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.

Signature:

Yizhou Chen April 01, 2024

Fast Mixed Precision Algorithms for Toeplitz Least Squares Problems

By

Yizhou Chen

James G. Nagy, Ph.D.
Advisor

Department of Mathematics

James G. Nagy, Ph.D.
Advisor

Julianne Chung, Ph.D.
Committee Member

Michael Rogers, Ph.D.
Committee Member

2024

Fast Mixed Precision Algorithms for Toeplitz Least Squares Problems

By

Yizhou Chen

James G. Nagy, Ph.D.
Advisor

An abstract of
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
Bachelor of Science with Honors

Department of Mathematics

2024

Abstract

Fast Mixed Precision Algorithms for Toeplitz Least Squares Problems
By Yizhou Chen

This thesis presents a fast mixed precision algorithm to solve least squares problems $\min \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2 + \alpha^2\|\boldsymbol{x}\|_2^2$ when $\boldsymbol{A}$ is a Teoplitz matrix (α could be zero for general problems), which arise in many applications like signal deblurring. This algorithm utilizes the special structure of the Toeplitz matrix to construct the Cholesky factorization of matrix $\boldsymbol{A}^\top\boldsymbol{A}$ in lower precision to solve for $\boldsymbol{x}$ and executes GMRES iterative refinement in higher precision to improve the accuracy of the solution. It is found that the precision used for the computation of the Cholesky factorization does not affect the accuracy of the result in cases when $\boldsymbol{b}$ is corrupted by noise. Using double precision in both refinement and calculating the residual is the most efficient, requiring only 1 refinement iteration. However, using single precision for refinement and double for calculating the residual can reach the same level of accuracy with one or two refinement iteration(s).

Fast Mixed Precision Algorithms for Toeplitz Least Squares Problems

By

Yizhou Chen

James G. Nagy, Ph.D.
Advisor

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
Bachelor of Science with Honors

Department of Mathematics

2024

Acknowledgments

I'm extremely grateful to my amazing advisor Dr. James Nagy. This thesis would not have been possible without his support and guidance. I would like to extend my sincere thanks to my family and friends, who encouraged me when I felt stuck. At last, I want to thank Dr. Julianne Chung and Dr. Michael Rogers, who provided valuable advice on my thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this thesis we consider least squares problems

$$\min \|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}\|_2^2 \tag{1.1}$$

and

$$\min \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2 + \alpha^2 \|\boldsymbol{x}\|_2^2 = \min \left\| \begin{bmatrix} \boldsymbol{b} \\ \boldsymbol{0} \end{bmatrix} - \begin{bmatrix} \boldsymbol{A} \\ \alpha \boldsymbol{I} \end{bmatrix} \boldsymbol{x} \right\|_2^2 \tag{1.2}$$

where $\boldsymbol{b} \in \mathbb{R}^m$, $\boldsymbol{x} \in \mathbb{R}^n$, α is a scalar, $\boldsymbol{I} \in \mathbb{R}^{n \times n}$ is the identity matrix, and $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ $(m \geq n)$ is a Toeplitz matrix; that is, a matrix with the following structure

$$\boldsymbol{A} = \begin{bmatrix} a_0 & a_{-1} & \cdots & a_{-(n-1)} \\ a_1 & a_0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & a_{-1} \\ \vdots & & \ddots & a_0 \\ \vdots & & & a_1 \\ \vdots & & & \vdots \\ a_{m-1} & \cdots & \cdots & a_{m-n} \end{bmatrix}.$$

Throughout this thesis we refer to equation (1.1) as a Toeplitz least squares problem, and equation (1.2) as a regularized Toeplitz least squares problem. The regularized version arises in applications where $\boldsymbol{A}$ is ill-conditioned and there are unknown errors or noise in the entries of $\boldsymbol{b}$. The scalar α is added to balance the residual norm and the $\boldsymbol{x}$ norm, with larger α favoring small $\boldsymbol{x}$ norm solutions and small α favoring small residual norm solutions. Least squares problems involving Toeplitz matrices arise in many applications, including signal or image deblurring [8], [11], and seismic tomography [3].

There are many efficient algorithms to solve Toeplitz least squares problems, including approaches to compute QR factorizations of $\boldsymbol{A}$, and preconditioned iterative algorithms [3]. Others [2] have done mixed precision iterative refinement to further improve the accuracy for least squares problem with well-conditioned matrices, but no work, at least to our knowledge, is done for ill-conditioned problems that arise from discretization of inverse problems. Moreover, the mixed precision refinement method has not been explored for structured linear systems. In this thesis we develop new efficient mixed precisions algorithms, exploiting the Toeplitz structure and low-precision computations to compute a partial QR factorization, and iterative refinement in high-precision to ensure accuracy.

This thesis is structured as follows. In Chapter 2 we describe a motivational example: the discretization of deconvolution into a least squares problem and reasons why regularization is needed. Chapter 3 introduces least squares problems in detail and approaches to solve them. We also discuss techniques used in a later Chapter to solve Toeplitz least squares problems, like Givens rotations, and the standard refinement process to improve the solution accuracy. Chapter 4 discusses the methods for solving Toeplitz least squares specifically, i.e. constructing Cholesky factorization of $\boldsymbol{A}^\top \boldsymbol{A}$ and also presented some challenges of the method. Chapter 5 presents the results of some numerical experiments for the algorithm for solving Toeplitz least

squares problems, and Chapter 6 present concluding remarks.

Chapter 2

Deconvolution

In this chapter we describe the important application of deconvolution to motivate the need to develop efficient algorithms for Toeplitz least squares problems, and why regularization may be needed in some applications. The convolution of two functions a and x is written as

$$b(t) = \int_{-\infty}^{\infty} a(s-t)x(s)ds\,. \tag{2.1}$$

In many cases the kernel function $a(s-t)$ decays to zero on its tail ends (e.g., like a Gaussian function), in which case one can consider a finite interval of integration,

$$b(t) = \int_{c}^{d} a(s-t)x(s)ds\,. \tag{2.2}$$

For the problem of convolution we are given functions a and x, and we want to find the function b. Deconvolution is an inverse problem, where we are given a and b, and we want to find x. Deconvolution is much more difficult than convolution. In most practical situations we cannot compute x using analytical procedures, and we need to rely on numerical approximations.

Generally, we use a quadrature rule to approximate the integration,

$$\int_c^d a(s-t)x(s)ds \approx \sum_{j=1}^{n} w_j a(s_j - t)x(s_j)\,, \tag{2.3}$$

where $s_j \in [c, d]$ are nodes of the division for the area and w_j are the corresponding weights [9]. Note that we also cannot represent $x(s_j)$ exactly, which leads to the approximation

$$\sum_{j=1}^{n} w_j a(s_j - t)x(s_j) \approx b(t). \tag{2.4}$$

In the interval $[c, d]$, there are n sub-intervals and here we assume each sub-interval has equal length. Therefore, each sub-interval has length $\frac{d-c}{n}$. If, more specifically, the mid-point rule is applied to discretize equation (2.2), we will have the set of points:

$$\begin{aligned}
s_1 &= \tfrac{1}{2}\tfrac{d-c}{n} \\
s_2 &= \tfrac{1}{2}\tfrac{d-c}{n} + \tfrac{d-c}{n} \\
&\vdots \\
s_j &= \tfrac{1}{2}\tfrac{d-c}{n} + (j-1)\tfrac{d-c}{n} \\
&\vdots \\
s_n &= \tfrac{1}{2}\tfrac{d-c}{n} + (n-1)\tfrac{d-c}{n}\,.
\end{aligned}$$

Equation (2.4) then becomes:

$$b(t) \approx \sum_{j=1}^{n} \frac{d-c}{n} a(s_j - t)x(s_j)\,. \tag{2.5}$$

We can also discretize equation (2.2) in terms of t in the same way, where a set of points $t_1, t_2, \cdots, t_m$ is obtained with $t_i = \frac{1}{2}\frac{d-c}{n} + (i-1)\frac{d-c}{n}$. Therefore, a system of

linear equations can be achieved:

$$\begin{aligned}
\frac{d-c}{n}\left(a(s_1-t_1)x(s_1)+a(s_2-t_1)x(s_2)+\cdots+a(s_n-t_1)x(s_n)\right) &= b(t_1)\\
\frac{d-c}{n}\left(a(s_1-t_2)x(s_1)+a(s_2-t_2)x(s_2)+\cdots+a(s_n-t_2)x(s_n)\right) &= b(t_2)\\
&\vdots\\
\frac{d-c}{n}\left(a(s_1-t_m)x(s_1)+a(s_2-t_m)x(s_2)+\cdots+a(s_n-t_m)x(s_n)\right) &= b(t_m)\,.
\end{aligned}$$

This can be written into a matrix-vector multiplication as

$$\frac{d-c}{n}\begin{bmatrix} a(s_1-t_1) & a(s_2-t_1) & \cdots & a(s_n-t_1)\\ a(s_1-t_2) & a(s_2-t_2) & \cdots & a(s_n-t_2)\\ \vdots & & & \\ a(s_1-t_m) & a(s_2-t_m) & \cdots & a(s_n-t_m)\end{bmatrix}\begin{bmatrix} x(s_1)\\ x(s_2)\\ \vdots\\ x(s_n)\end{bmatrix}=\begin{bmatrix} b(t_1)\\ b(t_2)\\ \vdots\\ b(t_m)\end{bmatrix}, \quad (2.6)$$

which is the linear system $\boldsymbol{b}=\boldsymbol{A}\boldsymbol{x}$, with

$$\begin{aligned}
a_{ij} &= (\tfrac{d-c}{n})a(s_j-t_i) = (\tfrac{d-c}{n})a\left((j-i)\tfrac{d-c}{n}\right)\\
x_i &= x(s_j) \qquad\qquad , i\in[1,2,\cdots,m]\,, j\in[1,2,\cdots,n]\,.\\
b_i &= b(t_i)
\end{aligned}$$

$\boldsymbol{A}\in\mathbb{R}^{m\times n}$, $\boldsymbol{x}\in\mathbb{R}^n$, and $\boldsymbol{b}\in\mathbb{R}^m$.

Since the deconvolution problem can be approximated by a set of linear equations, one only needs to solve the system $\boldsymbol{b}=\boldsymbol{A}\boldsymbol{x}$ for $\boldsymbol{x}$, after computing matrix $\boldsymbol{A}$ from the kernel function.

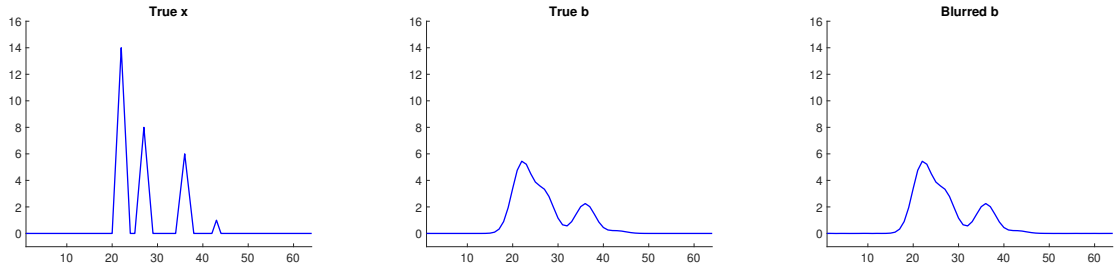
A simple example of such a problem is shown, which is also used for testing the new mixed precision algorithms. The problem is similar to a signal deblurring problem, where the goal is to reconstruct the sharp image from a blurred one. In this

example, the blur is caused by atmospheric turbulence and the kernel is described as a Gaussian function

$$k = \exp\left(-\frac{1}{2}\left(\frac{X}{s_1}\right)^2 - \frac{1}{2}\left(\frac{Y}{s_2}\right)^2\right), \tag{2.7}$$

where s_1 is the standard deviation of the Gaussian along the horizontal direction and s_2 is the standard deviation of the Gaussian along the vertical direction [8]. The true solution $\boldsymbol{x}$ is constructed, and the kernel function is evaluated at various points depending on the size of $\boldsymbol{x}$. Then, the Toeplitz matrix $\boldsymbol{A}$ is built using the kernel function. At last, $\boldsymbol{b}$ without any noise is computed by multiplying the $\boldsymbol{A}$ with the true solution $\boldsymbol{x}$.

A Matlab built-in function \ can be used to solve the system of equations, $\boldsymbol{x} = \boldsymbol{A}\backslash\boldsymbol{b}$. A one dimensional example of signal deblurring problem is used here as an illustration. Figure 2.1 shows the true signal $\boldsymbol{x}_{\text{true}}$, the true right-hand-side $\boldsymbol{b}$, and the right-hand-side with 0.1% noise. If no noise is introduced in $\boldsymbol{b}$, the relative error



(a) (b) (c)

Figure 2.1: (a)The true signal; (b) True right-hand-side; (c) Blurred Measurements.

between the computed solution and true solution:

$$\frac{\|x_{\text{com}} - x_{\text{true}}\|_2^2}{\|x_{\text{true}}\|_2^2}, \tag{2.8}$$

is approximately $3.96{*}10^{-9}$. Figure 2.2 shows the true signal and the solution acquired

using \ on true $\boldsymbol{b}$, where they largely overlap.

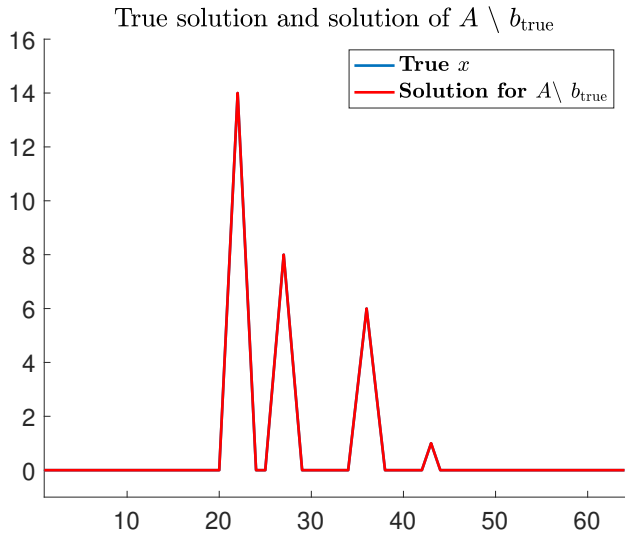
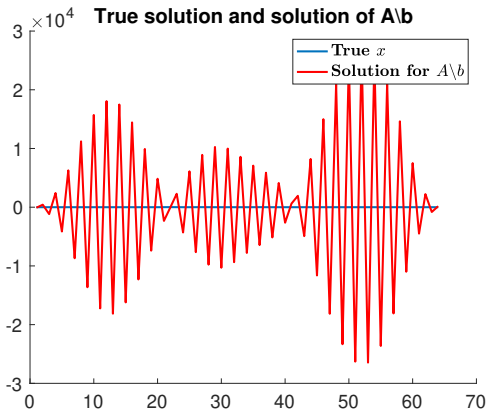
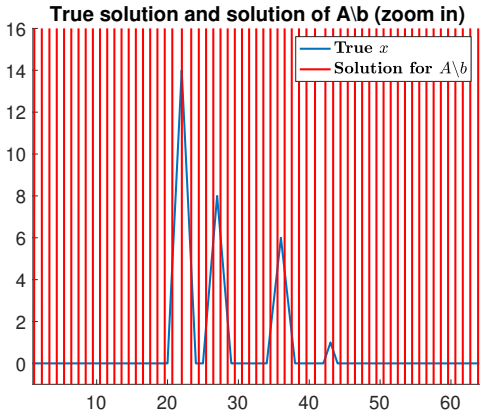


Figure 2.2: The solution acquired using backslash on true $\boldsymbol{b}$.

However, if additive noise is included in $\boldsymbol{b}$, even it is small, the built-in function performs badly. That is, if we attempt to solve $\boldsymbol{Ax} = \hat{\boldsymbol{b}}$, where $\hat{\boldsymbol{b}} = \boldsymbol{b} + \boldsymbol{e}$ with $\frac{\|\boldsymbol{e}\|_2}{\|\boldsymbol{b}\|_2} = 0.001$, (i.e. 0.1% noise), then the relative forward error becomes $4.90 * 10^3$. Figure 2.3 shows the true signal and the solution acquired by \ on noisy $\boldsymbol{b}$.



(a)



(b)

Figure 2.3: (a)The solution using \ on noisy $\boldsymbol{b}$; (b) Zoomed in version of the solution using \ on noisy $\boldsymbol{b}$.

If there is unknown noise in $\boldsymbol{b}$, regularization can be used to mitigate the influence caused by the noise. Specifically, $\boldsymbol{A}$ is amended to $\begin{bmatrix} \boldsymbol{A} \\ \alpha \boldsymbol{I} \end{bmatrix}$, where α is the regularization parameter, and $\boldsymbol{b}$ is padded with zeros to correct the dimension. With regularization and $\alpha = 0.001$, the relative forward error is reduced to 0.1396. Figure 2.4 shows the true signal and the solution acquired by \ after regularization.

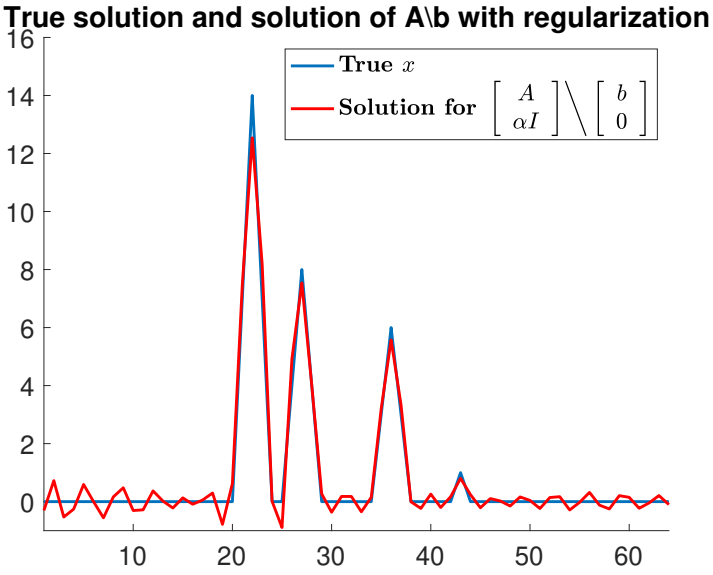


Figure 2.4: The solution acquired using \ on noisy $\boldsymbol{b}$ after regularization, with $\alpha = 0.001$.

However, since matrix $\boldsymbol{A}$ is a Toepltiz matrix, more efficient algorithms can be implemented utilizing its special structure, like computing QR factorization and Cholesky factorization. Therefore, in Chapter 3, the general idea of the efficient algorithm is explained and illustrated.

Chapter 3

Least Squares Problems

For the linear system $\boldsymbol{Ax} = \boldsymbol{b}$ acquired from deconvolution, most of the time, matrix $\boldsymbol{A}$ is rectangular, meaning that the number of equations is different from the number of unknowns. Therefore, $\boldsymbol{b}$ may not in the column space of $\boldsymbol{A}$, showing that the linear system does not have a solution. Thus, the goal is to find $\boldsymbol{x}$ that minimizes the distance between $\boldsymbol{Ax}$ and $\boldsymbol{b}$, i.e. the residual $\boldsymbol{r} = \boldsymbol{b} - \boldsymbol{Ax}$, and this type of problem is called the least squares problem. The objective function is equation (1.1).

In this chapter, we review techniques to solve least squares problems, with a particular focus on Givens rotations. We also discuss rank-1 updating and downdating problems, and iterative refinement, which will be needed in Chapter 4.

3.1 Methods to solve least squares problems

There exist many techniques to solve the least squares problem: using normal equations, using the QR decomposition, and using the Singular Value Decomposition (SVD) factorization. In this thesis, we focus on using normal equations. Equation (1.1) can be written as:

$$f(\boldsymbol{x}) = \|\boldsymbol{b} - \boldsymbol{Ax}\|_2^2 = (\boldsymbol{b} - \boldsymbol{Ax})^\top (\boldsymbol{b} - \boldsymbol{Ax})\,. \tag{3.1}$$

To find the minimum point of function f, we find the $\boldsymbol{x}$ that sets the gradient of the function, $\bigtriangledown f$, to 0.

$$\bigtriangledown f(\boldsymbol{x}) = \boldsymbol{A}^\top \boldsymbol{b} - \boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{x} = 0\,. \tag{3.2}$$

Therefore, we need to solve the equation

$$\boldsymbol{A}^\top \boldsymbol{b} = \boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{x}\,, \tag{3.3}$$

which can be easily solved using the QR factorization of $\boldsymbol{A}$.

The QR factorization is defined in such way that for $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, there exist an orthogonal matrix $\boldsymbol{Q} \in \mathbb{R}^{m\times m}$ and an upper triangular matrix $\boldsymbol{R} \in \mathbb{R}^{m\times n}$ such that [15]

$$\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{R}. \tag{3.4}$$

From equation (3.4), we can compute the Cholesky factorization of $\boldsymbol{A}^\top \boldsymbol{A}$:

$$\boldsymbol{A}^\top \boldsymbol{A} = \boldsymbol{R}^\top \boldsymbol{Q}^\top \boldsymbol{Q}\boldsymbol{R} = \boldsymbol{R}^\top \boldsymbol{R}\,. \tag{3.5}$$

Since $\boldsymbol{R}$ is an upper triangular matrix, solving equation (3.3) becomes very cheap, involving one forward substitution, solving $\boldsymbol{R}^\top \boldsymbol{v} = \boldsymbol{A}^\top \boldsymbol{b}$ for $\boldsymbol{v}$ and one backward substitution, solving $\boldsymbol{R}\boldsymbol{x} = \boldsymbol{v}$ for $\boldsymbol{x}$. The total cost of solving least squares using normal equations and finding QR factorization of $\boldsymbol{A}$ is approximately $mn^2 + \frac{1}{3}n^3$ floating point operations (flops) [15].

There are many ways to compute the QR factorization of $\boldsymbol{A}$, for example Gram-Schmidt orthogonalization [1], modified Gram-Schmidt [10], Householder reflections [6, Ch. 5], and Givens rotations [5]. Givens rotations are often used when matrix $\boldsymbol{A}$ is sparse and structured.

A Givens rotation rotates a vector $\boldsymbol{x} \in \mathbb{R}^2$ so that one component is zeroed out but the length is preserved, meaning the norm of $\boldsymbol{x}$ remains the same. A Givens

rotation in 2-dimension is denoted as the following:

$$\boldsymbol{G}(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} = \begin{bmatrix} c & -s \\ s & c \end{bmatrix}. \tag{3.6}$$

It is easy to verify that $\boldsymbol{G}(\theta)$ is an orthogonal matrix: that is, $\boldsymbol{G}(\theta)^\top \boldsymbol{G}(\theta) = \boldsymbol{G}(\theta)\boldsymbol{G}(\theta)^\top = \boldsymbol{I}$. Given $\boldsymbol{x} \in \mathbb{R}^2$, the angle θ, or equivalently the scalars s and c, are chosen so that

$$\begin{bmatrix} c & -s \\ s & c \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} \sqrt{x_1^2 + x_2^2} \\ 0 \end{bmatrix}, \tag{3.7}$$

because of the length preservation. Equation (3.7) along with the orthogonality of $\boldsymbol{G}$ provide two easy expressions for c and s to construction $\boldsymbol{G}$.

$$\begin{aligned} c &= \frac{x_1}{\sqrt{x_1^2 + x_2^2}}, \\ s &= \frac{-x_2}{\sqrt{x_1^2 + x_2^2}}. \end{aligned} \tag{3.8}$$

A Givens rotation can be generalized to higher dimensions with the same idea as in 2-dimensional case. In the m-dimensional case, matrix $\boldsymbol{G}$ becomes:

$$\boldsymbol{G}(i, j, \theta) = \begin{bmatrix} 1 & 0 & \cdots & \cdots & \cdots & \cdots \\ 0 & \ddots & & & & \\ \vdots & & c & \cdots & -s & \\ \vdots & & & \ddots & & \\ \vdots & & s & \cdots & c & \\ \vdots & & & & & \ddots \end{bmatrix}. \tag{3.9}$$

It is an identity matrix, except that entry (i, i) is c, entry (i, j) is $-s$, entry (j, i) is s, and entry (j, j) is c. Then, $\boldsymbol{G}(i, j, \theta)\boldsymbol{A}$ only modifies row i and row j, and $\boldsymbol{A}\boldsymbol{G}(i, j, \theta)$ only modifies column i and column j. If applied selectively through left

multiplication, Givens rotations can be used to reduce a matrix to upper triangular form. Specifically, we use

$$\begin{aligned} \boldsymbol{G}_k \cdots \boldsymbol{G}_2 \boldsymbol{G}_1 \boldsymbol{A} &= \boldsymbol{R} \\ \Rightarrow \boldsymbol{A} &= \boldsymbol{G}_1^\top \boldsymbol{G}_2^\top \cdots \boldsymbol{G}_k^\top \boldsymbol{R} \\ \boldsymbol{A} &= \boldsymbol{Q}\boldsymbol{R}\,, \end{aligned} \tag{3.10}$$

where

$$\boldsymbol{G}_j = \boldsymbol{G}(j,n) \cdots \boldsymbol{G}(j,j+2)\boldsymbol{G}(j,j+1)\,,$$

$\boldsymbol{R}$ is upper triangular, and $\boldsymbol{Q} = \boldsymbol{G}_1^\top \boldsymbol{G}_2^\top \cdots \boldsymbol{G}_k^\top$. Here, we omitted θ in the expression because we do not need to compute it explicitly. Notice that here $\boldsymbol{Q}$ is not explicitly constructed, but stored as the values of c and s in each elimination step.

3.2 Rank-1 updating

In Chapter 4, we describe a fast algorithm for computing the Cholesky factorization of $\boldsymbol{A}^\top \boldsymbol{A}$, when $\boldsymbol{A}$ is a Toeplitz matrix. The scheme requires recursively computing rows of the Cholesky factorization through a sequence of rank-1 updating and downdating problems. In this section we describe how Givens rotations can be used to solve the rank-1 updating problem.

Consider a known upper triangular matrix $\boldsymbol{U}$, where $\boldsymbol{U} \in \mathbb{R}^{m \times m}$, and a row vector $\boldsymbol{z}^\top$, where $\boldsymbol{z} \in \mathbb{R}^m$. Givens rotations can be used to efficiently compute the Cholesky of $\boldsymbol{U}^\top \boldsymbol{U} + \boldsymbol{z}\boldsymbol{z}^\top$, which is called the rank-1 updating problem.

The rank-1 updating problem is the same as finding the QR factorization of matrix $\boldsymbol{C} \in \mathbb{R}^{(m+1)\times m}$ such that $\boldsymbol{C} = \begin{bmatrix} \boldsymbol{U} \\ \boldsymbol{z}^\top \end{bmatrix}$. The reasoning is that $\boldsymbol{C}^\top \boldsymbol{C} =$

$\begin{bmatrix} \boldsymbol{U}^\top & \boldsymbol{z} \end{bmatrix} \begin{bmatrix} \boldsymbol{U} \\ \boldsymbol{z}^\top \end{bmatrix} = \boldsymbol{U}^\top\boldsymbol{U} + \boldsymbol{z}\boldsymbol{z}^\top = \hat{\boldsymbol{R}}^\top\hat{\boldsymbol{R}}$. Here, $\hat{\boldsymbol{R}} = \begin{bmatrix} \hat{\boldsymbol{U}} \\ \boldsymbol{0}^\top \end{bmatrix}$.

Then, matrix $\boldsymbol{C}$ will have the following structure

$$\boldsymbol{C} = \begin{bmatrix} * & * & * & \cdots & * \\ 0 & * & * & \cdots & * \\ 0 & 0 & * & \cdots & * \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & * \\ * & * & * & \cdots & * \end{bmatrix}.$$

To find the $\hat{\boldsymbol{R}}$ for the QR factorization of $\boldsymbol{C}$, we only need to zero out the last row of $\boldsymbol{C}$. The first row is used to zero out $\boldsymbol{C}(m+1,1)$, and equation (3.8) is used with $\boldsymbol{C}(1,1)$ and $\boldsymbol{C}(m+1,1)$ to find the corresponding Givens rotation elements c and s. Then, c and s will be applied to the entire first and last rows as shown in equation (3.11), with $i = 1$.

$$\begin{aligned} \boldsymbol{C}(i, i:n) &= c * \boldsymbol{C}(i, i:n) - s * \boldsymbol{C}(m+1, i:n)\,, \\ \boldsymbol{C}(m+1, i:n) &= s * \boldsymbol{C}(i, i:n) + c * \boldsymbol{C}(m+1, i:n)\,. \end{aligned} \tag{3.11}$$

Here, the colon is a Matlab notation, denoting that it is from the beginning to the end. Thus, $\boldsymbol{C}(i, i:n)$ means $[c_{i,i}, c_{i,i+1}, \cdots, c_{i,n}]$.

Afterwards, $\boldsymbol{C}$ becomes

$$\boldsymbol{C} = \begin{bmatrix} * & * & * & \cdots & * \\ 0 & * & * & \cdots & * \\ 0 & 0 & * & \cdots & * \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & * \\ 0 & * & * & \cdots & * \end{bmatrix}. \tag{3.12}$$

The second row is used to zero out $\boldsymbol{C}(m+1,2)$. $\boldsymbol{C}(2,2)$ and $\boldsymbol{C}(m+1,2)$ are plugged in as x_1 and x_2 respectively in equation (3.8) to find c and s. Then, c and s are applied to the second and last rows from entry 2 to n as equation (3.11) with $i = 2$.

Then, $\boldsymbol{C}$ becomes

$$\boldsymbol{C} = \begin{bmatrix} * & * & * & \cdots & * \\ 0 & * & * & \cdots & * \\ 0 & 0 & * & \cdots & * \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & * \\ 0 & 0 & * & \cdots & * \end{bmatrix}.$$

Following the same logic, $\hat{\boldsymbol{R}}$ is easily found with Givens rotations.

3.3 Rank-1 downdating

Besides the rank-1 updating problem, there is a rank-1 downdating problem, where the goal is to find the Cholesky factorization, $\tilde{\boldsymbol{R}}^\top \tilde{\boldsymbol{R}}$, for matrix $\boldsymbol{U}^\top \boldsymbol{U} - \boldsymbol{w}\boldsymbol{w}^\top$. Here, $\boldsymbol{U} \in \mathbb{R}^{m \times m}$ and $\boldsymbol{w} \in \mathbb{R}^m$. Hyperbolic rotations are used to efficiently compute $\tilde{\boldsymbol{R}}$, see page 16.

Define a matrix $\boldsymbol{S}$ such that

$$\boldsymbol{S} = \begin{bmatrix} \boldsymbol{I}_m & \boldsymbol{0} \\ \boldsymbol{0}^\top & -1 \end{bmatrix}.$$

Therefore, we can break $\boldsymbol{U}^\top\boldsymbol{U} - \boldsymbol{w}\boldsymbol{w}^\top$ into the multiplication of three matrices:

$$\boldsymbol{U}^\top\boldsymbol{U} - \boldsymbol{w}\boldsymbol{w}^\top = \begin{bmatrix} \boldsymbol{U} \\ \boldsymbol{w}^\top \end{bmatrix}^\top \boldsymbol{S} \begin{bmatrix} \boldsymbol{U} \\ \boldsymbol{w}^\top \end{bmatrix}.$$

Following the same logic as Givens rotations, the product of a set of hyperbolic rotations is denoted as matrix $\boldsymbol{H}$, where $\boldsymbol{H} \in \mathbb{R}^{(m+1)\times(m+1)}$. It satisfies two properties [6]

$$\boldsymbol{H}\boldsymbol{S}\boldsymbol{H}^\top = \boldsymbol{S}, \tag{3.13}$$

$$\boldsymbol{H}^\top \begin{bmatrix} \boldsymbol{U} \\ \boldsymbol{w}^\top \end{bmatrix} = \begin{bmatrix} \tilde{\boldsymbol{U}} \\ 0 \end{bmatrix} = \tilde{\boldsymbol{R}}. \tag{3.14}$$

In this way,

$$\begin{aligned} \begin{bmatrix} \boldsymbol{U} \\ \boldsymbol{w}^\top \end{bmatrix}^\top \boldsymbol{S} \begin{bmatrix} \boldsymbol{U} \\ \boldsymbol{w}^\top \end{bmatrix} &= \begin{bmatrix} \boldsymbol{U} \\ \boldsymbol{w}^\top \end{bmatrix}^\top \boldsymbol{H}\boldsymbol{S}\boldsymbol{H}^\top \begin{bmatrix} \boldsymbol{U} \\ \boldsymbol{w}^\top \end{bmatrix} \\ &= \left(\boldsymbol{H}^\top \begin{bmatrix} \boldsymbol{U} \\ \boldsymbol{w}^\top \end{bmatrix} \right)^\top \boldsymbol{S} \left(\boldsymbol{H}^\top \begin{bmatrix} \boldsymbol{U} \\ \boldsymbol{w}^\top \end{bmatrix} \right). \\ &= \begin{bmatrix} \tilde{\boldsymbol{R}} \\ 0 \end{bmatrix}^\top \boldsymbol{S} \begin{bmatrix} \tilde{\boldsymbol{R}} \\ 0 \end{bmatrix} \\ &= \tilde{\boldsymbol{R}}^\top \tilde{\boldsymbol{R}} \end{aligned} \tag{3.15}$$

The general form of $\boldsymbol{H}$ is

$$\boldsymbol{H}(i,j,\theta) == \begin{bmatrix} 1 & 0 & \cdots & \cdots & \cdots & \cdots \\ 0 & \ddots & & & & \\ \vdots & & c & \cdots & -s & \\ \vdots & & & \ddots & & \\ \vdots & & -s & \cdots & c & \\ \vdots & & & & & \ddots \end{bmatrix}, \tag{3.16}$$

where $c = \cosh(\theta)$ and $s = \sinh(\theta)$. It is also an identity matrix, with the exception that entry (i,i) is c, entry (i,j) is $-s$, entry (j,i) is $-s$, and entry (j,j) is c.

If we consider a 2-dimensional hyperbolic rotation, we want to find c and s to satisfy.

$$\begin{bmatrix} c & -s \\ -s & c \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} \sqrt{x_1^2 - x_2^2} \\ 0 \end{bmatrix}. \tag{3.17}$$
$$c^2 - s^2 = 0$$

Using equation (3.17), c and s can be easily calculated [6].

$$\begin{aligned} c &= \frac{x_1}{\sqrt{x_1^2 - x_2^2}} \\ s &= \frac{x_2}{\sqrt{x_1^2 - x_2^2}} \end{aligned} \tag{3.18}$$

Then, following the same logic as applying Givens rotations, we focus on the two entries, where one is used to zero out the other one. We use the two entries to find corresponding c and s as in equation (3.17). Then, we apply them to the two corresponding rows and move to the next entry that needs to be zeroed out. At last, we would get $\tilde{\boldsymbol{R}} = \begin{bmatrix} \tilde{\boldsymbol{U}} \\ \mathbf{0}^\top \end{bmatrix}$ efficiently.

3.4 Iterative refinement

The Cholesky factorization of $\boldsymbol{A}^\top \boldsymbol{A}$ acquired using Givens rotation can help us solve the normal equations, and thus solve the least squares problem. However, the result may be insufficiently accurate because of rounding errors, whose effect is magnified by the ill-conditioning of matrix $\boldsymbol{A}$. Thus, iterative refinement can be used to improve accuracy and stability [2].

One approach for iterative refinement is stated as follows [2]. The initial approximation for the solution $\boldsymbol{x}_0$ is needed.

1. At i^{th} iteration, we compute residual $\boldsymbol{r}_i = \boldsymbol{A}^\top \boldsymbol{b} - \boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{x}_i$.

2. We solve $\boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{d}_i = \boldsymbol{r}_i$.

3. At last, the solution is updated as $\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + \boldsymbol{d}_i$.

Many techniques can be used to solve the linear system in step 2, and Generalized Residual (GMRES) algorithm [13] is used here. Moreover, we could utilize a preconditioner to accelerate convergence of the GMRES algorithm. A left preconditioner is used here, where instead of solving $\boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{d}_i = \boldsymbol{r}_i$ directly, we construct a preconditioner $\boldsymbol{M}$, using the knowledge we have about $\boldsymbol{A}^\top \boldsymbol{A}$, and solve $\boldsymbol{M}^{-1}\boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{d}_i = \boldsymbol{M}^{-1}\boldsymbol{r}_i$. GMRES with left preconditioner can help achieve relatively good accuracy of the solution for the correction equation [2]. One way to construct $\boldsymbol{M}$ is that we compute the Cholesky decomposition of $\boldsymbol{A}^\top \boldsymbol{A}$ and using forward and backward substitution to solve for the solution.

To further accelerate the speed of the refinement process, three different precision levels are used. We could choose one precision for computing the preconditioner, (e.g. half,or 16 bit), one for solving step 2 iteratively, (e.g. single,or 32 bit), and at last, one for computing the residual in each iteration, (e.g. double,or 64 bit),.

In the next chapter, we provide more details of how to efficiently solve the Toeplitz least squares problem. Specifically, utilizing its special structure, algorithms to find

the Cholesky factorization of $\boldsymbol{A}^\top \boldsymbol{A}$ is described, as well as the algorithms to directly find the inverse of the Cholesky factorization, i.e. $\boldsymbol{R}^{-\top}$. We also present a new approach for iterative refinement for Toeplitz systems, as well as some challenges in the process.

Chapter 4

Approaches for Toeplitz Systems

4.1 The Cholesky factorization for Toeplitz least squares problem

In this thesis we are considering Toeplitz least squares problems, meaning matrix $\boldsymbol{A}$ has the structure that each descending diagonal from left to right is constant, as shown in Chapter 1. This type of problem can be easily solved if we could find the Cholesky factorization of $\boldsymbol{A}^\top \boldsymbol{A}$, as discussed in Chapter 3. Due to the special structure of $\boldsymbol{A}$, efficient algorithms are developed to find the matrix $\boldsymbol{R}$, where $\boldsymbol{R}^\top \boldsymbol{R} = \boldsymbol{A}^\top \boldsymbol{A}$, and this chapter will review the two algorithms from paper [12].

$\boldsymbol{A} \in \mathbb{R}^{m \times n}$, can be partitioned in the following two ways:

$$\boldsymbol{A} = \begin{bmatrix} a_0 & \boldsymbol{u}^\top \\ \boldsymbol{v} & \boldsymbol{A}_0 \end{bmatrix} = \begin{bmatrix} \boldsymbol{A}_0 & \tilde{\boldsymbol{u}} \\ \tilde{\boldsymbol{v}}^\top & a_{m-n} \end{bmatrix}, \tag{4.1}$$

where a_0 and a_{m-n} are two scalars, $\boldsymbol{u}, \tilde{\boldsymbol{v}} \in \mathbb{R}^{n-1}$ and $\boldsymbol{v}, \tilde{\boldsymbol{u}} \in \mathbb{R}^{m-1}$. Note we have $\boldsymbol{A}_0$ for both partitions, which is only due to the Toeplitz structure of $\boldsymbol{A}$. Similarly, we

can partition $\boldsymbol{R}$ in two ways:

$$\boldsymbol{R} = \begin{bmatrix} r_{11} & \boldsymbol{z}^\top \\ \boldsymbol{0} & \boldsymbol{R}_b \end{bmatrix} = \begin{bmatrix} \boldsymbol{R}_t & \tilde{\boldsymbol{z}} \\ \boldsymbol{0}^\top & r_{nn} \end{bmatrix}, \tag{4.2}$$

where r_{11} and r_{nn} are two scalars, and $\boldsymbol{z}, \tilde{\boldsymbol{z}} \in \mathbb{R}^{n-1}$. Plugging these partitions into equation (3.5) will result two equations,

$$\begin{bmatrix} r_{11}^2 & r_{11}\boldsymbol{z}^\top \\ r_{11}\boldsymbol{z} & \boldsymbol{z}\boldsymbol{z}^\top + \boldsymbol{R}_b^\top \boldsymbol{R}_b \end{bmatrix} = \begin{bmatrix} a_0^2 + \boldsymbol{v}^\top \boldsymbol{v} & a_0 \boldsymbol{u}^\top + \boldsymbol{v}^\top \boldsymbol{A}_0 \\ a_0 \boldsymbol{u} + \boldsymbol{A}_0^\top \boldsymbol{v} & \boldsymbol{u}\boldsymbol{u}^\top + \boldsymbol{A}_0^\top \boldsymbol{A}_0 \end{bmatrix}, \tag{4.3}$$

and

$$\begin{bmatrix} \boldsymbol{R}_t^\top \boldsymbol{R}_t & \boldsymbol{R}_t^\top \tilde{\boldsymbol{z}} \\ \tilde{\boldsymbol{z}}^\top \boldsymbol{R}_t & \tilde{\boldsymbol{z}}^\top \tilde{\boldsymbol{z}} + r_{nn}^2 \end{bmatrix} = \begin{bmatrix} \boldsymbol{A}_0^\top \boldsymbol{A}_0 + \tilde{\boldsymbol{v}}\tilde{\boldsymbol{v}}^\top & \boldsymbol{A}_0^\top \tilde{\boldsymbol{u}}^\top + a_{m-n}\tilde{\boldsymbol{v}} \\ \tilde{\boldsymbol{u}}^\top \boldsymbol{A}_0 + a_{m-n}\tilde{\boldsymbol{v}}^\top & \tilde{\boldsymbol{u}}^\top \tilde{\boldsymbol{u}} + a_{m-n}^2 \end{bmatrix}. \tag{4.4}$$

From the last entry of equation (4.3) and the first entry of equation (4.4), we have:

$$\begin{aligned} \boldsymbol{z}\boldsymbol{z}^\top + \boldsymbol{R}_b^\top \boldsymbol{R}_b &= \boldsymbol{u}\boldsymbol{u}^\top + \boldsymbol{A}_0^\top \boldsymbol{A}_0 \\ \boldsymbol{R}_t^\top \boldsymbol{R}_t &= \boldsymbol{A}_0^\top \boldsymbol{A}_0 + \tilde{\boldsymbol{v}}\tilde{\boldsymbol{v}}^\top . \end{aligned}$$

Therefore, we have

$$\boldsymbol{R}_b^\top \boldsymbol{R}_b = \boldsymbol{R}_t^\top \boldsymbol{R}_t + \boldsymbol{u}\boldsymbol{u}^\top - \tilde{\boldsymbol{v}}\tilde{\boldsymbol{v}}^\top - \boldsymbol{z}\boldsymbol{z}^\top . \tag{4.5}$$

$\boldsymbol{u}$, $\tilde{\boldsymbol{v}}$, and $\boldsymbol{z}$ are known, and our goal is to find $\boldsymbol{R}$, or equivalently $\boldsymbol{R}_b$ and $\boldsymbol{R}_t$. This is done recursively, one row at a time.

Notice that using equation (4.3), we can easily calculate

$$r_{11}^2 = a_0^2 + \boldsymbol{v}^\top \boldsymbol{v} \tag{4.6}$$

$$\boldsymbol{z}^\top = \frac{a_0}{r_{11}} \boldsymbol{u}^\top + \frac{1}{r_{11}} \boldsymbol{v}^\top \boldsymbol{A}_0 \,. \tag{4.7}$$

Thus, we would be able to know the first row of $\boldsymbol{R}_t$, and hence the first row of $\boldsymbol{R}$. Then, plugging this first row of $\boldsymbol{R}_t$ into the first part of the equation (4.5), we can use the updating techniques discussed in Chapter 3 to get the first row of $\boldsymbol{R}_1$ (see section 3.2, where:

$$\boldsymbol{R}_1^\top \boldsymbol{R}_1 = \boldsymbol{R}_t^\top \boldsymbol{R}_t + \boldsymbol{u}\boldsymbol{u}^\top \,. \tag{4.8}$$

After we get first row of $\boldsymbol{R}_1$, we plug it into the second part of equation (4.5). We can perform a downdating step (see section 3.3) and get the first row of $\boldsymbol{R}_2$, where:

$$\boldsymbol{R}_2^\top \boldsymbol{R}_2 = \boldsymbol{R}_1^\top \boldsymbol{R}_1 - \tilde{\boldsymbol{v}}\tilde{\boldsymbol{v}}^\top \,. \tag{4.9}$$

Similarly, we plug the first row of $\boldsymbol{R}_2$ into the last part of equation (4.5), and perform another downdating step get the first row of $\boldsymbol{R}_b$, where:

$$\boldsymbol{R}_b^\top \boldsymbol{R}_b = \boldsymbol{R}_1^\top \boldsymbol{R}_1 - \boldsymbol{z}\boldsymbol{z}^\top \,. \tag{4.10}$$

From the two partitions of $\boldsymbol{R}$ shown in equation (4.2), we know that the first row of $\boldsymbol{R}_b$ is the second row of $\boldsymbol{R}_t$. Then we plug it into equations (4.8-4.10) to get the second row of $\boldsymbol{R}_b$, which is the third row of $\boldsymbol{R}_t$, and the process goes on until the complete $\boldsymbol{R}$ is generated.

Notice here solving equation (4.8) is the rank-1 updating problem and solving equations (4.9) and (4.10) are two rank-1 downdating problems. Therefore, these can be solved efficiently using Givens rotations and hyperbolic rotations introduced in Chapter 3. To solve equation (4.8), we only need to find a series of Givens rotations

$\boldsymbol{Q} = \boldsymbol{G}_{n-1}\boldsymbol{G}_{n-2}\cdots\boldsymbol{G}_1$, such that

$$\boldsymbol{Q}\begin{bmatrix} \boldsymbol{R}_t \\ \boldsymbol{u}^\top \end{bmatrix} = \begin{bmatrix} \boldsymbol{R}_1 \\ \boldsymbol{0}^\top \end{bmatrix}. \tag{4.11}$$

To solve equations (4.9) and (4.10), we need to find two series of hyperbolic rotations $\boldsymbol{Y}^1$ and $\boldsymbol{Y}^2$ such that

$$\boldsymbol{Y}^1\begin{bmatrix} \boldsymbol{R}_1 \\ \tilde{\boldsymbol{v}}^\top \end{bmatrix} = \begin{bmatrix} \boldsymbol{R}_2 \\ \boldsymbol{0}^\top \end{bmatrix}. \tag{4.12}$$

$$\boldsymbol{Y}^2\begin{bmatrix} \boldsymbol{R}_2 \\ \boldsymbol{z}^\top \end{bmatrix} = \begin{bmatrix} \boldsymbol{R}_b \\ \boldsymbol{0}^\top \end{bmatrix}. \tag{4.13}$$

4.2 The Cholesky factorization for regularized Toeplitz least squares problem

This thesis considers regularized Toeplitz least squares problem, following equation (1.2). Therefore, the normal equation is:

$$(\boldsymbol{A}^\top\boldsymbol{A} + \alpha^2\boldsymbol{I})\boldsymbol{x} = \boldsymbol{A}^\top\boldsymbol{b}\,, \tag{4.14}$$

where $\boldsymbol{I}$ is an identity matrix.

Therefore, we want to compute the Cholesky for matrix $\boldsymbol{A}^\top\boldsymbol{A} + \alpha^2\boldsymbol{I}$ instead. Notice that this matrix only modifies the diagonal elements of $\boldsymbol{A}^\top\boldsymbol{A}$. Therefore, equations (4.3-4.4) become

$$\begin{bmatrix} r_{11}^2 & r_{11}\boldsymbol{z}^\top \\ r_{11}\boldsymbol{z} & \boldsymbol{z}\boldsymbol{z}^\top + \boldsymbol{R}_b^\top\boldsymbol{R}_b \end{bmatrix} = \begin{bmatrix} a_0^2 + \boldsymbol{v}^\top\boldsymbol{v} + \alpha^2 & a_0\boldsymbol{u}^\top + \boldsymbol{v}^\top\boldsymbol{A}_0 \\ a_0\boldsymbol{u} + \boldsymbol{A}_0^\top\boldsymbol{v} & \boldsymbol{u}\boldsymbol{u}^\top + \boldsymbol{A}_0^\top\boldsymbol{A}_0 + \alpha^2\boldsymbol{I} \end{bmatrix}, \tag{4.15}$$

and

$$\begin{bmatrix} \boldsymbol{R}_t^\top \boldsymbol{R}_t & \boldsymbol{R}_t^\top \tilde{\boldsymbol{z}} \\ \tilde{\boldsymbol{z}}^\top \boldsymbol{R}_t & \tilde{\boldsymbol{z}}^\top \tilde{\boldsymbol{z}} + r_{nn}^2 \end{bmatrix} = \begin{bmatrix} \boldsymbol{A}_0^\top \boldsymbol{A}_0 + \tilde{\boldsymbol{v}}\tilde{\boldsymbol{v}}^\top + \alpha^2 \boldsymbol{I} & \boldsymbol{A}_0^\top \tilde{\boldsymbol{u}}^\top + a_{m-n}\tilde{\boldsymbol{v}} \\ \tilde{\boldsymbol{u}}^\top \boldsymbol{A}_0 + a_{m-n}\tilde{\boldsymbol{v}}^\top & \tilde{\boldsymbol{u}}^\top \tilde{\boldsymbol{u}} + a_{m-n}^2 + \alpha^2 \end{bmatrix}. \quad (4.16)$$

Therefore,

$$r_{11}^2 = a_0^2 + \boldsymbol{v}^\top \boldsymbol{v} + \alpha^2 . \quad (4.17)$$

The formula for $\boldsymbol{z}$ is unchanged as stated in equation (4.7). From the last entry of equation (4.15) and first entry of equation (4.16), we have the relation:

$$\begin{aligned} \boldsymbol{z}\boldsymbol{z}^\top + \boldsymbol{R}_b^\top \boldsymbol{R}_b &= \boldsymbol{u}\boldsymbol{u}^\top + \boldsymbol{A}_0^\top \boldsymbol{A}_0 + \alpha^2 \boldsymbol{I} \\ \boldsymbol{R}_t^\top \boldsymbol{R}_t &= \boldsymbol{A}_0^\top \boldsymbol{A}_0 + \tilde{\boldsymbol{v}}\tilde{\boldsymbol{v}}^\top + \alpha^2 \boldsymbol{I} . \end{aligned}$$

The $\alpha^2 \boldsymbol{I}$ term cancels out, resulting in the same relation as equation 4.5. Thus, the algorithm for computing Cholesky factorization of regularized Toeplitz least squares problem is generally the same as the one with no regularization, with minor modification of r_{11}. The pesudo-code is shown in Alg 1, modified from [12].

4.3 The inverse Cholesky factorization for Toeplitz least squares problem

Instead of computing the Cholesky factorization for $\boldsymbol{A}^\top \boldsymbol{A}$, it would be faster to solve the normal equation, if we can directly compute the inverse of Cholesky factorization $\boldsymbol{R}^{-\top}$. In this way, it only needs matrix-vector multiplications to solve for $\boldsymbol{x}$, instead of forward and backward substitutions.

$$\boldsymbol{x} = (\boldsymbol{R}^{-1}\boldsymbol{R}^{-\top})\boldsymbol{A}^\top \boldsymbol{b} . \quad (4.18)$$

Algorithm 1 The Cholesky factorization $\boldsymbol{R}$ for regularized Toeplitz least squares problem

Let $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ be a Toeplitz matrix, α be the regularization parameter, and $\boldsymbol{u}$, $\boldsymbol{v}$, and $\tilde{\boldsymbol{v}}$ be defined as in equation (4.1). This algorithm computes the Cholesky factor, $\boldsymbol{R}$, of $\boldsymbol{A}^\top\boldsymbol{A} + \alpha^2\boldsymbol{I}$.

$\boldsymbol{R}(1,1) = r_{11} = \sqrt{a_0^2 + \boldsymbol{v}^\top\boldsymbol{v} + \alpha^2}$

$\boldsymbol{z} = (a_0\boldsymbol{u} + \boldsymbol{A}_0^\top\boldsymbol{v})/r_{11}$

$\boldsymbol{R}(1, 2:n) = \boldsymbol{z}^\top$

for $k = 1, 2, \cdots, n-1$ **do**

$\boldsymbol{r}_t(k:n-1) = \boldsymbol{R}(k, k:n-1)$

$[c, s] = \text{givens}(\boldsymbol{r}_t(k), \boldsymbol{u}(k))$

$$\begin{bmatrix} \boldsymbol{r}_1^\top \\ \boldsymbol{u}^\top \end{bmatrix} = \begin{bmatrix} c & -s \\ s & c \end{bmatrix} \begin{bmatrix} \boldsymbol{r}_t^\top \\ \boldsymbol{u}^\top \end{bmatrix}$$

$[c, s] = \text{hyp}(\boldsymbol{r}_1(k), \tilde{\boldsymbol{v}}(k))$

$$\begin{bmatrix} \boldsymbol{r}_2^\top \\ \tilde{\boldsymbol{v}}^\top \end{bmatrix} = \begin{bmatrix} c & -s \\ -s & c \end{bmatrix} \begin{bmatrix} \boldsymbol{r}_1^\top \\ \tilde{\boldsymbol{v}}^\top \end{bmatrix}$$

$[c, s] = \text{hyp}(\boldsymbol{r}_2(k), \boldsymbol{z}(k))$

$$\begin{bmatrix} \boldsymbol{r}_b^\top \\ \boldsymbol{z}^\top \end{bmatrix} = \begin{bmatrix} c & -s \\ -s & c \end{bmatrix} \begin{bmatrix} \boldsymbol{r}_2^\top \\ \boldsymbol{z}^\top \end{bmatrix}$$

$\boldsymbol{R}(k+1, k+1:n) = \boldsymbol{r}_b(k:n-1)$

end for

Nagy [12] describes such an algorithm that directly computes $\boldsymbol{R}^{-\top}$.

Consider an orthogonal matrix $\boldsymbol{Q}$, which comes from Givens rotations, and suppose it satisfies the following:

$$\boldsymbol{Q}\begin{bmatrix} \boldsymbol{R}_0 & \boldsymbol{0} \\ \boldsymbol{g}^\top & 1 \end{bmatrix} = \begin{bmatrix} \tilde{\boldsymbol{R}}_0 & \boldsymbol{q} \\ \boldsymbol{0}^\top & \kappa \end{bmatrix}. \tag{4.19}$$

Here $\boldsymbol{R}_0, \tilde{\boldsymbol{R}}_0 \in \mathbb{R}^{(n-1)\times(n-1)}$, $\boldsymbol{g} \in \mathbb{R}^{n-1}$, and κ is a non-zero scalar. Moreover, both $\boldsymbol{R}_0$ and $\tilde{\boldsymbol{R}}_0$ are upper triangular, and $\boldsymbol{R}_0$ is non-singular. If we transpose both sides of equation (4.19), we would have:

$$\begin{bmatrix} \boldsymbol{R}_0^\top & \boldsymbol{g} \\ \boldsymbol{0}^\top & 1 \end{bmatrix}\boldsymbol{Q}^\top = \begin{bmatrix} \tilde{\boldsymbol{R}}_0^\top & \boldsymbol{0} \\ \boldsymbol{q}^\top & \kappa \end{bmatrix}. \tag{4.20}$$

Inverting both sides of equation (4.20) will result

$$\boldsymbol{Q}\begin{bmatrix} \boldsymbol{R}_0^{-\top} & -\boldsymbol{R}_0^{-\top}\boldsymbol{g} \\ \boldsymbol{0}^\top & 1 \end{bmatrix} = \begin{bmatrix} \tilde{\boldsymbol{R}}_0^{-\top} & \boldsymbol{0} \\ -\frac{\boldsymbol{q}^\top \tilde{\boldsymbol{R}}_0^{-\top}}{\kappa} & \frac{1}{\kappa} \end{bmatrix}. \tag{4.21}$$

From the last column of equation (4.21), we acquire the relationship

$$\boldsymbol{Q}\begin{bmatrix} -\boldsymbol{R}_0^{-\top}\boldsymbol{g} \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{0} \\ \frac{1}{\kappa} \end{bmatrix}, \tag{4.22}$$

where $\boldsymbol{Q}$ can be found by a series of Givens rotations: $\boldsymbol{Q} = \boldsymbol{G}_{n-1}\boldsymbol{G}_{n-2}\cdots\boldsymbol{G}_1$.

Then, if we have a hyperbolic rotation matrix $\boldsymbol{Y}$ such that:

$$\boldsymbol{Y}\begin{bmatrix} \boldsymbol{R}_0 & \boldsymbol{0} \\ \boldsymbol{g}^\top & 1 \end{bmatrix} = \begin{bmatrix} \tilde{\boldsymbol{R}}_0 & \boldsymbol{q} \\ \boldsymbol{0}^\top & \kappa \end{bmatrix}. \tag{4.23}$$

Then if we take inverses and then transpose on both sides of equation (4.23), we obtain:

$$\boldsymbol{Y}^{-\top}\left(\begin{bmatrix}\boldsymbol{R}_0 & \boldsymbol{0}\\ \boldsymbol{g}^\top & 1\end{bmatrix}\right)^{-\top} = \left(\begin{bmatrix}\tilde{\boldsymbol{R}}_0 & \boldsymbol{q}\\ \boldsymbol{0}^\top & \kappa\end{bmatrix}\right)^{-\top}. \tag{4.24}$$

From Chapter 3, we know that the hyperbolic rotation matrix satisfies equation (3.13). Also, note that $\boldsymbol{S}^{-1} = \boldsymbol{S}$ and $\boldsymbol{S}^\top = \boldsymbol{S}$. If we take the inverse and transpose on both sides of equation (3.13):

$$\boldsymbol{S} = (\boldsymbol{Y}\boldsymbol{S}\boldsymbol{Y}^\top)^{-\top} = \boldsymbol{Y}^{-\top}\boldsymbol{S}\boldsymbol{Y}^{-1}. \tag{4.25}$$

Therefore,

$$\boldsymbol{Y}^{-\top} = \boldsymbol{S}\boldsymbol{Y}\boldsymbol{S}. \tag{4.26}$$

Plugging equation (4.26) into equation (4.24), we have:

$$\boldsymbol{Y}\boldsymbol{S}\left(\begin{bmatrix}\boldsymbol{R}_0 & \boldsymbol{0}\\ \boldsymbol{g}^\top & 1\end{bmatrix}\right)^{-\top} = \boldsymbol{S}\left(\begin{bmatrix}\tilde{\boldsymbol{R}}_0 & \boldsymbol{q}\\ \boldsymbol{0}^\top & \kappa\end{bmatrix}\right)^{-\top}. \tag{4.27}$$

After computing the transpose inverse of the two matrices, plug them in to equation (4.27) and multiply with $\boldsymbol{S}$ to obtain:

$$\boldsymbol{Y}\begin{bmatrix}\boldsymbol{R}_0^{-\top} & -\boldsymbol{R}_0^{-\top}\boldsymbol{g}\\ \boldsymbol{0}^\top & -1\end{bmatrix} = \begin{bmatrix}\tilde{\boldsymbol{R}}_0^{-\top} & \boldsymbol{0}\\ \frac{\boldsymbol{q}^\top\tilde{\boldsymbol{R}}_0^{-\top}}{\kappa} & -\frac{1}{\kappa}\end{bmatrix}. \tag{4.28}$$

Focusing on the last column of equation (4.28), we have:

$$\boldsymbol{Y}\begin{bmatrix}-\boldsymbol{R}_0^{-\top}\boldsymbol{g}\\ -1\end{bmatrix} = \begin{bmatrix}\boldsymbol{0}\\ -\frac{1}{\kappa}\end{bmatrix} \rightarrow \boldsymbol{Y}\begin{bmatrix}\boldsymbol{R}_0^{-\top}\boldsymbol{g}\\ 1\end{bmatrix} = \begin{bmatrix}\boldsymbol{0}\\ \frac{1}{\kappa}\end{bmatrix}, \tag{4.29}$$

where $\boldsymbol{Y}$ can be found by the product of a series of hyperbolic rotations $\boldsymbol{Y} = \boldsymbol{H}_{n-1} \cdots \boldsymbol{H}_1$. Following the same logic of solving recursive updating and downdating problems, as equations (4.11-4.13), we have:

$$\boldsymbol{Q} \begin{bmatrix} \boldsymbol{R}_t^{-\top} \\ \boldsymbol{0}^\top \end{bmatrix} = \begin{bmatrix} \boldsymbol{R}_1^{-\top} \\ \boldsymbol{h}_1^\top \end{bmatrix}. \tag{4.30}$$

$$\boldsymbol{Y}^1 \begin{bmatrix} \boldsymbol{R}_1^{-\top} \\ \boldsymbol{0}^\top \end{bmatrix} = \begin{bmatrix} \boldsymbol{R}_2^{-\top} \\ \boldsymbol{h}_2^\top \end{bmatrix}. \tag{4.31}$$

$$\boldsymbol{Y}^2 \begin{bmatrix} \boldsymbol{R}_2^{-\top} \\ \boldsymbol{0}^\top \end{bmatrix} = \begin{bmatrix} \boldsymbol{R}_b^{-\top} \\ \boldsymbol{h}_3^\top \end{bmatrix}. \tag{4.32}$$

The relationship between rows of $\boldsymbol{R}^{-\top}$, $\boldsymbol{R}_t^{-\top}$, and $\boldsymbol{R}_b^{-\top}$ is acquired from equation (4.2):

$$\boldsymbol{R}^{-\top} = \begin{bmatrix} \frac{1}{r_{11}} & \boldsymbol{0}^\top \\ \frac{-\boldsymbol{R}_b^{-\top}\boldsymbol{z}}{r_{11}} & \boldsymbol{R}_b^{-\top} \end{bmatrix} = \begin{bmatrix} \boldsymbol{R}_t^{-\top} & \boldsymbol{0} \\ \frac{-\tilde{\boldsymbol{z}}^\top \boldsymbol{R}_t^{-\top}}{r_{nn}} & \frac{1}{r_{nn}} \end{bmatrix}. \tag{4.33}$$

Following the same logic of Section 4.2, the regularization only changes the expression of r_{11}. Thus, the algorithm for computing the inverse Cholesky factor of regularized Toeplitz least squares problem is generally the same as that with no regularization as shown in paper [12]. The pseudo-code is shown in Alg 2.

4.4 Refinement for regularized Toeplitz least squares problem with preconditioner

After finding the Cholesky factorization $\boldsymbol{R}$ or $\boldsymbol{R}^{-\top}$, we could utilize it in the refinement process, as preconditioners. Carson, Higham, and Pranesh [2] describes such a process, where they aimed to solve the second step of refinement $\boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{d}_i = \boldsymbol{r}_i$, by

Algorithm 2 The inverse Cholesky factorization for regularized Toeplitz least squares problem

Let $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ be a Toeplitz matrix, α be the regularization parameter, and $\boldsymbol{u}$, $\boldsymbol{v}$, and $\tilde{\boldsymbol{v}}$ be defined as in equation (4.1). This algorithm computes $\boldsymbol{L} = \boldsymbol{R}^{-\top}$, where $\boldsymbol{R}$ is the Cholesky factor of $\boldsymbol{A}^\top\boldsymbol{A} + \alpha^2\boldsymbol{I}$.

$r_{11} = \sqrt{a_0^2 + \boldsymbol{v}^\top\boldsymbol{v} + \alpha^2}$

$\boldsymbol{z} = (a_0\boldsymbol{u} + \boldsymbol{A}_0^\top\boldsymbol{v})/r_{11}$, $\boldsymbol{L}(1,1) = 1/r_{11}$

$\delta = \gamma_1 = \gamma_2 = 1$, $\boldsymbol{h}_1^\top = \boldsymbol{h}_2^\top = \boldsymbol{h}_3^\top = \boldsymbol{0}^\top$

for $k = 1, 2, \cdots, n-1$ **do**

$\boldsymbol{l}_t(1:k) = \boldsymbol{L}(k, 1:k)$

$\rho = \boldsymbol{l}_t^\top\boldsymbol{u}(1:k)$

$[c, s] = \text{givens}(\delta, -\rho)$

$\delta = s\rho + c\delta$

$$\begin{bmatrix} \boldsymbol{l}_1^\top \\ \boldsymbol{h}_1^\top \end{bmatrix} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} \boldsymbol{l}_t^\top \\ \boldsymbol{h}_1^\top \end{bmatrix}$$

$\beta = \boldsymbol{l}_1^\top\tilde{\boldsymbol{v}}(1:k)$

$[c, s] = \text{hyp}(\gamma_1, \beta)$

$\gamma_1 = -s\beta + c\gamma_1$

$$\begin{bmatrix} \boldsymbol{l}_2^\top \\ \boldsymbol{h}_2^\top \end{bmatrix} = \begin{bmatrix} c & -s \\ -s & c \end{bmatrix} \begin{bmatrix} \boldsymbol{l}_1^\top \\ \boldsymbol{h}_2^\top \end{bmatrix}$$

$\beta = \boldsymbol{l}_2^\top\boldsymbol{z}(1:k)$

$[c, s] = \text{hyp}(\gamma_2, \beta)$

$\gamma_2 = -s\beta + c\gamma_2$

$$\begin{bmatrix} \boldsymbol{l}_b^\top \\ \boldsymbol{h}_3^\top \end{bmatrix} = \begin{bmatrix} c & -s \\ -s & c \end{bmatrix} \begin{bmatrix} \boldsymbol{l}_2^\top \\ \boldsymbol{h}_3^\top \end{bmatrix}$$

$\boldsymbol{L}(k+1, 1:k+1) = [-\boldsymbol{l}_b^\top\boldsymbol{z}/r_{11}\,, \boldsymbol{l}_b^\top]$

end for

solving the preconditioned system: $\boldsymbol{M}^{-1}\boldsymbol{A}^\top\boldsymbol{A}\boldsymbol{d}_i = \boldsymbol{M}^{-1}\boldsymbol{r}_i$ using GMRES. With regularization, the main ideal is essentially the same. We aim to solve $(\boldsymbol{A}^\top\boldsymbol{A}+\alpha^2\boldsymbol{I})\boldsymbol{d}_i = \boldsymbol{r}_i$, by solving the preconditioned system: $\boldsymbol{M}^{-1}(\boldsymbol{A}^\top\boldsymbol{A}+\alpha^2\boldsymbol{I})\boldsymbol{d}_i = \boldsymbol{M}^{-1}\boldsymbol{r}_i$ using GMRES.

If we use $\boldsymbol{R}^\top\boldsymbol{R}$ as $\boldsymbol{M}$, we only need one forward substitution and one backward substitution to get $\boldsymbol{c}$. If we use $\boldsymbol{R}^{-1}\boldsymbol{R}^{-\top}$ as $\boldsymbol{M}^{-1}$, only two matrix-vector multiplications are needed, which accelerates the refinement process.

The pseudo-code for the Cholesky-based iterative refinement method using three precision levels is modified from Carson, Higham, and Pranesh [2] to solve regularized Toeplitz least squares problems, as shown in Alg 3. Here, we denote that the Cholesky factorization is computed in precision pr_1, (e.g. half), the working precision for refinement is pr_2, (e.g. single), and the residual precision is pr_3, (e.g.double). The convergence is determined by monitoring the relative residual norm in pr_2.

Algorithm 3 Cholesky-based GMRES iterative refinement for the regularized Toeplitz least squares

Let $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, the regularization parameter α, and $\boldsymbol{b} \in \mathbb{R}^m$. This algorithm solves the least squares problem as equation (1.2) using Cholesky-based GMRES.
Cast $\boldsymbol{A}$, $\boldsymbol{b}$, and α into precision pr_1
Compute Cholesky factorization $\boldsymbol{R}^\top\boldsymbol{R} = \boldsymbol{A}^\top\boldsymbol{A} + \alpha^2\boldsymbol{I}$ in precision pr_1
Solve $\boldsymbol{R}^\top\boldsymbol{R}\boldsymbol{x}_0 = \boldsymbol{A}^\top\boldsymbol{b}$
for $i = 0 : i_{max} - 1$ **do**
  Compute $\boldsymbol{r}_i = \boldsymbol{A}^\top\boldsymbol{b} - (\boldsymbol{A}^\top\boldsymbol{A} + \alpha^2\boldsymbol{I})\boldsymbol{x}_i$ at precision pr_3 and cast the $\boldsymbol{r}_i$ to pr_2
  Solve $\boldsymbol{M}^{-1}(\boldsymbol{A}^\top\boldsymbol{A} + \alpha^2\boldsymbol{I})\boldsymbol{d}_i = \boldsymbol{M}^{-1}\boldsymbol{r}_i$ by GMRES at pr_2, where $\boldsymbol{M} = \boldsymbol{R}^\top\boldsymbol{R}$. Note the matrix-vector multiplications with $\boldsymbol{A}^\top\boldsymbol{A}$ is done in precision pr_3, and the solution of the system is casted into pr_2
  $\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + \boldsymbol{d}_i$ at precision pr_2
  if converge **then**
    return $\boldsymbol{x}_{i+1}$
  end if
end for

The inverse Cholesky-based iterative refinement is very similar to Alg 3, where $\boldsymbol{M}^1$ can be directly found by $\boldsymbol{R}^{-\top}$.

We implemented the algorithm using both the Cholesky and inverse Cholesky factorizations as preconditioners for refinement on 1-D image deblurring problems using

a MATLAB package named **PRblur**. However, some challenges emerged. When implementing the inverse Cholesky for refinement in lower precision, we needed to execute inner products in lower precision several times in each iteration as shown in Alg 2, which caused large round-off errors. Though there are occurrences of inner products in the Cholesky then refinement algorithm, the number of them is relatively small, producing reasonable relative errors.

To avoid the problem, we tried to find other techniques to solve the rank-1 downdating problem than hyperbolic rotation, hoping we could avoid the use of lower-precision inner products. Stewart and Stewart [14] introduced the hyperbolic Householder transformation, where instead of $\boldsymbol{Y}$, a vector $\boldsymbol{h}^*$ was computed such that

$$(\boldsymbol{I} + \boldsymbol{S}\boldsymbol{h}^*\boldsymbol{h}^{*\top}/\mu) \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} \sigma \\ 0 \end{bmatrix},$$

where σ and μ are certain scalars found by methods in Stewart and Stewart [14]. Here x_1 and y_1 are two scalar entries, since we focused on each individual entry at a time to eliminate the lower one. However, in order for the Householder transformation to work as expected, y_1 needs to be smaller than x_1. Otherwise, the scalar ρ will be a complex number. When running the 1-D image deblurring problem using Householder transformation, y_1 was not always smaller than x_1. Therefore, we focused on computing Cholesky factorization and using refinement only, instead of the approach of computing the inverse of Cholesky factorization for the rest of the research.

Chapter 5

Numerical Experiments

We applied the algorithm of computing the Cholesky factorization and GMRES refinement on some test cases that are similar to image deblurring problems. They are small-scale, i.e. $\boldsymbol{A} \in \mathbb{R}^{64\times 64}$ and 1-dimensional, and we chose a Gaussian signal blur. This means the kernel function introduced in Chapter 2 is a Gaussian function. However, $\boldsymbol{A}$ is very ill-conditioned; its condition number is approximately 1.46×10^8. Therefore, Tikhonov regularization is needed, and $\boldsymbol{A}$ is augmented as shown in equation (1.2).

There exists many ways to generate the regularization parameter α, for example, General Cross Validation (GCV) [7], Weighted GCV [4], the Discrepancy Principle, and so on. Here, we chose the Optimal Regularization Parameter method, since the true solution is known to us. The goal of the Optimal Regularization Parameter method is: $\min \|\boldsymbol{x} - \boldsymbol{x}_{\text{true}}\|_2^2$, where $\boldsymbol{x}$ satisfies the normal equations:

$$(\boldsymbol{A}^\top \boldsymbol{A} + \alpha^2 \boldsymbol{I})\boldsymbol{x} = \boldsymbol{A}^\top \boldsymbol{b}\,, \tag{5.1}$$

and $\boldsymbol{x}_{\text{true}}$ is the true solution for the deblurring test problem. Therefore, the Optimal Regularization Parameter method enables us to assess the maximum potential performance of the algorithm. However, note that in reality, since the true solution

Table 5.1: Relative errors for 0% noise level.

Iteration	$relE_{hdd}$	$relE_{hsd}$	$relE_{sdd}$	$relE_{ssd}$
0	3.923×10^{1}	3.923×10^{1}	3.975×10^{0}	3.975×10^{0}
1	1.787×10^{-1}	1.848×10^{-1}	4.742×10^{-3}	3.797×10^{-1}
2	1.787×10^{-1}	1.787×10^{-1}	4.240×10^{-3}	7.257×10^{-1}
3	-	1.787×10^{-1}	4.275×10^{-3}	3.536×10^{-1}
4	-	1.787×10^{-1}	-	2.758×10^{-1}
5	-	1.787×10^{-1}	-	1.876×10^{-1}
6	-	1.787×10^{-1}	-	1.423×10^{-1}
7	-	1.787×10^{-1}	-	1.289×10^{-1}
8	-	1.787×10^{-1}	-	1.044×10^{-1}
9	-	1.787×10^{-1}	-	9.898×10^{-2}

is unknown, one needs to adopt other methods to find the regularization parameter.

For the test problems, we investigated different noise levels of $\boldsymbol{b}$ and different precision levels for the algorithm. Most computers use double precision, where each floating number is represented using 64 bits. However, computing with lower precision like single precision (32 bits) and half precision (16 bits) has attracted the interest of many researchers because of its advantage to decrease the computational time. We wanted to investigate the performance of the algorithm in different precision levels, so we used a MATLAB function named **chop** presented by Higham and Pranesh to simulate lower precision arithmetic.

For each test problem, we calculated the relative error following equation (2.8) of each refinement step, and Table 5.1 shows the relative errors for test cases with no noise. Then, $relE_{hdd}$ means that it is the relative errors when the Cholesky factorization is computed in half precision, the refinement process in double precision, and the residuals calculated in double precision. Other relative error symbols follow the same logic.

Here, iteration 0 means it is the initial solution using Cholesky factorization for the Toeplitz least squares problem without any refinement step. Then iteration 1 means we carried out the refinement once. $relE_{hdd}$ has only 3 entries, meaning the algorithm

Table 5.2: Relative errors for 0.1% noise level.

Iteration	$relE_{hdd}$	$relE_{hsd}$	$relE_{sdd}$	$relE_{ssd}$
0	3.760×10^{1}	3.760×10^{1}	1.569×10^{-1}	1.569×10^{-1}
1	1.824×10^{-1}	1.946×10^{-1}	1.521×10^{-1}	1.521×10^{-1}
2	1.824×10^{-1}	1.824×10^{-1}	-	1.521×10^{-1}
3	-	1.824×10^{-1}	-	1.521×10^{-1}
4	-	1.824×10^{-1}	-	1.521×10^{-1}
5	-	1.824×10^{-1}	-	1.521×10^{-1}
6	-	1.824×10^{-1}	-	1.521×10^{-1}
7	-	1.824×10^{-1}	-	1.521×10^{-1}
8	-	1.824×10^{-1}	-	1.521×10^{-1}
9	-	1.824×10^{-1}	-	1.521×10^{-1}

Table 5.3: Relative errors for 1% noise level.

Iteration	$relE_{hdd}$	$relE_{hsd}$	$relE_{sdd}$	$relE_{ssd}$
0	2.306×10^{1}	2.306×10^{1}	2.424×10^{-1}	2.424×10^{-1}
1	2.422×10^{-1}	2.425×10^{-1}	2.422×10^{-1}	2.422×10^{-1}
2	2.422×10^{-1}	2.422×10^{-1}	-	2.422×10^{-1}
3	-	2.422×10^{-1}	-	2.422×10^{-1}
4	-	2.422×10^{-1}	-	2.422×10^{-1}
5	-	2.422×10^{-1}	-	2.422×10^{-1}
6	-	2.422×10^{-1}	-	2.422×10^{-1}
7	-	2.422×10^{-1}	-	2.422×10^{-1}
8	-	2.422×10^{-1}	-	2.422×10^{-1}
9	-	2.422×10^{-1}	-	2.422×10^{-1}

stopped after carrying out refinement twice. The Table 5.2 shows the relative errors for test cases with noise level 0.1%. The Table 5.3 shows the relative errors for test cases with noise level 1%. At last, the Table 5.4 shows the relative errors for test cases with noise level 10%.

To better visualize the results, we plotted the true signal $\boldsymbol{x}_{\text{true}}$ and signal acquired from half-single-double. Figure 5.1 shows the three signals with 0% noise, 0.1% noise, 1% noise, and 10% noise respectively.

We chose the higher precision for refinement than computing Cholesky factorization because we wanted to improve the accuracy result, so that we could representing each floating number more accurately. Based on the relative errors, the precision

Table 5.4: Relative errors for 10% noise level.

Iteration	$relE_{hdd}$	$relE_{hsd}$	$relE_{sdd}$	$relE_{ssd}$
0	5.786×10^{-1}	5.786×10^{-1}	4.773×10^{-1}	4.773×10^{-1}
1	4.773×10^{-1}	4.773×10^{-1}	4.773×10^{-1}	4.773×10^{-1}
2	-	4.773×10^{-1}	-	4.773×10^{-1}
3	-	4.773×10^{-1}	-	4.773×10^{-1}
4	-	4.773×10^{-1}	-	4.773×10^{-1}
5	-	4.773×10^{-1}	-	4.773×10^{-1}
6	-	4.773×10^{-1}	-	4.773×10^{-1}
7	-	4.773×10^{-1}	-	4.773×10^{-1}
8	-	4.773×10^{-1}	-	4.773×10^{-1}
9	-	4.773×10^{-1}	-	4.773×10^{-1}

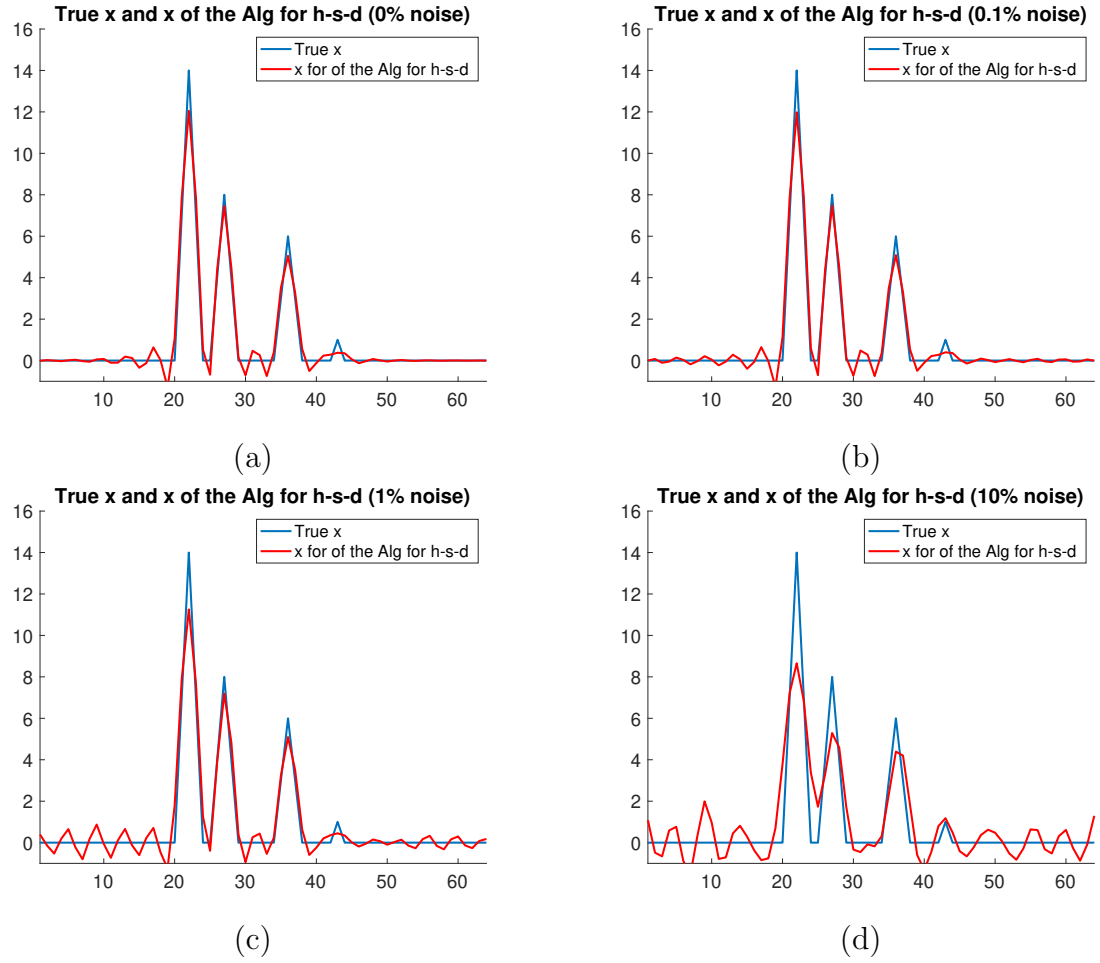


Figure 5.1: The result calculated by the fast algorithm using half-single-double for $\boldsymbol{b}$ (a) with 0% noise; (b) 0.1% noise; (c) 1% noise; (d) 10% noise.

which the computation of the Cholesky factorization is in does not affect the accuracy of the algorithm when the noise level is higher, comparing $relErr_{hdd}$ and $relErr_{sdd}$, $relErr_{hsd}$ and $relErr_{ssd}$ for both 1% and 10% noise. Therefore, choosing half is more reasonable, as it can decrease computational cost with fewer digits stored for larger noise. Moreover, from the relative errors, if we executed the refinement in double and then computed residuals in double, the algorithm is highly efficient, with only one or two iteration(s) of GMRES refinement needed, regardless the precision used for Cholesky factorization.

For the test cases, $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ is a square matrix. Therefore, using the efficient approach described in Chapter 4, computing the Cholesky factorization requires $\mathbf{O}(n^2)$ operations [12]. Before entering the refinement process, one matrix-vector multiplication is needed, which can be done efficiently with fast Fourier transforms because of its Toeplitz structure. Thus, the work unit of one matrix-vector multiplication is $\mathbf{O}(n \log n)$. It also requires two triangular solves (one backward and one forward substitution), whose total work unit is $2\mathbf{O}(n^2)$. Therefore, before any refinement, the total work unit is $3\mathbf{O}(n^2) + \mathbf{O}(n \log n)$.

For refinement process, if both work precision and residual precision are in double, one refinement iteration requires 2 matrix-vector multiplications and 2 triangular solves, following Alg 3. If the working precision is single while the residual precision is double, one refinement iteration requires 2 matrix-vector multiplications in double precision and 2 triangular solves in single precision.

Table 5.3 is used as an example, since they reach approximately the same accuracy level in the end. Each half-precision arithmetic computation is a quarter of the cost of double-precision arithmetic, and each single-precision arithmetic computation is half of double-precision arithmetic. Therefore, for half-double-double case, the total work unit is $\frac{19}{4}\mathbf{O}(n^2) + \frac{17}{4}\mathbf{O}(n \log n)$, with two refinement iterations. For half-single-double case, the total work unit is $\frac{39}{4}\mathbf{O}(n^2) + \frac{73}{4}\mathbf{O}(n \log n)$, with 9 refinement iterations. Here

the maximum iteration is set to be 9 iterations, but half-single-double case reaches the same level of accuracy as half-double-double after 1 refinement iteration. Thus, if we could find a more efficient stopping criteria for half-single-double, where it would stop after 1 iteration, the cost for it will be smaller than half-double-double, yet reaches approximately the same accuracy.

Chapter 6

Concluding Remarks

The Toeplitz least squares problem is common in signal/image deblurring, as well as other applications. However, due to the ill-conditioning of the Toeplitz matrix $\boldsymbol{A}$, any noise in the observation $\boldsymbol{b}$ will be magnified, causing significant relative errors. Therefore, regularization is often added to balance the residual and the magnitude of the solution, mitigating the effect of the noise and the ill-conditioning of $\boldsymbol{A}$. Cholesky factorization of $\boldsymbol{A}^\top \boldsymbol{A}$ is a useful tool to solve least squares efficiently, and by exploiting the special structure of the Toeplitz matrix, an efficient algorithm is developed to compute the Cholesky factorization.

To further improve the accuracy of the solution for the least squares problem, GMRES refinement is implemented. The Choleky factorization of $\boldsymbol{A}^\top \boldsymbol{A}$ is then re-utilized as a preconditioner for that process to accelerate the algorithm. Furthermore, three different precisions are used for computing Cholesky, running refinement, and calculating the residuals to speed up the process, by using fewer digits to represent each floating point number. At last, small 1-D signal deblurring problems are used as test cases for the algorithm. It is shown that for small noise levels, computing Cholesky factorization in single precision generates better results than computing that in half precision. However, for larger noise, the precision in which the computation of

Cholesky factorization is has no effect on the final accuracy. Also, using half-double-double is highly efficient, usually requiring 1 or 2 refinement steps.

Half-single-double or single-single-double can also be efficient, only requiring 1 refinement, if a good stopping criteria is found. Thus, one future step is to investigate the stopping criteria so that the algorithm is terminated when the relative errors stop changing, which is also applicable to other problems when the true solution is unknown. Moreover, further research can be done on extending the algorithm to 2-D signal deblurring problems with larger sizes, using Kronecker products, as well as on methods to choose suitable regularization parameters.

Bibliography

[1] Åke Björck. Solving linear least squares problems by Gram-Schmidt orthogonalization. *BIT Numerical Mathematics*, 7(1):1–21, 1967.

[2] Erin Carson, Nicholas J Higham, and Srikara Pranesh. Three-precision GMRES-based iterative refinement for least squares problems. *SIAM Journal on Scientific Computing*, 42(6):A4063–A4083, 2020.

[3] Raymond H Chan, James G Nagy, and Robert J Plemmons. FFT-based preconditioners for Toeplitz-block least squares problems. *SIAM journal on numerical analysis*, 30(6):1740–1768, 1993.

[4] Julianne Chung, James G Nagy, Dianne P O'Leary, et al. A weighted GCV method for Lanczos hybrid regularization. *Electronic Transactions on Numerical Analysis*, 28(149-167):2008, 2008.

[5] Alan George and Michael T Heath. Solution of sparse linear least squares problems using Givens rotations. *Linear Algebra and its Applications*, 34:69–83, 1980.

[6] Gene H Golub and Charles F Van Loan. *Matrix Computations*. JHU press, 2013.

[7] Per Christian Hansen. *Discrete Inverse Problems: Insight and Algorithms*. SIAM, 2010.

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

[9] Per Cristian Hansen. Numerical aspects of deconvolution. *Lecture Notes of the Department of Informatics and Mathematical Modelling of the Technical University of Denmark*, 2000.

[10] Steven J Leon, Åke Björck, and Walter Gander. Gram-Schmidt orthogonalization: 100 years and more. *Numerical Linear Algebra with Applications*, 20(3):492–532, 2013.

[11] Nicola Mastronardi, Phillip Lemmerling, Anoop Kalsi, DP O'Leary, and Sabine Van Huffel. Implementation of the regularized structured total least squares algorithms for blind image deblurring. *Linear algebra and its applications*, 391:203–221, 2004.

[12] James G Nagy. Fast inverse QR factorization for Toeplitz matrices. *SIAM Journal on Scientific Computing*, 14(5):1174–1193, 1993.

[13] Youcef Saad and Martin H Schultz. Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on scientific and statistical computing*, 7(3):856–869, 1986.

[14] Michael Stewart and GW Stewart. On hyperbolic triangularization: Stability and pivoting. *SIAM Journal on Matrix Analysis and Applications*, 19(4):847–860, 1998.

[15] Lloyd N Trefethen and David Bau. *Numerical Linear Algebra*, volume 181. SIAM, 2022.