectra II*, pages 181–187, 1994.

[63] K. Miller. Least squares methods for ill-posed problems with a prescribed bound. *SIAM J. Math. Anal.*, 1(1):52–74, 1970.

[64] V. A. Morozov. *Regularization Methods for Ill-Posed Problems*. CRC Press, Boca Raton, FL, 1993.

[65] J. G. Nagy and J. Kamm. Kronecker product and SVD approximations for separable spatially variant blurs. In F. T. Luk, editor, *Advanced Signal Processing Algorithms, Architectures, and Implementations VIII*, volume 3461, pages 358–369. SPIE, 1998.

[66] 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.

[67] J. G. Nagy and D. P. O'Leary. Fast iterative image restoration with a spatially varying PSF. In F. T. Luk, editor, *Advanced Signal Processing Algorithms, Architectures, and Implementations VII*, volume 3162, pages 388–399. SPIE, 1997.

[68] J. G. Nagy and D. P. O'Leary. Restoring images degraded by spatially-variant blur. *SIAM J. Sci. Comput.*, 19:1063–1082, 1998.

[69] J. G. Nagy and K. M. Palmer. Steepest descent, CG, and iterative regularization of ill-posed problems. *BIT*, 43(5):1003–1017, 2005.

[70] J. G. Nagy, K. M. Palmer, and L. Perrone. Iterative methods for image deblurring: A Matlab object oriented approach. *Numerical Algorithms*, 36:73–93, 2004.
See also: http://www.mathcs.emory.edu/~nagy/RestoreTools.

[71] J. G. Nagy, V. P. Pauca, R. J. Plemmons, and T. C. Torgersen. Space-varying restoration of optical images. *J. Optical Soc. Amer. A*, 14:3162–3174, 1997.

[72] J. G. Nagy, R. J. Plemmons, and T. C. Torgersen. Iterative image restoration using approximate inverse preconditioning. *IEEE Trans. on Image Processing*, 15:1151–1162, 1996.

[73] J. G. Nagy and Z. Strakoš. Enforcing nonnegativity in image reconstruction algorithms. In D. C. Wilson, et. al., editor, *Mathematical Modeling, Estimation, and Imaging*, volume 3461, pages 182–190. SPIE, 2000.

[74] J. Nocedal and S. Wright. *Numerical Optimization*. Springer, New York, 1999.

[75] D. P. O'Leary and J. A. Simmons. A bidiagonalization-regularization procedure for large scale discretizations of ill-posed problems. *SIAM J. Sci. Stat. Comp.*, 2(4):474–489, 1981.

[76] C. C. Paige and M. A. Saunders. Algorithm 583 LSQR: Sparse linear equations and sparse least squares problems. *ACM Trans. Math. Soft.*, 8:195–209, 1982.

[77] C. C. Paige and M. A. Saunders. LSQR: An algorithm for sparse linear equations and sparse least squares. *ACM Trans. Math. Soft.*, 8:43–71, 1982.

[78] D. L. Phillips. A technique for the numerical solution of certain integral equations of the first kind. *Journal of the Association for Computing Machinery*, 9(1):84–97, 1962.

[79] M. Piana and M. Bertero. Projected Landweber method and preconditioning. *Inverse Problems*, 13:441–463, 1997.

[80] N. Raghunath, T. L. Faber, S. Suryanarayanan, and J. R. Votaw. Motion correction of PET brain images through deconvolution: II. Practical implementation and algorithm optimization. *Phys. Med. Biol.*, 54(3):813–829, 2009.

[81] S. J. Reeves. Fast image restoration without boundary artifacts. *IEEE Trans. Image Proc.*, 14:1448–1453, 2005.

[82] W. H. Richardson. Bayesian-based iterative methods for image restoration. *J. Optical Soc. Amer.*, 62:55–59, 1972.

[83] G. M. Robbins and T. S. Huang. Inverse filtering for linear shift-variant imaging systems. *Proc. of the IEEE*, 60:862–872, 1972.

[84] L. I. Rudin, S. Osher, and E. Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D*, 60:259–268, 1992.

[85] Y. Saad. On the rates of convergence of the Lanczos and the block-Lanczos methods. *SIAM J. Numer. Anal.*, 17(5):687–706, 1980.

[86] Y. Saad. *Iterative Methods for Sparse Linear Systems, 2nd Ed.* SIAM, Philadelphia, 2003.

[87] A. A. Sawchuk. Space-variant image restoration by coordinate transformations. *J. Opt. Soc. Am.*, 64:138–144, 1974.

[88] S. Serra-Capizzano. A note on antireflective boundary conditions and fast deblurring models. *SIAM J. Sci. Comput.*, 25(4):1307–1325, 2003.

[89] M. I. Sezan and H. J. Trussell. Prototype image constraints for set-theoretic image restoration. *IEEE Trans. Signal Processing*, 39:2275–2285, 1991.

[90] G. Sharma and H. J. Trussell. Set theoretic signal restoration using an error in variables criterion. *IEEE Trans. Image Processing*, 6:1692–1697, 1997.

[91] D. L. Snyder, C. W. Hammoud, and R. L. White. Image recovery from data acquired with a charge-coupled-device camera. *J. Opt. Soc. Am. A*, 10:1014–1023, 1993.

[92] D. L. Snyder, C. W. Helstrom, and A. D. Lanterman. Compensation for readout noise in CCD images. *J. Opt. Soc. Am. A*, 12:272–283, 1994.

[93] V. N. Strakhov and S. V. Vorontsov. Digital image deblurring with SOR. *Inverse Problems*, 24(2):doi: 10.1088/0266–5611/24/2/025024, 2008.

[94] A. N. Tikhonov. Regularization of incorrectly posed problems. *Soviet Math. Doklady*, 4:1624–1627, 1963.

[95] A. N. Tikhonov. Solution of incorrectly formulated problems and the regularization method. *Soviet Math. Doklady*, 4:501–504, 1963.

[96] A. N. Tikhonov and V. Y. Arsenin. *Solutions of Ill-Posed Problems.* Winston, Washington, D. C., 1977.

[97] H. J. Trussell and S. Fogel. Identification and restoration of spatially variant motion blurs in sequential images. *IEEE Trans. Image Proc.*, 1:123–126, 1992.

[98] H. J. Trussell and B. R. Hunt. Image restoration of space-variant blurs by sectional methods. *IEEE Trans. Acoust. Speech, Signal Processing*, 26:608–609, 1978.

[99] Y. Tsaig and D. L. Donoho. Extensions of compressed sensing. *Signal Processing*, 86(3):549–571, 2006.

[100] J. M. Varah. Pitfalls in the numerical solution of linear ill-posed problems. *SIAM J. Sci. Stat. Comp.*, 4(2):164–176, 1983.

[101] C. R. Vogel. Optimal choice of a truncation level for the truncated SVD solution of linear first kind integral equations when data are noisy. *SIAM J. Numer. Anal.*, 23(1):109–117, 1986.

[102] C. R. Vogel. *Computational Methods for Inverse Problems.* SIAM, Philadelphia, PA, 2002.

[103] S. V. Vorontsov, V. N. Strakhov, S. M. Jefferies, and K. J. Borelli. Deconvolution of astronomical images using SOR with adaptive relaxation. *Optics Express*, 19(14):13509–13524, 2011.