Distribution Agreement

In presenting this thesis or dissertation as a partial fulfillment of the requirements for an advanced degree from Emory University, I hereby grant to Emory University and its agents the non-exclusive license to archive, make accessible, and display my thesis or dissertation in whole or in part in all forms of media, now or hereafter known, including display on the world wide web. I understand that I may select some access restrictions as part of the online submission of this thesis or dissertation. I retain all ownership rights to the copyright of the thesis or dissertation. I also retain the right to use in future works (such as articles or books) all or part of this thesis or dissertation.

Signature:

_____ _____

Clarissa Catherine Garvey Date

Truncated Singular Value Decomposition Approximation for Structured Matrices via Kronecker Product Summation Decomposition

By

Clarissa Catherine Garvey

Doctor of Philosophy

Computer Science and Informatics

James Nagy

Advisor

Lars Ruthotto

Committee Member

Jun Kong

Committee Member

Bree Ettinger

Committee Member

Accepted:

Lisa A. Tedesco, Ph.D.

Dean of the James T. Laney School of Graduate Studies

Date

Truncated Singular Value Decomposition Approximation for Structured Matrices via Kronecker Product Summation Decomposition

By

Clarissa Catherine Garvey
B.S., Rochester Institute of Technology, 2013
M.S., Emory University, 2016

Advisor: James Nagy, Ph.D.

An abstract of
A dissertation submitted to the Faculty of the
James T. Laney School of Graduate Studies of Emory University
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy
in Computer Science and Informatics
2018

Abstract

Truncated Singular Value Decomposition Approximation for Structured Matrices via Kronecker Product Summation Decomposition

By Clarissa Catherine Garvey

Singular value decompositions are a particularly attractive matrix factorization for ill-posed problems because singular value magnitudes reveal information about the relative importance of data in the matrix. However, computing a singular value decomposition is typically computationally infeasible for large problems, as the cost for traditional methods, such as Lanczos bidiagonalization-based approaches and randomized methods, scales linearly with the number of entries in the matrix times the number of singular values computed. In this work we present two new algorithms and one new hybrid approach for computing the singular value decomposition of matrices cheaply approximable as an ordered Kronecker summation decomposition. Unlike previous work using ordered Kronecker summation decompositions, the factorizations these methods produce are more accurate for certain classes of matrices and have nonnegative singular values. The three proposed methods are also faster, with lower computational and spatial complexity, although also lower accuracy, than traditional methods. Our Kronecker-based methods therefore enable singular value decomposition approximations on larger matrices than traditional methods, while providing more accurate results in many cases than previous Kronecker-based singular value decompositions. We demonstrate the efficacy of these methods on a variety of image deconvolution problems for which the image is modeled as a regular grid of data.

Truncated Singular Value Decomposition Approximation for Structured Matrices via Kronecker Product Summation Decomposition

By

Clarissa Catherine Garvey
B.S., Rochester Institute of Technology, 2013
M.S., Emory University, 2016

Advisor: James Nagy, Ph.D.

A dissertation submitted to the Faculty of the
James T. Laney School of Graduate Studies of Emory University
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy
in Computer Science and Informatics
2018

Acknowledgments

First and foremost, I owe a huge thanks to my advisor, Jim Nagy. He is an exceptional advisor, and his unwavering support made the PhD process enjoyable. I am extremely grateful to him.

In addition, there are many other people who have helped me get to this point. I owe a great thanks to them all, including:

- Amanda Faulkner and Randy Jenkins, for enabling me to go to college focusing in science and mathematics.
- Nathan Cahill, for introducing me to image processing, sparking my passion.
- My parents, who have always supported my academic efforts.
- Tammany Grant and Maya Nair, for taking away the option of quitting when I doubted whether I would succeed.
- My brother Nick, who has been one of my biggest supporters, and to whom I owe a great deal of my success.
- And finally but not least, Dan Johnson, who has supported me both from near and afar during this whole process.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Many interesting problems can be formulated as a linear inverse problem contaminated by noise. The motivation for this work is one such problem, namely the image processing problem of deconvolution. In standard deconvolution, a known blurry image and a linear operator describing the blurring process are used to recover an estimate of the true, clear image. Due to the physical realities of imaging, the blurred image is also contaminated with noise [1, 25].

Consider the mathematical formulation, temporarily ignoring noise (we will discuss the noisy formulation in detail in Chapter 2). We have a system $\boldsymbol{K}\boldsymbol{x} = \boldsymbol{d}$ where $\boldsymbol{K}$ is a known matrix of size $N \times N$, $\boldsymbol{d}$ is a known vector, and the vector $\boldsymbol{x}$ is unknown. If $\boldsymbol{K}$ is small, this problem is commonly solved by taking a factorization of $\boldsymbol{K}$ and efficiently solving the linear equation with those factors. If $\boldsymbol{K}$ has special properties, more efficient factorization can be used; for example, Cholesky factorization can be used on symmetric positive definite matrices as opposed to QR factorization [21]. For our problems, $\boldsymbol{K}$ is typically ill-posed [1, 2, 25], which makes a different factorization, the singular value decomposition (SVD), desirable.

Singular value decompositions are a useful but costly factorization method. Solving systems and adding regularization is straightforward once an SVD has been com-

puted. But obtaining an SVD is not so simple. Computing a full SVD on an unstructured $N \times N$ matrix takes $O(N^3)$ time, with a worse constant than QR and Cholesky factorization [21]. In real applications, this produces a large effect. In addition, the SVD at least doubles storage costs compared to the original matrix. For sparse matrices, computing an SVD typically results in a non-sparse factorization. As N gets large, computing an SVD therefore becomes prohibitive even for sparse matrices.

In image deconvolution, and in many other problems, the system involved is large. Although scientific images can be small, real images are often at least 1024 pixels in their smaller dimension. For a blurring problem, this results in a huge blurring operator matrix of size greater than a million by a million.

Computationally and spatially cheap approximations of singular value decompositions are desirable to avoid the issues arising from working with massive systems. One such approximation is the truncated SVD, which also has the effect of combating corruption by noise as discussed in Section 2.3. In a truncated SVD, the components of the factorization are constructed up to a certain index $k < N$. This provides a rank-k approximation of the original matrix, and naturally reduces storage from $O(N^2)$ for the full SVD to $O(kN)$ for the truncated SVD when a compact representation is used.

There are several existing methods for computing truncated SVDs. One of the most popular is a Lanczos bidiagonalization approach, described further in Section 7.2. Lanczos methods enjoy high accuracy at the cost of computational time. A more recent category of methods is the randomized framework described by Halko, Martinsson, and Tropp [24], also described further in Section 7.3. This approach uses random initialization to converge fairly quickly to an accurate approximation. Variants of both Lanczos and randomized methods have been created for certain matrix structures, such as symmetric matrices. However, they do not fully exploit a subtle structure arising in image deconvolution.

That structure appears in the matrix blurring operator in image deconvolution

problems. The blur operator is easily represented as a sum of Kronecker products for many types of blur [29,49,50] (also see Section 2.5 for details). The main focus of this work is effectively exploiting the Kronecker summation structure to compute a very cheap truncated SVD (TSVD). Exploiting that structure is not a new idea. Previous work on the subject, described in Chapter 3, used a simple approach to construct a TSVD approximation from a Kronecker product summation [29, 30, 38]. Although highly effective in certain cases, this approach has severe limitations, including the possibility of negative approximate singular values. This baseline method is described in detail in Chapter 3.

Here we propose several alternative approaches, discussing their strengths and weaknesses. First, in Chapter 4, we discuss a method that discards less information than the original approach by using early truncation in the approximation. This method does a worse job than the baseline approach at estimating the smallest computed singular values, which motivates a variant described in Chapter 5. This second approach is more effective than both the baseline and first new approach. It uses significant permutation of the data prior to truncating, improving estimation of the true TSVD. Finally, we describe a hybrid method in Chapter 6 that combines the baseline method with either proposed approach.

We run experiments in all chapters to showcase the strengths and weaknesses of each method. We test each method to compute a direct reconstruction solving the image deconvolution problem, detailed in Section 2.1, with a true, clear satellite image $\boldsymbol{x}$ shown in Figure 1.1. The resulting reconstruction from each chapter and on each of the three example blurs is shown in Figure 1.2. Each row corresponds to a different example, with the blurred image for that example shown in the final column. Although the most visually appealing image is not always the most accurate as discussed in Chapters 4-5 and Appendix A, we can see that our proposed methods are a significant improvement on the baseline method described in Chapter 3. Similar

summarizations for the other experiments we run can be found in Appendices B and C.

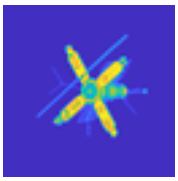


Figure 1.1: True satellite image. This is the true $\boldsymbol{x}$ used in all examples.

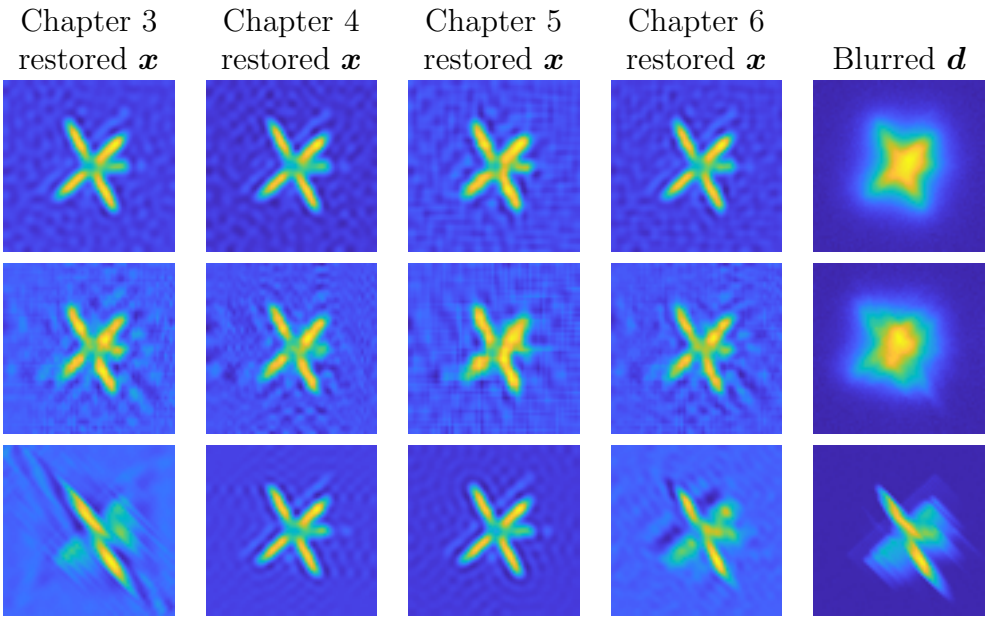


Figure 1.2: All-way restoration comparison. Each row is a different test example with a blurred image as shown on the right. The restorations for each method are shown in the first four columns.

These approaches were designed to work for the image deconvolution problem, but they can be generalized. Any problem involving matrices with similar structures (detailed in Section 2.5) can benefit from these methods.

Furthermore, the image deconvolution problem has a variant called blind deconvolution in which the matrix $\boldsymbol{K}$ is not known exactly. We detail one approach to extend these methods to a blind deconvolution framework in Chapter 8, although we conclude that the benefits to using that specific framework are limited.

The primary strength of our methods is that they enable more compact storage and cheaper computations. Whereas traditional TSVD methods, as previously mentioned, may have storage costs that scale linearly with the size of the matrix for a fixed truncation index, our methods' storage scales with the square root of the matrix size. This enables the factorization to be computed for matrices that are much larger than is possible with methods like Lanczos-based and randomized methods, which are agnostic to the Kronecker structure. Because of the compact representation, constructing and applying the proposed factorizations is cheap. A comparison of the accuracy and computational cost of our method and existing methods is given in Chapter 7.

The trade-off for our methods' speed and compression is accuracy. While other comparable methods can be used to extremely high (and tunable) precision, the precision of our method is moderate and not tunable to arbitrary accuracy. Our methods are therefore useful cheap approximations, but not good for applications that require high precision. Fortunately, there are numerous applications for which cheap but not highly accurate approximations are sufficient, including: preconditioning, many image deconvolution problems, and initial guesses for various methods. We explore these experimental frameworks for each of our proposed methods as well as the baseline method. We find that our cheap approximations are powerful tools in these applications.

Chapter 2

Background

This chapter provides the mathematical background for the methods described in the remaining chapters of this work. We detail the theoretical and conceptual underpinnings of image deconvolution, singular value decompositions, Kronecker products, the relationship between Kronecker products and SVDs, and conclude with a brief discussion of how our work relates to comparable work on tensors.

2.1 Image Deconvolution

Image deconvolution, also called restoration and deblurring, is the process of restoring a clear image from a blurry image. There are many approaches to this problem [1, 2, 25]; of them, we focus on a particular model-based approach, where the process causing the image blur is known exactly or approximately. This works well for a variety of imaging applications, such as the original motivating deconvolution problem of astronomical imaging from satellites [2, 7, 32]. Given the blurry image and a model of the blur, we seek to restore the clear image.

In our applications, the blurry image is measured directly, but the blur model is only indirectly measured as a point spread function (PSF). A point spread function indicates how a single point source of light is spread out due to the blur in the optical

system. An example of a Gaussian PSF from the RestoreTools package [39] is shown in Figure 2.1.

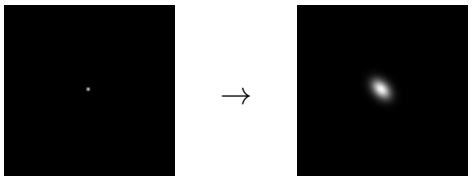


Figure 2.1: An example PSF. The original point source, on the left, is blurred so the light spreads as shown on the right.

This blur may be caused by the atmosphere, for example, when imaging stars from ground-based telescopes on the earth [3, 25, 48]; in this case, the blur may be approximately Gaussian [2, p. 28]. Or, the blur could result from motion of a camera relative to the object in the image [1, 2, 25]. There are many sources of blur and, as detailed in Chapters 3-6, the properties of the PSF can significantly impact the restoration.

Images can be thought of as a collection of several point sources of light [25, p. 4]. If we know how a single point source of light would be blurred, we can figure out how the many point sources of light in a clear image are blurred to create a blurry image. Mathematically, we formulate the blur as a linear operator acting on the image as introduced in Chapter 1. With knowledge of the operator, we try to undo the blur. We will see in Section 2.3 why naively approaching this problem can give poor results, but for now we focus on how the linear system is constructed.

The structure of the operator depends on two factors, the first of which is how the PSF applies to the image. It is not always the case that the same PSF applies to the entire image, as different parts of a picture may be blurred differently. We focus on the case where the PSF is indeed constant over the image; the PSF is then called spatially invariant. Spatially invariant PSFs can result in special structures, depending on a second factor, boundary conditions.

Boundary conditions prescribe how the PSF is treated at image boundaries. In image restoration problems, we typically seek to restore an image that is the same size as the blurred image, but is clear [1, 2, 25]. But a point spread function takes a clear point of light and spreads it out over a larger area. As a result, the blurred image depends not only on what is within the scene captured, but on what is nearby as well. For example, imagine taking a picture of a cat that is sitting next to a chair. Even if the chair is not within the bounds of the captured image, some of the light from the chair may be spread and blurred onto the image of the cat. Similarly, some of the light from the cat is spread outside the boundary of the picture. The blurred image is therefore missing data necessary to complete the restoration.

To solve this issue, we impose boundary conditions on the deblurring problem. Boundary conditions tell us the behavior of data at or beyond the boundary of a system. Typically, we wish to assume that outside our picture, the scene took some easy-to-use form. For example, we may use zero boundary conditions, which assume that there was no light outside the observed scene (which may be realistic, for example, in astronomical imaging). A blurring example with zero boundary conditions is shown in Figure 2.2. Periodic boundary conditions assume that the scene repeats in a tiling fashion, and reflective (also called reflexive) boundary conditions assume that the scene is mirrored at its edges. Choice of boundary condition can significantly impact the success of the restoration. Together, the PSF, its variance over space, and the choice of boundary conditions determine the form of the linear operator acting on the image.

Figure 2.2: An inverted example with a PSF and zero boundary conditions. With inversion, zero boundaries appear white. The image is blurred by the PSF as though the image was surrounded by white in the inverted color space.

This brings us to a proper mathematical formulation of the problem. We formulate the image deblurring problem as

$$\boldsymbol{d} = \boldsymbol{K}\boldsymbol{x} + \boldsymbol{e} \tag{2.1}$$

where:

- $\boldsymbol{d}$ is an observed, blurred image of size $n \times n$ represented as a vector of length $N = n^2$;
- $\boldsymbol{K}$ is a real-valued blurring operator of size $N \times N$, perhaps not exactly known;
- $\boldsymbol{x}$ is an unknown true, clear image represented as a vector of length N; and
- $\boldsymbol{e}$ is an unknown noise (error) in the observed image, represented as a vector of length N.

We start with the blurred image $\boldsymbol{d}$ and an unnamed point spread function matrix. The point spread function is used with the chosen boundary conditions to construct the blurring operator $\boldsymbol{K}$. In real applications the measurement of the blurry image and PSF are not exact, which causes unknown noise to contaminate the system. Nonetheless, this problem is an inverse problem, which we typically solve using the approximation $\boldsymbol{x} \approx \boldsymbol{K}^{-1}\boldsymbol{d}$ which has error $\boldsymbol{K}^{-1}\boldsymbol{e}$. In cases where $\boldsymbol{K}^{-1}$ does not exist,

we use an approximation instead. An example of one such approximation is detailed in Section 2.3.

In the formulation (2.1), the blurred and true images are represented as a vector. The images are transformed from their natural (two-dimensional) matrix form to a vector (one-dimensional) form for computational purposes. In order to transform an image matrix into a vector, we can choose many ways of ordering the elements of the vector. Two natural choices are column-major and row-major ordering. In column-major ordering, elements are placed in the vector consecutively moving down a column until the end, then moving to the top of the next column and continuing. Row-major ordering is similar, but moving left to right along rows until the end and moving to the next row below. Choosing between row and column major ordering can significantly affect memory access patterns for large datasets. As a result, care should be given to this choice. Our work uses column-major ordering.

In the multiplication $\boldsymbol{K}\boldsymbol{x}$, each row in $\boldsymbol{K}$ blurs a corresponding pixel in $\boldsymbol{x}$. That is, each row in $\boldsymbol{K}$ is the PSF shifted and with boundary conditions applied, reshaped into a row vector that blurs a pixel in $\boldsymbol{x}$. Given a PSF and boundary conditions, we can avoid forming $\boldsymbol{K}$ if we only need to compute matrix-vector multiplications. Iterative solution techniques exploit the quick computation that this enables [6, p. 613]. We will use this function multiplication form for some experiments in Chapter 7.

Techniques which rely on factorization of the matrix $\boldsymbol{K}$ instead of using implicit multiplication can benefit from the structure that results from spatially invariant PSFs with given boundary conditions. For row- and column-major ordering, $\boldsymbol{K}$ has a matrix structure that is block-Toeplitz-with-Toeplitz-blocks (BTTB), block-Toeplitz-plus-Hankel-with-Toeplitz-plus-Hankel-blocks (BTHTHB), or block-circulant-with-circulant-blocks (BCCB) for zero, reflective, and periodic boundary conditions respectively [25, pp. 36-38].

The image deconvolution problem is the main focus of our work because the aforementioned structures lend themselves to cheap computation in our proposed algorithm. However, as will become clear in Section 2.5, the techniques described work well on other problems, too, provided the operator $\boldsymbol{K}$ has an exploitable structure.

2.2 Blind Deconvolution

In addition to the standard deconvolution problem (2.1) in which the clear image $\boldsymbol{x}$ and the noise $\boldsymbol{e}$ are unknown, we briefly explore a blind deconvolution framework in Chapter 8. In the blind deconvolution formulation of the linear blur problem (2.1), the blur operator $\boldsymbol{K}$ is not assumed to be exactly known. Instead, the blur operator is treated as an unknown quantity with an initial estimate. Restorations seek to improve the estimation of the blur operator to then improve the restored image.

There are two broad classes of approaches to the blind deconvolution framework. The first and original techniques [8,44], known as a priori methods, first estimate the true blur operator and then use that estimate in a traditional, non-blind deconvolution framework. The focus for these methods is to obtain the best blur operator estimate possible. The second class simultaneously restores improved estimates of the blur operator and the true image. Here, the steps are not separated. There are many approaches to simultaneous blind deconvolution; we refer interested readers to the survey works [7] and [32]: the 1996 survey by Kundur and Hatzinakos [32] gives a broad overview of fundamentals and techniques, and Campisi and Egiazarian's work [7] provides a recent survey from a Bayesian perspective. We use a specific framework detailed in Chapter 8, which was proposed by Dykes et al. [14], to improve our approximate SVD of the blur operator $\boldsymbol{K}$.

2.3 Singular Value Decompositions

The problem formulation (2.1) is a linear inverse problem. One method of solving linear inverse problems is to compute a factorization of the matrix $\boldsymbol{K}$ into a product of matrices for which solving a linear system is easy. It may not be easy to compute $\boldsymbol{K}^{-1}$ directly (if it exists), but computing an explicit or implicit approximate inverse of the factorization can be much more tractable.

There are many factorization techniques. The particular choice of factorization can depend on the structure of the matrix $\boldsymbol{K}$. For example, Cholesky factorization can only be used on matrices which are symmetric positive definite. The singular value decomposition (SVD) is an attractive factorization in part because it enables straightforward solution of linear systems and, as we will see, particularly makes it easy to adjust the solution via regularization if needed. The singular value decomposition decomposes a matrix $\boldsymbol{K}$ into a product $\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$ (or $\boldsymbol{V}^*$, the conjugate transpose, for complex $\boldsymbol{K}$; we assume $\boldsymbol{K}$ is real), where $\boldsymbol{U}$ and $\boldsymbol{V}$ are orthonormal and $\boldsymbol{\Sigma}$ is a non-negative diagonal matrix, typically chosen so that the entries are sorted in descending order going down the diagonal. If and only if the diagonal entries of $\boldsymbol{\Sigma}$ are nonzero, the matrix $\boldsymbol{K}$ is full rank, and the inverse of $\boldsymbol{K}$ is $\boldsymbol{K}^{-1} = \boldsymbol{V}\boldsymbol{\Sigma}^{-1}\boldsymbol{U}^T$ [21].

There are several factors that prohibit the SVD from being used commonly on general problems. The foremost limitation is that computing the singular value decomposition is costly in storage and computation time. The factorization at least doubles the amount of data used to store a matrix (and is much worse for sparse matrices due to loss of sparsity), and the computation takes $O(N^3)$ time to compute for a $N \times N$ matrix, with a larger constant than other factorization methods [21]. For very large matrices, this is prohibitively expensive.

An added difficulty that is not unique to the SVD arises if $\boldsymbol{K}$ is not numerically full rank. The matrix $\boldsymbol{K}$ may be ill-conditioned, being either rank deficient or having singular values that decay towards zero without a significant gap between values.

In the former case, $\boldsymbol{K}$ is not invertible. In the latter case, trying to compute $\boldsymbol{\Sigma}^{-1}$ results in extremely large values on the diagonal. If the i^{th} diagonal entry of $\boldsymbol{\Sigma}$ is σ_i, then the i^{th} entry on the diagonal of $\boldsymbol{\Sigma}^{-1}$ is $\frac{1}{\sigma_i}$ which is large for small σ_i. When then solving the image deconvolution problem (2.1), the restored solution has amplified noise: using $\boldsymbol{u}_i$ to denote the i^{th} column of $\boldsymbol{U}$ (and similarly for $\boldsymbol{V}$), the solution $\boldsymbol{x} + \boldsymbol{V}\boldsymbol{\Sigma}^{-1}\boldsymbol{U}^T\boldsymbol{e} = \boldsymbol{x} + \sum_{i=1}^{N} \frac{\boldsymbol{u}_i^T\boldsymbol{e}}{\sigma_i}\boldsymbol{v}_i$ causes the error to be amplified by division by small σ_i. This corrupts the result. However, we can modify the singular value decomposition to fix this.

Using a truncated version of the singular value decomposition can help both with the cost of the algorithm and the issue of amplifying noise. Instead of computing a full singular value decomposition to exactly represent $\boldsymbol{K}$, we can compute only the first k singular values and vectors, where $k \leq N$. This permits a compact representation where the matrices $\boldsymbol{U}$ and $\boldsymbol{V}$ are size $N \times k$ and the matrix $\boldsymbol{\Sigma}$ is size $k \times k$. The storage cost is then reduced to $O(Nk)$ entries, and the computation time is reduced to $O(N^2k)$. This factorization does not have an exact inverse, but we can compute its pseudoinverse $\boldsymbol{K}^\dagger = \boldsymbol{V}\boldsymbol{\Sigma}^\dagger\boldsymbol{U}^T$. If $\boldsymbol{K}$ is chosen so that small σ_i are removed from the factorization, then the restoration $\boldsymbol{x} + \sum_{i=1}^{k} \frac{\boldsymbol{u}_i^T\boldsymbol{e}}{\sigma_i}\boldsymbol{v}_i$ does not severely amplify the noise $\boldsymbol{e}$. This technique is one of many ways of providing regularization to constrain the solution and reduce error.

Other sources of regularization complement truncation. One common choice of regularization in image processing is Tikhonov regularization [16, 28] which penalizes the norm of the solution $\boldsymbol{x}$. That is, in the least squares formulation of (2.1)

$$\min_{\boldsymbol{x}} \|\boldsymbol{K}\boldsymbol{x} - \boldsymbol{d}\|_2^2$$

we add an additional penalty term:

$$\min_{\boldsymbol{x}} \|\boldsymbol{K}\boldsymbol{x} - \boldsymbol{d}\|_2^2 + \lambda\|\boldsymbol{x}\|_2^2. \tag{2.2}$$

Some forms of Tikhonov regularization do not penalize the norm of $\boldsymbol{x}$ directly, but penalize the norm of a vector $\boldsymbol{Lx}$ for some matrix $\boldsymbol{L}$; for more details, see [6, p. 346], [21, p. 309], [25, p. 72], [45]. For our discussion, we use $\boldsymbol{L}$ as the identity matrix. In the SVD framework, this can be viewed as Tikhonov filtering, in which we weight the i^{th} singular value as $\frac{\sigma_i^2+\lambda^2}{\sigma_i^2}\sigma_i = \frac{\sigma_i^2+\lambda^2}{\sigma_i}$ [25, p. 90]. When the pseudoinverse is taken, the inverse value is then $\frac{\sigma_i}{\sigma_i^2+\lambda^2}$, which goes to zero as σ_i goes to zero. This prevents small singular values from amplifying noise in the solution, and can be particularly effective for ill-posed problems. λ is typically chosen to be small to prevent larger singular values from shifting far enough to smooth the solution excessively and therefore destroy the details desired in reconstructions. However, the specific choice of λ that produces the qualitatively "best" results for a problem depends on $\boldsymbol{K}$. Parameter-choosing methods such as L-curve [26], cross-validation [20], and discrepancy principle [15, 35] can be used to choose a regularization parameter.

We have discussed two filtering methods for the singular value decomposition: Tikhonov and truncation, which may be used in combination. Here we briefly mention some alternative regularization methods to provide references to the interested reader. Broad overviews of filtering methods can be found in [27, pp. 99-131], and a survey of other regularization methods is available in [43, 51].

Alternative methods include iterative regularization methods [27, pp. 135-172], which in the imaging framework are edge-agnostic. Edge-preserving methods may have better performance on imaging problems because edges contain important information images. One popular edge-preserving regularization method is total variation [41]. For a detailed look into these methods and generalizations of Tikhonov regularization, see [51]. Tikhonov and truncated singular value decompositions are not edge-preserving. Despite this, Tikhonov regularization is sufficient for our experiments in Chapters 3-7. We use Tikhonov regularization to overcome limitations of the algorithms presented herein when we choose not to truncate our SVDs (as is the

default for the method in Chapter 3) or when truncation does not provide sufficient regularization.

As the matrix $\boldsymbol{K}$ gets extremely large, a direct solution via factorization methods such as the SVD becomes infeasible. Instead, iterative methods, such as Krylov subspace methods, are used. For these methods often the full matrix $\boldsymbol{K}$ is not needed, but instead functions for computing matrix-vector and potentially matrix-transpose-vector products are used. These methods can attain accelerated convergence through the use of preconditioners which approximate the inverse of the matrix. Good preconditioners are cheap to store and apply, and are good approximations of the original matrix [21]. When stored cheaply, truncated singular value decomposition (TSVD) approximations fit these criteria.

2.4 Kronecker Products

We now arrive at the concept which, together with the notion of a singular value decomposition, provides the foundation for our proposed algorithms. That idea is the Kronecker product operator and corresponding matrix decomposition [40, 49, 50].

The Kronecker product $\boldsymbol{C}$ of matrices $\boldsymbol{A} \in \mathbb{R}^{j\times k}$ and $\boldsymbol{B} \in \mathbb{R}^{\ell\times m}$ is

$$\boldsymbol{C} = \boldsymbol{A} \otimes \boldsymbol{B} = \begin{bmatrix} a_{1,1}\boldsymbol{B} & \dots & a_{1,k}\boldsymbol{B} \\ \vdots & & \vdots \\ a_{j,1}\boldsymbol{B} & \dots & a_{j,k}\boldsymbol{B} \end{bmatrix} \tag{2.3}$$

which has size $j\ell \times km$ [49, p. 85]. Each block in the product is the matrix $\boldsymbol{B}$ scaled by an entry of $\boldsymbol{A}$.

This convenient structure (2.3) arises in many applications either directly as (2.3) or using the form discussed in Section 2.5. Our motivating example is image deconvolution, but Kronecker products appear in quantum computing [49] and certain

partial differential equations, among other applications [23]. In Section 2.7 we discuss how Kronecker products are a specific type of tensor representation, and how we can interpret them in the image deconvolution context.

The Kronecker product representation is useful and desirable in applications because it has properties that enable cheap computation. These include ([49, pp. 85-87]):

1. $(\boldsymbol{A} \otimes \boldsymbol{B})^T = \boldsymbol{A}^T \otimes \boldsymbol{B}^T$,
2. $(\boldsymbol{A} \otimes \boldsymbol{B})(\boldsymbol{C} \otimes \boldsymbol{D}) = \boldsymbol{AC} \otimes \boldsymbol{BD}$ (note that standard multiplication has higher precedence than Kronecker products do), and
3. $(\boldsymbol{A} \otimes \boldsymbol{B})\boldsymbol{x} = \text{vec}(\boldsymbol{BXA}^T)$ where $\text{vec}(\boldsymbol{X}) = \boldsymbol{x}$ using column-major ordering.

Kronecker products allow for the representation of a dimensionally large matrix as the result of operation on smaller matrices. In the above definition (2.3), $\boldsymbol{C}$ has $O(jk\ell m)$ elements, but can be stored using $O(jk + \ell m)$ elements, which can be a tremendous savings. In addition, operations like multiplication can be performed without explicitly forming the large matrix. The multiplication property 3 above enables $O((\ell + j)mk)$ multiplication with a vector instead of the standard $O(jk\ell m)$ multiplication cost.

As a result, it is nearly always advantageous when dealing with Kronecker products to store the relevant matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ rather than storing the full matrix $\boldsymbol{C}$. To get the best benefit of compression, ideally $j \approx \ell$ and $k \approx m$ so that $\boldsymbol{A}$ and $\boldsymbol{B}$ are roughly the same size.

2.5 Kronecker Product Summation Decomposition

It is rare for an arbitrarily-chosen matrix $\boldsymbol{K} \in \mathbb{R}^{N \times N}$ to be expressible as a Kronecker product in a compressive manner. Of course, we could write $\boldsymbol{K} = [1]_{1\times 1} \otimes \boldsymbol{K}$, but

this would provide no savings.

However, thanks to a result by Van Loan and Pitsianis [50], we can express arbitrary matrices as a sum of Kronecker products. Given any matrix $\boldsymbol{K}$ of size $n^2 \times n^2$, we can express the matrix as a summation of R Kronecker products of matrices of size $n \times n$:

$$\boldsymbol{K} = \sum_{i=1}^{R} \boldsymbol{A}_i \otimes \boldsymbol{B}_i \text{ where } \boldsymbol{A}_i, \boldsymbol{B}_i \text{ are size } n \times n. \tag{2.4}$$

The terms $\boldsymbol{A}_i$ and $\boldsymbol{B}_i$ are computed by taking the SVD of a rearrangement of $\boldsymbol{K}$. The steps for this computation are:

1. Partition $\boldsymbol{K}$ into blocks of size $n \times n$. Number the blocks consecutively using column-major ordering, calling the i^{th} block $\boldsymbol{K}^{(i)}$.
2. Let $\boldsymbol{k}^{(i)} = \text{vec}(\boldsymbol{K}^{(i)})$.
3. Let $\tilde{\boldsymbol{K}} = \begin{bmatrix} \boldsymbol{k}^{(1)T} \\ \boldsymbol{k}^{(2)T} \\ \vdots \end{bmatrix}$
4. Compute the SVD $\tilde{\boldsymbol{K}} = \boldsymbol{U}^{(*)}\boldsymbol{\Sigma}^{(*)}\boldsymbol{V}^{(*)T}$ with $\boldsymbol{U}^{(*)} = \begin{bmatrix} \boldsymbol{u}_1^{(*)} & \boldsymbol{u}_2^{(*)} & \dots \end{bmatrix}$, $\boldsymbol{V}^{(*)} = \begin{bmatrix} \boldsymbol{v}_1^{(*)} & \boldsymbol{v}_2^{(*)} & \dots \end{bmatrix}$, and the i^{th} diagonal entry of $\boldsymbol{\Sigma}^{(*)}$ as $\sigma_i^{(*)}$.
5. Then $\text{vec}(\boldsymbol{A}_i) = \sqrt{\sigma_i^{(*)}}\boldsymbol{u}_i^{(*)}$ and $\text{vec}(\boldsymbol{B}_i) = \sqrt{\sigma_i^{(*)}}\boldsymbol{v}_i^{(*)}$

For the blur operator matrix $\boldsymbol{K}$ we use as an example throughout this work, this decomposition can be intuitively interpreted as a splitting of the blur into a sum of separable blurs. The $\boldsymbol{A}_i$ terms apply blur horizontally and the $\boldsymbol{B}_i$ terms apply blur vertically along an image. See Section 2.7 for details.

There are a few points of note here. First, because $\boldsymbol{A}_i$ and $\boldsymbol{B}_i$ stem from the singular values and vectors in an SVD of $\tilde{\boldsymbol{K}}$, $\boldsymbol{A}_i \otimes \boldsymbol{B}_i$ is the i^{th} most significant term in the Kronecker summation (2.4). That is, for a fixed number of terms r in the

summation, $\min_{\boldsymbol{E}_i,\boldsymbol{F}_i} \|\boldsymbol{K}-\sum_{i=1}^{r} \boldsymbol{E}_i\otimes\boldsymbol{F}_i\|_F$ is minimized by exactly the $\boldsymbol{A}_i$ and $\boldsymbol{B}_i$ constructed by the above procedure [49]. Using this fact, we can use standard TSVD methods like Golub-Kahan bidiagonalization (see Section 7.2 and the references therein) to compute a truncated approximate summation and reduce computational cost:

$$\boldsymbol{K} \approx \sum_{i=1}^{r} \boldsymbol{A}_i \otimes \boldsymbol{B}_i; \tag{2.5}$$

this is explored in more detail in Chapters 3-6.

There are seemingly alarming aspects to this decomposition method. If $\tilde{\boldsymbol{K}}$ does not have exploitable structure, taking the SVD of $\tilde{\boldsymbol{K}}$ requires $O(n^6) = O(N^3)$ operations. This is no cheaper than computing an SVD of $\boldsymbol{K}$, which is the goal we are trying to reach. This is the first issue we must address. The second is that this is not a magical form of compression: R, which we call the Kronecker rank, is limited by the number of nonzero singular values of the matrix $\tilde{\boldsymbol{K}}$. Therefore R can be as large as n^2, which means have $O(n^4) = O(N^2)$ elements stored, the same as the original matrix $\boldsymbol{K}$.

Both of these problems are remedied if $\tilde{\boldsymbol{K}}$ is of a known, low rank. The rank of $\tilde{\boldsymbol{K}}$ is R, and if $R \ll n^2$, then computing the terms $\boldsymbol{A}_i$ and $\boldsymbol{B}_i$ takes $O(n^4 R)$ time. Further, storage costs drop to $O(n^2 R)$, which is much more manageable than the $O(n^4)$ for full-rank $\tilde{\boldsymbol{K}}$.

Image deconvolution is an ideal candidate for Kronecker summation decomposition because the blur operator $\boldsymbol{K}$ has a known, low Kronecker rank. For image deblurring problems, finding the Kronecker product summation decomposition of the blur operator (which is taking the SVD of $\tilde{\boldsymbol{K}}$) amounts to taking the singular value decomposition of the point spread function (PSF) that generates the operator [36]. This reduces the cost of computing (2.4) from $O(N^3)$ to $O(N^{\frac{3}{2}})$. Further, the Kronecker rank R of the blur operator is capped at the size in one dimension of the PSF,

which is typically the size of the blurred image in one dimension, n.

In the preceding discussion, the limitation that $\boldsymbol{K}$ be a square matrix of size $n^2 \times n^2$ slipped in. Blur operators are naturally square, but the restriction that each dimension is of square size is equivalent to restricting the blurred images to square images. However, the analysis holds for images of nonsquare size $n \times m$; in this case, the blocks in $\boldsymbol{K}$ during the partitioning step are taken to be size $m \times n$. Nonetheless, herein we will refer to our image matrices as size $n \times n$ and corresponding vectors as size $n^2 \times 1$, and the blur operator as size $n^2 \times n^2$. For convenience, we call $n^2 = N$.

2.6 SVDs of Kronecker Products

Our goal is to cheaply approximate the TSVD of a matrix. The Kronecker product decomposition (2.4) and approximation (2.5) help us achieve this goal.

For now, we restrict ourselves back to a single Kronecker product, rather than a summation, to begin work towards a TSVD approximation based on the summation decomposition; Chapter 3 begins discussion of an approach based on the summation decomposition.

We can cheaply take the SVD of matrices with a Kronecker product structure. Given matrices $\boldsymbol{A}$ with singular value decomposition $\boldsymbol{U}_A\boldsymbol{\Sigma}_A\boldsymbol{V}_A^T$ and $\boldsymbol{B}$ with singular value decomposition $\boldsymbol{U}_B\boldsymbol{\Sigma}_B\boldsymbol{V}_B^T$,

$$\begin{aligned}\boldsymbol{A} \otimes \boldsymbol{B} &= (\boldsymbol{U}_A\boldsymbol{\Sigma}_A\boldsymbol{V}_A^T) \otimes (\boldsymbol{U}_B\boldsymbol{\Sigma}_B\boldsymbol{V}_B^T) \\ &= (\boldsymbol{U}_A \otimes \boldsymbol{U}_B)(\boldsymbol{\Sigma}_A \otimes \boldsymbol{\Sigma}_B)(\boldsymbol{V}_A^T \otimes \boldsymbol{V}_B^T) \\ &= (\boldsymbol{U}_A \otimes \boldsymbol{U}_B)(\boldsymbol{\Sigma}_A \otimes \boldsymbol{\Sigma}_B)(\boldsymbol{V}_A \otimes \boldsymbol{V}_B)^T.\end{aligned}$$

We call these matrices $\boldsymbol{U}_A \otimes \boldsymbol{U}_B = \boldsymbol{U}$, $\boldsymbol{\Sigma}_A \otimes \boldsymbol{\Sigma}_B = \boldsymbol{\Sigma}$, and $\boldsymbol{V}_A \otimes \boldsymbol{V}_B = \boldsymbol{V}$. Then the singular value decomposition of $\boldsymbol{A} \otimes \boldsymbol{B} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$.

The benefits in this derivation are masked by its simplicity. Recall that computing the SVD of a matrix is expensive, taking $O(N^3)$ operations for a matrix of size $N \times N$. By splitting the matrix into the Kronecker product of two $n \times n$ matrices and computing their SVDs independently, we have reduced the time complexity to $O(N^{\frac{3}{2}})$. For problems where N is huge, on the order of a million or larger, such as arises for the image deconvolution problem on a 1024×1024 image, this is a massive saving.

Beyond the time complexity benefits, we also have improved the memory usage. As before, we do not form the results of the Kronecker products explicitly; instead, the Kronecker product components are stored and used for computations. The matrices $\boldsymbol{U}$ and $\boldsymbol{V}$ are normally size $N \times N$, but are now stored as matrices of size $n \times n$. This takes the memory complexity from $O(N^2) = O(n^4)$ down to $O(n^2)$. Matrices which are impossible to store in memory normally can be stored using a Kronecker product format.

Storing $\boldsymbol{\Sigma}$ using Kronecker products provides an additional, small savings. Because $\boldsymbol{\Sigma}_A$ and $\boldsymbol{\Sigma}_B$ are diagonal matrices, instead of storing $O(n^2)$ zeros, we can store the nonzero diagonal entries using vectors and not matrices. This reduces the memory for storing $\boldsymbol{\Sigma}$ without using Kronecker structure from $O(n^2)$ to $O(n)$ when we do exploit the structure.

Once again, the restriction to a single term Kronecker product is not realistic for many applications. Chapter 3 begins to explore how we can expand the ease of computation afforded here into a summation of Kronecker products.

2.7 A Note on Tensors

Readers familiar with tensors may recognize the Kronecker product as a tensor operation and the Kronecker summation decomposition as a type of tensor decomposition.

In this work, we focus on a linear operator (matrix-vector oriented) perspective of the Kronecker product rather than a tensor perspective. Yet, a great deal of work has been done on tensor SVDs. Here we detail some of the relevant work done from a tensor perspective, and explain how our work relates to the tensor-based approaches. Much of the discussion here stems from a report by Kilmer and Martin on a 2004 tensor workshop [31]. Their report uses notation and language that is accessible for those who are unfamiliar with tensor notation but are comfortable with linear algebraic formulations, and we suggest it as a starting point for those interested in learning more about tensor representations. Additionally, the survey by Grasedyck on low-rank tensor approximations [23] provides a more in-depth discussion of tensor representations.

The SVD of a matrix is a form of tensor decomposition in which we express a matrix, which is a rank two tensor, as a sum of products of rank one tensors. The singular value matrix $\boldsymbol{\Sigma}$ is called a core tensor which determines how the singular vectors are combined to form the full matrix $\boldsymbol{K}$ [12, 31]. We will use $\times$ to denote a tensor product so as to differentiate it from our matrix-oriented Kronecker product; in literature, $\circ$ is often used as well [31]. Using the same notation for the singular value decomposition as before with $\boldsymbol{K} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}$, we can write the tensor form of an SVD of a $N \times N$ matrix $\boldsymbol{K}$ as

$$\begin{aligned}
\boldsymbol{\Sigma} \times \boldsymbol{U} \times \boldsymbol{V}^T &= \sum_{i=1}^{N}\sum_{j=1}^{N} \sigma_{ij}(\boldsymbol{u}_i \times \boldsymbol{v}_j) \\
&= \sum_{i=1}^{N}\sum_{j=1}^{N} \sigma_{ij}\boldsymbol{u}_i\boldsymbol{v}_j^T \\
&\quad \text{which, because } \boldsymbol{\Sigma} \text{ is diagonal,} \\
&= \sum_{i=1}^{N} \sigma_{ii}\boldsymbol{u}_i\boldsymbol{v}_i^T \\
&= \sum_{i=1}^{N} \sigma_{ii}(\boldsymbol{u}_i \times \boldsymbol{v}_i).
\end{aligned}$$

Here, $\boldsymbol{\Sigma}$ is a core tensor: it dictates how the rank one basis tensors $\boldsymbol{u}_i$ and $\boldsymbol{v}_i$ are combined to form the matrix $\boldsymbol{K}$ [31].

Tensor decompositions, including the Tucker decomposition and higher order SVD, often seek to represent a rank m tensor $\mathcal{K}$ in a similar form as above, but where tensor products of m rank 1 tensors are used in the decomposition. Let N_i denote the size of $\mathcal{K}$ along the i^{th} index, then the decompositions are of the form

$$\begin{aligned}\mathcal{K} &= \sum_{i_1}^{N_1}\sum_{i_1}^{N_1}\dots\sum_{i_m}^{N_m}\sigma_{i_1,i_2,\dots,i_m}u_{i_1}^{(1)}\times u_{i_2}^{(2)}\times\dots\times u_{i_m}^{(m)}\\ &= \boldsymbol{\Sigma}\times\boldsymbol{U}^{(1)}\times\boldsymbol{U}^{(2)}\times\dots\times\boldsymbol{U}^{(m)}.\end{aligned}$$

Here, $\boldsymbol{\Sigma}$ is the core tensor with the same number of indices as $\mathcal{K}$. The core dictates how the columns of the matrices $\boldsymbol{U}^{(i)}$ are combined to form the tensor $\mathcal{K}$. For general tensor decompositions, the matrices $\boldsymbol{U}^{(i)}$ are not restricted to be unitary [13,31]. But for generalizations of the singular value decomposition, the matrices $\boldsymbol{U}^{(i)}$ are chosen to be unitary: $\boldsymbol{U}^{(i)*}\boldsymbol{U}^{(i)} = \mathbb{1}$ where * denotes conjugate transpose and $\mathbb{1}$ is the identity matrix. For $m = 2$, $\boldsymbol{U}^{(1)}$ is analogous to the left singular vector matrix $\boldsymbol{U}$, $\boldsymbol{U}^{(2)}$ is analogous to transpose of the right singular vector matrix $\boldsymbol{V}$, and $\boldsymbol{\Sigma}$ is the singular value matrix $\boldsymbol{\Sigma}$ [9, 12, 31]. However, it is not guaranteed that $\boldsymbol{\Sigma}$ is pseudo-diagonal: $\boldsymbol{\Sigma}$ may (and for many matrices is guaranteed to) have nonzero elements along indices other than those for which $i_1 = i_2 = \dots = i_m$ [9, 12].

How do tensor representations relate to our blur operator $\boldsymbol{K}$? Recall that we have the representation (2.4),

$$\boldsymbol{K} = \sum_{i=1}^{R}\boldsymbol{A}_i\otimes\boldsymbol{B}_i.$$

We have taken our image and flattened it into a vector (rank 1 tensor) representation from its original two-dimensional (rank 2) format. Similarly, we are representing the blur operator as a matrix, but we could choose to represent it with three or four

indices. In the three index case, each row of the matrix $\boldsymbol{K}$ is reshaped into a matrix, and in the four index case we separate the rank three tensor along the columns (or analogously rows) of the image they act on. This is subtly evident in the summation form (2.4).

Recall that one of the properties of Kronecker products is that when multiplied by a vector $\boldsymbol{x} = \text{vec}(\boldsymbol{X})$,

$$(\boldsymbol{A} \otimes \boldsymbol{B})\boldsymbol{x} = \boldsymbol{B}\boldsymbol{X}\boldsymbol{A}^T = \boldsymbol{B}(\boldsymbol{A}\boldsymbol{X}^T)^T.$$

Using the image processing example, the matrix $\boldsymbol{X}$ is our original image. Each entry of the matrix $(\boldsymbol{A}\boldsymbol{X}^T)^T$ is a weighted sum of entries along a row of $\boldsymbol{X}$, where the weights stem from $\boldsymbol{A}$; this follows directly from the definition of matrix-matrix multiplication. Similarly, the product $\boldsymbol{B}\boldsymbol{X}$ has entries which are weighted sums of entries along a column of $\boldsymbol{X}$, with the weights originating from $\boldsymbol{B}$. Matrix multiplication is associative, so $(\boldsymbol{B}\boldsymbol{X})\boldsymbol{A}^T = \boldsymbol{B}(\boldsymbol{X}\boldsymbol{A}^T)$. And recall that $\boldsymbol{K}$ is a blur operator, so its decomposition is a representation of how an image gets blurred. With these facts together, we can note that the summation decomposition form (2.4) decomposes the blur operator $\boldsymbol{K}$ into a summation of separable blur operators acting in the horizontal and vertical directions along the image. The matrices $\boldsymbol{A}_i$ act along rows, correspond to horizontal blur, and the matrices $\boldsymbol{B}_i$ likewise correspond to vertical blur. The separability refers to the fact that we may choose to apply the horizontal and vertical blur in any order (this is often called a separable filter, see eg. [25]).

To see this work, we can take a Kronecker summation decomposition of a blur operator, and visualize what happens when all of the $\boldsymbol{A}_i$ matrices or all of the $\boldsymbol{B}_i$ matrices are replaced with the identity operator (corresponding to no blur). We show this for a PSF that is used, albeit scaled for a coarser problem, in Chapters 3-6. The PSF is shown below in Figure 2.3. We also show the clear and blurred image (using

zero boundary conditions) in Figure 2.4. The PSF, clear image, and blurred image are size 256×256.

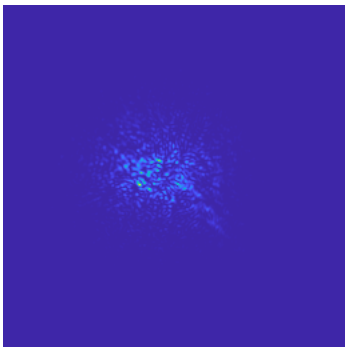


Figure 2.3: The PSF used to blur our image.

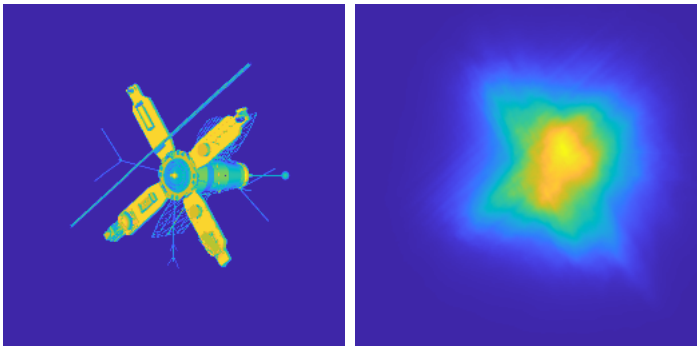


Figure 2.4: Clear and blurred satellite. The true satellite image is shown on the left, and the blurred image is shown on the right.

We represent our blur operator in summation form (2.4): $\boldsymbol{K} = \sum_{i=1}^{R} \boldsymbol{A}_i \otimes \boldsymbol{B}_i$. When we change this to $\sum_{i=1}^{R} \mathbb{1}_n \otimes \boldsymbol{B}_i$ by setting $\boldsymbol{A}_i = \mathbb{1}_n$ for all i, we are left with vertical blur as expected, shown in Figure 2.5.

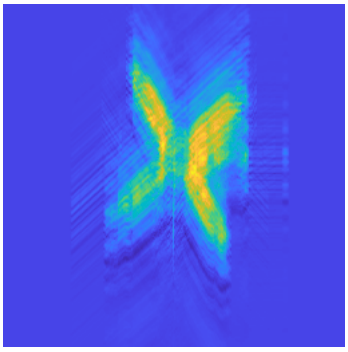


Figure 2.5: Vertical Separable Blur. The vertically blurred image resulting from setting $\boldsymbol{A}_i = \mathbb{1}$ in the Kronecker summation representation of the blur operator.

Similarly, if we let our blur operator have the form $\sum_{i=1}^{R} \boldsymbol{A}_i \otimes \mathbb{1}_n$, where we have set $\boldsymbol{B}_i = \mathbb{1}$ from the original decomposition, we get an image blurred along the horizontal direction as shown in Figure 2.6.

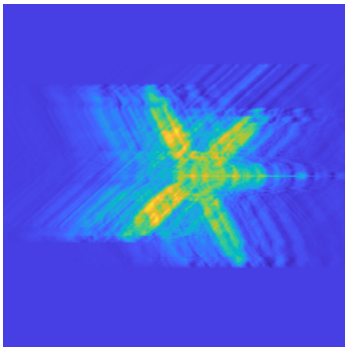


Figure 2.6: Horizontal Separable Blur. The horizontally blurred image resulting from setting $\boldsymbol{B}_i = \mathbb{1}$ in the Kronecker summation representation of the blur operator.

One benefit of this semantic understanding of the decomposition (2.4) is that it gives us some intuition into the relationship between our blur operator and the PSF. If the PSF is well-approximated by a separable blur operator, such as a separable Gaussian [25], then we expect that $\boldsymbol{K}_1 \approx \boldsymbol{K}$. We will see that this is a useful condition for the method described in Chapter 3. When the blur operator is not roughly separable, we would instead use the methods described in Chapters 4-5. We detail the performance of each method further in those chapters.

Now that we understand that our Kronecker decomposition is akin to a factorization into separable operators acting on our grid of data, we revisit the connection of our SVD approximation methods to broader tensor decompositions. Standard tensor decompositions seek an exact representation of the tensor, including the Tucker decomposition [47], HOSVD [12], and hierarchical decompositions [22]. The approaches in this paper take a slightly different perspective. We start with the decomposition (2.4), which is ordered in the same way that an SVD is chosen to be ordered. Using the fact that the term $\boldsymbol{A}_1 \otimes \boldsymbol{B}_1$ is the best separable representation of the blur $\boldsymbol{K}$, we use its singular vectors as the basis for our factorization. Specifically, we factorize $\boldsymbol{A}_1 = \boldsymbol{U}_A \boldsymbol{\Sigma}_A \boldsymbol{V}_A^T$ and $\boldsymbol{B}_1 = \boldsymbol{U}_B \boldsymbol{\Sigma}_B \boldsymbol{V}_B^T$, and compute an approximation of the form $(\boldsymbol{U}_A \otimes \boldsymbol{U}_B)\boldsymbol{T}(\boldsymbol{V}_A \otimes \boldsymbol{V}_B)^T$. In an exact representation, $\boldsymbol{T} = (\boldsymbol{U}_A \otimes \boldsymbol{U}_B)^T \boldsymbol{K} (\boldsymbol{V}_A \otimes \boldsymbol{V}_B)$. We choose to approximate $\boldsymbol{T}$ to yield a factorization with the properties of a singular value decomposition; namely, $\boldsymbol{T}$ must be diagonal and non-negative or represented as an SVD-like factorization. This is akin to taking an approximate factorization of a core tensor. In Chapter 3, the approximation we choose is to take the diagonal of the matrix $\boldsymbol{T}$. In Chapter 4, we truncate the terms $(\boldsymbol{U}_A \otimes \boldsymbol{U}_B)$ and $(\boldsymbol{V}_A \otimes \boldsymbol{V}_B)$, compressing $\boldsymbol{T}$ and enabling us to take a full SVD factorization of the compressed version. The variant in Chapter 5 is similar to the method in Chapter 4, but with permutation added. As we will see experimentally in Chapters 3-6, these approximations work in a variety of cases, with varying levels of effectiveness depending on the properties of the operator $\boldsymbol{K}$. Further, we focus on the 2D case without generalization to higher dimensions, in which case the decomposition (2.4) takes the form [37]

$$\boldsymbol{K} = \sum_{i=1}^{R} \boldsymbol{A}_i \otimes \boldsymbol{B}_i \otimes \boldsymbol{C}_i.$$

More general tensor methods are designed to work in higher dimensional representations. For the methods we propose, this is a potential area of future work.

Chapter 3

Baseline Diagonal Method

In this chapter we detail the first method [29] that was proposed for using the Kronecker summation decomposition (2.4) and equivalently approximation (2.5) to compute an approximate TSVD. Although this method is simple, it is highly effective in certain cases, and the resulting accuracy is a high bar to beat considering its speed.

3.1 Derivation

The derivation of this method begins with the Kronecker decomposition (2.4):

$$\boldsymbol{K} = \sum_{i=1}^{R} \boldsymbol{A}_i \otimes \boldsymbol{B}_i.$$

The first term in this decomposition, $\boldsymbol{A}_1 \otimes \boldsymbol{B}_1$, is the most significant in that it minimizes $\min_{\boldsymbol{A},\boldsymbol{B}} \|\boldsymbol{K} - \boldsymbol{A} \otimes \boldsymbol{B}\|_F$; see Section 2.5 for details. Call the first term $\boldsymbol{K}_1 = \boldsymbol{A}_1 \otimes \boldsymbol{B}_1$. Because it is the most significant term in the summation, it has the most information about $\boldsymbol{K}$ of any single summation term, so we will treat it specially.

Let the singular value decompositions $\boldsymbol{A}_1 = \boldsymbol{U}_A \boldsymbol{\Sigma}_A \boldsymbol{V}_A^T$ and $\boldsymbol{B}_1 = \boldsymbol{U}_B \boldsymbol{\Sigma}_B \boldsymbol{V}_B^T$.

Then by the properties of Kronecker products,

$$\boldsymbol{K}_1 = (\boldsymbol{U}_A \otimes \boldsymbol{U}_B)(\boldsymbol{\Sigma}_A \otimes \boldsymbol{\Sigma}_B)(\boldsymbol{V}_A \otimes \boldsymbol{V}_B)^T.$$

Define $\boldsymbol{U}_1 = \boldsymbol{U}_A \otimes \boldsymbol{U}_B$, $\boldsymbol{\Sigma}_1 = \boldsymbol{\Sigma}_A \otimes \boldsymbol{\Sigma}_B$, and $\boldsymbol{V}_1 = \boldsymbol{V}_A \otimes \boldsymbol{V}_B$. As with all matrices expressible in Kronecker product form, these large matrices ($\boldsymbol{U}_1$, $\boldsymbol{V}_1$, and $\boldsymbol{\Sigma}_1$) are not stored explicitly. Instead, the component matrices in the Kronecker products are stored. While the fully formed matrices are size $N \times N = n^2 \times n^2$, the matrices in the Kronecker products are size $n \times n$.

With this SVD of $\boldsymbol{K}_1$, we can rewrite $\boldsymbol{K}$ using the orthonormality of $\boldsymbol{U}_1$ and $\boldsymbol{V}_1$:

$$\begin{aligned}
\boldsymbol{K} &= \sum_{i=1}^{R} \boldsymbol{A}_i \otimes \boldsymbol{B}_i \\
&= \boldsymbol{A}_1 \otimes \boldsymbol{B}_1 + \sum_{i=2}^{R} \boldsymbol{A}_i \otimes \boldsymbol{B}_i \\
&= \boldsymbol{U}_1 \boldsymbol{\Sigma}_1 \boldsymbol{V}_1^T + \boldsymbol{U}_1 \boldsymbol{U}_1^T \left(\sum_{i=2}^{R} \boldsymbol{A}_i \otimes \boldsymbol{B}_i \right) \boldsymbol{V}_1 \boldsymbol{V}_1^T \\
&= \boldsymbol{U}_1 \left(\boldsymbol{\Sigma}_1 + \sum_{i=2}^{R} \boldsymbol{U}_1^T (\boldsymbol{A}_i \otimes \boldsymbol{B}_i) \boldsymbol{V}_1 \right) \boldsymbol{V}_1^T \\
&= \boldsymbol{U}_1 \left(\boldsymbol{\Sigma}_1 + \sum_{i=2}^{R} \boldsymbol{U}_A^T \boldsymbol{A}_i \boldsymbol{V}_A \otimes \boldsymbol{U}_B^T \boldsymbol{B}_i \boldsymbol{V}_B \right) \boldsymbol{V}_1^T.
\end{aligned}$$

This is exact, not approximate. The left- and rightmost matrices, $\boldsymbol{U}_1$ and ${\boldsymbol{V}_1}^T$, are orthonormal, which is a step towards the goal of a singular value decomposition form. But the interior matrix, which we call $\boldsymbol{T} = \boldsymbol{\Sigma}_1 + \sum_{i=2}^{R} \boldsymbol{U}_A^T \boldsymbol{A}_i \boldsymbol{V}_A \otimes \boldsymbol{U}_B^T \boldsymbol{B}_i \boldsymbol{V}_B$, is not diagonal nor nonnegative, and we want both of those properties in our singular value matrix. Further, it is size $N \times N$, the same size as the original matrix $\boldsymbol{K}$. We can no more easily take its SVD than we can take the SVD of $\boldsymbol{K}$.

Kamm and Nagy [29] proposed constructing an approximate TSVD by taking the

diagonal entries of $\boldsymbol{T}$ as an approximation of the singular values. That is,

$$\begin{aligned}
\boldsymbol{K} &= \boldsymbol{U}_1 \left(\boldsymbol{\Sigma}_1 + \sum_{i=2}^{R} \boldsymbol{U}_A^T \boldsymbol{A}_i \boldsymbol{V}_A \otimes \boldsymbol{U}_B^T \boldsymbol{B}_i \boldsymbol{V}_B \right) \boldsymbol{V}_1^T \\
\boldsymbol{K} &\approx \boldsymbol{U}_1 \mathrm{diag} \left(\boldsymbol{\Sigma}_1 + \sum_{i=2}^{R} \boldsymbol{U}_A^T \boldsymbol{A}_i \boldsymbol{V}_A \otimes \boldsymbol{U}_B^T \boldsymbol{B}_i \boldsymbol{V}_B \right) \boldsymbol{V}_1^T \\
&= \boldsymbol{U}_1 \left(\boldsymbol{\Sigma}_1 + \sum_{i=2}^{R} \mathrm{diag}(\boldsymbol{U}_A^T \boldsymbol{A}_i \boldsymbol{V}_A) \otimes \mathrm{diag}(\boldsymbol{U}_B^T \boldsymbol{B}_i \boldsymbol{V}_B) \right) \boldsymbol{V}_1^T.
\end{aligned}$$

Calling $\boldsymbol{\Sigma}_{\text{diag}} = \boldsymbol{\Sigma}_1 + \sum_{i=2}^{R} \mathrm{diag}(\boldsymbol{U}_A^T \boldsymbol{A}_i \boldsymbol{V}_A) \otimes \mathrm{diag}(\boldsymbol{U}_B^T \boldsymbol{B}_i \boldsymbol{V}_B)$, the approximated SVD is then

$$\boldsymbol{K} \approx \boldsymbol{U}_1 \boldsymbol{\Sigma}_{\text{diag}} \boldsymbol{V}_1^T, \tag{3.1}$$

which can be truncated to a desired matrix rank. We may also choose to use approximation (2.5) so that $\boldsymbol{\Sigma}_{\text{diag}} = \boldsymbol{\Sigma}_1 + \sum_{i=2}^{r} \mathrm{diag}(\boldsymbol{U}_A^T \boldsymbol{A}_i \boldsymbol{V}_A) \otimes \mathrm{diag}(\boldsymbol{U}_B^T \boldsymbol{B}_i \boldsymbol{V}_B)$.

3.2 Discussion

At first glance, it may seem absurd to throw away so much information by taking $\mathrm{diag}(\boldsymbol{T})$ instead of using a different method that preserves more information. There are cases where this is indeed a large detriment, as we will see experimentally in Section 3.4. However, in some cases it proves to not be a significant problem.

When does this work well? Perhaps obviously, when $\boldsymbol{T}$ has a very strong diagonal component compared to its off-diagonal. This is not a rigorous mathematical statement; diagonal dominance is not required. However, if a large amount of the information in $\boldsymbol{T}$ is contained in the diagonal compared to the off-diagonal, then taking its diagonal loses a potentially acceptable amount of information.

This situation arises when $\boldsymbol{\Sigma}_1$ is significant compared to $\sum_{i=2}^{R} \mathrm{diag}(\boldsymbol{U}_A^T \boldsymbol{A}_i \boldsymbol{V}_A) \otimes \mathrm{diag}(\boldsymbol{U}_B^T \boldsymbol{B}_i \boldsymbol{V}_B)$; in other words, when $\boldsymbol{K}_1$ is an extremely good approximation of $\boldsymbol{K}$.

This is not on the whole unusual. In the image deconvolution problem, for example, this is equivalent to saying that the PSF has a very strongly dominant largest singular value, with a large ratio of its first to second singular value $\frac{\sigma_1}{\sigma_2}$. This happens in practice regularly. There are numerous examples in the toolkit RestoreTools [39], two of which we use for several experiments in Section 3.4 and Chapters 4-6.

On the other hand, if $\boldsymbol{K}_1$ is a poor approximation to $\boldsymbol{K}$, then this approximation performs poorly. In that case, $\sum_{i=2}^{R} \boldsymbol{U}_A^T \boldsymbol{A}_i \boldsymbol{V}_A \otimes \boldsymbol{U}_B^T \boldsymbol{B}_i \boldsymbol{V}_B$ will tend to have significant off-diagonal values, which are thrown away in the computation. Examples of the performance in both cases are demonstrated in Section 3.4.

There is one other issue with this method of approximating a TSVD: the "singular values" in $\boldsymbol{\Sigma}_{\text{diag}}$ are not guaranteed to be nonnegative. Normally, this is not an insurmountable problem. If a factorization is highly accurate and maintains all the other properties of a singular value decomposition but produces some negative singular values, the signs of the errant singular values and corresponding left- or right-singular vectors can be changed simultaneously. The product of the factors remains the same, but now the singular values are all positive. This is not possible when Kronecker products are used. Because of Kronecker structure, we cannot change the sign of one single singular vector at a time. If we negate one singular vector of, for example, either $\boldsymbol{V}_A$ or $\boldsymbol{V}_B$, we negate n singular vectors in $\boldsymbol{V}_1 = \boldsymbol{V}_A \otimes \boldsymbol{V}_B$. The same holds for the left singular vectors. We therefore cannot change the sign of individual singular values. And having negative singular values violates the definition of a singular value decomposition. When the method performs well, this tends to not be a problem because $\boldsymbol{\Sigma}_1 \approx \boldsymbol{\Sigma}_{\text{diag}}$ and $\boldsymbol{\Sigma}_1$ has nonnegative entries. However, when the method performs poorly, this is a more significant limitation.

3.3 Time Complexity

The main strength of the baseline method is its extremely cheap computation cost. The time complexity of the algorithm is $O(n^3 r)$ for well-structured $\boldsymbol{K}$, as derived in this section.

To begin the derivation, we start by explicitly stating the steps in the algorithm. These can vary between implementations, but here we discuss what is currently known to be the most efficient general method for computing the TSVD approximation (3.1).

1. Compute the approximate Kronecker product decomposition $\boldsymbol{K} \approx \sum_{i=1}^{r} \boldsymbol{A}_i \otimes \boldsymbol{B}_i$ (2.5). In the image deconvolution problem, this amounts to taking the SVD of the PSF matrix.

2. Compute the SVD $\boldsymbol{A}_1 \otimes \boldsymbol{B}_1 = \boldsymbol{U}_1 \boldsymbol{\Sigma}_1 \boldsymbol{V}_1^T$ indirectly by computing $\boldsymbol{A}_1 = \boldsymbol{U}_A \boldsymbol{\Sigma}_A \boldsymbol{V}_A^T$ and $\boldsymbol{B}_1 = \boldsymbol{U}_B \boldsymbol{\Sigma}_B \boldsymbol{V}_B^T$.

3. Compute $\boldsymbol{\Sigma}_{\text{diag}} = \boldsymbol{\Sigma}_1 + \sum_{i=2}^{r} \text{diag}(\boldsymbol{U}_A^T \boldsymbol{A}_i \boldsymbol{V}_A) \otimes \text{diag}(\boldsymbol{U}_B^T \boldsymbol{B}_i \boldsymbol{V}_B)$.

4. Return the resulting SVD approximation $\boldsymbol{U}_1 \boldsymbol{\Sigma}_{\text{diag}} \boldsymbol{V}_1^T$.

Theorem 1. *If computing the Kronecker summation decomposition (2.5) takes $O(n^3 r)$ operations and $\boldsymbol{K}$ has size $N \times N$ with $N = n^2$, the method described above for computing (3.1) runs in $O(n^3 r)$ operations. More generally, if computing (2.5) takes $O(\mathcal{T})$ operations, the above method takes $O(\mathcal{T} + n^3 r)$ operations.*

Proof. We demonstrate the running times for each step of the algorithm as listed above are bounded by $O(\mathcal{T} + n^3 r)$, and conclude the whole algorithm is therefore bounded the same time complexity.

1. For the image deconvolution problem, taking the SVD of the PSF, which has size bounded by $n \times n$, takes $O(n^3)$. For more general $\boldsymbol{K}$, we call the time complexity of this step $O(\mathcal{T})$.

2. Computing the full SVDs of $\boldsymbol{A}_1$ and $\boldsymbol{B}_1$, which are size $n \times n$, takes $O(n^3)$ operations.

3. For each i from 2 to r, computing the products $\boldsymbol{U}_A^T \boldsymbol{A}_i \boldsymbol{V}_A$ and $\boldsymbol{U}_B^T \boldsymbol{B}_i \boldsymbol{V}_B$ takes $O(n^3)$ operations. Extracting the diagonal for each matrix into a vector takes $O(n)$ operations per matrix. Computing the Kronecker product of the resulting diagonals takes $O(n^2)$ for each i. Summing the i^{th} diagonal matrix into a running summation takes $O(n^2)$ operations.

   This is a total of $O(n^3 + n^2 + n) = O(n^3)$ per i, for a total of $O(n^3 r)$ operations.

4. The product returned is not explicitly formed; instead the components $\boldsymbol{U}_1$, $\boldsymbol{\Sigma}_{\text{diag}}$, and $\boldsymbol{V}_1^T$ are returned as-is (with $\boldsymbol{U}_1$ and $\boldsymbol{V}_1$ represented as their Kronecker product components). This takes $O(1)$ operations.

The complexity of the whole algorithm is bounded by the sum complexities of each step: $O(\mathcal{T}) + O(n^3) + O(n^3 r) + O(1) = O(\mathcal{T} + n^3 r)$. The overall time complexity is therefore $O(\mathcal{T} + n^3 r)$, which is $O(n^3 r)$ for the image deconvolution problem. □

For the baseline algorithm to be most efficient, $\boldsymbol{K}$ should have structure that enables computing the approximation $\boldsymbol{K} \approx \sum_{i=1}^{r} \boldsymbol{A}_i \otimes \boldsymbol{B}_i$ in at most $O(n^3 r)$ operations.

3.4 Performance

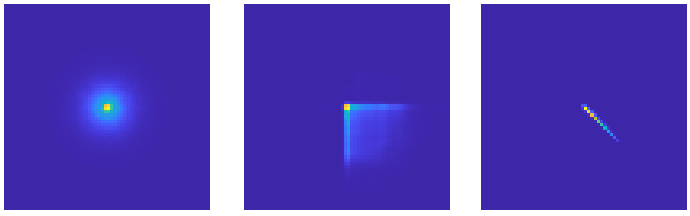
As mentioned in Section 3.2, the baseline method performs very well in situations where $\boldsymbol{K}_1 \approx \boldsymbol{K}$. This can be particularly seen in Subsection 3.4.1, which visually compares direct reconstructions for different $\boldsymbol{K}$ using the baseline method. However, when $\boldsymbol{K}$ is not well approximated by $\boldsymbol{K}_1$, we will see that performance degrades.

Here, we introduce the experimental frameworks that we will use in Chapters 3-6. In each chapter, we run the same experiments using a different approximation algorithm.

3.4.1 Direct Reconstruction

The first experimental framework we explore is a direct image reconstruction using the baseline approximation of a blur operator. We first explain the setup that we will reuse throughout this work, then show the results for the baseline method.

Using the same image deconvolution framework discussed in Section 2.1, we tested each SVD approximation method from Chapters 3-6 on three different blur operators. The first example we refer to as the Satellite Example, the second as the Grain Example, and the third as the Motion Example. The PSFs for each example are shown in Figure 3.1. For each example, we show the true image, the restoration with negative values set to zero (which typically hides noise in the restoration), the restoration with negative values kept, and the blurred image. We also give the peak signal-to-noise ratio (PSNR, see [1, 2]) compared to the true satellite image shown in Figure 3.2; a higher PSNR indicates a better reconstruction. Note that the images are displayed using MATLAB's `imagesc` command, so the resulting visualizations have deep blue as the minimum value and yellow as the largest value in the image, regardless of the actual intensities. In all cases, the image is a 64×64 satellite image available from the RestoreTools package [39], and a full $r = 64$ is used. These are the constants among all examples.



"Satellite" PSF "Grain" PSF "Motion" PSF

Figure 3.1: PSFs for restorations. The three PSFs used for our image deconvolution examples.

We vary a few aspects of the restorations between examples. These changing parameters are summarized in Table 3.1. Firstly, the PSFs are varied as shown in

Figure 3.1. The Satellite Example PSF is from `satellite.mat` in RestoreTools, and the Grain Example PSF is from `Grain.mat` from the same package. The Motion Example PSF is from the upcoming IRTools package by Gazzola, Hansen, and Nagy [18]. The Satellite Example is intended to be an easy example, showing that all methods can produce good results on certain problems. The truncation index k is chosen to be high, at $k = 600$. The Grain Example is intended to differentiate the method described in Chapter 4 from the others, specifically by reducing the truncation index to $k = 400$. And finally, the Motion Example is used to demonstrate a failure case of the baseline method. We reduce the noise to 1% here to show that the method, rather than noise, is the problem. For all the examples, we do not expect visible ringing artifacts from the choice of boundary conditions because the test image, shown in Figure 3.2, does not have significant nonzero data at or near the edges. Nonetheless, we use reflective boundary conditions for the Grain Example as a proof of concept. For the other examples we use zero boundary conditions, which are realistic for the satellite image.

PSF	k	**Gaussian Noise**	**Boundary Conditions**
Satellite	600	2%	Zero
Grain	400	2%	Reflective
Motion	550	1%	Zero

Table 3.1: Parameters for deconvolution examples. We wish to see how each method performs under different conditions, using the parameters here.

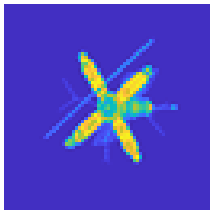
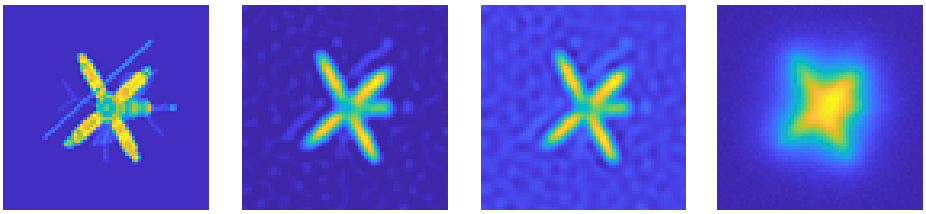


Figure 3.2: Satellite image. This image serves as the ground-truth image $\boldsymbol{x}$ that we seek to restore.

Notably, we do actually use a truncated SVD approximation for all methods, including the baseline method. If a method computes all singular values by default, like the baseline method which computes all N singular values, the singular values are truncated to the largest k values, regardless of ordering.

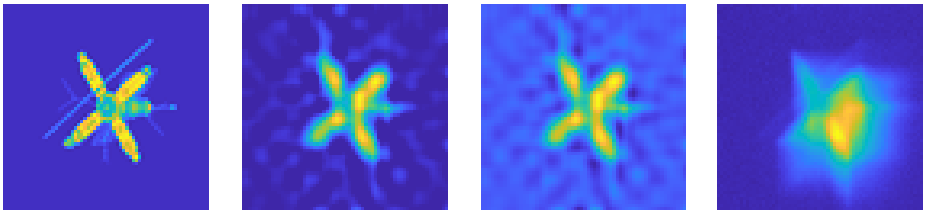
Having established the experimental framework, we now proceed to how the baseline SVD approximation method (3.1) performs on those experiments. The first example is the Satellite Example. The PSF in this example is very smooth and round, with rapidly decaying singular values (notably, $\frac{\sigma_1}{\sigma_2} \approx 6$). We therefore expect the baseline method to approximate the blur operator well and produce a good reconstruction. Indeed, this is the case as is evident in Figure 3.3. The PSNR for this example is 46.9 dB.



True $\boldsymbol{x}$ Restored $\boldsymbol{x} > 0$ Restored $\boldsymbol{x}$ Blurred $\boldsymbol{d}$

Figure 3.3: Baseline Satellite Example Restoration. The baseline method yields a good reconstruction, shown in the middle two images, of the blurred image on the right. The true satellite image is shown on the left.

The second example, called the Grain Example, starts to show degraded performance as is evident in Figure 3.4. For this PSF, $\frac{\sigma_1}{\sigma_2} \approx 23$, which is very high. We expect the baseline method to perform well in general, but recall that we have limited our truncation index to a low $k = 400$. For the baseline method, the restoration is borderline acceptable, but does not have high detail, and is not nearly as good as the Satellite Example. The PSNR for this example is 42.8 dB.



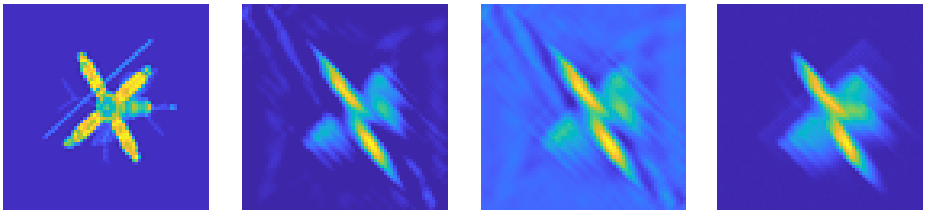
True $\boldsymbol{x}$ Restored $\boldsymbol{x} > 0$ Restored $\boldsymbol{x}$ Blurred $\boldsymbol{d}$

Figure 3.4: Baseline Grain Example Restoration. The baseline method yields a moderately good reconstruction for this example. The blurred image is shown on the right, and the true satellite image is shown on the left.

In Figure 3.4, we start to see notable differences between the image with negative values truncated to zero and the image that keeps negative values. There is substantial noise in this image, as is evident by the third image showing the restoration with negative values. To see a side-by-side comparison with the Satellite Example results, see Figure 3.6.

The final example uses a PSF that originates from simulated motion blur. We refer to this example as the Motion Example. The PSF, shown in Figure 3.1, is from the upcoming IRTools package by Gazzola, Hansen, and Nagy [18]. This PSF has a small band of nonzero elements, dominated by values along the diagonal. The diagonal decays fairly smoothly. The singular values of the PSF therefore decay slowly compared to the previous two examples, with $\frac{\sigma_1}{\sigma_2} \approx 1.2$. Consequentially, $\boldsymbol{K}_1$ is a poor approximation of $\boldsymbol{K}$, so we expect the baseline method to give a poor approximation.

We tested this in the Motion Example. The results are shown in Figure 3.5. The reconstruction is visually not much better than the original blurred image. The PSNR for this example is 39.0 dB, lower than the PSNRs for the Satellite (46.9 dB) and Grain (42.8 dB) examples.

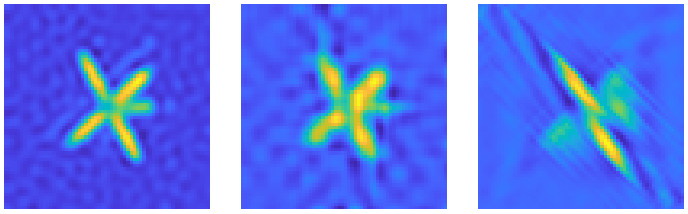


True $\boldsymbol{x}$ Restored $\boldsymbol{x} > 0$ Restored $\boldsymbol{x}$ Blurred $\boldsymbol{d}$

Figure 3.5: Baseline Motion Example Restoration. The baseline method yields a poor reconstruction for the Motion Example. The blurred image is shown on the right, and the true satellite image is shown on the left. The reconstructions in the middle are not much better than the original blurred image.

Here, considerable error corrupts the restored solution. This is evident in both the negative-truncated image and the complete restoration. This type of problem motivates the need for a method that uses more information to approximate $\boldsymbol{K}$.

Figure 3.6 visually summarizes the results for the three experiments in this section.



Satellite Grain Motion

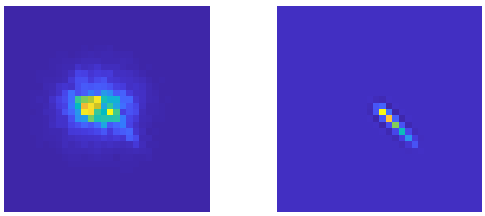
Figure 3.6: Baseline Restoration Summary. This summary image shows the restored images $\boldsymbol{x}$ computed using the baseline method (3.1) for each restoration example.

3.4.2 Approximated Singular Values

One measure of performance for SVD approximations is the relative error in approximate singular values. Although the singular vectors matter in getting a good approximation as well, we defer experimentation with the singular vectors because a proof bounding their error was provided by Chang (not Garvey) in [17]. We check to see how the baseline method, and each subsequent method in the following chapters, approximates the singular values of the original matrix. To be clear, a good singular

value approximation does not mean that the method well-approximates the matrix $\boldsymbol{K}$; it is possible to have relatively accurate singular values but incorrect corresponding singular vectors. For example, singular vectors corresponding to the top 10 singular values in the true SVD of $\boldsymbol{K}$ may be orthogonal to the top 10 singular vectors in the approximate TSVD. Nonetheless, good singular value estimates are necessary in a good approximation, and we can gain insight into the performance of each method by looking at the error in estimated singular values.

Both of the test problems for singular values use blur operators originating from 32×32 images, so that the blur operator itself is of size 1024×1024. This enables a fast full SVD. The first example uses a new PSF not from the reconstruction examples. The PSF represents an atmospheric blur and is available as `AtmosphericBlur50.mat` in the RestoreTools package [39]. The rescaled Motion PSF and new Atmospheric Blur PSF are shown in Figure 3.7. The blur operators are constructed using those PSFs and zero boundary conditions, and we compute the top $k = 100$ singular values for both examples. We will call the first example the Singular Value Atmospheric Blur Example, and the second example the Singular Value Motion Example. For reference, the reconstructions for each method are visually comparable for this problem, and none are excellent.



Atmospheric PSF Motion PSF

Figure 3.7: Singular Value Experiment PSFs. The PSF (left) used in the Singular Value Atmospheric Blur Example, from RestoreTools, and the PSF used for the Singular Value Atmospheric Blur Example (right). Both PSFs are 32×32.

The first $k = 100$ singular values computed by the baseline method (3.1) for the

Singular Value Atmospheric Blur operator are shown in the left of Figure 3.8, with the relative error in the singular values to the right.

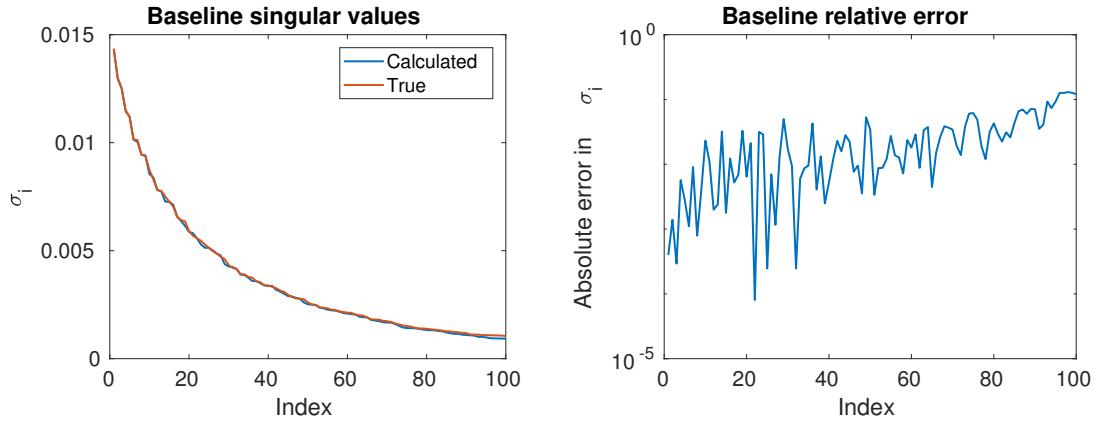


Figure 3.8: Baseline Atmospheric Blur Singular Values. The singular values (left) and relative error (right) are shown for the baseline method approximation of the Singular Value Atmospheric Blur Example. The singular values have relative error which oscillates, but has a relatively stable average despite the oscillation.

As expected, the approximate singular values are moderately good for this example. The error is in the range of 10^{-3} through 10^{-2} for most of the values, and gets as high as the order of 10^{-1} for the smallest values.

For the motion blur PSF, the performance is considerably different. Recall that this PSF has slowly decaying singular values compared to the other examples, so the baseline method is expected to poorly approximate the blur operator $\boldsymbol{K}$. Indeed, this is the case, and the estimated singular values have high error as shown by the singular values and relative error in Figure 3.9. The error in this example quickly reaches and exceeds 10^{-1}. Even for the most accurate singular values, the error starts above 10^{-3}, which is higher than in the previous example. The baseline method therefore both gives a poor reconstruction for the motion blur example, as discussed in Subsection 3.4.1, and poorly approximates the singular values of the blur operator.

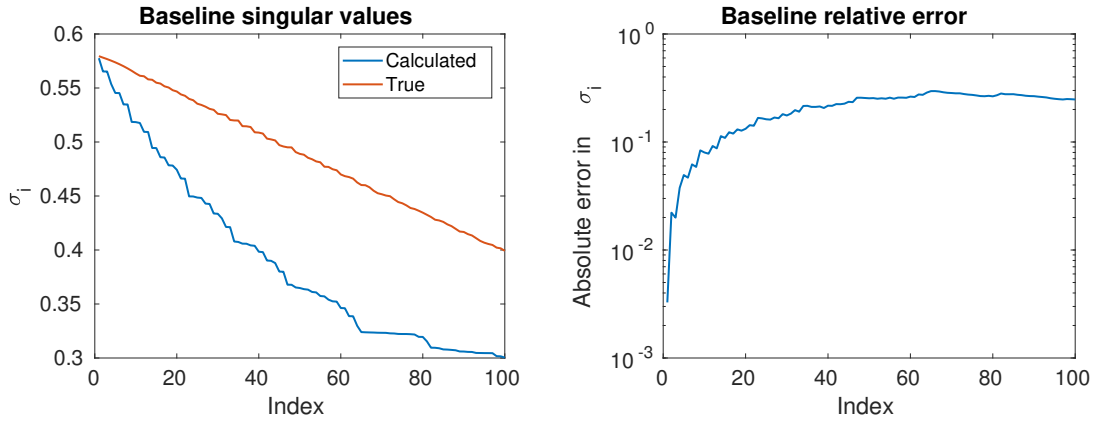


Figure 3.9: Baseline Motion Singular Values. The singular values (left) and relative error (right) are shown for the baseline method approximation of the Singular Value Motion Example. The singular values have much higher error for this example, starting below 10^{-2} relative error and increasing above 10^{-1} relative error.

For problems where $\boldsymbol{K} \approx \boldsymbol{K}_1$, we expect and indeed see that the baseline method approximates the singular values of $\boldsymbol{K}$ well. But when $\boldsymbol{K}_1$ is a poor approximation of $\boldsymbol{K}$, as in the Singular Value Motion Example, the baseline may give highly inaccurate estimates of the singular values.

3.4.3 Preconditioning

Cheap approximations of the system matrix are useful for iterative methods that can be accelerated by preconditioners (for an introduction to preconditioning, we recommend the books [42, pp. 275-368], [21, pp. 650-666], and [6, pp. 688-709], as well as the survey [5]). We tested each method as a preconditoner for a preconditioned conjugate gradient least squares (PCGLS) method applied to a 256×256 image deconvolution problem with the Satellite Example PSF. For this problem, since each method gave comparable results, we were concerned with how quick the solution was. In particular, we separated out the time to construct the operator from the time required to converge iterations. We present the results for each method in their specific chapter, and summarize the results in Appendix C.

For each method, we ran 25 timing trials. The blur operator was constructed using zero boundary conditions. We used $k = 1520$ and restricted $r = 20$ from the maximum of $R = 64$. The problem used 1% Gaussian noise in the right-hand side. Calling the preconditoner $\boldsymbol{M}$ and the residual $\boldsymbol{r}$, iterations stopped when the norm of $\boldsymbol{M}^{-1}\boldsymbol{K}\boldsymbol{r}$ was less than 10^{-10}. To ensure that the solutions were smooth, we added Tikhonov regularization with regularization parameter $\lambda = 0.02$.

The baseline method benefited from computing a full SVD approximation for this problem. As evident in the following chapters, methods that compute a TSVD approximation require a high enough truncation index so that the significant singular vectors are captured in computation.

The baseline method converged in 16 iterations compared to 345 iterations without a preconditioner. The average total time to complete all computation was 0.481 seconds, compared to 6.04 seconds for no preconditoner. It took an average of 0.101 seconds to construct the approximate SVD, and completing the iterations took only 0.380 seconds on average once the preconditioner was constructed. These results are summarized in table below.

	No preconditioner	Baseline
Iterations	345	16
Setup time (sec)	0.0	0.101
Calculate time (sec)	6.04	0.380
Total time (sec)	6.04	0.461

Table 3.2: Baseline Preconditioner Timings. A summary of the timing results for the PCLGS problem, with the baseline method used as a preconditioner, is shown. The baseline method saves time and takes fewer iterations compared to an unpreconditioned system.

The baseline method significantly increased the rate of convergence, both by decreasing the number of iterations required and decreasing the total wall-clock time required to achieve convergence.

3.5 Summary

The baseline method proposed by Kamm and Nagy in [29] constructs an SVD which approximates $\boldsymbol{K}$ using a Kronecker product summation decomposition of a matrix. The singular vectors are the singular vectors resulting from the most significant term in the Kronecker summation, and the singular values are the diagonal of a related matrix. The method is fast, with an $O(n^3 r)$ running time on the image deconvolution problem. The baseline method gives good results when $\boldsymbol{K}$ is well-approximated by the most significant term in the Kronecker decomposition, but works poorly if the approximation is poor. Furthermore, the singular values computed may be irrecoverably negative.

In Chapter 4, we begin to explore methods that try to improve upon those limitations. The key goals are to use more information and therefore perform better on a wider variety of matrices $\boldsymbol{K}$, to ensure the singular values are nonnegative, and maintain a feasible running time.

Chapter 4

Kronecker Truncation Method

In a first attempt to improve upon the baseline, this chapter details a new alternative approach for computing a TSVD approximation from a Kronecker summation decomposition (2.4) or approximation (2.5). This method uses early truncation of the singular vector matrices to cheaply enable a potentially more accurate computation.

4.1 Derivation

The baseline method has two points of approximation. The first is truncating the Kronecker summation decomposition (2.4) from an exact R terms to the most significant r terms in (2.5). This has the same mathematical justification as a TSVD, and is sound in many cases with r dependent on the singular value decay of $\tilde{\boldsymbol{K}}$. The second approximation is taking the diagonal of the matrix $\boldsymbol{T}$, which leads to the issue that we wish to remedy: inaccurate approximation for certain $\boldsymbol{K}$ and potentially negative "singular values."

We start our new approximation by using the same reasoning as the baseline method, and from there derive a different form of secondary approximation. We have

$$\boldsymbol{K} = \boldsymbol{U}_1 \left(\boldsymbol{\Sigma}_1 + \sum_{i=2}^{R} \boldsymbol{U}_1^T (\boldsymbol{A}_i \otimes \boldsymbol{B}_i) \boldsymbol{V}_1 \right) \boldsymbol{V}_1^T.$$

Here, this is exact and full. The interior term, $\boldsymbol{T} = \boldsymbol{\Sigma}_1 + \sum_{i=2}^{R} \boldsymbol{U}_1^T (\boldsymbol{A}_i \otimes \boldsymbol{B}_i) \boldsymbol{V}_1$, is of size $N \times N$, which prohibits factorization such as an SVD. The question motivating the truncation method proposed in this chapter is: What if we made $\boldsymbol{T}$ smaller without throwing away too much information?

We can accomplish this by using a truncated SVD on the term $\boldsymbol{K}_1$. Instead of

$$\boldsymbol{K}_1 = (\boldsymbol{U}_A \otimes \boldsymbol{U}_B)(\boldsymbol{\Sigma}_A \otimes \boldsymbol{\Sigma}_B)(\boldsymbol{V}_A^T \otimes \boldsymbol{V}_B^T)$$

which is used without truncation to construct the baseline method (3.1), we use a truncation of each of the terms in the singular value decomposition. Temporarily denoting the truncation of a generic matrix $\boldsymbol{X}_Y$ to p columns as $\boldsymbol{X}_{Y,p}$, we approximate

$$\boldsymbol{K}_1 \approx (\boldsymbol{U}_{A,\ell} \otimes \boldsymbol{U}_{B,m})(\boldsymbol{\Sigma}_{A,\ell} \otimes \boldsymbol{\Sigma}_{B,m})(\boldsymbol{V}_{A,\ell}^T \otimes \boldsymbol{V}_{B,m}^T).$$

That is, we compactly truncate the terms corresponding to the SVD of $\boldsymbol{A}_1$ to ℓ singular values and vectors, and truncate the terms for $\boldsymbol{B}_1$ to m singular values and vectors. Defining $\boldsymbol{U}_{1,k} = \boldsymbol{U}_{A,\ell} \otimes \boldsymbol{U}_{B,m}$ where $k = \ell m$, we can write this as

$$\boldsymbol{K}_1 \approx \boldsymbol{U}_{1,k} \boldsymbol{\Sigma}_{1,k} \boldsymbol{V}_{1,k}^T. \tag{4.1}$$

There is a subtlety worth noting here. Truncation of the individual terms in the Kronecker product is not the same as truncating the final columns of the fully formed matrices. Recall the definition of a Kronecker product, (2.3). Kroncker products have blocks consisting of the second multiplicand scaled by an entry of the first multiplicand. Using $\boldsymbol{U}_1$ as an example and denoting the entry at row i and column j

in $\boldsymbol{U}_A$ as $u_{A,ij}$, taking the Kronecker product

$$\begin{aligned}\boldsymbol{U}_1 &= \boldsymbol{U}_A \otimes \boldsymbol{U}_B \\ &= \begin{bmatrix} u_{A,11}\boldsymbol{U}_B & \dots & u_{A,1n}\boldsymbol{U}_B \\ \vdots & & \vdots \\ u_{A,n1}\boldsymbol{U}_B & \dots & u_{A,nn}\boldsymbol{U}_B \end{bmatrix}.\end{aligned}$$

Here, truncating the columns of $\boldsymbol{U}_A$ removes whole blocks in the resulting matrix. However, truncating columns of $\boldsymbol{U}_B$ removes columns from each block in the resulting matrix. These columns are not all on the right-hand side of the matrix $\boldsymbol{U}_1$; they are in every block. So the truncation here is not the standard truncation of the final columns of the matrices $\boldsymbol{U}_1$ and $\boldsymbol{V}_1$, and similarly not the final diagonal entries in $\boldsymbol{\Sigma}_1$.

This is an important distinction because the singular values in $\boldsymbol{\Sigma}_1$ are not sorted in descending order. Because $\boldsymbol{\Sigma}_A$ and $\boldsymbol{\Sigma}_B$ are sorted, the Kronecker product has blocks which are internally sorted (because $\boldsymbol{\Sigma}_B$ is sorted), but the whole diagonal is not. For example, denoting the i^{th} diagonal entry in $\boldsymbol{\Sigma}_1$ as $\sigma_{1,i}$ and the singular values in $\boldsymbol{\Sigma}_A$ and $\boldsymbol{\Sigma}_B$ similarly, $\sigma_{1,n} = \sigma_{A,1}\sigma_{B,n}$ and $\sigma_{1,n+1} = \sigma_{A,2}\sigma_{B,1}$. $\sigma_{B,n}$ is the smallest singular value in $\boldsymbol{\Sigma}_B$, so it is likely that $\sigma_{i,n} < \sigma_{1,n+1}$. By truncating the matrices $\boldsymbol{U}_A$, $\boldsymbol{U}_B$, $\boldsymbol{V}_A$, and so on, we hope to preserve more significant singular values in the truncated matrix $\boldsymbol{\Sigma}_{1,k}$ and therefore more accurately approximate $\boldsymbol{U}_1$ with $\boldsymbol{U}_{1,k}$.

With that detail solidified, we return to the derivation, with the TSVD approximation of $\boldsymbol{K}_1$ as (4.1). We proceed with the same derivation as the baseline method but substitute our truncated matrices for the full ones, yielding

$$\begin{aligned}\boldsymbol{K} &= \sum_{i=1}^{R} \boldsymbol{A}_i \otimes \boldsymbol{B}_i \\ &= \boldsymbol{A}_1 \otimes \boldsymbol{B}_1 + \sum_{i=2}^{R} \boldsymbol{A}_i \otimes \boldsymbol{B}_i\end{aligned}$$

$$
\begin{aligned}
&\approx \boldsymbol{U}_{1,k}\boldsymbol{\Sigma}_{1,k}\boldsymbol{V}_{1,k}^T + \boldsymbol{U}_{1,k}\boldsymbol{U}_{1,k}^T \left(\sum_{i=2}^{R} \boldsymbol{A}_i \otimes \boldsymbol{B}_i \right) \boldsymbol{V}_{1,k}\boldsymbol{V}_{1,k}^T \\
&= \boldsymbol{U}_{1,k} \left(\boldsymbol{\Sigma}_{1,k} + \sum_{i=2}^{R} \boldsymbol{U}_{1,k}^T (\boldsymbol{A}_i \otimes \boldsymbol{B}_i) \boldsymbol{V}_{1,k} \right) \boldsymbol{V}_{1,k}^T \\
&= \boldsymbol{U}_{1,k} \left(\boldsymbol{\Sigma}_{1,k} + \sum_{i=2}^{R} \boldsymbol{U}_{A,\ell}^T \boldsymbol{A}_i \boldsymbol{V}_{A,\ell} \otimes \boldsymbol{U}_{B,m}^T \boldsymbol{B}_i \boldsymbol{V}_{B,m} \right) \boldsymbol{V}_{1,k}^T.
\end{aligned}
$$

Here, there is a new interior term $\boldsymbol{T}_k = \boldsymbol{\Sigma}_{1,k} + \sum_{i=2}^{R} \boldsymbol{U}_{A,\ell}^T \boldsymbol{A}_i \boldsymbol{V}_{A,\ell} \otimes \boldsymbol{U}_{B,m}^T \boldsymbol{B}_i \boldsymbol{V}_{B,m}$. Because of truncation, this term is now size $k \times k$ instead of the original $N \times N$ for the exact $\boldsymbol{T}$ in (3.1). If we choose $k \ll N$, then taking a full SVD of $\boldsymbol{T}_k$ is not prohibitive. We may also choose to use the Kronecker summation approximation (2.5) so that $\boldsymbol{T}_k = \boldsymbol{\Sigma}_{1,k} + \sum_{i=2}^{r} \boldsymbol{U}_{A,\ell}^T \boldsymbol{A}_i \boldsymbol{V}_{A,\ell} \otimes \boldsymbol{U}_{B,m}^T \boldsymbol{B}_i \boldsymbol{V}_{B,m}$.

Let $\boldsymbol{T}_k = \boldsymbol{U}_t \boldsymbol{\Sigma}_t \boldsymbol{V}_t^T$ be the singular value decomposition of $\boldsymbol{T}_k$. Then we have

$$
\boldsymbol{K} \approx \boldsymbol{U}_{1,k}\boldsymbol{U}_t\boldsymbol{\Sigma}_t\boldsymbol{V}_t^T\boldsymbol{V}_{1,k}^T \tag{4.2}
$$

This is an approximate TSVD of the matrix $\boldsymbol{K}$ with singular values $\boldsymbol{\Sigma}_t$, left singular vector matrix $\boldsymbol{U}_{1,k}\boldsymbol{U}_t$, and right singular value matrix $\boldsymbol{V}_{1,k}\boldsymbol{V}_t$.

4.2 Discussion

This algorithm and resulting approximation (4.2) remedies both the issues that arose with the baseline method, but creates new limitations in the process. The truncation method preserves more information than the baseline method when $\boldsymbol{K}_1$ is a poor approximation of $\boldsymbol{K}$, and it is impossible for the approximated singular values to be negative because they stem from a singular value decomposition. However, we will see that subtle error causes the smallest singular values to significantly underestimate the true singular values.

In Section 4.1, we explored how using Kronecker products affects the ordering of

singular values and vectors. $\mathbf{\Sigma}_{1.k}$ is formed not by truncating $\mathbf{\Sigma}_1$ to the first k entries, but by truncating $\mathbf{\Sigma}_A$ and $\mathbf{\Sigma}_B$. This helps preserve the singular values and vectors corresponding to larger singular values. However, this does not guarantee that the largest k singular values of $\mathbf{\Sigma}_1$ are present in $\mathbf{\Sigma}_{1,k}$; it is nearly always the case that we discard some singular values which are greater than the smallest values we keep. Equivalently, we keep some singular values from $\mathbf{\Sigma}_1$ which are less significant than singular values which are discarded.

If $\boldsymbol{K} = \boldsymbol{K}_1$, then the true singular values of $\boldsymbol{K}$ are the singular values in $\mathbf{\Sigma}_1$. In this case the baseline method exactly computes the true singular values for the whole matrix. A non-compact truncation, in which singular values are zeroed out rather than removing rows and columns in the factorization, can then be used to truncate to the exact top k singular values. However, if we use the newly-proposed truncation method described in Section 4.1, we underestimate the smallest of the top k singular values because we keep true singular values which are not the k most significant values. For different $\boldsymbol{K}$ we may have a larger or smaller fraction of the true top k values depending on the decay of the singular values from $\mathbf{\Sigma}_A$ and $\mathbf{\Sigma}_B$; we can check by comparing the values we keep to all values in $\mathbf{\Sigma}_1$.

This example is pathological because it is unlikely $\boldsymbol{K} = \boldsymbol{K}_1$, but the logic extends to the case when $\boldsymbol{K} \neq \boldsymbol{K}_1$. Particularly in cases where the singular values of $\tilde{\boldsymbol{K}}$ decay rapidly and therefore the Kronecker summation decomposition (2.5) is heavily weighted towards the first term, $\boldsymbol{K}_1$ provides a good estimate of $\boldsymbol{K}$. Then underestimating the singular values of $\boldsymbol{K}_1$ is likely to cause an underestimation of the singular values of $\boldsymbol{K}$.

Even in cases where $\boldsymbol{K}$ is not well-approximated by $\boldsymbol{K}_1$ because $\tilde{\boldsymbol{K}}$ has slowly decaying singular values, $\boldsymbol{K}_1$ is still the best single Kronecker term approximation of $\boldsymbol{K}$. Using a less accurate approximation of $\boldsymbol{K}_1$ by incorrectly truncating to the top k terms should tend to produce a less accurate approximation of $\boldsymbol{K}$ than using

a proper TSVD approximation of $\boldsymbol{K}_1$. As we will see experimentally in Section 4.4, there are cases where this is beneficial rather than detrimental, particularly for image restoration. In general, we aim for the most accurate approximation possible while maintaining low computational cost.

We will also see an example in Section 4.4 demonstrating that a skewed choice of ℓ and m, so that $\ell \not\approx m$, can yield a poor approximation. Recall from Section 2.7 that in the Kronecker summation decomposition (2.4) the matrices $\boldsymbol{A}_i$ and $\boldsymbol{B}_i$ respectively correspond to separable blur in the horizontal and vertical directions of the image. If we poorly approximate $\boldsymbol{A}_1$ by choosing a small truncation index ℓ, we can expect that we will get a less accurate result in the horizontal direction. Similarly, choosing a small m will yield a less accurate result in the vertical direction. If we want our estimation to be equally good in both directions, we should choose $\ell \approx m$. But if we know a priori that the blur acts much more strongly in one direction, such as a mostly-horizontal motion blur (such as shown in [25, p. 26]), we may choose to skew ℓ and m to increase the accuracy in the horizontal direction (i.e. raise ℓ) and decrease accuracy in the vertical direction (i.e. lower m). As shown in Section 4.3, the time complexity depends linearly on $(\ell+m)n^2r$. Holding n and r constant a skewed choice of ℓ and m slightly increases the time complexity, because $(\ell + m)$ is minimized by $\ell = m = \sqrt{k}$ for $\ell, n \in \mathbb{R}, \ell m = k$. However, skewing ℓ and m is not prohibitively expensive due to the overall time complexity derived in Section 4.3.

4.3 Time Complexity

In this section we show that the truncation method takes $O(n^3+(l+m)n^2r+k^2r+k^3)$ time to compute for image deconvolution problems. In practice, this is not better than the baseline, but it is nonetheless fast enough to be feasible for very large matrices.

The steps of the truncation-based Kroncker TSVD approximation algorithm are

as follows, with the first step the same as the baseline method:

1. Compute the approximate Kronecker product decomposition $\boldsymbol{K} \approx \sum_{i=1}^{r} \boldsymbol{A}_i \otimes \boldsymbol{B}_i$.
2. Compute the TSVD $\boldsymbol{A}_1 \otimes \boldsymbol{B}_1 \approx \boldsymbol{U}_{1,k}\boldsymbol{\Sigma}_{1,k}\boldsymbol{V}_{1,k}^T$ indirectly by computing $\boldsymbol{A}_1 \approx \boldsymbol{U}_{A,\ell}\boldsymbol{\Sigma}_{A,\ell}\boldsymbol{V}_{A,\ell}^T$ and $\boldsymbol{B}_1 \approx \boldsymbol{U}_{B,m}\boldsymbol{\Sigma}_{B,m}\boldsymbol{V}_{B,m}^T$.
3. Compute $\boldsymbol{T}_k = \boldsymbol{\Sigma}_{1,k} + \sum_{i=2}^{r} \boldsymbol{U}_{A,\ell}^T\boldsymbol{A}_i\boldsymbol{V}_{A,\ell} \otimes \boldsymbol{U}_{B,m}^T\boldsymbol{B}_i\boldsymbol{V}_{B,m}$.
4. Compute the SVD $\boldsymbol{T}_k = \boldsymbol{U}_t\boldsymbol{\Sigma}_t\boldsymbol{V}_t^T$.
5. Return the resulting SVD approximation $\boldsymbol{U}_{1,k}\boldsymbol{U}_t\boldsymbol{\Sigma}_t\boldsymbol{V}_t^T\boldsymbol{V}_{1,k}^T$.

Theorem 2. *If computing the Kronecker summation decomposition (2.5) takes $O(\mathcal{T})$ operations, the truncation method takes $O(\mathcal{T} + n^3 + (l+m)n^2r + k^2r + k^3)$ operations.*

Proof. The time complexities for each step of the algorithm are:

1. As with the baseline, the time complexity of computing the Kronecker summation decomposition is called $O(\mathcal{T})$.
2. Computing the rank ℓ TSVD of the $n \times n$ matrix $\boldsymbol{A}_1$ takes $O(n^2\ell)$ operations and computing the rank m TSVD of the $n \times n$ matrix $\boldsymbol{B}_1$ takes $O(n^2m)$ operations. This step totals $O((\ell + m)n^2)$ operations.
3. Each multiplication $\boldsymbol{U}_{A,\ell}^T\boldsymbol{A}_i\boldsymbol{V}_{A,\ell}$ is of matrices of size $\ell \times n$, $n \times n$, and $n \times \ell$ respectively. This can be computed in $O(n^2\ell)$ time. Similarly, the multiplications $\boldsymbol{U}_{B,m}^T\boldsymbol{B}_i\boldsymbol{V}_{B,m}$ take $O(n^2m)$ operations. Forming the Kronecker product for each term in the summation and adding it to a running partial sum of $\boldsymbol{T}_k$ takes $O(k^2)$ multiplications and additions ($k = \ell m$).

   There are $r - 1$ such computations. Adding in the diagonal entries of $\boldsymbol{\Sigma}_{1,k}$ does not increase the time complexity. This step totals $O((\ell + m)n^2r + k^2r)$.
4. Taking the full SVD of the $k \times k$ matrix $\boldsymbol{T}_k$ takes $O(k^3)$ operations.

5. As with the baseline method, returning the result takes $O(1)$ operation.

The final time complexity is bounded by the sum of the complexities from each step. The sum of the steps' complexities is $O(\mathcal{T}) + O(n^3) + O((l+m)n^2r + k^2r) + O(k^3) = O(\mathcal{T} + n^3 + (l+m)n^2r + k^2r + k^3)$ operations, which is therefore the time complexity of the whole algorithm. □

As a corollary, for image deconvolution problems which have an $O(n^3)$ bound on $\mathcal{T}$, this algorithm takes $O(n^3 + (l+m)n^2r + k^2r + k^3)$ operations.

At first glance, this time complexity is difficult to compare to the baseline, which has time complexity $O(n^3r)$. In our experiments, we found that r can typically be chosen so small as to be considered a constant without degrading performance heavily, and typically $k > n$ to be effective. In practice, this algorithm is therefore slower than the baseline method, although not prohibitively so.

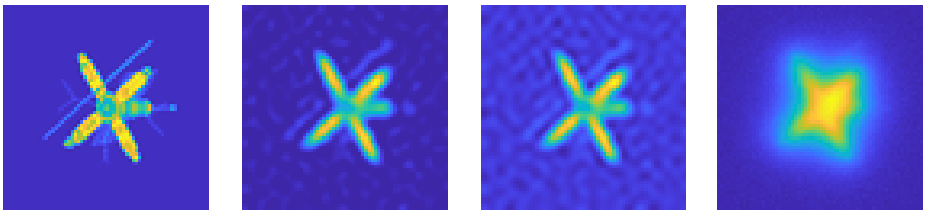
4.4 Performance

The truncation method has some benefits and drawbacks in its performance. As with the baseline method, we look at several examples based on image reconstruction: the visual results, the singular values, and how the approximation works as a preconditioner. We will see that the results are visually excellent for these examples due to the inclusion of smaller singular values. The drawback is that this underestimation due to Kronecker ordering results in error for the singular value approximations. Visual comparisons between the restorations by the truncation method and the the methods from Chapters 3 and 5-6 are available in Figure 1.2 and Appendix A.

4.4.1 Direct Reconstruction

We look at the same same reconstruction examples as we did in Subsection 3.4.1, starting with the Satellite Example. In this example, the baseline method gave a

good approximation, and this continues to be the case with the truncation method. For this method, we split the truncation index $k = 600$ into $\ell = 25$ and $m = 24$. The result of the reconstruction is shown in Figure 4.1 below. The PSNR for this example is 46.9 dB, which matches the PSNR for the baseline method on this example.



True $\boldsymbol{x}$ Restored $\boldsymbol{x} > 0$ Restored $\boldsymbol{x}$ Blurred $\boldsymbol{d}$

Figure 4.1: Truncation Satellite Example Restoration. The truncation method yields a good reconstruction of the blurred image. In particular, the reconstruction with negative values truncated, the second from the left, is highly accurate, only missing the fine details of the original.

As with the baseline method, the truncation method performs well for this case. There is a small amount of noise, discernible in the image that includes negative values in what should be the darkest regions of the image, but the object is still clearly visible.

Most of the examples here use a sufficiently high truncation index to get reasonable reconstruction results. What happens if the truncation index is dropped? We will look at this question for the Satellite Example in this and the next chapter to contrast these two proposed methods, and understand how the truncation method performs with different truncation indices. If we restrict the truncation index to $k = 256$ with $\ell = 16 = m$, we get the reconstruction shown in Figure 4.2, which includes negative values to show the noise pattern more clearly.

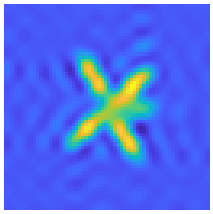


Figure 4.2: Balanced Restricted Truncation Restoration. This image shows a restoration using the truncation method on the Satellite Example with the truncation index decreased from $k = 600$ to $k = 256$, with an even splitting $\ell = m$. There is more noise and fewer details than the original restoration, but the image is still acceptable.

This reconstruction loses some of the fine details, generally appearing blurrier than a truncation index of $k = 600$. Still, it is not terrible. We chose a balanced $\ell = m$ for this example, and as we expect the reconstruction is not notably worse in the vertical or horizontal direction. But if we skew the truncation $k = 256$ so that $\ell = 8$ and $m = 32$, we expect that the approximation of $\boldsymbol{A}_1$, corresponding to the most significant horizontal separable blur term, will be poor, therefore making our reconstruction less accurate horizontally. Indeed, we see this result in the resulting reconstruction shown in Figure 4.3. This reconstruction is very poor. This highlights one of the limitations of the truncation method: for blurs with comparable vertical and horizontal components, the best results tend to arise when $\ell \approx m$, which restricts the truncation index k to numbers which are close to squares.

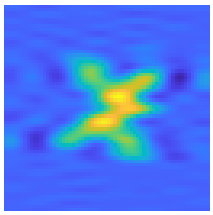
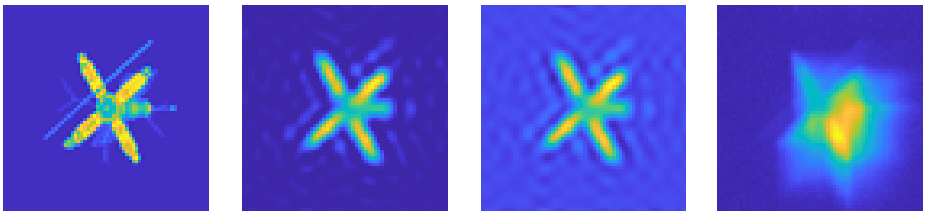


Figure 4.3: Imbalanced Restricted Truncation Restoration. This image shows a restoration using the truncation method on the Satellite Example with the truncation index decreased to $k = 256$, with an uneven splitting $\ell = 8$ and $m = 32$. With such an imbalanced splitting, the restoration becomes warped and much less accurate.

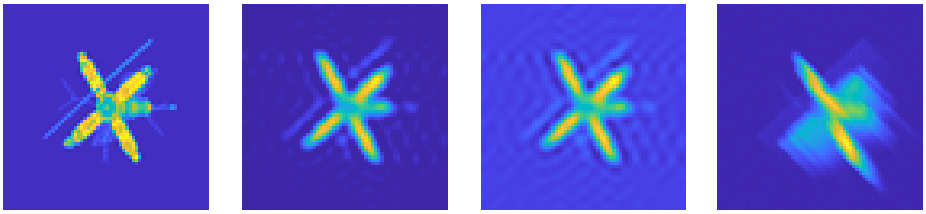
The second recurring example, the Grain Example, has fairly good performance for the truncation method with $\ell = m = 20$. Compared to the baseline method and the method proposed in Chapter 5, the truncation method generally includes different singular vectors. This results in significantly different patterns of error and noise. In particular, the truncation method tends to include singular vectors associated with smaller singular values as explained in Section 4.2. In image processing, this often means including finer details. In Figure 4.4, we see the fairly detailed reconstruction from the truncation method. There are some ringing effects in the reconstruction, which we discuss in the next chapter. The PSNR for this example is 46.2 dB, which is an improvement on the PSNR of 42.8 dB for the baseline method.



True $\boldsymbol{x}$ Restored $\boldsymbol{x} > 0$ Restored $\boldsymbol{x}$ Blurred $\boldsymbol{d}$

Figure 4.4: Truncation Grain Example Restoration. The truncation method yields a good reconstruction of the blurred image in the Grain Example. There is more noise visible than the Satellite Example, as seen in the second from the right image which includes negative values, but the reconstruction is still of good quality.

In the Motion Example, we can even more clearly see what sets the truncation method apart from the baseline method. Whereas the baseline method had a poor reconstruction with significant error, the truncation method using $\ell = 22$ and $m = 25$ produces a reconstruction that has a level of detail as high as the Satellite Example. This is shown in Figure 4.5. The PSNR for this example is 47.6 dB, which is a large improvement on the baseline method's PSNR of 39.0 dB for this example.

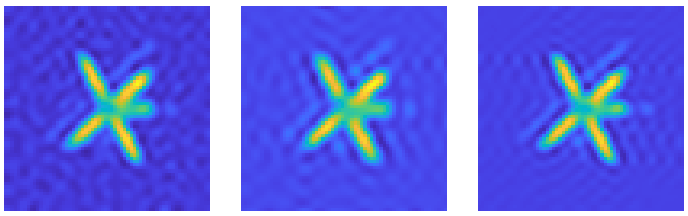


True $\boldsymbol{x}$ Restored $\boldsymbol{x} > 0$ Restored $\boldsymbol{x}$ Blurred $\boldsymbol{d}$

Figure 4.5: Truncation Motion Example Restoration. The truncation method also yields a good reconstruction of the blurred image in the Motion Example. Whereas the baseline method yielded a poor reconstruction, the reconstruction using the truncation method is good enough to include, for example, the thin rod running along parallel to a set of solar panels.

Provided a high enough truncation index k is used, and the split of k into ℓ and m is balanced, the truncation method can produce good restoration results.

Figure 5.6 visually summarizes the results for the three experiments in this section.



Satellite Grain Motion

Figure 4.6: Truncation Method Restoration Summary. This summary image shows the restored images $\boldsymbol{x}$ computed using the truncation method (4.2) for each restoration example.

4.4.2 Approximated Singular Values

As with the baseline method, we ask: how well does the truncation method (4.2) approximate the singular values of the matrix $\boldsymbol{K}$? There is a pattern evident in both the Singular Value Atmospheric Blur Example and the Singular Value Motion Example: the truncation method consistently underestimates the smallest singular values. Recall that the truncation method keeps singular values of $\boldsymbol{K}_1$ that are less

significant than some of the singular values it discards. The expected result is an under-estimate of the smallest singular values kept.

We see this clearly in the Singular Value Atmospheric Blur Example, shown in Figure 4.7. The error starts out smaller than the baseline method produces, but then rises to be higher than the baseline for the smallest singular values. To see a comparison of all methods' singular value approximations, see Appendix B.

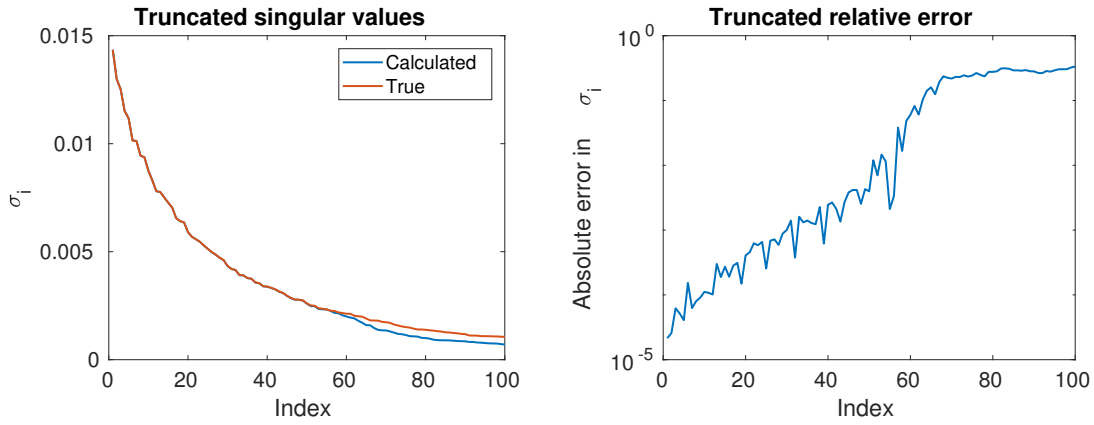


Figure 4.7: Truncation Atmospheric Blur Singular Values. The singular values (left) and relative error (right) are shown for the truncation method approximation of the Singular Value Atmospheric Blur Example. The largest singular values are a good approximation, with around 10^{-5} relative error. The error increases significantly as index increases, with the smallest singular values being a poor estimate.

The underestimation of singular values is even more pronounced in the Singular Value Motion Example. However, recall that the baseline method also underestimated all the singular values, including the largest singular values, for this example. We will see in Chapters 5 and 6 that the pattern of reasonably estimating the largest singular values while underestimating the smallest singular values is not unique to the truncation method for this example. Nonetheless, the relative error becomes extreme for this example, nearing 1.

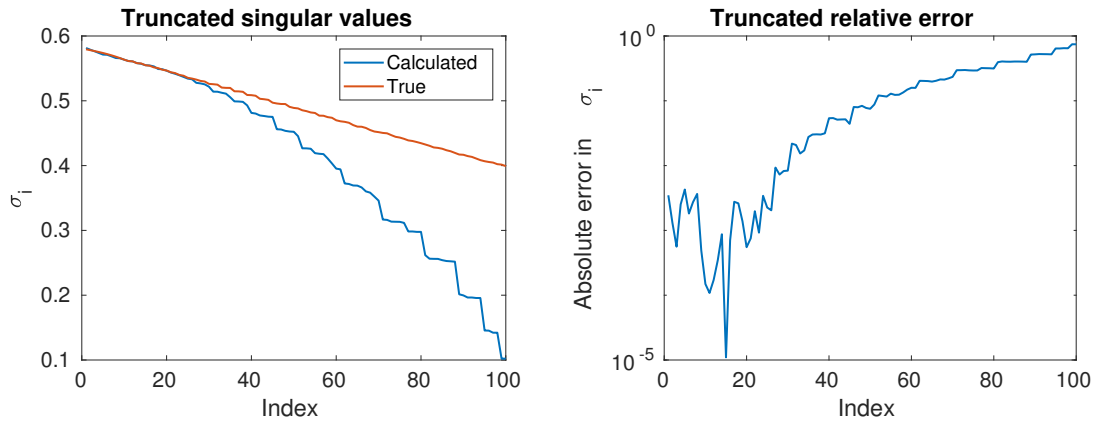


Figure 4.8: Truncation Motion Singular Values. The singular values (left) and relative error (right) are shown for the truncation method approximation of the Singular Value Motion Example. Like the baseline method, the truncation method tends to give a poor estimate of the singular values for this example. However, some of the larger singular values are approximated reasonably well by the truncation method.

For the largest singular values, the truncation method yields a reasonably accurate estimation. However, it should not be used to estimate the singular values near the truncation boundary. As discussed in Chapter 6, one way to use the more accurate singular value computation of the truncation method without suffering from the less-accurate values near the boundary is to compute a too-large TSVD approximation and then truncate the result back down to the desired rank. In the Singular Value Atmospheric Blur Example, Figure 4.7 shows that truncating down to a rank of $k = 60$ from the computed rank of 100 would yield excellent results.

4.4.3 Preconditioning

Unlike the baseline method, the truncation method does not construct a full set of singular vectors and values by default. For preconditioning, this can cause problems. For example if we use a low-rank preconditioner $\boldsymbol{M}$ to precondition a full rank system $\boldsymbol{Kx} + \boldsymbol{e} = \boldsymbol{d}$, we get the system $\boldsymbol{MKx} + \boldsymbol{Me} = \boldsymbol{Md}$. The product $\boldsymbol{MK}$ has a lower rank than $\boldsymbol{K}$, making this problem under-determined. Similarly,

right-preconditioning causes the restored solution to exist in a smaller subspace than the original problem. When using a low rank preconditioner, the rank must be high enough to capture most of the information about the problem.

With this restriction, we used a rank of 1520 for this problem, split into $\ell = 38$ and $m = 40$. The rest of the parameters for this example are identical to the baseline method. It took 1.10 seconds on average to construct the truncated preconditioner, and the iterations took a total of 0.179 seconds to converge, with 7 iterations required for convergence. So, in total, the PCG method converged in 1.28 seconds for the truncation method. This total time is slower than the using the baseline as a preconditioner but faster than no preconditioner. The time taken on iterations is faster for the truncation method than the baseline method, which indicates that the truncation method may be more useful in cases where there are many right-hand sides. Table 4.1 shows the comparative times for no preconditioner versus the truncation method. See Appendix C for a full comparison between all methods.

	No preconditioner	Truncated
Iterations	345	7
Setup time (sec)	0.0	1.10
Calculate time (sec)	6.04	0.179
Total time (sec)	6.04	1.28

Table 4.1: Truncation Preconditioner Timings. A summary of the timing results for the PCLGS problem, with the truncation method used as a preconditioner, is shown. The truncation method saves time and takes fewer iterations compared to an unpreconditioned system.

Despite the added computational cost compared to the baseline method, the truncation method is an effective preconditioner. In particular, it dramatically decreases the number of iterations required for convergence.

4.5 Summary

The truncation method proposed in this chapter remedies the limitations of the baseline method from Chapter 3. The singular values computed by the truncation method use more information in the original matrix $\boldsymbol{K}$ than the baseline method, and the singular values are guaranteed to be nonnegative. In addition, the singular vectors computed depend on more than the first term $\boldsymbol{K}_1$ in the Kronecker summation decomposition of $\boldsymbol{K}$. At the same time, the truncation method has new limitations: it discards information that is more significant than it keeps, causing the smallest singular values to be significantly underestimated.

In practice, this results in mixed performance. Because the truncation method is typically slower than the baseline, the benefits to accuracy may be difficult to justify for applications where accuracy in singular values and vectors is important. In the next chapter, we explore a method that improves upon the baseline without incurring the significant limitations from truncation seen here.

Chapter 5

Reordered Kronecker Method

The baseline TSVD approximation suffers from discarding too much information and not adhering to the definition of singular values. The truncation method suffers from the ordering limitations of Kronecker products of sorted diagonal matrices. The method proposed in this chapter addresses the limitations of both prior methods. We accomplish this by adjusting the truncation method to reorder the singular values of the first term in the Kronecker summation decomposition so that they are sorted. With careful use of data structures, we avoid incurring excess cost in the process.

5.1 Derivation

As with the truncation method, we begin our derivation having separated the first term in the Kronecker summation decomposition (2.4) from the rest:

$$\boldsymbol{K} = \boldsymbol{U}_1 \boldsymbol{\Sigma}_1 \boldsymbol{V}_1^T + \boldsymbol{U}_1 \boldsymbol{U}_1^T \left(\sum_{i=2}^{R} \boldsymbol{A}_i \otimes \boldsymbol{B}_i \right) \boldsymbol{V}_1 \boldsymbol{V}_1^T.$$

As noted in Chapter 4.2, the singular values in $\boldsymbol{\Sigma}_1$ are not sorted in descending order along the diagonal. Although truncating the terms $\boldsymbol{\Sigma}_A$ and $\boldsymbol{\Sigma}_B$ yields better results than truncating $\boldsymbol{\Sigma}_1$ directly, for general matrices $\boldsymbol{K}$ we still discard some singular

values in $\boldsymbol{\Sigma}_1$ which are larger than those we keep. Instead, we wish to keep the top k singular values in $\boldsymbol{\Sigma}_1$ and discard the remaining values.

Fortunately, sorting is computationally cheap in this context. We can easily represent $\boldsymbol{\Sigma}_1$ as a vector $\boldsymbol{\sigma}_1$ containing the diagonal entries of $\boldsymbol{\Sigma}_1$. Then, we sort $\boldsymbol{\sigma}_1$ to determine a permutation $\boldsymbol{P}$ such that $\boldsymbol{P}^T\boldsymbol{\sigma}_1$ is sorted. We do not need to store $\boldsymbol{P}$ explicitly, but can instead store the mapping that reorders $\boldsymbol{\sigma}_1$ into sorted order, and the mapping that undoes that sorting so we implicitly have both $\boldsymbol{P}$ and $\boldsymbol{P}^T$.

With this permutation, we can rewrite

$$\begin{aligned}\boldsymbol{K}_1 &= \boldsymbol{U}_1\boldsymbol{\Sigma}_1\boldsymbol{V}_1^T \\ &= \boldsymbol{U}_1\boldsymbol{P}\boldsymbol{P}^T\boldsymbol{\Sigma}_1\boldsymbol{P}\boldsymbol{P}^T\boldsymbol{V}_1^T.\end{aligned}$$

We now have a permuted Kronecker-based SVD of $\boldsymbol{K}_1$. As before, Kronecker product form is maintained; the matrices $\boldsymbol{U}_1$, $\boldsymbol{\Sigma}_1$, and $\boldsymbol{V}_1$ are not formed explicitly, nor is the permutation explicitly multiplied. However, conceptually we now have new effective singular vector and value matrices $\boldsymbol{U}_1\boldsymbol{P}$, $\boldsymbol{P}^T\boldsymbol{\Sigma}_1\boldsymbol{P}$, and $\boldsymbol{V}_1\boldsymbol{P}$. We proceed with a similar derivation to the truncated version of the algorithm:

$$\begin{aligned}\boldsymbol{K} &= \boldsymbol{U}_1\boldsymbol{P}\boldsymbol{P}^T\boldsymbol{\Sigma}_1\boldsymbol{P}\boldsymbol{P}^T\boldsymbol{V}_1^T + (\boldsymbol{U}_1\boldsymbol{P})(\boldsymbol{P}^T\boldsymbol{U}_1^T)\left(\sum_{i=2}^{R}\boldsymbol{A}_i\otimes\boldsymbol{B}_i\right)(\boldsymbol{V}_1\boldsymbol{P})(\boldsymbol{P}^T\boldsymbol{V}_1^T) \\ &= \boldsymbol{U}_1\boldsymbol{P}\left(\boldsymbol{P}^T\boldsymbol{\Sigma}_1\boldsymbol{P} + (\boldsymbol{P}^T\boldsymbol{U}_1^T)\left(\sum_{i=2}^{R}\boldsymbol{A}_i\otimes\boldsymbol{B}_i\right)(\boldsymbol{V}_1\boldsymbol{P})\right)\boldsymbol{P}^T\boldsymbol{V}_1^T \\ &= \boldsymbol{U}_1\boldsymbol{P}\left(\boldsymbol{P}^T\boldsymbol{\Sigma}_1\boldsymbol{P} + \boldsymbol{P}^T\left(\sum_{i=2}^{R}\boldsymbol{U}_A\boldsymbol{A}_i\boldsymbol{V}_A^T\otimes\boldsymbol{U}_B\boldsymbol{B}_i\boldsymbol{V}_B^T\right)\boldsymbol{P}\right)\boldsymbol{P}^T\boldsymbol{V}_1^T.\end{aligned}$$

The mathematical viability of this algorithm hinges upon the last line above. The Kronecker product summation remains as in the previous algorithms, with permutation applied to the result of the summation. Because we can cheaply compute the products $\boldsymbol{U}_A\boldsymbol{A}_i\boldsymbol{V}_A^T$ and $\boldsymbol{U}_B\boldsymbol{B}_i\boldsymbol{V}_B^T$, we can first calculate those products and then ef-

ficiently compute the permutation. We detail the algorithm for this in Subsection 5.2.1.

But there is one more step before we can permute those values: we must introduce truncation. As with the truncated method in Chapter 4, we seek to truncate the interior quantity $\boldsymbol{P}^T\boldsymbol{\Sigma}_1\boldsymbol{P} + \boldsymbol{P}^T\left(\sum_{i=2}^{R}\boldsymbol{U}_A\boldsymbol{A}_i\boldsymbol{V}_A^T \otimes \boldsymbol{U}_B\boldsymbol{B}_i\boldsymbol{V}_B^T\right)\boldsymbol{P}$ to enable SVD computation. Because $\boldsymbol{P}^T\boldsymbol{\Sigma}_1\boldsymbol{P}$ is sorted, we can truncate it to the top k values. With $\mathbb{1}_k$ as the size $k \times k$, identity matrix, we can write a column truncation operator $\boldsymbol{S} = \begin{bmatrix} \mathbb{1}_k \\ \boldsymbol{0}_{N-k,k} \end{bmatrix}$. Again, this is not formed explicitly as a matrix; instead a function call version of truncation is performed. To prevent the notation from getting unduly cumbersome, denote the reordered and truncated matrices $\bar{\boldsymbol{U}} = \boldsymbol{U}_1\boldsymbol{P}\boldsymbol{S}$, $\bar{\boldsymbol{V}} = \boldsymbol{V}_1\boldsymbol{P}\boldsymbol{S}$, and $\bar{\boldsymbol{\Sigma}} = \boldsymbol{S}^T\boldsymbol{P}^T\boldsymbol{\Sigma}_1\boldsymbol{P}\boldsymbol{S}$. Then proceeding with the derivation:

$$\boldsymbol{K} \approx \bar{\boldsymbol{U}}\left(\bar{\boldsymbol{\Sigma}} + \boldsymbol{S}^T\boldsymbol{P}^T\left(\sum_{i=2}^{R}\boldsymbol{U}_A\boldsymbol{A}_i\boldsymbol{V}_A^T \otimes \boldsymbol{U}_B\boldsymbol{B}_i\boldsymbol{V}_B^T\right)\boldsymbol{P}\boldsymbol{S}\right)\bar{\boldsymbol{V}}^T.$$

Let $\boldsymbol{T}_{\text{perm}} = \bar{\boldsymbol{\Sigma}} + \boldsymbol{S}^T\boldsymbol{P}^T\left(\sum_{i=2}^{R}\boldsymbol{U}_A\boldsymbol{A}_i\boldsymbol{V}_A^T \otimes \boldsymbol{U}_B\boldsymbol{B}_i\boldsymbol{V}_B^T\right)\boldsymbol{P}\boldsymbol{S}$. If we take the SVD $\boldsymbol{T}_{\text{perm}} = \boldsymbol{U}_k\boldsymbol{\Sigma}_k\boldsymbol{V}_k^T$ and let $\breve{\boldsymbol{U}} = \bar{\boldsymbol{U}}\boldsymbol{U}_k = \boldsymbol{U}_1\boldsymbol{P}\boldsymbol{S}\boldsymbol{U}_k$, $\breve{\boldsymbol{V}} = \bar{\boldsymbol{V}}\boldsymbol{V}_k = \boldsymbol{V}_1\boldsymbol{P}\boldsymbol{S}\boldsymbol{V}_k$, then we arrive at the reordered Kronecker TSVD approximation:

$$\boldsymbol{K} \approx \breve{\boldsymbol{U}}\boldsymbol{\Sigma}_k\breve{\boldsymbol{V}}^T. \tag{5.1}$$

5.2 Discussion

Despite the obfuscating notation, this reordering method boils down to a variation on the truncation method. The two differences are: we permute everything, and we no longer truncate $\boldsymbol{U}_1$, $\boldsymbol{\Sigma}_1$, and $\boldsymbol{V}_1$ by truncating their Kronecker product components. We are first applying a permutation operation and then truncate. As always, we do not *actually* perform these multiplications on the singular vector matrices, because

we want to keep Kronecker structure of $\boldsymbol{U}_1$ and $\boldsymbol{V}_1$. So we instead carry around $\boldsymbol{U}_A$, $\boldsymbol{U}_B$, and likewise for $\boldsymbol{V}_1$, and multiply with each term in sequence.

For example, say we want to get the pseudoinverse TSVD solution to the image deconvolution problem (2.1), $\boldsymbol{x}_{\text{TSVD}} = \breve{\boldsymbol{V}}\boldsymbol{\Sigma}_k^{-1}\breve{\boldsymbol{U}}^T\boldsymbol{d}$. (Recall, $\breve{\boldsymbol{V}} = \boldsymbol{V}_1\boldsymbol{P}\boldsymbol{S}\boldsymbol{V}_k$, and $\breve{\boldsymbol{U}}$ is similarly defined.) To compute $\boldsymbol{x}_{\text{TSVD}}$ we would:

1. Compute $\boldsymbol{U}_1^T\boldsymbol{d}$ as $\text{vec}(\boldsymbol{U}_B^T\boldsymbol{D}\boldsymbol{U}_A)$ where $\boldsymbol{D}$ is $\boldsymbol{d}$ reshaped column-wise into a matrix.

2. Permute the result using the row reordering map.

3. Truncate the result to the first k entries.

4. Multiply $\boldsymbol{V}_k$ times the result.

5. Element-wise divide the result by $\boldsymbol{\sigma}_k$, the vector representation of $\boldsymbol{\Sigma}_k$.

6. Multiply $\boldsymbol{U}_k^T$ times the result.

7. Pad the result with $N - k$ zeros (so, pad it with zeros to length N).

8. Permute to undo the row reordering. Call the resulting vector $\boldsymbol{x}_p$.

9. Compute $\text{vec}(\boldsymbol{V}_B\boldsymbol{X}_p\boldsymbol{V}_A^T)$ where $\text{vec}(\boldsymbol{X}_p) = \boldsymbol{x}_p$. The result is $\boldsymbol{x}_{\text{TSVD}}$.

Although this requires several steps, each step is computationally cheap. The most expensive step is either multiplication with $\boldsymbol{V}_k$ (and equally $\boldsymbol{U}_k$) at $O(k^2)$ or multiplication with $\boldsymbol{V}_1$ (and $\boldsymbol{U}_1$) at $O(n^3)$ depending on the relative size of k and n. (See Section 5.3 for time complexity analysis of the work to construct, rather than use, the factorization.) And once again, like the truncation method, storage is cheap, at $O(k^2 + n^2)$ to represent matrices of size up to $O(n^2k)$.

5.2.1 Algorithm Details

The most challenging part of this algorithm is efficiently computing $\boldsymbol{T}_{\text{perm}}$ and related data structures, particularly the reordering map. Improper implementation can make computing $\boldsymbol{T}_{\text{perm}}$ cost $O(n^4 r)$ operations (if $\boldsymbol{U}_A \boldsymbol{A}_i \boldsymbol{V}_A^T \otimes \boldsymbol{U}_B \boldsymbol{B}_i \boldsymbol{V}_B^T$ is explicitly formed for each i from 2 to r). This would be a serious degradation of the computation time. To clarify the efficient algorithm, we detail it here.

Recall $\boldsymbol{T}_{\text{perm}} = \boldsymbol{S}^T \boldsymbol{P}^T \boldsymbol{\Sigma}_1 \boldsymbol{P} \boldsymbol{S} + \boldsymbol{S}^T \boldsymbol{P}^T \left(\sum_{i=2}^{R} \boldsymbol{U}_A \boldsymbol{A}_i \boldsymbol{V}_A^T \otimes \boldsymbol{U}_B \boldsymbol{B}_i \boldsymbol{V}_B^T \right) \boldsymbol{P} \boldsymbol{S}$. Here, $\boldsymbol{P}$ and $\boldsymbol{S}$ are operators which are written as matrices; in implementation they are not formed. $\boldsymbol{S}$ is a truncation operator; to represent it, we only need to know the sizes N and k being truncated from and to. Representing $\boldsymbol{P}$ is not difficult, but requires more attention.

When applied on the left, $\boldsymbol{P}^T$ is the row reordering that maps the vector $\boldsymbol{\sigma}_1$, representing the diagonal of $\boldsymbol{\Sigma}_1$, into sorted order. Applied on the right, it is the column reordering which maps the sorted row vector $\boldsymbol{\sigma}_1^T \boldsymbol{P}$ back into the original order $\boldsymbol{\sigma}_1^T$. Similarly, $\boldsymbol{P}$ applied on the left is the row reordering that maps the sorted vector $\boldsymbol{P}^T \boldsymbol{\sigma}_1$ back to the original ordering $\boldsymbol{\sigma}_1$. When $\boldsymbol{P}$ is applied on the right, it reorders the columns of $\boldsymbol{\sigma}_1^T$ into sorted order. Because $\boldsymbol{\Sigma}_1$ is square, the mappings for rows and columns are the same. The upshot is that, for the most efficient computation, we should pre-compute two mappings: the mapping of the rows of $\boldsymbol{\sigma}_1$ from their original order into sorted order, and the inverse of that mapping which returns the sorted vector to its original order. These mappings are different for different matrices $\boldsymbol{K}$ because they depend on the relative decay of the singular values in $\boldsymbol{\Sigma}_A$ and $\boldsymbol{\Sigma}_B$, but are computed once per $\boldsymbol{K}$.

The mapping is cheap and easy to compute, and in some programming languages, it is a built-in function. For example, in MATLAB the `sort` method has a version with two output arguments, the second of which is the mapping that, when applied to the original vector, gives the sorted result. If such a function is not available, a

parallel sort works. Call $\boldsymbol{i}$ the vector containing the values 1 to N in sorted order. Then we can do an $O(N\log(N)) = O(n^2\log(n))$ sort on $\boldsymbol{\sigma}_1$ where, for every element we move in $\boldsymbol{\sigma}_1$, we move the element at the same index in $\boldsymbol{i}$. Call $\bar{\boldsymbol{i}}$ the resulting permutation of $\boldsymbol{i}$. What does $\bar{\boldsymbol{i}}$ give us? If to move to sorted order, an entry of $\boldsymbol{\sigma}_1$ moves from index p in the original vector to index q in the sorted vector, then the vector $\bar{\boldsymbol{i}}$ contains the value p at index q. So the first entry in $\bar{\boldsymbol{i}}$ indicates the row at which the maximum value of $\boldsymbol{\sigma}_1$ is stored, the second entry in $\bar{\boldsymbol{i}}$ indicates where the second-to-maximum value of $\boldsymbol{\sigma}_1$ is stored, and so on. So, if we want to know which rows of $\boldsymbol{\sigma}_1$ contain the top k values, we can use the top k entries of $\bar{\boldsymbol{i}}$.

For use in the full factorization, we need to compute the inverse permutation as well, but this is not necessary to calculate $\boldsymbol{T}_{\text{perm}}$. Because we left multiply by $\boldsymbol{P}^T$ and right multiply by $\boldsymbol{P}$, we only need the mapping corresponding to sorting. And further, we only need the mapping corresponding to sorting for the top k entries of $\boldsymbol{\sigma}_1$. As we just saw, this is given by the first k entries of $\bar{\boldsymbol{i}}$. We should keep a separate copy of $\bar{\boldsymbol{i}}$ to be used as $\boldsymbol{P}$ for the full factorization, and can then truncate $\bar{\boldsymbol{i}}$ to its first k entries as $\bar{\boldsymbol{k}}$.

$\boldsymbol{T}_{\text{perm}}$ is constructed iteratively using partial summations. It is first initialized using the sorted $\bar{\boldsymbol{\Sigma}}$; it is important that the initial value for the partial sum of $\boldsymbol{T}_{\text{perm}}$ is a matrix, because the full quantity is a matrix. Then each term corresponding to an index i in the summation $\boldsymbol{S}^T\boldsymbol{P}^T\left(\sum_{i=2}^{R}\boldsymbol{U}_A\boldsymbol{A}_i\boldsymbol{V}_A^T\otimes\boldsymbol{U}_B\boldsymbol{B}_i\boldsymbol{V}_B^T\right)\boldsymbol{P}\boldsymbol{S}$ is added iteratively. The products $\boldsymbol{U}_A\boldsymbol{A}_i\boldsymbol{V}_A^T$ and $\boldsymbol{U}_B\boldsymbol{B}_i\boldsymbol{V}_B^T$ must be computed explicitly for each iteration. However, the permutation and truncation mapping to go from those terms to $\boldsymbol{S}^T\boldsymbol{P}^T(\boldsymbol{U}_A\boldsymbol{A}_i\boldsymbol{V}_A^T\otimes\boldsymbol{U}_B\boldsymbol{B}_i\boldsymbol{V}_B^T)\boldsymbol{P}\boldsymbol{S}$ is the same for every index. Using this fact, we can avoid forming the Kronecker products $\boldsymbol{U}_A\boldsymbol{A}_i\boldsymbol{V}_A^T\otimes\boldsymbol{U}_B\boldsymbol{B}_i\boldsymbol{V}_B^T$ explicitly.

The quantity we are trying to compute is a permutation and truncation of a Kronecker product. To discuss the mappings involved, we need to give names to several components of the computation. Let $\boldsymbol{U}_A\boldsymbol{A}_i\boldsymbol{V}_A^T = \ddot{\boldsymbol{A}}_i$ and $\boldsymbol{U}_B\boldsymbol{B}_i\boldsymbol{V}_B^T = \ddot{\boldsymbol{B}}_i$,

so $\boldsymbol{U}_A\boldsymbol{A}_i\boldsymbol{V}_A^T \otimes \boldsymbol{U}_B\boldsymbol{B}_i\boldsymbol{V}_B^T = \ddot{\boldsymbol{A}}_i \otimes \ddot{\boldsymbol{B}}_i$. Let the entry at row r and column c of an arbitrary matrix $\boldsymbol{Z}$ be denoted as $\boldsymbol{Z}(r,c)$. To map from the entries of $\ddot{\boldsymbol{A}}_i$ and $\ddot{\boldsymbol{B}}_i$ to their Kronecker product, we have

$$\ddot{\boldsymbol{A}}_i \otimes \ddot{\boldsymbol{B}}_i(r,c) = \ddot{\boldsymbol{A}}_i\left(\left\lfloor\frac{r-1}{n}\right\rfloor + 1, \left\lfloor\frac{c-1}{n}\right\rfloor + 1\right)\ddot{\boldsymbol{B}}_i\left([r-1]\%n+1, [c-1]\%n+1\right)$$

where % denotes the modulus operator. In this case, we are assuming r and c are one-indexed; for a zero-indexed version, the mapping is simply

$$\ddot{\boldsymbol{A}}_i \otimes \ddot{\boldsymbol{B}}_i(r,c) = \ddot{\boldsymbol{A}}_i\left(\left\lfloor\frac{r}{n}\right\rfloor, \left\lfloor\frac{c}{n}\right\rfloor\right)\ddot{\boldsymbol{B}}_i\left(r\%n, c\%n\right).$$

The entries we need in $\ddot{\boldsymbol{A}}_i \otimes \ddot{\boldsymbol{B}}_i$ are the entries in the intersection of the rows and columns with indices equal to entries of $\bar{\boldsymbol{k}}$. That is, each index (r,c) we need to compute has both r and c coming from an entry of $\bar{\boldsymbol{k}}$. Further, the set of rows and columns are equal. We can therefore compute the indices $\bar{\boldsymbol{a}} = \left\lfloor\bar{\boldsymbol{k}}/n\right\rfloor$ and $\bar{\boldsymbol{b}} = \bar{\boldsymbol{k}}\%n$, where both division / and modulus % are applied to each element of $\bar{\boldsymbol{k}}$. The vectors $\bar{\boldsymbol{a}}$ and $\bar{\boldsymbol{b}}$ contain the rows (and columns) of $\ddot{\boldsymbol{A}}_i$ and $\ddot{\boldsymbol{B}}_i$ that remain in the permuted and truncated $\boldsymbol{S}^T\boldsymbol{P}^T\left(\boldsymbol{U}_A\boldsymbol{A}_i\boldsymbol{V}_A^T \otimes \boldsymbol{U}_B\boldsymbol{B}_i\boldsymbol{V}_B^T\right)\boldsymbol{P}\boldsymbol{S}$. So instead of forming the full Kronecker products $\ddot{\boldsymbol{A}}_i \otimes \ddot{\boldsymbol{B}}_i$, we can compute the k^2 elements that are not truncated out by multiplying specific entries of $\ddot{\boldsymbol{A}}_i$ and $\ddot{\boldsymbol{B}}_i$. Denote the entries at index j of $\bar{\boldsymbol{a}}$ as $\bar{\boldsymbol{a}}(j)$ and likewise for $\bar{\boldsymbol{b}}$. To compute the entry of $\boldsymbol{S}^T\boldsymbol{P}^T(\boldsymbol{U}_A\boldsymbol{A}_i\boldsymbol{V}_A^T \otimes \boldsymbol{U}_B\boldsymbol{B}_i\boldsymbol{V}_B^T)\boldsymbol{P}\boldsymbol{S}$ at row r column c, we multiply the entry of $\ddot{\boldsymbol{A}}_i$ at row $\bar{\boldsymbol{a}}(r)$ and column $\bar{\boldsymbol{a}}(c)$ with the entry of $\ddot{\boldsymbol{B}}_i$ at $\bar{\boldsymbol{b}}(r)$ and column $\bar{\boldsymbol{b}}(c)$. Explicitly,

$$\boldsymbol{S}^T\boldsymbol{P}^T\left(\ddot{\boldsymbol{A}}_i \otimes \ddot{\boldsymbol{B}}_i\right)\boldsymbol{P}\boldsymbol{S}(r,c) = \ddot{\boldsymbol{A}}_i(\bar{\boldsymbol{a}}(r), \bar{\boldsymbol{a}}(c))\ddot{\boldsymbol{B}}_i(\bar{\boldsymbol{b}}(r), \bar{\boldsymbol{b}}(c)).$$

The permutation-and-truncation-Kronecker mappings $\bar{\boldsymbol{a}}$ and $\bar{\boldsymbol{b}}$ only need to be computed once for all indices i from 2 to r in the Kronecker summation to compute

$\boldsymbol{T}_{\text{perm}}$. Once computed for the first time, the mappings can be used for all of the iterations to compute partial summations. The work for each iteration is then limited to computing the products.

However, the implementation details of the mapping can have a significant impact on computation time. This approach gives a considerable time complexity savings, at $O(n^2 \log(n) + k^2 r)$ as detailed in Section 5.3, but the complexity masks memory retrieval concerns. $\ddot{\boldsymbol{A}}_i$ and $\ddot{\boldsymbol{B}}_i$ have n^2 entries each. If n is large and the whole matrices cannot fit in cache, then sequential accesses to rows and columns of the matrices may cause cache misses. It is important to be careful about the ordering of the indexes accessed while computing each partial summation.

There are many steps required to compute $\boldsymbol{T}_{\text{perm}}$, making the reordering method the most complicated of the methods discussed so far. However, the increased cost in time necessary to implement the algorithm is worth it due to the improved accuracy. The reordering method is the most broadly accurate of the methods up to this point.

5.2.2 Efficacy

The reordering truncation method avoids the pitfalls of the methods from Chapters 3 and 4. The singular values are non-negative and the amount of important information that gets discarded in $\boldsymbol{K}_1$ is minimized. Section 5.4 shows its actual performance on test problems. Compared to the baseline and truncation method, it is the best choice to get good performance for an operator $\boldsymbol{K}$ for which no a priori knowledge about $\boldsymbol{K}_1$ is known. On matrices $\boldsymbol{K}$ for which $\boldsymbol{K}_1$ is a poor approximation, it vastly outperforms the baseline. And on matrices for which $\boldsymbol{K}_1$ is a good approximation, the reordering method typically yields (with an exception noted shortly) more accurate reconstruction results compared to the reconstructions from the true TSVD than the simple truncation method for the same truncation index; this is because the reordering method includes more significant singular vectors. This is detailed experimentally in

Appendix A. One pathological case when the reordering method does not give better results is when all singular values are equal in $\boldsymbol{\Sigma}_1$, in which case the reordering and truncation methods produce identical results. Further, with the reordering method there is no restriction on the truncation index chosen. Unlike the simple truncation method, the reordering method enables use of prime truncation indexes k. We will see that the truncation method does produce more fine details in its reconstructions, which may be preferable for some applications. But this is less accurate compared to the true TSVD reconstruction, as verified by the truncation method producing less accurate singular value approximation.

To be clear, this reordering method is not generally highly accurate. The goal is not to exactly compute a TSVD, but to instead cheaply approximate the TSVD. By approximating $\boldsymbol{K}$ with $r < R$ terms in the Kronecker summation and truncating the matrix factorization, information is lost. However, as a cheap approximation, this gives the best results in general of the methods described.

5.3 Time Complexity

In this section we show that the reordering method is computationally feasible, with an $O(n^3 r + k^2 r + k^3)$ running time for the image deconvolution problem. As with the basic truncation method, this is not as fast as the baseline, but is computationally feasible. This time complexity masks that care should be taken in certain steps to optimize performance for modern computer architectures; see Subsection 5.2.1 for details.

Recall the steps of the reordering Kronecker-based TSVD algorithm:

1. Compute the approximate Kronecker product decomposition $\boldsymbol{K} \approx \sum_{i=1}^{r} \boldsymbol{A}_i \otimes \boldsymbol{B}_i$.
2. Compute the SVD $\boldsymbol{A}_1 \otimes \boldsymbol{B}_1 = \boldsymbol{U}_1 \boldsymbol{\Sigma}_1 \boldsymbol{V}_1^T$ indirectly by computing $\boldsymbol{A}_1 = \boldsymbol{U}_A \boldsymbol{\Sigma}_A \boldsymbol{V}_A^T$ and $\boldsymbol{B}_1 = \boldsymbol{U}_B \boldsymbol{\Sigma}_B \boldsymbol{V}_B^T$.

3. Sort the vector $\boldsymbol{\sigma}_1$ (corresponding to $\boldsymbol{\Sigma}_1$) to get $\bar{\boldsymbol{i}}$ and therefore $\bar{\boldsymbol{k}}$ as detailed in Subsection 5.2.1, as well as $\bar{\boldsymbol{\Sigma}}_1$. Store a copy of $\bar{\boldsymbol{i}}$ and its inverse permutation to represent $\boldsymbol{P}$.

4. Compute $\bar{\boldsymbol{a}}$ and $\bar{\boldsymbol{b}}$ from $\bar{\boldsymbol{k}}$ as detailed in Subsection 5.2.1.

5. Compute $\boldsymbol{T}_{\text{perm}}$ by initializing to $\bar{\boldsymbol{\Sigma}}_1$ and then, for each index i from 2 to r,

   (a) compute $\ddot{\boldsymbol{A}}_i = \boldsymbol{U}_A \boldsymbol{A}_i \boldsymbol{V}_A^T$ and $\ddot{\boldsymbol{B}}_i = \boldsymbol{U}_B \boldsymbol{B}_i \boldsymbol{V}_B^T$ and,

   (b) using $\bar{\boldsymbol{a}}$ and $\bar{\boldsymbol{b}}$, compute $\boldsymbol{S}^T \boldsymbol{P}^T \left(\ddot{\boldsymbol{A}}_i \otimes \ddot{\boldsymbol{B}}_i \right) \boldsymbol{P} \boldsymbol{S}$ as detailed in subsection 5.2.1. Add this to the partial summation of $\boldsymbol{T}_{\text{perm}}$.

6. Compute the SVD $\boldsymbol{T}_{\text{perm}} = \boldsymbol{U}_k \boldsymbol{\Sigma}_k \boldsymbol{V}_k^T$.

7. Create a data structure to represent $\boldsymbol{S}$; it must store N and k.

8. Return the resulting SVD approximation $\boldsymbol{U}_1 \boldsymbol{P} \boldsymbol{S} \boldsymbol{U}_k \boldsymbol{\Sigma}_k \boldsymbol{V}_k^T \boldsymbol{S}^T \boldsymbol{P}^T \boldsymbol{V}_1^T$ where $\boldsymbol{P}$ and $\boldsymbol{S}$ are represented using compact data structures.

Theorem 3. *If computing the Kronecker summation decomposition 2.5 takes $O(\mathcal{T})$ operations, the reordering method takes $O(\mathcal{T} + n^3 r + k^2 r + k^3)$ operations.*

Proof. The time complexity of each step is:

1. As with the baseline, the time complexity of computing the Kronecker summation decomposition is called $O(\mathcal{T})$.

2. Similarly, computing the full SVD of $\boldsymbol{A}_1$ and $\boldsymbol{B}_1$, which are size $n \times n$, takes $O(n^3)$ operations.

3. The parallel sort of two vectors of length n^2 takes $O(n^2 \log(n))$ operations.

4. Each index in the length k vector $\bar{\boldsymbol{k}}$ undergoes a constant number of operations to compute $\bar{\boldsymbol{a}}$ and $\bar{\boldsymbol{b}}$; this therefore takes $O(k)$ operations.

5. Initializing $\boldsymbol{T}_{\text{perm}}$ takes $O(k^2)$ operations. For each i from 2 to r,

   (a) Computing $\ddot{\boldsymbol{A}}_i$ and $\ddot{\boldsymbol{B}}_i$ takes $O(n^3)$ operations.

   (b) Computing the products to get the partial sum $\boldsymbol{S}^T\boldsymbol{P}^T\left(\ddot{\boldsymbol{A}}_i \otimes \ddot{\boldsymbol{B}}_i\right)\boldsymbol{P}\boldsymbol{S}$ takes $O(k^2)$ operations.

   Accounting for each iteration, this step then takes $O(n^3 r + k^2 r)$ operations.

6. Taking the SVD of the $k \times k$ matrix $\boldsymbol{T}_{\text{perm}}$ takes $O(k^3)$ operations.

7. Storing N and k takes $O(1)$ operations.

8. Returning the already-computed quantities takes $O(1)$ time.

The time complexity of the algorithm is the sum of the complexities of each step in the algorithm. The reordering method therefore has a time complexity of $O(\mathcal{T} + O(n^3)+O(n^2\log(n))+O(k)+O(n^3r+k^2r)+O(k^3)+O(1) = O(\mathcal{T}+n^3r+k^2r+k^3)$. □

In theory and practice this is the slowest method to compute. But the actual time used is not much worse than the standard truncation method (4.2). The increase in accuracy can justify using the reordering method in some applications.

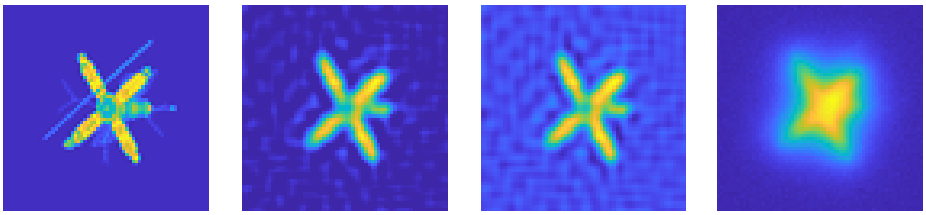
5.4 Performance

5.4.1 Direct Reconstruction

The first test of the reordering method's performance is its ability to reconstruct images when used as a direct method. It performs well, within the limitation that computing more singular values and vectors incurs larger cost. With a limited set of singular values and vectors, the reconstructions vary from good to borderline.

In the Satellite Example, the reconstruction is about as good as the reconstructions created by the baseline method, with slightly less fine detail due to inclusion of

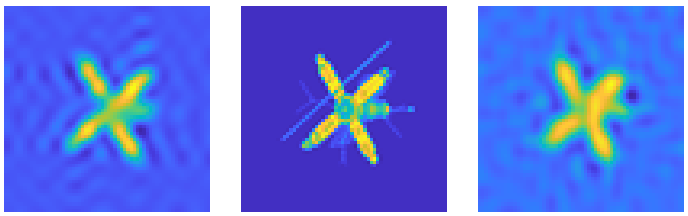
different singular vectors. This can be seen in Figure 5.1 below. This result is unsurprising, as we expect the reordering method to work well for this example. The PSNR for this example is 45.1 dB, which is just under that of the baseline and truncation methods which both have a PSNR of 46.9 dB.



True $\boldsymbol{x}$ Restored $\boldsymbol{x} > 0$ Restored $\boldsymbol{x}$ Blurred $\boldsymbol{d}$

Figure 5.1: Reordering Satellite Example Restoration. The reordering method yields a good reconstruction for this example.

Before proceeding to the Grain Example, we revisit the effect of restricting the Satellite Example to a smaller $k = 256$ terms. In Figure 5.2, we show enlarged versions of the reconstructions created using the truncation and reordering method. Different singular vectors are used in each restoration. The truncation method has a notable ringing effect, whereas the reordering method has different, more uniform noise pattern. The truncated method reconstructs a slimmer set of solar panels, whereas the reordering method gives wider, more uniformly accurate reconstructions of the solar panels. The truncation method has hints of the thin rod; the reordering method does not. Overall, the truncation method restoration is likely preferable.

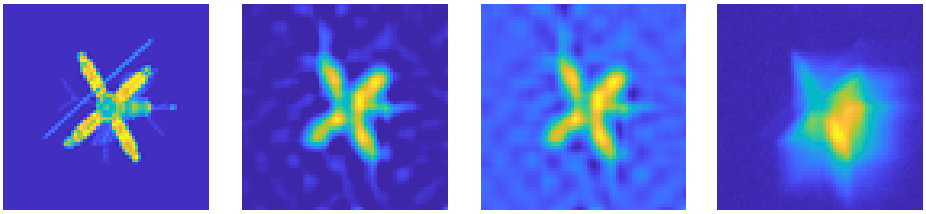


Truncated True Reordered

Figure 5.2: Truncated versus Reordered Method for Restricted Satellite Example. This figure shows the truncated method restoration, true image, and reordering method restoration for the Grain Example. The truncation method has more fine details, but is less accurate on coarser details. The noise pattern differs between the restorations as well.

The upshot is that it can be difficult to know ahead of time which method will give better performance for a specific application. The truncation method naturally includes singular vectors with lower singular values, so it reconstructs more fine detail. But, this can come at the expense of some of the broader data in the image. The restoration results depend on the characteristics of the blur operator and PSF (particularly the decay of singular values of both) and the image being reconstructed. Often, computing a small example is efficient to generalize to a larger example. Explicitly, this is process is to downscale the PSF and image, test on a small problem with both methods, and use whichever works better for the full-sized problem. Down-scaling too heavily may obfuscate fine details, so we caution against down-scaling excessively.

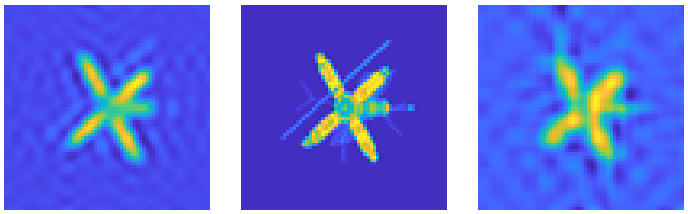
With that aside, we return to the more general examples discussed in each chapter. The Grain Example reconstruction is visually worse than the reordering method restoration from the Satellite Example. The Grain Example reconstruction, shown in Figure 5.3, is the same quality as the reconstruction using baseline method, but less detailed than the restoration from the truncation method. The PSNR for this example is 42.7 dB, which is nearly identical to the baseline method PSNR of 42.8 dB and less than the truncation method PSNR of 46.2 dB.



True $\boldsymbol{x}$ Restored $\boldsymbol{x} > 0$ Restored $\boldsymbol{x}$ Blurred $\boldsymbol{d}$

Figure 5.3: Reordering Grain Example Restoration. The reordering method yields a moderate quality reconstruction for the Grain Example. The restored image is much more blurred in this reconstruction than the Satellite Example.

The more noteworthy part of this example is how the reconstruction differs from the truncation method. Figure 5.4 below shows the different reconstructions. In this example, the truncation method uses significantly different singular values than the reordering method. The noise pattern is very different among the two reconstructions. This is a case where the reconstruction from the truncation method is likely preferable to the reconstruction from the reordering method, despite giving a less accurate restoration compared to a true rank-400 TSVD as discussed in Appendix A.

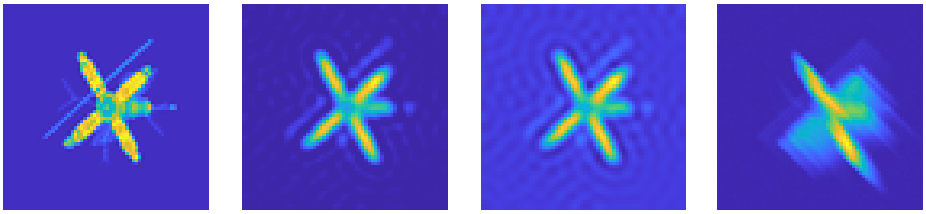


Truncated True Reordered

Figure 5.4: Truncated versus Reordered Method for Grain Example. This figure shows the truncated and reordering method restorations for the full Grain Example. Because it includes more fine details, the truncation method restoration is preferable for this case, even though it is less accurate to the true rank 400 TSVD restoration.

The final example is the Motion Example. Recall that the baseline method performed poorly for this example. However, the reordering method gives a good reconstruction using $k = 550$ singular values and vectors as shown in Figure 5.5. We do not compare the result to the truncation method result here because both give high

quality restorations with little visual difference. The PSNR for this example is 47.7 dB, which almost matches the truncation method's PSNR of 47.6 dB, and is much better than the PSNR of 39.0 dB for the baseline method.

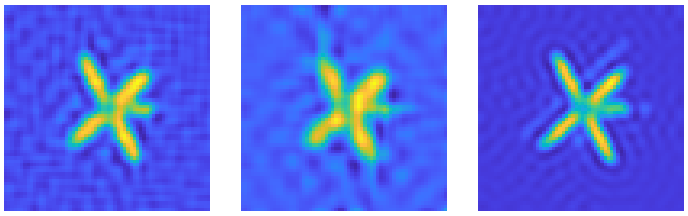


True $\boldsymbol{x}$ Restored $\boldsymbol{x} > 0$ Restored $\boldsymbol{x}$ Blurred $\boldsymbol{d}$

Figure 5.5: Reordering Motion Example Restoration. The reordering method reconstruction for the motion example is excellent, with fine details lacking. Unlike the baseline method, the reordering method yields a successful restoration.

Of the methods in Chapters 3-5, the reordering method is typically the most accurate to a true rank k TSVD restoration. In some cases, this makes it less preferable than the truncation method for restoring images, especially when the fine details of the reconstruction are important.

Figure 5.6 visually summarizes the results for the three experiments in this section.



Satellite Grain Motion

Figure 5.6: Reordering Method Restoration Summary. This summary image shows the restored images $\boldsymbol{x}$ computed using the reordering method (5.1) for each restoration example.

5.4.2 Approximated Singular Values

The second test of the reordering method (5.1) is its ability to approximate singular values. Its performance is problem-dependent as seen with the baseline and truncation

methods.

The first test is the Singular Value Atmospheric Blur Example. For this problem, the reordering method gives a close approximation of the singular values for most of the values, but the approximation worsens near the tail. Indeed, the relative error in the computed singular values increases as the singular value index increases, as seen in Figure 5.7 below. Notably, compared to the truncation method, the singular values start to significantly under-estimate the true singular values at a later point. The inflection here is around index 80, whereas for the truncation method this inflection occurs around index 60. Otherwise, their performance is similar, making the reordering method more accurate overall.

Compared to the baseline method, the reordering method has lower error for the highest singular values, and higher error for the smallest singular values. Appendix B compares all methods simultaneously.

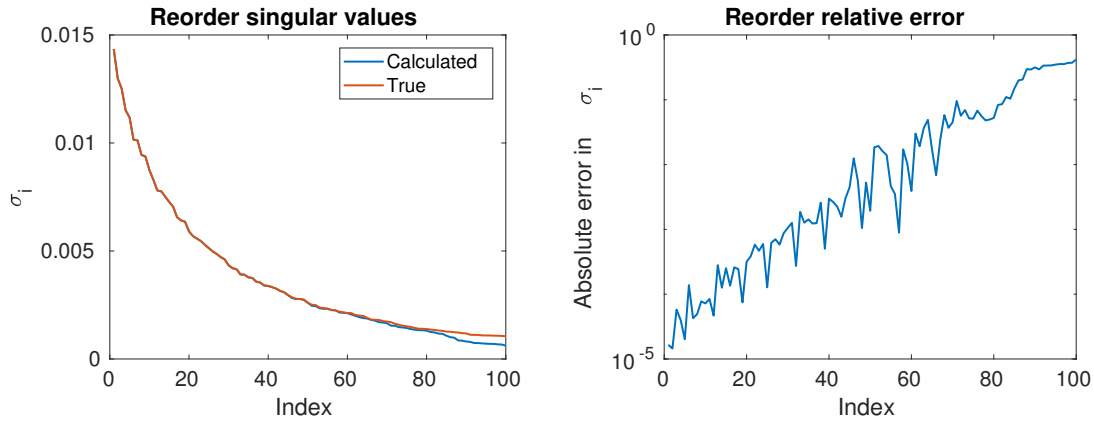


Figure 5.7: Reordering Atmospheric Blur Singular Values. The singular values and relative error are shown for the reordering method approximation of the Singular Value Atmospheric Blur Example. The largest singular values are a good approximation, with around 10^{-5} relative error. The error increases significantly as index increases, with the smallest singular values being a poor estimate. Notably, the smallest singular values become a poor estimate more slowly than the truncation method.

The reordering method gives a much worse estimate of the singular values in the more-challenging Singular Value Motion Example. The error is very similar to

the truncation method, with slightly higher error for most of the singular values. However, the error is smaller than the baseline method, which even more severely underestimates the singular values. Only for the smallest singular values does the baseline method give a better estimate than the reordering method.

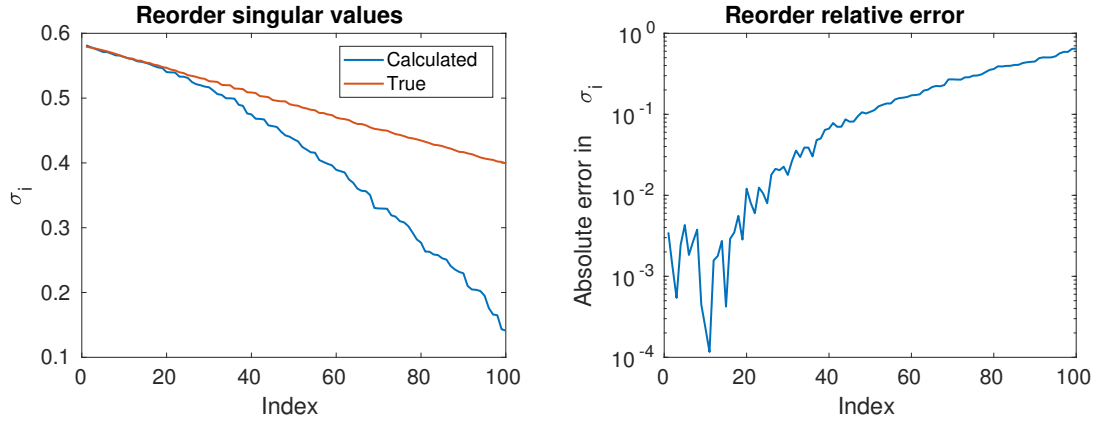


Figure 5.8: Reordering Motion Singular Values. The singular values and relative error are shown for the reordering method approximation of the Singular Value Motion Example. Like the two prior methods, the reordering method gives a poor estimate of the singular values, particularly the smallest values. Like the truncation method, some of the larger singular values are approximated reasonably well, despite the overall poor estimation.

The reordering method significantly corrects the error in the smallest singular values computed by the truncation method. However, the reordering method can still give inaccurate results for problems that are difficult for each method, such as the Singular Value Motion Example.

5.4.3 Preconditioning

The reordering method performs nearly identically to the truncation method in the preconditioning test. The setup of the reordering method, which requires computing the approximate TSVD, took very slightly longer, and both methods took the same number of iterations to converge. Computing the TSVD took on average 1.22 seconds. In total, completing the 7 iterations took 0.202 seconds, for a total computation time

from start to finish of 1.43 seconds. This total time is only .10 second slower than the truncation method, which is not significant enough to conclude the reordering method will consistently be slower for this test problem. So, like the truncation method, the reordering method is considerably faster than not using a preconditioner but slower than the baseline method, with the most significant difference in the setup time. As with the truncation method, the reordering method may be preferable over the baseline in cases when multiple right-hand sides are used, or when $\boldsymbol{K}$ is poorly approximated by $\boldsymbol{K}_1$. Appendix C details the differences between the performance of each preconditioner.

	No preconditioner	Reordering
Iterations	345	7
Setup time (sec)	0.0	1.22
Calculate time (sec)	6.04	0.208
Total time (sec)	6.04	1.43

Table 5.1: Reordering Method Timings for PCGLS. A summary of the timing results for the PCLGS problem, with the reordering method used as a preconditioner, is shown. The reordering method saves time and takes fewer iterations compared to an unpreconditioned system.

The reordering method is an effective preconditioner overall, reducing the number of iterations and time taken to converge compared to not using a preconditioner.

5.5 Summary

The reordering method solves the issues that arose from the prior two methods: it does not have negative singular values, it can produce good results when $\boldsymbol{K}_1 \not\approx \boldsymbol{K}$, and it does not severely underestimate the smallest singular values it computes. Consequentially, it is the most accurate method of the three in general. For some cases, more accurate can be less preferable, such as certain image restoration problems. Regardless, the reordering method is also the most complicated method to implement.

The reordering method uses several data structures that are unnecessary for the prior methods. Additionally, the algorithm involves memory accesses which can be inefficient if caching is not carefully considered. However, when implemented carefully, the algorithm is close the complexity of the truncation method, making it feasible for reasonable truncation index choices k. The reordering method is therefore both effective and efficient.

Chapter 6

Hybrid Method

The baseline method can give excellent results for problems when $\boldsymbol{K} \approx \boldsymbol{K}_1$. When this is not the case, the results are notably worse, although the relative error is fairly consistent across singular value indices. Additionally, the baseline method is extremely cheap to compute. The reordering and truncation methods tend to be more accurate on the largest singular values and less accurate on the smallest singular values. This leads to the question: can we get the benefits of both, namely, high accuracy on the largest singular values and reasonable accuracy with low cost on the lowest singular values? That is the goal of using the hybrid method derived in this chapter, which combines the strengths of the truncation-based methods with the baseline method.

6.1 Derivation

The hybrid method can be applied to the reordering or standard truncation method. Here, we discuss the derivation for the standard truncation method, but the same procedure yields a hybrid reordering method.

We will start by re-deriving the truncation method from a different perspective than the original derivation in Section 4.1. From the derivation of the baseline method

in Section 3.1, we have

$$\boldsymbol{K} = \boldsymbol{U}_1 \left(\boldsymbol{\Sigma}_1 + \boldsymbol{U}_1^T \left(\sum_{i=2}^{R} \boldsymbol{A}_i \otimes \boldsymbol{B}_i \right) \boldsymbol{V}_1 \right) \boldsymbol{V}_1^T. \tag{6.1}$$

We want to partition the SVD $\boldsymbol{K}_1 = \boldsymbol{U}_1 \boldsymbol{\Sigma}_1 \boldsymbol{V}_1^T$ so we can separate the values and vectors kept in the truncation method from those that are discarded. Recall, though, that in the truncation method we do not actually truncate $\boldsymbol{U}_1 = \boldsymbol{U}_A \otimes \boldsymbol{U}_B$ and the related matrices; we instead individually truncate $\boldsymbol{U}_A$, $\boldsymbol{U}_B$, and so on. Let $\boldsymbol{Q}$ be the matrix that reorders $\boldsymbol{U}_1$ so that $\boldsymbol{U}_{A,\ell} \otimes \boldsymbol{U}_{B,m}$ (as defined in Section 4.1) is equal to the first k columns of $\boldsymbol{U}_1 \boldsymbol{Q}$. Because $\boldsymbol{Q}$ is a permutation matrix, $\boldsymbol{K}_1 = \boldsymbol{U}_1 \boldsymbol{Q} \boldsymbol{Q}^T \boldsymbol{\Sigma}_1 \boldsymbol{Q} \boldsymbol{Q}^T \boldsymbol{V}_1^T$. We can use this permutation to get our desired partitioning, such that

$$\begin{aligned} \boldsymbol{U}_1 \boldsymbol{Q} &= \begin{bmatrix} \boldsymbol{U}_{1,k} & \boldsymbol{U}_{1,0} \end{bmatrix} \\ \boldsymbol{Q}^T \boldsymbol{\Sigma}_1 \boldsymbol{Q} &= \begin{bmatrix} \boldsymbol{\Sigma}_{1,k} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{\Sigma}_{1,0} \end{bmatrix} \\ \boldsymbol{V}_1 \boldsymbol{Q} &= \begin{bmatrix} \boldsymbol{V}_{1,k} & \boldsymbol{V}_{1,0} \end{bmatrix}. \end{aligned}$$

Recall that the truncation method uses matrices $\boldsymbol{U}_{1,k}$, $\boldsymbol{\Sigma}_{1,k}$, and $\boldsymbol{V}_{1,k}$. Here, we define $\boldsymbol{U}_{1,0}$, $\boldsymbol{\Sigma}_{1,0}$, and $\boldsymbol{V}_{1,0}$ to be the matrices containing the permuted vectors discarded by the truncation method. (We give more detail on this permutation $\boldsymbol{Q}$ in Subsection 6.2.1.)

Then we can rework the above equality (6.1) to use this partitioning. We avoid using the partitioning of $\boldsymbol{U}_1 \boldsymbol{Q}$ and $\boldsymbol{V}_1 \boldsymbol{Q}$ until necessary to reduce visual clutter. Call $\boldsymbol{W} = \boldsymbol{Q}^T \boldsymbol{U}_1^T \left(\sum_{i=2}^{R} \boldsymbol{A}_i \otimes \boldsymbol{B}_i \right) \boldsymbol{V}_1 \boldsymbol{Q}$. Partitioning $\boldsymbol{W}$ to correspond with the partitioning of $\boldsymbol{K}_1$ above, we get

$$\begin{aligned}\boldsymbol{K} &= \boldsymbol{U}_1\boldsymbol{Q}\left(\begin{bmatrix}\boldsymbol{\Sigma}_{1,k} & \mathbf{0}\\ \mathbf{0} & \boldsymbol{\Sigma}_{1,0}\end{bmatrix} + \begin{bmatrix}\boldsymbol{W}_{1,1} & \boldsymbol{W}_{1,2}\\ \boldsymbol{W}_{2,1} & \boldsymbol{W}_{2,2}\end{bmatrix}\right)\boldsymbol{Q}^T\boldsymbol{V}_1^T \\ &= \boldsymbol{U}_1\boldsymbol{Q}\left(\begin{bmatrix}\boldsymbol{\Sigma}_{1,k}+\boldsymbol{W}_{1,1} & \mathbf{0}\\ \mathbf{0} & \boldsymbol{\Sigma}_{1,0}\end{bmatrix} + \begin{bmatrix}\mathbf{0} & \boldsymbol{W}_{1,2}\\ \boldsymbol{W}_{2,1} & \boldsymbol{W}_{2,2}\end{bmatrix}\right)\boldsymbol{Q}^T\boldsymbol{V}_1^T.\end{aligned}$$

Here, $\boldsymbol{\Sigma}_{1,k} + \boldsymbol{W}_{1,1} = \boldsymbol{T}_k$ as defined in Section 4.1. Incorporating the SVD $\boldsymbol{T}_k = \boldsymbol{U}_t\boldsymbol{\Sigma}_t\boldsymbol{V}_t^T$ and using $\mathbb{1}$ to denote the identity matrix,

$$\begin{aligned}\boldsymbol{K} &= \boldsymbol{U}_1\boldsymbol{Q}\left(\begin{bmatrix}\boldsymbol{U}_t\boldsymbol{\Sigma}_t\boldsymbol{V}_t^T & \mathbf{0}\\ \mathbf{0} & \boldsymbol{\Sigma}_{1,0}\end{bmatrix} + \begin{bmatrix}\mathbf{0} & \boldsymbol{W}_{1,2}\\ \boldsymbol{W}_{2,1} & \boldsymbol{W}_{2,2}\end{bmatrix}\right)\boldsymbol{Q}^T\boldsymbol{V}_1^T \\ &= \boldsymbol{U}_1\boldsymbol{Q}\left(\begin{bmatrix}\boldsymbol{U}_t & \mathbf{0}\\ \mathbf{0} & \mathbb{1}\end{bmatrix}\begin{bmatrix}\boldsymbol{\Sigma}_t & \mathbf{0}\\ \mathbf{0} & \boldsymbol{\Sigma}_{1,0}\end{bmatrix}\begin{bmatrix}\boldsymbol{V}_t^T & \mathbf{0}\\ \mathbf{0} & \mathbb{1}\end{bmatrix} + \begin{bmatrix}\mathbf{0} & \boldsymbol{W}_{1,2}\\ \boldsymbol{W}_{2,1} & \boldsymbol{W}_{2,2}\end{bmatrix}\right)\boldsymbol{Q}^T\boldsymbol{V}_1^T \\ &= \boldsymbol{U}_1\boldsymbol{Q}\begin{bmatrix}\boldsymbol{U}_t & \mathbf{0}\\ \mathbf{0} & \mathbb{1}\end{bmatrix}\left(\begin{bmatrix}\boldsymbol{\Sigma}_t & \mathbf{0}\\ \mathbf{0} & \boldsymbol{\Sigma}_{1,0}\end{bmatrix} + \begin{bmatrix}\mathbf{0} & \boldsymbol{U}_t^T\boldsymbol{W}_{1,2}\\ \boldsymbol{V}_t\boldsymbol{W}_{2,1} & \boldsymbol{W}_{2,2}\end{bmatrix}\right)\begin{bmatrix}\boldsymbol{V}_t^T & \mathbf{0}\\ \mathbf{0} & \mathbb{1}\end{bmatrix}\boldsymbol{Q}^T\boldsymbol{V}_1^T. \quad (6.2)\end{aligned}$$

Recall the partitioning $\boldsymbol{U}_1\boldsymbol{Q} = \begin{bmatrix}\boldsymbol{U}_{1,k} & \boldsymbol{U}_{1,0}\end{bmatrix}$ and likewise with $\boldsymbol{V}_1\boldsymbol{Q}$. We can then re-write the left-hand orthogonal matrix product as

$$\begin{aligned}\boldsymbol{U}_1\boldsymbol{Q}\begin{bmatrix}\boldsymbol{U}_t & 0\\ 0 & \mathbb{1}\end{bmatrix} &= \begin{bmatrix}\boldsymbol{U}_{1,k} & \boldsymbol{U}_{1,0}\end{bmatrix}\begin{bmatrix}\boldsymbol{U}_t & 0\\ 0 & \mathbb{1}\end{bmatrix} \\ &= \begin{bmatrix}\boldsymbol{U}_{1,k}\boldsymbol{U}_t & \boldsymbol{U}_{1,0}\end{bmatrix}.\end{aligned}$$

Similarly, we have that

$$\boldsymbol{V}_1\boldsymbol{Q}\begin{bmatrix}\boldsymbol{V}_t & 0\\ 0 & \mathbb{1}\end{bmatrix} = \begin{bmatrix}\boldsymbol{V}_{1,k}\boldsymbol{V}_t & \boldsymbol{V}_{1,0}\end{bmatrix}.$$

Substituting this into (6.2) above yields

$$\boldsymbol{K} = \begin{bmatrix} \boldsymbol{U}_{1,k}\boldsymbol{U}_t & \boldsymbol{U}_{1,0} \end{bmatrix} \left(\begin{bmatrix} \boldsymbol{\Sigma}_t & 0 \\ 0 & \boldsymbol{\Sigma}_{1,0} \end{bmatrix} + \begin{bmatrix} 0 & \boldsymbol{U}_t^T\boldsymbol{W}_{1,2} \\ \boldsymbol{V}_t\boldsymbol{W}_{2,1} & \boldsymbol{W}_{2,2} \end{bmatrix} \right) \begin{bmatrix} \boldsymbol{V}_t^T\boldsymbol{V}_{1,k}^T \\ \boldsymbol{V}_{1,0}^T \end{bmatrix}.$$

We have arrived at the truncation method. If we truncate the left and right singular vector matrices here to the first k columns (which are $\boldsymbol{U}_{1,k}\boldsymbol{U}_t$ and $\boldsymbol{V}_{1,k}\boldsymbol{V}_t$), and truncate the interior terms accordingly, we exactly have the truncation method described in Chapter 4. However, instead of truncating, we will continue deriving the hybrid method by manipulating the interior terms.

Let $\boldsymbol{D}_w = \text{diag}(\boldsymbol{W}_{2,2})$ and $\boldsymbol{\Sigma}_{\text{diag},0} = \boldsymbol{\Sigma}_{1,0} + \boldsymbol{D}_w$ (this notation is used intentionally, as the values in $\boldsymbol{\Sigma}_{\text{diag},0}$ are singular values from the baseline method, but reordered). Then we can re-write

$$\boldsymbol{K} = \begin{bmatrix} \boldsymbol{U}_{1,k}\boldsymbol{U}_t & \boldsymbol{U}_{1,0} \end{bmatrix} \left(\begin{bmatrix} \boldsymbol{\Sigma}_t & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{\Sigma}_{\text{diag},0} \end{bmatrix} + \begin{bmatrix} \boldsymbol{0} & \boldsymbol{U}_t^T\boldsymbol{W}_{1,2} \\ \boldsymbol{V}_t\boldsymbol{W}_{2,1} & \boldsymbol{W}_{2,2} - \boldsymbol{D}_w \end{bmatrix} \right) \begin{bmatrix} \boldsymbol{V}_t^T\boldsymbol{V}_{1,k}^T \\ \boldsymbol{V}_{1,0}^T \end{bmatrix}.$$

Here, we add approximation by dropping the second term of the parenthetical interior of this factorization. Specifically, we can approximate

$$\begin{aligned} \boldsymbol{K} &\approx \begin{bmatrix} \boldsymbol{U}_{1,k}\boldsymbol{U}_t & \boldsymbol{U}_{1,0} \end{bmatrix} \begin{bmatrix} \boldsymbol{\Sigma}_t & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{\Sigma}_{\text{diag},0} \end{bmatrix} \begin{bmatrix} \boldsymbol{V}_t^T\boldsymbol{V}_{1,k}^T \\ \boldsymbol{V}_{1,0}^T \end{bmatrix} \\ &= \boldsymbol{U}_1\boldsymbol{Q} \begin{bmatrix} \boldsymbol{U}_t & \boldsymbol{0} \\ \boldsymbol{0} & \mathbb{1} \end{bmatrix} \begin{bmatrix} \boldsymbol{\Sigma}_t & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{\Sigma}_{\text{diag},0} \end{bmatrix} \begin{bmatrix} \boldsymbol{V}_t^T & \boldsymbol{0} \\ \boldsymbol{0} & \mathbb{1} \end{bmatrix} \boldsymbol{Q}^T\boldsymbol{V}_1^T. \end{aligned} \tag{6.3}$$

Approximation (6.3) is the hybrid method of approximation using the truncation method. The first k singular values and corresponding vectors are from the truncation method, and the last $N-k$ singular values and vectors are from the baseline method.

6.2 Discussion

6.2.1 Permutation Matrix

On its own, the fact that a permutation matrix $\boldsymbol{Q}$ exists which reorders $\boldsymbol{U}_1$ and $\boldsymbol{V}_1$ so that their columns align with the truncation method is not particularly helpful for those who wish to implement the actual permutation. Here, we detail the construction of this permutation matrix and discuss efficient implementation. We derive the permutation using $\boldsymbol{U}_1$ as an example, but the same arguments apply to $\boldsymbol{V}_1$ and $\boldsymbol{\Sigma}_1$. For those seeking only an explicit formula, see Equation (6.4).

The goal of the permutation matrix $\boldsymbol{Q}$, when applied on the right-hand side of $\boldsymbol{U}_1$, is to move the columns corresponding to a truncation of the matrices that form $\boldsymbol{U}_1$ to the leftmost columns, stably, and to move all of the displaced columns to the right of those, also stably. (A stable sorting retains the original ordering of equivalent values; in this case, there are three equivalent bins: columns included in the truncation, columns displaced by the reordering of the first bin, and columns which are unmoved in the permutation.)

Consider an example in which $\boldsymbol{U}_A, \boldsymbol{U}_B \in \mathbb{R}^{10\times 10}$ where we want to truncate to the first 4 columns of $\boldsymbol{U}_A$ and the first 3 columns of $\boldsymbol{U}_B$ (that is, $\ell = 4$ and $m = 3$). If we form the full Kronecker product $\boldsymbol{U}_1 = \boldsymbol{U}_A \otimes \boldsymbol{U}_B$, the columns that we want to keep are not adjacently located, as shown in Figure 6.1.

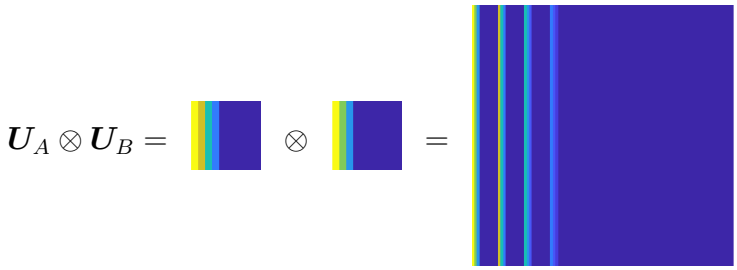


Figure 6.1: Standard Kronecker ordering. If we consider only the first 4 columns of $\boldsymbol{U}_A$ and the first 3 columns of $\boldsymbol{U}_B$, the columns appear as 4 non-adjacent bands of 3 columns each in $\boldsymbol{U}_1$.

$\boldsymbol{U}_1$ is composed of blocks, each of which is $\boldsymbol{U}_B$ scaled by an entry of $\boldsymbol{U}_A$. The Kronecker product always is of a consistent structure. If we truncate by zeroing out the entries we wish to remove, as in Figure 6.1, the resulting sparsity pattern is banded. With $\boldsymbol{U}_A$ truncated to ℓ columns and $\boldsymbol{U}_B$ truncated to m columns, there are ℓ bands of width m columns each. The start of a band is every n columns. So in this example, we have $\ell = 4$ bands each of width $m = 3$, occurring every $n = 10$ columns.

The permutation matrix $\boldsymbol{Q}$ moves these bands to the left, preserving the relative ordering of the columns within the bands. Figure 6.2 illustrates this behavior.

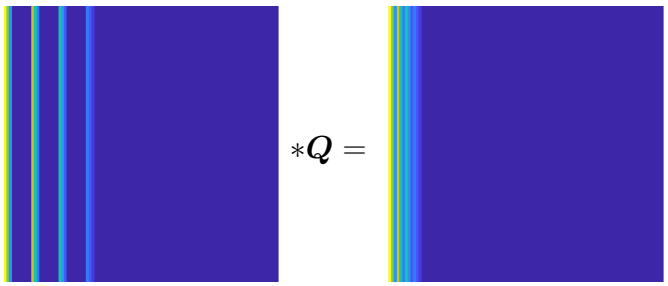


Figure 6.2: Permuted Kronecker order. The result of permuting the non-discarded columns in $\boldsymbol{U}_1$ using $\boldsymbol{Q}$. The permutation $\boldsymbol{Q}$ moves the non-discarded columns to the left.

What should $\boldsymbol{Q}$ depend on? Recall, we need to move ℓ bands of size m occurring every n columns. So $\boldsymbol{Q}$ depends on ℓ, m, and n. These fully define the truncation banding pattern as long as both $\boldsymbol{U}_A$ and $\boldsymbol{U}_B$ are size $n \times n$.

We can now start to piece together the structure of $\boldsymbol{Q}$. This discussion uses one-indexing, and can be converted to zero-indexing by removing the additions and subtractions of 1 throughout. The first band in $\boldsymbol{U}_1$ is already in place, so $\boldsymbol{Q}$ starts with m ones along its diagonal. The second band should be moved into place after the first, and occurs at column indices $n+1$ through $n+m$ in $\boldsymbol{U}_1$. The third band occurs at column indices $2n+1$ through $2n+m$ in $\boldsymbol{U}_1$. In general, the j^{th} band occurs at column indices $(j-1)n+1$ through $(j-1)n+m$, with j between 1 and ℓ, inclusive, in the full Kronecker product matrix. Further, the i^{th} band starts at column $(i-1)m+1$ in the permuted matrix $\boldsymbol{U}_1\boldsymbol{Q}$. The second band in our example is reordered into place by columns $m+1=4$ through $2m=6$ in $\boldsymbol{Q}$. The first entry in the second band needs to be moved from column $n+1=11$ of $\boldsymbol{U}_1$, so there is a 1 in column $m+1=4$ and row $n+1=11$ of $\boldsymbol{Q}$. Similarly, the rest of the band puts a 1 at column 5 row 12 and column 6 row 13. The third band starts with a 1 at column $2m+1=7$ and row $2n+1=21$. This pattern continues for the remaining elements. So we have built up the following structure, where no entry indicates a zero, $\vdots$ indicates seven rows of zeros, and $\vdots$ indicates $n(n-\ell)=60$ rows of zeros:

$$\text{Columns 1 through 12 of } \boldsymbol{Q} = \begin{bmatrix} 1 & & & & & & & & & & & \\ & 1 & & & & & & & & & & \\ & & 1 & & & & & & & & & \\ & & & \vdots & & & & & & & & \\ & & & 1 & & & & & & & & \\ & & & & 1 & & & & & & & \\ & & & & & 1 & & & & & & \\ & & & & & \vdots & & & & & & \\ & & & & & & 1 & & & & & \\ & & & & & & & 1 & & & & \\ & & & & & & & & 1 & & & \\ & & & & & & & & \vdots & & & \\ & & & & & & & & & 1 & & \\ & & & & & & & & & & 1 & \\ & & & & & & & & & & & 1 \\ & & & & & & & & & & & \vdots \\ & & & & & & & & & & & \vdots \end{bmatrix}.$$

This moves the bands that are kept by the truncation into place on the left. Next

is to move the bands that are displaced by truncation to the right. For our truncation to ℓ columns in $\boldsymbol{U}_A$ and m columns in $\boldsymbol{U}_B$, we have a total of ℓm columns kept. That means that the first displaced column, which starts at index $m+1$ in $\boldsymbol{U}_1$, should move to index $\ell m+1$. The displaced columns come in bands in $\boldsymbol{U}_1$ like the columns kept by truncation do: for every band of m columns kept by truncation, there are $n-m$ columns to the right that are not kept. So the first band of removed columns moves from column indices $m+1$ through n to indices $\ell m+1$ through $\ell m+n-m$. In our example, this means columns $m+1=4$ through $n=10$ move to columns $\ell m+1=13$ through $\ell m+n-m=19$. The bands in $\boldsymbol{U}_1$ which get displaced are at indices $(j-1)n+m+1$ through jn for integers j between 1 and ℓ, inclusive. The jth band is moved to column positions $\ell m+(j-1)(n-m)+1$ through $\ell m+j(n-m)$ in $\boldsymbol{U}_1\boldsymbol{Q}$.

For our example, the first two bands displaced in $\boldsymbol{Q}$ have the following form, where $\cdot$ represents $m=3$ rows of zeros and $\mathbf{:}$ represents 80 rows of zeros:

$$\text{Columns 13 through 26 of } \boldsymbol{Q} = \begin{bmatrix} \cdot & & & & & & & & & & & & & \\ 1 & & & & & & & & & & & & & \\ & 1 & & & & & & & & & & & & \\ & & 1 & & & & & & & & & & & \\ & & & 1 & & & & & & & & & & \\ & & & & 1 & & & & & & & & & \\ & & & & & 1 & & & & & & & & \\ & & & & & & 1 & & & & & & & \\ & & & & & & & 1 & & & & & & \\ & & & & & & & \cdot & & & & & & \\ & & & & & & & & 1 & & & & & \\ & & & & & & & & & 1 & & & & \\ & & & & & & & & & & 1 & & & \\ & & & & & & & & & & & 1 & & \\ & & & & & & & & & & & & 1 & \\ & & & & & & & & & & & & & 1 \\ & & & & & & & & & & & & & \mathbf{:} \end{bmatrix}.$$

There are two more bands in columns 27 through 40 with a similar pattern.

The last $n(n-\ell)=60$ columns have no permutation; the ones in $\boldsymbol{Q}$ are along the diagonal.

This gives us the full structure of $\boldsymbol{Q}$ for this example. Figure 6.3 shows this full

structure of $\boldsymbol{Q}$.

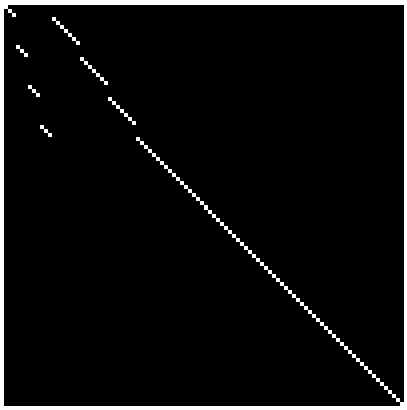


Figure 6.3: The resulting permutation matrix with $\ell = 4$, $m = 3$, and $n = 10$. Here, white represents 1 and black represents 0.

How can we generalize the structure succinctly? In general, for given ℓ, m, and n,

$$\boldsymbol{Q}^T = \begin{pmatrix} \begin{bmatrix} \mathbb{1}_\ell & \mathbf{0} \end{bmatrix}_{\ell \times n} \otimes \begin{bmatrix} \mathbb{1}_m & \mathbf{0} \end{bmatrix}_{m \times n} \\ \\ \begin{bmatrix} \mathbb{1}_\ell & \mathbf{0} \end{bmatrix}_{\ell \times n} \otimes \begin{bmatrix} \mathbf{0} & \mathbb{1}_{n-m} \end{bmatrix}_{(n-m) \times n} \\ \\ \begin{bmatrix} \mathbf{0} & (\mathbb{1}_n \otimes \mathbb{1}_{n-\ell}) \end{bmatrix}_{n(n-\ell) \times n^2} \end{pmatrix} \tag{6.4}$$

We have written $\boldsymbol{Q}^T$ instead of $\boldsymbol{Q}$ because separating the terms by rows is visually clearer. The first row block (column block in $\boldsymbol{Q}$) corresponds to the columns in $\boldsymbol{U}_1$ kept by truncation. The second row block corresponds to the displaced and truncated columns. The final block corresponds to the truncated but not displaced columns of $\boldsymbol{U}_1$. As with the previous methods, we maintain an efficient representation of this matrix rather than multiplying it explicitly while forming the factorization.

6.2.2 Efficacy

Now that we have the permutation clarified, we return to assessing the hybrid method overall. Unsurprisingly, the hybrid method works as well as the methods that are hybridized to make it. Both the truncation method and reordering method have good accuracy for the largest singular values, although the reordering method generally has higher accuracy than the truncation method. For matrices where $\boldsymbol{K}_1$ approximates $\boldsymbol{K}$ well, the baseline method has consistent and reasonable accuracy throughout. By combining the high accuracy of the largest singular values with a stable accuracy for the smallest singular values, we get the strengths of both methods with fewer drawbacks.

However, if the baseline method is very inaccurate because $\boldsymbol{K}_1$ is a bad approximation of $\boldsymbol{K}$, then the hybrid method combining, for example, the reordering and baseline methods will not help much compared to using the reordering method alone. The hybrid method works primarily on a truncation method (the standard truncation method in Chapter 4 or the reordered truncation method in Chapter 5) plus the baseline. Although it is possible to do a hybrid of the reordering and truncation method, it makes little sense: both are less accurate for the smallest singular values, and both have significantly higher computational cost than the baseline for computing large numbers of singular values. The upshot is that the accuracy of the smallest singular values is limited by the baseline's accuracy.

One modification to the hybrid method can increase accuracy, especially for the standard truncation method: we can compute the truncated method portion of the singular values and vectors, then truncate that factorization down further. For example, we may choose to compute to $k = 100$ with the truncation method. Instead of using those 100 values, we can truncate down further to an effective $\hat{k} = 67 < k$ (which happens to be prime). This cuts off some of the less-accurate "tail" singular values and vectors computed by the truncation method. Because the baseline already

computes approximations of all singular values and vectors, this does not incur extra computation in that regard. And because the interior matrices do not have Kronecker structure in general, we can really do the truncation. However, truncating the interior changes the structure of the permutation matrix $\boldsymbol{Q}$.

Essentially, to create the new permutation matrix we keep the first $\hat{k}$ columns, move over the necessary displaced band columns, and the rest is the identity. Using the previous example, we had $n = 10$, $\ell = 4$, and $m = 3$, so $k = \ell m = 12$. Suppose we wish to truncate this to $\hat{k} = 5$ entries. In $\boldsymbol{Q}$, column 5 has a 1 at row 12. In the new matrix $\widehat{\boldsymbol{Q}}$, the first 5 columns are the same as in $\boldsymbol{Q}$. Then, columns $k + 1 = 13$ through $k + \left(\left\lceil \frac{\hat{k}}{m} \right\rceil - 1\right)(n - m) = 19$ of $\boldsymbol{Q}$ are placed in columns $\hat{k} + 1 = 6$ through $\hat{k} + \left(\left\lceil \frac{\hat{k}}{m} \right\rceil - 1\right)(n - m) = 12$ of $\widehat{\boldsymbol{Q}}$. Here, $\left\lceil \frac{\hat{k}}{m} \right\rceil$ is the number of partial bands kept originally. We subtract one to get the number of relevant displaced bands removed by original truncation, and $(n - m)$ is the size of those displaced bands. The remaining columns in $\widehat{\boldsymbol{Q}}$ are the identity. This example is shown in Figure 6.4.

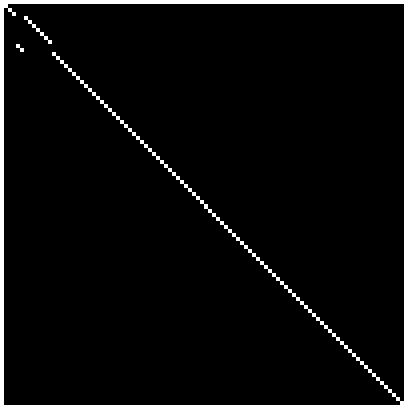


Figure 6.4: Structure of $\widehat{\boldsymbol{Q}}$. This figure shows the structure of the permutation matrix for the extra-truncated hybrid method with $n = 10$, $\ell = 3$, $m = 4$, and $\hat{k} = 5$. White represents a 1, and black represents a zero.

Re-adjusting $\boldsymbol{Q}$ into $\widehat{\boldsymbol{Q}}$ is not significantly costly compared to computing the approximate TSVD. Even more broadly, the steps to combine two methods (typically, one of the reordering or truncation method with the baseline method) into a hybrid

method is not costly compared to computing the separate TSVDs. With careful implementation, the hybrid method is also efficient to use. The outer singular vector matrices remain in Kronecker product format, so they are cheap to multiply with. Efficient implementations of the permutation can require as little as $\hat{k}$ operations to apply. Then the multiplications of the remaining interior terms of the hybrid approximation, when implemented efficiently, incur the cost of the truncated method's multiplications plus scaling the remaining elements by a diagonal. The multiplication cost is therefore not prohibitive.

6.3 Time Complexity

The cost of computing the hybrid method is the sum of the complexities of the baseline method, chosen truncation-based method, and cost to create the permutation matrix $\boldsymbol{Q}$. The permutation matrix is less costly to construct (at $O(N) = O(n^2)$ depending on implementation) than the approximation from the baseline method, so it does not affect the overall time complexity. For general $\boldsymbol{K}$, where computing the decomposition (2.4) takes $O(\mathcal{T})$ time, the hybrid using the standard truncation method then has a cost of $O(\mathcal{T}) + O(n^3 + (l+m)n^2r + k^2r + k^3) + O(n^3r) = O(\mathcal{T} + n^3r + k^2r + k^3)$. This is the same as the complexity of the hybrid reordering method: $O(\mathcal{T}) + O(n^3r + k^2r + k^3) + O(n^3r) = O(\mathcal{T} + n^3r + k^2r + k^3)$. The standard truncation method has a slightly better time complexity, therefore, than its hybrid. On the other hand, the reordering method has the same time complexity as its hybrid.

6.4 Performance

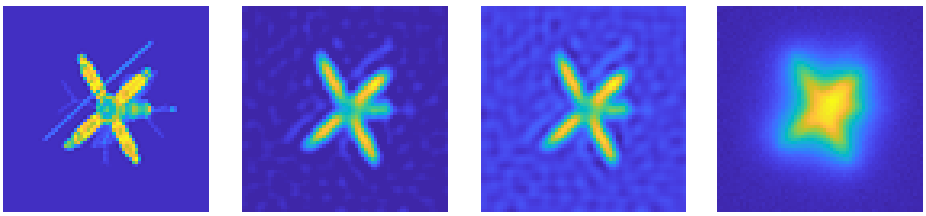
In this section we explore the performance of the hybrid method in a variety of experimental settings. Specifically, we look at the performance of a hybrid of the truncation and baseline methods. Because the hybrid method combines these two

methods, its performance is expected to be as good as both methods, but it also suffers from any limitations shared between the two.

6.4.1 Direct Reconstruction

The hybrid method can have varied performance, typically middling between the performance of the truncation and baseline methods. One methodological detail significantly impacts all of the reconstructions. Each reconstruction uses a truncated SVD. To compute the truncated hybrid, the truncation method was computed to the stated truncation index k for all examples, and the baseline method was computed completely, with all singular values and vectors. To determine the singular values and vectors kept in the hybrid method, the largest k singular values were selected from the hybrid. In cases where the singular values from the truncation method gave significant underestimates, this typically meant that the baseline method dominated the reconstruction.

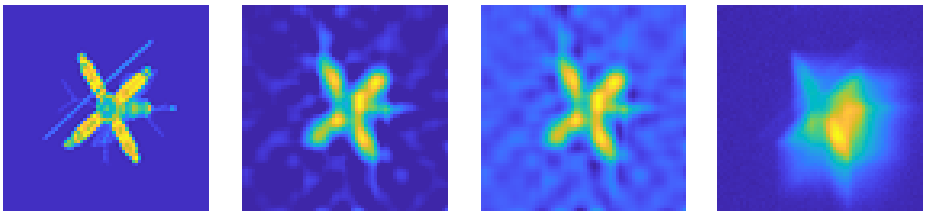
Both the truncation method and baseline method perform well for the Satellite Example, and combining the two in the hybrid method unsurprisingly gives a good reconstruction. This is shown in Figure 6.5. The PSNR for the hybrid method is 46.8 dB, which nearly identical to that of the baseline and truncation method with have a PSNR of 46.9 dB, and very slightly higher than the PSNR of the reordering method at 45.1 dB.



True $\boldsymbol{x}$ Restored $\boldsymbol{x} > 0$ Restored $\boldsymbol{x}$ Blurred $\boldsymbol{d}$

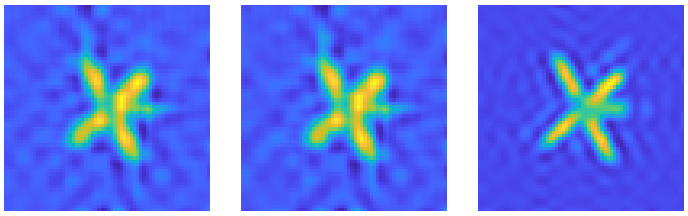
Figure 6.5: Hybrid Satellite Example Restoration. The hybrid method method yields a good reconstruction for the Satellite Example, following suit of the three methods discussed previously.

For the Grain Example, the hybrid result is nearly indistinguishable from the baseline method reconstruction. Only the noise pattern in the image differs between the two. Recall that a small truncation index of $k = 400$, split into $\ell = m = 20$ for the truncation method, is used. Further, the decay of the singular values for $\boldsymbol{K}$ in this example is slower than previous methods. The truncation method underestimates the singular values, and the contribution of the baseline method ends up dominating the reconstruction as a result. This is shown in Figures 6.6 and 6.7. The PSNR for this example is 42.6 dB, comparable to that of the baseline (42.8 dB) and reordering (42.7 dB) methods, but less than the truncation method PSNR of 46.2 dB.



True $\boldsymbol{x}$ Restored $\boldsymbol{x} > 0$ Restored $\boldsymbol{x}$ Blurred $\boldsymbol{d}$

Figure 6.6: Hybrid Grain Example Restoration. The singular values computed by the baseline method dominate those computed by the truncation method, causing the restoration to appear nearly identical to the baseline restoration.

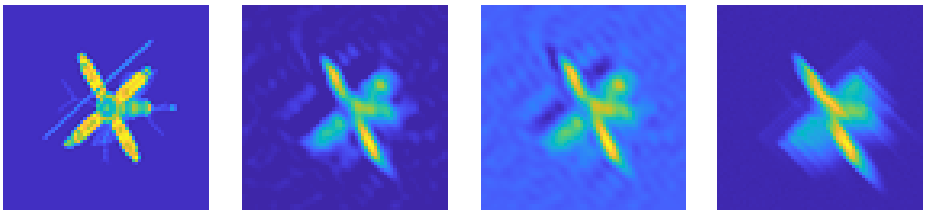


Baseline Hybrid Truncated

Figure 6.7: Grain Example Hybrid Comparison. The singular values from the baseline method dominate the hybrid method, causing the restoration to appear much more like the restoration from the baseline method than the truncation method restoration. In this case, the hybrid has less fine details than the truncation method.

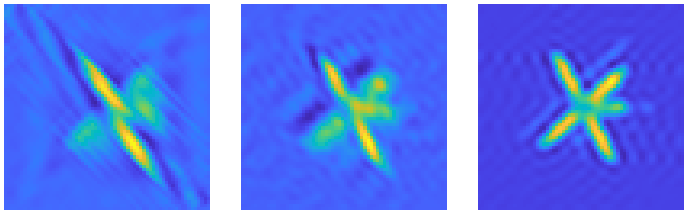
The final example is the Motion Example. For this example, the baseline gave a very poor reconstruction, but the truncation method gave a great reconstruction.

Further, the truncation method gave higher estimates for the largest singular values, and lower estimates for the smaller singular values. We can then expect the hybrid reconstruction to be worse than the truncation method, but much better than the baseline reconstruction. Indeed, this is what we see in the comparison in Figure 6.9. The reconstruction for just the hybrid method is shown in Figure 6.8. The PSNR of this reconstruction is 40.1 dB, just above the baseline method PSNR of 39.0 dB, and significantly below the PSNR for the truncation (47.6 dB) and reordering (47.7 dB) methods.



True $\boldsymbol{x}$ Restored $\boldsymbol{x} > 0$ Restored $\boldsymbol{x}$ Blurred $\boldsymbol{d}$

Figure 6.8: Hybrid Motion Example Restoration. For the motion example, the hybrid method acts as a true hybrid. The result is a moderate-to-poor reconstruction with some fine details still appearing.



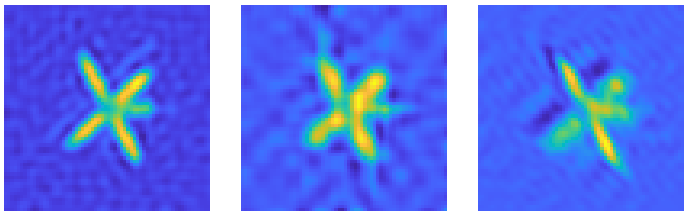
Baseline Hybrid Truncated

Figure 6.9: Motion Example Hybrid Comparison. The singular values from the truncation method dominate for the largest indices, but singular values from the baseline method dominate for the smallest indices. The result is a middling restoration that is better than the baseline method and worse than the truncation method.

These reconstructions, in accordance with the patterns from the previous chapters, show the performance of the hybrid method when truncated significantly. Its performance is more useful when it is not truncated so aggressively, and the corresponding

reconstructions (provided that they are properly regularized) are more useful. For these examples, it is hard to give a much better reconstruction than the truncation method provides.

Figure 6.10 visually summarizes the results for the three restoration experiments in this section.



Satellite Grain Motion

Figure 6.10: Hybrid Method Restoration Summary. This summary image shows the restored images $\boldsymbol{x}$ computed using the hybrid method (6.3) for each restoration example.

6.4.2 Approximated Singular Values

The hybrid method (6.3) enables more accurate singular value computation by exploiting the benefits of the baseline and truncation methods. The truncation method tends to accurately estimate the largest singular values, whereas the baseline can accurately estimate lower singular values for certain problems. In these examples, we use $\ell = m = 8$ to compute the TSVD approximation with the truncation method, yielding 64 approximate singular values and vectors. The remaining 36 singular values and vectors come from the baseline method.

In the Singular Value Atmospheric Example, the hybrid method gives a good estimate of the highest singular values. As the index increases, the singular values are estimated more poorly, until the final values have an oscillatory but on average level error varying around 10^{-2}. This is shown in Figure 6.11.

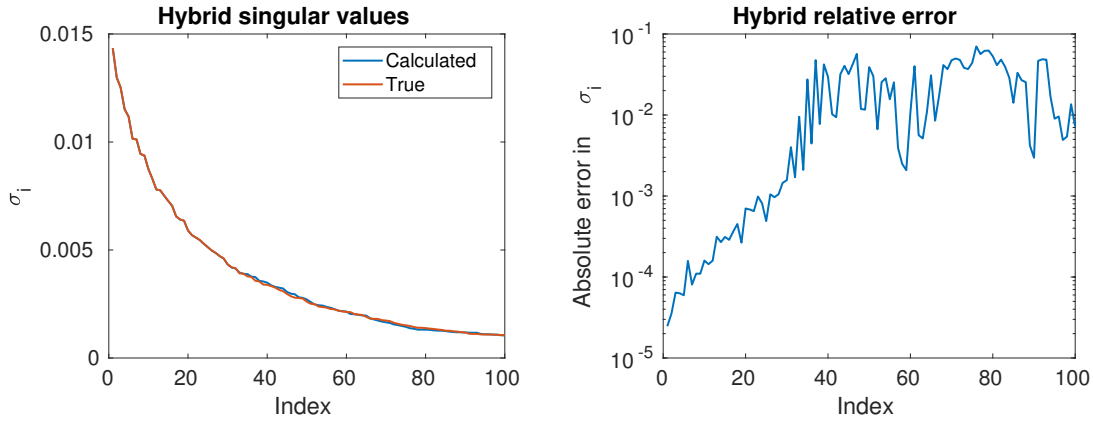


Figure 6.11: Hybrid Atmospheric Blur Singular Values. The hybrid method singular values combine the accurate beginning from the truncation method with the less-inaccurate tail of the baseline method to produce a better approximation than using one method alone.

This error is slightly higher than the truncated method with $\ell = m = 10$ for the highest values, but lower than both the baseline and truncation methods for the smallest values. A full comparison between all methods for the error in singular value computation is given in Appendix B.

The more challenging Singular Value Motion Example yields an unsurprisingly worse result. The error from the hybrid method is close to that of the truncation method, although it does not get as high for the lowest indices which are computed using the baseline method. However, since a lower ℓ and m are used in the truncation calculation, the error is slightly higher for the hybrid method than the truncation method on the largest singular values. The computed singular values and error for the hybrid method is shown in Figure 6.12. As with the Singular Value Atmospheric Blur Example, a comparison between all methods is given in Appendix B.

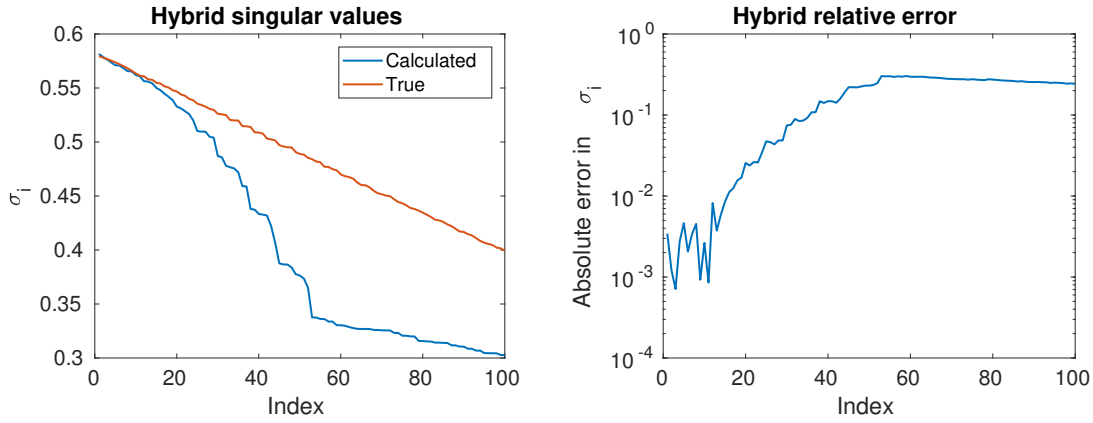


Figure 6.12: Hybrid Motion Singular Values. The hybrid method again combines the better estimates from the truncation and baseline methods. Although the result is not excellent, it is better for the largest values than the baseline method and better for the smallest values than the truncation method.

One of the primary strengths of the hybrid method is that it combines the benefits of the truncation (or reordering) method with those of the baseline method to produce more accurately computed singular values. These examples demonstrate that strength.

6.4.3 Preconditioning

The cost of computing the hybrid method, however, is not a strength. We expect it to be slower than both the baseline and truncation method to construct. Depending on how the singular values align between the two methods, it may allow a preconditioned system to converge faster than the baseline and truncation methods, but it may also converge in more iterations. Certainly, we expect that it will take longer to complete a single iteration than the baseline method due to overhead in the extra multiplications required.

When we tested it as a preconditioner for the conjugate gradient least squares method (CGLS), we found it to be slower in all regards than the truncation and baseline methods. It took more iterations (19 as opposed to 16 and 7 for the baseline

and truncation methods, respectively) and more total time (1.85 seconds). Constructing the preconditioner and completing the iterations were not faster than the two methods it hybridized. The difference between the hybrid and truncation method setup times was small enough so as to not be conclusive, with a difference of less than 0.10 seconds. Appendix C shows a detailed comparison of the timings of all methods for reference.

The hybrid preconditioner did, however, beat using no preconditioner. The hybrid method took a total of 1.85 seconds, split between 1.19 seconds to compute the factorization and 0.657 seconds to complete iterations, as opposed to the total 6.04 seconds with no preconditioner. Table 6.1 below summarizes the timing results for the hybrid method compared to not using a preconditioner.

	No preconditioner	Hybrid
Iterations	345	19
Setup time (sec)	0.0	1.19
Calculate time (sec)	6.04	0.657
Total time (sec)	6.04	1.85

Table 6.1: Hybrid Method Timings for PCGLS. A summary of the timing results for the PCLGS problem, with the hybrid method used as a preconditioner, is shown. The hybrid method saves time and takes fewer iterations compared to an unpreconditioned system.

Because of its comparatively lackluster performance as a preconditioner relative to the other methods, we cannot recommend the hybrid preconditioner for a first try at preconditioning. However, it does significantly improve convergence compared to not using a preconditioner.

6.5 Summary

The hybrid method combines the best aspects of the truncation methods with the best aspect of the baseline method. It uses the largest singular values and vectors from

the truncation methods, while still cheaply computing reliably accurate lower singular values and vectors. Adding extra truncation to the truncation method portion of the approximate TSVD can further improve accuracy by cutting off the inaccurately-estimated smallest singular values (and corresponding vectors) of the truncation-based methods. However, when the baseline method fails because $\boldsymbol{K}_1 \not\approx \boldsymbol{K}$, it makes little sense to use the hybrid method. Problem knowledge is therefore required in deciding when to use the method. But, in the situations where it is applicable, the hybrid method gives improved accuracy without a significant sacrifice to speed.

Chapter 7

Comparison to Related Methods

The methods presented thus far are not the only TSVD approximation methods. There are three particularly popular methods for computing approximate TSVDs and related factorizations which this chapter focuses on: Fourier transforms, Golub-Kahan-Lanczos bidiagonalization, and randomized methods. Each method has different strengths, and in some cases they would be a better choice than the methods discussed here. If computing the Kronecker product decomposition (2.4) or approximation (2.5) is prohibitive, then the methods proposed in Chapters 3-6 cannot be applied. If directly tunable accuracy is desired, the matrix is too big to fit into memory, or parallel architectures are available, randomized methods are a good choice [24]. For situations where high accuracy is needed but speed is less important, Golub-Kahan-Lanczos bidiagonalization is useful [21, 33]. In cases where the matrix $\boldsymbol{K}$ is block circulant with circulant blocks, Fourier transforms have minimal storage costs and are highly accurate (there is a multitude of work on Fourier Transforms, including [1, 11, 25]). Sections 7.1-7.3 explain in detail the nuances of when to use these methods.

However, in some cases the matrix $\boldsymbol{K}$ is extremely large and has structure that allows a cheap approximate decomposition (2.5), without the need for high accuracy.

For example, preconditioners that are cheap and only moderately accurate can accelerate the convergence of iterative methods [21]. And in some imaging applications that are particularly ill-posed, an exact decomposition provides little visual benefit over moderately accurate approximations.

These and other benefits make our proposed and the baseline methods relevant in some cases, but not all cases. The methods discussed here are popular for good reason, and are useful in a variety of situations.

7.1 Fourier Transforms

The first common method of approximating a matrix factorization with similar structure to a TSVD is Fourier transforms. Fourier transforms are used in many disciplines, notably including signal processing, due to their very low computation cost and exact accuracy (in exact arithmetic) [1, 25]. Fourier transforms change the basis of a signal from the spatial domain to the Fourier domain. The Fourier basis builds up a signal based on its frequency components. This is a natural transformation in image processing, where images can be intuitively represented as a sum of their frequency components [1, 25].

The Fourier transformation is defined as follows. Given a circulant or block circulant with circulant block matrix $\boldsymbol{K}$ and using * to denote the conjugate transpose, the discrete Fourier transform matrix is defined as the matrix $\boldsymbol{F}$ such that

$$\boldsymbol{F}\boldsymbol{K}\boldsymbol{F}^* = \boldsymbol{\Lambda}$$

where $\boldsymbol{\Lambda}$ is a diagonal matrix containing the eigenvalues of $\boldsymbol{K}$ and $\boldsymbol{F}^*\boldsymbol{F} = \boldsymbol{F}\boldsymbol{F}^* = \mathbb{1}$.

With $\phi = e^{-2\pi i/N}$, where $i = \sqrt{-1}$, the entries of $\boldsymbol{F}$ are given by [21]

$$\boldsymbol{F} = \frac{1}{\sqrt{N}} \begin{bmatrix} 1 & 1 & ... & \dots & 1 \\ 1 & \phi & \phi^2 & \dots & \phi^{(N-1)} \\ 1 & \phi^2 & \phi^4 & \dots & \phi^{2(N-1)} \\ \vdots & & & & \vdots \\ 1 & \phi^{(N-1)} & \phi^{2(N-1)} & \dots & \phi^{(N-1)^2} \end{bmatrix}.$$

Matrix-matrix products involving the Fourier matrices take only $O(N^2 \log(N^2)) = O(n^4 \log(n))$ operations to compute by using fast Fourier transforms. This factorization therefore takes $O(n^4 \log(n))$ operations to compute. Unlike a TSVD, the Fourier Transformation does not produce components in order of importance, so we cannot truncate to reduce costs. Once the transform is constructed, matrix-vector multiplication take $O(n^2 \log(n))$ operations. With our methods, the multiplications take $O(n^3 + k^2)$ operations, so a Fourier transform is faster to use once constructed.

This factorization is exact within machine precision. There are two reasons not to use it for every problem. The first is the time complexity to construct the transform. Fourier transforms are much cheaper than standard matrix factorization that require $O(N^3) = O(n^6)$ computations, but slower than our proposed factorizations for limited k, which require $O(n^3 r + k^2 r + k^3)$ operations to construct. This may be offset by reduced time for matrix-vector multiplications as previously mentioned, but the initial cost can be prohibitive. In addition, Fourier transforms have an additional restriction: as described, the factorization works only on circulant (or block circulant with circulant block) matrices. There is an extension to Toeplitz matrices, but it increases the already-large matrix size by a factor of 4. Circulant (and block-circulant with circulant block) matrices in the image deblurring problem correspond to periodic boundary conditions. For non-periodic matrices, performance degrades.

The complexities we have discussed so far are time complexity, but one of the

major benefits of our proposed method is the low storage costs. As discussed in Section 5.1, the storage cost of the reordering method, which has the highest storage cost of all of the non-hybrid methods, is $O(n^2 + k^2)$. (For comparison, the baseline method has storage cost $O(n^2)$, the truncation method has cost $O(n(\ell+m)+k^2)$, and the hybrid method, when implemented efficiently, has $O(n^2+k^2)$ memory complexity.) By comparison, decomposition using a Fourier transform does not require forming the unitary matrices $\boldsymbol{F}$, so the only values stored are the N eigenvalues in $\boldsymbol{\Lambda}$, making the storage cost $O(n^2)$. It is often the case that $k > n$, so this is often lower than the storage complexity for our method. And in practice, the Fourier method only requires storage of one set of n^2 elements, whereas our reordering method requires storage of four matrices of size n^2, three matrices of size k^2, and a permutation of length n^2. So our storage costs are many times larger.

Fourier transforms are highly accurate, cheap to store, and fast to use. For problems with circulant matrices, Fourier transforms are a good choice. However, for high accuracy on non-circulant $\boldsymbol{K}$, our methods provide an alternative that is faster to compute but slower to use and which requires slightly more storage.

7.2 Golub-Kahan-Lanczos Bidiagonalization

Golub-Kahan-Lanczos bidiagonalization, herein Lanczos bidiagonalization, is a commonly used method for computing the TSVD of a matrix. Lanczos bidiagonalization is the process of reducing a matrix into a bidiagonal matrix via orthogonal transformations. The result of a full application of Lanczos bidiagonalization is the factorization

$$\boldsymbol{K} = \boldsymbol{U}\boldsymbol{B}\boldsymbol{V}^T$$

where $\boldsymbol{U}, \boldsymbol{V} \in \mathbb{R}^{N \times N}$ with $\boldsymbol{U}^T\boldsymbol{U} = \mathbb{1}$ and $\boldsymbol{V}^T\boldsymbol{V} = \mathbb{1}$, and $\boldsymbol{B}$ is lower bidiagonal with [33]

$$\boldsymbol{B} = \begin{bmatrix} \alpha_1 & & & & \\ \beta_1 & \alpha_2 & & & \\ & \beta_2 & \alpha_3 & & \\ & & \ddots & \ddots & \\ & & & \beta_{n-1} & \alpha_n \end{bmatrix}.$$

Typically, the process is run only to k iterations, which computes $\boldsymbol{U}$ and $\boldsymbol{V}$ to their first k columns $\boldsymbol{U}_k$ and $\boldsymbol{V}_k$, and the bidiagonal matrix up to k rows and columns yielding $\boldsymbol{B}_k$. Then the SVD of the bidiagonal matrix $\boldsymbol{B}_k$ is computed cheaply to complete the SVD calculation [19].

During the course of computing the columns of $\boldsymbol{U}_k$ and $\boldsymbol{V}_k$, orthogonality is lost due to errors resulting from division by small numbers [21]. To remedy this, the columns must be reorthogonalized fully or in part. Failure to reorthogonalize the columns can result in extremly inaccurate results.

In the Lanczos bidiagonalization algorithm without reorthogonalization, the primary cost is in computing matrix-vector multiplications using $\boldsymbol{K}$ and $\boldsymbol{K}^T$. When the matrices are fully formed, this has a cost of $O(N^2)$ per iteration, or $O(N^2k) = O(n^4k)$ total. Costs can be lower per iteration for sparse or structured $\boldsymbol{K}$. However, with full reorthogonalization the most recently-computed Lanczos vector must be updated based on each previously-computed Lanczos vector, which has a cost of $O(Nk^2) = O(n^2k^2)$ regardless of the structure of $\boldsymbol{K}$. This can be expensive, and dominates the cost if the matrix-vector products are especially cheap. The Lanczos bidiaigonalization procedure is more expensive than the steps to convert the bidiagonal matrix to diagonal form and complete an SVD factorization. The cost for the algorithm is therefore $O(n^4k + n^2k^2)$ for general matrices or, if the cost of a matrix-vector multiplication is C, $O(Ck + n^2k^2)$ [46]. This is higher than our reordering method's cost

of $O(n^3r + k^2r + k^3)$ for typical choices of n and k.

Perhaps more crucially, there is a significant difference in storage costs between our proposed method and the Lanczos bidiagonalization routine. Storing $\boldsymbol{K}$ in Kronecker product summation decomposition form (2.5) has a baseline cost of $O(n^2r)$ compared to the full matrix storage cost of $O(n^4)$. The Lanczos process creates k vectors of length $N = n^2$, for a cost of $O(n^2k)$. Typically $k > r$, so this increases storage costs compared to the Kronecker summation approximation of the original matrix. And compared to the cost of $O(n^2 + k^2)$ for storing the reordering method, the Lanczos method's storage costs can be prohibitive for very large matrices.

However, the Lanczos bidiagonalization routine for computing an SVD is the clear winner with respect to accuracy. Errors of computed singular values are so low as to be negligible, although they can be adjusted indirectly by reducing accuracy of reorthogonalization in many implementations of the algorithm. The following subsection (7.2.1) details the results of running a standard software for computing a Lanczos-based TSVD with standard software versus our algorithm.

7.2.1 Experimental Comparison

We compare our implemented methods to a popular Lanczos bidiagonalization method `lansvd` implemented in the package PROPACK [33]. The `lansvd` method allows for several levels of customization, including reducing the amount of reorthogonalization, computing truncated SVDs as opposed to full SVDs, and using function calls to compute matrix-vector (and matrix-transpose-vector) products instead of the full form of the matrix. Because it can be cheaper to use a function call form of a matrix-vector product, we use that form of the `lansvd` method to test against, with different levels of reorthogonalization.

We ran two tests against the `lansvd` method to determine relative speed and accuracy compared to our methods. For the speed test, we varied the truncation index

k for both methods to show how the timing scaled with rank. We used the reordering method, which tends to be the slowest of our proposed non-hybrid methods. We used a new, speckled PSF, shown in Figure 7.2, which is available in the upcoming IRTools package [18] using the test problem `PRblurspeckle`. For the reordering method, we used a full $r = R = 64$ terms in the Kronecker summation decomposition, which meant the method was slowed as much as possible. For the `lansvd` method, we tested both with full reorthogonalization and with reorthogonalization turned off. Figure 7.1 shows the results of these timing runs.

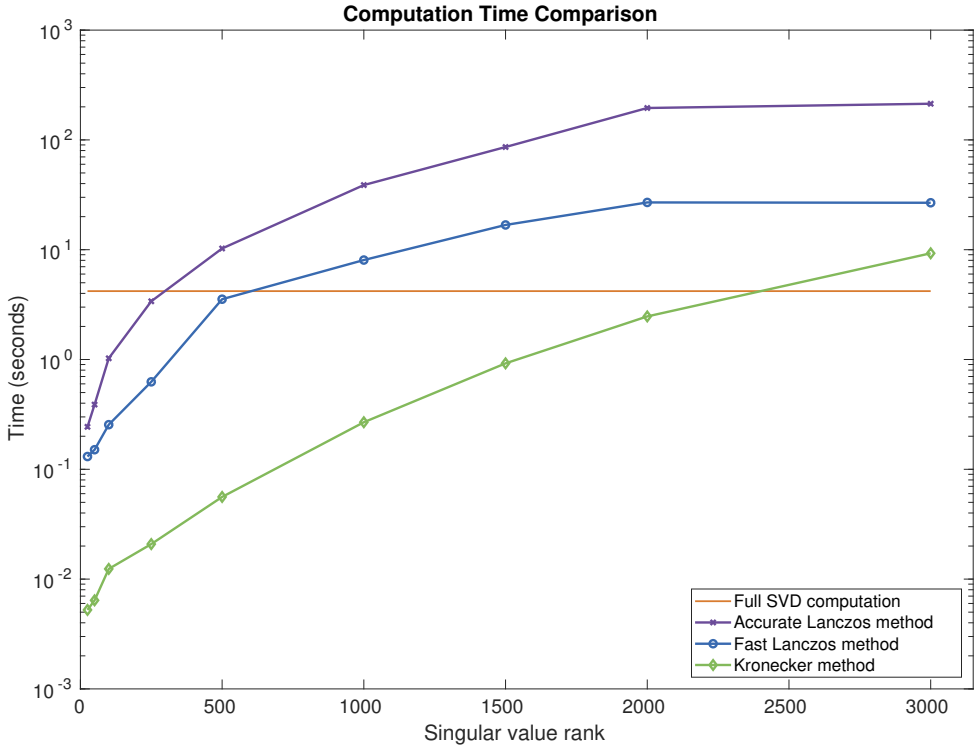


Figure 7.1: Lanczos versus Reordering Method Timings. Timings for various truncation indices k to compute a TSVD using `lansvd` versus our Kronecker-based reordering method. The reordering method is faster for all truncation indexes, even when `lansvd` has reorthogonalization turned off. The time taken to compute a full SVD using MATLAB's `svd` method is shown for comparison.

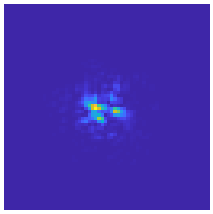


Figure 7.2: Speckled PSF. The PSF used to compare the Lanczos bidiagonalization SVD method against our Kronecker reordering method.

Clearly, the Kronecker reordering method produces results much more quickly than the `lansvd` method. Although `lansvd` completes significantly faster without reorthogonalization than with reorthogonalization, it still is never as fast as the reordering method. Both methods eventually become slower than computing a full SVD with the MATLAB command `svd`; this is expected, due to overhead in the algorithms.

When it comes to accuracy, though, `lansvd` easily wins. The singular values constructed using `lansvd` with reorthogonalization were near machine precision (around 10^{-15} relative error) of the "true" values computed using MATLAB's `svd` command, whereas the relative error of the singular values computed using the reordering method had around 10^{-6} relative error. With reorthogonalization turned off, the Lanczos-based method produced incredibly inaccurate results.

As expected, the primary strength of Lanczos bidiagonalization methods are their high accuracy. However, our methods are faster at computing a TSVD approximation.

7.2.2 Lanczos with Kronecker Products

The Lanczos bidiagonalization process does not, by default, exploit Kronecker product structure in its computations. Here, we discuss one approach to add Kronecker product structure to the Lanczos process. Bentbib et al. [4] explored this problem for limited Kronecker structure such that $\boldsymbol{K} = \boldsymbol{A}_1 \otimes \boldsymbol{B}_1$ exactly; we develop the idea further to incorporate Kronecker summation structure more broadly throughout the algorithm. The description herein is theoretical only; we have not run experiments,

as they are outside the scope of this dissertation. This is an area of future work that we encourage others to explore.

GKB Algorithm given $\boldsymbol{d}$, $\boldsymbol{K}$.

1. $\beta_1 = ||\boldsymbol{d}||_2$, $\boldsymbol{u}_1 = \boldsymbol{d}/\beta_1$
2. $\boldsymbol{v}_1 = \boldsymbol{K}^T\boldsymbol{u}_1, \alpha_1 = ||\boldsymbol{v}_1||, \boldsymbol{v}_1 = \boldsymbol{v}_1/\alpha_1$
3. for $i = 2 : k$
4. $\quad \boldsymbol{u}_i = \boldsymbol{K}\boldsymbol{v}_{i-1} - \alpha_{i-1}\boldsymbol{u}_{ik-1}$
5. $\quad$ #reorthogonalize $\boldsymbol{u}_i$
6. $\quad \beta_i = ||\boldsymbol{u}_i||_2$
7. $\quad \boldsymbol{u}_i = \boldsymbol{u}_i/\beta_i$
8. $\quad \boldsymbol{v}_i = \boldsymbol{K}^T\boldsymbol{u}_i - \beta_i\boldsymbol{v}_{i-1}$
9. $\quad$ #reorthogonalize $\boldsymbol{v}_i$
10. $\quad \alpha_i = ||\boldsymbol{v}_i||_2$
11. $\quad \boldsymbol{v}_i = \boldsymbol{v}_i/\alpha_i$

Algorithm 1: Lanczos-Golub-Kahan Bidiagonalization Algorithm. The result of k iterations of this procedure is a size $k \times k$ bidiagonalization of the input matrix, $\boldsymbol{K}$.

Call the j^{th} column of $\boldsymbol{U}$ $\boldsymbol{u}_j$.

Reorthogonalize Algorithm given $\boldsymbol{U}$, i:

1. for $j = 1 : i - 1$
2. $\quad \boldsymbol{u}_i = \boldsymbol{u}_i - (\boldsymbol{u}_j^T\boldsymbol{u}_i)\boldsymbol{u}_j$

Algorithm 2: Reorthogonalization algorithm for Lanczos-Golub-Kahan bidiagonalization. This can be used to reorthogonalize the vectors $\boldsymbol{u}_i$ and $\boldsymbol{v}_i$ constructed in Algorithm 1.

There are two layers of Kronecker structure in this proposed Lanczos algorithm modification. The first layer approximates $\boldsymbol{K}$ as a Kronecker summation decomposi-

tion (2.4). This minimally impacts the structure of computations in Algorithm 1. In place of the full matrix-vector multiplications, multiplications use a Kronecker summation decomposition of $\boldsymbol{K}$. This reduces the time complexity in the base algorithm with dense $\boldsymbol{K}$ from $O(N^2) = O(n^4)$ to $O(N^{\frac{3}{2}}r) = O(n^3 r)$ for each multiplication, which brings the algorithm from $O(n^4k + n^2k^2)$ to $O(n^3rk + n^2k^2)$. For small r, this can be a significant savings, depending on k. Unfortunately, this does not improve the storage complexity, because the computed Lanczos vectors are still dense.

We can attempt to fix the storage problems by use of the second layer of approximation: keeping the vectors $\boldsymbol{u}_j$ and $\boldsymbol{v}_j$ in Kronecker form as well. This requires more attention than only representing $\boldsymbol{K}$ as a Kronecker summation. $\boldsymbol{u}_1$ is set based on a choice of starting vector such as the right-hand side $\boldsymbol{d}$ (although other choices are possible). Suppose we start with $\boldsymbol{u}_1 = \boldsymbol{u}_a \otimes \boldsymbol{u}_b$. In line 2 of Algorithm 1, we already run into the first difficulty: multiplying $\left(\sum_{i=1}^{r} \boldsymbol{A}_i \otimes \boldsymbol{B}_i\right)(\boldsymbol{u}_a \otimes \boldsymbol{u}_b) = \sum_{i=1}^{r} \boldsymbol{A}_i\boldsymbol{u}_a \otimes \boldsymbol{B}_i\boldsymbol{u}_b$ results in $\boldsymbol{v}_1$ that has Kronecker structure, but of Kronecker rank r. This is an increase from the Kronecker rank of $\boldsymbol{u}_1$, which is only 1. In fact, each matrix-vector multiplication with our approximation of $\boldsymbol{K}$ (or its transpose) increases the Kronecker rank of the vector by a multiplicative factor of up to r. Furthermore, if two vectors with a Kronecker summation form are added, the resulting rank can be up to the sum of the individual ranks. Together, these result in an exponential increase of Kronecker rank.

One common method to combat the explosion of Kronecker rank is to truncate the result to a lower Kronecker rank [34]. This results in approximate $\boldsymbol{u}_i$ and $\boldsymbol{v}_i$, as opposed to the near-exact computations of the original aglorithm. Truncation is particularly imprecise for reorthogonalization. The reorthogonalization process requires many additions of vectors in Kronecker form, each of which have to be truncated back to a reasonable Kronecker rank. This makes the reorthogonalization process inexact, which is likely to degrade the algorithm's performance.

There are various methods for truncating Kronecker products down to a reduced Kronecker rank. For example, a single term of highest magnitude may be used, or the vector may be fully formed and re-converted back to Kronecker structure. This is similar to the problem of expressing tensors with a low-rank decomposition or low-rank approximation; see, for example, [23, 34]. Different methods vary considerably in accuracy and cost.

Nonetheless, assume that we do use truncation of Kronecker rank. How does this affect the time and storage complexity of the algorithm? If truncation takes $O(\tau)$ time and can only be applied after a vector of too-high rank is computed rather than implicitly during the computation, the time complexity adjusts accordingly. The algorithm without reorthogonalization has significantly decreased costs. If $\boldsymbol{u}_j$ and $\boldsymbol{v}_j$ are kept at Kronecker rank 1 then the multiplications in lines 2, 4, and 8 of Algorithm 1 now each take $O(n^2 r)$ time instead of $O(n^3 r)$. Addition only incurs the cost of truncation in this scheme, since it requires combining two separate Kronecker sums into one and truncating. The costs for additions are dwarfed by the costs for multiplications, and do not change the time complexity. The algorithm's cost without reorthogonalization is reduced from $O(n^3 rk)$ down to $O(n^2 rk + k\tau)$. More generally, if each $\boldsymbol{u}_j$ is kept to Kronecker rank at most r_u and $\boldsymbol{v}_j$ is kept to Kronecker rank at most r_v, the cost without reorthogonalization is $O(n^2 rk(r_u + r_v) + k\tau)$.

Including reorthogonalization changes the algorithm's complexity. The reorthogonalization Algorithm 2 requires subtracting scaled versions of Kronecker vectors. The resulting vector requires truncation, which has the most obvious impact on complexity. However, there is also an inner product of vectors in the algorithm. Similar to the norms required in the main Algorithm 1, these can be efficiently computed by exploiting Kronecker product properties to avoid constructing the full vectors.

We start with the definition of an inner product for vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ of the same size. Using $.*$ to denote element-wise multiplication and z_i or $(\boldsymbol{z})_i$ to denote the i^{th}

entry of a vector $\boldsymbol{z}$, $\boldsymbol{x}^T\boldsymbol{y} = \sum_i (\boldsymbol{x}.*\boldsymbol{y})_i$. For general vectors of length n, $(\boldsymbol{a}\otimes\boldsymbol{b}).*(\boldsymbol{c}\otimes\boldsymbol{d}) = (\boldsymbol{a}.*\boldsymbol{c})\otimes(\boldsymbol{b}.*\boldsymbol{d})$. Recall the structure of the Kronecker product for vectors $\boldsymbol{\mu}$ and $\boldsymbol{\nu}$ of length n:

$$\boldsymbol{\mu}\otimes\boldsymbol{\nu} = \begin{bmatrix} \mu_1\boldsymbol{\nu} \\ \mu_2\boldsymbol{\nu} \\ \vdots \\ \mu_n\boldsymbol{\nu} \end{bmatrix}.$$

The sum $\sum_i (\boldsymbol{\mu}\otimes\boldsymbol{\nu})_i = \left(\sum_i \mu_i\right)\left(\sum_j \nu_j\right)$. This means that we can sum the elements in the first vector $\boldsymbol{\mu}$, then multiply the result scaled by sum of the elements in the second vector $\boldsymbol{\nu}$. Then to compute the inner product $(\boldsymbol{a}\otimes\boldsymbol{b})^T(\boldsymbol{c}\otimes\boldsymbol{d})$ we first compute $\boldsymbol{\mu} = \boldsymbol{a}.*\boldsymbol{c}$ and $\boldsymbol{\nu} = \boldsymbol{b}.*\boldsymbol{d}$, sum the elements of $\boldsymbol{\mu}$ yielding the scalar sum Σ_μ, then sum the elements in $\boldsymbol{\nu}$ to get the sum Σ_ν, and compute $\Sigma_\mu\Sigma_\nu$. This is $O(n)$ compared to $O(n^2)$ for fully forming the vectors of length N.

What if we do not have single-term Kronecker product vectors, but instead sums? The procedure is similar, but the complexity increases linearly with the Kronecker rank of *each* vector. Suppose we have $\boldsymbol{g} = \sum_{i=1}^{r_g} \boldsymbol{a}_i\otimes\boldsymbol{b}_i$ and $\boldsymbol{h} = \sum_{i=1}^{r_h} \boldsymbol{c}_i\otimes\boldsymbol{d}_i$ with $\boldsymbol{a}_i, \boldsymbol{b}_i, \boldsymbol{c}_i, \boldsymbol{d}_i \in \mathbb{R}^n$ for all i. Then

$$\begin{aligned}
\boldsymbol{g}^T\boldsymbol{h} &= \sum_{i=1}^{r_g}\sum_{j=1}^{r_h}\sum_{k=1}^{n} (\boldsymbol{a}_i.*\boldsymbol{c}_j)\otimes(\boldsymbol{b}_i.*\boldsymbol{d}_j)_k \\
&= \sum_{i=1}^{r_g}\sum_{j=1}^{r_h}\left(\sum_{k=1}^{n}(\boldsymbol{a}_i.*\boldsymbol{c}_j)_k\right)\left(\sum_{\ell=1}^{n}(\boldsymbol{b}_i.*\boldsymbol{d}_j)_\ell\right) \\
&\text{Letting } \boldsymbol{\mu}^{(ij)} = \boldsymbol{a}_i.*\boldsymbol{c}_j \text{ and } \boldsymbol{\nu}^{(ij)} = \boldsymbol{b}_i.*\boldsymbol{d}_j, \\
&= \sum_{i=1}^{r_g}\sum_{j=1}^{r_h}\left(\sum_{k=1}^{n}\boldsymbol{\mu}_k^{(ij)}\right)\left(\sum_{\ell=1}^{n}\boldsymbol{\nu}_\ell^{(ij)}\right) \\
&\text{Letting } \sum_{k=1}^{n}\boldsymbol{\mu}_k^{(ij)} = \Sigma_{\mu,ij} \text{ and } \sum_{\ell=1}^{n}\boldsymbol{\nu}_\ell^{(ij)} = \Sigma_{\nu,ij},
\end{aligned}$$

$$= \sum_{i=1}^{r_g} \sum_{j=1}^{r_h} \Sigma_{\mu,ij} \Sigma_{\nu,ij}.$$

Computing $\Sigma_{\mu,ij}$ and $\Sigma_{\nu,ij}$ takes $O(n)$ for each i and j. This algorithm then takes $O(r_g r_h n)$ time. This is cheaper than the $O(N) = O(n^2)$ time for computing the inner product of the full vectors if and only if $r_g r_h < n$. In this algorithm, it is crucial to keep the Kronecker rank of the vectors low.

We can now deduce the overall time complexity. Assume all vectors $\boldsymbol{u}_i$ are truncated to a Kronecker rank r_u and all vectors $\boldsymbol{v}_i$ are similarly truncated to Kronecker rank r_v, with a truncation cost of $O(\tau)$. If truncation is done outside the loop at line 1 of Algorithm 2, then this algorithm has cost $O(n^2 rk(r_u + r_v) + nk^2(r_u^2 + r_v^2) + k\tau)$. If truncation is done inside the loop, the complexity is instead $O(n^2 rk(r_u + r_v) + nk^2(r_u^2 + r_v^2) + k^2\tau)$ (the final $k\tau$ term becomes $k^2\tau$). In summary:

Algorithm Variant	**Complexity**
Original	$O(n^4 k + n^2 k^2)$
$\boldsymbol{K}$ as Kronecker sum, only	$O(n^3 rk + n^2 k^2)$
$\boldsymbol{K}$ and vectors as Kronecker sums, truncate after reorthogonalization loop	$O(n^2 rk(r_u + r_v) + nk^2(r_u^2 + r_v^2) + k\tau)$
$\boldsymbol{K}$ and vectors as Kronecker sums, truncate in reorthogonalization loop	$O(n^2 rk(r_u + r_v) + nk^2(r_u^2 + r_v^2) + k^2\tau)$

Table 7.1: Time complexity comparison for variations of the Lanczos-Golub-Kahan bidiagonalization routine.

One considerable downside to this algorithm is the resulting structure of data. At the end, the columns of the matrices $\boldsymbol{U}_k$ and $\boldsymbol{V}_k$ are represented as vectors, not matrices, in Kronecker product summation form. This seriously constrains, for example, multiplications with the matrix. Multiplication with the matrix can proceed using the "weighted column summation" form of multiplication, where, for example, each

column $\boldsymbol{u}_i$ in $\boldsymbol{U}_k$ is summed with a weight: $\boldsymbol{U}_k\boldsymbol{x} = \sum_{i=1}^{k} x_i\boldsymbol{u}_i$, and transpose multiplication can proceed naturally. In all cases, the results of multiplication have the same rank problems with increasing Kronecker rank as appear in the Lanczos algorithm.

Despite the structural issues, there is some storage savings from this method. Normally, storing the Lanczos vectors takes $O(n^2k)$ space, but using the Kronecker method reduces this to $O(nk(r_u + r_v))$. This can enable computations on larger matrices than the standard method, at the cost of lower accuracy.

Without experimenting, it is unclear whether the storage and time savings are worth the loss of accuracy in this modification. We reiterate that experiments and future work are essential to determine the viability of this algorithm in practice.

7.2.3 Summary

Lanczos-Golub-Kahan bidiagonalization is a powerful algorithm that enables computation of the truncated SVD of a matrix. If high accuracy is essential, it is a good choice, along with the randomized methods discussed in Section 7.3. However, our Kronecker-based methods are preferable if accuracy can be reduced for time savings, or if storing the singular vectors is prohibitively memory-intensive. It may be possible to modify Lanczos bidiagonalization to incorporate Kronecker structure to reduce time and space costs.

7.3 Randomized Algorithms

Like Lanczos bidiagonalization, the general class of randomized methods for computing a truncated SVD detailed by Halko, Martinsson, and Tropp [24] work on a variety of problems, including cases when the matrix $\boldsymbol{K}$ is only accessible via function call form to compute matrix vector products. These stable methods use a randomized starting guess and deterministic algorithms to get their solution. This results in a

few unique benefits, such as straightforward parallelization and directly tunable error bounds. However, the time complexity is not necessarily less than Lanczos methods depending on certain parameter choices; this method still involves a reorthogonalization step. Also, these methods have same issue of storage cost that appears in Lanczos bidiagonalization due to the dense storage of singular vectors.

The main method proposed by Halko, Martinsson, and Tropp in [24] for computing an approximate TSVD is as follows. We start by choosing a random set of (slightly more than) k vectors $\boldsymbol{\omega}_i$ (for example, Gaussian random vectors) which form the columns of a matrix $\boldsymbol{\Omega}$. These are used as random samples to the range of $\boldsymbol{K}$ by computing $\boldsymbol{Y} = \boldsymbol{K}\boldsymbol{\Omega}$ or, if the singular values of $\boldsymbol{K}$ decay slowly, $\boldsymbol{Y} = (\boldsymbol{K}\boldsymbol{K}^T)^q\boldsymbol{K}\boldsymbol{\Omega}$ for a chosen factor $q \in \mathbb{Z}^+$. Once $\boldsymbol{Y}$ is computed, a thin QR (see [21]) $\boldsymbol{Y} = \boldsymbol{Q}\boldsymbol{R}$ is calculated. Power iteration steps may be interleaved with thin QR steps to prevent undue error from roundoff. $\boldsymbol{Q}$ gives an orthonormal basis that is expected to span the first k singular vectors of $\boldsymbol{K}$. This is used to compute a small matrix $\boldsymbol{B} = \boldsymbol{Q}^T\boldsymbol{K}$. Then, as with the algorithms presented here, an SVD of the small matrix $\boldsymbol{B}$ is used to approximate the SVD of the full matrix: $\boldsymbol{B} = \boldsymbol{U}_B\boldsymbol{\Sigma}_B\boldsymbol{V}_B^T$ and therefore $\boldsymbol{K} \approx \boldsymbol{Q}\boldsymbol{U}_B\boldsymbol{\Sigma}_B\boldsymbol{V}_B^T$. There are variants on this method using an oversampling parameter p to choose how many columns of $\boldsymbol{\Omega}$ to compute, different choices of the power iteration count q, optimizations for sparse and function-based $\boldsymbol{K}$, and more.

In the image deconvolution problem, a compact PSF may enable fast multiplications with the matrix $\boldsymbol{K}$. For this reason, we discuss both the base method for full, dense $\boldsymbol{K}$ and the function call version which does not require explicit storage of $\boldsymbol{K}$.

Halko, Martinsson, and Tropp provide time complexities for their algorithms. For dense matrices, the cost is $O(n^4 \log(k) + n^2k^2)$. For matrices which do not require explicit storage of $\boldsymbol{K}$ and have multiplication cost $O(T_{mult})$, the cost is $O(T_{mult}(k + p)(q+1)+n^2k^2)$. In image deblurring problems, T_{mult} is $\Omega(n^2)$, although typical costs are $\Omega(n^3)$ (while "big O" is an asymptotic *upper* bound, "big Ω" is an asymptotic

lower bound [10, p. 48]). Cheaper multiplications indicate that less pixels are blurred, which can correspond to a problem with less severe blur; often, the blur is significant and widespread. Of the examples used in Chapters 3-6, only the motion blur had a compact PSF with $\Theta(n^3)$ multiplication cost ("big Θ" is both "big O" and "big Ω": an asymptotically tight bound [10, pp. 45-46]). If the function form of the randomized algorithm is used, the cost should be no more than $O(n^3(k+p)(q+1)+n^2k^2)$, which, because p and q are so small as to often be constant, is approximately $O(n^3k+n^2k^2)$. Compared to the time complexity of $O(n^3r + k^2r + k^3)$ for the reordering method proposed here, this is expected to be slower for typical k (usually $k > n$ to be effective) and r (r is at most n, but typically chosen to be very small).

To be clear, this is not to say that the randomized methods are slow. Those algorithms are much more general than the methods presented here (no Kronecker structure is required), and they can be tuned to have much higher accuracy than our methods. Higher accuracy requires larger choices of $k + p$ and potentially q for matrices $\boldsymbol{K}$ with slow singular value decay. If a less strict accuracy bound is acceptable, the algorithms can be run quickly, especially because they are so easily parallelizable. Unlike the Lanczos algorithm and similar methods, the singular vectors can be computed simultaneously rather than sequentially. But this may not be the limit to acceleration for the randomized methods. Adjusting the algorithms to use Kronecker structure, as was discussed for the Lanczos-based methods in Subsection 7.2.2, could improve the complexity so as to be faster than the methods proposed here. This would require more adjustment than just using a Kronecker summation decomposition (2.5) of $\boldsymbol{K}$, which has $T_{mult} = O(n^3r)$, and is not better than the cost discussed above.

Like the time complexity, the storage complexity of the randomized methods is higher than the Kronecker methods proposed here. As with the Lanczos-based methods, the singular vectors are stored densely with the randomized methods. The min-

imum storage required is then $\Omega(n^2k)$, whereas the Kronecker-based methods have storage cost $O(n^2 + k^2)$. The randomized methods do not require explicitly forming $\boldsymbol{K}$, which would increase the cost further, but even without forming $\boldsymbol{K}$ the storage cost is larger as k gets large. Again, $O(n^2k)$ is typical for methods that do not exploit special structure, but it is larger than our proposed method.

The category where the randomized methods are better than our proposed methods is accuracy. They can be tuned to be arbitrarily accurate (within the computation limits of computers), albeit with a considerable speed penalty. Our proposed methods, in comparison, have a bounded, moderate accuracy. Like the methods discussed in Section 7.2, our algorithms are worse by a landslide in the accuracy category.

The randomized methods proposed by Halko, Martinsson, and Tropp are powerful, general methods for computing several factorizations, including the TSVD. Compared to our methods, they are, in theory, likely to be slower and require a higher storage cost, but are more accurate. It is possible that adding Kronecker structure to the computations could improve the speed and storage costs, although the accuracy may be reduced depending on how the structure was added.

7.4 Conclusion

In this chapter, we looked at three alternative methods to our proposed Kronecker-based TSVD approximation: fast Fourier transforms; Lanczos-Golub-Kahan bidiagonalization; and randomized methods proposed by Halko, Martinsson, and Tropp. Fourier transforms are easy to store, fast to use, and highly accurate if the matrix $\boldsymbol{K}$ has a circulant-like (such as block circulant with circulant block) structure. They are a less accurate approximation for other matrix structures, and slower than our proposed methods to construct. Lanczos-based methods give very high accuracy, but have a high cost for reorthogonalization. If only a few singular values and vectors

need to be computed (k is small), these methods are a good choice. The randomized methods have a more directly tunable accuracy and can be easily parallelized compared to Lanczos methods, although the time complexity is the same as the Lanczos methods. These methods work well for situations where high accuracy is desired as well. In addition, for extremely large matrices these methods can work with very few passes over the data. But if there is not a block circulant structure, the singular vectors are too costly to store or the time taken for the these methods is prohibitive, and some accuracy can be sacrificed to get lower storage and considerable speed, then our proposed methods are a good choice. Further, because the singular vectors are stored in Kronecker format, operating with the resulting decomposition is fast. This can be especially useful in situations where the decomposition is computed once but used many times, such as for a preconditioner.

Chapter 8

Blind Deconvolution

In this chapter we apply the methods developed in Chapters 3-6 to the blind deconvolution problem. Taking a somewhat unconventional approach, our aim is to determine if the blind deconvolution framework can improve the singular value approximations of our methods. This can enable better reconstructions, but our primary objective focuses on the factorization, rather than restoration.

8.1 Blind Deconvolution Framework

The blind deconvolution problem, introduced in Chapter 1, is a variant on the standard image deconvolution problem in which the blur operator is not exactly known. Instead, typically we have an initial guess of the blur operator $\boldsymbol{K}_0$ along with an observed blurred image $\boldsymbol{d}$, and seek to solve

$$\boldsymbol{K}\boldsymbol{x} + \boldsymbol{e} = \boldsymbol{d}$$

where both $\boldsymbol{x}$ and $\boldsymbol{K}$ are unknown.

To make this problem tractable, we place constraints on $\boldsymbol{K}$. If we have an initial guess $\boldsymbol{K}_0$, we often penalize deviation of the reconstructed operator $\boldsymbol{K}$ from $\boldsymbol{K}_0$.

The same restriction can be imposed on $\boldsymbol{x}$ deviating from $\boldsymbol{x}_0$, the initial reconstruction. Mathematically, we can impose those constraints using regularization (with parameters α_1 and α_2) in a least squares formulation of the problem:

$$\min_{\boldsymbol{K},\boldsymbol{x}} \|\boldsymbol{K}\boldsymbol{x} - \boldsymbol{d}\|_2^2 + \alpha_1\|\boldsymbol{K} - \boldsymbol{K}_0\|_F^2 + \alpha_2\|\boldsymbol{x} - \boldsymbol{x}_0\|_2^2. \tag{8.1}$$

Here, we elect to use 2-norms for vectors and Frobenius norm for matrices, but different norms can be used in practice.

This formulation (8.1) still is extremely broad. Although we penalize deviations of $\boldsymbol{K}$ from $\boldsymbol{K}_0$, the optimization space is huge, as $\boldsymbol{K}$ has $N^2 = n^4$ entries that can be changed. Based on the work by Dykes et al. [14], we constrain our blind deconvolution by assuming that our approximation $\boldsymbol{K}_0$ has the correct singular vector matrices. That is, our minimization problem gains the additional constraint that $\boldsymbol{K}_0 = \boldsymbol{U}_0\boldsymbol{\Sigma}_0\boldsymbol{V}_0^T \Leftrightarrow \boldsymbol{K} = \boldsymbol{U}_0\boldsymbol{\Sigma}\boldsymbol{V}_0^T$ for an unknown $\boldsymbol{\Sigma}$. As shown in [14] for the Lanczos decomposition, this enables us to re-write some of the general blind deconvolution formulation (8.1):

$$\begin{aligned}
\|\boldsymbol{K} - \boldsymbol{K}_0\|_F^2 &= \|\boldsymbol{U}_0\boldsymbol{\Sigma}_0\boldsymbol{V}_0^T - \boldsymbol{U}_0\boldsymbol{\Sigma}\boldsymbol{V}_0^T\|_2^2 \\
&= \|\boldsymbol{U}_0(\boldsymbol{\Sigma}_0 - \boldsymbol{\Sigma})\boldsymbol{V}_0^T\|_F^2 \\
&= \|\boldsymbol{\Sigma}_0 - \boldsymbol{\Sigma}\|_F^2 \\
&\text{Using } \boldsymbol{\sigma}_0 = \operatorname{diag}(\boldsymbol{\Sigma}_0) \text{ and } \boldsymbol{\sigma} = \operatorname{diag}(\boldsymbol{\Sigma}), \\
&= \|\boldsymbol{\sigma}_0 - \boldsymbol{\sigma}\|_2^2.
\end{aligned}$$

Similarly,

$$\begin{aligned}
\|\boldsymbol{K}\boldsymbol{x} - \boldsymbol{d}\|_2^2 &= \|\boldsymbol{U}_0\boldsymbol{\Sigma}\boldsymbol{V}_0^T\boldsymbol{x} - \boldsymbol{d}\|_2^2 \\
&= \|\boldsymbol{\Sigma}\boldsymbol{V}_0^T\boldsymbol{x} - \boldsymbol{U}_0^T\boldsymbol{d}\|_2^2
\end{aligned}$$

and

$$\|\boldsymbol{x} - \boldsymbol{x}_0\|_2^2 = \|\boldsymbol{V}_0^T\boldsymbol{x} - \boldsymbol{V}_0^T\boldsymbol{x}_0\|_2^2.$$

Calling $\boldsymbol{y} = \boldsymbol{V}_0^T\boldsymbol{x}$, $\boldsymbol{y}_0 = \boldsymbol{V}_0^T\boldsymbol{x}_0$, $\tilde{\boldsymbol{d}} = \boldsymbol{U}_0^T\boldsymbol{d}$, and using $.*$ to denote element-wise multiplication, we get the greatly simplified problem

$$\min_{\boldsymbol{\sigma},\boldsymbol{y}} \|\boldsymbol{\sigma}.*\boldsymbol{y} - \tilde{\boldsymbol{d}}\|_2^2 + \alpha_1\|\boldsymbol{\sigma} - \boldsymbol{\sigma}_0\|_2^2 + \alpha_2\|\boldsymbol{y} - \boldsymbol{y}_0\|_2^2. \tag{8.2}$$

Let the i^{th} entry of a general vector $\boldsymbol{z}$ be called z_i and the i^{th} entry of $\boldsymbol{y}_0$ be called $\boldsymbol{y}_{0,i}$. Then, as shown in [14], this problem immediately reduces to the fully separated problems

$$\min_{\sigma_i,y_i} (\sigma_i y_i - \tilde{d}_i)^2 + \alpha_1(\sigma_i - \sigma_0)^2 + \alpha_2(y_i - y_{0,i})^2. \tag{8.3}$$

The details of how to minimize this polynomial are given by Dykes et. al in [14]. Once an optimal $\boldsymbol{\sigma}$ and corresponding $\boldsymbol{y}$ are calculated, we recover our approximation of the optimal $\boldsymbol{x}$ using $\boldsymbol{x} \approx \boldsymbol{V}_0\boldsymbol{y} = \boldsymbol{V}_0\boldsymbol{V}_0^T\boldsymbol{x}$. Recall that $\boldsymbol{V}_0$ has truncated columns, so this is not an exact computation.

Why would we go through all this work, and with a heavily simplifying assumption that the singular vectors in $\boldsymbol{K}_0$ is exact? The hope is that we can use real observations to improve our approximation of the blur operator. By incorporating information about the right-hand side $\boldsymbol{d}$, we gain more information about the true blur operator than we do if we only have the matrix $\boldsymbol{K}$. Especially in situations where we use a truncated approximation with $r < R$ terms in the Kronecker summation, this may allow us to improve our estimates of the singular values more than we could otherwise.

8.2 Discussion

In the preceding derivation, the impact of using truncated $\boldsymbol{U}_0$ and $\boldsymbol{V}_0$ may not have been clear. Here, we emphasize the heavy limitations that stem from using a truncated approximation.

From a broad perspective not focused on linear algebra, if we do not truncate our SVD, then equation (8.2) is equivalent to solving N equations (8.3) for a total of $2N$ variables. But when we truncate the SVD to k terms, we solve for only $2k$ variables. If we need all N variables in, for example, $\boldsymbol{x}$ to capture the true image, then we cannot expect to recover a good approximation of the true image with our TSVD of size k.

In linear algebraic terms, if the rank of our approximation of $\boldsymbol{K}$ has rank k but the true matrix $\boldsymbol{K}$ has a much higher rank $O(N)$, we cannot expect to recover a good solution with our approximation.

If we use a bad approximation $\boldsymbol{y}_0 = \boldsymbol{V}_0^T \boldsymbol{x}_0$, we also expect significant error. The goal is to use the right-hand side to inform an update of the blur operator. But if we have too low of a truncation index k, we use the wrong information to inform our update. Similarly, if there is a high level of noise $\|\boldsymbol{e}\|_2^2$, the singular value update will be corrupted.

The upshot is that in general we cannot expect that a blind reconstruction from (8.2) will be significantly better than a reconstruction from a non-blind framework (2.1). The reconstruction may even be worse.

With these limitations in mind, we expect that low truncation indices will give poor results, as will problems largely contaminated by noise. In Section 8.3, we conduct experiments to determine the performance of our method in this blind deconvolution framework. Our experiments are guided by these limitations, and are restricted in their generality as a result.

8.3 Experimental Performance

We ran the blind deconvolution problem with significant and specific constraints and found that, given those constraints, blind deconvolution improves the estimates of the smallest singular values for some test problems. As we will discuss, these constraints are excessively restrictive for most practical applications. Overwhelmingly, this framework of blind deconvolution does not improve the estimates of singular values, nor does it improve the reconstructions produced. We begin by discussing the case that did work, and follow with a list of cases that did not give positive results. At the end, we provide a practical guide for using this method.

We tested the blind deconvolution framework (8.2) using the PSF from the Satellite Example introduced in Section 3.4. We used test images of size 32×32 and used zero boundary conditions on the blur operator. Starting with this matrix, we computed its SVD $\boldsymbol{K} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$ using MATLAB's `svd` command. To create our true operator $\boldsymbol{K}_*$ we perturbed the singular values of the matrix with 1% Gaussian noise (relative to the magnitude of each individual singular value), yielding $\boldsymbol{K}_* = \boldsymbol{U}\boldsymbol{\Sigma}_*\boldsymbol{V}^T$. Our right-hand side $\boldsymbol{d}$ was computed as $\boldsymbol{d} = \boldsymbol{K}_*\boldsymbol{x}_{\text{true}} + \boldsymbol{e}$, where $\boldsymbol{e}$ is a vector of 1% (relative to the magnitude of $\boldsymbol{K}_*\boldsymbol{x}_{\text{true}}$) Gaussian noise. The matrix $\boldsymbol{K}_*$ was used to generate the problem, but the matrix $\boldsymbol{K}$ was used to generate our initial matrix $\boldsymbol{K}_0$.

We generated the initial estimate $\boldsymbol{K}_0$ using the methods in Chapters 3-6 to approximate the matrix $\boldsymbol{K}$. We computed the initial guess $\boldsymbol{x}_0$ using a Tikhonov regularized least squares solution with regularization parameter of $\lambda = 10^{-5}$. For the regularization of the minimization problem, we used regularization parameters $\alpha_1 = 10^5$ (greatly penalizing the deviation of $\boldsymbol{y}$ from $\boldsymbol{y}_0$) and $\alpha_2 = 10^{-10}$ (barely penalizing deviations of $\boldsymbol{\sigma}$ from $\boldsymbol{\sigma}_0$).

The methods which yielded positive results were the baseline and hybrid methods. It is unsurprising that the hybrid method worked when the baseline method worked, since most of the data in the hybrid method comes from the baseline method. For

these methods, the blind deconvolution framework improved the estimates of the smallest singular values once sorted (recall that due to Kronecker product structure, the singular values are not sorted for the baseline and therefore hybrid methods). The singular values and a smoothed version of the corresponding relative error are shown in Figure 8.1 for the hybrid method. The hybrid method is a hybrid of the truncation and baseline method, with $\ell = m = 16$. The error is smoothed to show the average of 5 errors, centered around the given index. This is necessary to damp large fluctuations between indices and therefore actually see plots clearly.

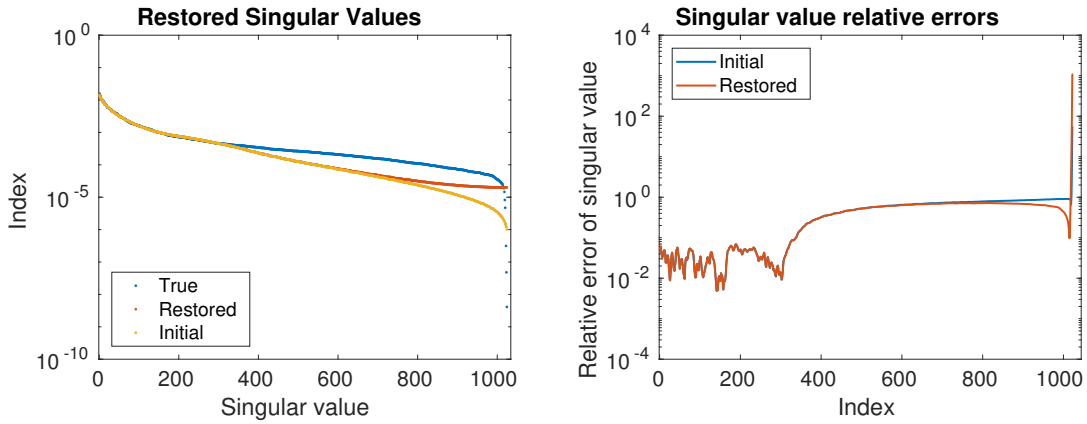


Figure 8.1: Blind Deconvolution Hybrid Results. The true, initial, and improved singular values for the hybrid method blind deconvolution problem. The singular values' estimates are considerably improved for the smallest singular values, excluding the last few entries.

For this problem, there is a notable improvement in the error of singular values with indices 800 and above, excluding the very final few which are over-estimated. The largest singular values are unaffected, and are comparably accurately estimated with the initial and restored singular values.

The effect was less pronounced, but still present, for the baseline method. Figure 8.2 shows the computed initial and improved restored singular values for the baseline method, as well as the error. Because the singular values are difficult to see on the plot, a zoomed view of some of the smallest (excluding the very last few) singular

values is shown to the right. We see that for these singular values, the restored singular values are closer to the true values than the values from the initial TSVD approximation.

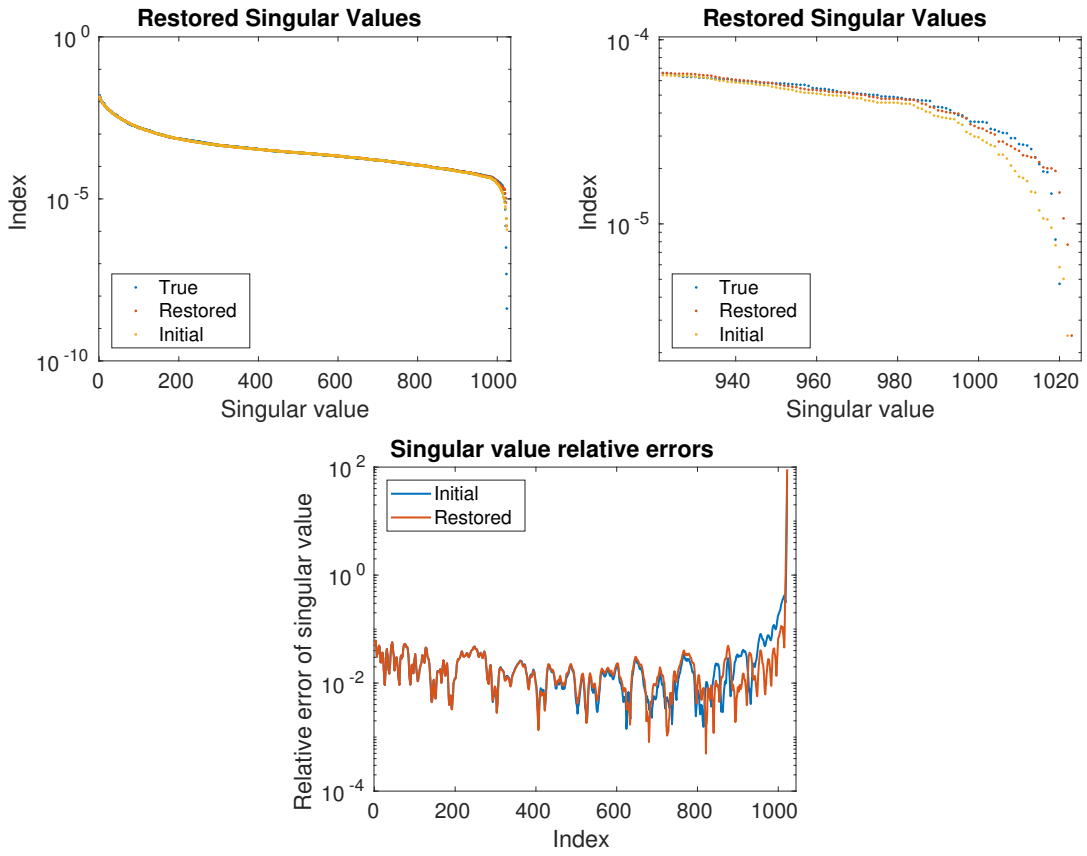


Figure 8.2: Blind Deconvolution Baseline Results. The true, initial, and improved singular values for the baseline method blind deconvolution problem. Although the effect is less pronounced in this example, the last several hundred singular values have an improved estimate using the blind deconvolution framework, again excluding the very last few values.

These two situations showed a moderate improvement in the singular values. Notably, we deviated here from the original framework proposed by Dykes et al. in a subtle way: the singular vectors we used were not actually the true singular vectors in the operator that blurred the image. We used approximate singular vectors which, while they importantly spanned the same space, may not have corresponded to the exact singular vectors in the exact same order as the true operator. This is perhaps

more realistic than assuming singular vectors are exact to the true matrix and in the same order.

If you did, however, use the same singular vectors to construct the true blur operator, you may expect the performance to improve. This was not the case. We found that either the singular values didn't improve, or rounding errors caused significant error in the reconstructed singular values.

In fact, most of our tests did not produce positive results. Using the truncation method or reordering method to test either approach, the one for which we reported positive results or the one just described with exact singular vectors, did not work. As mentioned previously, compressing the data caused serious difficulties with these methods, especially as finite precision arithmetic errors accumulated.

Using a different PSF gave poor results. Recall, the Satellite Example PSF was the one for which the TSVD approximations were unanimously excellent among all methods. With a more difficult to approximate operator, such as the Motion Blur Example's operator, the results were poor for all methods.

Adding too much noise (here we use a low 1%) yielded poor results because the right-hand side we used to inform the singular value updates were too far from the true blurred image.

And finally, not skewing the regularization parameters heavily yields poor results. Here we used $\alpha_1 = 10^5$ and $\alpha_2 = 10^{-10}$. This heavily penalizes changes in $\boldsymbol{y}$ from $\boldsymbol{y}_0$, but enables the singular values to change relatively freely from their initial values. If we allow α_1 and α_2 to be even remotely close in magnitude, the singular values change only negligibly and the reconstruction changes considerably. Unfortunately, the changes that typically resulted from $\boldsymbol{y}$ deviating from $\boldsymbol{y}_0$ worsened, rather than improved, the reconstruction. We also tested varying the regularization parameter α_2 per singular value index so that the smallest singular values were less penalized than the largest; this did not improve the results.

8.4 Summary

The situations in which this method performs poorly are considerably more frequent than the situations in which this method performs well for the image deconvolution problem. Because the blind deconvolution framework is so cheap (both in computation time and memory), the cost for trying it is low, and does not hurt. If the results deviate only a slight amount from the original singular values, they may be worth using. If there are large deviations, then it is likely the solution is corrupt, and not worth using. In most cases, however, the result is that the singular values do not change significantly.

Chapter 9

Conclusion

We have derived three new methods for approximating a truncated singular value decomposition based on an ordered Kronecker product summation decomposition. The first method uses a truncation of the singular vectors of the most significant Kronecker term to enable a computationally feasible singular value decomposition. The second method is a variant of the first, in which the singular vectors and values of the first Kronecker term are reordered so that the singular values are sorted in descending magnitude prior to truncation. The final method is a hybridization scheme that allows either of these two methods, the standard truncation method or the reordered truncation method, to be combined with the simple and computationally cheap baseline method.

When applied to image deconvolution problems, these methods produce decompositions which approximate the blur operator effectively, enabling useful restorations. The truncation method tends to underestimate the smallest singular values, but gives good restorations due to its inclusion of lower significance singular vectors and values. The reordering method typically gives the most accurate representation of the original matrix. And the hybrid method enables cheap approximations that have the benefits of both the baseline method (ease of computing many singular values and vectors)

and either the truncation or reordering method (which perform better in general on larger singular values).

Arguably the most significant benefit to all of the Kronecker-based methods described, including the baseline method, is that they reduce the memory complexity of the problem. Storage costs are $O(n^2 + k^2)$ for all methods, including storage of the singular vectors. Traditional methods have $O(n^2 k)$ storage cost for the singular vectors. Our proposed methods and the baseline method enable computations on larger matrices than possible using traditional methods. This benefit offsets the drawback of a less accurate solution.

We further explored extending these works to a blind deconvolution setting to determine whether the singular value estimates could be improved using additional data, namely the blurred right-hand side. Overwhelmingly, the answer is that no, the estimates tend to not be improved in this framework, although there are a few situations in which error may be reduced for the smallest singular values using the baseline or hybrid methods.

Our methods are fast to use, cheap to compute, and easy to store. Despite their relatively low accuracy compared to commonly used methods for approximating TSVDs and other factorizations, the methods proposed here are useful in applications with extremely large ill-posed matrices which can be cheaply decomposed into an ordered Kronecker summation decomposition. We encourage future work to determine if the Kronecker framework can be added to more commonly used methods, such as Lanczos bidiagonalization or randomized methods.

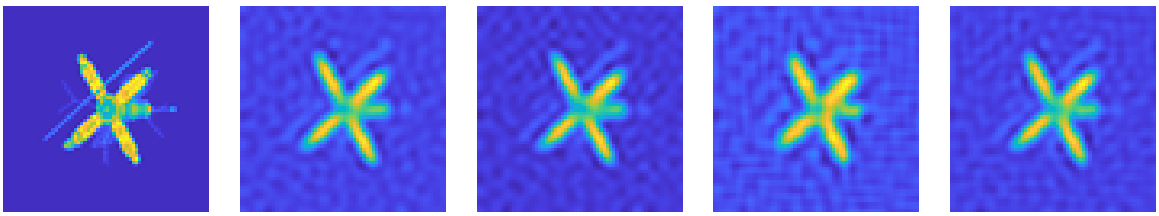
Appendix A

Comparison of Reconstructions

This appendix shows the results from each different method described in Chapters 3-6 for the image reconstruction examples. By viewing the results simultaneously, we can more easily see the variations between the different methods. For all figures, we include negative values to make noise patterns more obvious.

A.1 Satellite Example

The first example we discussed was the satellite example. Recall that all methods produced good results for this example. Indeed, we see this in Figure A.1.



True Baseline Truncated Reordered Hybrid

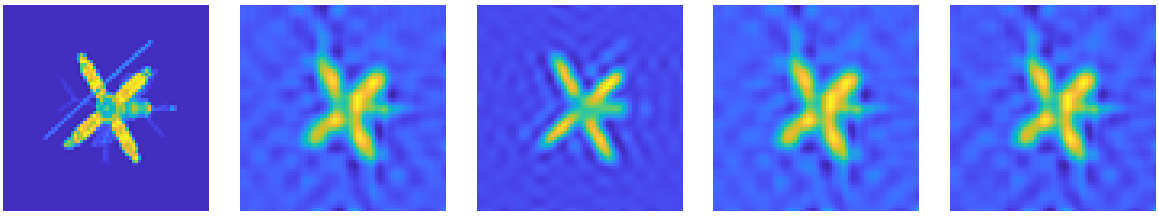
Figure A.1: Satellite Example Comparison. For the Satellite Example, all methods performed well, although the fine details are slightly less prominent for the reordering method than other methods.

Notably, the fine details on the reordering method reconstruction are slightly less

pronounced than for the other methods, with the noise pattern obfuscating the thin rod parallel to the solar panels. Regardless, the reconstruction for all methods is good enough to be useful.

A.2 Grain Example

The second example is the Grain example, for which some methods started to produce poor results. Indeed, the truncation method was the only method to produce significant fine detail in the reconstruction. This can be clearly seen in Figure A.2.



True Baseline Truncated Reordered Hybrid

Figure A.2

The reconstructions besides the truncation method restoration vary primarily in their noise patterns, but the variation is mild. The truncation method is supposed to be less accurate than the reordering method, so it may be surprising that the reconstruction is better. But the accurate reconstruction *is* less accurate to a true rank-400 reconstruction. The reconstruction using MATLAB's `svds` command to compute a true rank-400 TSVD of the blur operator $\boldsymbol{K}$ is shown in Figure A.3. This restoration closely resembles all restorations except the truncation method reconstruction.

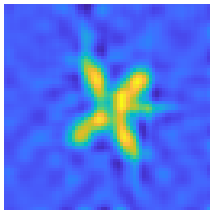
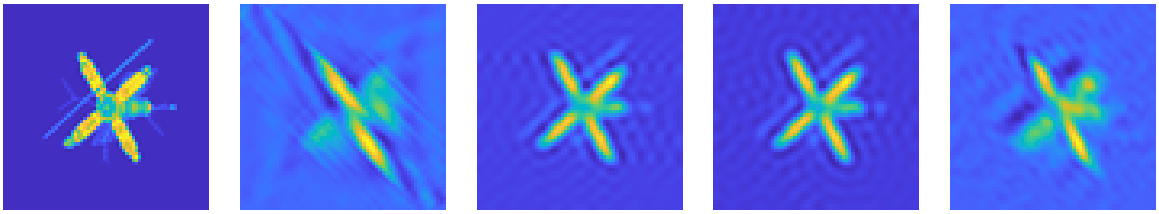


Figure A.3: True Grain Example Reconstruction. This figure shows the reconstruction produced by a true rank 400 TSVD of the Grain Example blur operator.

A.3 Motion Example

The final example we tested is the Motion Example. For this example, the baseline method gave extremely poor results, but the truncation and reordering method produced good results with fine details. The hybrid method, being a combination of the poor baseline estimate and the truncation method reconstruction, produced middling results.



True Baseline Truncated Reordered Hybrid

Figure A.4: Motion Example Comparison. In the Motion Example, we saw the poor performance of the baseline method when $\boldsymbol{K}_1$ is a poor approximation of the blur operator $\boldsymbol{K}$. The new truncation and reordering methods perform excellently on this case.

Appendix B

Comparison of Singular Values

The singular value approximation tests showed the results for each TSVD approximation method separately. In this appendix, we see error graphs consolidated together, enabling comparisons between the performance of each method. The errors in these plots have been smoothed compared to the true error plots shown in Chapters 3-6; the plots here use an average of the relative errors for indices $i-1$ through $i+1$ to determine the error for index i. This dampens the highly oscillatory fluctuations seen previously, allowing us to see the different errors more clearly.

B.1 Singular Value Atmospheric Blur Example

The first example for singular value approximations was the Singular Value Atmospheric Blur Example. This example showed the strength of the truncation, reordering, and therefore hybrid method for the largest singular values, and the fairly consistent error for the baseline method on all singular values. The smoothed errors for all methods are shown in Figure B.1.

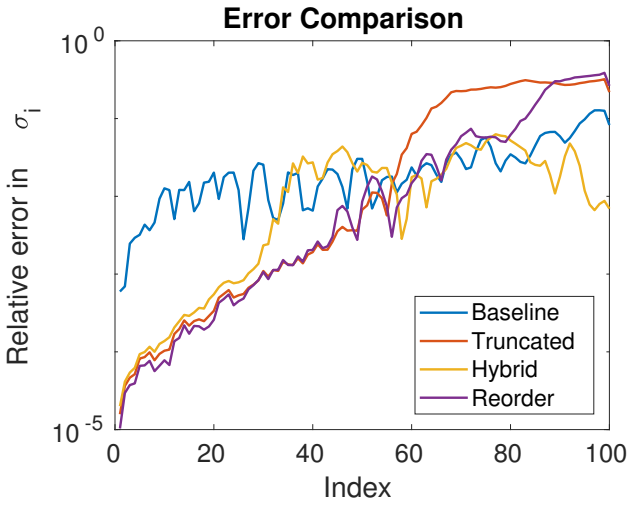


Figure B.1: Singular Value Atmospheric Blur Comparison. This figure shows the smoothed relative errors for all methods on the first singular value approximation example. Eventually, most methods reach a fairly high relative error for the smallest singular values.

This clearly shows the strength of the hybrid method: its error is lower than the baseline method for the largest singular values and lower than the reordering and truncation methods for the largest singular values.

B.2 Singular Value Motion Example

The second and significantly more challenging example was the Singular Value Motion Example. Each method gave less accurate estimates for this example than for the first. The baseline method gave particularly poor estimates for the largest singular values, with error above 10^{-2} for almost all indices. However, consistent to the expected pattern, the baseline method had lower relative error than the truncation and reordering method for the smallest singular values. The hybrid method therefore gave results that combined these strengths. These results are shown in Figure B.2.

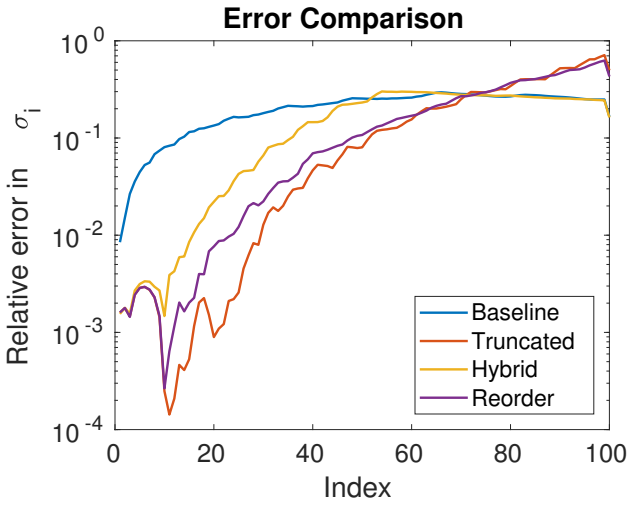


Figure B.2: Singular Value Motion Comparison. This figure shows the smoothed relative errors for all methods on the second singular value approximation example. All methods perform worse for this example than the first, with the baseline method giving a particularly poor estimate of the largest singular values.

Appendix C

Comparison of Preconditioners

This appendix summarizes the timing results from the preconditioning example used to test each of the Kronecker-based methods.

First, we visually represent the timing results in Figure C.1. Clearly, not preconditioning takes the most time and the baseline method takes the least time. However, we can also see that the reordering and truncation methods take less time to complete their iterations than the baseline method. Most of the time taken for the reordering and truncation methods is in constructing the precondtioner. If there are many right-hand sides to solve for, then the higher initial cost can pay off in faster iterations to converge to a true solution for the many right-hand sides. Table C.1 shows the same timing results summarized in Figure C.1, and additionally shows the number of iterations required to converge for each method.

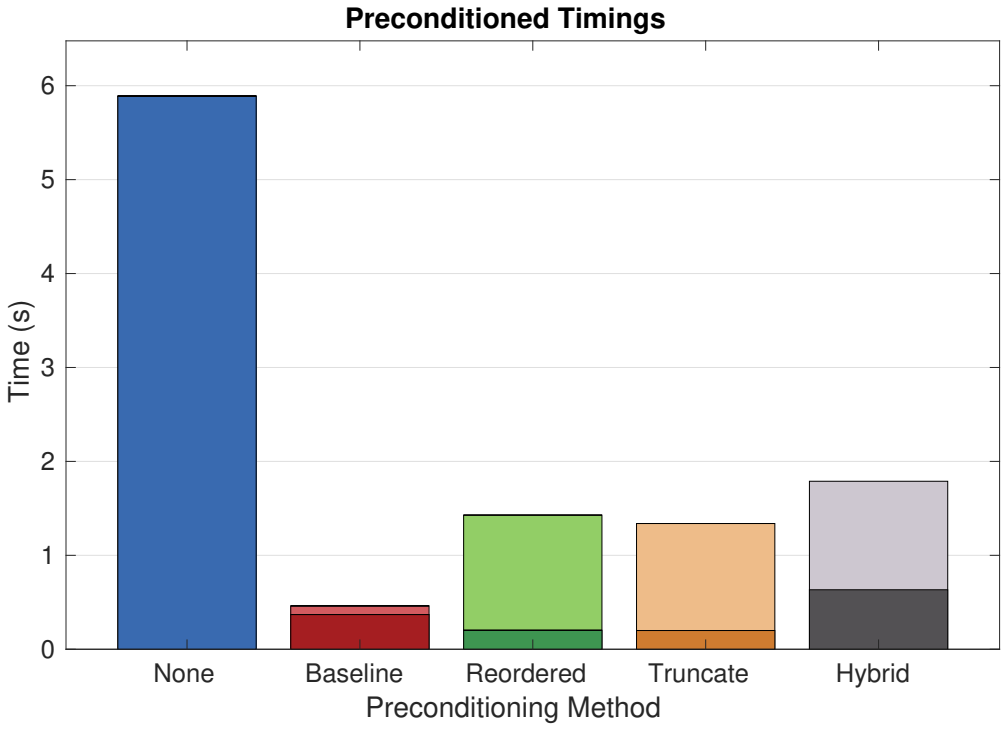


Figure C.1: Preconditioner Timing Comparison. The lighter section of each bar represents the time taken to construct the preconditioner, and the darker portion is the time taken to complete iterations.

	Without	Baseline	Truncated	Reordered	Hybrid
Iterations	345	16	7	7	19
Setup time (sec)	0.0	0.101	1.10	1.22	1.19
Calculate time (sec)	6.04	0.380	0.180	0.208	0.657
Total time (sec)	6.04	0.481	1.28	1.43	1.85

Table C.1: Preconditioner Times for All Methods. This table shows the timing results for all methods, including a split of the computations into setup and iteration (calculate) times. Additionally, the number of iterations required to reach convergence is shown.

Bibliography

[1] Andrews, H., and Hunt, B. *Digital image processing*. Prentice-Hall, Inc., Englewood Cliffs, NJ, 1997.

[2] Banham, M. R., and Katsaggelos, A. K. Digital image restoration. *IEEE signal processing magazine 14*, 2 (1997), 24–41.

[3] Bardsley, J. M. Wavefront reconstruction methods for adaptive optics systems on ground-based telescopes. *SIAM Journal on Matrix Analysis and Applications 30*, 1 (2008), 67–83.

[4] Bentbib, A., El Guide, M., Jbilou, K., and Reichel, L. Global golub–kahan bidiagonalization applied to large discrete ill-posed problems. *Journal of Computational and Applied Mathematics 322* (2017), 46–56.

[5] Benzi, M. Preconditioning techniques for large linear systems: a survey. *Journal of computational Physics 182*, 2 (2002), 418–477.

[6] Björck, Å. *Numerical methods in matrix computations*. Springer, 2016.

[7] Campisi, P., and Egiazarian, K. *Blind image deconvolution: theory and applications*. CRC press, 2016.

[8] Cannon, M. Blind deconvolution of spatially invariant image blurs with phase. *IEEE Transactions on Acoustics, Speech, and Signal Processing 24*, 1 (1976), 58–63.

[9] Chen, J., and Saad, Y. On the tensor svd and the optimal low rank orthogonal approximation of tensors. *SIAM Journal on Matrix Analysis and Applications 30*, 4 (2009), 1709–1734.

[10] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. *Introduction to Algorithms, Third Edition*, 3rd ed. The MIT Press, 2014.

[11] Davis, P. J. *Circulant matrices*. American Mathematical Soc., 2012.

[12] De Lathauwer, L., De Moor, B., and Vandewalle, J. A multilinear singular value decomposition. *SIAM journal on Matrix Analysis and Applications 21*, 4 (2000), 1253–1278.

[13] De Lathauwer, L., De Moor, B., and Vandewalle, J. On the best rank-1 and rank-(r 1, r 2,..., rn) approximation of higher-order tensors. *SIAM journal on Matrix Analysis and Applications 21*, 4 (2000), 1324–1342.

[14] Dykes, L., Ramlau, R., Reichel, L., Soodhalter, K., and Wagner, R. Lanczos-based fast blind deconvolution methods. Tech. rep., Johann Radon Institute for Computation and Applied Mathematics, 2017.

[15] Engl, H. Discrepancy principles for tikhonov regularization of ill-posed problems leading to optimal convergence rates. *Journal of optimization theory and applications 52*, 2 (1987), 209–215.

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

[17] Garvey, C., Meng, C., and Nagy, J. Singular value decomposition approximation via kronecker summations for imaging applications, 2018.

[18] Gazzola, S., Hansen, P. C., and Nagy, J. G. Ir tools: A matlab package of iterative regularization methods and large-scale test problems, 2017.

[19] Golub, G., and Kahan, W. Calculating the singular values and pseudo-inverse of a matrix. *Journal of the Society for Industrial and Applied Mathematics, Series B: Numerical Analysis 2*, 2 (1965), 205–224.

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

[21] Golub, G. H., and Van Loan, C. F. *Matrix computations*, 4 ed. John Hopkins University Press, 2013.

[22] Grasedyck, L. Hierarchical singular value decomposition of tensors. *SIAM Journal on Matrix Analysis and Applications 31*, 4 (2010), 2029–2054.

[23] Grasedyck, L., Kressner, D., and Tobler, C. A literature survey of low-rank tensor approximation techniques. *GAMM-Mitteilungen 36*, 1 (2013), 53–78.

[24] Halko, N., Martinsson, P.-G., and Tropp, J. A. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review 53*, 2 (2011), 217–288.

[25] Hansen, P., Nagy, J., and O'Leary, D. *Deblurring Images: Matrices, Spectra, and Filtering*. SIAM, 2006.

[26] Hansen, P. C. Analysis of discrete ill-posed problems by means of the l-curve. *SIAM review 34*, 4 (1992), 561–580.

[27] Hansen, P. C. *Rank-deficient and discrete ill-posed problems: numerical aspects of linear inversion*, vol. 4. Siam, 2005.

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

[29] Kamm, J., and Nagy, J. G. Kronecker product and svd approximations in image restoration. *Linear Algebra and its Applications 284*, 1 (1998), 177–192.

[30] Kamm, J., and Nagy, J. G. Optimal kronecker product approximation of block toeplitz matrices. *SIAM J. Matrix Anal. Appl. 22*, 1 (2000), 155–172.

[31] Kilmer, M. E., and Martin, C. M. Decomposing a tensor. *Siam news 37*, 9 (2004), 19–20.

[32] Kundur, D., and Hatzinakos, D. Blind image deconvolution. *IEEE signal processing magazine 13*, 3 (1996), 43–64.

[33] Larsen, R. M. Propack-software for large and sparse svd calculations. *Available online. URL http://sun.stanford.edu/rmunk/PROPACK* (2004), 2008–2009.

[34] Lee, K., and Elman, H. C. A preconditioned low-rank projection method with a rank-reduction scheme for stochastic partial differential equations. *SIAM Journal on Scientific Computing 39*, 5 (2017), S828–S850.

[35] Morozov, V. A. On the solution of functional equations by the method of regularization. In *Soviet Math. Dokl* (1966), vol. 7, pp. 414–417.

[36] Nagy, J. G. Decomposition of block toeplitz matrices into a sum of kronecker products with applications in image restoration. Tech. rep., Southern Methodist University, 1996.

[37] Nagy, J. G., and Kilmer, M. E. Kronecker product approximation for preconditioning in three-dimensional imaging applications. *IEEE Transactions on Image Processing 15*, 3 (2006), 604–613.

[38] Nagy, J. G., Ng, M. K., and Perrone, L. Kronecker product approximations for image restoration with reflexive boundary conditions. *SIAM J. Matrix Anal. Appl. 25*, 3 (2003), 829–841.

[39] Nagy, J. G., Palmer, K. M., and Perrone, L. Iterative methods for image deblurring: A Matlab object oriented approach. *Numerical Algorithms 36* (2004), 73–93.

[40] Pitsianis, N. P. *The Kronecker product in approximation and fast transform generation*. PhD thesis, Cornell University, 1997.

[41] Rudin, L. I., Osher, S., and Fatemi, E. Nonlinear total variation based noise removal algorithms. *Physica D: nonlinear phenomena 60*, 1-4 (1992), 259–268.

[42] Saad, Y. *Iterative methods for sparse linear systems*, vol. 82. siam, 2003.

[43] Samarasinghe, P. D., and Kennedy, R. A. Regularized image restoration. In *Image Restoration-Recent Advances and Applications*. InTech, 2012.

[44] Stockham, T. G., Cannon, T. M., and Ingebretsen, R. B. Blind deconvolution through digital signal processing. *Proceedings of the IEEE 63*, 4 (1975), 678–692.

[45] Tikhonov, A. N., Goncharsky, A., Stepanov, V., and Yagola, A. G. *Numerical methods for the solution of ill-posed problems*, vol. 328. Springer Science & Business Media, 2013.

[46] Trefethen, L. N., and Bau III, D. *Numerical Linear Algebra*. SIAM, 1997.

[47] Tucker, L. R. Some mathematical notes on three-mode factor analysis. *Psychometrika 31*, 3 (1966), 279–311.

[48] Tyson, R. K. *Principles of adaptive optics*. CRC press, 2015.

[49] Van Loan, C. F. The ubiquitous kronecker product. *Journal of computational and applied mathematics 123*, 1 (2000), 85–100.

[50] Van Loan, C. F., and Pitsianis, N. Approximation with kronecker products. In *Linear algebra for large scale and real-time applications*. Springer, 1993, pp. 293–314.

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