Distribution Agreement

In presenting this thesis as a partial fulfillment of the requirements for a 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 in whole or in part in all forms of media, now or hereafter now, 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. I retain all ownership rights to the copyright of the thesis. I also retain the right to use in future works (such as articles or books) all or part of this thesis.

Raul Enrique Platero April 12, 2017

Least Squares Updating for Kronecker Products

by

Raul Enrique Platero

Dr. James Nagy

Adviser

Department of Mathematics and Computer Science

Dr. James G. Nagy

Adviser

Dr. Michael K. Rogers

Committee Member

Dr. Effrosyni Seitaridou

Committee Member

2017

Least Squares Updating for Kronecker Products

by

Raul Enrique Platero

Dr. James Nagy

Adviser

Abstract of

A thesis submitted to the Faculty of Emory College

of Emory University in partial fulfillment

of the requirements of the degree of

Bachelor of Sciences with Honors

Department of Mathematics and Computer Science

2017

Abstract

Least Squares Updating for Kronecker Products

By Raul Enrique Platero

In this thesis, we consider an application within image processing where Kronecker products naturally arise. In image deblurring, we attempt to reconstruct the original image from a given blurred image. Due to the ill-conditioned nature of such matrices, along with the large size of the blurring matrix, it becomes difficult to use direct techniques to find the solution to this problem. Therefore, we can use iterative methods and in particular, we used LSQR. However, in order for us to use LSQR efficiently, we constructed a preconditioner using the rank-one updating problem extended to Kronecker products. We show that this method can provide us with a good preconditioner and through some deblurring examples, the solution is as accurate while reducing the average run time.

Least Squares Updating for Kronecker Products

by

Raul Enrique Platero

Advisor : Dr. James Nagy

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

Department of Mathematics and Computer Science

2017

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this thesis we consider the following important situation that arises in many applications: *Given a solution to a mathematical problem, efficiently compute a new solution when the problem is slightly modified.* This is generally referred to as an updating problem. There can be many ways in which the problem is modified, but the basic aim is to use the solution of the original problem (and any intermediate quantities needed to compute it) to efficiently compute the solution to a new problem.

We focus mainly on linear least squares problems where the matrix is modified by the addition of a new rank-one matrix. Since least squares problems are often solved using a QR factorization we consider methods to update this factorization. In particular, we develop a new approach for matrices that have a Kronecker product structure. We also describe how this problem arises in an image deblurring application.

In order for us to begin the discussion on the application of extending the updating QR problem for Kronecker product to image deblurring, we will introduce QR factorization and Givens rotations and their uses along with the updating problem. Then we will briefly introduce the Kronecker product.

1.1 QR Factorization

QR factorization (or QR decomposition) is a factorization method for matrices that is often used to solve linear least squares problems. Linear least square problems are problems having the following form:

$$\min_{\mathbf{x}} \|A\mathbf{x} - \mathbf{b}\|_2^2, \tag{1.1}$$

where $A \in \mathbb{R}^{m\times n}$, $\mathbf{x} \in \mathbb{R}^n$, and $b \in \mathbb{R}^m$. Least squares problems are a particularly useful formulation since often times there may not be an exact answer to

$$Ax = b$$

and hence we would like to find a good solution to the problem

$$Ax \approx b,$$

which is formulated in equation (1.1).

QR factorization factors a matrix into an orthogonal matrix Q and an

upper triangular matrix R. If $A \in \mathbb{R}^{m \times n}$, then there exist matrices Q and R such that

$$A = QR, \tag{1.2}$$

where $Q \in \mathbb{R}^{m \times m}$, $Q^T Q = I$ (i.e Q is an orthogonal matrix) and $R \in \mathbb{R}^{m \times n}$. QR factorization is often used with linear least squares problems because the 2-norm is invariant to orthogonal transformations, meaning that

$$\|A\mathbf{x} - \mathbf{b}\|_2 = \|R\mathbf{x} - Q^T\mathbf{b}\|_2 \tag{1.3}$$

for a given QR factorization of A. There are a few methods to compute the QR factorization of a matrix. For such derivations along with an algorithmic presentation of such derivations, one can refer to Å. Björck's book on matrix computations [2] or G.H. Golub and C.F. Van Loan's book on matrix computations [5].

1.2 Givens Rotations

Givens rotations [4] (or Plane rotations) are a method to construct an orthogonal matrix that can introduce zeros to a matrix. Givens rotations are named after the American mathematician Wallace Givens. These rotations are quite useful, especially when introducing zeros selectively to a given matrix.

A Givens rotation is a matrix that represents a clockwise rotation by an

angle θ.

$$G = \begin{bmatrix} c & s \\ -s & c \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix}$$

If we want to apply this transformation to an $n \times n$ (or even $m \times n$) matrix, we must embed the Givens rotation to be an $n \times n$ (or $m \times m$) matrix. The matrix representation of this is:

$$G_{ij}(\theta) = \begin{bmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & c & & s & & \\ & & & \ddots & & & \\ & & -s & & c & & \\ & & & & & \ddots & \\ & & & & & & 1 \end{bmatrix},$$

where i and j are determined by the entries that are to become zero. Although this representation exists, it isn't necessary to explicitly form this version of the Givens rotation. In fact, it is equivalent to applying to 2×2 Givens rotation to the subset (two rows containing the entry that will become zero and the entry that we will use to create the zero) of the full matrix.

To better understand Givens rotations, we will give a sketch of how they can be used to transform a full 3×3 matrix into an upper triangular matrix. The focus of the sketch will be on the uses of Givens rotations and not the construction of a Givens rotation. Suppose, the 3×3 matrix has random

non-zero elements denoted by the letter "x". Although the same letter is used, the values need not be the same.

$$A = \begin{bmatrix} x & x & x \\ x & x & x \\ x & x & x \end{bmatrix}.$$

Since Givens rotations can be applied to introduce zeros to a matrix, we will systematically pick which of the entries we want to be zero until we get the desired matrix. First, let's start with the entry at $A(3,1)$. The entry at $A(2,1)$ will be used to introduce this first zero. To do this, we will only be looking at the second and third row of A. In this sketch we are rotating the element at i into the element of j and thus creating a zero at j. Applying the first Givens rotation for a suitable θ_k will produce the following:

$$G_{23}(\theta_1)^T A = \begin{bmatrix} x & x & x \\ \bar{x} & \bar{x} & \bar{x} \\ 0 & \bar{x} & \bar{x} \end{bmatrix}.$$

We will denote that the entry has changed from the original entry by writing $\bar{x}$. Subsequent changes to altered entries will not be monitored. We will be creating zeros in the first column and then move onto the second and the third. The next step is to zero out entry $A(2,1)$ by rotating entry $A(1,1)$

into this entry. It is produced by applying $G_{12}(\theta_2)$ in the following manner:

$$G_{12}(\theta_2)^T G_{23}(\theta_1)^T A = \begin{bmatrix} \bar{x} & \bar{x} & \bar{x} \\ 0 & \bar{x} & \bar{x} \\ 0 & \bar{x} & \bar{x} \end{bmatrix}.$$

Now that the first column is reduced, we will move on to the next column. It is important to note that a zero entry rotated into a zero entry will remain zero, a non-zero entry rotated to a zero entry will change the zero entry into a non-zero entry. Thus, it becomes important to choose how one will reduce a matrix wisely. We will follow the same pattern in reducing the subsequent columns as the first column. In our case, there is simply just one more entry to introduce a zero into. The end result will be the following:

$$G_{23}(\theta_3)^T G_{12}(\theta_2)^T G_{23}(\theta_1)^T A = \begin{bmatrix} \bar{x} & \bar{x} & \bar{x} \\ 0 & \bar{x} & \bar{x} \\ 0 & 0 & \bar{x} \end{bmatrix}.$$

The matrix A has been successfully reduced to an upper triangular matrix. The above expression can be re-written to be

$$(G_{23}(\theta_1) G_{12}(\theta_2) G_{23}(\theta_3))^T A = R.$$

Because the product of orthogonal matrices is an orthogonal matrix, we let

$Q = G_{23}(\theta_1)G_{12}(\theta_2)G_{23}(\theta_3)$, and we obtain

$$Q^T A = R \quad \Longrightarrow \quad A = QR.$$

Therefore, we can apply a series of orthogonal matrices to a matrix to create a triangular matrix.

1.3 Updating problem

Now let's introduce the updating problem and how it relates to QR factorization. Although we can use various direct factorization techniques to solve linear systems, it becomes useful to consider a different type of problem where direct factorization may not be the best. These types of problems are updating problems. Updating problems are problems where we already have a matrix that has been factored but want to solve a similar matrix that closely resembles the first matrix but has yet to be factored. One such class of problem is adding a row.

Appending a Row

Suppose we have a matrix A that has already been factored using QR factorization. Also, suppose we want to append a row to the original matrix A. We want to exploit the previously computed QR factorization of A to factor to the new matrix composed of A with an additional row. Here, we do not want to directly compute the factorization of this new matrix since it

closely resembles the original matrix. This problem can be addressed in the following manner:

$$\tilde{A} = \begin{bmatrix} A \\ \mathbf{u}^T \end{bmatrix}.$$

We can note that

$$\begin{bmatrix} Q^T & 0 \\ 0 & 1 \end{bmatrix} \tilde{A} = \begin{bmatrix} Q^T A \\ \mathbf{u}^T \end{bmatrix} = \begin{bmatrix} R \\ \mathbf{u}^T \end{bmatrix}.$$

Since our goal is to find the QR factorization for this new matrix $\tilde{A}$, we need to introduce zeros into the last row such that the matrix will become upper triangular again. We also need to do it in such a way that we produce orthogonal matrices so that we can produce the matrix $\tilde{Q}$. This can be done using Givens rotations. Therefore, using Givens rotations we get

$$G_{(n-1)n}(\theta_n)^T G_{(n-2)n}(\theta_{n-1})^T \dots G_{1n}(\theta_1)^T \begin{bmatrix} R \\ \mathbf{u}^T \end{bmatrix} = \begin{bmatrix} \hat{R} \\ 0 \end{bmatrix} = \tilde{R}.$$

So, we have found a way to reduce our matrix $\tilde{A}$ into an upper triangular matrix $\tilde{R}$ thus providing us with a factorization. Since we were able to exploit the original matrix A's factorization, this generates a more efficient method for computing the decomposition of $\tilde{A}$. Å. Björck's book on matrix computations [2] provides a more in depth discussion on the computational complexity of this method.

Appending a column

The next type of updating problem is where we add a column to A. This is solved similarly to how we solve the appending a row problem. First, suppose we have a matrix A and the QR factorization is already computed. Suppose we are given a new matrix that consists of A along with an extra column:

$$\tilde{A} = \left[\begin{array}{c|c} A & \mathbf{u}^T \end{array}\right].$$

We want to also compute its QR factorization indirectly. We again can use Givens rotations to introduce zeros to the extra column so that we get an upper triangular matrix. Å. Björck's book [2] also provides a detailed explanation of this type of problem and the derivation of the method along with its computational complexity.

Rank-one Update

In this next class of updating problem, we now want to add a rank-one matrix to the original matrix and still exploit the previously computed QR factorization.

$$\tilde{A} = A + \mathbf{w}\mathbf{z}^T,$$

where $\mathbf{w}$, $\mathbf{z}$ are column vectors. We will the discussion and derivation for this class of updating problem until section 3.2.

1.4 Kronecker Product

Now we will look at structured matrices. Particularly, we will consider the Kronecker product. The Kronecker product is a generalized outer product that is performed on two matrices that results in a block matrix.

$$K = A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \dots & a_{1n}B \\ a_{21}B & a_{22}B & \dots & a_{2n}B \\ \vdots & \vdots & & \vdots \\ a_{m1}B & a_{m2}B & \dots & a_{mn}B \end{bmatrix}$$

The Kronecker product is also known as tensor product and it arises naturally in multidimensional data fitting [2]. A more in depth look at the Kronecker product is presented in the following chapter.

Chapter 2

Kronecker Product

Research on the Kronecker product began in the 1800s and now one can easily find the Kronecker product along with its basic properties in most linear algebra literature. Although the notation is first stated to have been used by the German mathematician Johann Georg Zehfuss, it had been credited to Leopold Kronecker. Kronecker presented some of its basic properties to his students in a series of lectures [8]. Independently, near the same time period, Hurwitz and Stéphanos developed some of these properties. A few years after them, Rados also independently published a paper on some of the basic properties of the Kronecker product [8].

Despite the various people contributing to the development of this matrix operation, Leopold Kronecker received the credit and it is now known as the Kronecker product. In this chapter, we will explain in more detail the Kronecker product and a few key properties. We will also briefly discuss

some of the applications of the Kronecker product.

2.1 Definition

The Kronecker product is denoted by $A \otimes B$ and it is computed in the following manner:

$$K = A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \dots & a_{1n}B \\ a_{21}B & a_{22}B & \dots & a_{2n}B \\ \vdots & \vdots & & \vdots \\ a_{m1}B & a_{m2}B & \dots & a_{mn}B \end{bmatrix}.$$

To better understand the definition, here are a couple examples. The first example illustrates how to apply the definition where both A and B are square matrices.

$$\text{If } A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \text{ and } B = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \text{ then}$$

$$A \otimes B = \begin{bmatrix} 1B & 2B \\ 3B & 4B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 2 & 0 \\ 0 & 1 & 0 & 2 \\ 3 & 0 & 4 & 0 \\ 0 & 3 & 0 & 4 \end{bmatrix}.$$

Note that the resulting matrix will have the dimensions 4×4 since A and B were both 2×2 matrices $((2 * 2) \times (2 * 2))$.

The next example will be applying the definition when A is not square.

$$\text{If } A = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 4 & 5 \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \text{ then}$$

$$A \otimes B = \begin{bmatrix} 1B & 2B & 3B \\ 3B & 4B & 5B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 2 & 0 & 3 & 0 \\ 0 & 1 & 0 & 2 & 0 & 3 \\ 3 & 0 & 4 & 0 & 5 & 0 \\ 0 & 3 & 0 & 4 & 0 & 5 \end{bmatrix}.$$

As we can see, the definition is still very easily applied for non-square matrices. In this case, the dimensions of the resulting matrix will be 4×6 $((2 * 2) \times (3 * 2))$. The dimensions of the resulting matrix can begin to grow very quickly as the dimensions of A and B increase.

2.2 Properties

Now we will begin to present some key properties of the Kronecker product without proof. Proofs for each of these properties can be found in A.J. Laub's book [11].

Property 1

The first property is about the transpose of a Kronecker product. This

property is different than one would expect, based on matrix multiplication:

$$(A \otimes B)^T = A^T \otimes B^T. \tag{2.1}$$

Property 2

The inverse of a Kronecker product behaves similar to the transpose:

If A and B are invertible, then $A \otimes B$ is invertible and

$$(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}. \tag{2.2}$$

Property 3

This next property is used often, although it isn't used explicitly.

If A and B are orthogonal, then $A \otimes B$ is also orthogonal. (2.3)

The importance of this property is that now we know that for a given QR factorization for a matrix A that is produced from the Kronecker product, we can express the orthogonal matrix Q as a Kronecker product of two orthogonal matrices.

Property 4

This property is also useful for expressing the QR factorization for a matrix produced in terms of the Kronecker product.

$$(A \otimes B)(C \otimes D) = AC \otimes BD \tag{2.4}$$

For example, if we combine this property along with property 3, we can see that if $K = A \otimes B$ and $K = QR$, then $K = (Q_1 \otimes Q_2)(R_1 \otimes R_2)$.

2.3 Applications

Once the Kronecker product was properly studied, it became easy to find applications for it in various fields. For example, as previously stated, in signal and image processing it became even easier to formulate problems because often times the data naturally formed the Kronecker product. In particular, certain aspects of image deblurring can be expressed using the Kronecker product [7].

Another field where the Kronecker product can be used is in computer vision. In both 2D [12] and 3D [3] computer vision, the introduction of the Kronecker product simplified many well-known facts within that field.

In information theory, there has also been many applications of the Kronecker product that helped develop methods such as covariance estimation [15, 6]. In stability theory, the Kronecker product was used to reformulate well-known equations such as the Sylvester and Lyapunov equations [11].

Lastly, the Kronecker product has even found its way to the natural sciences. There are applications of the Kronecker product in the biological sciences [14] and in neuroscience [1]. The Kronecker product is a compact form that has found its way to various different fields. Therefore, understanding the properties of the Kronecker product is important to recognize

potential applications for it.

Chapter 3

New Preconditioner Based on QR Updating

Although the Kronecker product has many uses in various fields, we will focus on one specific application: image processing. In particular, we will focus on image deblurring. A blurred image $\boldsymbol{b}$ is given, where

$$\boldsymbol{b} = A\boldsymbol{x} + \boldsymbol{\eta},$$

where A is a blurring matrix, $\boldsymbol{x}$ is the true (but unknown) image, and $\boldsymbol{\eta}$ is unknown additive noise. Here we try to solve for $\boldsymbol{x}$. The blurring matrix is defined by:

$$K = A_1 \otimes B_1 + A_2 \otimes B_2 + \cdots + A_n \otimes B_n \tag{3.1}$$

However, a good approximation for equation (3.1) usually ends up being simply the first two terms,

$$K \approx A_1 \otimes B_1 + A_2 \otimes B_2 \,. \tag{3.2}$$

Furthermore, the second term, $B_1 \otimes B_2$ can be approximated with rank-one matrices so that equation (3.2) becomes

$$K \approx A = A_1 \otimes B_1 + \boldsymbol{w}\boldsymbol{z}^T, \tag{3.3}$$

where $\boldsymbol{w} = \boldsymbol{w}_1 \otimes \boldsymbol{w}_2$ and $\boldsymbol{z} = \boldsymbol{z}_1 \otimes \boldsymbol{z}_2$. Now equation (3.3) provides us with a rank-one matrix and a Kronecker product where we can apply the updating problem.

3.1 Tikhonov Regularization

For ill-posed problems (problems where the solution does not depend continuously on the data), it becomes difficult to solve the least squares problem (defined in equation (1.1)) since noise in the data is amplified in the solution [2]. Equation (3.3) turns out to be a discrete ill-posed problem and so it becomes helpful to reformulate the problem in a different manner.

Tikhonov regularization is used to accomplish this. Tikhonov regulariza-

tion is expressed in the damped least squares problem

$$\min_{\boldsymbol{x}} \left\{ ||A\boldsymbol{x} - \boldsymbol{b}||_2^2 + \lambda^2 ||\boldsymbol{x}||_2^2 \right\}. \tag{3.4}$$

The regularization parameter λ affects the smoothness of the solution. λ is a scalar between 0 and 1. When λ is 0, we go back to least squares problem. There are many methods to compute this constant **??**. The solution to problem (3.4) is equivalent to

$$\min_{\boldsymbol{x}} \left\| \begin{bmatrix} A \\ \lambda I \end{bmatrix} \boldsymbol{x} - \begin{bmatrix} \boldsymbol{b} \\ \boldsymbol{0} \end{bmatrix} \right\|_2^2. \tag{3.5}$$

which is an overdetermined least squares problem. We can express problem (3.5) using equation (3.3) as

$$\min_{\boldsymbol{x}} \left\| \begin{bmatrix} A_1 \otimes A_2 + \boldsymbol{w}\boldsymbol{z}^T \\ \lambda I \end{bmatrix} \boldsymbol{x} - \begin{bmatrix} \boldsymbol{b} \\ \boldsymbol{0} \end{bmatrix} \right\|_2^2. \tag{3.6}$$

Once we find the solution to equation (3.6), we will find the true image to image deblurring problem.

There are many techniques available to solve equation (3.6). One commonly used method is the QR factorization (section 1.1), however when considering extremely large matrices, QR factorization may not be the most efficient. We will use LSQR [13] which is an algorithm recommended for

large least squares problems. MATLAB has the built-in function `lsqr` which we will use in all of our experiments.

3.2 Rank-One Updating Problem

In order for us to efficiently compute the solution to equation (3.6) using LSQR, we need to continue developing the rank-one updating problem for Kronecker products. As it was previously stated in section 1.3, the rank-one updating problem [4] is stated as computing the QR factorization for a matrix that consists of a rank-one matrix added to another matrix (we will call this matrix A). Suppose we have the QR factorization for the original matrix, we want to still exploit the previously computed QR factorization.

$$\tilde{A} = A + \mathbf{w}\mathbf{z}^T,$$

where $\mathbf{w}$, $\mathbf{z}$ are column vectors. The first step is to re-write the stated problem using the information provided (QR factorization of the original matrix A).

$$\tilde{A} = A + \mathbf{w}\mathbf{z}^T = QR + \mathbf{w}\mathbf{z}^T = Q[R + Q^T\mathbf{w}\mathbf{z}^T].$$

We focus on the second term within the brackets ($Q^T\mathbf{w}\mathbf{z}^T$) and work to make it upper triangular. This can be done easily by reducing $\mathbf{w}$ into the

standard unit vector $\mathbf{e}_1$ using Givens rotations which results in

$$\tilde{A} = Q[R + \overline{Q}Q^T\mathbf{w}\mathbf{z}^T] = Q\overline{Q}[\overline{Q}^T R + c\mathbf{e}_1\mathbf{z}^T].$$

Upon inspection, we notice that $\overline{Q}^T R$ is upper Hessenberg. An upper Hessenberg matrix is an upper triangular matrix where the subdiagonal below the major diagonal does not consist of only zeros and an upper Hessenberg matrix is denoted as H. What now remains is an upper Hessenberg matrix added to a sparse matrix, $c\mathbf{e}_1\mathbf{z}^T$, with nonzero entries only on the first row and the remaining entries are zeros. Adding these two matrices together will result in another upper Hessenberg matrix. This can now be transformed to the desired upper triangular form using Givens rotations. This results in the following:

$$\tilde{A} = Q\overline{Q}[H] = Q\overline{Q}\hat{Q}\tilde{R} = \tilde{Q}\tilde{R}$$

where $\tilde{Q} = Q\overline{Q}\hat{Q}$.

3.3 Rank-One Update for Kronecker Products

We can follow the same scheme for Kronecker products as developed in section 3.2 to extend the updating problem to Kronecker products.

Suppose we are given matrices A_1 and A_2 and their corresponding QR

factorizations,

$$A = A_1 \otimes A_2 + \mathbf{w}\mathbf{z}^T = (Q_1 \otimes Q_2)(R_1 \otimes R_2) + (\mathbf{w}_1 \otimes \mathbf{w}_2)(\mathbf{z}_1 \otimes \mathbf{z}_2)^T.$$

Normally, we would like to compute the complete QR factorization of A efficiently. However, since we are focusing on solving the damped least squares problem, we will be more concerned about using the information the updating scheme can provide. Therefore we will modify the scheme slightly.

This problem is restated as

$$A = (Q_1 \otimes Q_2)[(R_1 \otimes R_2) + (Q_1^T \otimes Q_2^T)(\mathbf{w}_1 \otimes \mathbf{w}_2)(\mathbf{z}_1 \otimes \mathbf{z}_2)^T]. \tag{3.7}$$

The first step is to reduce the second term into an upper triangular matrix. This is accomplished in the same manner using Givens rotations

$$A = (Q_1 \otimes Q_2)(\bar{Q_1} \otimes \bar{Q_2})[(\bar{Q_1}^T \otimes \bar{Q_2}^T)(R_1 \otimes R_2) + v(\mathbf{e}_1 \otimes \mathbf{e}_1)(\mathbf{z}_1 \otimes \mathbf{z}_2)^T], \tag{3.8}$$

where v is a scalar. Note that the first term can be rewritten so that the problem now becomes

$$A = (\tilde{Q}1 \otimes \tilde{Q}_2)[(H_1 \otimes H_2) + v(\mathbf{e}_1 \otimes \mathbf{e}_1)(\mathbf{z}_1 \otimes \mathbf{z}_2)^T]. \tag{3.9}$$

Note that H_1 and H_2 are upper Hessenberg matrices. From here, rather than continuing, we will stop and resume consideration of the damped least

squares problem.

3.4 LSQR in Image Deblurring

LSQR is an iterative algorithm for computing solutions to linear least squares problems. Since it is an iterative algorithm, the cost of computing the solutions is dependent on the cost of each iteration and the number of iterations. Therefore, if we can increase the convergence rate, this method can provide an efficient way to solve the damped least squares problem.

The acceleration of convergence is accomplished by using preconditioners. In order for us to solve the image deblurring problem, we must first find an efficient preconditioner and then solve the problem using LSQR with the provided preconditioner. Let's first focus on the preconditioner. We want to find a matrix M that has the following properties:

- M can be computed efficiently.
- Solving linear systems with M and M^T can be done efficiently.
- M has the property that $M^TM \approx A^TA + \lambda^2 I$. Ideally, if $M^TM - (A^TA + \lambda^2 I)$ is a matrix of rank r, then LSQR will converge in at most r iterations.

The idea is that we want to find a preconditioner that speeds up the convergence enough that it mitigates the additional costs associated with the preconditioner.

We will use the rank-one updating scheme for constructing the preconditioner, in particular we will use the matrices H_1 and H_2 and the scalar v. Consider the singular value decomposition [5] of H_1 and H_2:

$$H_1 = U_1 \Sigma_1 V_1^T \quad \text{and} \quad H_2 = U_2 \Sigma_2 V_2^T .$$

We will use the following as our preconditioner:

$$M = D(V_1 \otimes V_2)^T , \quad \text{where } D = (\Sigma_1^2 \otimes \Sigma_2^2 + \lambda^2 I)^{1/2} .$$

Notice that

$$\begin{aligned}
M^T M &= [D(V_1 \otimes V_2)^T]^T D(V_1 \otimes V_2)^T \\
&= (V_1 \otimes V_2) D D (V_1 \otimes V_2)^T \\
&= (V_1 \otimes V_2)(\Sigma_1^2 \otimes \Sigma_2^2 + \lambda^2 I)(V_1 \otimes V_2)^T \\
&= (V_1 \otimes V_2)(\Sigma_1^2 \otimes \Sigma_2^2 + \lambda^2 (I_1 \otimes I_2))(V_1 \otimes V_2)^T \\
&= V_1 \Sigma_1^2 V_1^T \otimes V_2 \Sigma_2^2 V_2^T + \lambda^2 V_1 V_1^T \otimes V_2 V_2^T \\
&= V_1 \Sigma_1^T U_1^T U_1 \Sigma_1 V_1^T \otimes V_2 \Sigma_2^T U_2^T U_2 \Sigma_2 V_2^T + \lambda^2 I \\
&= H_1^T H_1 \otimes H_2^T H_2 + \lambda^2 I \\
&= (H_1 \otimes H_2)^T (H_1 \otimes H_2) + \lambda^2 I \\
&= H^T H + \lambda^2 I .
\end{aligned}$$

Now if $A = A_1 \otimes A_2 + \boldsymbol{w}\boldsymbol{z}^T$, and $H = H_1 \otimes H_2$, observe that using what our

rank-one updating scheme produced in equation (3.9) we get

$$\begin{aligned} A^TA &= [H^T + v(\mathbf{z}_1\mathbf{e}_1^T \otimes \mathbf{z}_2\mathbf{e}_1^T)]Q^TQ[H + v(\mathbf{e}_1 \otimes \mathbf{e}_1)\mathbf{z}^T] \\ &= H^TH + v(H_1^T\mathbf{e}_1\mathbf{z}_1^T \otimes H_2^T\mathbf{e}_1\mathbf{z}_2^T) \\ &\qquad + v(\mathbf{z}_1\mathbf{e}_1^TH_1 \otimes \mathbf{z}_2\mathbf{e}_1^TH_2) + v^2(\mathbf{z}_1\mathbf{e}_1^T\mathbf{e}_1\mathbf{z}_1^T \otimes \mathbf{z}_2\mathbf{e}_1^T\mathbf{e}_1\mathbf{z}_2^T) \\ &= H^TH + R, \end{aligned}$$

where R is the sum of the three remaining rank-one matrices. Now if we add $\lambda^2 I$ to both sides,

$$\begin{aligned} A^TA + \lambda^2 I &= H^TH + \lambda^2 I + R \\ &= M^TM + R \end{aligned}$$

and we satisfy one of the desired properties for a preconditioner. Moreover, we can show that rank$(R) \leq 3$. Recall that rank$(A \otimes B)$ =rank(A)rank(B) [10] and that rank$(A + B) \leq$ rank(A)+rank(B) [9]. Since

$$(A^TA + \lambda^2 I) - M^TM = R$$

and since rank$(R) \leq 3$, if we use LSQR with this preconditioner, we will reach its stopping criteria in at most three iterations.

Furthermore, M can be computed efficiently using our rank-one updating scheme for Kronecker products. Lastly, solving linear systems with M and

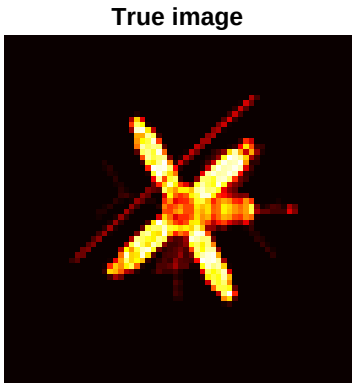
M^T can be done efficiently since V_1 and V_2 are both orthogonal and since D is a diagonal matrix. Therefore, we have satisfied the three conditions and M is indeed a good preconditioner.

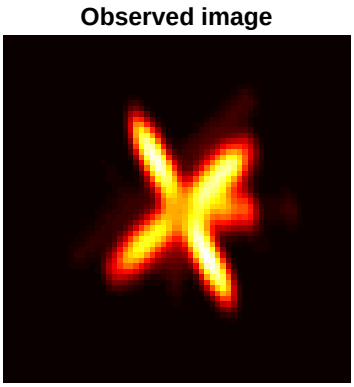
Chapter 4

Numerical Results

Now that the method for efficiently computing the solution of a regularized least squares problem using the Kronecker product has been established, we will present a few experiments. For each experiment we are comparing between using LSQR with no preconditioner versus LSQR with our defined preconditioner.

The first experiment was for a blurred image of a satellite, where A is a matrix of size $4,096 \times 4,096$. The blurred image and the true image are shown in Figure 4.1. The matrix A was approximated from a point spread function and then used in an attempt to reconstruct the original image. With the use of our preconditioner, the reconstructed image (Figure 4.2) was computed. Using our preconditioner, LSQR reached a convergence with significantly fewer iterations than without a preconditioner, as expected. A plot of the relative error at each iteration is found in Figure 4.3.





True image. Blurred image.

Figure 4.1: True image of satellite and blurred image of satellite of size 64×64. The true image was blurred and then the blurred image was passed to our algorithm in an attempt to reconstruct the original image.

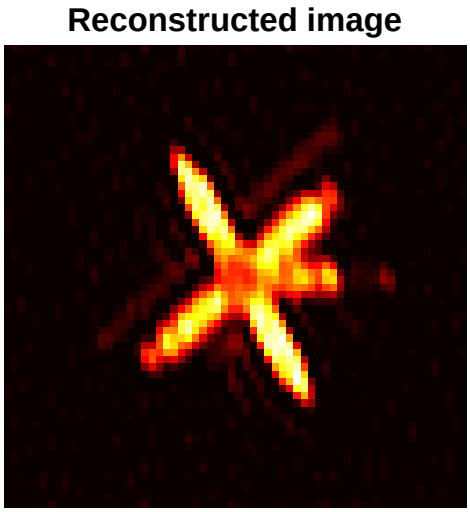


Figure 4.2: Reconstructed image satellite when using LSQR with our preconditioner on the blurred image of size 64×64.

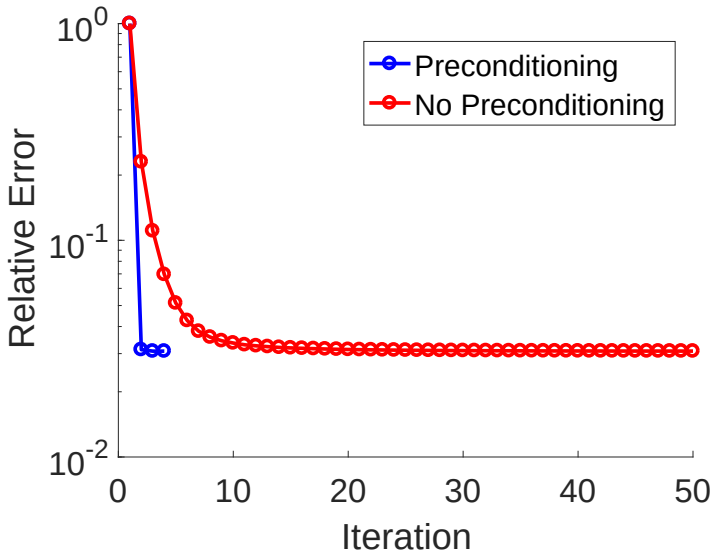
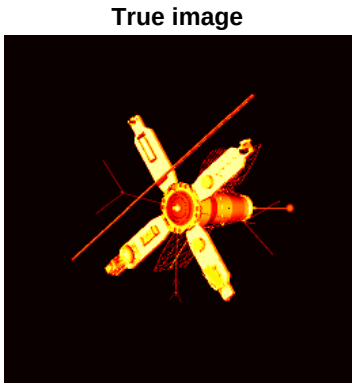
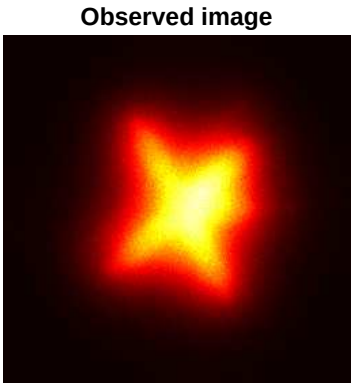


Figure 4.3: Plot of normalized relative error at each iteration when using LSQR with our preconditioner on the blurred image of size 64×64.

Now, we will repeat the same experiment but with a 256×256 satellite image having a matrix A of size $65,536 \times 65,536$. Figure 4.4 is the true (original) image and the blurred image. Similarly, the blurred image in Figure 4.4 was approximated and the reconstructed image was computed using LSQR with our preconditioner. Again, convergence was reached within 3 iterations as expected. The plot of the relative error is found in Figure 4.6.

Both experiments support our claim that LSQR is guaranteed to terminate within 3 iterations. Also, the experiments seem to suggest that LSQR with our preconditioner is indeed faster even though extra costs are incurred by having to compute the preconditioner and solve a linear system containing the preconditioner for each iteration. This is verified by the timings shown





True image. Blurred image.

Figure 4.4: True image of satellite and blurred image of satellite of size 256×256. The true image was blurred and then the blurred image was passed to our algorithm in an attempt to reconstruct the original image.

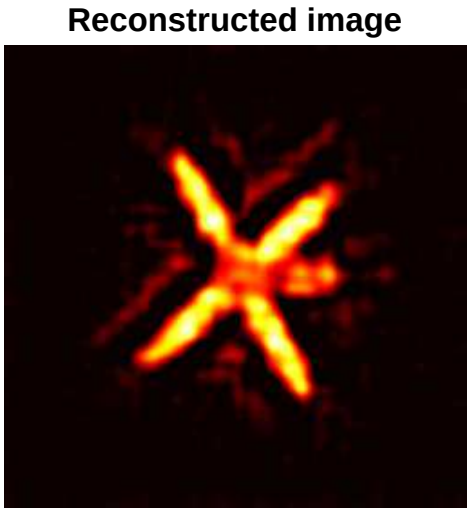


Figure 4.5: Reconstructed image satellite when using LSQR with our preconditioner on the blurred image of size 256×256.

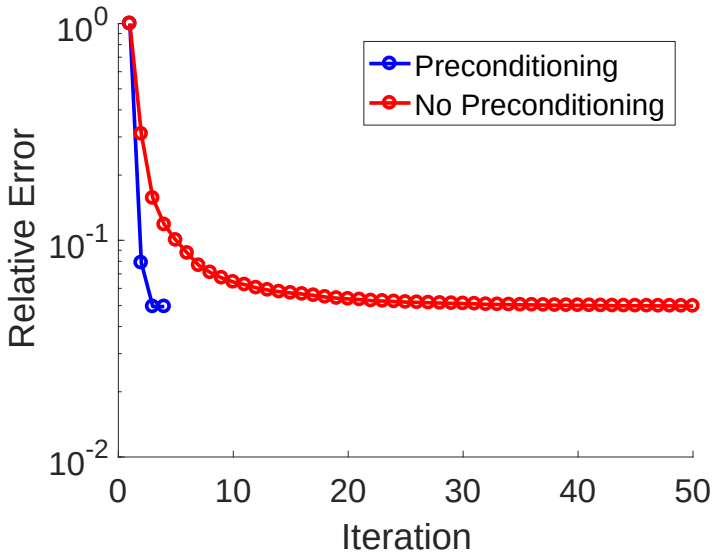


Figure 4.6: Plot of normalized relative error at each iteration when using LSQR with our preconditioner on the blurred image of size 256×256.

in Table 4.1.

	Preconditioner			**No Preconditioner**		
size	Time (s)	rel.error	# Itr.	Time (s)	rel.error	# Itr.
$4,096$	0.0707	$8.2326 \cdot 10^{-5}$	3	0.5155	$8.2369 \cdot 10^{-5}$	50
$65,536$	0.5140	$3.5307 \cdot 10^{-4}$	3	11.5040	$3.5505 \cdot 10^{-4}$	50

Table 4.1: For both sizes of matrices, and for LSQR with both preconditioner and no preconditioner, table of average run time, relative error, and number of iterations. Averages over 50 runs.

Chapter 5

Conclusion

In image deblurring problems, we would like to solve the true or original image $\boldsymbol{x}$ from a blurred image $\boldsymbol{b}$

$$\boldsymbol{b} = A\boldsymbol{x} + \boldsymbol{\eta},$$

where A is a matrix that models the blurring operation. Since A is very large, it can be difficult to find the solution using direct techniques.

When we are considering large matrices that construct the blurred image, iterative methods become a good choice. In particular, we can use LSQR to find a solution to the ill-posed image deblurring problem. In order for us to construct a competitive method, we found an efficient preconditioner for LSQR so that there will be at most three iterations. We have found that using the rank-one updating scheme for Kronecker products provides us with

an efficient preconditioner. Using this preconditioner provides us with a fast and accurate method compared to using no preconditioner.

The numerical experiments that were conducted support our claim of an efficient preconditioner. This leads us to believe that this method is very promising. There are many avenues to explore for future research. For example, extending our approach to rank-k modifications, where $k > b$.

Bibliography

[1] Fetsje Bijma, Jan C. de Munck, and Rob M. Heethaar. The spatiotemporal meg covariance matrix modeled as a sum of kronecker products. *NeuroImage*, 27, 2005.

[2] Ake Bjorck. *Numerical Methods in Matrix Computations.* Springer, Cham, 2015.

[3] Andrea Fusiello. A matter of notation: Several uses of the kronecker product in 3d computer vision. *Pattern Recognition Letters*, 28:2127–2132, 2007.

[4] Philip E. Gill, Walter Murray, and Margaret H. Wright. *Numerical Linear Algebra and Optimization*, volume 1. Addison-Wesley Publishing Company, Redwood City, California, 1996.

[5] Gene H. Golub and Charles F. Van Loan. *Matrix Computations.* Johns Hopkins University Press, Baltimore, third edition, 1996.

[6] Kristjan Greenewald, Theodoros Tsiligkaridis, and Alfred O. Hero III. Kronecker sum decompositions of space-time data. In *5th IEEE International Workshop on Computational Advances in Multi-Sensor Adaptive Processing*, 2013.

[7] Per Christian Hansen, James G. Nagy, and Dianne P. O'Leary. *Deblurring Images: Matrices, Spectra, and Filtering*. SIAM: Fundamentals of Algorithms, 2006.

[8] H. V. Henderson, F. Pukelsheim, and S.R. Searle. On the history of the kronecker product. *Linear and Multilinear Algebra*, 14:113–120, 1983.

[9] Roger A. Horn and Charles R. Johnson. *Matrix Analysis*. Cambridge, 1985.

[10] Roger A. Horn and Charles R. Johnson. *Topics in Matrix Analysis*. Cambridge, 1991.

[11] Alan J. Laub. *Matrix Analysis for Scientists and Engineers*. SIAM, 2005.

[12] Juan Liu, Emmanouil Psarakis, and Ioannis Stamos. Automatic kronecker product model based detection of repeated patterns in 2d urban images. In *International Conference on Computer Vision*, 2013.

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

[14] D. V. Savostyanov, S. V. Dolgov, J. M. Werner, and Ilya Kuprov. Exact nmr simulation of protein-size spin systems using tensor train formalism. *PHYSICAL REVIEW B 90, 085139*, 2014.

[15] Theodoros Tsiligkaridis and Alfred O. Hero III. Covariance estimation in high dimensions via kronecker product expansions. *arXiv:1302.2686v10 [stat.ME]*, 2013.