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:

Peijian Ding April 8, 2021

Accelerated Alternating Minimization for X-ray Tomographic Reconstruction

By

Peijian Ding

James Nagy, Ph.D.
Advisor

Department of Mathematics

James Nagy, Ph.D.
Advisor

Yuanzhe Xi, Ph.D.
Committee Member

Gordon Berman, Ph.D.
Committee Member

2021

Accelerated Alternating Minimization for X-ray Tomographic Reconstruction

By

Peijian Ding

James 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

2021

Abstract

Accelerated Alternating Minimization for X-ray Tomographic Reconstruction
By Peijian Ding

While Computerized Tomography (CT) images can help detect disease such as Covid-19, regular CT machines are large and expensive. Cheaper and more portable machines suffer from errors in geometry acquisition that downgrades CT image quality. The errors in geometry can be represented with parameters in the mathematical model for image reconstruction. To obtain a good image, we formulate a nonlinear least squares problem that simultaneously reconstructs the image and corrects for errors in the geometry parameters. We develop an accelerated alternating minimization scheme to reconstruct the image and geometry parameters.

Accelerated Alternating Minimization for X-ray Tomographic Reconstruction

By

Peijian Ding

James 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

2021

Acknowledgments

I would like to thank my thesis advisor Dr. James Nagy for his consistent support since the start of the honors program. Without his help, this work would not be possible. I would also like to thank Dr. Yuanzhe Xi and Dr. Gordon Berman for being on my honors committee and teaching me wonderful course material.

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

1.1 Motivation

Tomography is a technique of displaying representations of a cross section through an object through the use of some penetrating waves such as X-ray or ultrasound. In simple words, it allows us to see the inside of an object without breaking it. Thus, tomography is widely used in medical imaging, seismic exploration, and material science. In medical imaging, a Computerized Tomography (CT) Scan creates a cross-sectional image of human body by combining X-ray images taken from different angles. During a CT scan, the patient lies on a bed that slowly moves through the gantry while the X-ray tube rotates around the patient and shoots X-ray beams through the human body, received by a detector. See Figure 1.1. Then, an image of the cross section of the human body is reconstructed following a mathematical procedure. Although CT images can help doctors to diagnose and monitor diseases such as Covid-19, regular CT machines are heavy and expensive and not widely available in less developed areas. The goal of this project is to compensate for cheaper and more portable machines by solving for geometry parameters such as the source-to-object distance that may not be calibrated precisely during the imaging process.

Source-to-object distance measures how far away the center of the object is from the X-ray source. Since the source-to-object distance may vary from angle to angle, the restored image will be corrupted if a constant value of the geometry parameter is assumed for all the angles. The source-to-object distance is an important factor that determines the quality of images as illustrated in Figure 1.2. Thus, our algorithm will significantly improve the image quality by taking into account the variation in geometry parameters.

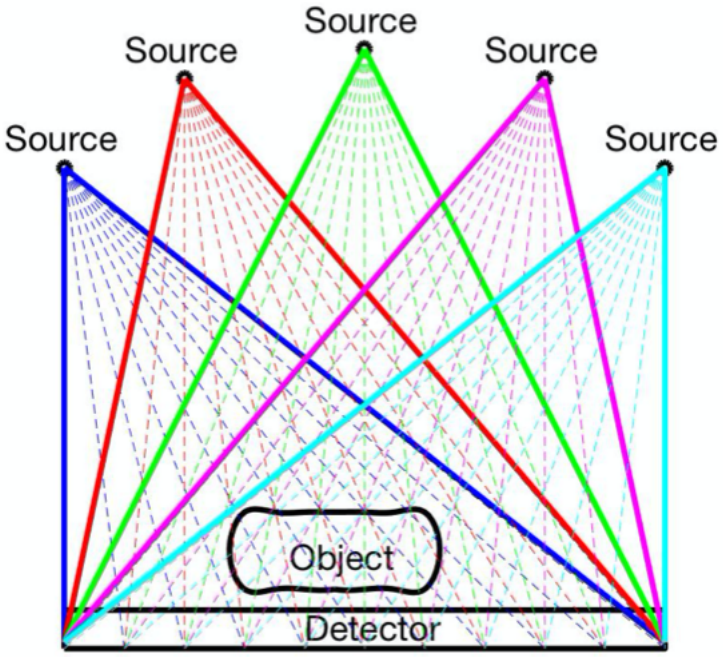

Figure 1.1: This is a simple illustration of our tomography problem.

This thesis is organized as follows. In the rest of the first chapter, we discuss the mathematics of tomography and the setup of the inverse problem. In Chapter 2, an alternating minimization scheme to solve the image and source-to-object distance is introduced. In Chapter 3, we discuss the acceleration algorithms. In Chapter

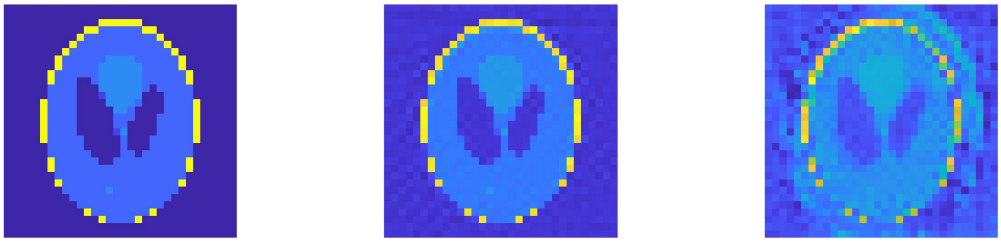

Figure 1.2: 32×32 true image of a Shepp–Logan phantom (left), image computed by taking into account the correct source-to-object distance (middle), image computed by using incorrect source-to-object distance (right)

4, numerical experiments are presented. Finally, concluding remarks are given in Chapter 5.

1.2 Mathematics of Computed Tomography

This section introduces some concepts in computed tomography that will help readers understand the background of this research. In this section, we primarily focus on 2D computed tomography that uses X-ray beams to obtain cross section images inside the human body. The most common types of 2D CT are parallel beam projection, illustrated in Figure 1.3 and fan beam projection, illustrated in Figure 1.4.

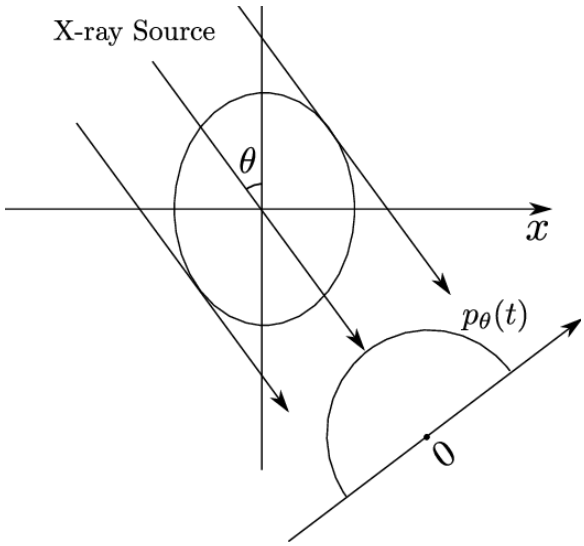

Figure 1.3: Geometry of parallel beam projection [11].

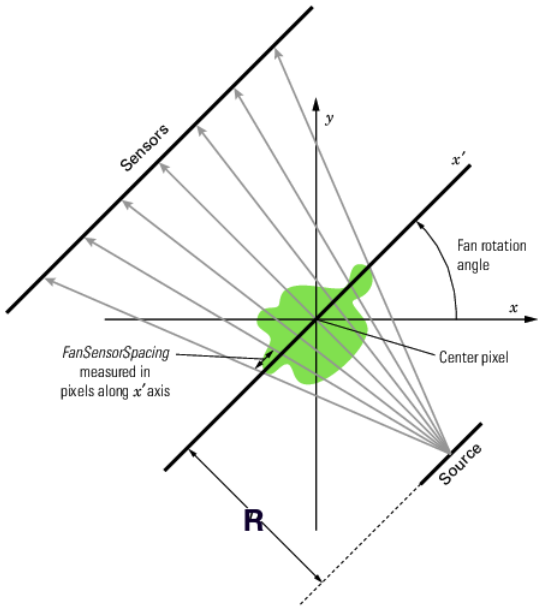

Figure 1.4: Fan beam projection (image from MATLAB).

The projection is the line integral along the path in which the X-ray beam passes. In parallel beam projection, the projection is simply the line integral along the path of the parallel beams. In the fan beam geometry, the projection is the line integral along the paths that radiate from a single source. The idea of X-ray tomography is enabled by Beer's Law which relates the reduction of light intensity to the path length and material dependent attenuation coefficient. Thus, the reduction of X-ray intensity is proportional to the attenuation coefficient which increases with physical density of the material. CT works by passing X-ray beams through human body and measuring the reduced X-ray intensity due to object density at the detector. Intuitively, when an X-ray beam passes through air, not much energy will be absorbed by the medium. In contrast, tissues in the human body have higher density than air allowing the soft tissues to be seen clearly when surrounded by a much lighter object such as air [1]. The objective is to find attenuation coefficients, which depend on material and location in the body, using Beer's Law. Beer's Law for a monochromatic X-ray beam through a homogeneous material is:

$$I = I_0 e^{-\mu x} \tag{1.1}$$

I_0 is the intensity of the X-ray from the source, I is the intensity at the detector, μ is the attenuation coefficient, and x is the length of the X-ray path through the material [9]. In practice, materials are usually heterogeneous such as the different tissues and bones inside a human body. Thus, the attenuation coefficient can be a function of coordinates on a plane. We express it as $\mu(x, y)$ in the following equation:

$$I = I_0 e^{\int_{ray} \mu(x,y) ds} \tag{1.2}$$

Instead of the direct product of an attenuation coefficient of a homogeneous material and the path length, we take an integral of the attenuation function along the X-ray path for the heterogeneous material. Through some algebraic manipulation, we can express the ray integral as:

$$P(\theta, t) = \log\left(\frac{I}{I_0}\right) = \int_{ray} \mu(x, y) ds \tag{1.3}$$

θ is the angle of the X-ray source and t is the length of the perpendicular line to the detector from the origin. Together, they are the normal representation of the position of detector, which is represented by a line in x-y coordinates. Taking the ray integral of $\mu(x, y)$ is called the Radon Transformation of $\mu(x, y)$. There are many different methods to recover the image such as back-projection, filtered back-projection, and iterative reconstruction. These methods assume the precise knowledge of the ray path through the object, which is obtained by knowing precise geometric information, such as distance from the X-ray source to the center of the object. In this thesis, we focus on an iterative reconstruction approach by solving a nonlinear least squares problem that will simultaneously reconstruct the image and correct for errors in certain geometrical parameters. We introduce the inverse problem in the next section and set up the

inverse problem in X-ray CT imaging.

1.3 Inverse Problem

An inverse problem is the process of recovering the causal factors from a set of observations. Mathematically,[1] we want to determine the vector x that produces the data b by a forward map A. The goal is to recover the solution x by solving a system of equations.

$$Ax = b \tag{1.4}$$

A is a $m \times n$ matrix, b is a vector of length m, and x is the solution of length n In practice, inverse problems, such as those considered in this thesis, arise from discretizing ill-posed problems. By the definition of 20^{th}-century French mathematician Jacques Hadamard, well-posedness is defined as the following:

- a solutions exists
- the solution is unique
- the solution's behavior changes continuously with the initial conditions.

By the above definition, ill-posed problems are the problems that are not well-posed. Typically, discretizing a ill-posed problem results in a very ill-conditioned matrix A. Using the **singular value decomposition** (SVD), we can write the

[1] In this chapter, we admit an ambiguous use notation, using (x, y) to denote a coordinate in the plane, and x to denote the vector representation of the unknown image. We hope this ambiguity is not too confusing, and that the two usages are clear from the context.

solution x in the following form:

$$\begin{aligned} x &= A^{-1}b \\ &= V\Sigma^{-1}U^T b \\ &= \sum_{i=1}^{n} \frac{u_i^T b}{\sigma_i} v_i \end{aligned} \tag{1.5}$$

Equation (1.5) assumes that $A \in \mathbb{R}^{n\times n}$ is nonsingular, but a similar representation holds for rectangular and singular matrices. $A = U\Sigma V^T$ is the SVD of A, where U and V are orthogonal matrices of size $m \times m$ and $n \times n$ respectively. Σ is a diagonal matrix containing singular values on its diagonal. u_i and v_i are, respectively, column vectors of the orthogonal matrices U and V. The singular values σ_i are ordered in a way such that $\sigma_1 \geqslant \sigma_2 \geqslant \sigma_3 \geqslant .... \geqslant \sigma_n \geqslant 0$. Ill-posed problems typically have small singular values very close to zero. In practice, the data we obtain always contains some noise. Therefore, $b = b_{true} + \eta$, where b_{true} is the noise free data and η is the noise, usually modeled as a vector with random values. Then, small singular values will amplify the error to our solution vector x. Thus, a regularization term is usually added to filter out errors contributed by small singular values. More details about regularization will be discussed in the second chapter where the linear least squares problem is introduced.

Now that we have a basic understanding of the ill-posedness of inverse problems, we formulate the X-ray CT imaging problem using matrix vector product. The data vector b, which is called the Sinogram, contains the projections $p(\theta, t)$ we introduced in the previous section. We formulate b as $b = vec(P(\theta, t))$, where $P(\theta, t)$ is defined

as [9]:

$$P(\theta,t) = \begin{bmatrix} p(\theta_1,t_1) & p(\theta_1,t_2) & ... & p(\theta_1,t_n) \\ p(\theta_2,t_1) & p(\theta_2,t_2) & ... & p(\theta_2,t_n) \\ \vdots & \vdots & \vdots & \\ p(\theta_n,t_1) & p(\theta_n,t_2) & ... & p(\theta_n,t_n) \end{bmatrix} \tag{1.6}$$

The image vector x is represented as a vector[2] form of the function $f(x,y)$, which represents the attenuation at the position $(x,y) \in \mathbb{R}^2$. [9].

$$f(x,y) = \begin{bmatrix} f(x_1,y_1) & f(x_1,y_2) & ... & f(x_1,y_n) \\ f(x_2,y_1) & f(x_2,y_2) & ... & f(x_2,y_n) \\ \vdots & \vdots & \vdots & \\ f(x_n,y_1) & f(x_n,y_2) & ... & f(x_n,y_n) \end{bmatrix} \tag{1.7}$$

The forward operator A is generated based on discretizing the image into pixels and taking into account the X-ray path through each pixel. The complete process to generate the matrix A is convoluted and outside the scope of this thesis. In this thesis, we utilize existing MATLAB software, **IRtools** and **AIRtools** [15][17], to generate 2D tomography images and test data.

From motivation presented in Section 1.1, we have explained that geometry acquisition can vary from angle to angle. Since the forward operator A is determined by geometry parameters r, we form the following nonlinear least squares problem:

$$\arg\min_{x,r} ||A(r)x - b|| \tag{1.8}$$

x represents the image vector. A is the forward operator that is a function of r and maps the true image x to the Sinogram b. In this work we consider only the source-to-object distance, but this model extends to other geometry parameters, such as

[2]We apologize again for the abuse of notation in this chapter. Here the k^{th} entry of the vector x is the function value of $f(x_i,y_i)$, where $k = (j-1)n + i$.

angles. Because the problem is ill-posed, we also need to incorporate a regularization procedure to avoid amplifying noise when reconstructing x. This is discussed in more detail in Section 2.2.3.

Chapter 2

Alternating Minimization Scheme

In order to solve the nonlinear least squares problem, an intuitive approach to consider is the Block Coordinate Descent (BCD) algorithm. In this chapter, we discuss the linear and nonlinear least squares problem involved in BCD. We also briefly discuss the variable projection approach and argue the advantage of BCD over variable projection for our problem in Equation (1.8).

2.1 Block Coordinate Descent

Block Coordinate Descent is a simple approach to solve an optimization problem. Its idea is based on the general Coordinate Descent (CD) algorithm. Because of its lack of sophistication, most optimization researchers have not focused on this approach until recently when CD approaches were found to be computationally competitive to other reputable alternatives in various applications such as machine learning [27]. The iterative Coordinate Descent method has also been found to have fast convergence in CT image reconstruction [28]. Therefore, the Coordinate Descent method is worth investigating in our problem of reconstructing 2D X-ray images.

Consider an unconstrained minimization problem

$$\arg\min_{y} f(y) \tag{2.1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is continuous and differentiable. The Coordinate Descent algorithm minimizes the objective function along a single component i_k of the gradient ∇f at the current point while fixing the rest of the components. For each iteration k, the objective function f is updated along the i_k component of the gradient with a step size of α. The component i_k can be updated in a cyclic fashion in which $i_0 = 1$ [27]:

$$i_{k+1} = [i_k \mod n] + 1, \quad k = 0, 1, 2...$$

The algorithm for Coordinate Descent is the following:

Algorithm 1: Coordinate Descent Algorithm

1 set $k \leftarrow 0$ and choose $y^0 \in \mathbb{R}^n$;
2 **repeat**
3 choose index $i_k \in \{1, 2, ..., n\}$;
4 $y^{k+1} \leftarrow y^k - \alpha[\nabla f(y^k)]_{i_k} e_{i_k}$, $\alpha > 0$;
5 $k \leftarrow k + 1$;
6 **until** *termination criteria*;

where $[\nabla f(y^k)]_{i_k}$ represents the i_k component of the gradient evaluated at the point y^k.

In BCD, instead of updating the objective function along one gradient at a time, minimization is done on a hyper-plane whose dimension is smaller than the dimension of the vector y. In our problem, the solution vector w is composed of two components (x, r), where x is the image vector and r is the geometry vector. We can simply use an alternating minimization method with the vector x in one block whereas r is in

another. In each iteration x is updated while r is fixed and vice versa.

So, we write the algorithm as:

Algorithm 2: Alternating Minimization Scheme to Reconstruct Geometry and Image Parameters

---

1 $k \leftarrow 0$, choose $r_0 \in \mathbb{R}^n$ **repeat**

2 $\quad x_{k+1} = \arg\min_{x_{k+1}} ||b - A(r_k)x||;$

3 $\quad r_{k+1} = \arg\min_{R_{k+1}} ||b - A(r)x_{k+1}||;$

4 **until** *termination criteria*;

We remark that for ill-posed problems, Step 2 typically requires incorporating a regularization procedure to avoid amplifying noise when computing x_{k+1}. This is discussed further in Section 2.2.3. There is another property in our matrix that makes this alternating minimization scheme favorable. The matrix A can be written as

$$A(r) = \begin{bmatrix} A(r_1) \\ \vdots \\ A(r_i) \\ \vdots \\ A(r_k) \end{bmatrix}$$

for $i = 1, 2, 3..., k$, where k is the number of the set of angles where the geometry parameters differ. For example, consider the number of angles as a discrete set of integer values from 0 to 359. If source-to-object distance differs at each angle, there would be $k = 360$ parameters of r. If the source-to-object distance stays the same for a set of angles and there are k such sets, the number of the set of angles where the geometry parameters differ is k. To make the writing succinct, we refer to this as the **number of angles** for the rest of the thesis.

The separability of the geometry parameters allows us to solve for a much smaller system at a time, and they are completely independent so they can be updated simultaneously on a parallel computing architecture, which dramatically lowers the computing time. In fact, the dimension of the parameter $r(i)$ is one, where $r(i)$ represents the i^{th} entry in the geometry parameter. Comparing to this alternating minimization scheme, Variable Projection [20][22] would not be able to utilize this matrix property because its matrix multiplication process would take away the separability property. Thus, it makes this alternating minimization scheme more worthwhile to investigate.

In order to perform the alternating minimization algorithm, we introduce linear least squares solvers and nonlinear least squares solvers respectively in the following sections.

2.2 Linear Least Squares Problem

It is common in practice that the exact solution of a system of equations does not exist. The method of least squares minimizes the sum of squared residuals to find the best approximate solution. When an exact solution exists, the minimum of the residual is zero. In our X-ray tomography problem, we have used **IRhybrid_lsqr** from **IRTools** package in MATLAB to solve the least squares problem when the geometry parameter r is given:

$$\min_x ||A(r)x - b||_2^2$$

Before presenting the method, we provide some mathematical background including normal equations, QR factorization for least squares, regularization, parameter choice methods, and the LSQR algorithm in the following sections.

2.2.1 Normal Equations

A theoretical approach to compute a linear least squares solution of an over-determined system is the normal equations,

$$A^T A\hat{x} = A^T b.$$

The intuition behind this approach is to find the projection p of the data vector b on the column space of A. The smallest possible error $e = b - p$ is orthogonal to the column space. By this approach if A has full column rank, we obtain $x = (A^T A)^{-1} A^T b$. Since the condition number of $A^T A$ is the squared of condition number of A, we can lose accuracy by forming the normal equations. Since our problem is ill-posed and typically very ill-conditioned, for the purpose of this thesis, we do not spend too much content on the normal equations approach. More details can be found in [24].

2.2.2 QR Factorization and Least Squares Problem

In this subsection, we introduce a method to solve the least squares (LS) problem using QR factorization. We write the QR factorization of a full column rank $m \times n$ matrix as:

$$A = QR = Q \begin{bmatrix} R_1 \\ 0 \end{bmatrix}$$

where Q is $m \times m$ orthogonal matrix and R_1 is an upper triangular matrix. The least squares problem is equivalent to

$$\begin{aligned} ||b - Ax||_2^2 &= ||Q^T b - Rx||_2^2 \\ &= \left\| \hat{b} - \begin{bmatrix} R_1 \\ 0 \end{bmatrix} x \right\|_2^2 \\ &= \left\| \begin{bmatrix} \hat{b_1} - R_1 x \\ \hat{b_2} \end{bmatrix} \right\|_2^2 \\ &= ||\hat{b_1} - R_1 x||_2^2 + ||\hat{b_2}||_2^2 \end{aligned}$$

where $\hat{b} = Q^T b$ and $\hat{b_1}$ is $n \times 1$ and $\hat{b_2}$ is $(m - n) \times 1$. Since $||\hat{b_2}||_2^2$ depends solely on the data vector, we can easily see that minimizing $||b - Ax||_2^2$ is equivalent to finding the solution to $R_1 x = \hat{b_1}$. Solving an upper triangular system is easily done by back substitution. In MATLAB, the backslash operator will easily compute this for us. The least squares problem can also be solved by using the SVD. Since the SVD is more computationally expensive than QR, it is generally not used to solve least squares problems, especially when A is a large matrix. The rest of this subsection focuses on QR, but we return to the SVD in Section 2.2.3 when we discuss regularization.

In practice, it is very common to encounter a rank deficient matrix where $A \in \mathbb{R}^{m \times n}$ and $rank(A) < n$. So, here we describe the QR factorization with column pivoting to solve the rank deficient least squares problem. We write QR with column pivoting as:

$$A\Pi = QR = Q \begin{bmatrix} R_{11} & R_{12} \\ 0 & 0 \end{bmatrix} \tag{2.2}$$

where Π is a permutation matrix. This process can be done by using Householder reflections. A Householder reflection is a matrix of the form $Q_u = I - 2uu^T$ where $||u||_2 = 1$. The matrix Q_u is orthogonal and symmetric. The name reflection comes

from the fact that $Q_u x$ is the reflection of x in the plane through 0 perpendicular to u [10]. Given a vector x, it is easy to find a Householder reflection such that all the entries are zeroed out in x except the first one. This is extremely helpful in QR factorization because we can apply Householder reflections to zero out entries below the diagonal entries in the matrix until we get an upper triangular matrix. The procedure to do QR factorization with column pivoting using Householder reflections is the following:

1. Set $i = 1$ and let m and n be the number of rows and columns in the matrix A
2. Choose the column with the largest 2-norm from sub-matrix $A(i : m, i : n)$
3. Swap with the i^{th} column
4. Perform a Householder reflection to zero out every entry below the i^{th} entry in the i^{th} column
5. $i = i + 1$ Repeat step 2-4
6. Stop if the column with the largest 2-norm in the sub-matrix is equal to zero

where $A(i : m, i : n)$ is MATLAB notation and represents rows of A from i to m and columns of A from i to n. An illustration of this procedure can be shown in the following steps to produce the QR factorization of A with column pivoting.

After choosing the column with the largest 2-norm and swapping it with the first column by using the permutation matrix Π_1, we multiply A with the Householder reflection matrix Q_1^T

$$A \leftarrow Q_1^T A \Pi_1 = \left[\begin{array}{c|c} * & * \\ \hline 0 & \\ \vdots & A_{22}^1 \end{array}\right]$$

Then, we choose the column that has the largest 2-norm in the sub-matrix A_{22}^1 and swap it with the first column in A_{22}^1 using a permutation matrix Π_2. Then, we apply

the Householder reflection Q_2^T.

$$A \leftarrow Q_2^T Q_1^T A \Pi_1 \Pi_2 = \left[\begin{array}{cc|c} * & * & ... \\ 0 & * & ... \\ \hline 0 & 0 & \\ \vdots & \vdots & A_{22}^2 \end{array}\right]$$

Now, we repeat the process on A_{22}^2 until the matrix is the form:

$$A \leftarrow Q_k^T \cdots Q_2^T Q_1^T A \Pi_1 \Pi_2 \cdots \Pi_k = \begin{bmatrix} R_{11} & R_{12} \\ 0 & 0 \end{bmatrix}$$

We can write the QR factorization of the rank deficient matrix A as $A\Pi = QR$ where $\Pi = \Pi_1 \Pi_2 ... \Pi_k$ and $Q = Q_1 Q_2 .. Q_k$.

For a rank deficient least squares problem, we can write the least squares problem as:

$$||b - Ax||_2^2 = ||b - A\Pi\Pi^T x||_2^2 = ||Q^T b - R\hat{x}||_2^2$$

where $\hat{x} = \Pi^T x$.

Let c, $y \in \mathbb{R}^r$, $d \in \mathbb{R}^{m-r}$, $z \in \mathbb{R}^{n-r}$ where $Q^T b = \begin{bmatrix} c \\ d \end{bmatrix}$ and $\hat{x} = \begin{bmatrix} y \\ z \end{bmatrix}$ Then the least squares problem becomes:

$$\begin{aligned} ||b - A\hat{x}||_2^2 &= \left\| \begin{bmatrix} c \\ d \end{bmatrix} - \begin{bmatrix} R_{11} & R_{12} \\ 0 & 0 \end{bmatrix} \begin{bmatrix} y \\ z \end{bmatrix} \right\|_2^2 \\ &= ||c - R_{11} y - R_{12} z||_2^2 + ||d||_2^2 \end{aligned}$$

The norm will be minimized if we minimize the first norm in the above equation for some y and z. We can solve for y by letting $z \in \mathbb{R}^{n-r}$ be an arbitrary vector. So, we minimize the original least squares problem by finding the solution to $c = R_{11} y + R_{12} z$.

A common choice for the sub-vector z is the zero vector which would simplify our calculation of $R_{12}z$. Then, the solution to the least squares problem is

$$x = \Pi\hat{x} = \Pi \begin{bmatrix} y \\ z \end{bmatrix}$$

Now that we established some basic tools to solve the rank deficient least squares problem. We still need the tools that smooth the ill-posedness in our 2D X-ray tomography problem.

2.2.3 Regularization

Typically in inverse problems, the data we obtain is not the exact data. This is also the case in our X-ray tomography problem. Ideally, we want to solve $Ax = b$ but the linear system that we actually solve is:

$$A\hat{x} = b = b_{true} + \eta \tag{2.3}$$

where η is the noise in our measurement data, b_{true} is the noise-free data, $A \in \mathbb{R}^{m\times n}$, and $b \in \mathbb{R}^m$. For the simplicity of the following proofs, we assume A is full rank and $m = n$. The observation will also hold for more general cases when we compute a pseudo-inverse instead of an exact inverse.

By using the SVD of A where σ_i is the i^{th} singular value and u_i, v_i are the i^{th} column of the left and right orthogonal matrix U and V, we first recall that $Av_i = \sigma_i u_i$, and we note that we can find scalars α_i and η_i such that

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

Using these relations, we obtain

$$\begin{aligned} b &= Ax + \eta \\ &= A\sum_{i=1}^{n} \alpha_i v_i + \sum_{i=1}^{n} \eta_i u_i \\ &= \sum_{i=1}^{n} \alpha_i A v_i + \sum_{i=1}^{n} \eta_i u_i \\ &= \sum_{i=1}^{n} \alpha_i \sigma_i u_i + \sum_{i=1}^{n} \eta_i u_i \\ &= \sum_{i=1}^{n} (\alpha_i \sigma_i + \eta_i) u_i \end{aligned} \tag{2.4}$$

Then, notice that

$$\begin{aligned} A^{-1} u_i &= V \Sigma^{-1} U^T u_i \\ &= V \Sigma^{-1} e_i \\ &= \frac{1}{\sigma_i} v_i \end{aligned} \tag{2.5}$$

Thus,

$$\begin{aligned} \hat{x} &= A^{-1} b \\ &= \sum_{i=1}^{n} (\alpha_i \sigma_i + \eta_i) A^{-1} u_i \\ &= \sum_{i=1}^{n} \left(\alpha_i + \frac{\eta_i}{\sigma_i} \right) v_i \end{aligned} \tag{2.6}$$

From this result we observe that the noise in the data is magnified by the small singular values. Since $\sigma_1 \geqslant \sigma_2 \geqslant ... \geqslant \sigma_n \geqslant 0$, larger indices correspond to smaller singular values. The computed solution is dominated by noise amplified by division of small singular values. Thus, we need regularization schemes to filter out this noise.

In particular, a regularized solution can be written as:

$$\hat{x}_{reg} = \sum_{i=0}^{n} \phi_i \left(\alpha_i + \frac{\eta_i}{\sigma_i} \right) v_i \tag{2.7}$$

where the scalar $0 \leqslant \phi_i \leqslant 1$ is called a filter factor. As σ_i decreases, the filter factors should approach zero so that the noise contributed by the small singular values are filtered out. In the following paragraphs, we introduce the Truncated Singular Value Decomposition (TSVD) and Tikhonov Regularization.

Truncated Singular Value Decomposition

Since the noise is magnified by the small singular value, the most intuitive approach is to cut off the small singular values by setting them to zero. This is called the Trucated SVD regularization. The TSVD solution to the inverse problem is given by

$$\hat{x}_{reg} = \sum_{i=1}^{k} \frac{b^T u_i}{\sigma_i} v_i \tag{2.8}$$

where $k \leqslant n$. Equation (2.8) is equivalent to setting $\phi_i = 0, \forall i > k$ and $\phi_i = 1, \forall i \leqslant k$. The critical part in the TSVD is identifying the threshold k. One approach is the elbow method – choosing k at a significant drop-off of singular values, as illustrated by Figure 2.1.

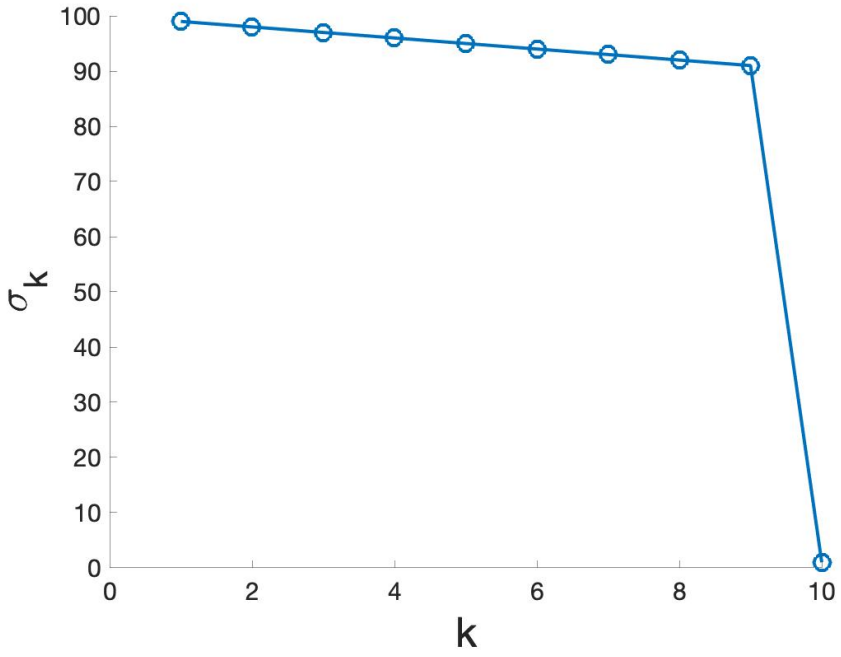

Figure 2.1: The singular values plot of a 10×10 diagonal matrix whose singular values are respectively 99, 98 ... 91, and 1

In this case, $k = 9$ can be easily identified as the threshold for the TSVD approach. However, typically in inverse problems, singular values decay smoothly, as illustrated by Figure 2.2.

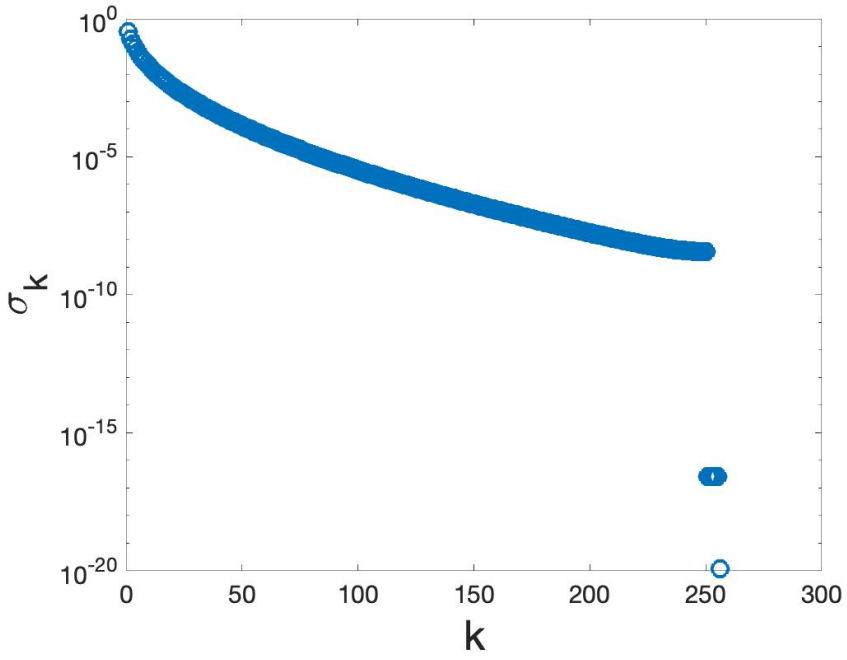

Figure 2.2: The singular values of the 256×256 test problem *heat* generated from Regularization Tools in MATLAB

In cases like Figure 2.2, a reliable cut-off threshold for singular values is hard to find. Note that $k = 250$ is not a good choice for the cut-off because the singular values are very close to zero for smaller indices k (e.g. $\sigma_{200} \approx 10^{-8}$). Also, since the matrix A can be very large in practice, it is not feasible to compute the SVD of large matrices. Therefore, we need more advanced regularization techniques to deal with smoothly decaying singular values.

Tikhonov Regularization

We introduce the classical Tikhonov Regularization to solve ill-posed problems. The regularized solution $\hat{x}_{reg}$ is the unique solution to the following:

$$\min_x ||Ax - b||_2^2 + \lambda^2 ||x||_2^2 \tag{2.9}$$

where λ is the regularization parameter that controls the smoothness of the regularized solution. Equation (2.9) is equivalent to the following:

$$\min_x \left\| \begin{bmatrix} A \\ \lambda I \end{bmatrix} x - \begin{bmatrix} b \\ 0 \end{bmatrix} \right\|_2^2 \tag{2.10}$$

Then the normal equations for this least squares problem can be written as:

$$(A^T A + \lambda^2 I)\hat{x}_{reg} = A^T b. \tag{2.11}$$

From the normal equations, we can obtain the following:

$$\begin{aligned} \hat{x}_{reg} &= (A^T A + \lambda^2 I)^{-1} A^T b \\ &= (V\Sigma^2 V^T + \lambda^2 I)^{-1} A^T b \\ &= (\lambda^2 (\frac{1}{\lambda^2} V\Sigma^2 V^T + VIV^T))^{-1} A^T b \\ &= (\lambda^2 V (\frac{1}{\lambda^2}\Sigma^2 + I)V^T)^{-1} A^T b \\ &= (V(\Sigma^2 + \lambda^2 I)V^T)^{-1} A^T b \\ &= V(\Sigma^2 + \lambda^2 I)^{-1} V^T V \Sigma U^T b \\ &= \sum_{i=1}^{n} \frac{\sigma_i}{\sigma^2 + \lambda^2} v_i u_i^T b \\ &= \sum_{i=1}^{n} \phi_i \frac{b^T u_i}{\sigma_i} v_i \end{aligned} \tag{2.12}$$

where the filtering factor is $\phi_i = \frac{\sigma_i^2}{\sigma_i^2 + \lambda^2}$. We assumed A has full rank for the simplicity of the proof. The pseudo-inverse can be used for the rank deficient case and the resulting equation would sum from $i = 1$ to $rank(A) = r$ instead of the full column rank n. The modified Tikhonov regularization is not too much different from the classical Tikhonov except that it uses the 2-norm of Lx instead of x in Equation(2.9),

where L is a $p \times n$ matrix with $p \leqslant n$.

$$\min_x ||Ax - b||_2^2 + \lambda^2 ||Lx||_2^2$$

In contrast to TSVD where singular values after σ_k are cut off, the Tikhonov regularization applies a smoother filter to all the singular values. Given a good regularization parameter, the large singular values would not be affected too much whereas the small values would be gradually filtered more as they approach zero. The quality of the filtering depends on how we choose the regularization parameter. If $\lambda = 0$, then $\phi = 1$ and we are directly calculating the inverse (or pseudo-inverse) solution. If we select a very large $\lambda \gg \sigma_1$, ϕ_i would approach zero and we would have over-smoothed the solution. In the next section, we introduce some methods that choose the regularization parameters.

2.2.4 Parameter Choice Methods

The choice of regularization parameter is critical to the quality of the regularized solution. Parameter choice methods can usually be divided into two classes depending on their assumption about error norm $||\eta||_2^2 = ||b - b_{true}||_2^2$ where b is the measured data and b_{true} is the noise free data [16]. The first class contains methods based on a good estimate of $||\eta||_2^2$. The second class includes methods that are not based on a good estimate of $||\eta||_2^2$, but seek to extract this information from the given right hand side b. In this section, we introduce the Generalized Cross-Validation method (GCV) which is a popular method in the second class. The underlying idea of GCV is that a good regularization parameter should predict missing values. For example, if some data point b_i is missing in the right hand side, the regularized solution should predict this missing value well. GCV can be written as:

$$G(\lambda) = \frac{n||A\hat{x}_{reg} - b||_2^2}{trace(I_m - AA_F^\dagger)^2} \tag{2.13}$$

where $\hat{x}_{reg}$ is the regularized solution, $A_F^\dagger = \sum_{i=1}^n \phi_i \frac{v_i u_i^T}{\sigma_i} = V\Sigma_F^\dagger U^T$, and $\Sigma_F^\dagger = diag(\frac{\phi_1}{\sigma_1}, \frac{\phi_2}{\sigma_2}, ..., \frac{\phi_n}{\sigma_n})$. The goal is to find λ such that $G(\lambda)$ is minimized. Now we simplify the numerator and the denominator of $G(\lambda)$ respectively:

$$\begin{aligned} ||A\hat{x}_{reg} - b||_2^2 &= ||\Sigma V^T x_{reg} - U^T b||_2^2 \\ &= ||\Sigma V^T V \Sigma_F^\dagger U^T b - U^T b||_2^2 \\ &= ||(\Sigma\Sigma_F^\dagger - I)U^T b||_2^2 \end{aligned}$$

$$\begin{aligned} trace(I_m - AA_F^\dagger) &= trace((I_m - U\Sigma\Sigma_F^\dagger U^T)) \\ &= trace(U(I_m - \Sigma\Sigma_F^\dagger)U^T) \\ &= trace(I_m - \Sigma\Sigma_F^\dagger) \end{aligned}$$

Thus, $G(\lambda)$ becomes:

$$G(\lambda) = \frac{n||(\Sigma\Sigma_F^\dagger - I)U^T b||_2^2}{trace(I_m - \Sigma\Sigma_F^\dagger)} \tag{2.14}$$

Since $\phi_i = \frac{\sigma_i^2}{\sigma_i^2 + \lambda^2}$ in the Tikhonov case,

$$\Sigma_F^\dagger = diag\left(\frac{\phi_1}{\sigma_1}, \frac{\phi_2}{\sigma_2}, ..., \frac{\phi_n}{\sigma_n}\right) = diag\left(\frac{\sigma_1}{\sigma_1^2 + \lambda^2}, \frac{\sigma_2}{\sigma_2^2 + \lambda^2}, ..., \frac{\sigma_n}{\sigma_n^2 + \lambda^2}\right)$$

Then we can write $G(\lambda)$ in the case of Tikhonov regularization as:

$$G(\lambda) = \frac{n\sum_{i=1}^n (\frac{\hat{b_i}}{\sigma_i^2 + \lambda^2})^2}{(\sum_{i=1}^n \frac{1}{\sigma_i^2 + \lambda^2})^2} \tag{2.15}$$

where $\hat{b} = U^T b$. Now we can solve for λ by using the function **fminbnd** in

MATLAB which is based on golden section search and parabolic interpolation. In practical applications, several studies have found that occasionally GCV would drastically under-smooth the solution by choosing the regularization parameter too small [8] [13] [25]. In [5], GCV has been found to over-smooth the solution in Lanczos-hybrid methods. Thus, here we describe the weighted GCV method:

$$G(w, \lambda) = \frac{n||A\hat{x}_{reg} - b||_2^2}{trace(I_m - wAA_F^{\dagger})^2} \tag{2.16}$$

where w is the weight parameter that determines the function $G(w, \lambda)$ along with λ. When $w = 1$, we have the non-weighted version of GCV like in Equation (2.13). When $w > 1$, the solution is smoother. When $w < 1$, the solution is less smooth. So far in weighted-GCV literature, only experimental approaches have been used to determine the value for w. The function **IRhybrid_lsqr** that we use to solve the linear least squares problem chooses w through an adaptive process [5]. This update process utilizes the earlier well-conditioned components to stabilize the impact from small singular values. Although the solution can be over-smoothed in the first few iterations due to a relatively large λ used for well-conditioned problem, these solutions are discarded. While this update method for w under-smooths values for large k, a method like GCV will be used to choose a stopping iteration so that k will not be too large. In the next subsection, we introduce the LSQR algorithm which would finally lead us to the hybrid LSQR algorithm [5].

2.3 The LSQR Algorithm

To introduce the LSQR algorithm, we need to first illustrate the **Golub-Kahan bidiagonalization** process [2].

Golub-Kahan bidiagonalization

We want to find the following factorization of matrix $A \in \mathbb{R}^{m\times n}$:

$$A = U \begin{bmatrix} B \\ 0 \end{bmatrix} V^T \tag{2.17}$$

where $U = (u_1, u_2, ..., u_m)$ and $V = (v_1, v_2, ..., v_n)$ are orthogonal matrices. B is a bidiagonal matrix:

$$B = B_n = \begin{bmatrix} \alpha_1 & & & \\ \beta_2 & \alpha_2 & & \\ & \beta_3 & \ddots & \\ & & \ddots & \alpha_n \\ & & & \beta_{n+1} \end{bmatrix} \tag{2.18}$$

where B_n stands for the bidiagonal matrix produced from the n^{th} iteration of the Golub-Kahan bidiagonalization.

If we set $U_1 = (u_1, u_2, ..., u_{n+1})$, then we have:

$$AV = U_1 B, \quad A^T U_1 = V B^T \tag{2.19}$$

Equating the j^{th} columns in these two equations we get:

$$\begin{aligned} A^T u_j &= \beta_j v_{j-1} + \alpha_j v_j \\ A v_j &= \alpha_j u_j + \beta_{j+1} u_{j+1} \end{aligned} \tag{2.20}$$

We set $\beta_1 v_0 \equiv 0$ and $\alpha_{n+1} v_{n+1} \equiv 0$. In the Golub-Kahan bidiagonalization, a starting vector $v_1 \in \mathbb{R}^n$ is chosen. Here, we introduce a variant of this algorithm which is more suited for solving least squares problems. We choose a starting vector $u_1 \in \mathbb{R}^m$ instead [2] [21]. Given $u_1 \in \mathbb{R}^m$, $||u_1||_2 = 1$, we recursively generate the vectors

$v_1, u_2, v_2, ..., u_{m+1}$ and corresponding elements in B_n respectively for $j = 1, 2, ...,$

$$r_j = A^T u_j - \beta_j v_{j-1}, \quad \alpha_j = ||r_j||_2, \quad v_j = r_j/\alpha_j \tag{2.21}$$

$$p_j = Av_j - \alpha_j u_j, \quad \beta_{j+1} = ||p_j||_2, \quad u_{j+1} = p_j/\beta_{j+1} \tag{2.22}$$

For this bidiagonalization scheme we have

$$u_j \in K_j(AA^T, u_1), \quad v_j \in K_j(A^T A, A^T u_1)$$

where $u_1, ..., u_j$ and $v_1, ..., v_j$ are orthogonal bases for these two Krylov spaces.

Best Approximation in the Krylov subspace

Given the linear least squares problem, we compute a sequence of approximate solutions [2].

We set

$$\beta_1 u_1 = b, \quad \alpha_1 v_1 = A^T u_1 \tag{2.23}$$

and for $j = 1, 2...,$

$$\begin{aligned} \beta_{j+1} u_{j+1} &= Av_j - \alpha_j u_j \\ \alpha_{j+1} v_{j+1} &= A^T u_{j+1} - \beta_{j+1} v_j \end{aligned} \tag{2.24}$$

where $\alpha_{j+1} \geqslant 0$ and $\beta_{j+1} \geqslant 0$ are determined so that $||u_{j+1}||_2 = ||v_{j+1}||_2 = 1$. After k steps we have

$$V_k = (v_1, ..., v_k), \quad U_{k+1} = (u_1, ...u_{k+1})$$

and

$$B_k = \begin{bmatrix} \alpha_1 & & & \\ \beta_2 & \alpha_2 & & \\ & \beta_3 & \ddots & \\ & & \ddots & \alpha_k \\ & & & \beta_{k+1} \end{bmatrix} \in \mathbb{R}^{(k+1)\times k} \tag{2.25}$$

The recurrence relation from Equation (2.23) and (2.24) can be written in the matrix form:

$$\beta_1 U_{k+1} e_1 = b \tag{2.26}$$

$$AV_k = U_{k+1} B_k, \quad A^T U_{k+1} = V_k B_k^T + \alpha_{k+1} v_{k+1} e_{k+1}^T \tag{2.27}$$

$x_k \in K_k$ is an approximate solution to x in the Krylov space, where $K_k = K_k(A^T A, A^T b)$. Since $K_k = span(V_k)$, we can write

$$x_k = V_k y_k. \tag{2.28}$$

Then multiplying y_k with the first equation in Equation (2.27), we obtain $Ax_k = AV_k y_k = U_{k+1} B_k y_k$, and then from Equation (2.26)

$$b - Ax_k = U_{k+1} t_{k+1}, \quad t_{k+1} = \beta_1 e_1 - B_k y_k. \tag{2.29}$$

Then $||b - Ax_k||_2$ is minimized over all $x_k \in span(V_k)$ by taking y_k to be the solution to the least squares problem:

$$\min_{y_k} ||B_k y_k - \beta_1 e_1||_2. \tag{2.30}$$

Note that we use the fact that 2-norm is invariant under orthogonal transformation

of U_{k+1}.

The LSQR algorithm

Since B_k is bidiagonal the sub-problem above can be solved by QR decomposition of B_k,

$$Q_k^T B_k = \begin{bmatrix} R_k \\ 0 \end{bmatrix}, \quad Q_k^T(\beta_1 e_1) = \begin{bmatrix} f_k \\ \phi_{k+1} \end{bmatrix} \tag{2.31}$$

where R_k is upper bidiagonal and

$$f_k = \begin{bmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_k \end{bmatrix}$$

The matrix Q_k is the product of Givens rotations $Q_k = G_{k,k+1}G_{k-1,k}, ..., G_{1,2}$ chosen to eliminate one subdiagonal element at a time. The solution vector y_k and the residual vector t_{k+1} are obtained from

$$R_k y_k = f_k, \quad t_{k+1} = Q_k^T \begin{bmatrix} 0 \\ \phi_{k+1} \end{bmatrix} \tag{2.32}$$

Then, we can solve for y_k by doing backward-substitution. Note that the QR factorization does not have to be computed from scratch every single iteration. Once a column is added, we can simply use a Givens rotation to eliminate the subdiagonal element.

Then, the best approximation x_k in Krylov space K_k can be computed by $x_k = V_k y_k$. It is also important to note that we do not have to store all the vectors $v_1, .., v_k$ in V_k. Instead, only storing one extra n-vector is needed. This can be done by using a

simple recursion for computing x_k from x_{k-1} [21].

2.3.1 Hybrid LSQR

All the subsections up till now have prepared the background in understanding the Hybrid LSQR algorithm (a weighted-GCV method for Lanczos-Hybrid Regularization). In Section 2.2.3, we discussed that regularization methods are needed to solve the ill-posed inverse problem in Equation (2.5). Since quality of regularization methods depends on the regularization parameter, it is helpful to consider various methods such as the discrepancy principle, generalized cross-validation, and L-curve. While having their merits, they all have disadvantages. The discrepancy principle requires information about the noise. Efficient implentation of GCV requires the singular value decomposition of A, which is impractical for large scale problems like ours. In L-curve, it's necessary to solve for several regularization parameters although this can be alleviated by exploiting redundancies in certain iterative methods [5] [4] [14].

In the previous subsection, we have shown that LSQR projects the linear least squares problem onto a Krylov subspace of small (increasing) dimensions. When being applied to ill-posed problems, LSQR exhibits a semi-convergence behavior which means that early iterations construct information related to the solution while later iterations construct information related to noise [5]. This can be compensated by applying a direct regularization method such as Tikhonov or TSVD, which can be solved cheaply on a small scale problem of the reduced linear least squares in the Krylov subspace. Comparing to LSQR, this hybrid method can effectively stablize the iterations [5]. Although at each iteration a new regularization parameter must be chosen, it is not computationally expensive for the projected problem. As mentioned in Section 2.2.4, GCV can occasionally overestimate or underestimate the regularization parameters, which would result in over-smoothing or under-smoothing the solutions. Thus, the adaptive weighted GCV, Equation (2.16), is used as the parameter choice

method in this hybrid approach.

To summarize the method, we project the large scale linear least squares problem onto a low-dimensional Krylov subspace where we can inexpensively apply a direct regularization method like the adaptive weighted-GCV.

In practice, we can directly use the **IRhybrid_lsqr** method from IRTools in MATLAB. The code is very user friendly and large linear least squares problems can be solved using various iterative methods that it provides.

2.4 Nonlinear Least Squares

In the alternating minimization scheme we proposed, we iteratively solve the image and the geometry parameters. While we have discussed methods to solve the linear least squares problem in the previous section, we need other tools to solve the nonlinear least squares problem:

$$\min_{r} ||A(r)x - b||_2 \tag{2.33}$$

where x is approximated by the linear least squares solution we obtained by using hybrid LSQR in Section 2.3.

We utilize the implicit filtering method which solves the bound-constraint optimization problem for which the derivative information is not available [18]. Since we do not have the derivative information of our objective function and a reasonable bound can be established for the geometry parameters in our tomographic reconstruction problem, implicit filtering serves as a good tool to solve our problem. In contrast to coordinate search, implicit filtering builds the local model of the objective function using a quasi-Newton method.

The implicit filtering uses a quasi-Newton method or a Gauss-Newton method to accelerate convergence. To establish some background knowledge, we introduce the

quasi-Newton method, Gauss-Newton method, and implicit filtering respectively.

2.4.1 Quasi-Newton and Gauss-Newton Method

To provide a foundation for understanding quasi-Newton and Gauss-Newton methods, we first introduce Newton's method.

Newton's Method

Newton's method is an iterative method for finding the roots $f(x) = 0$ of the differentiable function $f(x)$. The most basic version of Newton's method is described here. Given a single variable function f defined for real variable x, an initial guess x_0, and the derivative f', we employ the following iterative scheme to solve for the root, for $k = 0, 1, 2...$

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

If the function satisfies sufficient assumptions and the initial guess is close, x_{k+1} is a better approximation than x_k. Eventually, the iteration converges. The geometric interpretation of Newton's method is that the coordinate $(x_{k+1}, 0)$ is the intersection of the x-axis and the tangent line of $f(x)$ at point $(x_k, f(x_k))$.

In optimization, we apply Newton's method to the derivative of $f(x)$ and find the solution to $f'(x) = 0$. Suppose $f : \mathbb{R} \to \mathbb{R}$ is a twice-differentiable function and we want to minimize $f(x)$ on $x \in \mathbb{R}$. The second order derivative of f around x_k is approximated as:

$$f(x_k + t) \approx f(x_k) + f'(x_k)t + \frac{1}{2}f''(x_k)t^2$$

where $t = x_{k+1} - x_k$. If the second derivative is positive, we can find the minimum of this quadratic convex function by setting the derivative to zero

$$0 = \frac{d}{dt}(f(x_k) + f'(x_k)t + \frac{1}{2}f''(x_k)t^2) = f'(x_k) + f''(x_k)t$$

Then, the iteration scheme can be written as

$$x_{k+1} = x_k + t = x_k - \frac{f'(x_k)}{f''(x_k)} \tag{2.34}$$

Quasi-Newton Method

Quasi-Newton methods are used to either find the root or the minima (maxima) of a function. While the full Newton's method requires the Jacobian when finding zeros or Hessian when finding minima (maxima), quasi-Newton methods can be used when the Jacobian or Hessian is unavailable or too expensive to compute at each iteration. As in Newton's method, one uses a second-order approximation to find the minimum of the function $f(x)$:

$$f(x_k + \triangle x) \approx f(x_k) + \nabla f^T \triangle x + \frac{1}{2} \triangle x^T B \triangle x$$

where ∇f is the gradient, B is an approximate to the Hessian matrix. The gradient of this approximation is:

$$\nabla f(x_k + \triangle x) \approx \nabla f(x_k) + B \triangle x$$

Setting this gradient to zero gives

$$\triangle x = -B^{-1} \nabla f(x_k)$$

$$x_{k+1} = x_k + \triangle x$$

Strictly speaking, any method that replaces the exact Jacobian in the Newton method with an approximation is a quasi-Newton method. There are many methods to update the approximate Hessian matrix B [12][3][6].

Gauss-Newton Method

The Gauss-Newton method is a modification of Newton's method to solve nonlinear least squares problems. It can only be used to minimize the sum of squared function values but it does not require the second derivative information [19].

Given m functions $\rho = (\rho_1, \rho_2, .., \rho_m)$ and n variables $(\beta = \beta_1, \beta_2, ..., \beta_n)$ with $m \geqslant n$, the Gauss-Newton method iteratively solve for variables that minimize the function

$$S(\beta) = \sum_{i=1}^{m} \rho_i(\beta)^2$$

The Gauss-Newton method can be derived from Taylor series at some value $\beta^{(s)}$:

$$\rho(\beta) = \rho(\beta^{(s)}) + J_\rho(\beta^{(s)}) \triangle$$

where $\triangle = \beta - \beta^{(s)}$ and J_ρ is a Jacobian matrix with entries

$$(J_\rho)_{ij} = \frac{\partial \rho_i(\beta^{(s)})}{\partial \beta_j}$$

Thus, the minimization problem of $\rho(\beta)$ is converted to a linear least squares problem

$$\min ||\rho(\beta^{(s)}) + J_\rho(\beta^{(s)}) \triangle||_2^2$$

Using the normal equation, we derive the Gauss-Newton iterative scheme

$$\beta^{(s+1)} = \beta^{(s)} - (J_\rho^T J_\rho)^{-1} J_\rho^T \rho(\beta^{(s)})$$

2.4.2 Implicit Filtering

Implicit filtering extends coordinate search by approximating a gradient using the values of f on the stencil, uses that approximate gradient to build a model of f, and then searches for a better point using information from the model [18].

The basic idea behind the simplest possible form of implicit filtering is the following:

Given a base point x, a function value $f(x)$, a scale h, the algorithm begins evaluating at the $2N$ points on the stencil

$$S(x, h) = \{z | z = x \pm he_i\}$$

where e_i is the unit vector at the i^{th} direction. Let the minimum of the evaluated function at those $2N$ points be f_{min}. If $f_{min} \geqslant f_{base}$, reduce the scale h and signal stencil failure. Otherwise, calculate the stencil gradient of the function f, set the search direction d as the negative of the stencil gradient and perform the Armijo line search which takes in a starting point and a search direction, and determines the amount to move in a given direction that adequately reduces the objective function. Let the output value of the line search be f_{newt}. We then compare the value f_{min},f_{newt}, and f_{base}. Clearly, if $f_{newt} < f_{min}$, we accept the line search result as the new point. If would be less clear if $f_{min} < f_{newt} < f_{base}$. In the default setting of the MATLAB code **imfil**, it accepts the quasi-Newton step if $f_{newt} < f_{base}$ because numerical experiments suggest this setting performs better than favoring the stencil.

To accelerate convergence, Gauss-Newton or quasi-Newton method is used to update the search direction d. Also, bound constraints are used to scale the variables to roughly the same size.

To apply **imfil** to our problem, we can directly use the following line of code after providing a cost function, budget, bounds, and additional data.

```
options=imfil_optset('simple_function',1,'least_squares',1);
r(j) = imfil(r_0(j),@CostFun, budget, bounds, options, extra)
```

where $r(j)$ is the j^{th} entry of the geometry vector $r \in \mathbb{R}^k$, where k is the number of angles.

The development of implicit filtering and its implementation in MATLAB is a multi-year collaborative effort led by Dr. C. T. Kelley. An attempt to explain it comprehensively and clearly is beyond the scope of this thesis. More information on implicit filtering can be found on the *Implicit Filtering* book [18].

Chapter 3

Acceleration Schemes

In the previous chapter, we have introduced the alternating minimization scheme and the methods we use to solve least squares problems. In this chapter, we introduce methods that will accelerate the convergence of our minimization scheme.

3.1 Accelerated Block Coordinate Descent

Since we can divide variables in our least squares problem into two blocks – geometry parameters r and image x, it makes sense for us to directly investigate methods that accelerate the alternating minimization. We implemented **an inexact majorized Accelerated Block Coordinate Descent** (imABCD), which is an accelerated as well as inexact version of alternating minimization. This method can be applied to a four-block problem by dividing it into two larger blocks, but in our problem we do not have to do so. Theoretically, the proposed inexact acceleration method has a complexity of $O(\frac{1}{k^2})$ [7]. In our implementation, we simplify the accelerated algorithm as the following:

Algorithm 3: Inexact majorized Accelerated Block Coordinate Descent

Step 1 **Initialization** Let $t_0 = 1$ given $r_0 \in \mathbb{R}^N$ and $x_0 \in \mathbb{R}^n$, perform the following step for $k \geqslant 1$:

Step 2 $\tilde{r_k} = \arg\min_r ||b - A(r)x_{k-1}||_2^2;$

$\tilde{x_k} = \arg\min_x ||b - A(\tilde{r_k})x||_2^2 + \lambda^2||x||_2^2;$

Step 3 $\tilde{w_k} = (\tilde{x_k}, \tilde{r_k})$;

$t_k = \frac{1}{2}(1 + \sqrt{1 + 4t_{k-1}^2})$;

$w_k = \tilde{w_{k-1}} + \frac{t_{k-1}}{t_{k+1}}(\tilde{w_k} - \tilde{w_{k-1}})$, where $w_k = (x_k, r_k)$

where N stands for the number of angles. n is the length of the image vector.

Note that in practice an initial estimate r_0 is given. With this information, we can easily obtain x_0 by solving the linear least squares problem,

$$x_0 = \arg\min_x ||b - A(\tilde{r_0})x||_2^2 + \lambda^2||x||_2^2$$

As discussed in previous sections, the Tikhonov regularized least squares problems for x_0 and $\tilde{x}_k$ are solved using the hybrid scheme **IRhybrid_lsqr**. The nonlinear least squares problem for $\tilde{r_k}$ is solved using **imfil**. We have found that accelerating the solution vector x alone has yielded stabler results with slightly better accuracy than performing acceleration on both x and r. We show the numerical experiments in Chapter 4.

3.2 Anderson Acceleration

Anderson Acceleration, also called Anderson mixing, is a method used to accelerate the convergence of fixed point iteration. Note that we can write out alternating minimization scheme as a fixed point iteration

Algorithm 4: Fixed point iteration of image vector

1 **Initialization** Given $x_k \in \mathbb{R}^n$, output $x_{k+1} = g(x_k)$ from the following;

2 $r_{k+1} = \arg\min_r ||b - A(r)x_k||_2^2$;

3 $x_{k+1} = \arg\min_x ||b - A(r_{k+1})x||_2^2 + \lambda^2||x||_2^2$;

where $g : \mathbb{R}^n \to \mathbb{R}^n$ is the fixed point iteration of image vector x.

For this fixed point iteration, the general form of Anderson Acceleration is formed as the following:

Algorithm 5: Anderson Acceleration

1 Given x_0 and $m \geqslant 1$;

2 Set $x_1 = g(x_0)$, using Algorithm 4;

3 **for** *k=1,2,...* **do**

4 $m_k = min(m, k)$;

5 Set $F_k = (f_{k-m_k}, ..., f_k)$, where $f_i = g_i(x_i) - x_i$ and $g_i(x_i)$ comes from Algorithm 4;

6 Determine $\alpha^{(k)} = (\alpha_0^{(k)}, ..., \alpha_{m_k}^{(k)})^T$ that solves

7 $\min_\alpha ||F_k\alpha||_2$ s.t $\sum_{i=0}^{m_k} \alpha_i = 1$

8 Set $x_{k+1} = \sum_{i=0}^{m_k} \alpha_i^{(k)} g(x_{k-m_k+i})$, where $g(x_{k-m_k+i})$ is from Algorithm 4

9 **end**

We can cast the linear constrained optimization problem in Step 7 of the Algorithm 5 into an unconstrained form which is straightforward to solve and convenient for efficient implementation [26].

We define $\nabla f_i = f_{i+1} - f_i$ for each i and set $\nabla F = (\nabla f_{k-m_k}, ..., \nabla f_{k-1})$. Then the least squares problem is equivalent to

$$\min_{\gamma=(\gamma_0,...\gamma_{m_k-1})^T} ||f_k - F_k\gamma||_2$$

where $\alpha_0 = \gamma_0$ and $\alpha_i = \gamma_i - \gamma_{i-1}$, for $1 \leqslant i \leqslant m_k - 1$ and $\alpha_{m_k} = 1 - \gamma_{m_k - 1}$

This unconstrained least squares problem leads to a modified version of Anderson Acceleration:

Algorithm 6: Modified Anderson Acceleration

Given x_0 and $m \geqslant 1$;

Set $x_1 = g(x_0)$, using Algorithm 4;

for *k=1,2,...* **do**

  $m_k = min(m, k)$;

  Determine $\gamma^{(k)} = (\gamma_0^{(k)}, ..., \gamma_{m_k-1}^{(k)})^T$ that solves

  $\underset{\gamma=(\gamma_0,...\gamma_{m_k-1})^T}{min} ||f_k - F_k\gamma||_2$

  Set $x_{k+1} = g(x_k) - G_k\gamma^{(k)}$, where $g(x_k)$ comes from Algorithm 4.

end

where

$$x_{k+1} = g(x_k) - \sum_{i=1}^{m_k-1} \gamma_i^{(k)}[g(x_{k-m_k+i+1}) - g(x_{k-m_k+i})] = g(x_k) - G_k\gamma^{(k)}$$

with $G = (\nabla g_{k-m_k}, ..., \nabla g_{k-1})$, $\nabla g_i = g(x_{i+1}) - g(x_i)$

Homer Walker proposed implementation that efficiently updates the QR factors in the decomposition $F_k = Q_k R_k$. The basic logic is the following: every F_k is obtained from F_{k-1} with a column added on the right. If the resulting matrix has more columns than m, then delete one from the left. The column addition can be achieved by a modified Gram–Schmidt process. The deletion process is a little more complicated. We delete the first column on the left when $m_{k-1} = m$. If $F_{k-1} = QR$, then $F_{k-1}(:, 2:m) = QR(:, 2:m)$, where $R(:, 2:m)$ is upper-Hessenberg. Then, we can determine m Givens rotations to cancel out the entries in the sub-diagonal.

Chapter 4

Numerical Experiments

In this chapter, we make a few comparisons of different methods to solve the X-ray tomography problem illustrated in Equation (1.8). Firstly, we compare the speed of BCD exploiting the separability of geometry and the speed of BCD without using such property. Secondly, we compare results produced from different number of angles. Thirdly, we make comparisons of all the acceleration schemes. Fourthly, different regularization parameters in the linear least squares solvers are compared. Lastly, we compare the effects of different budgets on the results.

As mentioned in the first chapter, we use fan-beam projection for all our tomography problems for the sake of consistency. Note that we can easily adapt our code to solve for parallel beam projection by changing the **PRoptions** parameter for **PRtomo** problem in the **IRtools**.

In practice, a good initial guess of r is available and prior knowledge can help us set the bounds of r. We generate a test problem where true r values, r_{true}, are random numbers (chosen from a uniform distribution) between 1.5 and 2.5. We use a constant initial guess of $r = 2$ for all angles and set the bounds for the geometry parameters from 1.5 to 2.5. For consistency, we denote the forward operator as A, image size as n, number of angles as N_A, and noise level as $||\eta||_2$ for the rest of the chapter, where η is

a vector with random entries chosen from a normal distribution and $||\eta||_2 = \frac{||noise(:)||_2}{||b(:)||_2}$. Budget is a hyper-parameter in **imfil** that stands for the maximum number of function evaluations in the nonlinear least squares solver. Moreover, 0^{th} iteration is included in the relative errors figures below. This represents the relative error of the initial guesses with regard to the true solution.

4.1 BCD Exploiting Separability vs BCD

Let BCD that exploits separability be called BCDS. In this section, we compare the running time of BCD and that of BCDS. As we see in Figure 4.1 and Figure 4.2, BCDS dramatically speeds up the convergence because the separability allows us to solve a much smaller problem independently for one $r(i)$ at a time. The running time of image restoration using BCD is $16.985s$, about 5 seconds longer than BCDS's $11.888s$. The time of geometry reconstruction using BCD is $2226.581s$, much longer than BCDS's $115.922s$. This is also the reason BCDS is discussed first in this numerical experiment chapter. In our following experiments, we always use the BCDS to reduce the running time. We also notice semi-convergence behavior in the BCD algorithm, whereas BCDS do not run into such issues in our numerical experiments. Moreover, the geometry errors and reconstruction errors of BCDS are both better than that of BCD. The geometry errors are defined as the relative errors of geometry parameters, $\frac{||r-r_{true}||_2}{||r_{true}||_2}$. The reconstruction errors are defined as the relative errors of image, $\frac{||x-x_{true}||_2}{||x_{true}||_2}$. In Figure 4.3, we compare the 32×32 Shepp-Logan phantom, a standard test image of a human head [23]. The phantom reconstructed by BCDS is much closer to the true image than the one from BCD.

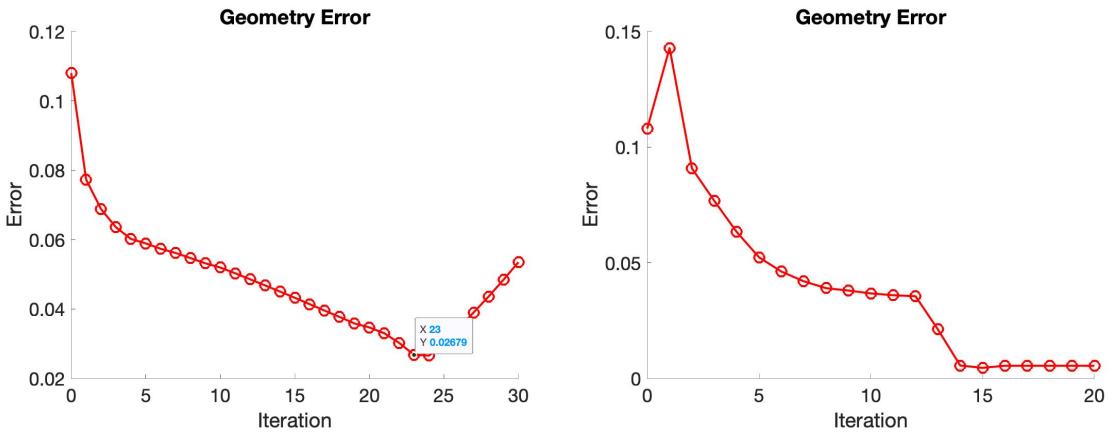

Figure 4.1: Comparison of relative errors of geometry parameters r: BCD (left) vs BCDS (right).

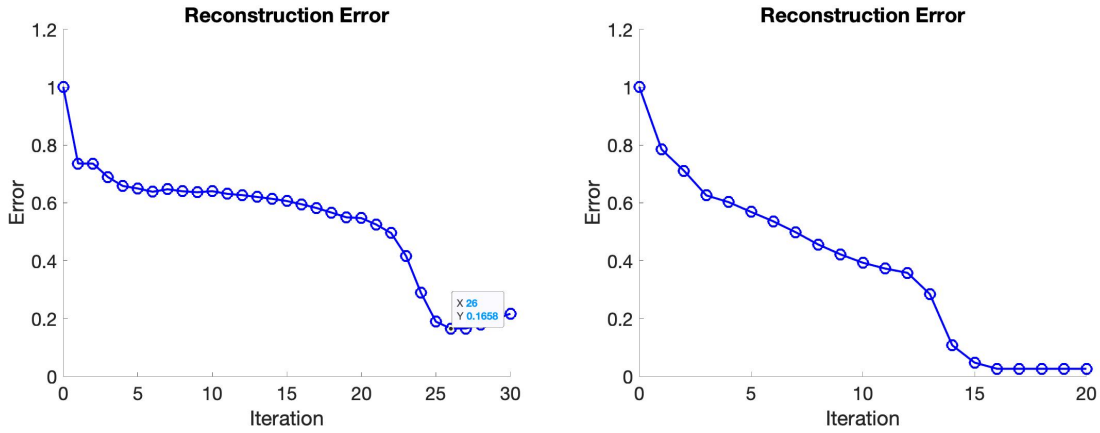

Figure 4.2: Comparison of relative errors of image vector x: BCD (left) vs BCDS (right).

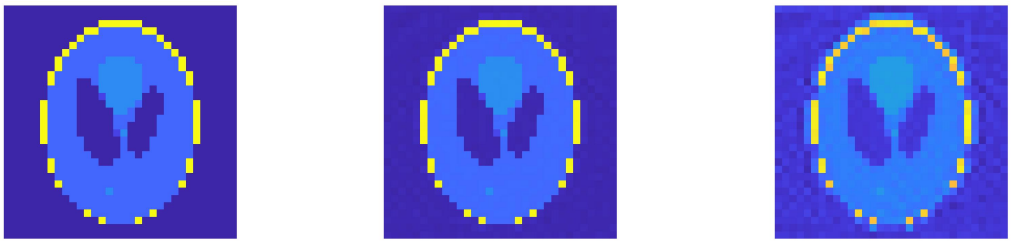

Figure 4.3: Comparison of 32×32 Shepp-Logan phantom: true image (left), BCDS image (middle), BCD image (right).

In this numerical experiment, $n = 32 \times 32$, $budget = 1000$, $||\eta||_2 = 0.01$, and

$N_A = 10$. A typical CT image using fan beam x-ray sources collects data at angles of one degree increment, from 0 to 359 degrees. In a perfect machine, the geometry parameters are precisely known each time the source is rotated to a new angle. In our experimental scenario, we assume the geometry parameters are only known approximately. To experiment with various scenarios, we assume that errors are introduced into the geometry parameters once every $\frac{360}{N_A}$ degrees. That is, with $N_A = 10$, errors occur once every 36 degrees.

Partially, N_A depends on the precision of machine calibration. For a good machine, N_A may be small. Also, N_A may depend on how precise we measure the data. For example, two geometry parameters that are different in terms of double precision may be rounded to the same number in single precision. In this scenario, the difference could be small enough that we can treat the two geometry parameters as being equal. If the number of angles is larger than the true number of angles, we may end up solving a larger problem than we need. For example, if for every 36 degrees only one geometry error is introduced, that is $N_{A_{true}} = 10$. If we assume $N_A = 20$, the average of the two geometry errors per 36 degrees would approximate the one true geometry error introduced in that set of angles. If the number of angles is smaller than the true number of angles, we end up solving a low dimension approximation. We do not seek to solve for the image exactly but aim to compute good approximations that yield much better results than not considering geometry parameters at all. In practice, we can think of number of angles as a hyper-parameter that practitioners can set based on their expertise and knowledge of the machine calibration. In our problems, we assume to know the number of angles.

4.2 Number of Angles

We compare the case where the number of angles is 5, 10, and 20 respectively. We want to explore the differences in relative error of r, relative error of x, the convergence, and the image quality.

In this comparison test, $n = 32 \times 32$, $||\eta||_2 = 0.01$, $budget = 10$. The reason that we can use a such small budget is we are essentially solving one scalar of r at a time. The dimension of that problem is one and the recommended formula to set up a good guess for budget is $10 * N^2$, where N is the length of the solution vector. Adjusting budget according to each particular problem as a hyper-parameter will further improve the quality of the result. Since we try to make apple-to-apple comparison, we keep budget the same for all three cases.

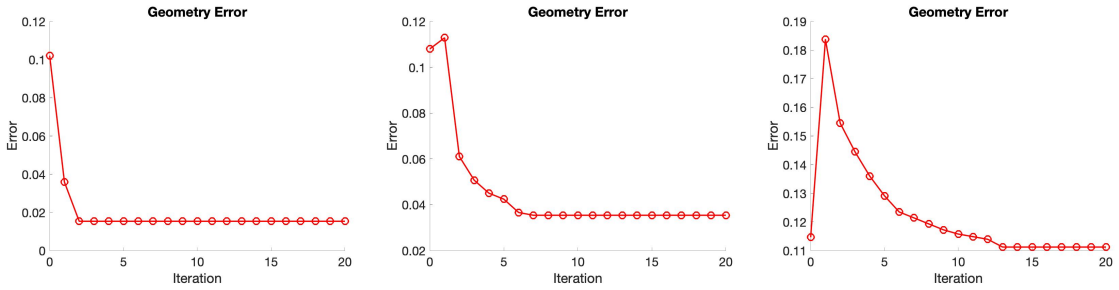

Figure 4.4: Comparison of relative errors of geometry parameter r: $N_A = 5$ (left), $N_A = 10$ (middle), $N_A = 20$ (right).

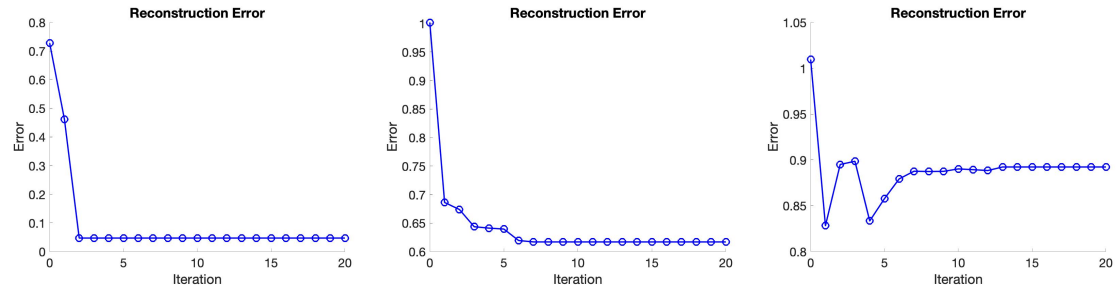

Figure 4.5: Comparison of relative errors of image vector x: $N_A = 5$ (left), $N_A = 10$ (middle), $N_A = 20$ (right).

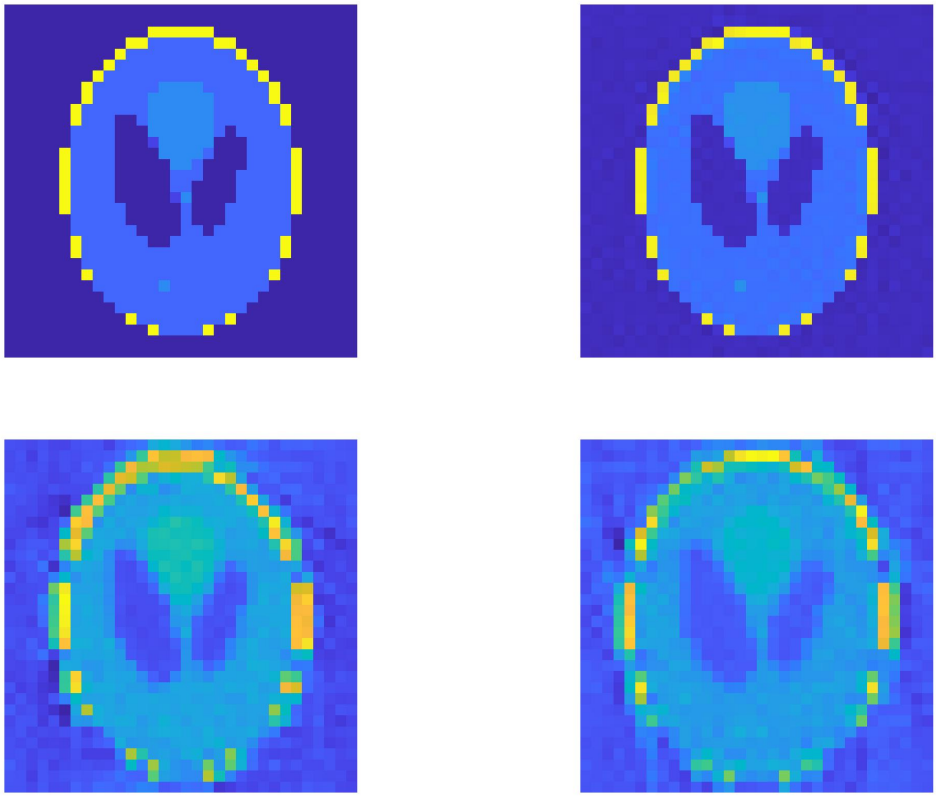

Figure 4.6: Comparison of 32×32 Shepp-Logan phantom: true image $(0,0)$, $N_A = 5$ $(0,1)$, $N_A = 10$ $(1,0)$, $N_A = 20$ $(1,1)$.

Note that both geometry and image parameters converge for $N_A = 5$ and $N_A = 10$. When $N_A = 20$, geometry error has a huge spike at the first iteration. Although our algorithm has gradually decreased the geometry error after the first iteration, the resulting geometry error is almost the same as the geometry error of the initial guess. The reconstruction error drops from more than 100% at the initial guess to below 85%, but the error climbed back up in the next few iterations. In the end, there was a 10% drop in relative error of the image comparing to the initial guess. There are two explanations for this unusual phenomenon happened when $N_A = 20$. Let $r^{(k)}$ and $x^{(k)}$ represent the value of r and x at k^{th} iteration in BCDS. First, $r^{(1)}$ is calculated from $x^{(0)}$ and an initial guess $r^{(0)}$. $r^{(1)}$ is particularly bad because how bad $x^{(0)}$ is. Despite this $x^{(0)}$, BCDS is still improving the geometry parameter after the first iteration.

Second, there seems to be an instability related to the size of N_A. This could be due to the fact that if N_A is large, the separable blocks have more columns than rows. For example, the size of A in a 32×32 test problem is 16200×1024. The number of rows in the separable block is $16200/20 = 810$, smaller than number of columns 1024. For this particular problem, the largest number of angles we can use without running into such instability issues is 12, which also have good convergent results. See Figure 4.7.

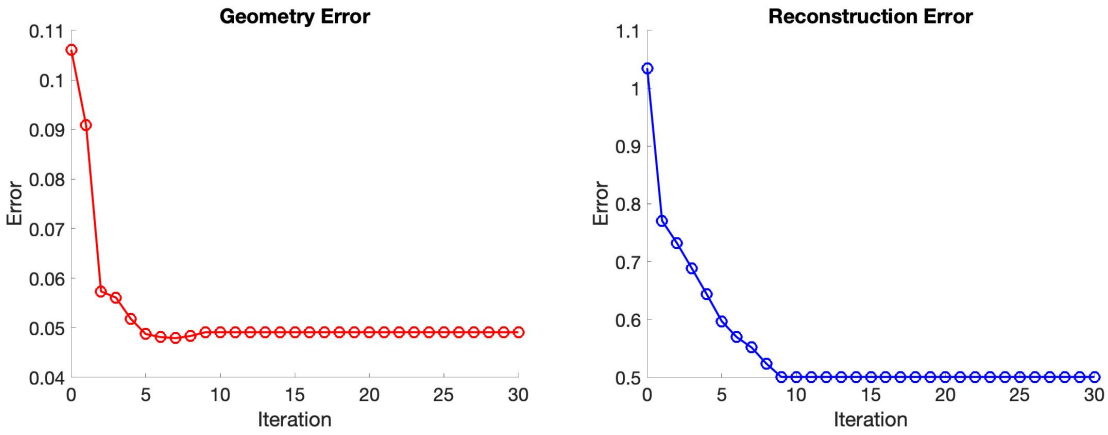

Figure 4.7: Geometry error and reconstruction error when $N_A = 12$.

When $N_A > 12$, we can use BCD to directly solve the problem.

4.3 Acceleration

In this section, we compare BCDS, imABCDS, and BCDS with Anderson acceleration, where imABCDS represents the inexact majorized accelerated block coordinate descent that exploits separability of geometry parameters. For all the comparisons made in this section, we use $n = 32 \times 32$, $N_A = 10$, $||\eta||_2 = 0.01$, and $budget = 100$. The memory size we use for all the Anderson accelerations is $m = 5$. We do not have to worry about rank deficiency because the row size 1024 is far greater than $m = 5$ and we always solve an over-determined problem.

Since we can implement Anderson acceleration in different ways depending on the usages of inner linear least squares solvers, we first compare different versions of Anderson acceleration before making comparisons with other methods. Moreover, there are also two implementations of imABCDS that we compare. After making those comparisons, we compare the best implementation of Anderson acceleration, the best implementation of imABCDS, and the BCDS method.

4.3.1 Anderson acceleration

In this section, we compare MATLAB **backslash** operator with **IRhybrid_lsqr** as two different inner linear least squares solvers for Anderson acceleration.

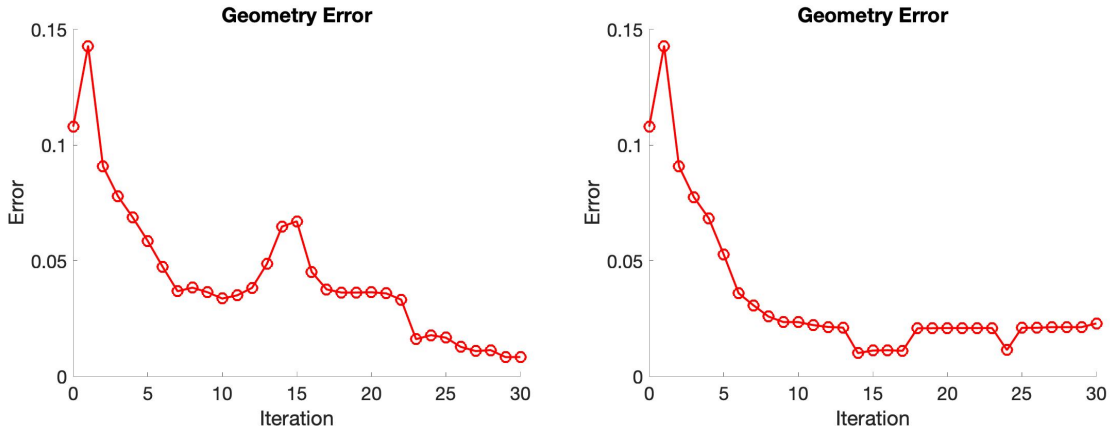

Figure 4.8: Geometry errors of BCDS with Anderson acceleration using **backslash** (left) and BCDS with Anderson acceleration using **IRhybrid_lsqr** (right).

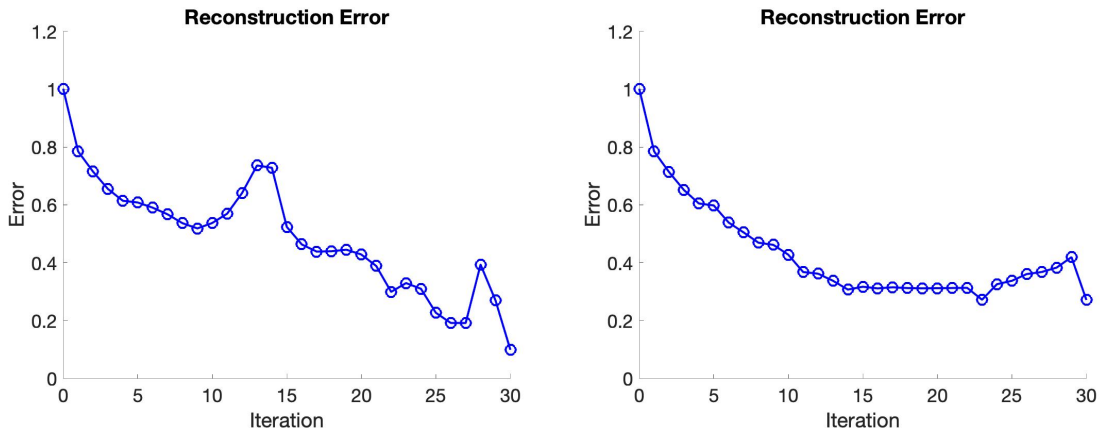

Figure 4.9: Reconstruction errors of BCDS with Anderson acceleration using **backslash** (left) and BCDS with Anderson acceleration using **IRhybrid_lsqr** (right).

We notice that the geometry errors of Anderson acceleration using **IRhybrid_lsqr** has a faster and stabler convergence than **backslash**. Due to its lack of regularization, the geometry errors of Anderson acceleration using **backslash** exhibits semi-convergence behavior in the first 15 iterations.

The reconstruction errors of Anderson acceleration using **IRhybrid_lsqr** also has a faster and stabler convergence than **backslash**. On the other hand, the reconstruction errors of Anderson acceleration using **backslash** gradually decreases over time with some spikes at iteration 14 and 28. Thus, **IRhybrid_lsqr** has a clear advantage in terms of both stability and speed of convergence over **backslash** when it is used to compute the linear least squares problems in Anderson acceleration, as described in Algorithm 6.

4.3.2 imABCDS

Since we can choose to apply the inexact majorized acceleration scheme on the image vector x only or apply it on both geometry parameter r and image vector x, we compare the two versions of imABCDS. We denote the former version of imABCDS as imABCDS-1. We denote the latter version that applies the acceleration on both

image and geometry parameters as imABCDS-b.

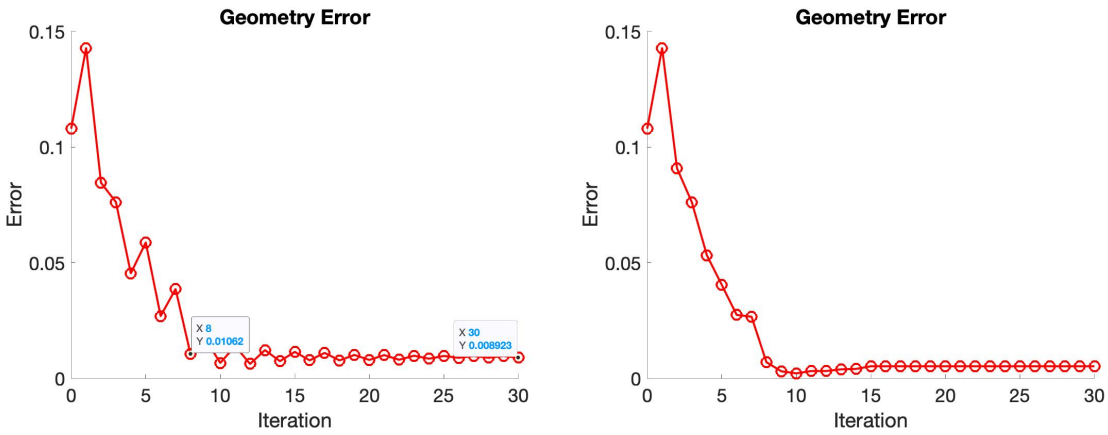

Figure 4.10: Geometry errors of imABCDS-b (left) and imABCDS-1 (right).

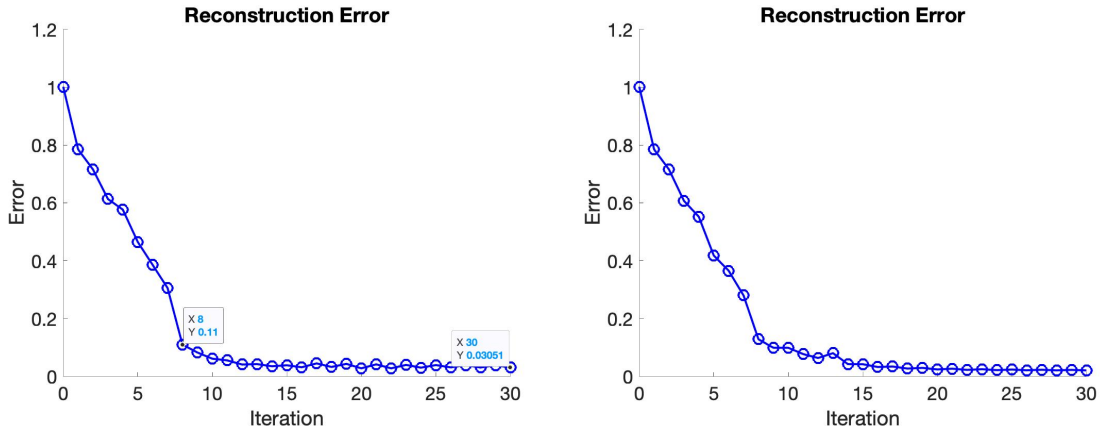

Figure 4.11: Image errors of imABCDS-b (left) and imABCDS-1 (right).

From the above figures, we can tell that although both approaches converge and produce similar results at the end of iterations, the geometry error of imABCDS-b oscillates more comparing to that of imABCDS-1. Thus, applying the acceleration scheme on image vector x alone is enough to produce stable and convergent results. For the sake of simplicity, when we mention imABCDS in the rest of the thesis, it always refers to imABCDS-1 .

4.3.3 imABCDS and BCDS

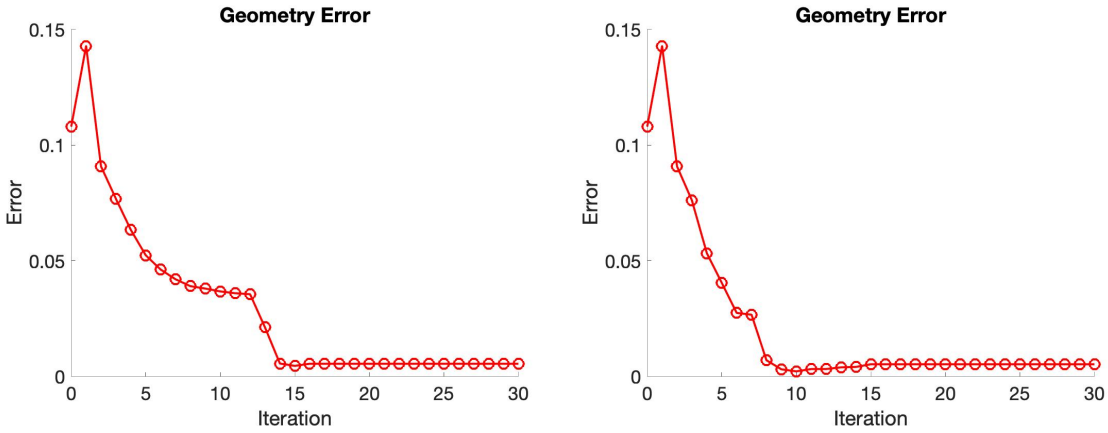

Figure 4.12: Geometry errors of BCDS (left) and imABCDS (right).

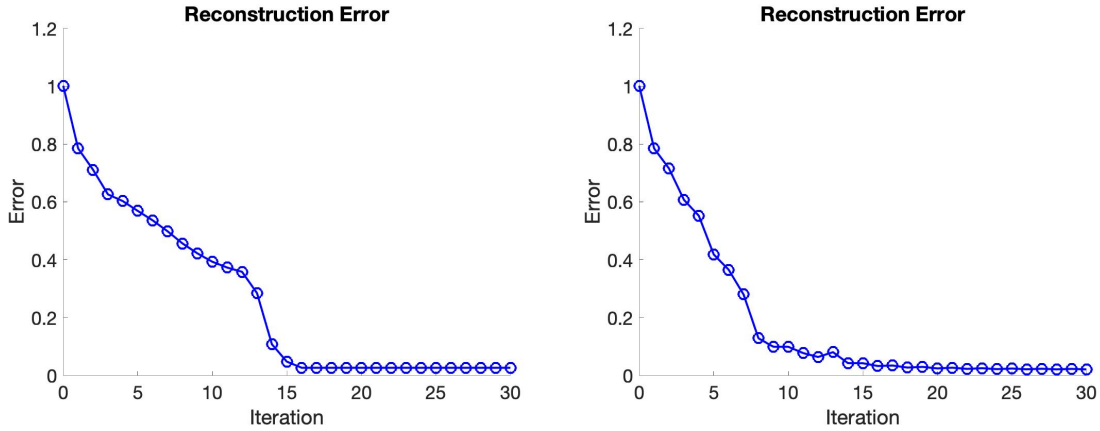

Figure 4.13: Reconstruction errors of BCDS (left) and imABCDS (right).

In BCDS, the geometry error converges at the 14^{th} iteration and the reconstruction error converges at 15^{th} iteration. The imABCDS and the Anderson Acceleration converge around the same iteration for both geometry and reconstruction errors. However, the levels of geometry and reconstruction errors of imABCDS are more superior to those of Anderson acceleration. One reason that Anderson acceleration does not work too well is our problem is not strictly a fixed point iteration of x. The nonlinear least squares solver depends on both x and a good initial guess of r. Another reason is Anderson Acceleration without safeguards cannot guarantee the

convergence of our original iterative scheme [29].

4.4 Regularization

In this section, we compare BCDS with different regularization parameters: no regularization, GCV, and adaptive weighted-GCV. The reason we use BCDS without the acceleration methods is that we want to see the direct impact of regularization on the alternating minimization scheme. We use $n = 32 \times 32$, $N_A = 10$, $||\eta||_2 = 0.01$, and $budget = 100$.

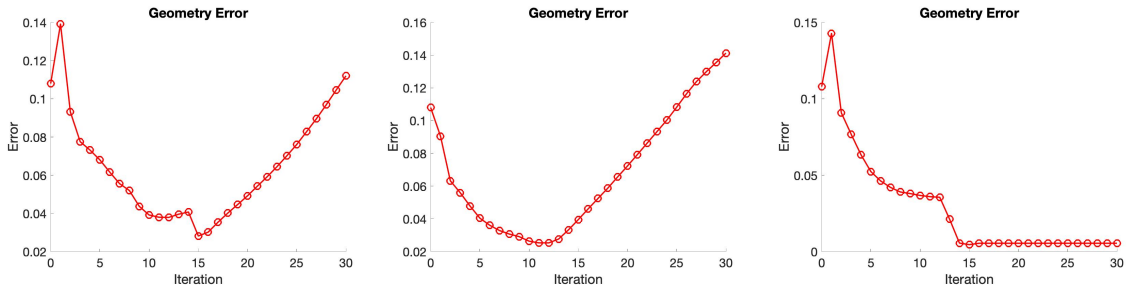

Figure 4.14: Geometry errors: No regularization (left), GCV (middle), W-GCV (right).

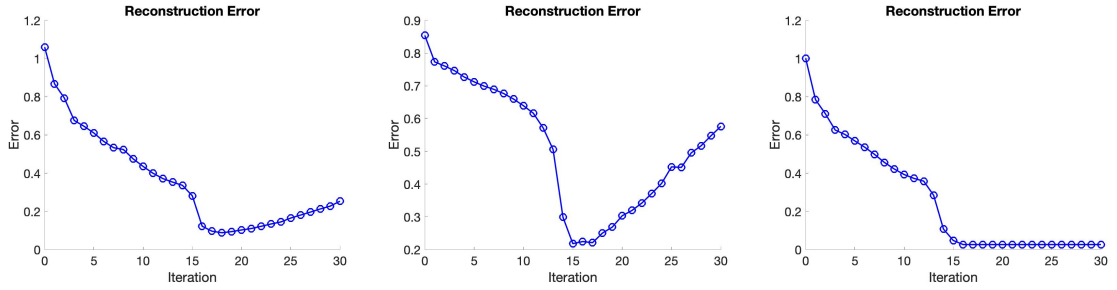

Figure 4.15: Reconstruction errors: No regularization (left), GCV (middle), W-GCV (right).

When there is no regularization, **IRhybrid_lsqr** is essentially an LSQR algorithm, which exhibits semi-convergence behavior. When GCV is used instead of the adaptive weighted-GCV, the geometry and image parameter errors also exhibit

semi-convergence behavior. The adaptive weighted-GCV helps stabilize LSQR's convergence and thus produces the best result.

4.5 Imfil Budget

We explore the effect of evaluation budgets in the nonlinear least squares solver on the geometry and reconstruction errors. We set $n = 32 \times 32$, $N_A = 10$, and $||\eta||_2 = 0.01$. We use $budget = 10, 100, 1000, 10000$. Since the budget size may greatly affect the nonlinear least squares solutions, we explore its effects on BCDS without any acceleration.

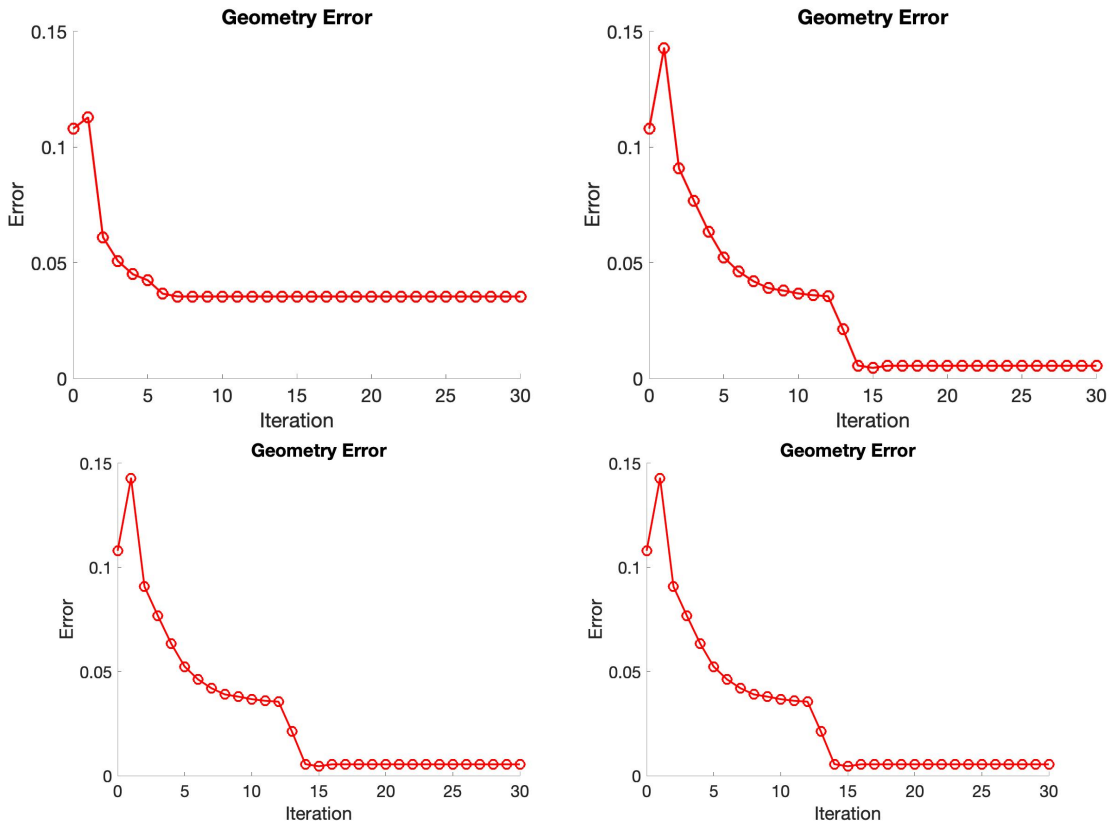

Figure 4.16: Geometry errors: $budget = 10$ $(0,0)$, $budget = 100$ $(0,1)$, $budget = 1000$ $(1,0)$, $budget = 10000$ $(1,1)$.

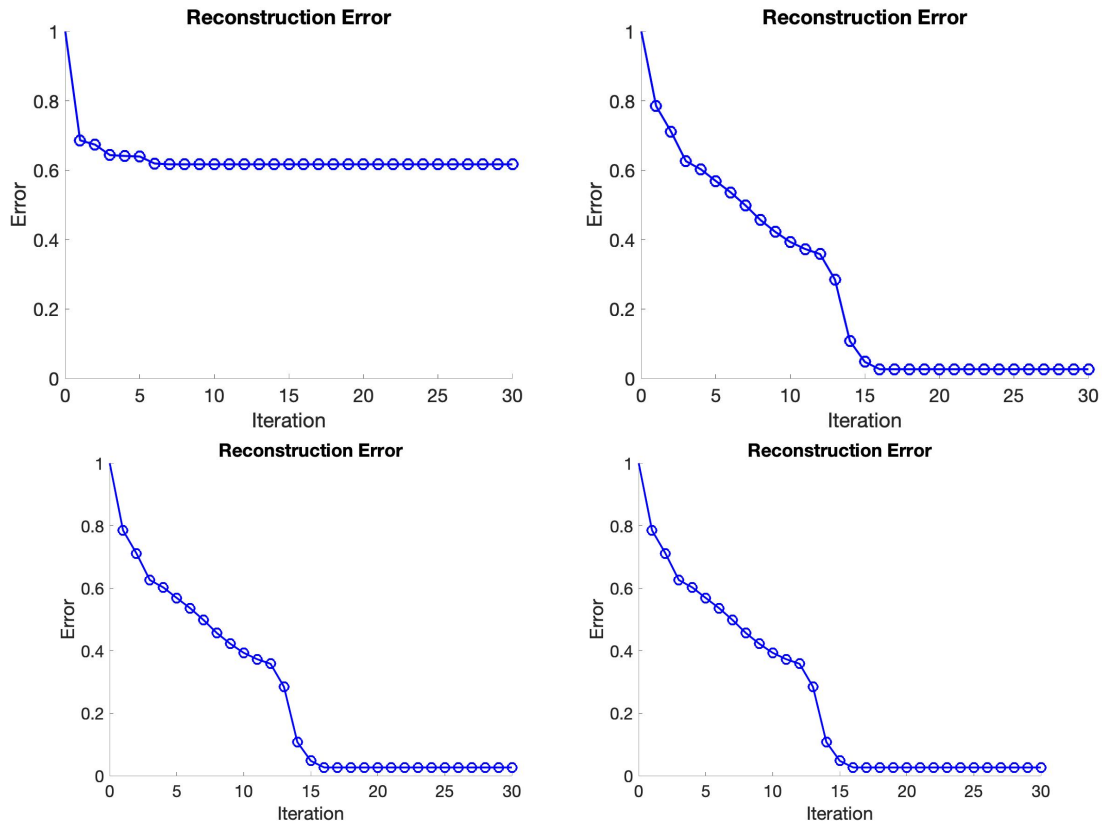

Figure 4.17: Reconstruction errors: $budget = 10$ $(0, 0)$, $budget = 100$ $(0, 1)$, $budget = 10000$ $(1, 0)$, $budget = 10000$ $(1, 1)$.

As we can see in the above figures, 100, 1000, and 10000 are almost equally good. The geometry errors and reconstruction errors are very small when 100, 1000, 10000 are used. Thus, 100 is the best budget out of the four because any more evaluation beyond 100 does not make the solution better. When budget is 1000 and 10000 respectively, we wasted many evaluations without making any progress. When 10 is used, even though we get convergent results earlier, the small budget terminate the algorithms before it finds a better minimization solution.

Chapter 5

Conclusion

In this thesis, we used an alternating minimization scheme to solve a tomographic image reconstruction problem. The linear least squares problem is solved by an adaptive weighted hybrid LSQR algorithm with Tikhonov regularization. The nonlinear least squares problem is solved by implicit filtering, a constrained optimization method that does not require derivative information. We also investigated imABCDS and Anderson mixing to accelerate the convergence. We have the following findings:

1. BCDS runs and converges much faster than the normal BCD because the separability of parameters allow us to solve each entry of r independently. However, there seems to be an instability when N_A is greater than a threshold calculated as $t = \frac{m}{n}$, where t is the largest integer such that n divides m, and m and n are the row and column size of the forward operator. In this case, we can directly apply BCD to solve the tomographic reconstruction problem.

2. Solving the linear least squares problem in Anderson acceleration with **IRhybrid_lsqr** produces a better result than using the **backslash** operator in terms of convergence and stability.

3. imABCDS is much less computationally costly than BCDS with Anderson acceleration because Anderson requires solving a linear least squares problem in

each iteration. imABCDS found better solutions for geometry and image than Anderson Acceleration in our tomographic reconstruction problem because the fixed point iteration of x also requires an update of r as input, and Anderson Acceleration without safeguards may encounter numerical instability.

4. The adaptive weighted hybrid LSQR algorithm with Tikhonov regularization stablizes the convergence more than applying an unweighted GCV to the LSQR algorithm. When no regularization method is used, the LSQR algorithm exhibits semi-convergence behavior. It reconstructs the solution at earlier iterations but noise at later iterations.

5. Choosing an appropriate budget for implicit filtering is important. When the budget is chosen too small, better solutions are not explored. When budget is chosen too big, many function evaluations are wasted without making progress. A suggested number from the **imfil** author is $10 * N^2$, where N is the length of r. We found this formula does not always give appropriate budget. Since we solve for each entry in the geometry parameter using separability, the dimension of each small problem is one. But, we have found 100 to be the best budget for 32×32 test problem with $N_A = 10$.

The imABCDs method has shown its success in our tomographic reconstruction problems. We believe this algorithm can be used to effectively solve X-ray tomography problems that have variations in the geometry parameter. A future direction towards more improvement would be adapting, applying, and advancing this algorithm on X-ray images produced in clinical trials.

Bibliography

[1] R. Bibb, D. Eggbeer, and A. Paterson. Chapter 2: Medical imaging. In *Medical Modelling*, pages 7–34. Woodhead Publishing, Oxford, 2 edition, 2015.

[2] Å. Björck. *Numerical Methods for Least Squares Problems*. Society for Industrial and Applied Mathematics, 1996.

[3] C. G. Broyden. A class of methods for solving nonlinear simultaneous equations. *Mathematics of Computation*, 19(92), 1965.

[4] D. Calvetti, G. H. Golub, and L. Reichel. Estimation of the L-curve via Lanczos bidiagonalization. *BIT Numerical Mathematics*, 39(4):603–619, 1999.

[5] J. Chung, J. G. Nagy, and D. P. O'Leary. A weighted GCV method for Lanczos hybrid regularization. *Electronic Transactions on Numerical Analysis*, 28, 2008.

[6] A. R. Conn, N. I. M. Gould, and P. L. Toint. Convergence of quasi-Newton matrices generated by the symmetric rank one update. *Mathematical Programming*, 50(1), 1991.

[7] Y. Cui, D. Sun, and K. Toh. Computing the best approximation over the intersection of a polyhedral set and the doubly nonnegative cone. *SIAM Journal on Optimization*, 29(4):2785–2813, 2019.

[8] D. Cummins, T. Filloon, and D. Nychka. Confidence intervals for nonparamet-

ric curve estimates: Toward more uniform pointwise coverage. *Journal of the American Statistical Association*, 96:233–246, 02 2001.

[9] D. Delgado. Iterative reconstruction methods of CT images using a statistical framework. *Oregon State University*, 2010.

[10] J. W. Demmel. *Applied Numerical Linear Algebra*. Society for Industrial and Applied Mathematics, 1997.

[11] A. Dogandzic, R. Gu, and K. Qiu. Mask iterative hard thresholding algorithms for sparse image reconstruction of objects with known contour. *Circuits, Systems and Computers, 1977. Conference Record. 1977 11th Asilomar Conference on*, 12 2011.

[12] R. Fletcher. *Practical Methods of Optimization*. John Wiley & Sons, New York, 2 edition, 1987.

[13] J. H. Friedman and B. W. Silverman. Flexible parsimonious smoothing and additive modeling. *Technometrics*, 31(1), 1989.

[14] A. Frommer and P. Maass. Fast CG-based methods for Tikhonov–Phillips regularization. *SIAM Journal on Scientific Computing*, 20(5):1831–1850, 1999.

[15] S. Gazzola, P. C. Hansen, and J. G. Nagy. IR tools: A MATLAB package of iterative regularization methods and large-scale test problems, 2018.

[16] P. C. Hansen. *Rank-Deficient and Discrete Ill-Posed Problems: Numerical Aspects of Linear Inversion*. Society for Industrial and Applied Mathematics, Philadelphia, 1998.

[17] P. C. Hansen and M. Saxild-Hansen. AIR tools — a MATLAB package of algebraic iterative reconstruction methods. *Journal of Computational and Applied*

Mathematics, 236(8):2167–2178, 2012. Inverse Problems: Computation and Applications.

[18] C. T. Kelley. Implicit filtering. *Society for Industrial and Applied Mathematics*, 2011.

[19] R. Mittelhammer, G. Judge, and D. Miller. *Econometric Foundations Pack with CD-ROM*. 01 2000.

[20] D. P. O'Leary and B. W. Rust. Variable projection for nonlinear least squares problems. *Computational Optimization and Applications*, 54(3):579–593, 2013.

[21] C. Paige and M. Saunders. LSQR: An algorithm for sparse linear equations and sparse least squares. *ACM Transactions on Mathematical Software*, 8(1).

[22] V. Pathuri-Bhuvana, S. Schuster, and A. Och. Joint calibration and tomography based on separable least squares approach with constraints on linear and nonlinear parameters. In *2020 28th European Signal Processing Conference (EUSIPCO)*, pages 1931–1935. IEEE, 2021.

[23] L. Shepp and B. F. Logan. The Fourier reconstruction of a head section. *IEEE Transactions on Nuclear Science.*, 1974.

[24] G. Strang. *Introduction to Linear Algebra*. Cambridge Press, Wellesley, 5 edition, 2016.

[25] R. Vio, P. Ma, W. Zhong, J. G. Nagy, L. Tenorio, and W. Wamsteker. Estimation of regularization parameters in multiple-image deblurring. *A&A*, 423(3):1179–1186, 2004.

[26] H. F. Walker. Anderson acceleration: Algorithms and implementations. *WPI Math. Sciences Dept. Report MS-6-15-50*, 2011.

[27] S. J. Wright. Coordinate descent algorithms. *Mathematical Programming*, 151(1):3–34, 2015.

[28] Z. Yu, J. Thibault, C. A. Bouman, K. D. Sauer, and J. Hsieh. Fast model-based X-ray CT reconstruction using spatially nonhomogeneous ICD optimization. *IEEE Transactions on Image Processing*, 20(1):161–175, 2011.

[29] J. Zhang, B. O'Donoghue, and S. Boyd. Globally convergent type-i anderson acceleration for nonsmooth fixed-point iterations. *SIAM Journal on Optimization*, 30(4):3170–3197, 2020.