X-ray CT Image Reconstruction on Highly-Parallel Architectures

by

Madison G. McGaffin

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Electrical Engineering: Systems)
in the University of Michigan
2015

Doctoral Committee:

Professor Jeffrey A. Fessler, Chair
Assistant Professor Laura K. Balzano
Associate Professor Marina A. Epelman
Assistant Professor Raj Rao Nadakuditi

Quidquid melius est quam repere

Acknowledgments

That this thesis exists at all is completely thanks to my advisor, Jeff Fessler. Jeff has been the best mentor, manager, sounding board, editor, and guide I could have hoped for. I am enormously grateful for both the latitude and structure has given me over the past five years. If there is anything useful in this thesis, it is almost certainly his doing.

The University of Michigan has been a wonderful place to be thanks to my fantastic professors and colleagues. I want to thank my committee for carving out time for me from their already insane schedules. Their feedback has made this a substantially stronger thesis. I am also grateful to my group mates – particularly Hung Nien and Donghwan Kim – for their invaluable perspectives, and for making a potentially lonely time enjoyable.

The last five years have been incredibly enjoyable thanks to my friends in Ann Arbor. To my housemates, Nick and Mitch; classmates, Matt, Rob, Paul, Pat, and Eric; and to Christie, Steve, Mike, and the many other people who brightened my time here: thank you. Your camaraderie is coffee for my soul.

Thank you, Darby, for your unwavering support and patience. I hope I can be even half as helpful when you write your thesis.

And finally: to my family. Thank you, Sam, Mom and Dad, for *everything*. This thesis belongs to you as much as to anyone.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF APPENDICES

ABSTRACT

X-ray CT Image Reconstruction on Highly-Parallel Architectures

by

Madison G. McGaffin

Chair: Jeffrey A. Fessler

Medical X-ray computed tomography (CT) produces 3D anatomical images by measuring the attenuation of X-rays transmitted through a patient at many locations and orientations and then processing the acquired 2D projections to form a 3D volume. Because the X-rays used are ionizing and potentially carcinogenic, doses ideally should be as low as possible. Unfortunately, conventional reconstruction algorithms, being based on mathematical idealizations of the CT measurement process, perform poorly at reduced doses.

Model-based image reconstruction (MBIR) methods use more accurate models of the CT acquisition process, the statistics of the noisy measurements, and noise-reducing regularization to produce potentially higher quality images even at reduced doses. They do this by minimizing a statistically motivated high-dimensional cost function; the high computational cost of numerically minimizing this function has prevented MBIR methods from reaching ubiquity in the clinic. Modern highly-parallel hardware like graphics

processing units (GPUs) may offer the computational resources to solve these reconstruction problems quickly, but simply "translating" existing algorithms designed for conventional processors to the GPU may not fully exploit the hardware's capabilities.

This thesis proposes GPU-specialized image denoising and image reconstruction algorithms. The proposed image denoising algorithm uses group coordinate descent with carefully structured groups. The algorithm converges very rapidly: in one experiment, it denoises a 65 megapixel image in about 1.5 seconds, while the popular Chambolle-Pock primal-dual algorithm running on the same hardware takes over a minute to reach the same level of accuracy.

For X-ray CT reconstruction, this thesis uses duality and group coordinate ascent to propose an alternative to the popular ordered subsets (OS) method. Similar to OS, the proposed method can use a subset of the data to update the image. Unlike OS, the proposed method is convergent. In one helical CT reconstruction experiment, an implementation of the proposed algorithm using one GPU converges more quickly than a state-of-the-art algorithm converges using four GPUs. Using four GPUs, the proposed algorithm reaches near convergence of a wide-cone axial reconstruction problem with over 220 million voxels in 11 minutes.

This thesis also proposes an algorithmic approach to circulant approximation of symmetric square matrices and a memory-efficient algorithm for computing structured majorizers to arbitrary matrices. These techniques provide algorithmic ways to design mathematical objects that conventionally have been designed by hand and may be useful for future algorithms.

CHAPTER 1

Introduction

All imaging systems collect measurements distorted by physical processes and corrupted by noise. With enough knowledge about the physical system and the statistical nature of the noise, these imperfections can be mitigated. Sometimes this process is routine: modern cellphone cameras routinely denoise the images they take. In other cases, recovering an image from a collection of noisy and distorted measurements is remarkable. Image inpainting [19], refocusing lightfield cameras [62, 68], and powerful denoising algorithms [82], sometimes seem more like magic than mathematics. X-ray computed tomography (CT) reconstruction is another example. By carefully modelling the physics of the CT scanner and the statistics of the noisy measurements, CT image reconstruction algorithms can produce high-quality 3D anatomical images from a huge number of noisy 2D views.

Image processing and reconstruction has become increasingly ambitious. Algorithm designers are tackling problems with more measurements and pixels and lower quality measurements; as these problems grow larger and more difficult, their computational burden increases significantly. Computer hardware has grown faster and more capable too, but a recent shift in hardware advances from serial execution speed to parallel computing has complicated the relationship. To keep up with the demand for faster algorithms, designers must create algorithms with their hardware in mind. Low computational complexity is good, but high parallelizability and regular memory access patterns are increasingly important too.

This thesis discusses designing image processing and reconstruction algorithms for the graphics processing unit (GPU). There is no program code in this thesis, although a considerable amount of code was written for some of our experiments. Instead, we focus on the mathematical design of algorithms that take advantage of the GPU's parallel processing capabilities. We have attempted to provide useful and practical results: except for

Chapter 7 most experiments measure the "wall-clock" runtime of each algorithm tested. We also compare our work with existing algorithms that are reasonably effective in CPU implementations but were not designed with the GPU in mind.

The next chapter includes background material. Beyond that, the rest of this thesis is organized as follows:

- Chapter 3 discusses circulant approximation. Many operators in image processing are approximately shift invariant, and approximating these operators with computationally efficient combinations of circulant and diagonal matrices is a common and effective technique. We present a novel algorithmic technique for approximating arbitrary square matrices with combinations of circulant and diagonal matrices. Even so, dense circulant convolution can be undesirably computationally expensive for some problems and on some hardware, like the GPU. To mitigate this problem, we propose an algorithm for approximating dense circulant operators with sparse convolutional filters. We demonstrate these techniques with problems from CT reconstruction and image denoising. This chapter is partially based on [51, 57].

- Chapter 4 presents an efficient image denoising algorithm for the GPU. The algorithm uses group coordinate descent with carefully designed groups and updates the image by solving many parallel one-dimensional problems. The image is updated in place, and the algorithm has very low memory overhead. The proposed algorithm converges very quickly in experiments denoising large 2D and 3D images. This chapter is based on [52, 54].

- Chapter 5 focuses on a challenging problem from CT reconstruction. Conventional approaches to solving quadratic minimization problems involving the CT system matrix (like preconditioning or the ordered subsets approximation) are often either nonconvergent or slow on 3D problems. We present a duality-based framework that yields impressive solution speeds and still converges to the "true" solution to the reconstruction problem. We combine the proposed approach with the denoising algorithm in Chapter 4 for a fast CT reconstruction algorithm. This chapter is partially based on [53].

- Chapter 6 presents a novel, fast, and convergent CT reconstruction algorithm using duality and group coordinate ascent. The algorithm is designed with the GPU in mind: it is composed of highly parallel steps with small memory requirements.

Data transfers to and from the GPU are interleaved with computation to improve throughput. We demonstrate state of the art convergence speeds on single and multiple GPUs. This chapter is partially based on [55, 56].

- Chapter 7 considers designing "minimal majorizing" matrices; that is, structured matrices that upper-bound a given matrix and have as small values as possible in a certain sense. Matrix majorizers are ubiquitous in image processing algorithms and are often designed by hand and without any guarantees of optimality. We present a novel memory-efficient approach to designing these matrices by optimizing a cost function and show results for a simple example. This chapter is based on unpublished work.

Finally, Chapter 8 contains some conclusions and directions for future work.

CHAPTER 2

Background

This chapter highlights some less well-known background material for the subjects discussed in this thesis.

2.1 Notation

We denote vectors as bold lowercase letters, e.g., $\mathbf{x} \in \mathbb{R}^N$. Matrices are bold uppercase letters, e.g., $\mathbf{A} \in \mathbb{R}^{M \times N}$.

Scalars are typeset with italics. Constants are written in uppercase, like N and M, while variables are written in lowercase, e.g., t. The entry of $\mathbf{A}$ in the ith row and jth column in written $[\mathbf{A}]_{ij}$ or a_{ij}. The jth column of $\mathbf{A}$ is written $\mathbf{a}_j$. Similarly, for vectors, x_j and $[\mathbf{x}]_j$ are both the jth element of $\mathbf{x}$. We prefer the former, and use the latter notation for more complicated expressions. Unless otherwise noted, the first index is 1.

If $\mathbf{D} \in \mathbb{R}^{N \times N}$ is a diagonal matrix, then $\mathbf{d} \in \mathbb{R}^N$ is its diagonal:

$$[\mathbf{D}]_{ij} = \begin{cases} d_i, & i = j \\ 0, & \text{else.} \end{cases} \tag{2.1}$$

If $\mathbf{C} \in \mathbb{R}^{N \times N}$ is a circulant matrix, then $\mathbf{c} \in \mathbb{R}^N$ is its convolution kernel:

$$[\mathbf{C}]_{ij} = c_{1+((i-j) \bmod N)}. \tag{2.2}$$

Functions from a multi-dimensional space to scalars are written in a sans serif font, e.g., $\mathsf{R} : \mathbb{R}^N \to \mathbb{R}$. Most of the time, these will be convex functions. One dimensional functions are written with lowercase Arabic or Greek letters. For example, $\psi(t) = \frac{1}{2}t^2$. If $\mathsf{F}(\mathbf{x}, \mathbf{y})$ is differentiable, its column gradient with respect to $\mathbf{x} \in \mathbb{R}^N$ is written $\nabla_{\mathbf{x}}\mathsf{F}(\mathbf{x}, \mathbf{y}) \in$

$\mathbb{R}^N$. The matrix of second derivatives, i.e., the Hessian, of a function $\mathsf{G} : \mathbb{R}^N \to \mathbb{R}$ evaluated at $\mathbf{x}_0$ is written $\nabla^2_{\mathbf{x}_0} \mathsf{G} \in \mathbb{R}^{N \times N}$.

This thesis discusses many iterative algorithms. Superscripted numbers in parentheses normally mean iteration number, e.g., $\mathbf{x}^{(n)}$ is the version of $\mathbf{x}$ from the nth iteration of some procedure.

Sometimes we talk about updating a variable "in place," a concept borrowed from computer programming languages. We do this when keeping track of iteration superscripts would become cumbersome, and only the "most recent" value of a variable is important. If we replace the value of the variable $\mathbf{u}$ with the new value $\mathbf{u}^+$, then we write $\mathbf{u} \leftarrow \mathbf{u}^+$.

Sets (that are almost always convex in this thesis) are normally indicated with uppercase Greek letters, e.g., Ω.

The minimizer of a function F over a set Ω is written

$$\widehat{\mathbf{x}} = \operatorname*{argmin}_{\mathbf{x} \in \Omega} \mathsf{F}(\mathbf{x}), \tag{2.3}$$

if it is unique. A sufficient condition for this is for F to be strongly convex and Ω to be convex. As a rule of thumb, the output of an optimization procedure is indicated with a hat, e.g., $\widehat{\mathbf{x}}$ or $\widehat{\mathbf{u}}^{(n+1)}$. Possibly approximate optimizers are written with a tilde, $\widetilde{\mathbf{x}}$, or are unadorned.

2.2 Convex optimization

Convex optimization is far too broad to even approximately summarize in this chapter. This thesis was written with the books by Boyd et al. [8], Bertsekas et al. [4], and Borwein et al. [5] near at hand. This section highlights a couple important subjects for this thesis.

2.2.1 Optimization transfer / the majorize-minimize technique

Let $\mathsf{F} : \mathbb{R}^N \to \mathbb{R}$ be a convex and differentiable cost function that we want to minimize over the convex set Ω:

$$\widehat{\mathbf{x}} = \operatorname*{argmin}_{\mathbf{x} \in \Omega} \mathsf{F}(\mathbf{x}). \tag{2.4}$$

Often it is impractical to minimize F "in one step" because $\mathbf{x}$ is high dimensional, F contains nonquadratic terms, or because the domain Ω is inconvenient. Instead, we iteratively refine an estimate of $\widehat{\mathbf{x}}$. In particular, we will generate a sequence of iterates $\left\{\mathbf{x}^{(n)}\right\}$ that steadily decrease the cost function F.

One way to do this is with optimization transfer, also called the majorize minimize method. We generate a series of functions $\mathsf{G}^{(n)}$ that satisfy the following two conditions:

$$\mathsf{G}^{(n)}\left(\mathbf{x}^{(n)}\right) = \mathsf{F}\left(\mathbf{x}^{(n)}\right), \tag{2.5}$$

$$\mathsf{G}^{(n)}(\mathbf{x}) \geq \mathsf{F}(\mathbf{x}) \qquad \forall \mathbf{x} \in \Omega. \tag{2.6}$$

We use these surrogate functions to refine our estimate of $\widehat{\mathbf{x}}$:

$$\mathbf{x}^{(n+1)} = \underset{\mathbf{x}\in\Omega}{\operatorname{argmin}} \ \mathsf{G}^{(n)}(\mathbf{x}). \tag{2.7}$$

Clearly, this produces a nonincreasing sequence of cost function values. Under mild regularity conditions [35], $\left\{\mathbf{x}^{(n)}\right\}$ converges to a minimizer of F.

An intuitive way to build the surrogate function $\mathsf{G}^{(n)}$ is with the Taylor-like expansion:

$$\mathsf{G}^{(n)}(\mathbf{x}) = \mathsf{F}\left(\mathbf{x}^{(n)}\right) + \left(\mathbf{x} - \mathbf{x}^{(n)}\right)'\nabla\mathsf{F}\left(\mathbf{x}^{(n)}\right) + \frac{1}{2}\left|\left|\mathbf{x} - \mathbf{x}^{(n)}\right|\right|_{\mathbf{M}}^2, \tag{2.8}$$

where the matrix $\mathbf{M} \succeq \mathsf{F}$ majorizes the Hessian of F. We discuss majorization more in Chapter 4, 5, 6 and 7.

2.2.2 Convex conjugates

Let $f : \Omega \to \mathbb{R}$, $\Omega \subseteq \mathbb{R}$ be a function. The convex conjugate of f is

$$f^*(z) = \sup_x \ zx - f(x). \tag{2.9}$$

The convex conjugate is always a convex function. The biconjugate of f, f^{**}, is the convex conjugate of f^*. If f is convex and closed, then $f^{**} = f$. This allows us to write the often helpful representation

$$f(x) = \sup_z \ xz - f^*(z). \tag{2.10}$$

Combined with a minimax theorem like the ones below, convex conjugacy can lead to useful algorithms.

2.2.3 Minimax theorems

Chapters 4, 5, 6, and 7 consider problems of the following form:

$$\operatorname*{argmin}_{\mathbf{x}\in\Omega} \sup_{\mathbf{u}\in\Gamma} \mathsf{S}(\mathbf{x},\mathbf{u}). \tag{2.11}$$

The inner supremum normally arises from a convex conjugate, but Chapter 7 uses a more exotic sort of duality. Often transposing the order of minimization and maximization leads to a useful algorithm. There are standard tools we use to perform this transposition: Sion's minimax theorem and Fenchel duality.

2.2.3.1 Sion's minimax theorem

Let $\mathsf{S}(\mathbf{x},\mathbf{u}) : \mathbb{R}^N \times \mathbb{R}^M \to \mathbb{R}$, $\Omega \subseteq \mathbb{R}^N$ and $\Gamma \subseteq \mathbb{R}^M$ be given. Sion's minimax theorem states [80] that

$$\inf_{\mathbf{x}\in\Omega} \sup_{\mathbf{u}\in\Gamma} \mathsf{S}(\mathbf{x},\mathbf{u}) = \sup_{\mathbf{u}\in\Gamma} \inf_{\mathbf{x}\in\Omega} \mathsf{S}(\mathbf{x},\mathbf{u}) \tag{2.12}$$

provided the following conditions hold:

- both Ω and Γ are convex,
- at least one of Ω or Γ is compact,
- S is quasiconcave and upper semicontinuous in $\mathbf{u}$ for any fixed $\mathbf{x} \in \Omega$, and
- S is quasiconvex and lower semicontinuous in $\mathbf{x}$ for any fixed $\mathbf{u} \in \Gamma$.

Sion's minimax theorem places relatively loose requirements on S, but the compactness requirement for one of the domains can be restrictive. Fenchel duality, which uses convex conjugates, can be more useful.

2.2.3.2 Fenchel duality

Let $\mathsf{F} : \mathbb{R}^N \to \{\infty\} \cup \mathbb{R}$ and $\mathsf{G} : \mathbb{R}^M \to \{\infty\} \cup \mathbb{R}$ be extended convex functions, and let $\mathbf{A} \in \mathbb{R}^{M\times N}$. Let Ω be the set over which F is finite and let Γ be the set over which G is

continuous. The image of Ω under $\mathbf{A}$ is

$$\mathbf{A}\Omega = \{\mathbf{u} : \exists \mathbf{x} \in \Omega \text{ such that } \mathbf{A}\mathbf{x} = \mathbf{u}\}. \tag{2.13}$$

Fenchel's duality theorem [5, Theorem 4.4.3] states that if $\mathbf{A}\Omega \cap \Gamma$ is not empty,

$$\inf_{\mathbf{x}} \mathsf{F}(\mathbf{x}) + \mathsf{G}(\mathbf{A}\mathbf{x}) = \sup_{\mathbf{u}} \; -\mathsf{F}^*(\mathbf{A}'\mathbf{u}) - \mathsf{G}^*(-\mathbf{u}) \tag{2.14}$$

$$= \sup_{\mathbf{u}} \; -\mathsf{F}^*(-\mathbf{A}'\mathbf{u}) - \mathsf{G}^*(\mathbf{u}). \tag{2.15}$$

The minimax structure in Fenchel's duality theorem is less obvious than in Sion's minimax theorem. If we rewrite G and F^* using biconjugacy, the relationship becomes more clear:

$$\inf_{\mathbf{x}} \mathsf{F}(\mathbf{x}) + \mathsf{G}(\mathbf{A}\mathbf{x}) = \inf_{\mathbf{x}} \sup_{\mathbf{u}} \; \mathsf{F}(\mathbf{x}) + \mathbf{u}'\mathbf{A}\mathbf{x} - \mathsf{G}^*(\mathbf{u}) \tag{2.16}$$

$$= \sup_{\mathbf{u}} \; -\mathsf{F}^*(-\mathbf{A}'\mathbf{u}) - \mathsf{G}^*(\mathbf{u}) \tag{2.17}$$

$$= \sup_{\mathbf{u}} \; -\left[\sup_{\mathbf{x}} \; \mathbf{x}'\mathbf{A}'\mathbf{u} - \mathsf{F}(-\mathbf{x})\right] - \mathsf{G}^*(\mathbf{u}) \tag{2.18}$$

$$= \sup_{\mathbf{u}} \; -\left[\sup_{\mathbf{x}} \; -\mathbf{x}'\mathbf{A}'\mathbf{u} - \mathsf{F}(\mathbf{x})\right] - \mathsf{G}^*(\mathbf{u}) \tag{2.19}$$

$$= \sup_{\mathbf{u}} \; \inf_{\mathbf{x}} \; \mathsf{F}(\mathbf{x}) + \mathbf{x}'\mathbf{A}\mathbf{u} - \mathsf{G}^*(\mathbf{u}). \tag{2.20}$$

We exploit Fenchel's duality theorem in Chapters 5 and 6.

2.3 Image denoising

Let $\mathbf{y} \in \mathbb{R}^M$ be noisy pixel measurements and $\mathbf{W}$ be a positive diagonal matrix. In chapters 3 and 4 we study the following edge-preserving image denoising problem:

$$\widehat{\mathbf{x}} = \underset{\mathbf{x}\in\Omega}{\operatorname{argmin}} \; \frac{1}{2}||\mathbf{x} - \mathbf{y}||^2_{\mathbf{W}} + \mathsf{R}(\mathbf{C}\mathbf{x}). \tag{2.21}$$

The matrix $\mathbf{C} \in \mathbb{R}^{K\times N}$ is a finite differences matrix. It computes the differences between every pixel in the image $\mathbf{x}$ and its neighbors. Conceptually, $\mathbf{C}$ is a stack of $N \times N$ banded matrices. Each banded matrix has 1s along the diagonal and a band of -1s corresponding

to a particular offset.

The function $\mathsf{R} : \mathbb{R}^K \to \mathbb{R}$ is a sum of convex functions:

$$\mathsf{R}(\mathbf{v}) = \sum_{k=1}^{K} \beta_k \psi(v_k). \tag{2.22}$$

The one-dimensional function ψ is called the potential function. It is even, convex, and often coercive. The nonnegative weights $\{\beta_k\}$ can be used to adjust the local strength of the regularizer. We can absorb C into (2.22) for the following more intuitive form:

$$\mathsf{R}(\mathbf{x}) = \sum_{j=1}^{N} \sum_{k \in \mathcal{N}_j} \beta_{jk} \psi(x_j - x_k). \tag{2.23}$$

The set $\mathcal{N}_j$ contains the spatial neighbors of the jth pixel. Conventionally the neighborhoods do not contain their centers: $j \notin \mathcal{N}_j$. The potential function penalizes differences between neighboring pixels. A heuristic description of noisy images is that they have a lot of "speckles" in them; i.e., they appear very rough. The regularizer R takes larger values for images with more differences between neighboring pixels, i.e., noisier images. By finding the minimizer of the cost function (2.21), we remove some of that noise.

The potential function has a significant qualitative effect on the edges in the denoised image $\widehat{\mathbf{x}}$. Potential functions $\psi(t)$ that are relatively large as $|t| \to \infty$, e.g., the quadratic potential $\frac{1}{2}t^2$ tend to over-penalize edges in the image, which appear to the regularizer as large differences between adjacent pixels. On the other hand, potential functions that are large around the origin, e.g., $\psi(t) = |t|$, tend to heavily-penalize small differences. This can lead to "cartoony" looking images with large uniform regions. Many potential functions attempt to split the difference, with quadratic-like behavior around the origin and absolute value-like behavior away from it.

Finally, the convex set Ω codifies the acceptable range of pixel values, e.g., $0 - 255$ gray levels.

Chapters 3 and 4 present algorithms for solving this denoising problem.

2.4 Model-based X-ray CT reconstruction

X-ray computed tomography (CT) is a widely-used medical, industrial and security imaging tool. During a CT scan, an X-ray source illuminates a region of space with radiation.

X-rays interact with the matter in that space, and some of these X-rays are detected by a receiver. By comparing the known intensity of the emitted X-rays on one side of the irradiated region to the received intensity on the other side, we can infer clinically useful information about the X-ray-interacting matter between the source and the detector. And by taking many measurements at different positions and orientations, we can reconstruct useful three-dimensional "images" of the X-ray absorbing matter within a region of interest.

2.4.1 Families of algorithms

There are two broad families of X-ray reconstruction algorithms: analytical methods and iterative methods. Most analytical methods are based on mathematical idealizations of the CT acquisition process [21, 29, 37]. Many do not incorporate information about noise statistics and correspondingly perform poorly at low signal-to-noise ratios. The upside to these algorithms is that they are very fast and able to perform a CT reconstruction in minutes or less. In this thesis, fast analytical methods are used to initialize slower iterative algorithms.

Model-based image reconstruction (MBIR) algorithms incorporate more information about CT physics and noise statistics [83]. Similar to the denoising problem above, they can use regularization to reduce noise. This additional information allows MBIR methods to produce higher quality images than analytical methods, and provide useful images and much lower doses, but generating an image with MBIR methods is often expensive and time consuming. This long runtime has conventionally meant that MBIR methods have achieved only limited adoption in the clinic.

One of the goals of this thesis, and other recent theses [45, 62], is to find faster ways to perform MBIR.

2.4.2 Optimization problem

We study the following model-based image reconstruction (MBIR) problem [83]:

$$\widehat{\mathbf{x}} = \underset{\mathbf{x} \geq \mathbf{0}}{\operatorname{argmin}} \ \frac{1}{2}||\mathbf{A}\mathbf{x} - \mathbf{y}||_{\mathbf{W}}^2 + \mathsf{R}(\mathbf{C}\mathbf{x}). \tag{2.24}$$

The edge-preserving regularizer R is the same as in image denoising; see Section 2.3. The CT system matrix, $\mathbf{A} \in \mathbb{R}^{M \times N}$, models the geometry of the CT system. Multiplication by

$\mathbf{A}$ is called projection and simulates the acquisition of an image by the CT scanner. The adjoint operation, multiplication by $\mathbf{A}'$, is called backprojection. The noisy measurements are contained in $\mathbf{y} \in \mathbb{R}^M$, and the positive matrix $\mathbf{W}$ conventionally contains the variances of those measurements, $\mathbf{W} = \underset{i}{\text{diag}}\left\{\frac{1}{\sigma_i^2}\right\}$.

Solving (2.24) numerically is challenging. The number of pixels in the image is very large, and $\mathbf{A}$ contains too many entries to store or factor. Computing a gradient of the cost function for an iterative algorithm is expensive because of the large number of measurements. The Hessian of the data-fit term, $\mathbf{A}'\mathbf{W}\mathbf{A}$ has high dynamic range because of the statistical weights and is difficult to precondition, as we explore in Chapter 3.

Many algorithms have been proposed to solve (2.24) quickly [1, 23, 44, 67, 74]. (We suggest a few more in Chapters 5 and 6.) The next section overviews one particularly influential algorithm.

2.4.3 Ordered subsets

The most expensive operation in a gradient-based method to solve the MBIR CT reconstruction problem (2.24) is computing the gradient of the quadratic "tomography" term:

$$\mathbf{g}^{(n)} = \nabla \frac{1}{2} \left|\left| \mathbf{A}\mathbf{x}^{(n)} - \mathbf{y} \right|\right|_{\mathbf{W}}^2 \tag{2.25}$$

$$= \mathbf{A}'\mathbf{W}\left(\mathbf{A}\mathbf{x}^{(n)} - \mathbf{y}\right) \tag{2.26}$$

$$= \sum_{v=1}^{N_{\text{views}}} \mathbf{A}_v{}'\mathbf{W}_v\left(\mathbf{A}_v\mathbf{x}^{(n)} - \mathbf{y}_v\right), \tag{2.27}$$

where $\mathbf{A}_v$ is the one-view CT projection matrix for view v. The term involves a forward projection (multiplication by $\mathbf{A}$) and a backprojection (multiplication by $\mathbf{A}'$) of all the views in the acquisition geometry. Forward and backprojecting a single view is not extremely expensive by itself, but because there are often thousands of views in a dataset, the total computation often overwhelms all other parts of the reconstruction algorithm.

Although there are many views in a CT acquisition, the views are often heavily correlated. The experiments in this thesis use data in which sequential views come from detector rotations less than 0.37 degrees apart! Assuming the data is somewhat consistent, for $v \approx k$,

$$\mathbf{A}_v{}'\mathbf{W}_v\left(\mathbf{A}_v\mathbf{x}^{(n)} - \mathbf{y}_v\right) \approx \mathbf{A}_k{}'\mathbf{W}_k\left(\mathbf{A}_k\mathbf{x}^{(n)} - \mathbf{y}_k\right). \tag{2.28}$$

This leads to the following ordered subsets approximation [1]:

$$\mathbf{g}^{(n)} \approx \frac{N_{\text{view}}}{|S_m|} \sum_{v \in S_m} \mathbf{A}_v{}' \mathbf{W}_v \big(\mathbf{A}_v \mathbf{x}^{(n)} - \mathbf{y}_v \big). \tag{2.29}$$

Ideally, the groups of views $\{S_m\}$ are chosen so the approximation (2.29) is accurate. In this thesis, each the $\{S_m\}$ partition the views into interleaved subsets that are evenly spaced apart by a constant offset.

The ordered subsets approximation is used in iterative algorithms, and more than one gradient approximation $\mathbf{g}^{(n)}$ will be computed before the algorithm is finished. The *ordered* part of "ordered subsets" means that the subsets $\{S_m\}$ are chosen in each iteration to mitigate compounding errors from the subsets approximation. Generally speaking, this means selecting subsets that are as uncorrelated from one another as possible. In this thesis, we select subsets using the "bit reversal" or "FFT" ordering; see [44] for more details.

The ordered subsets with separable quadratic surrogates (OS-SQS) [1] algorithm uses the ordered subsets approximation with a simple diagonal majorizer. Let $\mathbf{D}$ be a diagonal matrix that majorizes the Hessian of the MBIR reconstruction cost function (2.24):

$$\mathbf{D} \succeq \mathbf{A}'\mathbf{W}\mathbf{A} + \sup_{\mathbf{x}} \nabla^2 \mathsf{R}(\mathbf{x}). \tag{2.30}$$

The OS-SQS algorithm iteratively chooses a subset S_m and performs

$$\mathbf{x}^+ = \left[\mathbf{x}^- - \mathbf{D}^{-1} \left(\nabla \mathsf{R}\big(\mathbf{x}^-\big) + \frac{N_{\text{view}}}{|S_m|} \sum_{v \in S_m} \mathbf{A}_v{}' \mathbf{W}_v \big(\mathbf{A}_v \mathbf{x}^- - \mathbf{y}_v \big) \right) \right]_{\Omega}. \tag{2.31}$$

This algorithm computes $\mathbf{x}^+$ using only the views in S_m. Because the computational cost of the forward and backprojections dominates computing the regularizer gradient (which can be less expensive than a one-view forward-backprojection pair), this leads to an approximately $\frac{N_{\text{view}}}{|S_m|}$-times speedup. Conventionally, one "iteration" of OS-SQS involves one update using each of the subsets.

The disadvantage of the ordered subsets approximation is that the "subset balance" condition (2.29) is mostly heuristic. Algorithms that use the OS approximation without some sort of stabilizing relaxation have unknown convergence properties. Generally, using more subsets (and fewer views in each subset) leads to a looser approximation (2.29) and a more unstable algorithm. The duality-based method we use in Chapters 5 and 6

partially overcomes these problems and allows the image to be updated one *view* at a time while remaining convergent.

The OS-SQS algorithm was extended with first-order accelerations by Kim et al. [38, 44]. The resulting algorithm has demonstrated state of the art performance and serves as a benchmark for our reconstruction experiments in Chapters 5 and 6.

CHAPTER 3

Circulant Approximation

Many operators in image processing have "somewhat shift-invariant" behavior. Modeling these operators with combinations of circulant and diagonal matrices can yield fast algorithms, but most circulant approximations are currently designed by hand, guided by intuition rather than direct analysis of the operators in question. This chapter proposes a systematic way to analyze arbitrary symmetric linear operators. To make implementations of circulant preconditioners more efficient on the GPU, we present an algorithm that approximates a dense positive-definite circulant matrix with another, more efficient convolution-based operator. We present results from quadratic tomography problems and image denoising.

3.1 Introduction

Let $\mathbf{Y} \in \mathbb{R}^{N \times N}$ be a given symmetric operator. We suppose that $\mathbf{Y}$ is "somewhat shift invariant;" i.e., it can be well-approximated by a circulant matrix:

$$\widehat{\mathbf{C}} = \underset{\mathbf{C} \text{ circulant}}{\operatorname{argmin}} \frac{1}{2}||\mathbf{Y} - \mathbf{C}||_F^2, \tag{3.1}$$

with $\left|\left|\widehat{\mathbf{C}} - \mathbf{Y}\right|\right|_F^2$ small. This is an easy projection to perform:

$$\widehat{\mathbf{C}} = \frac{1}{N} \sum_{j=1}^{N} \operatorname{circ}\{\mathbf{P}_j \mathbf{y}_j\}, \tag{3.2}$$

This chapter is partially based on [51, 57].

where $\mathbf{y}_j$ is the jth column of $\mathbf{Y}$ and $\mathbf{P}_j$ rotates the coordinates of $\mathbf{y}_j$ until the jth element lies in the first entry. Purely circulant approximations like this have been successful in image denoising and 2D CT reconstruction [15, 74]. Although this approximation is easy to compute, many practical operators in imaging are not well approximated by a lone circulant matrix. Adding additional space-varying weights and modes of circulant behavior can improve performance [23, 27].

In this chapter, we consider approximating $\mathbf{Y}$ with sums of diagonal-circulant-diagonal matrix products:

$$\mathbf{Y} \approx \sum_{k=1}^{K} \mathbf{D}_k \mathbf{C}_k \mathbf{D}_k. \tag{3.3}$$

We use our proposed method to analyze a case where circulant preconditioning by approximating the cost function Hessian works very well (image denoising), and a case where it does not (quadratic 3D CT tomography problems). To improve performance on GPUs, we present a method to replace the full circulant matrices $\mathbf{C}_k$ with positive-definite sparse approximations.

3.2 Circulant decomposition

To tease out the circulant structures in $\mathbf{Y}$ we first apply a permutation operator. The approach we take is similar to the Kronecker product approximation technique in [84].

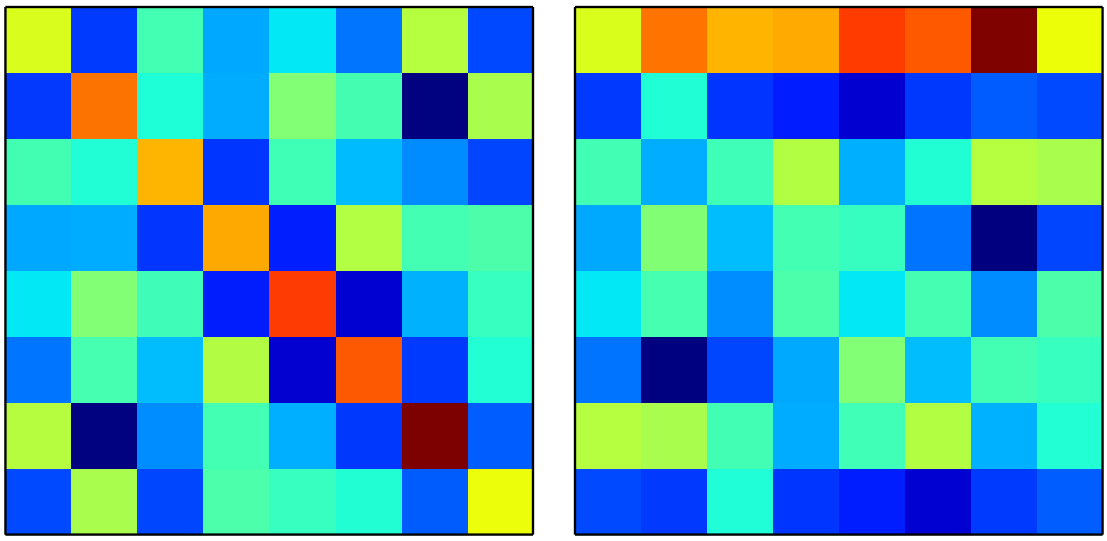
Let $\mathbf{P}_k$ be a permutation matrix that "rotates" the kth element of vectors it operates on to the 1st coordinate:

$$[\mathbf{P}_k \mathbf{v}]_j = v_{1+(|k+j-2| \bmod N)} \tag{3.4}$$

Let $\mathcal{C} : \mathbb{R}^{N \times N} \to \mathbb{R}^{N \times N}$ be the following matrix permutation operator:

$$\mathcal{C}(\mathbf{Y}) = \begin{bmatrix} \mathbf{P}_1 \mathbf{h}_1 & \mathbf{P}_2 \mathbf{h}_2 & \cdots & \mathbf{P}_N \mathbf{h}_N \end{bmatrix}. \tag{3.5}$$

This operator rotates the elements each column of $\mathbf{Y}$ until the coordinate that was on the diagonal is the 1st entry. Figure 3.1 illustrates the behavior of $\mathcal{C}$. Because $\mathcal{C}$ only permutes the entries of its argument it is invertible, linear, and preserves Frobenius norm.



(a) $\mathbf{M}$ (b) $\mathcal{C}(\mathbf{M})$

Figure 3.1: Illustration of a matrix $\mathbf{M}$ and its permuted form $\mathcal{C}(\mathbf{M})$.

Let $\mathbf{X}$ be a matrix of the following form:

$$\mathbf{X} = \sum_{k=1}^{K} \mathbf{C}_k \mathbf{D}_k, \tag{3.6}$$

with circulant $\{\mathbf{C}_k\}$ and diagonal $\{\mathbf{D}_k\}$. Applying $\mathcal{C}$ to a matrix of this form transforms it into a rank-K outer product[1]:

$$\mathcal{C}\left(\sum_{k=1}^{K} \mathbf{C}_k \mathbf{D}_k\right) = \sum_{k=1}^{K} \mathbf{c}_k \mathbf{d}_k{}'. \tag{3.7}$$

The $\{\mathbf{c}_k\}$ are the convolution kernels of the $\{\mathbf{C}_k\}$ and the $\{\mathbf{d}_k\}$ are the diagonals of the $\{\mathbf{D}_k\}$.

The permutation operator $\mathcal{C}$ transforms the diagonal-circulant matrix approximation problem,

$$\widehat{\mathbf{X}} = \underset{\{\mathbf{C}_k\},\{\mathbf{D}_k\}}{\operatorname{argmin}} \frac{1}{2} \left\| \mathbf{Y} - \sum_{k=1}^{K} \mathbf{C}_k \mathbf{D}_k \right\|_F^2, \tag{3.8}$$

[1]See Appendix A.

into a low-rank approximation problem,

$$= \underset{\{\mathbf{C}_k\},\{\mathbf{D}_k\}}{\operatorname{argmin}} \frac{1}{2} \left\| \mathcal{C}\left(\mathbf{Y} - \sum_{k=1}^{K} \mathbf{C}_k \mathbf{D}_k \right) \right\|_F^2 \tag{3.9}$$

$$= \underset{\{\mathbf{c}_k\},\{\mathbf{d}_k\}}{\operatorname{argmin}} \frac{1}{2} \left\| \mathcal{C}(\mathbf{Y}) - \sum_{k=1}^{K} \mathbf{c}_k \mathbf{d}_k{}' \right\|_F^2 . \tag{3.10}$$

The solution to (3.10), by the Eckart-Young theorem, is composed of the first K singular values and vectors of $\mathcal{C}(\mathbf{Y})$. One interesting result of (3.10) is that any square matrix can be written as a sum of N circulant-diagonal matrix products of the form (3.6). How well this approximation can be performed with a low number of terms depends on the singular value spectrum of $\mathcal{C}(\mathbf{Y})$. We use this property to investigate the suitability of circulant approximations for a couple applications in Section 3.3.

3.2.1 Large matrices

Although it is theoretically simple to invoke the singular value decomposition (SVD) of $\mathcal{C}(\mathbf{Y})$, computing the full SVD of an $N \times N$ matrix from an image processing problem often infeasible because of the high dimension N. There are many algorithms to approximate the SVD of a matrix from limited samples [6, 32, 75].

3.2.2 Post-processing

The low-rank approximation $\widehat{\mathbf{X}}$ is neither symmetric nor in the diagonal-circulant-diagonal form we want (3.3). This section gives a series of manipulations and approximations to get $\widehat{\mathbf{X}}$ into this form.

First, we compute the symmetric matrix $\mathbf{S} = \frac{1}{2}\left(\widehat{\mathbf{X}} + \widehat{\mathbf{X}}'\right)$. Because $\mathbf{Y}$ is symmetric, this

actually improves the quality of the approximation:

$$||\mathbf{S}-\mathbf{Y}|| = \left|\left|\frac{1}{2}\left(\widehat{\mathbf{X}}+\widehat{\mathbf{X}}'\right)-\mathbf{Y}\right|\right| \tag{3.11}$$

$$= \left|\left|\frac{1}{2}\left(\widehat{\mathbf{X}}-\mathbf{Y}\right)+\frac{1}{2}\left(\widehat{\mathbf{X}}'-\mathbf{Y}'\right)\right|\right| \tag{3.12}$$

$$\leq \frac{1}{2}\left|\left|\widehat{\mathbf{X}}-\mathbf{Y}\right|\right|+\frac{1}{2}\left|\left|\widehat{\mathbf{X}}'-\mathbf{Y}'\right|\right| \tag{3.13}$$

$$= \left|\left|\widehat{\mathbf{X}}-\mathbf{Y}\right|\right|. \tag{3.14}$$

Next, we need to write $\mathbf{S}$ as a sum of diagonal-circulant-diagonal matrices. Let $\mathbf{C}_k = \mathbf{E}_k+\mathbf{O}_k$ be the decomposition of the circulant matrix $\mathbf{C}_k$ into its even and odd components. That is, $\mathbf{E}_k = \mathbf{E}_k{}'$ and $\mathbf{O}_k = -\mathbf{O}_k{}'$. There exist diagonal matrices $\{\mathbf{G}_k\}$, $\{\mathbf{H}_k\}$ and scalars $\{\sigma_{k,1}\}$ and $\{\sigma_{k,2}\}$ such that[2]

$$\mathbf{S} = \sum_{k=1}^{K} \sigma_{k,1}\mathbf{G}_k\mathbf{E}_k\mathbf{G}_k + \sigma_{k,2}\mathbf{H}_k\mathbf{E}_k\mathbf{H}_k. \tag{3.15}$$

If one wants to approximate $\mathbf{H}$ using a sum of only positive-semidefinite terms, we need to further expand $\mathbf{E}_k$ into its positive-semidefinite and negative-definite terms. Let

$$\mathbf{E}_k = \mathbf{E}_k^+ + \mathbf{E}_k^- \tag{3.16}$$

with $\mathbf{E}_k^+ \succeq \mathbf{0}$ and $\mathbf{E}_k^- \prec \mathbf{0}$. Because $\mathbf{E}_k$ is circulant, its diagonalization via the discrete Fourier transform is readily accessible, so $\mathbf{E}_k^+$ and $\mathbf{E}_k^-$ can be easily computed. The following matrix is not necessarily optimal (as it does not account for interactions between the K terms in the sum), but is positive-semidefinite:

$$\begin{aligned}\widetilde{\mathbf{S}}^+ = \sum_{k=1}^{K} &\max\{0,\sigma_{k,1}\}\cdot\left(\mathbf{G}_k\mathbf{E}_k^+\mathbf{G}_k\right)+\max\{0,\sigma_{k,2}\}\cdot\left(\mathbf{H}_k\mathbf{E}_k^+\mathbf{H}_k\right)\\ &+\min\{0,\sigma_{k,1}\}\cdot\left(\mathbf{G}_k\mathbf{E}_k^-\mathbf{G}_k\right)+\min\{0,\sigma_{k,2}\}\cdot\left(\mathbf{H}_k\mathbf{E}_k^-\mathbf{H}_k\right).\end{aligned} \tag{3.17}$$

[2]See Appendix (B).

3.3 Two examples

In this section, we evaluate how well several Hessian matrices from denoising and CT reconstruction problems can be approximated by circulant matrices.

3.3.1 CT system Gram matrix

The "tomography Hessian," both without statistical weights ($\mathbf{A}'\mathbf{A}$) and with statistical weights ($\mathbf{A}'\mathbf{W}\mathbf{A}$) is an important operator in CT reconstruction. Many attempts have been made to model or precondition it with circulant operators, with varying degrees of success [15, 23, 57, 74, 89]. Folk knowledge holds the following:

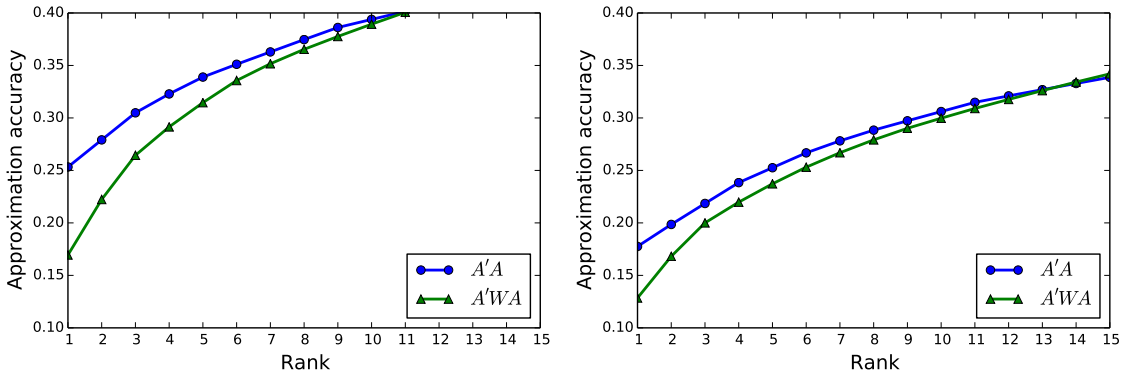
- it is much easier to model $\mathbf{A}'\mathbf{W}\mathbf{A}$ and $\mathbf{A}'\mathbf{A}$ in 2D than in 3D, and in 2D the $\mathbf{A}'\mathbf{A}$ is very well approximated with a circulant operator; and
- it is generally harder to model $\mathbf{A}'\mathbf{W}\mathbf{A}$ than $\mathbf{A}'\mathbf{A}$, due to considerable shift variance introduced by the statistical weights.

In this experiment, we mostly validate these claims.

The columns of the tomography Hessian are dense. Consequently, we cannot compute and store the entire Hessian. In this experiment, we downsampled a $512 \times 512 \times 96$ cone-beam axial geometry by a factor of 16 in all dimensions. We also considered a 16x-downsampled 512×512 two-dimensional geometry. We computed all entries of the tomography Hessian both with and without statical weights (from a simulated XCAT phantom [78]) on the downsampled geometry.

Figure 3.2 contains scree plots of the spectra of the permuted tomography Hessians for the two-dimensional and three-dimensional geometries. Figure 3.2 supports both the items of folk knowledge above. For both the two-dimensional and three-dimensional geometries, $\mathbf{A}'\mathbf{A}$ was easier to approximate with circulant and diagonal matrices than $\mathbf{A}'\mathbf{W}\mathbf{A}$. The two-dimensional Hessians also had significantly more concentrated spectra than the three-dimensional Hessians. Figures 3.3 and 3.4 illustrate the first few principle filters and weights for $\mathbf{A}'\mathbf{A}$ and $\mathbf{A}'\mathbf{W}\mathbf{A}$ in the downsampled three-dimensional geometry.

These are negative results. After circulant approximations were used so effectively to build circulant preconditioners in the 2D setting [74], we had hoped to find the same in 3D [57]. Unfortunately, we have found it challenging to design effective circulant approximations in 3D by hand. Instead of showing us a counterintuitive way to form such



(a) Two-dimensional geometry (b) Three-dimensional geometry

Figure 3.2: Scree plots for the spectrum of the permuted tomography Hessian for downsampled two-dimensional and three-dimensional geometries.

a circulant approximation, these results indicate that a good circulant approximation to $\mathbf{A'WA}$ and even $\mathbf{A'A}$ may not exist in 3D.

3.3.2 Denoising

Consider the edge-preserving regularizer R:

$$\mathsf{R}(\mathbf{Cx}) = \sum_{k=1}^{K} \beta_k \psi([\mathbf{Cx}]_k) \tag{3.18}$$

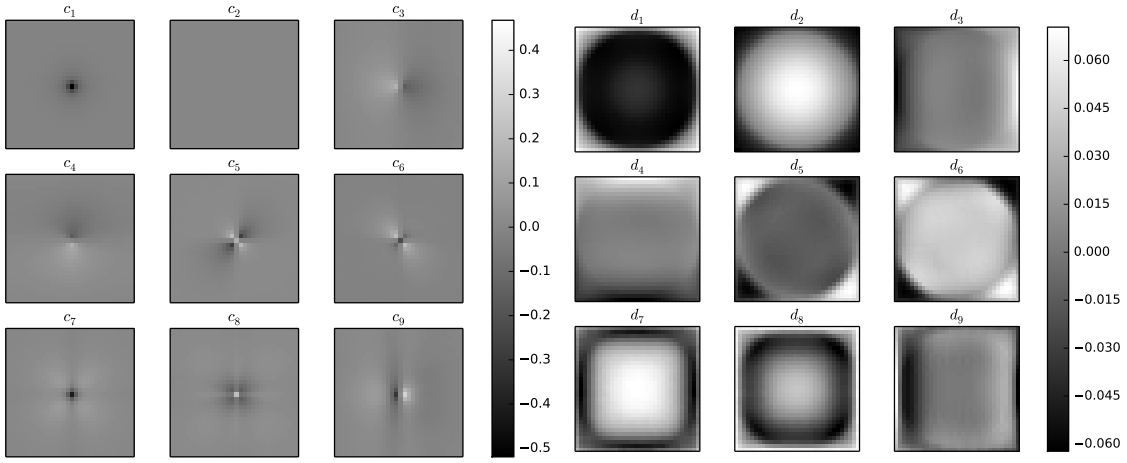
$$= \sum_{j=1}^{N} \sum_{k \in \mathcal{N}_j} \beta_{jk} \psi(x_j - x_k). \tag{3.19}$$

In this 2D example, we penalize all eight two-dimensional neighbors with the Fair potential function ψ,

$$\psi(t) = \delta^2 \left(\left| \frac{t}{\delta} \right| - \log \left(1 + \left| \frac{t}{\delta} \right| \right) \right), \tag{3.20}$$

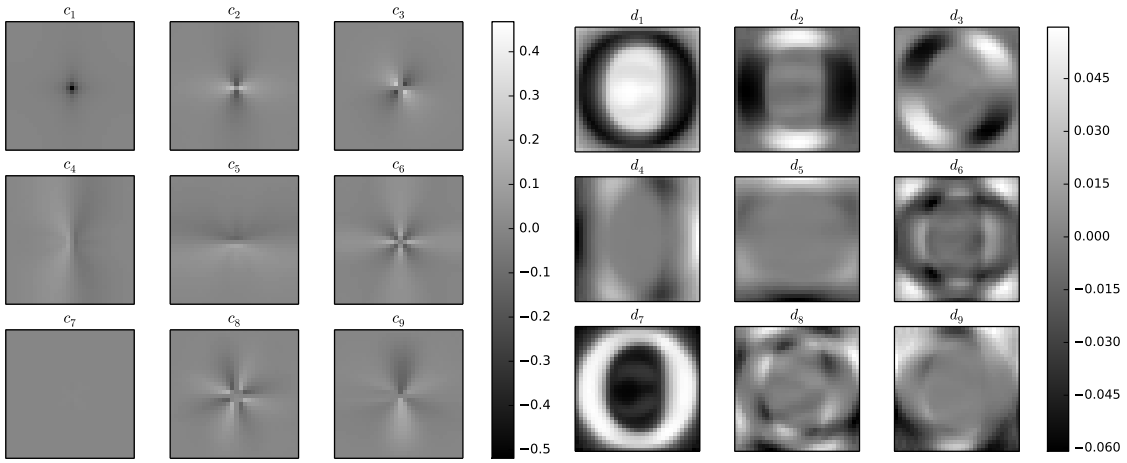
with $\delta = 1$ gray level. The weights β_{jk} are uniform.

Figure 3.5 illustrates the standard 512×512-pixel `cameraman` test image with and without additional additive white Gaussian noise. The Hessian of R is very sparse: each column will have at most nine nonzero entries. Consequently, it is feasible to compute the and store the permuted Hessian $\mathcal{C}(\nabla^2 \mathsf{R})$ as a 9×512^2 matrix. We did so, evaluating the



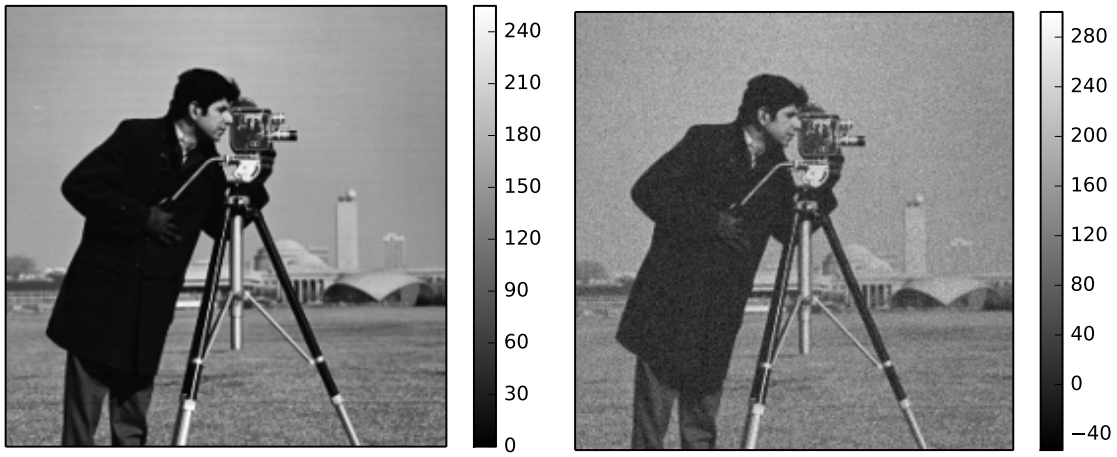
(a) Filters (b) Weights

Figure 3.3: Filters (left singular vectors) and weights (right singular vectors) from the circulant representation of $\mathbf{A}'\mathbf{A}$ in the three-dimensional geometry. Only the center slice of the filters and the weights is shown.



(a) Filters (b) Weights

Figure 3.4: Filters (left singular vectors) and weights (right singular vectors) from the circulant representation of $\mathbf{A}'\mathbf{W}\mathbf{A}$ in the three-dimensional geometry. Only the center slice of the filters and the weights is shown.



(a) Cameraman

(b) Noisy cameraman

Figure 3.5: Cameraman images with and without additional noise.

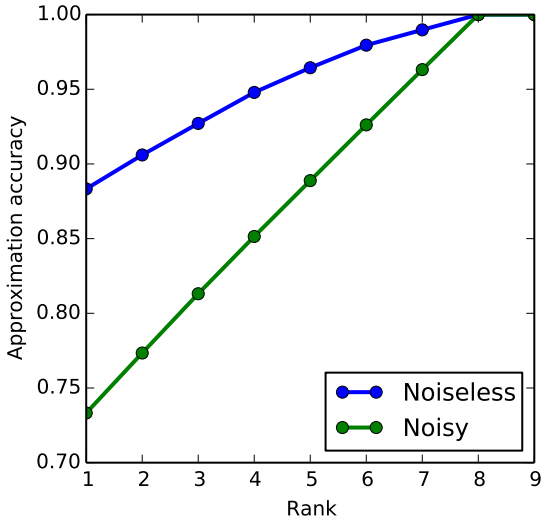
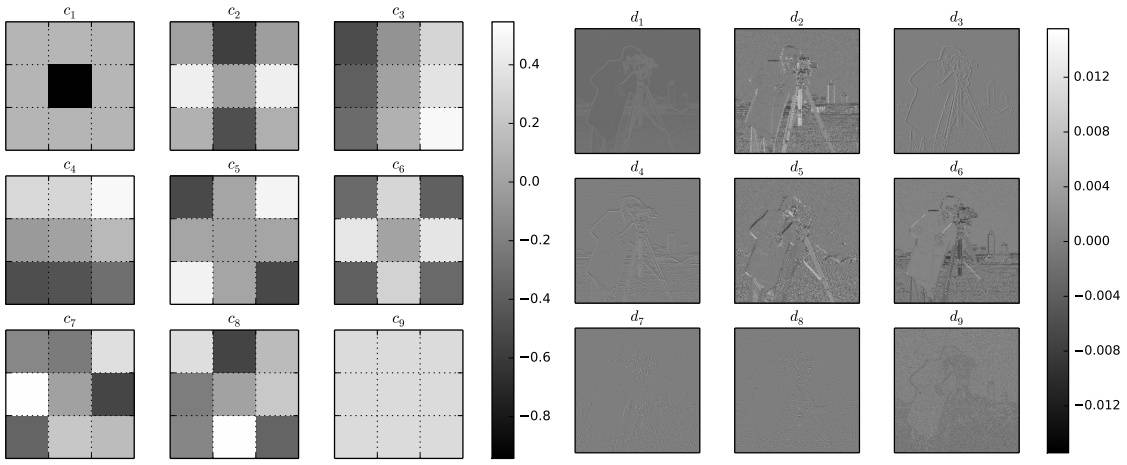


Figure 3.6: Scree plot of circulant spectra of the regularizer Hessian for the noisy and noiseless cases.



(a) Filters

(b) Weights

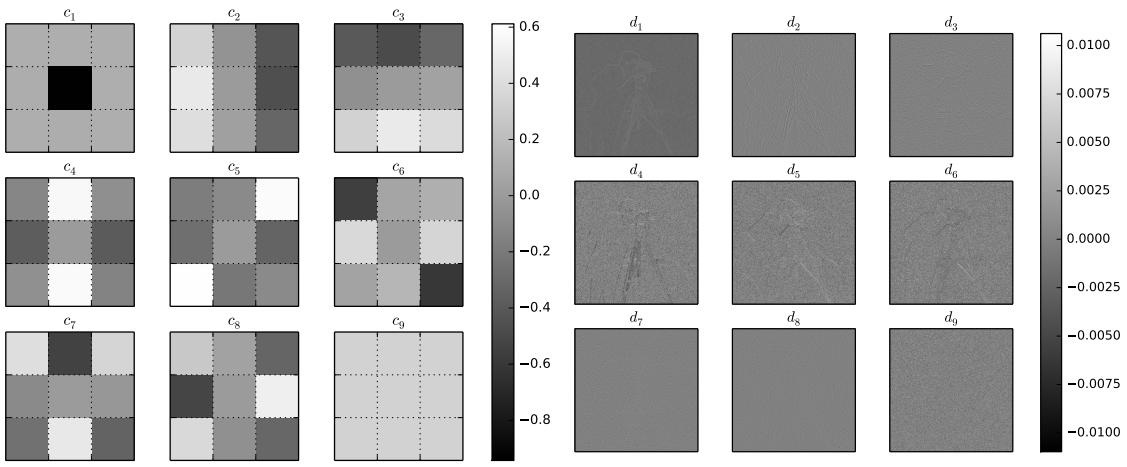
Figure 3.7: Filters (left singular vectors) and weights (right singular vectors) of the regularizer Hessian for the noiseless cameraman image.

regularizer Hessian at both the noiseless and noisy cameraman images. To test our expectation that the regularizer Hessian is "somewhat shift invariant," we generated scree plots of the singular values of $\mathcal{C}(\nabla^2 \mathsf{R})$ for both cases. Figure 3.6 demonstrates that the spectrum is highly concentrated; i.e., $\mathcal{C}(\nabla^2 \mathsf{R})$ can be succinctly approximated using a small number of circulant-diagonal matrix pairs. Intuitively, the spectrum in the noisy case was less concentrated than in the case without additional noise. In either case, this experiment validates our expectation that regularizer Hessian is "somewhat shift invariant."

Figures 3.7 and 3.8 illustrate the filters (left singular vectors) and weights (right singular vectors) derived from the singular value decomposition of the permuted regularizer Hessian, $\mathcal{C}(\nabla^2 R)$, for the noisy and noiseless cases, respectively. (The images are very high-DPI, and detail becomes clear upon zooming in.)

Viewed this way, the regularizer Hessian can be used as a sort of edge detector: each "mode" of $\nabla^2 \mathsf{R}$ (illustrated on the left) corresponds to a certain type of edge, and those edges are highlighted by the weights (illustrated on the right). In both cases, the first singular value/vector pair dominates the rest, and in both cases this is the only seemingly "anisotropic" mode.

Encouraged by these results, in Section 3.5 we design circulant preconditioners for 3D denoising problems that exploit the apparent shift-invariant structure in the regularizer Hessian. But first, in the next section, we present a way to approximate dense circulant matrices (i.e., the $\mathbf{C}_k$) with sparse positive-definite circulant approximations that are clas-



(a) Filters

(b) Weights

Figure 3.8: Filters (left singular vectors) and weights (right singular vectors) of the regularizer Hessian for the noisy cameraman image.

sically appropriate for preconditioning.

3.4 Sparse positive-definite convolution filters

Suppose we have a dense positive-definite circulant matrix $\mathbf{C}$ and our goal is to find a sparse circulant matrix $\mathbf{F}$ such that $\mathbf{F} \approx \mathbf{C}$. Mathematically, we seek

$$\widehat{\mathbf{F}} = \underset{\mathbf{F}\in\Omega, \mathbf{F}\succ\mathbf{0}}{\operatorname{argmin}} \; ||\mathbf{F} - \mathbf{C}||_p^p, \tag{3.21}$$

where $||\cdot||_p$ is the Schatten p-norm and Ω is the set of circulant matrices with a predetermined sparsity pattern:

$$\Omega = \Big\{ \mathbf{X} = \operatorname{circ}\{\mathbf{x}\} : [\mathbf{x}]_j \neq 0 \Leftrightarrow j \in \mathcal{I} \Big\}. \tag{3.22}$$

We assume the sparsity pattern $\mathcal{I}$ is symmetric and includes the diagonal.

Simultaneously satisfying the positive-definiteness constraint ($\mathbf{F} \succ \mathbf{0}$) and the footprint constraint ($\mathbf{F} \in \Omega$) is challenging. Without the positive-definiteness requirement, (3.21) is trivially solved by projecting onto Ω; i.e., setting all the entries of $\mathbf{C}$ outside $\mathcal{I}$ to zero. Under the Schatten ∞-norm and with no positive-definiteness requirement, (3.21) is the problem solved by the classical Parks-McClellan algorithm [72]. The design problem

(3.21) with the positive-definiteness requirement but no footprint requirement is trivial, as $\mathbf{C}$ is assumed to be positive-definite.

Fortunately, the Schatten p-norms are convex for $p \geq 1$ and both the positive-definite cone and Ω are convex, so we can use variable splitting to separate the two difficult constraints. We introduce the auxiliary variable $\mathbf{H} \in \mathbb{R}^{N \times N}$ with the constraint $\mathbf{H} = \mathbf{F}$ and rewrite (3.21) as

$$\widehat{\mathbf{F}} = \underset{\mathbf{F} \in \Omega, \mathbf{H} \succ \mathbf{0}}{\operatorname{argmin}} \; ||\mathbf{H} - \mathbf{C}||_p^p \qquad \text{such that } \mathbf{F} = \mathbf{H}. \tag{3.23}$$

The modified augmented Lagrangian corresponding to this constrained problem is

$$\mathsf{L}(\mathbf{F}, \mathbf{H}; \mathbf{\Gamma}) = ||\mathbf{H} - \mathbf{C}||_p^p + \frac{\mu}{2} ||\mathbf{F} - \mathbf{H} + \mathbf{\Gamma}||_F^2, \tag{3.24}$$

with penalty parameter μ and dual variable $\mathbf{\Gamma}$. The alternating directions method of multipliers updates are

$$\mathbf{F}^{(n+1)} = \underset{\mathbf{F} \in \Omega}{\operatorname{argmin}} \; \frac{\mu}{2} \big|\big| \mathbf{F} - \mathbf{H}^{(n)} + \mathbf{\Gamma}^{(n)} \big|\big|_F^2 = \operatorname{proj}_{\Omega} \big\{ \mathbf{H}^{(n)} - \mathbf{\Gamma}^{(n)} \big\}, \tag{3.25}$$

$$\mathbf{H}^{(n+1)} = \underset{\mathbf{H} \succeq \mathbf{0}}{\operatorname{argmin}} \; ||\mathbf{H} - \mathbf{C}||_p^p + \frac{\mu}{2} \big|\big| \mathbf{H} - \mathbf{F}^{(n+1)} - \mathbf{\Gamma}^{(n)} \big|\big|_F^2, \tag{3.26}$$

$$\mathbf{\Gamma}^{(n+1)} = \mathbf{\Gamma}^{(n)} + \mathbf{F}^{(n+1)} - \mathbf{H}^{(n+1)}. \tag{3.27}$$

As written, these updates are in terms of $\mathbb{R}^{N \times N}$ matrices. Fortunately, $\mathbf{F}^{(n)}$ is always a circulant matrix. With $\mathbf{\Gamma}^{(0)} = \mathbf{0}$, as is conventional, $\mathbf{H}^{(n)}$ and $\mathbf{\Gamma}^{(n)}$ are also always circulant matrices. We can then rewrite the ADMM updates in terms of the filter kernels of $\mathbf{F}^{(n)}$, $\mathbf{H}^{(n)}$, $\mathbf{C}$ and $\mathbf{\Gamma}^{(n)}$. The $\mathbf{H}$ update (3.26), can be further simplified in terms of the DFTs of $\mathbf{f}^{(n)}$, $\mathbf{c}$ and $\boldsymbol{\gamma}^{(n)}$, which we indicate with a tilde:

$$\mathbf{f}^{(n+1)} = \operatorname{proj}_{\Omega} \big\{ \mathbf{h}^{(n)} - \boldsymbol{\gamma}^{(n)} \big\}, \tag{3.28}$$

$$\mathbf{h}^{(n+1)} = \mathbf{U}_{\text{DFT}}{}' \widetilde{\mathbf{h}}^{(n+1)} \tag{3.29}$$

$$\widetilde{\mathbf{h}}^{(n+1)} = \underset{\widetilde{\mathbf{h}} \geq \mathbf{0}}{\operatorname{argmin}} \; \Big|\Big| \widetilde{\mathbf{h}} - \mathbf{U}_{\text{DFT}} \mathbf{c} \Big|\Big|_p^p + \frac{\mu}{2} \Big|\Big| \widetilde{\mathbf{h}} - \mathbf{U}_{\text{DFT}} \big(\mathbf{f}^{(n+1)} - \boldsymbol{\gamma}^{(n)} \big) \Big|\Big|_2^2 \tag{3.30}$$

$$\boldsymbol{\gamma}^{(n+1)} = \boldsymbol{\gamma}^{(n)} + \mathbf{f}^{(n+1)} - \mathbf{h}^{(n+1)}. \tag{3.31}$$

The $\mathbf{f}$ update (3.28) sets entries of $\mathbf{h}^{(n)} + \boldsymbol{\gamma}^{(n)}$ outside of the index set $\mathcal{I}$ to zero. The $\widetilde{\mathbf{h}}$ update, which computes the spectrum of $\mathbf{h}^{(n+1)}$, is a shrinkage-type update. For all $p \geq 1$

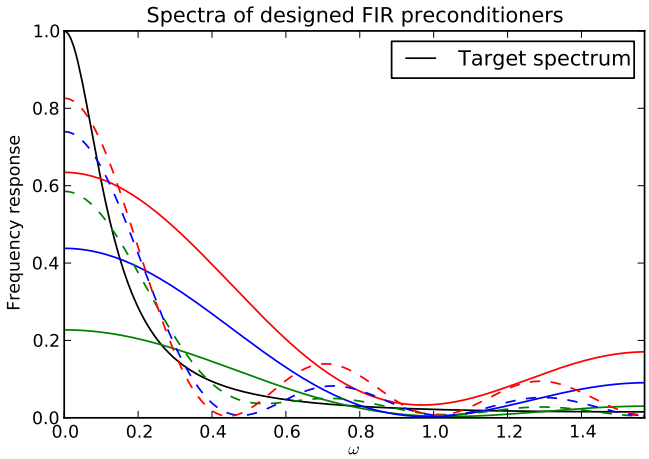


Figure 3.9: Profiles of 1D filters generated by the sparse filter design algorithm. The target spectrum, a low-pass filter similar to those in the denoising experiment, is drawn in black. Green, blue and red lines correspond to filters derived with the Schatten 1, 2 and ∞ norms. The solid lines are 5-tap filters, and the dashed lines are 11-tap filters.

except $p = \infty$, this optimization is separable in the entries of p. For common choices of p (e.g., $p = 1$, $p = 2$), the shrinkage operation can be written in closed form. Altogether, one iteration of the algorithm requires at most three FFTs and a number of coordinate-wise operations.

3.4.1 Footprint selection

In setting up the design problem, we took the index set $\mathcal{I}$ that determines the filter footprint as given. The successive thinning algorithm [3], for example, could be used to algorithmically determine the footprint. Another approach is to replace the projection in the $\mathbf{f}$-update (3.28) with a basis-pursuit-like update:

$$\mathbf{f}^{(n+1)} = \operatorname*{argmin}_{||\mathbf{f}||_0 \leq K} \left|\left|\mathbf{f} - \left(\mathbf{h}^{(n)} - \boldsymbol{\gamma}^{(n)}\right)\right|\right|_2^2. \tag{3.32}$$

We do not consider either of these alternatives further in this chapter. In practice, dense "box-like" footprints appear sufficiently effective. Using more coefficients increases the approximation accuracy at the cost of more expensive convolutions. The optimal trade-off between accuracy and efficiency, as usual, can vary from application to application.

Figure 3.9 illustrates a set of 1D example filters designed by the sparse approximation

algorithm for different footprints and Schatten p-norms. Naturally, increasing the number of taps in the designed filter increased approximation accuracy. Higher-order Schatten p-norms are more sensitive to maximum deviation, normally at the cost of stronger ripples in the spectrum. Despite these differences, we found that the choice of Schatten p-norm had no effect on the convergence rate of the resulting preconditioner algorithms.

3.4.2 Improving accuracy with iteration

Depending on the shape of the spectrum of the full circulant matrix $\mathbf{G}$, it may be more efficient to approximate $\mathbf{C} \approx \mathbf{F}^k \Leftrightarrow \mathbf{F} \approx \mathbf{C}^{1/k}$ for $k \geq 1$ instead of enlarging $\mathbf{F}$'s footprint. For example, an 11×11 two-dimensional convolution has the same "effective footprint" as two composed 5×5 convolutions. However, the 11×11 convolution requires 121 multiply-and-add (MAD) operations, but the two 5×5 convolutions require only 50.

3.5 Denoising experiment

The experiments in Section 3.3 indicate that the Hessian of the denoising problem can be well-approximated by a circulant matrix. This suggests that we can create an effective diagonal-circulant-diagonal preconditioner for these problems. In this section, we partition the image into regions and develop a circulant preconditioner in each one.

Consider the following unconstrained 3D denoising problem

$$\widehat{\mathbf{x}} = \underset{\mathbf{x}}{\operatorname{argmin}} \left\{ \mathsf{J}(\mathbf{x}) = \frac{1}{2}||\mathbf{x} - \mathbf{y}||^2 + \mathsf{R}(\mathbf{x}) \right\}. \tag{3.33}$$

We consider this denoising problem in the context of splitting-based X-ray CT reconstruction. Assume that some variable splitting has been performed to separate the tomographic data-fit term (involving the CT system matrix $\mathbf{A}$) and the edge-preserving regularizer seen in (3.33). In our application, the space-varying weights $\{\beta_{jk}\}$ are used to encourage uniform spatial resolution in the reconstructed object [81].

For this section, we assume that the potential function ψ in the edge preserving regularizer is twice differentiable. We estimate the local strength of the regularizer Hessian with the following approximation:

$$\alpha_j = \frac{\mathbf{e}_j{}'\mathsf{R}(\mathbf{x}_0 + \mathbf{e}_j\epsilon)}{\epsilon \mathbf{e}_j{}'\mathbf{C}'\mathbf{C}\mathbf{e}_j} \tag{3.34}$$

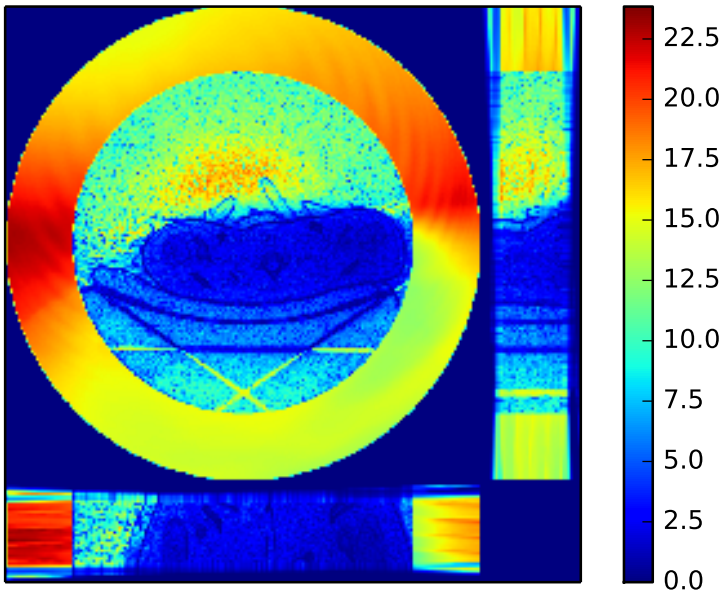


Figure 3.10: Images of α_j for the three central slices of a helical dataset. In clockwise order from the upper left, the slices are transaxial (xy), sagittal (yz) and coronal (xz).

with ϵ small, about 0.01. Figure 3.10 illustrates the α_js for the center slice of a helical chest reconstruction. The figure illustrates a useful property of the local regularizer strength: it is spatially slowly-varying.

3.5.1 Region partitioning

Figure 3.10 shows that the local regularizer strength is slowly-varying over space. In other words, $\alpha_j \approx \alpha_i$ for i spatially near j. Turning this relationship around, if we quantize the α_j into K values, $\{\widetilde{\alpha}_k\}_{k=1}^K$, then the coordinates with α_j best represented by $\widetilde{\alpha}_k$ tend to form contiguous regions of the image. More formally, define the region $\mathcal{R}_k$ as

$$\mathcal{R}_k = \left\{ j : k = \underset{l \in \{1,\ldots,K\}}{\operatorname{argmin}} \; (\alpha_j - \widetilde{\alpha}_l)^2 \right\}. \tag{3.35}$$

In each region $\mathcal{R}_k$, we approximate the Hessian with the shift-invariant matrix

$$\mathbf{H}_k = \mathbf{I} + \widetilde{\alpha}_k \mathbf{C}'\mathbf{C}. \tag{3.36}$$

Figure 3.11 illustrates the quantized values of α_j from the helical dataset illustrated in

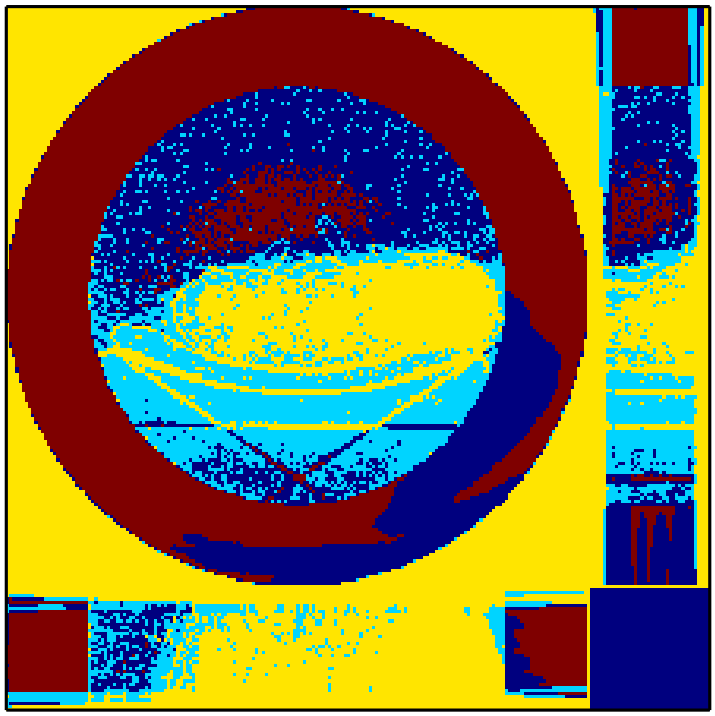


Figure 3.11: Image of the quantized values of α_j from the data illustrated in Figure 3.10 with $K = 4$.

Figure 3.10 with $K = 4$. In each of the differently colored regions we approximate the shift-varying Hessian with a different shift-invariant operator.

Recall that our goal is to design a preconditioner of the following form:

$$\mathbf{P} = \sum_{k=1}^{K} \mathbf{D}_k \mathbf{C}_k \mathbf{D}_k. \tag{3.37}$$

In [51] we used the $\mathbf{D}_k$ to partition the volume into the K different regions and use $\mathbf{C}_k = \mathbf{H}_k^{-1}$. Specifically,

$$\mathbf{d}_k = \operatorname{diag}\{\mathbf{D}_k\} \tag{3.38}$$

$$= \underset{j}{\operatorname{vec}} \begin{cases} 1, & j \in \mathcal{R}_k \\ 0, & \text{else.} \end{cases} \tag{3.39}$$

These $\mathbf{D}_k$ partition the image and the $\mathbf{C}_k$ are positive-definite, so the resulting preconditioner is positive-definite.

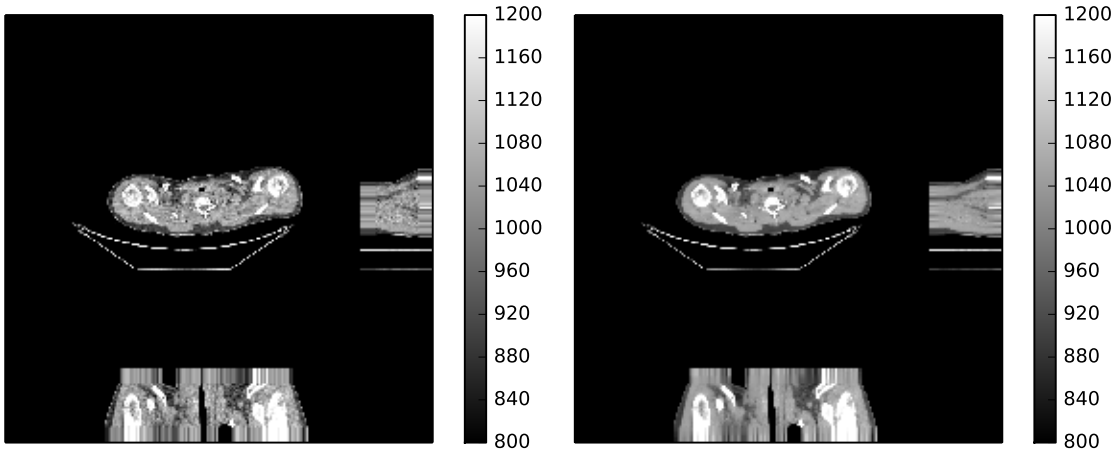
The partition-based weights are intuitive and simple to design, but the "hard" thresholding between regions may introduce undesired artifacts in the preconditioner's output, reducing its effectiveness as an approximation to the Hessian. Fortunately, in the case of the denoising problem, the filters $\mathbf{C}_k = \mathbf{H}_k^{-1}$ are all "low-pass"-like operators. Consequently, they tend to suppress the high-frequency "ringing" artifacts that appear at the boundary of these regions.

3.5.2 Implementation considerations

Consider implementing a general diagonal-circulant-diagonal preconditioner with the form

$$\mathbf{P} = \sum_{k=1}^{K} \mathbf{D}_k \mathbf{C}_k \mathbf{D}_k.$$

The "worst-case" scenario, in terms of computational complexity, is if the $\mathbf{C}_k$ are dense circulant matrices and the $\mathbf{D}_k$ are arbitrary. Storing the $\mathbf{C}_k$ and $\mathbf{D}_k$ on the GPU requires $2K$ image-sized vectors. This is impractical for CT reconstruction problems, where each image-sized vector can be over 670 MB. Consequently the $\mathbf{C}_k$ and $\mathbf{D}_k$ must be expensively transferred to the GPU, or the preconditioner must be implemented on the CPU (i.e., the



(a) Original noisy FBP image. (b) Denoised reference image.

Figure 3.12: Original and reference images for the denoising experiment.

gradient is transferred to the CPU, processed, then transferred back to the GPU). This is may be an unacceptably high cost. And, as discussed earlier, the computational cost of the K FFT and IFFT operations may also be too great.

If the $\mathbf{C}_k$ are instead sparse circulant filters, we must still store the K diagonals of the $\mathbf{D}_k$ on the GPU. This is still a considerable but possibly acceptable cost. The computational complexity of applying the filter is determined mostly by the K convolutions. If the filter footprints are small enough, this can be more efficient than the dense FFT operations. However as K grows, this may grow too large.

Finally suppose the $\mathbf{C}_k$ are sparse circulant filters and the $\mathbf{D}_k$ partition the image domain. Now we can store the $\mathbf{D}_k$ implicitly by instead storing the vector of class indices

$$\mathbf{v} = \operatorname*{vec}_j \{k : j \in \mathcal{R}_k\}. \tag{3.40}$$

Because the partitioning diagonal matrix $\mathbf{D}_k$ appears on both sides of the convolution $\mathbf{C}_k$, we can also reduce the complexity of the convolution. Observe that the jth pixel of $\mathbf{Pg}$ is a convolution of only pixels in $\mathcal{R}_k$, where $j \in \mathcal{R}_k$. Therefore computing each pixel in $\mathbf{d} = \mathbf{Pg}$ requires performing only one convolution. Consequently, the computational complexity of implementing the preconditioner is essentially independent of K.

3.5.3 Experiment

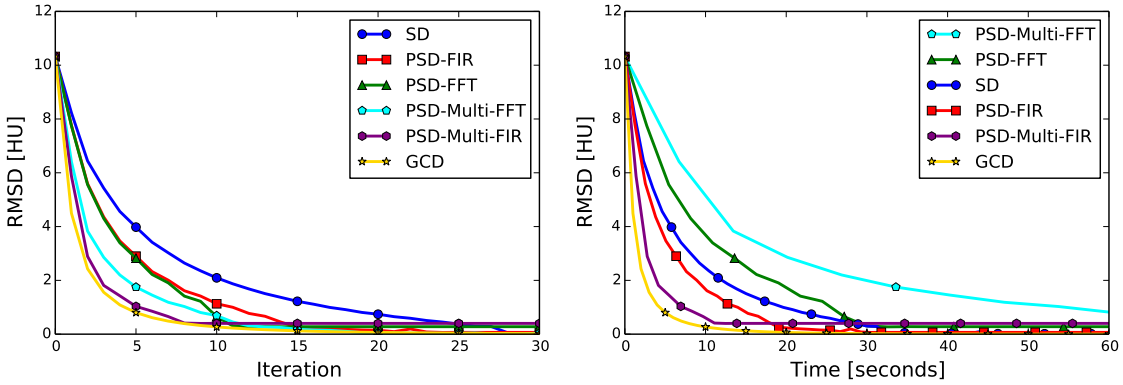
Figure 3.12a illustrates the central three slices of the noisy filtered backprojection image from a helical CT scan. We constructed a denoising problem (3.33) similar to one that would appear in as an inner step in a splitting-based CT reconstruction algorithm. The regularizer penalized differences with all 26 neighbors and used the q-generalized Gaussian potential function. A converged solution is illustrated in Figure 3.12b.

We ran the following algorithms and measured their distance at each iteration to the converged reference:

- steepest descent (SD) with no preconditioner;
- preconditioned steepest descent (PSD-FFT) with a one-term ($K = 1$) diagonal-circulant-diagonal preconditioner implemented with the FFT;
- PSD-FIR, which replaced the dense circulant operator in PSD-FFT with a sparse approximation;
- PSD-Multi-FFT, a $K = 4$-term diagonal-circulant-diagonal preconditioner with dense circulant operators; and
- PSD-Multi-FIR, which replaced the dense circulant operators with sparse approximations.
- GCD, a group coordinate descent algorithm we propose in Chapter 4.

The dense circulant operators were implemented on the CPU using multi-threaded `fftw` calls, and the sparse approximations were $7 \times 7 \times 7$ convolutions. These experiments were run on a workstation with an NVIDIA GTX 480 GPU and an Intel Core i7 860 CPU. We chose steepest descent (instead of conjugate gradients) for its relatively modest memory requirements, but expect similar trends in our results if the experiments were repeated with conjugate gradients.

Figure 3.13a illustrates the RMSD of each algorithm to the converged reference as a function of iteration. All the preconditioned methods converge more quickly per iteration than unpreconditioned steepest descent, and the preconditioner with more terms outperformed the one-term preconditioner. The "-FIR" algorithms, which used sparse approximations instead of dense FFT-based convolutions, performed comparably to their dense counterparts. Figure 3.13a supports both the Hessian parameterization technique for preconditioner design and the proposed sparse approximation algorithm.



(a) RMSD vs. iteration.

(b) RMSD vs. time.

Figure 3.13: Convergence curves for the denoising experiment.

The RMSD of each algorithm to the reference is plotted in Figure 3.13b against time. The computational burden of performing the FFTs on the CPU is striking: both the "-FFT" algorithms converge more slowly than unpreconditioned steepest descent. The cost of transferring the gradient to the host, performing multiple FFTs, and then transferring the preconditioned gradient back to the GPU completely erased the per-iteration acceleration illustrated in Figure 3.13a. On the other hand, the "-FIR" preconditioners, implemented entirely on the GPU, were relatively inexpensive to apply. The 4-term FIR preconditioner was within 2 HU RMSD of the reference in under five seconds. Finally, although the preconditioned steepest descent algorithms using the preconditioners in this section performed very well, the GCD algorithms we introduce in the next chapter converged even more rapidly.

3.6 Conclusions

In this chapter, we presented a method for systematically analyzing the circulant structure of square linear operators. We originally hoped to find useful approximations to various Hessians that appear in CT reconstruction, and we had mixed results. Our experiments on a heavily-downsampled CT system Gram matrix indicated that the system matrix $\mathbf{A}'\mathbf{A}$ is difficult to approximate with circulant matrices in three dimensions, even without the statistical weights $\mathbf{W}$. While our experiments with the regularizer Hessian lead to a fast preconditioner for image denoising problems, the performance of our preconditioners is superseded by the denoising algorithm in the next chapter. We present an algorithm to

solve the difficult-to-precondition tomography problem in Chapter 5.

CHAPTER 4

Fast GPU Denoising

This chapter focuses on the following image denoising problem:

$$\widehat{\mathbf{x}} = \underset{\mathbf{x}\in\Omega}{\operatorname{argmin}} \ \frac{1}{2}||\mathbf{x}-\mathbf{y}||_{\mathbf{W}}^2 + \mathsf{R}(\mathbf{x}) \tag{4.1}$$

with possibly nonuniform weights $\mathbf{W} = \operatorname{diag}_j \{w_j\}$, pixel bounds Ω, and the edge-preserving regularizer R. This type of problem arises from variable splitting algorithms [57, 67], majorize-minimize algorithms [1, 44, 58], and in algorithms for photographic enhancement. Using group coordinate descent, we develop an algorithm that iteratively performs thousands of independent one-dimensional pixel update problems. The algorithm has low memory requirements, is highly parallelizable, and maps very well to the GPU. The denoising updates in Chapter 6 are inspired by the algorithm in this chapter.

4.1 Introduction

Image acquisition systems produce measurements corrupted by noise. Removing that noise is called image denoising. Despite decades of research and remarkable successes, image denoising remains a vibrant field [13]. Over that time, image sizes have increased, the computational machinery available has grown in power and undergone significant architectural changes, and new algorithms have been developed for recovering useful information from noise-corrupted data.

Meanwhile, developments in image *reconstruction* have produced algorithms that rely on efficient denoising routines [57, 74]. The measurements in this setting are corrupted by

This chapter is partially based on [52, 54].

noise and distorted by some physical process. Through variable splitting and alternating minimization techniques, the task of forming an image is decomposed into a series of smaller iterated subproblems. One successful family of algorithms separates "inverting" the physical system's behavior from denoising the image. Majorize-minimize algorithms like [1, 42] also involve denoising-like subproblems. These problems can be very high-dimensional: a routine chest X-ray computed tomography (CT) scan has the equivalent number of voxels as a 40 megapixel image and the reconstruction must account for 3D correlations between voxels.

Growing problem sizes pose computational challenges for algorithm designers. Transistor densities continue to increase roughly with Moore's Law, but advances in modern hardware increasingly appear mostly in greater parallel-computing capabilities rather than single-threaded performance. Algorithm designers can no longer rely on developments in processor clock speed to ensure serial algorithms keep pace with increasing problem size. To provide acceptable performance for growing problem sizes, new algorithms should exploit highly parallel hardware architectures.

A poster-child for highly parallel hardware is the graphics processing unit (GPU). GPUs have always been specialized devices for performing many computations in parallel, but using GPU hardware for non-graphics tasks has in the past involved laboriously translating algorithms into "graphics terminology." Fortunately, in the past decade, programming platforms have developed around modern GPUs that enable algorithm designers to harness these massively parallel architectures using familiar C-like languages.

Despite these advances, designing algorithms for the GPU involves different considerations than designing for a conventional CPU. Algorithms for the CPU are often characterized by the number of floating point operations (FLOPs) they perform or the number of times they compute a cost function gradient. To accelerate convergence, algorithms may store extra information (e.g., previous update directions or auxiliary/dual variables) or perform "global" operations (e.g., line searches or inner products). These designs can accelerate an algorithm's per-iteration convergence or reduce the number of FLOPs required to achieve a desired level of accuracy, but their memory requirements do not map well onto the GPU.

An ideal GPU algorithm is composed of a series of entirely independent and parallel tasks performing the same operations on different data. The number of FLOPs can be less important than the parallelizability of those operations. Operations that are classically considered fast, like inner products and FFTs, can be relatively slow on the GPU due to

memory accesses. Memory is also a far more scarce resource on the GPU. This makes successful, but memory-hungry, frameworks like the primal-dual algorithm [10] or variable splitting less suitable on the GPU. Fully exploiting GPU parallelism requires algorithms with local memory accesses and limited memory requirements.

This chapter presents a pair of image denoising algorithms for the GPU. To exploit GPU parallelism, the algorithms use group coordinate descent (GCD) to decompose the image denoising problem into an iterated sequence of independent one-dimensional pixel-update subproblems. They avoid any additional memory requirements and are highly parallelizable. Both algorithms solve these inner pixel-update subproblems using the well-known majorize-minimize framework [34,35] and can handle a range of edge-preserving regularizers. Because of these properties, the proposed algorithms can efficiently solve large image denoising problems.

Section 4.1.1 introduces the image denoising framework and poses the two classes of problems our algorithms solve. Section 4.2 describes the shared GCD structure of our algorithms, and Section 4.3 describes how two specific algorithms solve the inner one-dimensional update problems. The experimental results from large-image denoising and X-ray CT reconstruction in Section 4.4 illustrate the proposed algorithms' performance, and Section 4.5 contains some concluding remarks.

4.1.1 Optimization-based image denoising

Let $\mathbf{y} \in \mathbb{R}^N$ be noisy pixel measurements collected by an imaging system. In this chapter, bold type indicates a vector quantity, and variables not in bold are scalars; the jth element of $\mathbf{y}$ is written y_j. Let w_j be some confidence we have in the jth measurement, e.g., $w_j = \frac{1}{\sigma_j^2}$, the inverse of the variance of y_j. Let $\mathbf{x} \in \chi \subseteq \mathbb{R}^N$ be a candidate denoised image, and let R denote a regularizer on $\mathbf{x}$. The penalized weighted least squares (PWLS) estimate of the image given the noisy measurements $\mathbf{y}$ is the minimizer of the cost function $\mathsf{J}(\mathbf{x})$:

$$\mathsf{J}(\mathbf{x}) = \frac{1}{2}||\mathbf{x} - \mathbf{y}||_{\mathbf{W}}^2 + \mathsf{R}(\mathbf{x}), \tag{4.2}$$

$$\widehat{\mathbf{x}} = \underset{\mathbf{x} \in \chi}{\operatorname{argmin}} \ \mathsf{J}(\mathbf{x}), \tag{4.3}$$

where $\mathbf{W} = \operatorname{diag}_j \{w_j\}$. The domain $\chi = \chi_1 \times \chi_2 \times \cdots \times \chi_N$, with χ_j convex, may codify a range of admissible pixel levels (e.g., 0-255 for image denoising) or nonnegative values for e.g., X-ray CT [83]. Similar to a prior distribution on $\mathbf{x}$, R is chosen to encourage

expectations we have for the image. A simple and popular choice is the first-order edge-preserving regularizer:

$$\mathsf{R}(\mathbf{x}) = \beta \sum_{j=1}^{N} \sum_{l \in \mathcal{N}_j} \kappa_{jl} \psi(x_j - x_l). \tag{4.4}$$

This regularizer imposes a higher penalty on $\mathbf{x}$ as its "roughness" (measured as the differences between nearby pixels) increases. The global parameter β and local parameters $\kappa_{jl} \geq 0$ adjust the strength of the regularizer relative to the data-fit term [26]. The set $\mathcal{N}_j$ contains the neighbors of the jth pixel, as selected by the algorithm designer. The neighborhoods do not contain their centers: i.e., $j \notin \mathcal{N}_j$. In 2D image denoising, using the four or eight nearest neighbors of the jth pixel are common choices; in 3D common choices are the six cardinal neighbors or the twenty-six adjacent voxels. This chapter focuses on these first-order neighborhoods in 2D and 3D, but the presented algorithms can be extended to larger neighborhoods and higher dimensions.

The symmetric and convex potential function ψ adjusts qualitatively how adjacent pixel differences are penalized. Examples of ψ are:

- the quadratic function, $\psi_{\text{quad}}(t) = \frac{1}{2}t^2$;
- smooth nonquadratic regularizers, e.g., the Fair potential [47]

$$\psi_{\text{Fair}}(t;\delta) = \delta^2(|t/\delta| - \log(1 + |t/\delta|));$$

- and the absolute value function, $\psi_{\text{abs}}(t) = |t|$.

Potential functions that are relatively small around the origin (e.g., ψ_{quad} and ψ_{Fair}) preserve small variations between neighboring pixels. The absolute value function is comparatively large around the origin, and can lead to denoised images with "cartoony" uniform regions [69]. On the other hand, potential functions that are relatively small away from the origin (e.g., ψ_{abs} and ψ_{Fair}) penalize large differences (i.e., edges) less than ψ_{quad}. Choosing one of these potential functions makes R an edge-preserving regularizer, and avoids over-smoothing edges in the denoised image $\widehat{\mathbf{x}}$, but it also makes the denoising problem (4.3) more difficult to solve.

Using ψ_{abs} in (4.4) yields the anisotropic TV regularizer [77].

4.2 Group coordinate descent

This section describes the "outer loop" of algorithms designed to solve (4.3) rapidly on the GPU. We use a superscript (n), e.g., $\mathbf{x}^{(n)}$, to indicate the state of a variable in the nth iteration of the algorithm.

Consider optimizing $\mathsf{J}(\mathbf{x})$ in (4.3) with respect to the jth pixel while holding the other pixels constant at $\mathbf{x} = \mathbf{x}^{(n)}$:

$$\underset{x_j \,:\, \mathbf{x} \in \chi}{\operatorname{argmin}} \frac{w_j}{2}(x_j - y_j)^2 + 2\beta \sum_{l \in \mathcal{N}_j} \kappa_{jl} \psi\Big(x_j - x_l^{(n)}\Big). \tag{4.5}$$

The only pixels involved in this optimization are the jth pixel and its neighbors, $\mathcal{N}_j$. Consequently, if the pixels in $\mathcal{N}_j$ are held constant, we can optimize over the jth pixel without any regard for the pixels outside $\mathcal{N}_j$.

Looping j through the pixels of $\mathbf{x}$, $j = 1, \ldots, N$, and performing the one-dimensional update (4.5) is called the coordinate descent algorithm [70]. This algorithm is convergent and monotone in cost function. However, because each optimization is performed serially, coordinate descent is ill-suited to modern highly parallel hardware like the GPU.

GCD algorithms instead optimize over a *group* of elements of $\mathbf{x}$ at a time while holding the others constant. The key to using GCD on a GPU efficiently is choosing appropriate groups that allow massive parallelism. Let $\mathcal{S}_1, \ldots, \mathcal{S}_M$ be a partition of the pixel coordinates of $\mathbf{x}$; we write $\mathbf{x} = [\mathbf{x}_{\mathcal{S}_1}, \ldots, \mathbf{x}_{\mathcal{S}_M}]$. A GCD algorithm that uses these groups to optimize (4.3) will loop over $m = 1, \ldots M$ and solve

$$\mathbf{x}_{\mathcal{S}_m}^{(n+1)} = \underset{\mathbf{x}_{\mathcal{S}_m} \,:\, \mathbf{x} \in \chi}{\operatorname{argmin}} \; \mathsf{J}\Big(\mathbf{x}_{\mathcal{S}_1}^{(n+1)}, \ldots \mathbf{x}_{\mathcal{S}_{m-1}}^{(n+1)}, \mathbf{x}_{\mathcal{S}_m}, \mathbf{x}_{\mathcal{S}_{m+1}}^{(n)}, \ldots, \mathbf{x}_{\mathcal{S}_M}^{(n)}\Big). \tag{4.6}$$

The mth group update subproblem (4.6) is a $|\mathcal{S}_m|$-dimensional problem in general. However, we can design the groups such that each of these subproblems decomposes into $|\mathcal{S}_m|$ completely independent one-dimensional subproblems. If

$$\forall m, \forall j \in \mathcal{S}_m, \quad \mathcal{N}_j \cap \mathcal{S}_m = \emptyset, \tag{4.7}$$

then in each of the group update subproblems (4.6), the neighbors of all the pixels being optimized are held constant. By the local separability property observed above, this breaks the optimization over the pixels in $\mathcal{S}_m$ into $|\mathcal{S}_m|$ *independent* one-dimensional subproblems.

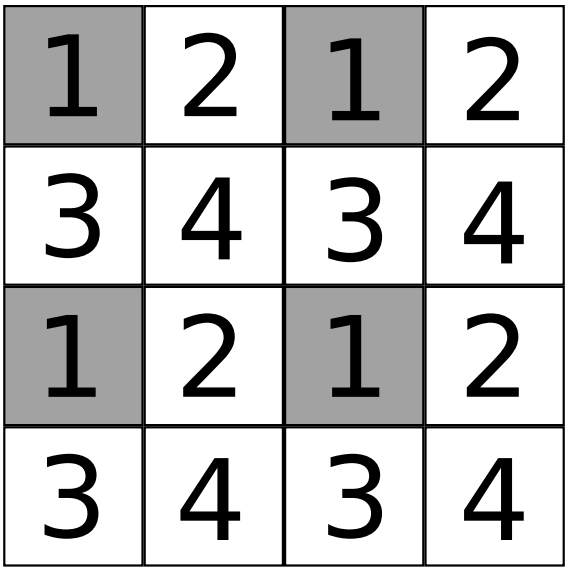


Figure 4.1: Illustration of the groups in (4.7) for a 2D imaging problem with $\mathcal{N}_j$ containing the four or eight pixels adjacent to the jth pixel. Optimizing over the pixels in $\mathcal{S}_1$ (shaded) involves independent one-dimensional update problems for each pixel in the group.

Figure 4.1 illustrates a set of groups that satisfies the "contains no neighbors" (4.7) requirement for a 2D problem and $\mathcal{N}_j$ containing the four or eight pixels adjacent to j. In 3D, both six-neighbor and twenty-six-neighbor $\mathcal{N}_j$ use eight groups arranged in a $2 \times 2 \times 2$ "checkerboard" pattern.

To summarize, we propose GCD algorithms for (4.3) that loop over the groups $m = 1, \ldots, M$ and update the pixels in $\mathcal{S}_m$:

$$\mathbf{x}_{\mathcal{S}_m}^{(n+1)} = \underset{\mathbf{x}_{\mathcal{S}_m} \,:\, \mathbf{x} \in \chi}{\operatorname{argmin}} \sum_{j \in \mathcal{S}_m} \Psi_j^{(n)}(x_j), \qquad \text{where} \tag{4.8}$$

$$\Psi_j^{(n)}(x_j) = \frac{w_j}{2}(x_j - y_j)^2 + 2\beta \sum_{l \in \mathcal{N}_j} \kappa_{jl} \psi\Big(x_j - x_l^{(n)}\Big). \tag{4.9}$$

Each of the $\Psi_j^{(n)}$ are independent one-dimensional functions and are minimized in parallel. Because the pixel updates are performed in-place, this algorithm requires no additional memory beyond storing $\mathbf{x}$, $\mathbf{y}$, $\mathbf{W}$ and the regularizer weights. In many cases, $\mathbf{W}$ and the regularizer weights are uniform, and the algorithm must store only two image-sized vectors! These low memory requirements make the GCD algorithm remarkably well-suited to the GPU. This GCD algorithm is guaranteed to decrease the cost function J monotonically. Convergence to a minimizer of J is ensured under mild regularity condi-

for $n = 1$ up to N_{iter} **do**
  for $m = 1$ up to M **do**
    Parfor $j \in \mathcal{S}_m$
      Minimize $\Psi_j^{(n)}(x_j)$ *w.r.t.* $x_j \in \chi_j$.
    EndParfor
  end for
end for

Figure 4.2: The GCD algorithm structure. The **Parfor** block contains $|\mathcal{S}_k|$ minimizations that are independent and implemented in parallel. Section 4.3 details these optimizations.

tions [35, 36]. Figure 4.2 summarizes the proposed algorithm structure.

4.3 One-dimensional subproblems

The complexity of solving each of the one-dimensional subproblems in (4.8) depends on the choice of potential function ψ. In this chapter, we consider two cases:

- when ψ is convex and differentiable (Section 4.3.1); and
- when ψ is the absolute value function, thus convex but not differentiable (Section 4.3.2).

One could also adapt these methods to non-convex potential functions ψ, albeit with weaker convergence guarantees. In all cases, we approximately solve the one-dimensional subproblem (4.8) using the well-known majorize-minimize (MM) approach, also called optimization transfer and functional substitution [12, 28]. In iteration n, the MM framework generates a surrogate function $\Phi_j^{(n)}$ that may depend on $\mathbf{x}^{(n)}$ and satisfies the following "equality" and "lies-above" properties:

$$\Phi_j^{(n)}\left(x_j^{(n)}\right) = \Psi_j^{(n)}\left(x_j^{(n)}\right) \tag{4.10}$$

$$\Phi_j^{(n)}(x_j) \geq \Psi_j^{(n)}(x_j) \qquad \forall x_j \in \chi_j. \tag{4.11}$$

Majorize-minimize methods update x_j by minimizing $\Phi_j^{(n)}$,

$$x_j^{(n+1)} = \underset{x_j \in \chi_j}{\operatorname{argmin}} \ \Phi_j^{(n)}(x_j). \tag{4.12}$$

Because χ_j is convex, we find the unconstrained solution to (4.12) then project it onto χ_j. This update is guaranteed to decrease both the 1D cost function $\Psi_j^{(n)}(x_j)$ and the global cost function J. Even though we are minimizing the surrogate instead of the single-pixel cost function $\Psi_j^{(n)}(x_j)$, the GCD-MM algorithm is convergent [35].

To implement the MM iteration (4.12), we need to efficiently construct and minimize the surrogate $\Phi_j^{(n)}$. The one-dimensional cost function $\Psi_j^{(n)}$ is the sum of a quadratic term and $|\mathcal{N}_j|$ often nonquadratically penalized differences (the $\psi\left(x_j - x_l^{(n)}\right)$ terms). Figure 4.3 illustrates an example $\Psi_j^{(n)}$ using only three neighbors and the absolute value potential function. The next two subsections describe how we construct a surrogate $\phi_{jl}^{(n)}$ for each of the nonquadratic terms in $\Psi_j^{(n)}$. Replacing each $\psi\left(x_j - x_l^{(n)}\right)$ in (4.9) with its surrogate $\phi_{jl}^{(n)}(x_j)$ gives us the following majorizer for $\Psi_j^{(n)}$ in (4.12):

$$\Phi_j^{(n)}(x_j) = \frac{w_j}{2}(x_j - y_j)^2 + 2\beta \sum_{l \in \mathcal{N}_j} \kappa_{jl} \phi_{jl}^{(n)}(x_j). \tag{4.13}$$

Constructing and minimizing (4.13) requires only a few registers and a small number of visits to each pixel in $\mathcal{N}_j$. This keeps the number of memory accesses low and the acccess pattern regular, which is necessary for good GPU performance.

4.3.1 Convex and differentiable potential function

First we consider the simpler case of a convex and differentiable cost function. Define the Huber curvature $\omega_{jl}^{(n)}$ as

$$\omega_{jl}^{(n)} = \frac{\psi'\left(x_j^{(n)} - x_l^{(n)}\right)}{x_j^{(n)} - x_l^{(n)}}. \tag{4.14}$$

If $\omega_{jl}^{(n)}$ is bounded and nonincreasing as $\left|x_j^{(n)} - x_l^{(n)}\right| \to \infty$, then the following quadratic surrogate majorizes $\psi\left(x_j - x_{jl}^{(n)}\right)$ at $x_j^{(n)}$ and has optimal (i.e., minimal) curvature [33, page 185]:

$$\phi_{jl}^{(n)}(x_j) = \psi\left(x_j^{(n)}\right) + \left(x_j - x_l^{(n)}\right)\psi'\left(x_j^{(n)} - x_l^{(n)}\right) + \frac{\omega_{jl}^{(n)}}{2}\left(x_j - x_l^{(n)}\right)^2. \tag{4.15}$$

Many potential functions have bounded and monotone nonincreasing Huber curvatures, including the Fair potential [47] and the q-Generalized Gaussian potential function sometimes used in X-ray CT reconstruction [83]. Because the Huber curvature is optimally small, the closed-form MM update,

$$x_j^{(n+1)} = x_j^{(n)} - \frac{w_j\left(x_j^{(n)} - y\right) + 2\beta \sum_{l \in \mathcal{N}_j} \kappa_{jl} \psi'\left(x_j^{(n)} - x_l^{(n)}\right)}{w_j + 2\beta \sum_{l \in \mathcal{N}_j} \kappa_{jl} \omega_{jl}^{(n)}}, \tag{4.16}$$

takes the largest step possible for a quadratic majorizer of the form (4.13). To implement (4.16) efficiently, we use (4.14) to replace the ψ' terms with the product of $\omega_{jl}^{(n)}$ and $x_j^{(n)} - x_l^{(n)}$. The resulting algorithm is implemented with only one potential function derivative per neighboring pixel.

4.3.2 The absolute value potential function

The quadratic majorizer in (4.15) applies to a class of differentiable potential functions. TV uses the absolute value potential function, and ψ_{abs} is not differentiable at the origin. In the previous section's terminology, the curvature $\omega_{jl}^{(n)}$ "explodes" if $x_j^{(n)} \approx x_l^{(n)}$. TV denoising encourages neighboring pixels to be identical to one another so this is a significant concern. Even if $x_j^{(n)} \neq x_l^{(n)}$ in practice [71], the exploding surrogate curvature may cause numerical problems.

A way to avoid this problem is to modify the curvatures to prevent the $\omega_{jl}^{(n)}$ from exploding. One approach is to replace ψ_{abs} with the hyperbola potential function, $\psi(t) = \sqrt{\epsilon + t^2} - \sqrt{\epsilon}$, with $\epsilon > 0$ small, or similar "corner-rounded" absolute-value-like function. While this makes the techniques in the previous section directly applicable, it changes the global cost function J, which may be suboptimal.

Another corner rounding approach is to "cap" the curvatures at ϵ^{-1} for small $\epsilon > 0$:

$$\omega_{jl,\epsilon}^{(n)} = \frac{1}{\max\left\{\epsilon, \left|x_j^{(n)} - x_l^{(n)}\right|\right\}}. \tag{4.17}$$

Unfortunately, the quadratic function with curvature $\omega_{jl,\epsilon}^{(n)}$ does not satisfy the "lies above" surrogate requirement (4.11) when $\left|x_j^{(n)} - x_l^{(n)}\right| < \epsilon$. Because $\Phi_j^{(n)}$ would not then be a "proper" surrogate for $\Psi_j^{(n)}$, a GCD algorithm based on (4.17) may not monotonically decrease the cost function J. Empirically, we found that using $\omega_{jl,\epsilon}^{(n)}$ appears to cause $\mathbf{x}^{(n)}$

to enter a suboptimal limit cycle around the optimum. Thus we developed the following duality approach.

4.3.2.1 Duality approach

One way to handle the absolute value function is to use its dual formulation [10,11,54,91]. We write the absolute value function implicitly in terms of a maximization over a dual variable $\gamma_{jl}^{(n)}$:

$$\left|x_j - x_l^{(n)}\right| = \max_{\gamma_{jl}^{(n)} \in [-1,1]} \gamma_{jl}^{(n)} \left(x_j - x_l^{(n)}\right). \tag{4.18}$$

Thus, by choosing any closed interval $\Omega^{(n)} \supseteq [-1,1]$, the following is a surrogate for $\left|x_j - x_l^{(n)}\right|$ that satisfies both the "equality" (4.10) and "lies above" (4.11) majorizer properties:

$$\phi_{jl,\Omega^{(n)}}^{(n)}(x_j) = \max_{\gamma_{jl}^{(n)} \in \Omega^{(n)}} \gamma_{jl}^{(n)} \left(x_j - x_l^{(n)}\right) \kappa_{jl} - \frac{\delta_{jl}^{(n)}}{2} \left(\left(\gamma_{jl}^{(n)}\right)^2 - 1 \right), \tag{4.19}$$

where $\delta_{jl}^{(n)} = \left|x_j^{(n)} - x_l^{(n)}\right| \kappa_{jl}$. When $\Omega^{(n)} = [-1,1]$, $\phi_{jl,\Omega^{(n)}}^{(n)} = \psi_{jl}^{(n)}$. Selecting $\Omega^{(n)}$ larger than $[-1,1]$ increases the domain of maximization in (4.19) and loosens the majorization, and satisfies the "equality" (4.10) and "lies above" (4.11) majorization conditions. Figure 4.4 illustrates $\phi_{jl,\Omega^{(n)}}^{(n)}$ for several choices of $\Omega^{(n)}$.

Let $D = |\mathcal{N}_j|$ be the number of neighbors of the jth pixel. Denote the vector of dual variables $\boldsymbol{\gamma}_j = \left[\gamma_{j1}^{(n)}, \ldots, \gamma_{jD}^{(n)}\right]$ and their domain $\boldsymbol{\Omega}^{(n)} = \Omega^{(n)} \times \cdots \times \Omega^{(n)}$. We plug $\phi_{jl,\Omega^{(n)}}^{(n)}$ into (4.13) to construct the surrogate function $\Phi_j^{(n)}$:

$$\Phi_j^{(n)}(x_j) = \operatorname*{argmax}_{\boldsymbol{\gamma}_j \in \boldsymbol{\Omega}^{(n)}} \mathcal{L}_j^{(n)}(x_j, \boldsymbol{\gamma}_j), \qquad \text{where} \tag{4.20}$$

$$\mathcal{L}_j^{(n)}(x_j, \boldsymbol{\gamma}_j) = \frac{w_j}{2}(x_j - y_j)^2 + 2\beta \sum_{l \in \mathcal{N}_j} \kappa_{jl} \gamma_{jl}^{(n)} \left(x_j - x_l^{(n)}\right) - \frac{\delta_{jl}^{(n)}}{2} \left(\left(\gamma_{jl}^{(n)}\right)^2 - 1 \right) \tag{4.21}$$

Figure 4.3 illustrates $\Psi_j^{(n)}$ and $\Phi_j^{(n)}$ for two values of $x_j^{(n)}$. Note that, unlike the "corner-rounding" approximations, $\Phi_j^{(n)}$ faithfully preserves the nondifferentiable "corner" of $\Psi_j^{(n)}$ at the minimizer, $x_j^{(n)} = 0.1$.

To implement the majorize-minimize procedure (4.12) by minimizing (4.21), we pass

into the dual domain. Observe that $\mathcal{L}_j^{(n)}$ is convex and continuous in x_j and concave and continuous in the $\gamma_{jl}^{(n)}$, and the set $\mathbf{\Omega}^{(n)}$ is compact. We invoke Sion's minimax theorem [80] to transpose the order of minimization and maximization:

$$\underset{x_j}{\operatorname{argmin}} \ \underset{\boldsymbol{\gamma}_j \in \mathbf{\Omega}^{(n)}}{\operatorname{argmax}} \ \mathcal{L}_j^{(n)}(x_j, \boldsymbol{\gamma}_j) = \underset{\boldsymbol{\gamma}_j \in \mathbf{\Omega}^{(n)}}{\operatorname{argmax}} \ \underset{x_j}{\operatorname{argmin}} \ \mathcal{L}_j^{(n)}(x_j, \boldsymbol{\gamma}_j). \tag{4.22}$$

The inner minimization over x_j can now be solved trivially in terms of $\boldsymbol{\gamma}_j^{(n)}$:

$$x_j(\boldsymbol{\gamma}_j) = y_j - \frac{\beta}{w} \sum_{l \in \mathcal{N}_j} \gamma_{jl}^{(n)} \kappa_{jl}. \tag{4.23}$$

Plugging (4.23) into (4.21) and maximizing over $\boldsymbol{\gamma}_j \in \mathbf{\Omega}^{(n)}$, we arrive at the following quadratic dual problem:

$$\boldsymbol{\gamma}_j^* \in \underset{\boldsymbol{\gamma}_j \in \mathbf{\Omega}^{(n)}}{\operatorname{argmax}} \ D^{(n)}(\boldsymbol{\gamma}_j), \qquad \text{where} \tag{4.24}$$

$$D^{(n)}(\boldsymbol{\gamma}_j) = -\frac{1}{2} \boldsymbol{\gamma}_j{}' \left(\mathbf{D} + \frac{1}{w} \boldsymbol{\beta}\boldsymbol{\beta}' \right) \boldsymbol{\gamma}_j + \boldsymbol{\gamma}_j{}' \mathbf{\Lambda} \boldsymbol{\beta}, \tag{4.25}$$

where $\mathbf{D} = \operatorname{diag}_l \left\{ 2\beta \delta_{jl}^{(n)} \right\}$, $\mathbf{\Lambda} = \operatorname{diag}_l \left\{ y_j - x_l^{(n)} \right\}$, and $\boldsymbol{\beta} = \operatorname{vec}_l \{ 2\beta \kappa_{jl} \}$. Because expanding $\Omega^{(n)}$ only "loosens" the majorization $\phi_{jl,\Omega^{(n)}}^{(n)}$ we simply define $\mathbf{\Omega}^{(n)}$ to include the pseudoinverse

$$\boldsymbol{\gamma}_j^+ = \left(\mathbf{D} + \frac{1}{w} \boldsymbol{\beta}\boldsymbol{\beta}' \right)^+ \mathbf{\Lambda} \boldsymbol{\beta}, \tag{4.26}$$

and then solve (4.24) by finding the pseudoinverse. In practice, this means we can solve the dual problem (4.24) as if it were unconstrained.

4.3.2.2 Solving the dual problem

The dual problem (4.24) has a diagonal-plus-rank-1 Hessian that can be trivially inverted when the diagonal matrix $\mathbf{D}$ is full rank. However, when at least one entry of $\mathbf{D}$ is small (i.e., when $x_j^{(n)} \approx x_l^{(n)}$ for some l), the problem becomes ill-conditioned and requires an iterative method or an expensive "direct method" (e.g., computing the eigenvalue decomposition of $\mathbf{D} + \frac{1}{w}\boldsymbol{\beta}\boldsymbol{\beta}'$ or the "matrix pseudoinverse lemma" [46]). We propose an iterative *minor*ize-*max*imize procedure that exploits the diagonal-plus-rank-1 Hessian.

This inner minorize-maximize procedure is iterative, so we denote the subiteration number with a superscripted m. The following function, $S_j^{(m)}(\boldsymbol{\gamma}_j)$ is a minorizer for $D_j^{(n)}(\boldsymbol{\gamma}_j)$ at $\boldsymbol{\gamma}_j^{(m)}$ in the sense that it satisfies the "equality" property (4.10) at $\boldsymbol{\gamma}_j^{(m)}$ and a "lies-below" property analogous to the "lies above" majorization property (4.11):

$$S_j^{(m)}(\boldsymbol{\gamma}_j) = D_j^{(n)}\left(\boldsymbol{\gamma}_j^{(m)}\right) + \left(\boldsymbol{\gamma}_j - \boldsymbol{\gamma}_j^{(m)}\right)'\nabla D_j^{(n)}\left(\boldsymbol{\gamma}_j^{(m)}\right) - \frac{1}{2}\left(\boldsymbol{\gamma}_j - \boldsymbol{\gamma}_j^{(m)}\right)'\left(\mathbf{D}_\epsilon + \frac{1}{w}\boldsymbol{\beta}\boldsymbol{\beta}'\right)\left(\boldsymbol{\gamma}_j - \boldsymbol{\gamma}_j^{(m)}\right), \tag{4.27}$$

where $\mathbf{D}_\epsilon = \text{diag}_l\left\{\max\left\{\epsilon, \mathbf{D}_{ll}\right\}\right\}$. Let $\mathbf{H}_\epsilon = \mathbf{D}_\epsilon + \frac{1}{w}\boldsymbol{\beta}\boldsymbol{\beta}'$. Substituting the "min" for a "max" in the MM procedure (4.12) leads to the following iterative procedure for solving (4.24):

$$\boldsymbol{\gamma}_j^{(m+1)} = \underset{\boldsymbol{\gamma}_j}{\text{argmax}}\ S_j^{(m)}(\boldsymbol{\gamma}_j) \tag{4.28}$$

$$= \mathbf{H}_\epsilon^{-1}\left(\boldsymbol{\Lambda}\boldsymbol{\beta} - \mathbf{M}_\epsilon\boldsymbol{\gamma}_j^{(m)}\right), \tag{4.29}$$

where $\mathbf{M}_\epsilon = \text{diag}_l\left\{\max\left\{0, \epsilon - \delta_{jl}^{(n)}\right\}\right\}$. We multiply by $\mathbf{H}_\epsilon^{-1}$ efficiently using the matrix inversion lemma.

The recursion (4.29) reveals an interesting quality of the minorize-maximize procedure. When all the neighbors $x_l^{(n)}$ are sufficiently different from $x_j^{(n)}$, $\mathbf{M}_\epsilon$ is the zero-matrix and the MM recursion (4.29) is stationary. In other words, $\boldsymbol{\gamma}_j^{(m)}$ converges in a single iteration. This corresponds to the case where the heuristic "capped-curvature" majorize-minimize algorithm produces a valid surrogate. On the other hand, when some $\delta_{jl}^{(n)} \approx 0$, the "capped-curvature" algorithm may produce an invalid majorizer, but the recursion (4.29) will eventually produce (by finding appropriate values for the corresponding $\boldsymbol{\gamma}_{jl}^{(n)}$) and minimize a valid majorizer for $\Psi_j^{(n)}$. A practical alternative to running an arbitrarily large number of inner minorize-majorize iterations is to track the cost function value $\Psi_j^{(n)}\left(x_j\left(\boldsymbol{\gamma}_j^{(m)}\right)\right)$ and terminate the minorize-maximize algorithm when

$$\Psi_j^{(n)}\left(x_j\left(\boldsymbol{\gamma}_j^{(m)}\right)\right) \leq \Psi_j^{(n)}\left(x_j^{(n)}\right). \tag{4.30}$$

This check was inexpensive to integrate into the minorize-maximize iteration, so we used it in the experiments below. Nonetheless, it is possible that in late iterations, as $x_j^{(n)} \approx x_l^{(n)}$, the domain $\boldsymbol{\Omega}^{(n)}$ grows and the majorizer $\Phi_j^{(n)}$ becomes increasingly loose. This would slow the convergence of $\mathbf{x}^{(n)} \to \widehat{\mathbf{x}}$.

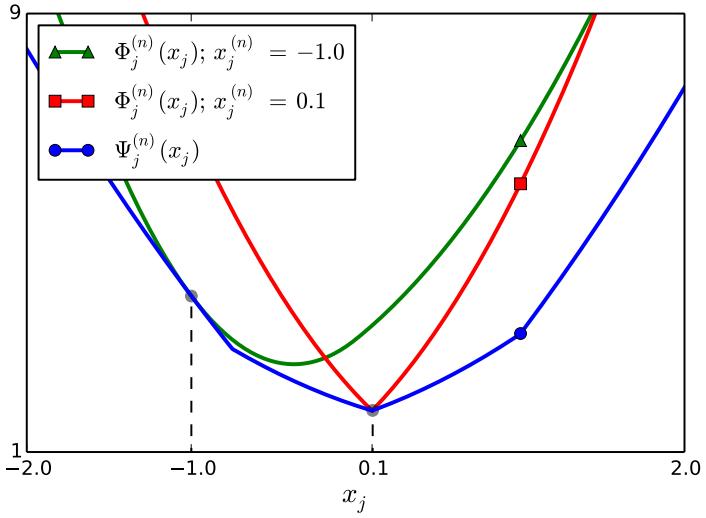


Figure 4.3: An example of the pixel-update cost function $\Psi_j^{(n)}$ with three neighbors and the absolute value potential function. The majorizer $\Phi_j^{(n)}$ described in Section 4.3.2.1 is drawn at two points: the suboptimal point $x_j^{(n)} = -1.0$ and the optimum $x_j^{(n)} = 0.1$. In both cases, $\Omega = [-3, 3]$.

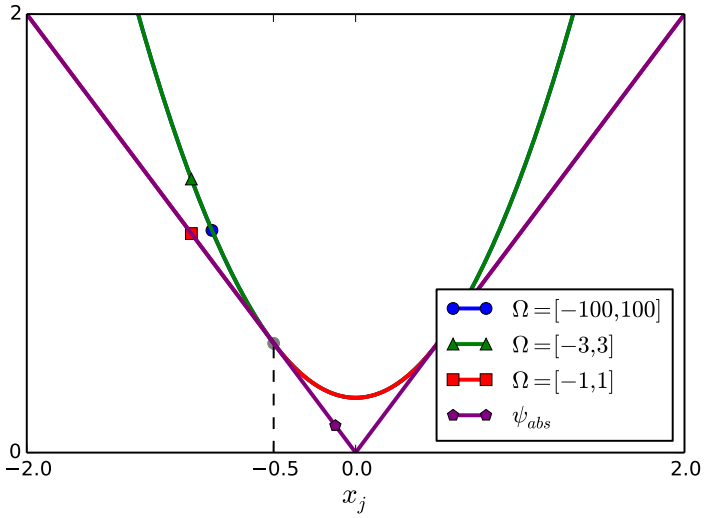
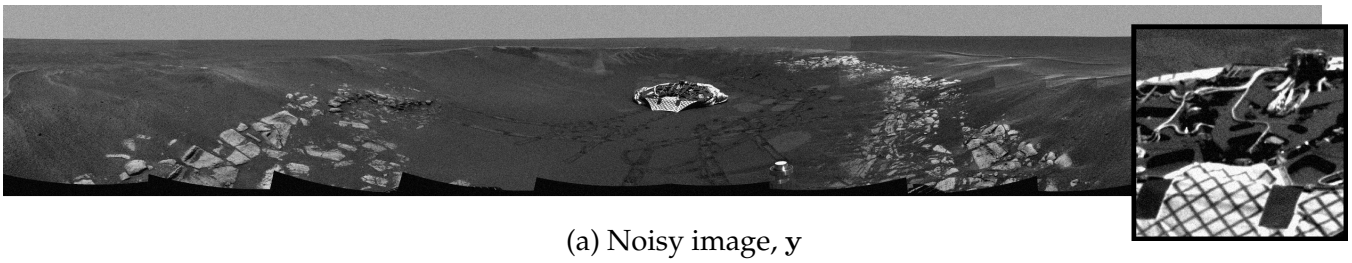
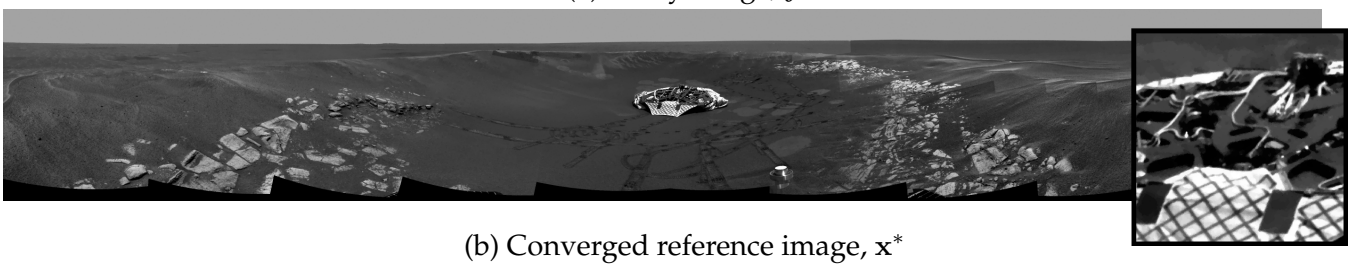


Figure 4.4: The absolute value potential function and the majorizer $\phi_\Omega^{(n)}(x_j)$ described in Section 4.3.2.1 with $x_j^{(n)} = -0.5$. Enlarging the domain Ω "loosens" the majorizer.



(a) Noisy image, $\mathbf{y}$



(b) Converged reference image, $\mathbf{x}^*$

Figure 4.5: Initial noisy and converged reference images from the TV denoising experiment in Section 4.4.1. The original image is an approximately 75-megapixel composite of pictures taken by NASA's Mars Opportunity Rover; the insets are 512×512-pixel subimages.

4.4 Experiments

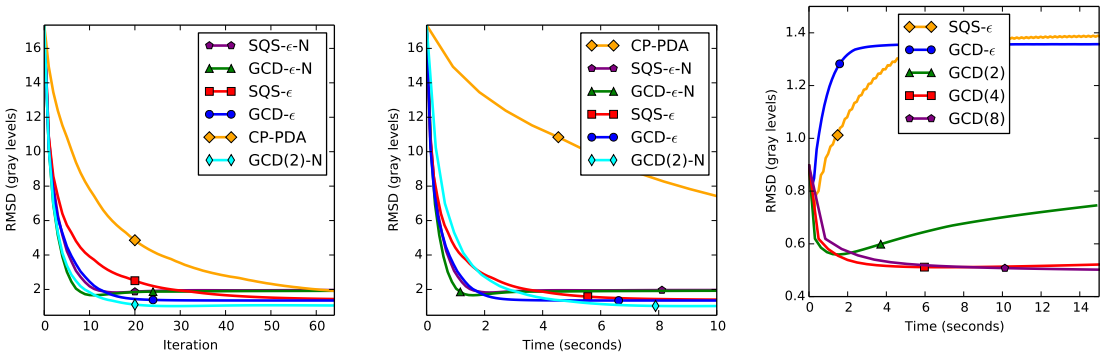
This section presents two experiments using the TV regularizer (Section 4.4.1) and a differentiable edge-preserving regularizer used in CT reconstruction (Section 4.4.2). All the algorithms in the following experiments were run on an NVIDIA Tesla C2050 GPU with 3 GB of memory and implemented in OpenCL.

In addition to the algorithms described above, we applied Nesterov's first-order acceleration [60] to the GCD algorithm after each loop through all the groups. Future research may establish the theoretical convergence properties of these accelerated algorithms, and they appear to be stable.

4.4.1 Anisotropic TV denoising

In 2004, the Mars Opportunity rover transmitted photographs of its landing site in the "Eagle Crater" back to Earth. Scientists at NASA/JPL combined these photographs into a 22,780 $\times$ 3,301-pixel (approximately 75 megapixel) grayscale image [9]. Pixels were represented by floating-point numbers between 0 and 255; storing each copy of the image required approximately 300 MB of memory.

We corrupted the composite image with additive white Gaussian noise with standard



(a) RMSD to $\mathbf{x}^*$ by iteration (b) RMSD to $\mathbf{x}^*$ by time (c) RMSD to $\mathbf{x}^*$ by time ($\mathbf{x}^{(0)}$ near $\mathbf{x}^*$)

Figure 4.6: Root-mean-squared-difference to the converged reference image $\mathbf{x}^*$ by iteration and time for the total variation denoising experiment in Section 4.4.1.

deviation $\sigma = 20$ gray levels (see Figure 4.5a). Then we denoised the corrupted image by solving the iterative denoising problem (4.3) with anisotropic total variation ($\psi = \psi_{\text{abs}}$) using all eight adjacent pixels ($|\mathcal{N}_j| = 8$), empirically selected regularizer weight $\beta = 7$, uniform weights ($\mathbf{W} = \mathbf{I}$, $\kappa_{jl} = 1$), and the constraint $x_j \in [0, 255]$. Figure 4.5b shows an effectively converged reference image, $\mathbf{x}^*$. All the algorithms in this section are initialized from the noisy data, $\mathbf{x}^{(0)} = \mathbf{y}$.

We ran the Chambolle-Pock primal-dual algorithm (CP-PDA) (Algorithm 2 in [10], adapted to anisotropic TV), the separable quadratic surrogates [1] (SQS-ϵ) algorithm with the "capped-curvature" corner-rounding approximation and the proposed GCD algorithm with the same corner-rounding approximation (GCD-ϵ). We also applied Nesterov's first-order acceleration to SQS (SQS-ϵ-N) and corner-rounded GCD (GCD-ϵ-N). Finally, we ran GCD with two inner iterations of the proposed duality-based majorizer and Nesterov's first-order acceleration (GCD(2)-N). In all cases, we chose $\epsilon = 2$. Figure 4.6 plots cost function and root mean-square difference (RMSD) to the reference image against algorithm iteration and time.

The Chambolle-Pock primal-dual algorithm converged rapidly in terms of iteration, but considerably more slowly as a function of time. This behavior, which is hidden when experiments are performed with small images, is a consequence of PDA's high memory requirements. Even on the NVIDIA Tesla with 3GB of memory, we could not store all the algorithm's variables (including the regularizer and data-fit weights) on the GPU at once. Consequently we needed to occasionally transfer memory between RAM and the

GPU, which slowed down PDA's convergence speed with respect to time. Because the PDA uses $|\mathcal{N}_j|$ image-sized dual variables, this memory burden would be even greater for a 3D denoising problem. At least with modern GPU hardware, algorithms with lower memory requirements like SQS-ϵ and the GCD algorithms seem more appropriate than PDA for large problems.

The SQS algorithm can be viewed as a one-group GCD algorithm, where surrogate functions are used to decouple the image update into a set of one-dimensional updates. In that light, the major differences between the SQS and GCD algorithms are pixel update order and majorizer looseness, and both of these differences appear to be advantages for GCD.

Although both the SQS-ϵ and GCD-ϵ algorithms in this experiment perform a corner-rounding approximation, GCD-ϵ's pixel update order appears to make it more robust to the error introduced by that approximation. This can be seen in the more accurate limit cycles reached by the GCD-ϵ algorithms compared to the respective SQS-ϵ algorithms. The GCD algorithms also do not need to majorize to produce one-dimensional subproblems; this makes GCD-ϵ's one-dimensional surrogate $\Phi_j^{(n)}$ "tighter" than the corresponding one-dimensional surrogate produced by SQS. This increases the step sizes that the GCDs algorithm take, as seen by GCD-ϵ reaching its limit cycle more rapidly than SQS-ϵ.

Unlike the SQS algorithms, the proposed GCD algorithm can achieve more accurate solutions by performing more iterations of the inner MM algorithm. This allows GCD(2)-N to rapidly achieve a more accurate solution than the corner-rounding algorithms.

4.4.1.1 Late-iteration behavior and multiple MM steps

To further explore the effect of the number of inner MM iterations on algorithm convergence, we also initialized GCD with

$$\mathbf{x}^{(0)} = \mathbf{x}^* + \mathbf{w}, \qquad \mathbf{w} \sim \mathrm{N}(\mathbf{0}, \mathbf{I}), \tag{4.31}$$

a point near the reference image. We ran GCD with up to 1, 2, 4 and 8 inner MM iterations. Each algorithm was terminated early if possible using the monotone-cost stopping criteria (4.30). Figure 4.6c plots RMSD to $\mathbf{x}^*$ against time for each configuration.

This experiment reveals two important things. First, unsurprisingly, increasing the maximum number of inner MM iterations allows the GCD algorithms to converge to a solution closer to $\mathbf{x}^*$. In all cases, the GCD algorithms produced a more accurate solution

than SQS-ϵ, including GCD-ϵ, which "corner-rounds" in a similar way. Second, while more inner iterations requires more time per outer iteration, algorithms with more inner iterations may converge more quickly in time than those with fewer. The markers in Figure 4.6c were all placed at the 12th iteration. Although GCD(4) took nearly half as long per iteration as GCD(8), the eight-inner-iterations algorithm converged roughly as quickly in time and to a more accurate limit cycle.

4.4.2 X-ray CT denoising

In diagnostic X-ray CT reconstruction, differentiable convex potential functions are often preferred to the absolute value potential function [83]. One choice of potential function is the q-generalized Gaussian (qGG),

$$\psi(t) = \frac{\frac{1}{2}|t|^p}{1 + |t/\delta|^{p-q}}. \tag{4.32}$$

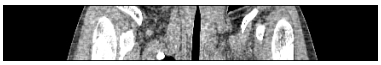
The qGG potential function is both convex and differentiable for appropriate choice of p, q and $\delta > 0$.

While CT reconstruction involves solving a more general regularized least-squares problem, variable splitting and alternating minimization methods can produce algorithms that handle the system physics and edge-preserving regularizer in separate subproblems. In some memory-conservative variable splitting approaches [57] or majorize-minimize algorithms using separable quadratic surrogates [1, 42], the regularizer appears in a denoising problem like (4.3).

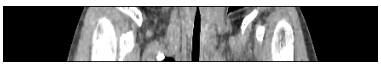
In this experiment we solved a denoising problem that could arise from a variable splitting X-ray CT reconstruction algorithm. The data came from a $512 \times 512 \times 65$-pixel helical shoulder image provided by GE Healthcare. Pixels were represented between 0 and 2,600 modified Hounsfield units (HU). We used the qGG potential function (with $q = 2$, $p = 1.2$ and $\delta = 10$ HU) and nonuniform regularizer weights typical of helical CT reconstruction [81]. The regularizer penalized all adjacent 3D neigbhors, i.e., $|\mathcal{N}_j| = 26$. We set the diagonal weight matrix $\mathbf{W}$ to

$$\mathbf{W} = \operatorname*{diag}_j \left\{ \frac{[\mathbf{A}'\mathbf{S}\mathbf{A}]_{jj}}{2} \right\}, \tag{4.33}$$

where $\mathbf{A}$ is the so-called CT system matrix and $\mathbf{S}$ contains the statistical weights of the

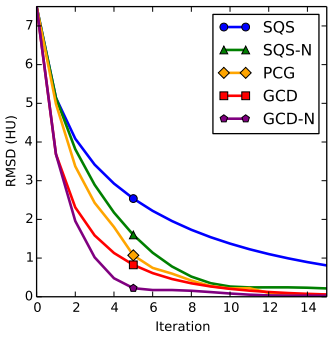


Noisy image, $\mathbf{y} = \mathbf{x}_{\text{FBP}}$

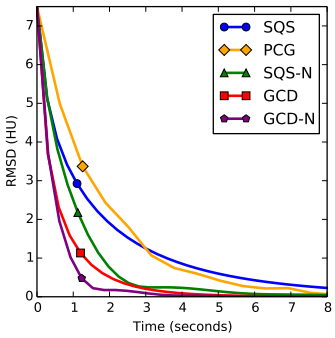


Converged reference image, $\mathbf{x}^*$

(a) Center slice of intial and converged images



(b) RMSD to $\mathbf{x}^*$ by iteration



(c) RMSD to $\mathbf{x}^*$ by time

Figure 4.7: Results from the X-ray CT denoising problem. Figure 4.7a displays the center slices of the initial noisy filtered backprojection image and the converged reference. Both are displayed on a 800 - 1200 modified Hounsfield unit (HU) scale.

measurements [83].

We initialized each algorithm with $\mathbf{x}^{(0)} = \mathbf{x}_{\text{FBP}}$, the output of the classical analytical filtered backprojection (FBP) algorithm. To include second-order methods like preconditioned conjugate gradients in our comparison, we dropped the conventional nonnegativity constraint used in X-ray CT. Figure 4.7a illustrates the center slice of $\mathbf{x}_{\text{FBP}}$ and an effectively converged reference image, $\mathbf{x}^*$.

We solved the denoising problem with the proposed GCD algorithm, the separable quadratic surrogate algorithm (SQS), and preconditioned conjugate gradients (PCG) using a diagonal preconditioner. We also ran GCD and SQS with Nesterov's first-order acceleration (GCD-N and SQS-N). Figures 4.7b and 4.7c plot the progress of each algorithm towards $\mathbf{x}^*$ as a function of iteration and time, respectively.

Preconditioned conjugate gradients converged quickly per iteration but comparably to SQS by time. The high computational cost of PCG on the GPU is caused by the algorithm's inner products and multiple inner steps; the diagonal preconditioner added negligible computational cost. Inner products are classically considered to be computationally cheap operations, but on the GPU and for this family of denoising problems, they are a considerable computational burden. The algorithms that perform only local mem-

ory accesses (SQS and GCD) and their accelerated variants converged significantly more quickly by wall time. Of these, GCD and GCD-N converged the fastest.

4.5 Conclusions

The trend in modern computing hardware is towards increased parallelism instead of better serial performance. This chapter presented image denoising algorithms for edge-preserving regularization that play to the strengths of GPUs, the exemplar of this parallelism trend. By avoiding operations like inner products or complex preconditioners and minimizing memory usage, the proposed GCD algorithms provide impressive convergence rates. The additional increase in performance provided by Nesterov's first-order acceleration is exciting, and further work is needed to characterize the theoretical behavior of the accelerated algorithms. This chapter focuses on gray scale images, but the general approach is extensible to color images and video.

CHAPTER 5

Duality and Tomography Problems

This chapter treats the following quadratic tomography optimization problem:

$$\widehat{\mathbf{x}} = \underset{\mathbf{x}}{\operatorname{argmin}} \ \frac{1}{2}||\mathbf{A}\mathbf{x} - \mathbf{y}||^2_{\mathbf{\Lambda_u}} + \frac{1}{2}||\mathbf{x} - \mathbf{z}||^2_{\mathbf{\Lambda_v}}. \tag{5.1}$$

where $\mathbf{A}$ is the X-ray CT system matrix. Problems of this sort can appear in majorize-minimize algorithms and variable-splitting algorithms [1, 42, 57, 63–66, 74], in addition to being a rudimentary form of model-based image reconstruction. We present a duality-based approach to solve this problem that bears similarities to ordered subsets (OS). Unlike OS, however, the proposed algorithm is convergent, even with an arbitrarily large number of subsets. We combine the proposed duality approach with the denoising algorithm in Chapter 4 for a fast X-ray CT reconstruction algorithm.

5.1 Introduction

Variable splitting-based X-ray CT reconstruction algorithms often have as an inner step a quadratic minimization problem involving the CT system matrix, $\mathbf{A}$. For illustration, consider an algorithm with a single "$\mathbf{v} = \mathbf{x}$" split:

$$\mathsf{L}(\mathbf{x}) = \frac{1}{2}||\mathbf{A}\mathbf{x} - \mathbf{y}||^2_{\mathbf{W}} \tag{5.2}$$

$$\widehat{\mathbf{x}} = \underset{\mathbf{x}}{\operatorname{argmin}} \ \min_{\mathbf{v} \in \Omega} \mathsf{L}(\mathbf{x}) + \mathsf{R}(\mathbf{v}) \qquad \text{such that } \mathbf{v} = \mathbf{x}, \tag{5.3}$$

This chapter is partially based on [53].

with edge-preserving regularizer R. Applying ADMM to this constrained optimization problem yields the following set of iterated updates:

$$\mathbf{x}^{(n+1)} = \underset{\mathbf{x}}{\operatorname{argmin}} \left\{ \mathsf{J}_{\mathbf{x}}^{(n)}(\mathbf{x}) = \mathsf{L}(\mathbf{x}) + \frac{1}{2}\left|\left|\mathbf{x} - \mathbf{v}^{(n)} + \boldsymbol{\eta_v}^{(n)}\right|\right|_{\boldsymbol{\Lambda}_\mathbf{v}}^2 \right\} \tag{5.4}$$

$$\mathbf{v}^{(n+1)} = \underset{\mathbf{v}\in\Omega}{\operatorname{argmin}}\ \mathsf{R}(\mathbf{v}) + \frac{1}{2}\left|\left|\mathbf{v} - \mathbf{x}^{(n+1)} + \boldsymbol{\eta_v}^{(n+1)}\right|\right|_{\boldsymbol{\Lambda}_\mathbf{v}}^2 \tag{5.5}$$

$$\boldsymbol{\eta_v}^{(n+1)} = \boldsymbol{\eta_v}^{(n)} + \mathbf{x}^{(n+1)} - \mathbf{v}^{(n+1)}. \tag{5.6}$$

In this case, the $\mathbf{x}$ update (5.4) is the quadratic "tomography subproblem." Similar subproblems appear in many other variable splitting and majorize-minimize algorithms [1, 42, 57, 63–66, 74].

5.2 The tomography subproblem

In theory the tomography subproblem (5.4) should not be difficult to solve: it is quadratic, unconstrained, and the system matrix $\mathbf{A}$ is sparse. Conventionally, one would think that a preconditioned gradient-based method would be effective. In two dimensional CT reconstruction, this is indeed the case [74]. But in three dimensions, accurate and efficient approximations to $\mathbf{A}'\mathbf{A}$ (let alone $\mathbf{A}'\mathbf{W}\mathbf{A}$) are difficult to find.

Regardless of how difficult it is to find a good preconditioner for the tomography problem, conventional gradient-based methods (intuitively) need to compute a gradient of the cost function $\mathsf{J}_{\mathbf{x}}^{(n)}$ every iteration:

$$\nabla \mathsf{J}_{\mathbf{x}}^{(n)}(\mathbf{x}) = \mathbf{A}'\mathbf{W}(\mathbf{A}\mathbf{x} - \mathbf{y}) + \boldsymbol{\Lambda}_\mathbf{v}\left(\mathbf{x} - \mathbf{v}^{(n)} + \boldsymbol{\eta_v}^{(n)}\right). \tag{5.7}$$

This requires a full forward ($\mathbf{A}\cdot$) and back ($\mathbf{A}'\cdot$) projection. For a reasonably-sized shoulder scan, this operation takes around two minutes on an Intel Xeon E7-8860 with a well-optimized multi-threaded `C` implementation and 12 threads. This is a disproportionally large cost compared to computing the regularizer gradient.

Ordered subsets (OS) approximates the tomographic portion of the gradient by using only a subset of the data and system matrix. This approximation enables the algorithm to update $\mathbf{x}$ while considering only a subset of the data and system matrix, thereby cutting down on computational costs. (Unfortunately, OS is an approximation and, outside of an incremental gradient scheme or without some form of relaxation, does not converge to

the true solution). The method we present below is similar to ordered subsets in this way: the image $\mathbf{x}$ is updated by visiting only a subset of the data and system matrix. Unlike OS, however, the proposed approach relies on no approximations and is convergent.

5.2.1 The duality trick

Instead of directly minimizing $\mathsf{J}_{\mathbf{x}}^{(n)}$ (5.4) with a preconditioned gradient-based method, we use a duality-based approach. First, rewrite $\mathsf{J}_{\mathbf{x}}^{(n)}$ implicitly in terms of the sinogram-sized variable $\mathbf{u}$:

$$\mathsf{J}_{\mathbf{x}}^{(n)}(\mathbf{x}) = \sup_{\mathbf{u}} \left\{ \mathsf{S}^{(n)}(\mathbf{x}, \mathbf{u}) = (\mathbf{A}\mathbf{x} - \mathbf{y})'\mathbf{W}\mathbf{u} + \frac{1}{2}\left|\left|\mathbf{x} - \mathbf{v}^{(n)} + {\boldsymbol{\eta}_{\boldsymbol{v}}}^{(n)}\right|\right|_{\boldsymbol{\Lambda}_{\mathbf{v}}}^2 - \frac{1}{2}||\mathbf{u}||_{\mathbf{W}}^2 \right\}. \tag{5.8}$$

This may be a surprising transformation! If we solve the inner maximization of $\mathsf{S}^{(n)}(\mathbf{x}, \mathbf{u})$ over $\mathbf{u}$:

$$\nabla_{\mathbf{u}} \mathsf{S}^{(n)}(\mathbf{x}, \mathbf{u}) = \mathbf{W}(\mathbf{A}\mathbf{x} - \mathbf{y})_{\mathbf{W}}^2 - \mathbf{W}\mathbf{u} = \mathbf{0}, \tag{5.9}$$

$$\mathbf{u}(\mathbf{x}) = \mathbf{A}\mathbf{x} - \mathbf{y}, \tag{5.10}$$

$$\mathsf{S}^{(n)}(\mathbf{x}, \mathbf{u}(\mathbf{x})) = (\mathbf{A}\mathbf{x} - \mathbf{y})'\mathbf{W}(\mathbf{A}\mathbf{x} - \mathbf{y}) - \frac{1}{2}||\mathbf{A}\mathbf{x} - \mathbf{y}||_{\mathbf{W}}^2 + \frac{1}{2}\left|\left|\mathbf{x} - \mathbf{v}^{(n)} + {\boldsymbol{\eta}_{\boldsymbol{v}}}^{(n)}\right|\right|_{\boldsymbol{\Lambda}_{\mathbf{v}}}^2 \tag{5.11}$$

$$= \mathsf{J}_{\mathbf{x}}^{(n)}(\mathbf{x}), \tag{5.12}$$

so the two forms are equivalent. Solving the tomography subproblem (i.e., minimizing $\mathsf{J}_{\mathbf{x}}^{(n)}$) is equivalent to finding the saddle point:

$$\mathbf{x}^{(n+1)} = \operatorname*{argmin}_{\mathbf{x}} \max_{\mathbf{u}} \mathsf{S}^{(n)}(\mathbf{x}, \mathbf{u}), \tag{5.13}$$

where solving the interior maximization first (and rewriting $\mathbf{u}$ in terms of $\mathbf{x}$) yields the original minimization over $\mathsf{J}_{\mathbf{x}}^{(n)}$.

Note that S is convex in $\mathbf{x}$, concave in $\mathbf{u}$, and continuous in both. In fact (5.13) can be written as an example of Fenchel duality (see Section 2.2.3.2) with the dual variable of the tomography term $\mathbf{p}$ scaled as $\mathbf{u} = \mathbf{W}\mathbf{p}$. We invoke Fenchel's duality theorem to reverse the order of minimization and maximization:

$$\min_{\mathbf{x}} \max_{\mathbf{u}} \mathsf{S}^{(n)}(\mathbf{x}, \mathbf{u}) = \max_{\mathbf{u}} \min_{\mathbf{x}} \mathsf{S}^{(n)}(\mathbf{x}, \mathbf{u}). \tag{5.14}$$

Peforming the now interior maximization over $\mathbf{x}$,

$$\mathbf{x}(\mathbf{u}) = \underset{\mathbf{x}}{\operatorname{argmin}}\ \mathsf{S}(\mathbf{x}, \mathbf{u}), \tag{5.15}$$

$$= \mathbf{v}^{(n)} - \boldsymbol{\eta_v}^{(n)} + \boldsymbol{\Lambda}_{\mathbf{v}}^{-1}\mathbf{A}'\mathbf{W}\mathbf{u}, \tag{5.16}$$

and plugging back in to the right-hand side of (5.14) yields the dual problem:

$$\mathbf{u}^{(n+1)} = \underset{\mathbf{u}}{\operatorname{argmax}}\ \left\{\mathsf{D}^{(n)}(\mathbf{u}) = \mathsf{S}^{(n)}(\mathbf{x}(\mathbf{u}), \mathbf{u})\right\} \tag{5.17}$$

$$= \underset{\mathbf{u}}{\operatorname{argmax}}\ -\frac{1}{2}\mathbf{u}'\big(\mathbf{W} + \mathbf{W}\mathbf{A}\boldsymbol{\Lambda}_v^{-1}\mathbf{A}'\mathbf{W}\big)\mathbf{u} + \mathbf{u}'\mathbf{W}\big(\mathbf{y} - \mathbf{A}\big(\mathbf{v}^{(n)} - \boldsymbol{\eta_v}^{(n)}\big)\big). \tag{5.18}$$

After finding an (approximate) solution to the dual problem (5.17), we find the resulting update $\mathbf{x}^{(n+1)}(\mathbf{u})$ using (5.16).

5.2.2 Solving the dual problem

The dual problem for the tomography subproblem (5.17) is an effectively unconstrained quadratic maximization problem over the dual variable $\mathbf{u}$ and involving the CT system matrix $\mathbf{A}$. At first glance, it may seem that we have made things more difficult. The dual problem must be solved with some iterative, probably gradient-based method, and the gradients of D are sinogram-sized, much larger than the image-sized gradients of $\mathsf{J}_{\mathbf{x}}$.

However, recall that our goal is not necessarily to solve the dual problem. That is, if $\widehat{\mathbf{u}}^{(n+1)}$ is the exact solution to (5.17), our goal is *not* to find $\mathbf{u}^{(n+1)}$ such that

$$\left|\left|\mathbf{u}^{(n+1)} - \widehat{\mathbf{u}}^{(n+1)}\right|\right|^2 \tag{5.19}$$

is small. Instead, our goal is to find $\mathbf{u}^{(n+1)}$ such that $\mathbf{x}^{(n+1)}\big(\mathbf{u}^{(n+1)}\big)$ (via (5.16)) is a good approximation to $\widehat{\mathbf{x}}^{(n+1)}$, the exact solution to the tomography problem. In other words, we want

$$\left|\left|\mathbf{x}\big(\mathbf{u}^{(n+1)}\big) - \widehat{\mathbf{x}}^{(n+1)}\right|\right|^2 = \left|\left|\boldsymbol{\Lambda}_{\mathbf{v}}^{-1}\mathbf{A}'\mathbf{W}\big(\mathbf{u}^{(n+1)} - \widehat{\mathbf{u}}^{(n+1)}\big)\right|\right|^2 \tag{5.20}$$

to be small. Since we're interested only in the convergence of the backprojection of $\mathbf{u}^{(n)}$ to the solution of the dual problem, we can update only a subset of the variables in $\mathbf{u}^{(n)}$ at a time. Doing so requires only a partial projection and backprojection, but can update

many of the pixels in the image.

5.2.3 View-by-view minorize-maximize algorithm

The method we used in [53] updates one view of $\mathbf{u}$ at a time. While this may result in a relatively modest convergence rate for $\mathbf{u} \to \widehat{\mathbf{u}}^{(n+1)}$, updating each view of $\mathbf{u}$ updates $\mathbf{x}(\mathbf{u})$. Experimental results indicate that this algorithm quickly finds an accurate approximation to the tomography subproblem.

We group the entries of $\mathbf{u}$ into blocks according to view: $\mathbf{u} = \begin{bmatrix}\mathbf{u}_1 \cdots \mathbf{u}_{N_\beta}\end{bmatrix}$. The gradient of $\mathsf{D}^{(n)}$ with respect to the βth block is

$$\nabla_{\mathbf{u}_\beta} \mathsf{D}^{(n)}(\mathbf{u}) = -\big(\mathbf{W}_\beta + \mathbf{W}_\beta \mathbf{A}_\beta \boldsymbol{\Lambda}_{\mathbf{v}}^{-1} \mathbf{A}_\beta{}' \mathbf{W}_\beta\big)\mathbf{u} + \mathbf{W}_\beta\big(\mathbf{y}_\beta + \mathbf{A}_\beta\big(\mathbf{v}^{(n)} - \boldsymbol{\eta}_{\boldsymbol{v}}{}^{(n)}\big)\big). \tag{5.21}$$

In the image domain, computing the gradient of the datafit term with respect to a subset of the pixels requires a loop over many views. On the other hand, (5.21) indicates that computing the gradient with respect to a subset of the rays in $\mathbf{u}$ involves only the corresponding rays in the CT system model. This is a significant decrease in computational cost.

Given the group gradient (5.21), the natural next step is to find a step size $\alpha_\beta^{(n)}$. Because the dual problem is quadratic, we may be tempted to find the optimal step size:

$$\alpha_\beta^{(n)} = \underset{\alpha}{\operatorname{argmax}}\ \mathsf{D}^{(n)}\Big(\mathbf{u}^{(n)} + \alpha_\beta^{(n)} \nabla_{\mathbf{u}_\beta} \mathsf{D}^{(n)}\big(\mathbf{u}^{(n)}\big)\Big). \tag{5.22}$$

Solving (5.22) amounts to performing one inner product in the image domain and one inner product of a view-sized vector. At first glance, this seems inexpensive, but recall that we are performing this operation for each view in the system matrix. Many helical scans have upwards of 7,000 views, and at that scale even relatively inexpensive image-sized inner products become impractical.

Instead of solving for the exact step size, we use a diagonal majorizer for the Hessian of $\mathsf{D}^{(n)}$ with respect to $\mathbf{u}_\beta$ [1]:

$$\mathbf{H}_\beta = \mathbf{W}_\beta + \mathbf{W}_\beta \mathbf{M}_\beta \mathbf{W}_\beta, \tag{5.23}$$

where

$$\mathbf{M}_\beta = \operatorname*{diag}_i \left\{ \left[\mathbf{A}_\beta \boldsymbol{\Lambda}_\mathbf{v}^{-1} \mathbf{A}_\beta{}' \mathbf{1} \right]_i \right\} \tag{5.24}$$

$$\succeq \mathbf{A}_\beta \boldsymbol{\Lambda}_\mathbf{v}^{-1} \mathbf{A}'_\beta. \tag{5.25}$$

Because the diagonal matrix $\boldsymbol{\Lambda}_\mathbf{v}$ is nominally independent of the data, the majorizers $\mathbf{M}_\beta$ can be computed offline and stored.[1] Since each $\mathbf{M}_\beta$ majorizes only a subset of the Hessian of the dual function $\mathsf{D}^{(n)}$ (because we are optimizing at each step over only a subset of the entries of $\mathbf{u}$), it does not appear that using this MM algorithm results in unacceptably small step sizes.

Figure 5.1 gives pseudocode for this view-by-view minorize-maximize algorithm. Processing $\mathbf{u}_\beta$ requires forward projecting and backprojecting the βth view and performing a few scalar operations. Consequently, running a loop through all the views in the system matrix requires a comparable amount of computation to performing a full forward and backprojection.

5.2.4 Update groups and order

The pseudocode in Figure 5.1 states to loop over the views "in a suitable order." In Section 5.4.1 we test this algorithm with several view orders:

- Sequential order, i.e., essentially in the order that the views were acquired by the scanner. This means that we visit views in the "most correlated" order.
- Bit-reversal order, or FFT order, which visits views in a deterministic order such that the views are spaced very far apart. This order can be constructed by representing all the view indices in binary, zero-padded to the next power of two above N_{view}, and sorted in reverse bit order.
- Random order, with the views selected uniformly and randomly without replacement.

Our experiments indicate that bit-reversal ordering and random ordering of the views appear to result in faster convergence than sequential ordering. This agrees with results

[1] In fact, for some system models, e.g., the separable footprint system model, [49], the $\mathbf{M}_\beta$ can be computed on-the-fly with each view with only marginal additional cost [42].

- Precompute denominators $\mathbf{m}_\beta = \mathbf{A}_\beta \boldsymbol{\Lambda}_\mathbf{v}^{-1} \mathbf{A}_\beta{}' \mathbf{1}_\beta$
- At (outer) iteration zero, $\mathbf{u}^{(0)} = \mathbf{0}$, $\boldsymbol{\gamma} = \mathbf{0}$
- Set $\mathbf{z}^{(n)} = \mathbf{v}^{(n)} - \boldsymbol{\eta_v}^{(n)}$.
- Loop over view β in a suitable order:

$$\begin{aligned}
\text{Project view} \qquad & \mathbf{d}_\beta^{(n+1)} = \mathbf{A}_\beta \left(\mathbf{z}^{(n)} + \boldsymbol{\Lambda}_\mathbf{v}^{-1} \boldsymbol{\gamma}^{(n)} \right) \\
\text{Compute update} \qquad & \mathbf{r}_\beta^{(n+1)} = \frac{\mathbf{u}_\beta^{(n)} - \mathbf{y}_\beta + \mathbf{d}_\beta^{(n+1)}}{\mathbf{1}_\beta + \mathbf{m}_\beta \odot \mathbf{w}_\beta} \\
\text{Update view} \qquad & \mathbf{u}_\beta^{(n+1)} = \mathbf{u}_\beta^{(n)} - \mathbf{r}_\beta^{(n+1)} \\
\text{Update backprojection} \qquad & \boldsymbol{\gamma} \leftarrow \boldsymbol{\gamma} + \mathbf{A}_\beta{}' \left(\mathbf{w}_\beta \odot \mathbf{r}_\beta^{(n+1)} \right)
\end{aligned}$$

- Output $\mathbf{x}^{(n+1)} = \mathbf{z}^{(n)} - \boldsymbol{\Lambda}_\mathbf{v}^{-1} \boldsymbol{\gamma}$

Figure 5.1: Pseudocode for the view-by-view duality-based algorithm to solve the tomography subproblem

from the algorithmic reconstruction technique (ART) literature [30,31], another sinogram-centric approach to tomographic reconstruction, that suggest updating an image with rays that are uncorrelated or nearly orthogonal yields faster convergence than updating more correlated sequentially.

5.2.4.1 View batches

In our experiments, we update only one view of $\mathbf{u}$ at a time. This limits the parallelizability of our algorithm, and there are several ways to instead update multiple projection view-sized groups of $\mathbf{u}$ simultaneously.

If our data is a helical scan with sufficiently many turns or was acquired by a detector with sufficiently few rows, some of the projection views may be nonoverlapping. That is, the rays in some projection views pass through disjoint sets of pixels. These disjoint views can be simultaneously updated without increasing the size of the diagonal majorizer $\mathbf{M}_\beta$.

Axial data sets or helical data sets with smaller coverage or wider detectors have fewer (or no) disjoint views. In these cases one can still update multiple views of $\mathbf{u}$ at once, but the entries of the diagonal majorizer $\mathbf{M}_\beta$ will grow, making the updates to each view of $\mathbf{u}$

smaller. Let $\mathbf{u}_g$ be a group of projection views. Precompute the diagonal majorizer

$$\mathbf{M}_g = \operatorname*{diag}_i \left\{ \left[\mathbf{A}_g \boldsymbol{\Lambda}_{\mathbf{v}}^{-1} \mathbf{A}_g' \mathbf{1} \right]_i \right\}, \tag{5.26}$$

where $\mathbf{A}_g$ contains the corresponding rows of $\mathbf{A}$. Because $\mathbf{u}_g$ contains views with overlapping rays, the elements of $\mathbf{M}_g$ will be larger than in the original one-view groups. The rest of the algorithm is identical to Figure 5.1, with single-view projections and backprojections replaced by multiple-view ones.

5.2.4.2 Smaller-than-view groups

Instead of updating multiple view-sized groups of $\mathbf{u}$ at once, we can instead update groups of $\mathbf{u}$ that contain subsets of projection views. This may be useful for CPU implementations, because conventionally CPUs can run fewer threads efficiently than GPUs. An interesting approach is to select subsets of a single projection view; e.g., have $\mathbf{u}_g$ contain every nth channel of a projection view. For small n, the backprojection of $\mathbf{u}_g$ may still update the same set of pixels of $\mathbf{x}$ as the group containing all the channels in that view. Because neighboring channels are "highly correlated" in the same way that neighboring views are, this approach may result in acceleration (due to computing fewer rays in the projection and backprojection steps) for architectures with less parallelism.

5.3 Reconstruction algorithm with tomography trick

If we use a duality-based algorithm to solve the tomography subproblem, we no longer really need to store $\mathbf{x}^{(n)}$, because $\mathbf{x}^{(n)}$ is determined entirely by $\mathbf{u}^{(n)}$ (5.16). Plugging (5.16) into the ADMM iterates (5.4)-(5.6) and simplifying yields the following equivalent alternative form:

$$\mathbf{u}^{(n+1)} = \operatorname*{argmin}_{\mathbf{u}} \ \frac{1}{2} \left|\left| \mathbf{u} - \mathbf{u}^{(n)} \right|\right|^2_{\mathbf{W}\mathbf{A}\boldsymbol{\Lambda}_{\mathbf{v}}^{-1}\mathbf{A}'\mathbf{W}} - \mathbf{u}'\mathbf{W}\left(\mathbf{A}\left(\mathbf{v}^{(n)} + \mathbf{v}^{(n)} - \mathbf{v}^{(n-1)}\right)\right) + \frac{1}{2}||\mathbf{u} - \mathbf{y}||^2_{\mathbf{W}} \tag{5.27}$$

$$\mathbf{v}^{(n+1)} = \operatorname*{argmin}_{\mathbf{v} \in \Omega} \ \frac{1}{2} \left|\left| \mathbf{v} - \mathbf{v}^{(n)} \right|\right|^2_{\boldsymbol{\Lambda}_{\mathbf{v}}} + \left(\mathbf{v} - \mathbf{v}^{(n)}\right)' \mathbf{A}' \mathbf{W} \mathbf{u}^{(n+1)} + \mathsf{R}(\mathbf{v}). \tag{5.28}$$

The $\boldsymbol{\eta}_v$ dual variable has disappeared into the $\mathbf{v}^{(n)} - \mathbf{v}^{(n-1)}$ "momentum" term in the $\mathbf{u}$-update (5.27).

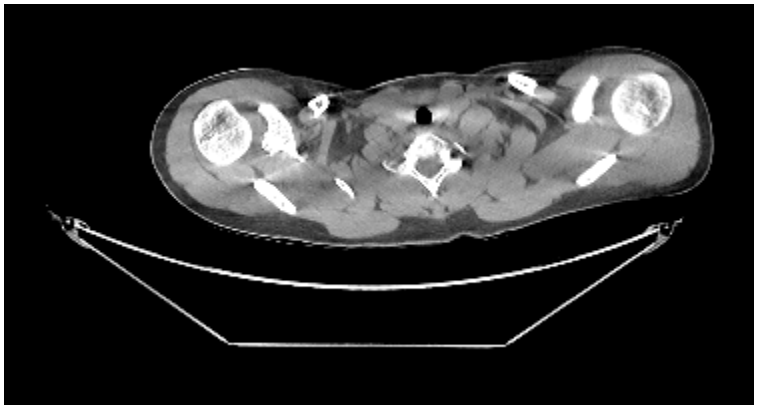


Figure 5.2: Center slice of the converged reference image for the Tikhonov-regularized example in Section 5.4.1, displayed on a $[800, 1200]$ modified HU window

There are two "proximal terms" in the new update subproblems, (5.27)-(5.28), and both of them involve the term $\mathbf{\Lambda_v}$. The term $\left|\left|\mathbf{u}-\mathbf{u}^{(n)}\right|\right|^2_{\mathbf{WA\Lambda_v^{-1}A'W}}$ in (5.27) suggests that selecting large $\mathbf{\Lambda_v}$ (and thus small $\mathbf{\Lambda_v^{-1}}$) would result in relatively rapid convergence for the $\mathbf{u}$ term. On the other hand, the $\left|\left|\mathbf{v}-\mathbf{v}^{(n)}\right|\right|^2_{\mathbf{\Lambda_v}}$ term in (5.28) suggests that a small $\mathbf{\Lambda_v}$ (and thus relatively large $\mathbf{\Lambda_v^{-1}}$) will allow the $\mathbf{v}$ variable to rapidly evolve. Further analysis based on these observations may help select a (possibly adaptive) scheme for setting $\mathbf{\Lambda_v}$, which is a common challenge for variable-splitting methods.

5.4 Experiments

This section presents three experiments: one that isolates the tomography subproblem, and two full reconstruction experiments. All the algorithms in this section were implemented in `OpenCL` with a lightweight `C` and `Python` wrapper. All experiments were run on an NIVIDA GTX 480 GPU with approximately 2.5 GB of RAM.

5.4.1 Tikhonov-regularized CT reconstruction

The tomography subproblem (5.4) is essentially a Tikhonov-regularized reconstruction problem. To examine the effect of different view access orders on the performance of the duality-based minorize-maximize algorithm in Figure 5.1, we solved a Tikhonov-

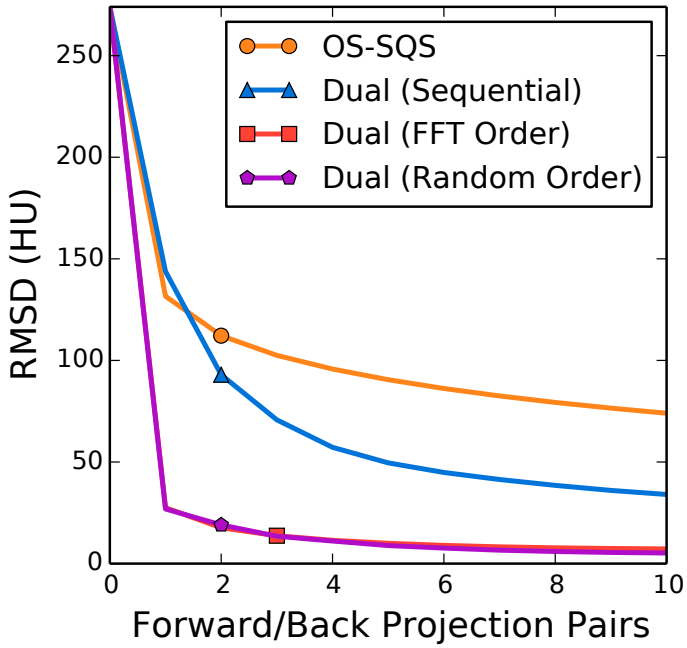


Figure 5.3: RMSD to converged reference for several algorithms in the Tikhonov-regularized CT reconstruction problem in Section 5.4.1

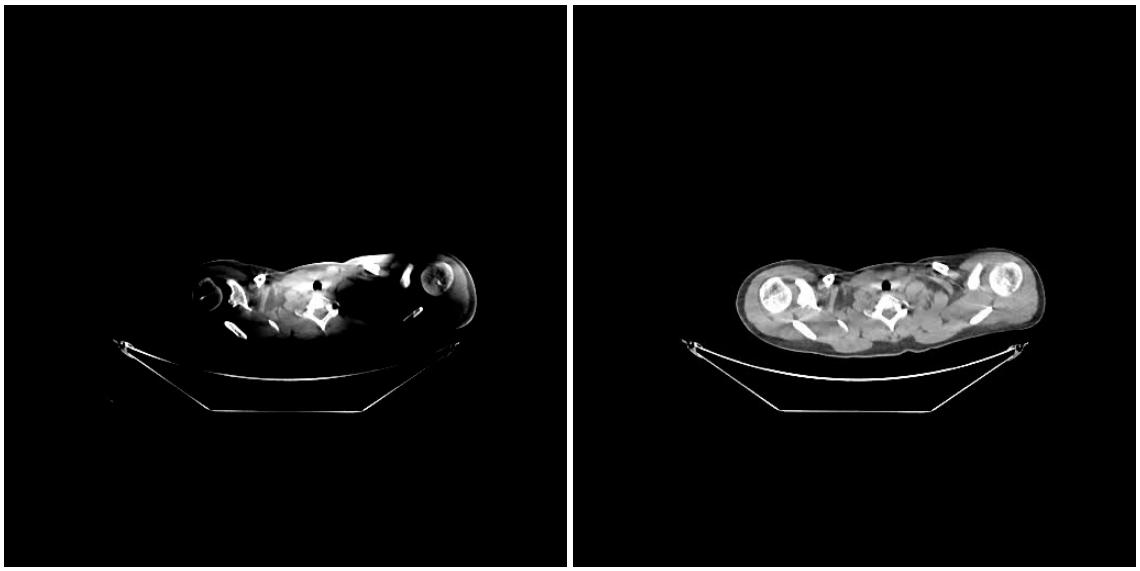
regularized reconstruction problem,

$$\widehat{\mathbf{x}} = \underset{\mathbf{x}}{\operatorname{argmin}} \ \frac{1}{2}||\mathbf{A}\mathbf{x} - \mathbf{y}||_{\mathbf{W}}^2 + \frac{\mu}{2}||\mathbf{x}||^2, \tag{5.29}$$

with small μ, for a helical shoulder scan dataset. The center slice of the converged reference image is given in Figure 5.2. All algorithms were initialized from $\mathbf{x}^{(0)} = \mathbf{0}$.

We ran 10 iterations of ordered subsets with separable quadratic surrogates (OS-SQS) [1] with 12 subsets. The more recent OS-momentum algorithm [43] was highly unstable with this number of subsets, and we excluded it from the graph. We also ran the proposed algorithm (Dual) with three different view-visit orders:

- sequential order, which accesses the views in same order that they were acquired;
- bit-reversal order, which accesses the views in a deterministic but highly non-correlated order; and
- random order, which accesses the views in random order.



Sequential order Random order

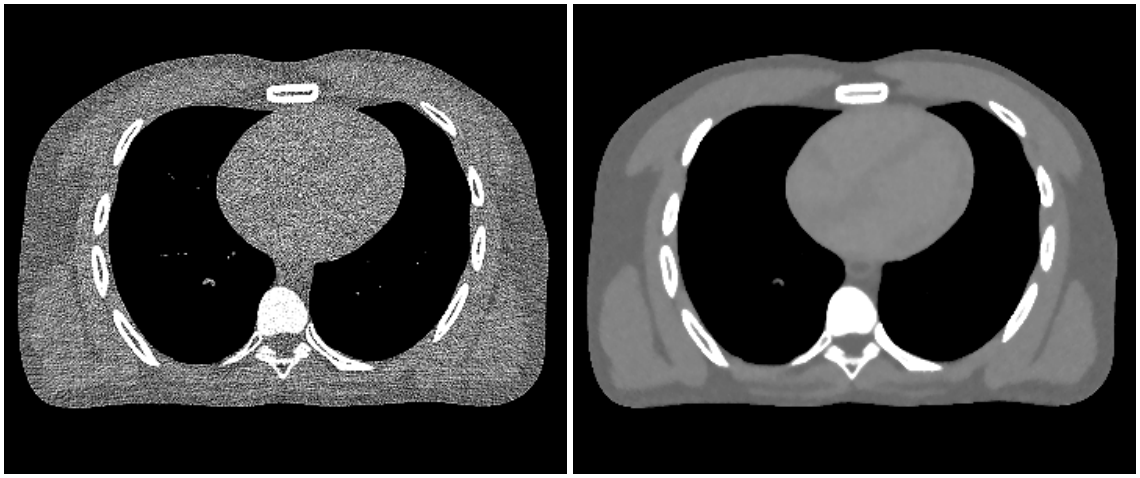
Figure 5.4: Center slice of Tikhonov-regularized reconstruction using the duality-based algorithm with sequential and random access orders after one full forward and backprojection; displayed on a [800, 1200] modified HU window.

Figure 5.3 illustrates the root-mean-squared-difference (RMSD) to the converged reference as a function of the number of full forward and backprojections each algorithm performed. The duality-based algorithms converged most rapidly, and algorithms that visited views in "uncorrelated" orders converged more rapidly than the algorithm with sequential ordering. The center slice of the volume for the dual algorithm with sequential and random order is illustrated in Figure 5.4. After only one iteration, the dual algorithm with random access order produces a surprisingly high-quality image.

5.4.2 Axial XCAT reconstruction

We implemented the ADMM algorithm with the duality-based solver described in Section 5.3. The penalty parameter $\mathbf{\Lambda_v} = \mu\mathbf{I}$ was tuned coarsely by hand. We approximately solved the dual problem by performing one iteration through all the views using proposed dual MM algorithm in bit-reversal (FFT) order. We ran eight iterations of the group coordinate descent denoising algorithm described in [52, 54] and Chapter 4 to approximately implement the $\mathbf{v}$ update (5.28).

We instantiated a $1024 \times 1024 \times 192$-voxel XCAT phantom and simulated projections onto the GE Lightspeed scanner [86] with 888 channels, 64 rows and 984 views. We cor-



Intial Reference

Figure 5.5: Center slice of initial image and converged reference for axial XCAT phantom reconstruction in Section 5.4.2

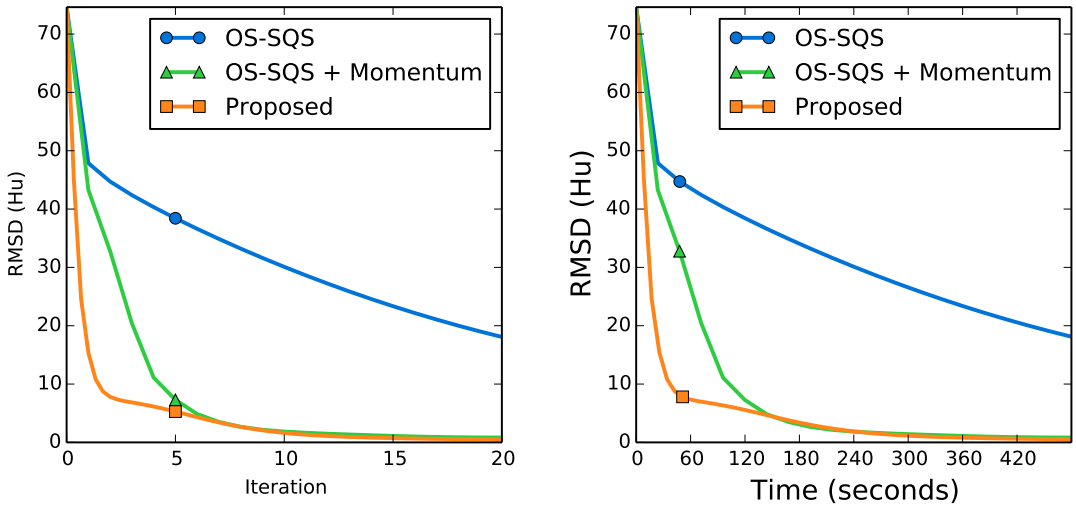
rupted the data with simulated Poisson noise and reconstructed onto a $512 \times 512 \times 96$-pixel grid. The edge-preserving regularizer penalized all 26 three-dimensional neighbors with the differentiable Fair potential. Figure 5.5 illustrates the center slice of the initial filtered backprojection image and the converged reference we used in our experiments.

The proposed algorithm is compared to the ordered subsets algorithm (OS-SQS) [1] and the momentum-accelerated ordered subsets algorithm (OS-SQS + Momentum). Both ordered subsets algorithms used 12 subsets.

Figure 5.6 illustrates RMSD vs. time and iteration plots for the three algorithms. The similarity between the two plots indicates the low overhead from the duality-based algorithm and the denoising subproblem. The proposed algorithm also converges more rapidly in early iterations than the OS-SQS + Momentum and, unlike the ordered subsets algorithms, will converge to the true solution $\widehat{\mathbf{x}}$.

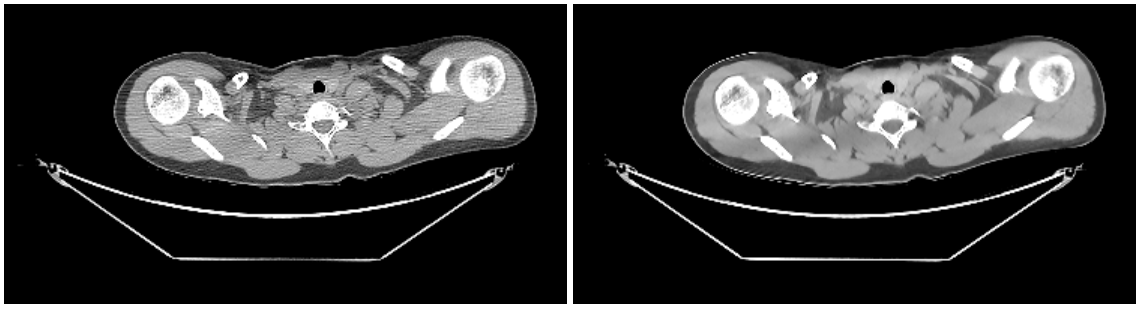
5.4.3 Helical shoulder reconstruction

We performed a similar experiment using patient data from GE Healthcare. This dataset was a $512 \times 512 \times 101$-pixel reconstruction of a helical shoulder scan. The system geometry had only 32 rows (instead of 64 in Section 5.4.2) and 7,146 views. Figure 5.7 illustrates the center slice of the initial filtered backprojection image and the converged reference. The edge-preserving regularizer penalized all 26 neighbors and used the q-generalized Gaussian penalty function.



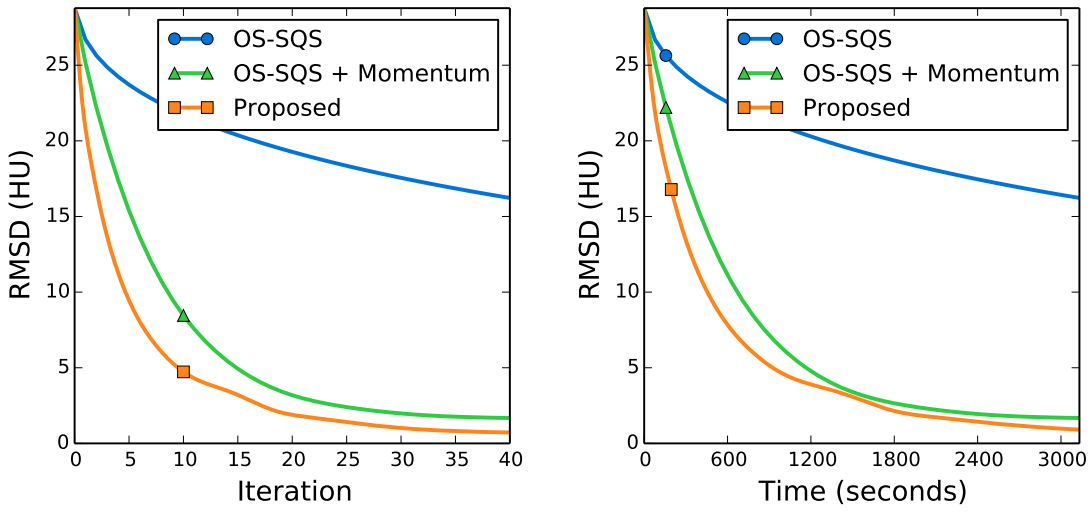
RMSD vs. iteration RMSD vs. time

Figure 5.6: Distance to reference image for several algorithms as a function of iteration and time for the axial XCAT phantom reconstruction in Section 5.4.2, displayed on a [800, 1200] modified HU window



Intial Reference

Figure 5.7: Center slice of initial image and converged reference for helical shoulder reconstruction in Section 5.4.3, displayed on a [800, 1200] modified HU window



RMSD vs. iteration RMSD vs. time

Figure 5.8: Distance to reference image for several algorithms as a function of iteration and time for the helical shoulder reconstruction in Section 5.4.3

We followed the same procedure in Section 5.4.2 and compared the proposed algorithm to OS and OS + Momentum. Both ordered subsets algorithms used 12 subsets. Figure 5.8 illustrates RMSD to the converged reference image by iteration and time.

We draw similar conclusions for this dataset as the previous one. The similarity between the per-iteration and per-time plots indicates that the additional overhead of the proposed ADMM algorithm in relatively small. The proposed algorithm also converges more rapidly than the ordered subsets-based algorithms, and can be seen to continue to converge towards the converged reference even after the OS algorithms have begun to approach their limit cycles.

5.5 Conclusions and future work

This chapter presented a duality-based approach to solving the tomography subproblem that appears in many splitting-based and majorize-minimize algorithms. We tested the algorithm in isolation on a Tikhonov-regularized CT reconstruction problem. The algorithm requires only a small amount of additional computation over ordered subsets-based algorithms, but unlike OS algorithms it is convergent. Most attractively, the proposed algorithm converges very quickly.

In our experiments, we updated the sinogram-sized dual variable $\mathbf{u}$ in projection view-sized groups. Section 5.2.4 lists several other options that we have not experimentally explored. Updating more than one group at a time may allow the algorithm to exploit the high level of parallelism on the GPU. On the other hand, updating a subset of a projection view at a time (e.g., updating every nth channel) may be more appropriate for hardware with less parallel capacity like the CPU. Future work is needed to explore these approaches.

We combined the tomography solver described in this chapter with the denoising algorithm in Chapter 4 using variable splitting and the alternating directions method of multipliers (ADMM) to form a CT reconstruction algorithm. The resulting algorithm converges comparably quickly to the popular OS+Momentum algorithm, but unlike OS-based algorithms will eventually find the true minimizer of the reconstruction cost function.

There are a few disadvantages of the proposed ADMM algorithm. There are a few parameters to tune: in addition to the usual ADMM penalty parameters, the algorithm designer must choose how many iterations to run when solving each subproblem. It is also not clear how to parallelize the proposed algorithm over many GPUs, whereas the multiple-GPU implementation of OS+Momentum in the next chapter is very straightforward.

CHAPTER 6

Fast X-ray CT Reconstruction on the GPU

This chapter proposed a purely duality-based CT reconstruction algorithm. It combines ideas from Chapters 4 and 5 using duality instead variable splitting and the alternating directions method of multipliers (ADMM). This structure allows the algorithm to tightly interleave denoising and tomography updates instead of doing them in chunks, as in the ADMM algorithm in Chapter 5.

We implement the proposed algorithm on single and multiple GPUs. To mitigate the cost of copying large amount of dual variables to and from the GPU, we perform computation-heavy operations while the transfers are occurring. Experiments on real and synthetic datasets indicate the algorithm converges rapidly.

6.1 Introduction

X-ray computed tomography (CT) model-based image reconstruction (MBIR) combines information about system physics, measurement statistics, and prior knowledge about images into a high-dimensional cost function [83]. The variate of this function is an image; the image that minimizes this cost function can contain less noise and fewer artifacts than those produced with conventional analytical techniques, especially at reduced doses [20, 22, 83].

The primary drawback of MBIR methods is how long it takes to find this minimizer. In addition to general optimization algorithms like conjugate gradients with specialized preconditioners [15, 27], a wide range of CT-specialized algorithms have been proposed to accelerate the optimization. One popular approach uses iterated coordinate

This chapter is partially based on [55, 56].

descent (ICD) to sequentially update pixels (or groups of pixels) of the image [25, 90]. ICD faces challenges from stagnating processor clock speeds and increasing parallelization in modern computing hardware. Variable splitting and alternating minimization techniques separate challenging parts of the cost function into more easily solved subproblems [65, 67, 73, 74]. When used with the ordered subsets (OS) approximation [1, 67], these algorithms can converge very rapidly. Unfortunately, without some sort of stabilizing relaxation, OS-based algorithms have uncertain convergence properties and can diverge under certain circumstances. Nonetheless, combining OS with accelerated first-order methods [39, 44] has produced simple algorithms with state of the art convergence speeds.

This paper proposes an algorithm that shares some properties with prior works. Like some variable splitting methods, our proposed algorithm consists of steps that consider parts of the cost function in isolation. Separating jointly challenging parts of the cost function from one another allows us to use specialized and fast solvers for each part. Our algorithm also uses a group coordinate optimization scheme, somewhat like ICD, but the variables it updates are in a dual domain; updating a small group of dual variables can simultaneously update a large number of pixels in the image. Like ordered subsets algorithms, our algorithm does need to visit all the measured data to update the image, but unlike OS algorithms without relaxation, the proposed algorithm has some convergence guarantees.

The next section sets up our MBIR CT reconstruction problem. Section 6.2 introduces the mathematics of the proposed algorithm, and Section 6.3 describes our single- and multiple-GPU implementations. Section 6.4 provides some experimental results and Section 6.5 gives some conclusions and directions for future work.

6.1.1 Model-based image reconstruction

Consider the following X-ray CT reconstruction problem [83]:

$$\widehat{\mathbf{x}} = \underset{\mathbf{x} \geq \mathbf{0}}{\operatorname{argmin}} \; \mathsf{L}(\mathbf{A}\mathbf{x}) + \mathsf{R}(\mathbf{C}\mathbf{x}). \tag{6.1}$$

The CT projection matrix $\mathbf{A} \in \mathbb{R}^{M \times N}$ models system physics and geometry, and the finite differencing matrix $\mathbf{C} \in \mathbb{R}^{K \times N}$ computes the differences between each pixel and its

neighbors. Both L and R are separable sums of convex functions:

$$\mathsf{L}(\mathbf{p}) = \sum_{i=1}^{M} l_i(p_i), \qquad \mathsf{R}(\mathbf{d}) = \sum_{k=1}^{K} r_k(d_k). \tag{6.2}$$

We call L the data-fit term because it penalizes discrepancies between the measured data $\mathbf{y} \in \mathbb{R}^M$ and the CT projection of $\mathbf{x}$. A common choice for L is a weighted sum of quadratic functions; i.e.,

$$l_i(p_i) = \frac{w_i}{2}(p_i - y_i)^2, \tag{6.3}$$

with the weight $w_i > 0$. Traditionally, the weight is the inverse of the variance of the ith measurement, $w_i = 1/\sigma_i^2$.

Similarly, R encourages the reconstructed image $\widehat{\mathbf{x}}$ to be smooth by penalizing the differences between neighboring pixels. R is a weighted sum of often nonquadratically penalized differences,

$$r_k(d_k) = \beta_k \psi(d_k). \tag{6.4}$$

The potential function ψ is convex, even and often coercive. The quadratic penalty function, $\phi(t) = \frac{1}{2}t^2$, while analytically tractable, tends to favor reconstructed images $\widehat{\mathbf{x}}$ with blurry edges because it penalizes large differences between neighboring pixels (i.e., edges) aggressively. Potential functions $\psi(t)$ that have a smaller rate of growth as $|t| \to \infty$ are called edge-preserving because they penalize these large differences less aggressively. Examples include the absolute value function $\phi(t) = |t|$ from total variation (TV) regularization, the Fair potential, and the q-generalized Gaussian. The positive weights $\{\beta_k\}$ are fixed and encourage certain resolution or noise properties in the image [14,81].

Heuristically, the functions L and R have opposing effects on the reconstructed image $\widehat{\mathbf{x}}$: L encourages data fidelity and can lift the noise from the data $\mathbf{y}$ into the reconstructed image, and R encourages smoothness at the cost of producing an image $\mathbf{x}$ that does not fit the measurements as well. While combining L and R into one reconstruction problem allows us to have the "best of both worlds" (i.e., smooth images that fit the noisy measurements), it significantly complicates the task of solving the reconstruction problem (6.1). Without the regularizer R, the reconstruction problem (6.1) could possibly be solved using a fast quadratic solver. Conversely, without the data-fit term L (or using a simpler one not involving the CT system matrix $\mathbf{A}$), (6.1) becomes a denoising problem,

for which many fast algorithms exist.

Variable splitting and alternating minimization provide a framework for separating L and R into different sub-problems [2]. The ADMM algorithm in [74] used a circulant approximation to the CT Gram matrix $\mathbf{A}'\mathbf{A}$ to provide rapid convergence rates for 2D CT problems. Unfortunately, the circulant approximation is less useful in 3D axial and helical CT geometries. We partially overcame these difficulties in [53] by using a duality-based approach to solving problems involving the CT system matrix, but the resulting algorithm still used ADMM, which has difficult-to-tune penalty parameters and relatively high memory use. Gradient-based algorithms like ordered subsets [1] (OS) with acceleration [44] and the linearized augmented Lagrangian method with ordered subsets [67] (OS-LALM), can produce rapid convergence rates but rely on an approximation to the gradient of data-fit term and have uncertain convergence properties. Some of these algorithms require generalizations to handle non-smooth regularizers like total variation.

This paper describes an extension of the algorithms we introduced in [53,56]. The proposed algorithm uses duality, group coordinate ascent with carefully chosen groups, and the majorize-minimize framework to rapidly solve the reconstruction problem (6.1). We extend the work in [56] by also considering the nonnegativity constraint $\mathbf{x} \geq \mathbf{0}$ in (6.1). Our algorithm is designed with the GPU in mind: while it uses a large number of variables, the "working set" for each of the algorithm's steps is small and easily fits in GPU memory. We stream these groups of variables to the GPU and hide the latency of these transfers by performing other, less memory-intensive computations. We show that the proposed algorithm can be implemented on a machine with multiple GPUs for additional acceleration.

6.2 Reconstruction algorithm

At a high level, our algorithm approximately performs the following iteration:

$$\mathbf{x}^{(n+1)} = \underset{\mathbf{x}}{\operatorname{argmin}} \ \mathsf{J}^{(n)}(\mathbf{x}), \qquad \text{where} \tag{6.5}$$

$$\mathsf{J}^{(n)}(\mathbf{x}) = \mathsf{L}(\mathbf{A}(\mathbf{x})) + \mathsf{R}(\mathbf{C}\mathbf{x}) + \mathsf{I}(\mathbf{x}) + \frac{\mu}{2}\left|\left|\mathbf{x} - \mathbf{x}^{(n)}\right|\right|^2, \tag{6.6}$$

with $\mu > 0$. We have expressed the nonnegativity constraint $\mathbf{x} \geq \mathbf{0}$ as the characteristic function I:

$$\mathsf{I}(\mathbf{x}) = \sum_{j=1}^{N} \iota_j(x_j), \qquad \text{where} \tag{6.7}$$

$$\iota_k(x) = \begin{cases} 0, & x \geq 0; \\ \infty, & \text{else.} \end{cases} \tag{6.8}$$

Although ι_k is discontinuous, it is convex.

The function on the right-hand side of (6.5) is as difficult to solve exactly as the original cost function (6.1), and the additional proximal term $\frac{\mu}{2}\left|\left|\mathbf{x} - \mathbf{x}^{(n)}\right|\right|^2$ would slow convergence if (6.5) were performed exactly. However, the proximal term allows us to use the following duality approach that "solves" (6.5) inexactly but quickly.

Let l_i^*, r_k^* and ι_j^* denote the convex conjugates of l_i, r_k and ι_j, respectively, e.g., :

$$l_i^*(u_i) = \sup_{p_i} \; p_i u_i - l_i(p_i). \tag{6.9}$$

Because l_i, r_k and ι_j are convex, they are equal to the convex conjugates of l_i^*, r_k^* and ι_j^*, respectively. We use this biconjugacy property to write

$$l_i(p_i) = \sup_{u_i} \; p_i u_i - l_i^*(u_i). \tag{6.10}$$

By summing over the indices i, j and k, we write L, R and I implicitly as the suprema of sums of one-dimensional dual functions:

$$\mathsf{L}(\mathbf{p}) = \sup_{\mathbf{u}} \; \sum_{i=1}^{M} p_i u_i - l_i^*(u_i) = \sup_{\mathbf{u}} \; \mathbf{u}'\mathbf{p} - \mathsf{L}^*(\mathbf{u}), \tag{6.11}$$

$$\mathsf{R}(\mathbf{d}) = \sup_{\mathbf{v}} \; \sum_{k=1}^{K} v_k d_k - r_k^*(d_k) = \sup_{\mathbf{v}} \; \mathbf{v}'\mathbf{d} - \mathsf{R}^*(\mathbf{v}), \tag{6.12}$$

$$\mathsf{I}(\mathbf{x}) = \sup_{\mathbf{z}} \; \sum_{j=1}^{N} z_j x_j - \iota_j^*(z_j) = \sup_{\mathbf{z}} \; \mathbf{z}'\mathbf{x} - \mathsf{I}^*(\mathbf{z}). \tag{6.13}$$

With (6.11)-(6.13), we rewrite the update problem (6.5) as

$$\mathbf{x}^{(n+1)} = \underset{\mathbf{x}}{\operatorname{argmin}} \sup_{\mathbf{u},\,\mathbf{v},\,\mathbf{z}} \mathsf{S}^{(n)}(\mathbf{x},\mathbf{u},\mathbf{v},\mathbf{z}), \tag{6.14}$$

$$\begin{aligned}\mathsf{S}^{(n)}(\mathbf{x},\mathbf{u},\mathbf{v},\mathbf{z}) \triangleq \frac{\mu}{2}\left|\left|\mathbf{x}-\mathbf{x}^{(n)}\right|\right|^2 + (\mathbf{A}'\mathbf{u}+\mathbf{C}'\mathbf{v}+\mathbf{z})'\mathbf{x} \\ - \mathsf{L}^*(\mathbf{u}) - \mathsf{R}^*(\mathbf{v}) - \mathsf{I}^*(\mathbf{z}).\end{aligned} \tag{6.15}$$

Reversing the order of minimization and maximization[1] yields:

$$\min_{\mathbf{x}} \sup_{\mathbf{u},\mathbf{v},\mathbf{z}} \mathsf{S}^{(n)}(\mathbf{x},\mathbf{u},\mathbf{v},\mathbf{z}) = \sup_{\mathbf{u},\mathbf{v},\mathbf{z}} \min_{\mathbf{x}} \mathsf{S}^{(n)}(\mathbf{x},\mathbf{u},\mathbf{v},\mathbf{z}). \tag{6.16}$$

The now inner minimization over $\mathbf{x}$ is trivial to perform. We solve for the minimizer $\mathbf{x}$ and write it in terms of the dual variables $\mathbf{u}$, $\mathbf{v}$ and $\mathbf{z}$:

$$\widetilde{\mathbf{x}}^{(n+1)}(\mathbf{u},\mathbf{v},\mathbf{z}) = \mathbf{x}^{(n)} - \frac{1}{\mu}(\mathbf{A}'\mathbf{u}+\mathbf{C}'\mathbf{v}+\mathbf{z}). \tag{6.17}$$

The value of $\mathbf{x}$ induced by the dual variables, $\widetilde{\mathbf{x}}^{(n+1)}(\mathbf{u},\mathbf{v},\mathbf{z})$, minimizes the update cost function (6.5) when $\mathbf{u}$, $\mathbf{v}$, and $\mathbf{z}$ maximize the following dual function[2]:

$$\mathsf{D}^{(n)}(\mathbf{u},\mathbf{v},\mathbf{z}) \triangleq \mathsf{S}^{(n)}\big(\widetilde{\mathbf{x}}^{(n+1)}(\mathbf{u},\mathbf{v},\mathbf{z}),\mathbf{u},\mathbf{v},\mathbf{z}\big) \tag{6.18}$$

$$\begin{aligned}&= -\frac{1}{2\mu}||\mathbf{A}'\mathbf{u}+\mathbf{C}'\mathbf{v}+\mathbf{z}||^2 \\ &\qquad + (\mathbf{A}'\mathbf{u}+\mathbf{C}'\mathbf{v}+\mathbf{z})'\mathbf{x}^{(n)} \\ &\qquad - \mathsf{L}^*(\mathbf{u}) - \mathsf{R}^*(\mathbf{v}) - \mathsf{I}^*(\mathbf{z}).\end{aligned} \tag{6.19}$$

Solving (6.19) induces a value of $\mathbf{x}$ (6.17) that minimizes the original update problem (6.5). We tackle this optimization problem using a stochastic group coordinate ascent algorithm described in the next section. To accelerate convergence, we propose to maximize the dual function $\mathsf{D}^{(n)}$ approximately. Under conditions similar to other alternating minimization algorithms like ADMM [18], the proposed algorithm is convergent even with these approximate updates; see Appendix E.

At a high level, our proposed algorithm iteratively performs the following steps:

[1] See Appendix C.
[2] See Appendix D.

1. form the dual function $\mathsf{D}^{(n)}$ using $\mathbf{x}^{(n)}$,

2. find $\mathbf{u}^{(n+1)}$, $\mathbf{v}^{(n+1)}$ and $\mathbf{z}^{(n+1)}$ by running iterations of the stochastic group coordinate ascent (SGCA) algorithm detailed in the following sections:

$$\mathbf{u}^{(n+1)}, \mathbf{v}^{(n+1)}, \mathbf{z}^{(n+1)} \approx \underset{\mathbf{u},\mathbf{v},\mathbf{z}}{\operatorname{argmax}} \ \mathsf{D}^{(n)}(\mathbf{u}, \mathbf{v}, \mathbf{z}), \tag{6.20}$$

3. and update $\mathbf{x}^{(n+1)}$:

$$\mathbf{x}^{(n+1)} = \widetilde{\mathbf{x}}^{(n+1)}\big(\mathbf{u}^{(n+1)}, \mathbf{v}^{(n+1)}, \mathbf{z}^{(n+1)}\big). \tag{6.21}$$

6.2.1 Stochastic group coordinate ascent

We propose to use a stochastic group coordinate ascent (SGCA) algorithm to perform the dual maximization (6.20). The algorithm iteratively selects a group of variables (in our case, a set of the elements of the dual variables $\mathbf{u}$, $\mathbf{v}$ and $\mathbf{z}$) via a random process and updates them to increase the value of the dual function $\mathsf{D}^{(n)}$. Because SGCA is convergent [79], enough iterations of the algorithm in this section will produce dual solutions $\mathbf{u}^{(n+1)}$, $\mathbf{v}^{(n+1)}$ and $\mathbf{z}^{(n+1)}$ that are arbitrarily close to true maximizers $\widehat{\mathbf{u}}^{(n+1)}$, $\widehat{\mathbf{v}}^{(n+1)}$ and $\widehat{\mathbf{z}}^{(n+1)}$. However, the solution accuracy that we really care about is how well the induced image $\mathbf{x}^{(n+1)} = \widetilde{\mathbf{x}}^{(n+1)}\big(\mathbf{u}^{(n+1)}, \mathbf{v}^{(n+1)}, \mathbf{z}^{(n+1)}\big)$ approximates exact minimizer of (6.5). The data-fit and regularizer dual variables $\mathbf{u}$ and $\mathbf{v}$ affect the induced image $\widetilde{\mathbf{x}}^{(n+1)}$, per (6.17), through the linear operators $\mathbf{A}'$ and $\mathbf{C}'$ respectively. These linear operators propagate the influence of a possibly small group of dual variables to many pixels: e.g., the elements of $\mathbf{u}$ corresponding to a single projection view are backprojected over a large portion of the image. Consequently, performing a relatively small number dual group updates can significantly improve the image $\widetilde{\mathbf{x}}^{(n+1)}(\mathbf{u}, \mathbf{v}, \mathbf{z})$.

An SGCA algorithm updates one group of variables at a time. We can form these groups arbitrarily, and as long each group is visited "often enough" the algorithm converges to a solution [79]. To exploit the structure of $\mathsf{D}^{(n)}$, we choose each group so that it contains elements from only $\mathbf{u}$, $\mathbf{v}$ or $\mathbf{z}$; i.e., no group contains elements from different variables. Sections 6.2.2, 6.2.3, and 6.2.4 describe the structures of and updates for each of these groups.

Because our SGCA algorithm updates elements of the dual variables in random order, conventional iteration notation becomes cumbersome. Instead, mirroring the algorithm's

implementation, we describe the updates as occurring "in-place." For example, if $\mathbf{u}_\beta$ is the set of elements of $\mathbf{u}$ corresponding to projection view β, then $\mathbf{u}_\beta \leftarrow \mathbf{u}_\beta^+$ means that we replace the contents of $\mathbf{u}_\beta$ in memory with $\mathbf{u}_\beta^+$. To refer to the "current" value of a dual variable in an update problem, we use a superscripted minus; e.g., $\mathbf{u}_\beta^-$. It is convenient to rewrite the quadratic and linear terms in $\mathsf{D}^{(n)}$ using this notation and (6.17) by rewriting the quadratic and linear terms in (6.19):

$$\begin{aligned}\mathsf{D}^{(n)}(\mathbf{u},\mathbf{v},\mathbf{z}) = -\frac{1}{2\mu}\Big|\Big|\mathbf{A}'\big(\mathbf{u}-\mathbf{u}^-\big) \\ + \mathbf{C}'\big(\mathbf{v}-\mathbf{v}^-\big)+\mathbf{z}-\mathbf{z}^-\Big|\Big|^2 \\ + (\mathbf{A}'\mathbf{u}+\mathbf{C}'\mathbf{v}+\mathbf{z})'\widetilde{\mathbf{x}} \\ - \mathsf{L}^*(\mathbf{u}) - \mathsf{R}^*(\mathbf{v}) - \mathsf{I}^*(\mathbf{z}),\end{aligned} \tag{6.22}$$

where the buffer $\widetilde{\mathbf{x}} = \widetilde{\mathbf{x}}^{(n+1)}(\mathbf{u}^-,\mathbf{v}^-,\mathbf{z}^-)$. After updating a group of dual variables, e.g., $\mathbf{u} \leftarrow \mathbf{u}^+$, we update $\widetilde{\mathbf{x}}$ to reflect the update (6.17). The following sections detail these dual variable updates.

6.2.2 Tomography (u) updates

Consider optimizing (6.22) with respect to some subset of the elements of $\mathbf{u}$,

$$\mathbf{u}_g^+ = \underset{\mathbf{u}_g}{\operatorname{argmax}}\ \mathsf{D}^{(n)}\big(\mathbf{u},\mathbf{v}^-,\mathbf{z}^-\big) \tag{6.23}$$

$$= \underset{\mathbf{u}_g}{\operatorname{argmax}}\ -\frac{1}{2\mu}\big|\big|\mathbf{u}_g-\mathbf{u}_g^-\big|\big|^2_{\mathbf{A}_g\mathbf{A}_g'} - \mathsf{L}_g^*(\mathbf{u}_g) + \mathbf{u}_g{}'\mathbf{A}_g\widetilde{\mathbf{x}}, \tag{6.24}$$

where $\mathbf{u}_g$ is a subset of the elements of $\mathbf{u}$. Each member of this group corresponds to a one-dimensional function in the original data-fit term (6.2) and also to a row of the projection matrix $\mathbf{A}$. The elements of $\mathbf{u}_g$ are coupled together in (6.24) by the matrix $\mathbf{A}_g\mathbf{A}_g{}'$, where $\mathbf{A}_g$ contains the rows of $\mathbf{A}$ corresponding to the group $\mathbf{u}_g$.

If $\mathbf{A}_g\mathbf{A}_g{}'$ were diagonal, i.e., if the rays corresponding to elements of $\mathbf{u}_g$ were nonoverlapping, then solving (6.24) would be trivial. (Of course this is also the case when $\mathbf{u}_g$ is a single element of $\mathbf{u}$). However, updating $\mathbf{u}_g$ using only nonoverlapping rays would limit the algorithm's parallelizability. Existing CT projector software may also not be well-optimized for computing the projection and backprojection of individual rays instead of e.g., a projection view at a time. If we allow $\mathbf{u}_g$ to contain overlapping rays, then the

coupling induced by $\mathbf{A}_g\mathbf{A}_g{}'$ makes (6.24) expensive to solve exactly. Instead of using a computationally expensive iterative method to find the exact solution to (6.24), we use a *min*orize-*max*imize technique [35, 48] to find an approximate solution that still increases the dual function $\mathsf{D}^{(n)}$.

Let the diagonal matrix $\mathbf{M}_g$ majorize $\mathbf{A}_g\mathbf{A}_g{}'$, i.e., the matrix $\mathbf{M_g} - \mathbf{A}_g\mathbf{A}_g{}'$ has no negative eigenvalues. Solving the following *separable* problem produces an updated $\mathbf{u}_g^+$ that increases the dual function $\mathsf{D}^{(n)}$:

$$\mathbf{u}_g^+ = \underset{\mathbf{u}_g}{\operatorname{argmax}} \; -\frac{1}{2\mu}\big|\big|\mathbf{u}_g - \mathbf{u}_g^-\big|\big|_{\mathbf{M}_g}^2 - \mathsf{L}_g^*(\mathbf{u}_g) + \mathbf{u}_g{}'\mathbf{A}_g\widetilde{\mathbf{x}}. \tag{6.25}$$

In the common case that $\mathsf{L}(\mathbf{Ax}) = \frac{1}{2}||\mathbf{Ax} - \mathbf{y}||_{\mathbf{W}}^2$, i.e., $l_i(p_i) = \frac{w_i}{2}(p_i - y_i)^2$, the dual L_g^* is

$$\mathsf{L}_g^*(\mathbf{u}_g) = \frac{1}{2}||\mathbf{u}_g||_{\mathbf{W}_g^{-1}}^2 + \mathbf{u}_g{}'\mathbf{y}_g \tag{6.26}$$

and the solution to (6.25) is

$$\mathbf{u}_g^+ = (\mathbf{W}_g\mathbf{M}_g + \mu\mathbf{I})^{-1}\mathbf{W}_g\big(\mu(\mathbf{A}_g\widetilde{\mathbf{x}} - \mathbf{y}_g) + \mathbf{M}_g\mathbf{u}_g^-\big). \tag{6.27}$$

It is computationally challenging to find an "optimal" diagonal majorizing matrix $\mathbf{M}_g \succeq \mathbf{A}_g\mathbf{A}_g{}'$, but the following matrix majorizes $\mathbf{A}_g\mathbf{A}_g{}'$ and is easy to compute [1]:

$$\mathbf{M}_g = \operatorname*{diag}_i\big\{[\mathbf{A}_g\mathbf{A}_g{}'\mathbf{1}]_i\big\}. \tag{6.28}$$

We precompute $\mathbf{M}_g$ for all groups g. This choice of $\mathbf{M}_g$ depends only on the system geometry through $\mathbf{A}_g$ and not on any patient-specific data. If the groups are selected in a regular way, the $\{\mathbf{M}_g\}$ will be very similar between groups and could be precomputed.

In our experiments, we used one group per view, and storing the diagonals of all the majorizers $\{\mathbf{M}_g\}$ took the same amount of memory as the noisy projections $\mathbf{y}$.

After updating the group $\mathbf{u}_g$ (6.27), we "backproject" the update into $\widetilde{\mathbf{x}}$ (6.17):

$$\widetilde{\mathbf{x}} \leftarrow \widetilde{\mathbf{x}}^- - \frac{1}{\mu}\mathbf{A}_g{}'\big(\mathbf{u}_g^+ - \mathbf{u}_g^-\big). \tag{6.29}$$

Altogether, updating $\mathbf{u}_g$ and $\widetilde{\mathbf{x}}$ requires a forward projection and backprojection for the rays in group g and a few vector operations (6.27).

6.2.3 Denoising ($\mathbf{v}$) updates

The regularizer R penalizes the differences between each pixel and its neighbors along a predetermined set of N_r directions chosen by the algorithm designer. Common choices for CT reconstruction are the three axis-aligned directions and the thirteen directions corresponding to all 3D neighbors of a voxel. The finite differencing matrix $\mathbf{C} \in \mathbb{R}^{K \times N}$ computes these differences, and each of the $K = N \cdot N_\mathrm{r}$ elements of the dual variable $\mathbf{v}$ is associated with one of these differences.

The dual vector $\mathbf{v}$ is enormous: in our experiments, $\mathbf{v}$ is as large as thirteen images. Storing a significant fraction of $\mathbf{v}$ on the GPU is impractical, so we want to update the image $\widetilde{\mathbf{x}}$ using a group of elements $\mathbf{v}_g$. To make that update efficient, we would like the group update problem,

$$\mathbf{v}_g^+ = \underset{\mathbf{v}_g}{\operatorname{argmax}} \; -\frac{1}{2\mu}\left|\left|\mathbf{v}_g - \mathbf{v}_g^-\right|\right|_{\mathbf{CC}'}^2 - \mathsf{R}_g^*(\mathbf{v}_g) + \mathbf{v}_g{}'\mathbf{C}_g\widetilde{\mathbf{x}}. \tag{6.30}$$

to decouple into set of independent one-dimensional update problems.

6.2.3.1 Group design

The elements of $\mathbf{v}_g$ are coupled in (6.30) only by the matrix $\mathbf{C}_g\mathbf{C}_g{}'$. This matrix is banded and sparse: it couples *differences* together that involve shared pixels. This coupling is very local; Figure 6.1 illustrates groups that contain only uncoupled elements of $\mathbf{v}$. Updating each of these groups of differences has a "denoising" effect on almost all the pixels (up to edge conditions) in the image and involves solving a set of independent one-dimensional subproblems.

There are many ways to form groups these "covering but not overlapping" groups of differences, but in our implementation we use the following simple "half-direction" groups. Every element of $\mathbf{v}$ can be uniquely represented by a pixel location $\mathbf{i} = (i_x, i_y, i_z)$ and an offset $\mathbf{o} = (o_x, o_y, o_z)$. The difference that v_k represents is between the pixels located at (i_x, i_y, i_z) and $(i_x + o_x, i_y + o_y, i_z + o_z)$. The elements of $\mathbf{v}$ corresponding to a single direction all share the same offset and differ only in their pixel locations.

For each difference direction $r = 1, \ \ldots, \ N_\mathrm{r}$, we form two groups, $\mathbf{v}_{r,e}$ and $\mathbf{v}_{r,o}$. We assign every other difference "along the direction r" to each group. For example, if r indicates vertical differences along the y axis then we assign to $\mathbf{v}_{r,e}$ differences with even i_y and those with odd i_y to $\mathbf{v}_{r,o}$. In Figure 6.1, the cyan group $\{v_9, v_{10}, v_{11}, v_{15}, v_{16}, v_{17}\}$ and

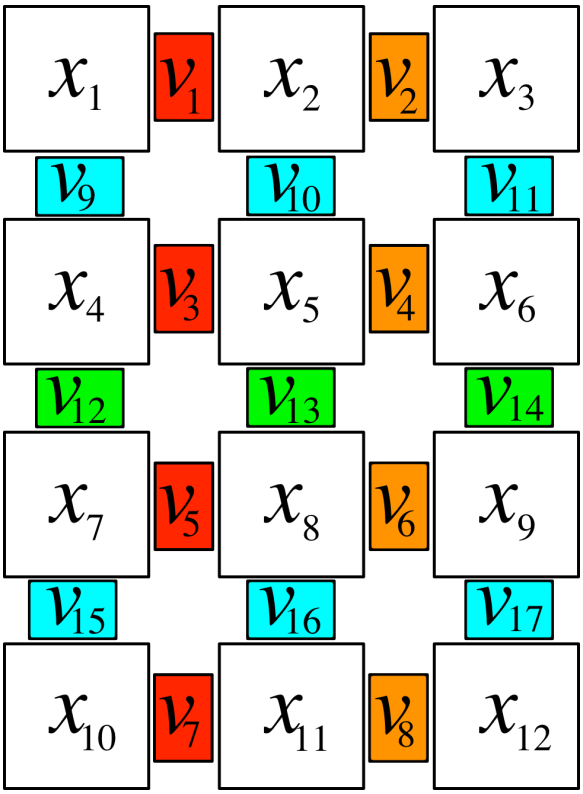


Figure 6.1: Illustration of groups of elements of the dual variable $\mathbf{v}$ for a two-dimensional denoising case. Elements of $\mathbf{v}$ are updated in groups such that none of the groups affect overlapping pixels. For examples, the horizontal differences $\{v_1, v_3, v_5, v_7\}$ are one group and $\{v_2, v_4, v_6, v_8\}$ are another.

the green group $\{v_{12}, v_{13}, v_{14}\}$ partition the vertical differences in this way.

More generally, let $\mathbf{o}_r = (o_x, o_y, o_z)$ be the offset corresponding to direction r. Let $\mathbf{c}_r \in \{0,1\}^3$ contain a single "1" in the coordinate corresponding to the first nonzero element of $\mathbf{o}_r$. For example,

$$\mathbf{o}_r = (0, 1, -1) \rightarrow \mathbf{c}_r = (0, 1, 0), \tag{6.31}$$

$$\mathbf{o}_r = (0, 0, 1) \rightarrow \mathbf{c}_r = (0, 0, 1). \tag{6.32}$$

Recall that $\mathbf{i}_k$ is the location associated with the difference v_k. We assign to $\mathbf{v}_{r,e}$ those differences v_k along the direction r such that $\mathbf{i}_k{}'\mathbf{o}_d$ is even.

6.2.3.2 One-dimensional subproblems

Assuming that $\mathbf{v}_g$ has been chosen so that its elements are uncoupled by $\mathbf{C}_g\mathbf{C}_g{}'$, the group update problem (6.30) decomposes into a set of one-dimensional difference update problems:

$$v_k^+ = \underset{v_k}{\text{argmax}} \; -\frac{1}{\mu}(v_k - \gamma)^2 - \beta_k \psi^*\left(\frac{v_k}{\beta_k}\right) \tag{6.33}$$

$$\gamma \triangleq v_k^- + \frac{\mu}{2}[\mathbf{C}\widetilde{\mathbf{x}}]_k, \tag{6.34}$$

where ψ^* is the convex conjugate of the potential function ψ. Some potential functions ψ have convenient convex conjugates that make (6.33) easy to directly solve:

- Absolute value: If $\psi(d) = |d|$, then ψ^* is the characteristic function of $[-1, 1]$. The solution to (6.33) is

$$\mathbf{v}_k^+ = [\gamma]_{[-\beta_k, \beta_k]}. \tag{6.35}$$

  i.e., the projection of γ onto the closed interval $[-\beta_k, \beta_k]$.

- Huber function: If $\psi(d)$ is the Huber function,

$$\psi(d) = \begin{cases} \frac{1}{2}d^2, & |d| \leq \delta \\ \delta\big(|d| - \frac{1}{2}\delta\big), & \text{else.} \end{cases} \tag{6.36}$$

then its dual is

$$\psi^*(v) = \frac{1}{2}v^2 + \iota_{[-\delta,\delta]}(v). \tag{6.37}$$

The solution to (6.33) is

$$\mathbf{v}_k^+ = \left[\frac{2\beta_k\gamma}{2\beta_k + \mu}\right]_{[-\beta_k\delta,\beta_k\delta]}, \tag{6.38}$$

In other cases, ψ^* more difficult to work with analytically. For example, the Fair potential,

$$\psi(d) = \delta^2\left(\left|\frac{d}{\delta}\right| - \log\left(1 + \left|\frac{d}{\delta}\right|\right)\right). \tag{6.39}$$

is easier to work with in the primal domain, where it has a closed-form "shrinkage" operator, than the dual domain. To exploit potential functions with convenient shrinkage operators but inconvenient convex conjugates, we exploit the convexity of ψ^* and invoke biconjugacy:

$$\beta_k\psi^*\left(\frac{v_k}{\beta_k}\right) = \sup_{q_k}\ q_k v_k - \beta_k\psi(q_k). \tag{6.40}$$

Combining (6.40) and (6.33),

$$v_k^+ = \operatorname*{argmax}_{v_k}\ \inf_{q_k}\ -\frac{1}{\mu}(v_k - \gamma)^2 - v_k q_k + \beta_k\psi(q_k). \tag{6.41}$$

By a similar Fenchel duality argument to (6.16), we reverse the "max" and "inf" in (6.41). The resulting expression involves ψ only through its "shrinkage" operator:

$$v_k^+ = \gamma - \frac{\mu}{2}q_k^+, \qquad \text{where} \tag{6.42}$$

$$q_k^+ = \operatorname*{argmin}_{q_k}\ \frac{\mu}{4}\left(q_k - \frac{2}{\mu}\gamma\right)^2 + \beta_k\psi(q_k). \tag{6.43}$$

After updating a group of differences $\mathbf{v}_g$, we update the buffer $\widetilde{\mathbf{x}}$ (6.17):

$$\widetilde{\mathbf{x}} \leftarrow \widetilde{\mathbf{x}}^- - \frac{1}{\mu}\mathbf{C}_g{}'\left(\mathbf{v}_g^+ - \mathbf{v}_g^-\right). \tag{6.44}$$

Because $\mathbf{v}_g$ contains variables corresponding to nonoverlapping differences, each element of $\mathbf{v}_g$ updates two pixels in $\widetilde{\mathbf{x}}$, and each pixel in $\widetilde{\mathbf{x}}$ is updated by only one difference in $\mathbf{v}_g$.

6.2.4 Nonnegativity ($\mathbf{z}$) updates

Updating each element of the image-sized dual variable $\mathbf{z}$ helps enforce the nonnegativity constraint on the corresponding pixel of $\widetilde{\mathbf{x}}$. The dual function $\mathsf{D}^{(n)}$ is separable in the elements of $\mathbf{z}$:

$$\mathbf{z}^+ = \operatorname*{argmax}_{\mathbf{z}} \; -\frac{1}{2\mu}\left|\left|\mathbf{z}-\mathbf{z}^-\right|\right|^2 + \mathbf{z}'\widetilde{\mathbf{x}} - \mathsf{I}^*(\mathbf{z}) \tag{6.45}$$

$$= \sum_k -\frac{1}{2\mu}(z_k-\eta_k)^2 - \iota_k^*(z_k), \qquad \text{where} \tag{6.46}$$

$$\eta_k \stackrel{\triangle}{=} z_k^- + \mu[\widetilde{\mathbf{x}}]_k. \tag{6.47}$$

The dual characteristic function ι_k^* is also a characteristic function, but on the nonpositive numbers:

$$\iota_k^*(z_k) = \begin{cases} \infty, & z_k > 0 \\ 0, & \text{else.} \end{cases} \tag{6.48}$$

We solve (6.46) by clamping η_k to the nonpositive numbers:

$$z_k^+ = [\eta_k]_{(-\infty,0]}. \tag{6.49}$$

After updating $\mathbf{z}$ (6.46) we update $\widetilde{\mathbf{x}}$:

$$\widetilde{\mathbf{x}} \leftarrow \widetilde{\mathbf{x}}^- - \frac{1}{\mu}\left(\mathbf{z}^+ - \mathbf{z}^-\right). \tag{6.50}$$

6.2.5 Warm starting

The dual variable updates in Sections 6.2.2, 6.2.3 and 6.2.4 find values for the dual variables, $\mathbf{u}^{(n+1)}$, $\mathbf{v}^{(n+1)}$ and $\mathbf{z}^{(n+1)}$ that approximately maximize the dual update problem (6.19). We use these dual variables and the induced solution $\widetilde{\mathbf{x}}^{(n+1)}\left(\mathbf{u}^{(n+1)}, \mathbf{v}^{(n+1)}, \mathbf{z}^{(n+1)}\right)$, stored

in the buffer $\widetilde{\mathbf{x}}$, to determine $\mathbf{x}^{(n+1)}$:

$$\mathbf{x}^{(n+1)} = \widetilde{\mathbf{x}}^{(n+1)}\big(\mathbf{u}^{(n+1)}, \mathbf{v}^{(n+1)}, \mathbf{z}^{(n+1)}\big) = \widetilde{\mathbf{x}}, \tag{6.51}$$

then re-form the outer update problem (6.5) and repeat the process.

We initialize our algorithm with all the dual variables set to zero. We could also reset the dual variables to zero every outer iteration, but this empirically leads to slow convergence rates. Instead, mirroring a practice in other alternating directions algorithms, we warm-start each outer iteration with the previous iteration's dual variable values. This has an extrapolation-like effect on the buffer $\widetilde{\mathbf{x}}$:

$$\begin{aligned}
\widetilde{\mathbf{x}} &\leftarrow \widetilde{\mathbf{x}}^{(n+2)}\big(\mathbf{u}^{(n+1)}, \mathbf{v}^{(n+1)}, \mathbf{z}^{(n+1)}\big) && (6.52)\\
&= \mathbf{x}^{(n+1)} - \frac{1}{\mu}\big(\mathbf{A}'\mathbf{u}^{(n+1)} + \mathbf{C}'\mathbf{v}^{(n+1)} + \mathbf{z}^{(n+1)}\big) && (6.53)\\
&= \mathbf{x}^{(n)} - \frac{2}{\mu}\big(\mathbf{A}'\mathbf{u}^{(n+1)} + \mathbf{C}'\mathbf{v}^{(n+1)} + \mathbf{z}^{(n+1)}\big) && (6.54)\\
&= \widetilde{\mathbf{x}}^{-} + \big(\mathbf{x}^{(n+1)} - \mathbf{x}^{(n)}\big). && (6.55)
\end{aligned}$$

After initializing $\widetilde{\mathbf{x}}$ with this "extrapolated" value, subsequent iterated dual updates refine the update. This extrapolation is just an initial condition for the iterative algorithm solving the dual problem (6.19). If the dual function $\mathsf{D}^{(n+1)}$ were maximized exactly then this extrapolation would have no effect on $\widehat{\mathbf{x}}^{(n+1)}$.

This section outlined the mathematical framework of our proposed CT reconstruction algorithm. Using duality and group coordinate ascent, we decomposed the process of solving the original reconstruction problem (6.1) into an iterated series of optimization steps, each considering only a portion of the original cost function. The next section describes how we implemented these operations on the GPU.

6.3 Implementation

For implementing the algorithm described in Section 6.2, graphics processing units (GPUs) have two important properties:

- GPUs can provide impressive speedups for highly parallel workloads, and;
- GPUs often have much less memory than their host computers.

The first property means that algorithm designers should favor independent operations with regular memory accesses. Our proposed algorithm consists of five operations, each of which can be efficiently implemented on the GPU:

- Tomography update (6.27): Updating the tomography dual variables corresponding to a group of projection views, $\mathbf{v}_g$, consists of projecting those views, a few vector operations, and then backprojecting those views. Implementing an efficient CT system model on the GPU is nontrivial, and we rely on previous work in this area [50, 85, 87].

- Denoising update (6.30): Updating a "half-difference" of elements of $\mathbf{v}$ is also highly parallel. We assign one thread to each element dual variable being updated; each thread updates two neighboring pixels of the image $\widetilde{\mathbf{x}}$. The workload for each thread is independent, and memory accesses are both local and regular.

- The nonnegativity update (6.49) and warm starting operation (6.55) both consist entirely of separable, parallelizable vector operations.

The GPU's memory constraints are very relevant for imaging problems with large amounts of data and many variables. For example, we performed the experiments in Section 6.4 on a machine with four NVIDIA Tesla C2050s having 3 GB of memory apiece. The wide-cone axial experiment in Section 6.4.4 requires about 894 MB each for the noisy data $\mathbf{y}$ and the statistical weights $\mathbf{W}$ when stored in single-precision 32-bit floating point. Storing the regularizer parameters $\{\beta_k\}$ and a single image $\mathbf{x}$ would take an additional 907 MB apiece. Altogether, storing one image and the parameters of the reconstruction problem would take about 2.7 GB, leaving no room for the algorithm to store any additional state on a single GPU!

Because the X-ray CT reconstruction problem (6.1) is so memory-intensive, many algorithms will need to periodically transfer some data between the GPU and the host computer. If performed naïvely, these transfers can have a significant effect on algorithm speed. Fortunately, modern GPUs can perform calculations and memory transfers simultaneously, so we can "hide" the latency of these transfers to some degree.

Initialize CPU memory: $\mathbf{x}^{(0)} = \mathbf{x}_{\text{FBP}}$, $\mathbf{u} = \mathbf{0}$, $\mathbf{v} = \mathbf{0}$, $\mathbf{z} = \mathbf{0}$,		
Initialize GPU memory: $\underbrace{\widetilde{\mathbf{x}}_{\text{GPU}} = \mathbf{x}^{(0)},\ \mathbf{b}_{\text{GPU}} = \mathbf{0},}_{\text{Image sized}}\ \underbrace{\mathbf{u}_{g,\,\text{GPU}} = \mathbf{0},\ \mathbf{y}_{g,\,\text{GPU}} = \mathbf{0},\ \mathbf{m}_{g,\,\text{GPU}} = \mathbf{0},\ \mathbf{w}_{g,\,\text{GPU}} = \mathbf{0}.}_{\text{Projection view sized}}$		
Repeat iterations n, until convergence:	Transfer	Computation on GPU
Send nonnegativity variables to GPU	$\mathbf{z}^{-} \rightarrow \mathbf{b}_{\text{GPU}}$	Update N_{tomo} views of $\mathbf{u}$ (6.27)
Nonnegativity update		Update $\mathbf{z}$ (6.49)
Send nonnegativity variables to host	$\mathbf{z}^{+} \leftarrow \mathbf{b}_{\text{GPU}}$	Update N_{tomo} views of $\mathbf{u}$ (6.27)
Repeat $N_{\text{view}}/(2N_{\text{subset}}N_{\text{tomo}})$ times:		
Choose half-difference $\mathbf{v}_g$ randomly		
Send $\mathbf{v}_g$ to GPU	$\mathbf{v}_g^{-} \rightarrow \mathbf{b}_{\text{GPU}}$	Update N_{tomo} views of $\mathbf{u}$ (6.27)
Denoising update		Denoising update (6.30)
Send updated $\mathbf{v}_g$ to host	$\mathbf{v}_g^{+} \leftarrow \mathbf{b}_{\text{GPU}}$	Update N_{tomo} views of $\mathbf{u}$ (6.27)
Send $\mathbf{x}^{(n)}$ to GPU	$\mathbf{x}^{(n)} \rightarrow \mathbf{b}_{\text{GPU}}$	
Warm-start $\widetilde{\mathbf{x}}^{(n+1)}$		$\mathbf{b}_{\text{GPU}} = \widetilde{\mathbf{x}}_{\text{GPU}}$; warm-start $\widetilde{\mathbf{x}}_{\text{GPU}}$ (6.55)
Send $\mathbf{x}^{(n+1)}$ to host	$\mathbf{x}^{(n+1)} \leftarrow \mathbf{b}_{\text{GPU}}$	

Figure 6.2: Pseudocode for the proposed algorithm. The buffer $\widetilde{\mathbf{x}}_{\text{GPU}}$ is updated on the GPU using (6.17) in every step as the dual variables are updated, and the buffer $\mathbf{b}_{\text{GPU}}$ stores other variables as they are needed on the GPU. Updating each view of $\mathbf{u}$ involves a one-view projection and backprojection and transferring a small amount of memory. The view weights $\mathbf{w}_g$, data $\mathbf{y}_g$, dual variables $\mathbf{u}_g$, and majorizer weights $\mathbf{m}_g$ are transferred to the GPU prior to updating $\mathbf{u}_g$. Only the updated $\mathbf{u}_g$ needs to be transferred back back to the host afterwards.

6.3.1 Streaming

Our algorithm has many variables: the dual variable $\mathbf{v}$ alone is often as large as 13 image volumes. Along with the reconstruction problem's parameters, this is far too much to fit simultaneously on the GPU for many realistic problem sizes. Fortunately, each of proposed algorithm's operations requires a comparatively small subset of these data. For example, performing a tomography update requires only $\widetilde{\mathbf{x}}$, and the data, weights and dual variables corresponding to the view being updated.

The algorithm in Figure 6.2 allocates on the GPU only

- a buffer containing $\widetilde{\mathbf{x}}$,
- an image-sized buffer for storing $\mathbf{z}$ or a subset of $\mathbf{v}$,
- the regularizer parameters $\{\beta_k\}$,

and several negligibly small view-sized buffers on the GPU. The dual variables are stored primarily on the host computer and transferred to and from the GPU as needed.

The tomography update requires a relatively small amount of data: several view-sized buffers. Even for the wide-cone reconstruction in Section 6.4.4, each tomography update requires less than 4 MB of projection-domain data. The projection and backprojection involved in the tomography update take much longer to perform than it takes to transfer the dual variables and weights to and from the GPU. Therefore, the tomography update is computation-bound. On the other hand, the nonnegativity, denoising, and warm-start operations require whole images to be transferred to and from the GPU with relatively small amounts of computation. The speed with which these operations can be performed is bounded by the latency of data transfers between the host computer and the GPU.

Modern GPUs can perform computations and transfer memory at the same time. This allows us to "hide" some of the cost of latency-bound operations by performing computation-bound operations and vice versa. The pseudocode in Figure 6.2 interleaves computation-bound and transfer-bound operations. After each large memory transfer is begun, the algorithm performs N_{tomo} tomography updates. These tomography updates serve to "hide" the latency of the large memory transfer by performing useful work instead of waiting for the relatively slow memory transfer to finish. Section 6.3.3 discusses selecting N_{tomo} and other algorithm parameters.

6.3.2 Multiple-device implementation

Besides providing more computational resources, implementing a CT reconstruction algorithm on multiple GPUs can reduce the memory burden on each device. Many "distributed" algorithms either store additional variables on each node and/or perform redundant calculations to avoid very expensive inter-node communications [7, 16, 40, 76]. These designs are based on the assumption that communication between devices is extremely expensive. It may be tempting to view multiple-GPU programming as a "distributed" setting, but at least in CT reconstruction, frequent communication between the host computer and the GPU(s) seems necessary due to GPU memory constraints. Adding additional GPUs that regularly communicate to the host need not significantly increase the total amount of communication the algorithm performs. Instead of using a more sophisticated "distributed" algorithm framework [40], we distribute the memory and computation of the single-GPU algorithm over multiple devices in a straightforward way.

Similar to [88, Appendix E], we divide all image-sized buffers into N_{device} chunks transaxially, e.g., along the y direction. Note that this is a different approach from [40, 76], where the image is divided axially along z. Each device also stores two $N_x N_z$-pixel padding buffers. Because the image-sized buffers are the largest buffers our proposed algorithm stores on the GPU, this effectively reduces the memory burden on each device by a factor of almost N_{device}.

6.3.2.1 Tomography update

The buffer $\widetilde{\mathbf{x}}$ is distributed across multiple GPUs. Fortunately, the tomography update (6.27) is linear in $\widetilde{\mathbf{x}}$. When updating the group of dual variables $\mathbf{u}_g$, each device projects its chunk of $\widetilde{\mathbf{x}}$ and sends the projection to the host computer. The host accumulates these projections, performs the update (6.27), and transmits the dual update $\mathbf{u}_g^+ - \mathbf{u}_g^-$ back to each device. Each device then backprojects the update into its chunk of the volume, updating the distributed $\widetilde{\mathbf{x}}$.

6.3.2.2 Denoising update

Every element of the dual variable $\mathbf{v}$ couples two pixels together. Most of these pairs of pixels lie on only one device; in these cases, the denoising update is simple and requires no additional communication between the GPUs. However, some of the elements of $\mathbf{v}$ couple pixels that are stored on different GPUs. Prior to performing the update for these

elements, the algorithm must communicate the pixels on the GPU boundaries between devices.

Fortunately, such communication is needed for only roughly a quarter of the denoising updates. Most of the "half-difference" groups in which $\mathbf{v}$ is updated require no communication. For example, in Figure 6.1 suppose that $\{x_1, \ldots, x_6\}$ were stored on one device and $\{x_7, \ldots, x_{12}\}$ are stored on another. Updating the green group of differences $\{v_{12}, v_{13}, v_{14}\}$ would require communication between the devices, but updating the cyan group $\{v_9, v_{10}, v_{11}, v_{15}, v_{16}, v_{17}\}$ would not.

6.3.2.3 Nonnegativity update and warm-starting

The nonnegativity update and warm-stating operation are both separable in the elements of the dual variables and $\widetilde{\mathbf{x}}$, so implementing these operations on multiple devices is straightforward.

6.3.3 Parameter selection

There are three parameters in the algorithm listed in Figure 6.2: N_{tomo}, N_{subset} and μ. These parameters affect the algorithm's convergence speed but not the converged image. This section gives the heuristics we used to set the values of these parameters, but the proposed algorithm can be used with a wide range of values for these variables.

The algorithm performs an outer update (i.e., increments the iteration n) after updating $N_{\text{view}}/N_{\text{subset}}$ views of $\mathbf{u}$, chosen randomly with replacement. In that time, it performs

$$N_{\text{denoise}} = \frac{N_{\text{view}}}{2N_{\text{tomo}}N_{\text{subset}}} \tag{6.56}$$

denoising updates. We heuristically set N_{denoise} to be large enough that the expected number of outer iterations between updating an element of $\mathbf{v}$ is about one. Because each denoising update modifies half of the elements of $\mathbf{v}$ corresponding to a single direction, this means $N_{\text{denoise}} \approx 2N_{\text{r}}$, where N_{r} is the number of directions along which the regularizer penalizes differences. For the common case of penalizing differences in 13 directions around each pixel (as in our experiments), we want $N_{\text{denoise}} \approx 26$. We then choose N_{subset} to be about twice N_{tomo}.

For example, in the shoulder case below, $N_{\text{view}} = 6852$. We set $N_{\text{tomo}} = 8$ and $N_{\text{subset}} = 18$, yielding $N_{\text{denoise}} = 23$. The wide cone case had $N_{\text{view}} = 984$; we used with $N_{\text{subset}} = 6$

and $N_{\text{tomo}} = 3$, resulting in $N_{\text{denoise}} = 27$.

We chose μ using the mean of the entries of the diagonal matrix $\mathbf{MW}$, where $\mathbf{W}$ contains the weights in the data-fit term and $\mathbf{M}$ contains the entries of all the diagonal majorizers for the tomography update (6.28):

$$\mu = \frac{\sum_{i=1}^{M} [\mathbf{MW}]_{ii}}{4 \cdot M}. \tag{6.57}$$

This heuristic is intended to yield a well-conditioned tomography update (6.27). Smaller μ would make outer proximal majorization tighter (6.5) at the cost of making the dual problem (6.19) possibly more ill-conditioned.

6.4 Experiments

This section presents experimental results on two datasets: a helical shoulder scan using real patient data in Section 6.4.3 and a wide-cone axial scan using simulated data in Section 6.4.4. For both cases we measured the progress of all algorithms tested towards a converged reference $\widehat{\mathbf{x}}$ using the root mean squared difference (RMSD):

$$\text{RMSD}\big(\mathbf{x}^{(n)}\big) = \sqrt{\frac{||\mathbf{x}^{(n)} - \widehat{\mathbf{x}}||_{\mathbf{M}_{\text{ROI}}}^2}{N_{\text{ROI}}}}. \tag{6.58}$$

The diagonal matrix M_{ROI} discards images' "end slices" that are needed due to the "long object problem" in cone-beam CT reconstruction but not used clinically.

All algorithms were implemented with the OpenCL GPU programming interface and the Rust programming language. We used the separable footprints CT system model [49]. Experiments were run on a machine with 48 GB of RAM and four aging NVIDIA Tesla C2050s with 3 GB of memory apiece. We expect that more modern hardware would accelerate the runtimes of all the algorithms tested, but their relative speeds would be mostly unchanged.

6.4.1 Ordered subsets

In addition to the proposed algorithm, we implemented multiple-device versions of the popular ordered subsets (OS) algorithm with several types of momentum-based acceleration [1, 39, 44]. The OS implementations store the following variables on the GPU:

- the current image $\mathbf{x}$,
- the coefficients of the diagonal majorizer $\mathbf{D} \succeq \mathbf{A}'\mathbf{W}\mathbf{A}$,
- an accumulator for the current search direction,
- the regularizer parameters $\{\beta_k\}$, and
- an additional image-sized variable to store the momentum state, if applicable.

The OS methods require more image-sized buffers than our proposed algorithm. Like the proposed algorithm, these image-sized volumes are divided across multiple GPUs transaxially, so the memory burden for each device decreases almost linearly with the number of devices. The devices must communicate pixels that lie on an edge between devices before computing a regularizer gradient; this happens N_{subset} times per iteration.

6.4.2 Equivalent iterations

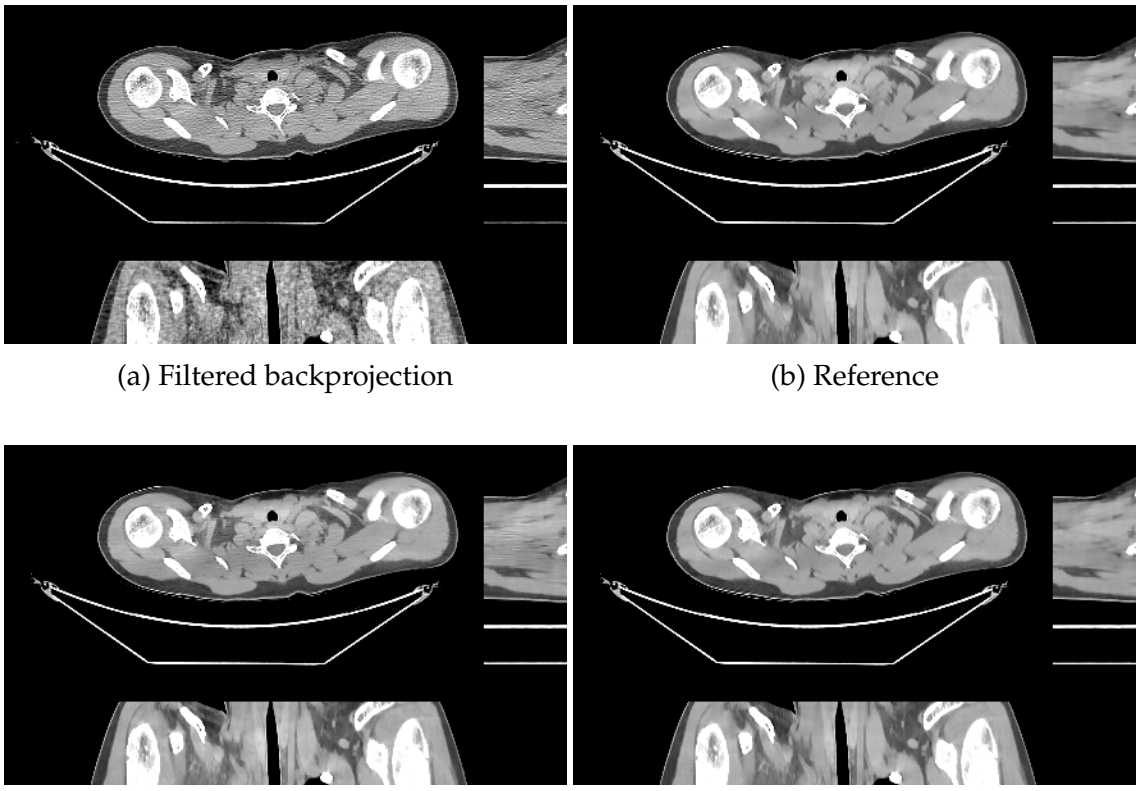
To meaningfully compare the per-iteration performance of the proposed algorithm and the ordered subsets (OS) algorithms we use "equivalent iterations," or equits [90]. Because the most computationally intensive part of these algorithms is projection and backprojection, one equit corresponds to computing roughly the same number of forward and backprojections:

- For OS, one equit corresponds to a loop through all the data.
- For the proposed algorithm, one equit corresponds to N_{subset} iterations; e.g., $\mathbf{x}^{(N_{\text{subset}})}$ and $\mathbf{x}^{(2N_{\text{subset}})}$ are the outputs of successive equits. Over this time, our algorithm computes about N_{view} forward and backprojections.

The proposed algorithm performs more denoising updates than OS and transfers more memory between the host and GPU in an equit, so each equit takes longer to perform. Nonetheless, as the following experiments show, the proposed algorithm converges considerably more quickly than the OS algorithms in terms of runtime.

6.4.3 Helical shoulder scan

Our first reconstruction experiment uses data from a helical scan provided by GE Healthcare. The data were 6852 views with 888 channels and 32 rows. We reconstructed the im-



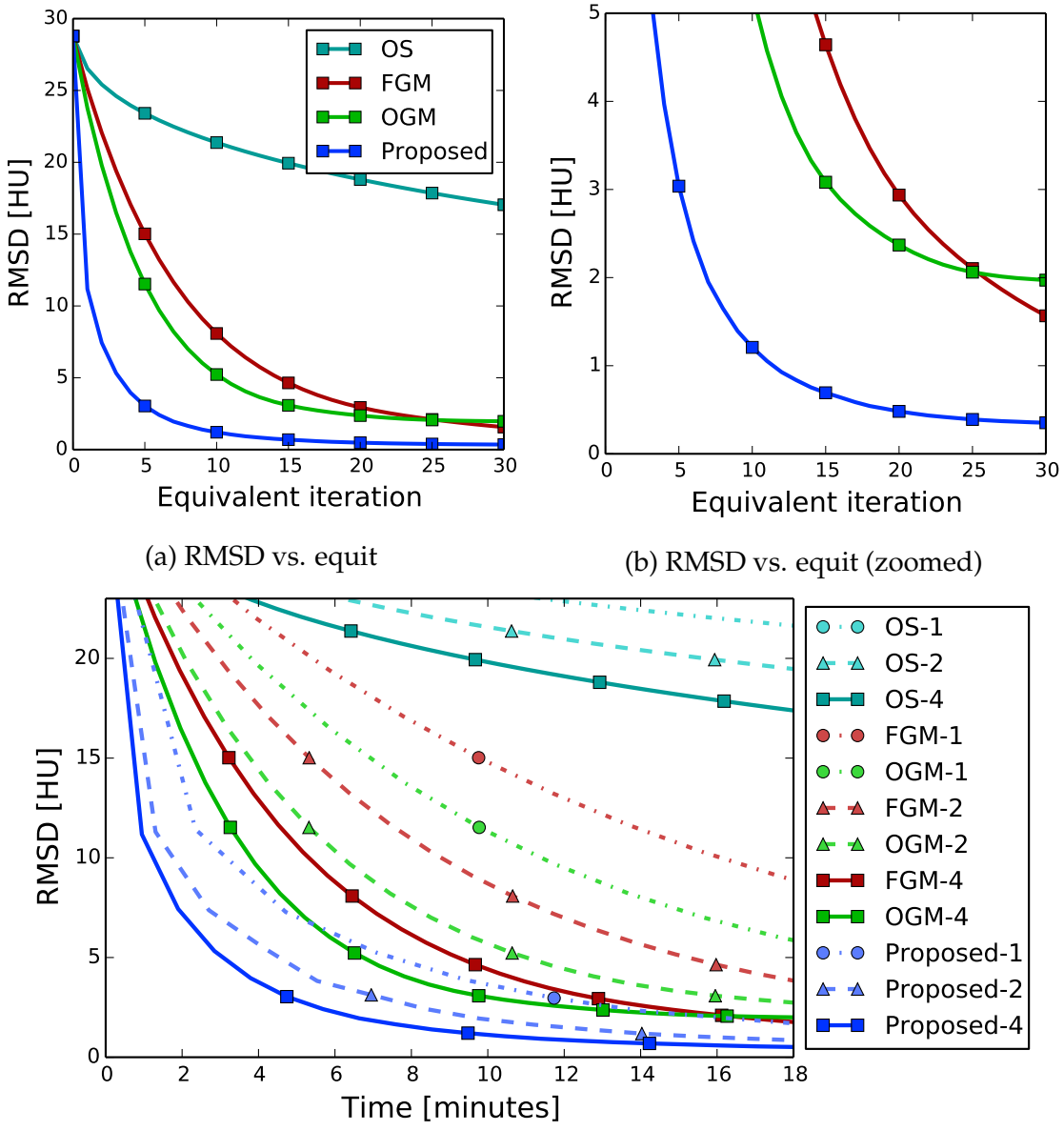
(a) Filtered backprojection

(b) Reference

(c) OS-OGM with 4 GPUs after 8 equits (5.2 minutes)

(d) Proposed with 4 GPUs after 5 equits (4.8 minutes)

Figure 6.3: Top row: initial FBP and reference images for the helical shoulder case in Section 6.4.3. Bottom row: images from the proposed algorithm and the state of the art OS-OGM algorithm on 4 GPUs after about 5 minutes. Images have been trimmed and are displayed in a $[800,\ 1200]$ modified Hounsfield unit window.



(a) RMSD vs. equit

(b) RMSD vs. equit (zoomed)

(c) RMSD vs. time

Figure 6.4: Convergence plots for the helical shoulder case in Section 6.4.3. Markers are placed every five equits. The proposed algorithm on one device converges about as quickly as the state of the art OS-OGM algorithm does on four devices. Additional devices provide further acceleration.

Algorithm	Per equit	Time to converge within 5 HU RMSD		2 HU RMSD	
OGM-1	114 sec.	1290 sec.	21.5 min.	3519 sec.	58.6 min.
OGM-2	62 sec.	702 sec.	11.7 min.	1786 sec.	29.8 min.
OGM-4	38 sec.	430 sec.	7.2 min.	1092 sec.	18.2 min.
Proposed-1	130 sec.	565 sec.	9.4 min.	973 sec.	16.2 min.
Proposed-2	82 sec.	332 sec.	5.5 min.	587 sec.	9.8 min.
Proposed-4	57 sec.	229 sec.	3.8 min.	408 sec.	6.8 min.

Table 6.1: Timings for the helical case in Section 6.4.3.

age on a $512 \times 512 \times 109$-pixel grid. The regularizer penalized differences along all 13 directions (i.e., 26 3D neighbors) with the Fair potential function (6.39) with $\delta = 10$ Hounsfield units (HU). All iterative algorithms were initialized using the filtered backprojection image in Figure 6.3a. Figure 6.3b shows an essentially converged reference, generated by running thousands of iterations of momentum-accelerated separable quadratic surrogates [44] (i.e., ordered subsets with one subset).

We ran the proposed algorithm with $N_{\text{tomo}} = 8$ and $N_{\text{subset}} = 18$. We also ran ordered subsets with 12 subsets (OS) [1], OS with Nesterov's momentum [44] (FGM), and OS with a faster acceleration [39] (OGM) on one, two and four GPUs. Figure 6.4 shows RMSD in Hounsfield units against time and equivalent iteration.

Figure 6.4a shows the proposed algorithm converging considerably faster than the OS-type algorithms in terms of equits, and unlike the OS-type algorithms will converge to the solution $\widehat{\mathbf{x}}$ if run for long enough[3]. The per-equit acceleration is not only a result of the proposed algorithm doing more work per equit than the OS algorithms: Figure 6.4c shows that the proposed algorithm on one GPU achieves early-iteration performance comparable to the fastest OS algorithm with four GPUs.

Table 6.1 lists several timings for the algorithms in this experiment. Although the OS algorithms achieved more dramatic speedups using multiple devices than the proposed algorithm, additional devices did help accelerate convergence. Figures 6.3c and 6.3d show images from both algorithms on four devices after about five minutes of computation. The proposed algorithm produced an image that much more closely matches the converged reference.

[3]See Appendix E

Algorithm	Per equit	Time to converge within 5 HU RMSD		2 HU RMSD	
OGM-2	85 sec.	976 sec.	16.3 min.	–	–
OGM-4	50 sec.	569 sec.	9.5 min.	1193 sec.	19.9 min.
Proposed-2	126 sec.	516 sec.	8.6 min.	939 sec.	15.6 min.
Proposed-4	98 sec.	380 sec.	6.3 min.	661 sec.	11.0 min.

Table 6.2: Timings for the axial case in Section 6.4.4.

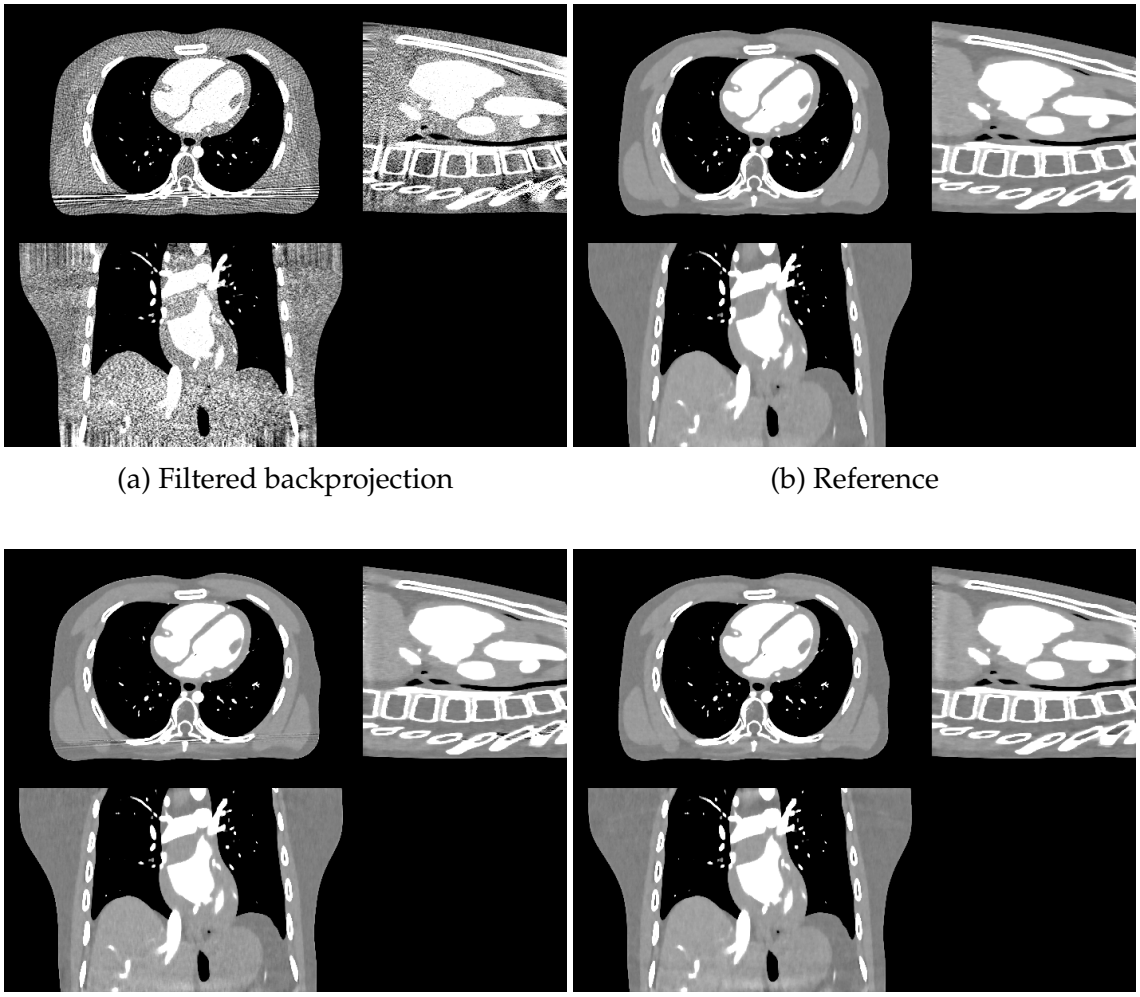
6.4.4 Wide-cone axial simulation

Our second experiment is a wide-cone axial reconstruction with a simulated phantom. We simulated a noisy scan of the XCAT phantom [78] taken by a scanner with 888 channels and 256 rows over a single rotation of 984 views. Images were reconstructed onto a $718 \times 718 \times 440$-pixel grid. As in Section 6.4.3, the regularizer used the Fair potential and penalized differences along all 13 neighboring directions. All iterative algorithms were initialized with the filtered backprojection image in Figure 6.5a. An essentially converged reference image is shown in Figure 6.5b.

This problem was too large to fit on one of our 3 GB GPUs, so we present results for two and four GPUs. We ran the same set of OS algorithms as the previous experiment with 12 subsets. The proposed algorithm used $N_{\text{subset}} = 6$ and $N_{\text{tomo}} = 3$.

Figures 6.6a and 6.6c show the progress of the tested algorithms towards the converged reference. The proposed algorithm running on two devices is about as fast as OS-OGM running on four devices, and additional devices accelerate convergence even more. Figures 6.5c and 6.5d illustrate outputs from OS-OGM and the proposed algorithm after about five minutes. After five minutes, the OS algorithm still contains noticeable streaks that the dual algorithm has already removed. Both algorithms have significant distance to the reference at the end slices of the image after five minutes.

Table 6.2 lists timings for OS-OGM and the proposed algorithm. The trends are similar to the smaller helical case in Table 6.1. The OS algorithms scale better ($1.7\times$ faster) than the proposed algorithm ($1.2\times$ faster) from two to four GPUs, but the acceleration provided by the proposed algorithm is enough to compensate for lower parallelizability.



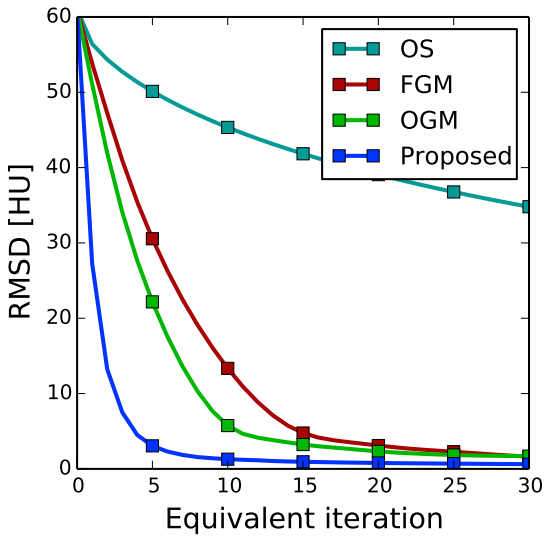
(a) Filtered backprojection

(b) Reference

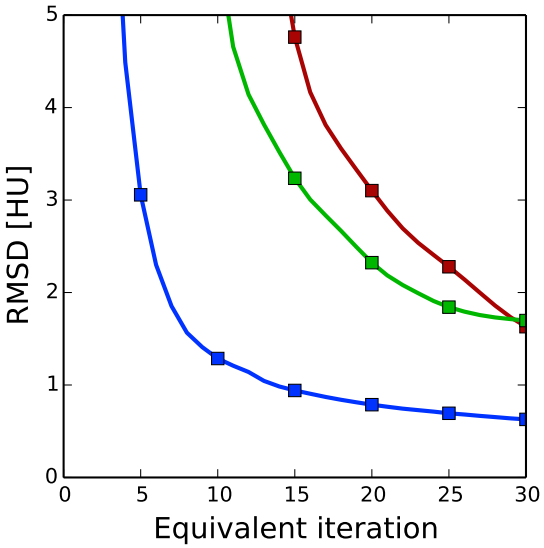
(c) OS-OGM with 4 GPUs after 6 equits (5.2 minutes)

(d) Proposed with 4 GPUs after 3 equits (4.7 minutes)

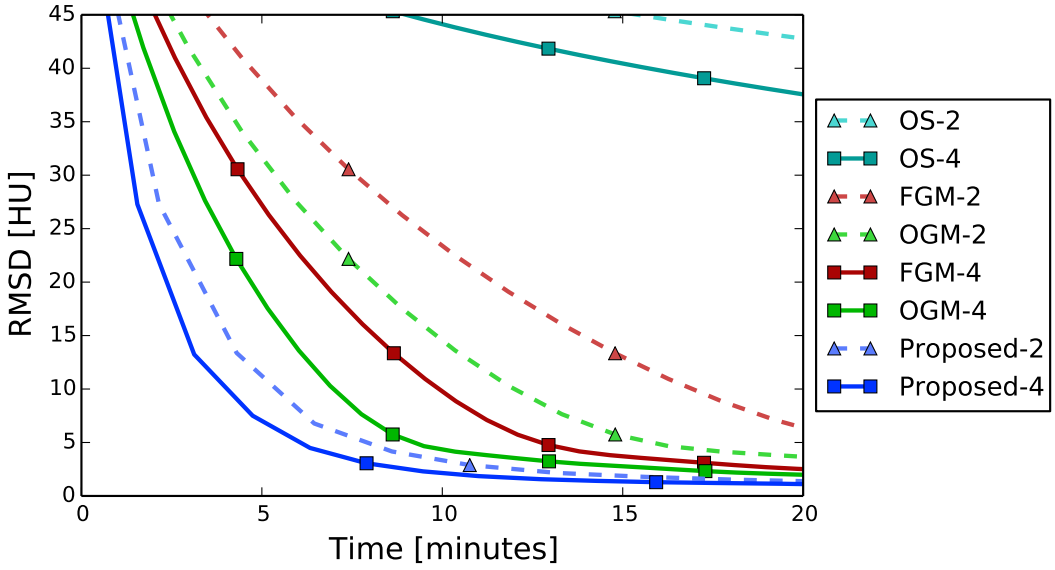
Figure 6.5: Top row: initial FBP and reference images for the wide-cone axial case in Section 6.4.4. Bottom row: images from the proposed algorithm and the state of the art OS-OGM algorithm on 4 GPUs after about 5 minutes. Images have been trimmed and are displayed in a $[800,\ 1200]$ modified Hounsfield unit window.



(a) RMSD vs. equit



(b) RMSD vs. equit (zoomed)



(c) RMSD vs. time

Figure 6.6: Convergence plots for the wide-cone axial case in Section 6.4.4. Markers are placed every five equits. The proposed algorithm converges about as quickly on two devices as OS-OGM does on four. Additional devices accelerate convergence.

6.5 Conclusions and future work

We presented a novel CT reconstruction algorithm that uses alternating updates in the dual domain. The proposed algorithm is fast in terms of per-iteration performance and "wall clock" runtime, and it converges more quickly than popular state of the art ordered subsets algorithms. The algorithm also eventually converges to the true solution of the statistical reconstruction problem, and it can handle a wide range of regularizers including the nondifferentiable total variation regularizer.

The algorithm also maps well onto the GPU. Many of its steps are highly parallelizable and perform regular memory accesses, which are essential qualities for good GPU performance. Although the algorithm stores many variables in the host computer's memory, the amount of memory required for each update is relatively small, and we hide the latency of transferring variables to and from the GPU by performing computation-bounded operations. Finally, the proposed algorithm is easily adapted for multiple GPUs, providing further acceleration and decreasing the memory burden on each GPU.

Due to communication overhead, the acceleration provided by adding additional GPUs showed diminishing returns. To achieve further acceleration, multiple computers (or groups of GPUs on a single node) may need to be combined using a "distributed" algorithm framework [7, 40]. How to best adapt the proposed algorithm to these frameworks is future work.

The proposed algorithm introduces a dual variable for each difference penalized by the edge-preserving regularizer R. While this memory cost is not too great for a reasonably-equipped modern computer when only the 13 neighbors around each pixel are considered by the regularizer, significantly increasing the number of differences computed may make the approach we take in this paper infeasible. Consequently, adapting the proposed algorithm for patch-based or nonlocal regularizers may be challenging.

The random process we use for choosing which groups of the tomography dual variable $\mathbf{u}$ and denoising dual variable $\mathbf{v}$ to update is basic and almost certainly suboptimal. A more sophisticated strategy may provide additional acceleration. We currently also keep the parameter μ constant, but future work will explore if modifying μ as the algorithm runs, e.g., [67], can accelerate convergence further. Finally, future work will explore replacing our heuristic operation counts (e.g., N_{tomo}) with algorithmically tuned values.

CHAPTER 7

Algorithmic majorizer design

Given a symmetric matrix $\mathbf{H}$, we say that the symmetric matrix $\mathbf{M}$ majorizes $\mathbf{H}$ if none of the eigenvalues of $\mathbf{M} - \mathbf{H}$ are negative. Matrix majorizers are central to majorize-minimize algorithms and are ubiquitous in image processing algorithms; e.g., Chapters 4, 5, and 6 of this thesis. Better majorizers can significantly affect how quickly algorithms converge [58], but majorizers are usually designed by hand on a per-problem basis. Algorithmic majorizer design expands the class of usable majorizers and reveals more effective majorizers, but a straightforward semidefinite programming approach requires too much memory to be computationally feasible.

This chapter presents an algorithmic approach to designing a majorizing matrix $\mathbf{M}$ for a general given symmetric $\mathbf{H}$. The algorithm we present has relatively low memory requirements, and it is practical for large problems where storing dense $N \times N$ matrices would be infeasible. The goal of this chapter is to enable the design of more exotic majorizers than are currently accessible using various inequalities. Hopefully this leads to tighter [17] majorizers and faster convergence [24].

7.1 Introduction

Let $\mathbf{H} \in \mathbb{R}^{N \times N}$ be a given symmetric positive semidefinite matrix. Our goal is to find another matrix $\mathbf{M} \succeq \mathbf{H}$ that majorizes $\mathbf{H}$. That is, we want the difference matrix $\mathbf{M} - \mathbf{H}$ to be positive semidefinite.

Conventionally, $\mathbf{M}$ is found using a collection of inequalities and matrix properties. A simple and common bound is $\lambda_{\max}(\mathbf{H})\mathbf{I}$, which is often used in optimization algorithms for which the cost function gradient is Lipschitz continuous (with Lipschitz constant

$\lambda_{\max}(\mathbf{H})$). This is a very loose bound. A tighter bound is the diagonal matrix

$$\mathbf{M}_{\text{SQS}} = \operatorname*{diag}_j \left\{ \sum_{i=1}^{N} |\mathbf{H}|_{ij} \right\}. \tag{7.1}$$

We call this the "separable quadratic surrogates" (SQS) majorizer due to its ubituity in ordered subsets with SQS (OS-SQS) algorithms [1]. If $\mathbf{H}$ contains only nonnegative entries (e.g., $\mathbf{A}'\mathbf{W}\mathbf{A}$, the Hessian of the X-ray CT datafit term) then $\mathbf{M}_{\text{SQS}}$ can be quickly computed via

$$\mathbf{M}_{\text{SQS}} = \operatorname*{diag}_j \left\{ [\mathbf{H}\mathbf{1}]_j \right\}. \tag{7.2}$$

Our experiments suggest that $\mathbf{M}_{\text{SQS}}$ is a fairly tight majorizer when $\mathbf{H}$ contains only nonnegative entries, and its ease of computation (7.2) make it a very useful tool. However, when $\mathbf{H}$ contains negative entries, $\mathbf{M}_{\text{SQS}}$ appears to be less tight. In this case, carefully designed majorizers that exploit the structure of $\mathbf{H}$, e.g., [58], can significantly improve on $\mathbf{M}_{\text{SQS}}$.

We will design majorizers of the following form:

$$\mathbf{M} = \mathbf{K}'\mathbf{D}\mathbf{K}, \tag{7.3}$$

where $\mathbf{D} \in \mathbb{R}^{K \times K}$ is a diagonal matrix and $\mathbf{K} \in \mathbb{R}^{K \times N}$, $K \geq N$ has full column rank. We assume the matrix $\mathbf{K}$ is given beforehand and focus on designing $\mathbf{D}$. A special case is $\mathbf{K} = \mathbf{I}$, which leads to a diagonal majorizer. To encourage more "tight" majorizers, we penalize the entries of $\mathbf{D}$:

$$\mathbf{M} = \mathbf{K}'\widehat{\mathbf{D}}\mathbf{K}, \qquad \text{where} \tag{7.4}$$

$$\widehat{\mathbf{D}} = \operatorname*{argmin}_{\mathbf{D}: \mathbf{K}'\mathbf{D}\mathbf{K} \succeq \mathbf{H}} \frac{1}{2} ||\mathbf{d}||_2^2. \tag{7.5}$$

7.1.1 Semidefinite programming approach

The design problem (7.5) is a convex optimization problem over a convex set and can be solved in a conceptually straightforward way with variable splitting and contrained optimization. Unfortunately, the constraint $\mathbf{K}'\mathbf{D}\mathbf{K} \succeq \mathbf{H}$ is difficult to enforce without storing and manipulating dense $N \times N$ matrices.

Introduce the auxiliary variable $\mathbf{G} = \mathbf{K}'\mathbf{D}\mathbf{K} - \mathbf{H}$. We formulate the design problem as

$$\widehat{\mathbf{D}} = \underset{\mathbf{D}}{\operatorname{argmin}} \ \min_{\mathbf{G}} \frac{1}{2}||\mathbf{d}||_2^2 \qquad \text{such that } \mathbf{G} = \mathbf{K}'\mathbf{D}\mathbf{K} - \mathbf{H} \text{ and } \mathbf{G} \succeq \mathbf{0}. \tag{7.6}$$

We enforce the $\mathbf{G} = \mathbf{K}'\mathbf{D}\mathbf{K} - \mathbf{H}$ constraint in the following augmented Lagrangian:

$$\mathsf{L}(\mathbf{D}, \mathbf{G}; \mathbf{E}) = \frac{1}{2}||\mathbf{d}||_2^2 + \frac{\mu}{2}||\mathbf{G} - \mathbf{K}'\mathbf{D}\mathbf{K} - \mathbf{H} + \mathbf{E}||_F^2. \tag{7.7}$$

The alternating directions method of multipliers (ADMM) [18] provides the following convergent procedure to find the solution of (7.6):

$$\mathbf{D}^{(n+1)} = \underset{\mathbf{D}}{\operatorname{argmin}} \ \mathsf{L}\big(\mathbf{D}, \mathbf{G}^{(n)}; \mathbf{E}^{(n)}\big), \tag{7.8}$$

$$\mathbf{G}^{(n+1)} = \underset{\mathbf{G}\succeq\mathbf{0}}{\operatorname{argmin}} \ \mathsf{L}\big(\mathbf{D}^{(n+1)}, \mathbf{G}; \mathbf{E}^{(n)}\big) \tag{7.9}$$

$$= \underset{\mathbf{G}\succeq\mathbf{0}}{\operatorname{argmin}} \ \frac{\mu}{2}\Big|\Big|\mathbf{G} - \big(\mathbf{K}'\mathbf{D}^{(n+1)}\mathbf{K} - \mathbf{H} - \mathbf{E}^{(n)}\big)\Big|\Big|_F^2, \tag{7.10}$$

$$\mathbf{E}^{(n+1)} = \mathbf{E}^{(n)} + \mathbf{K}'\mathbf{D}^{(n+1)}\mathbf{K} - \mathbf{H} - \mathbf{G}^{(n+1)}. \tag{7.11}$$

The updates (7.8)-(7.11) are easy to implement in principle, but (7.10) and (7.11) require storing dense $N \times N$ matrices. The $\mathbf{G}$ update (7.10), projection of the matrix $\mathbf{K}'\mathbf{D}^{(n+1)}\mathbf{K} - \mathbf{H} - \mathbf{E}^{(n)}$ onto the semidefinite cone, involves computing the eigenvalue decomposition of an $N \times N$ matrix and thresholding negative values [8]. The resulting $\mathbf{G}^{(n+1)}$ is a structureless $N \times N$ matrix, so the $\mathbf{E}^{(n+1)}$ computed in (7.11) is also a dense $N \times N$ matrix.

While (7.8)-(7.11) produces a series of iterates $\{\mathbf{D}^{(n)}\}$ that converge to the solution to (7.5), the associated computation and memory requirements are too great for practically sized N. The next section presents an algorithm that produces a suboptimal result but requires only a fraction of the memory.

7.2 Duality-based approach

The duality-based algorithm proposed in this section only stores vectors in $\mathbb{R}^K$ and $\mathbb{R}^N$ and does not store or operate on arbitrary $N \times N$ matrices. This makes it feasible to use this algorithm to majorize the large linear operators used in image processing.

Let Ω be the set of values of $\mathbf{d}$ for which $\mathbf{K}'\mathbf{D}\mathbf{K}$ majorizes $\mathbf{H}$:

$$\Omega = \{\mathbf{d} : \mathbf{K}'\mathbf{D}\mathbf{K} \succeq \mathbf{H}\}. \tag{7.12}$$

Our algorithm relies on the following characterization of the characteristic function on Ω, I_Ω:

$$\mathsf{I}_\Omega(\mathbf{d}) = \begin{cases} 0, & \mathbf{d} \in \Omega; \\ +\infty, & \text{else} \end{cases} \tag{7.13}$$

$$= \sup_{\mathbf{x}} \ \mathbf{x}'(\mathbf{H} - \mathbf{K}'\mathbf{D}\mathbf{K})\mathbf{x} \tag{7.14}$$

$$= \sup_{\mathbf{x}} \ -\mathbf{d}'(\mathbf{K}\mathbf{x})^2 + \mathbf{x}'\mathbf{H}\mathbf{x}, \tag{7.15}$$

where $(\mathbf{K}\mathbf{x})^2$ contains the elementwise squares of $\mathbf{K}\mathbf{x}$. Although we treat this like a conjugate representation of I_Ω, it does not use the convex conjugate of I_Ω. Plug (7.15) into the original design cost function (7.5):

$$\widehat{\mathbf{d}} = \operatorname*{argmin}_{\mathbf{d}} \ \sup_{\mathbf{x}} \ \mathsf{S}(\mathbf{d}, \mathbf{x}) \tag{7.16}$$

$$\mathsf{S}(\mathbf{d}, \mathbf{x}) = \frac{1}{2}||\mathbf{d}||^2 - \mathbf{d}'(\mathbf{K}\mathbf{x})^2 + \mathbf{x}'\mathbf{H}\mathbf{x}. \tag{7.17}$$

As is often the case when we write equations of the form (7.16), we want to switch the order of minimization and maximization. We cannot appeal to the well-known Sion minimax theorem or Fenchel duality theorem to perform this transposition. Nonetheless, there is no duality gap[1],

$$\inf_{\mathbf{d}} \ \sup_{\mathbf{x}} \ \mathsf{S}(\mathbf{d}, \mathbf{x}) = \sup_{\mathbf{x}} \ \inf_{d} \ \mathsf{S}(\mathbf{d}, \mathbf{x}), \tag{7.18}$$

[1]See Appendix F

and we proceed:

$$\widehat{\mathbf{d}} = \underset{\mathbf{d}}{\operatorname{argmin}}\ \mathsf{S}(\mathbf{d}, \mathbf{x}) \tag{7.19}$$

$$= (\mathbf{K}\widehat{\mathbf{x}})^2 \tag{7.20}$$

$$= \widetilde{\mathbf{d}}(\widehat{\mathbf{x}}) \tag{7.21}$$

$$\widehat{\mathbf{x}} = \underset{\mathbf{x}}{\operatorname{argmax}}\ \mathsf{D}(\mathbf{x}) \tag{7.22}$$

$$\mathsf{D}(\mathbf{x}) = \mathsf{S}\Big(\widetilde{\mathbf{d}}(\mathbf{x}), \mathbf{x}\Big) \tag{7.23}$$

$$= -\frac{1}{2}||\mathbf{K}\mathbf{x}||_4^4 + \mathbf{x}'\mathbf{H}\mathbf{x}. \tag{7.24}$$

Because S is strongly convex in $\mathbf{d}$, strong duality can be combined with the proof in Appendix D to show that the induced primal value $\widetilde{\mathbf{d}}(\widehat{\mathbf{x}})$ is the same solution to the original problem (7.5).

7.2.1 Solving the dual problem

The dual function D (7.23) is neither purely convex nor concave, so we cannot expect to find $\widehat{\mathbf{x}}$ by directly applying a gradient-based method. We use duality to rewrite the convex part of D for simplicity. The resulting function is not jointly concave in all its variables, so gradient-based methods and alternating optimization will converge only to a local maximum.

For the rest of this chapter, we assume that $\mathbf{H}$ can be decomposed as $\mathbf{H} = \mathbf{F}'\mathbf{F}$. This is often the case for the Hessians in image processing algorithms. Rewrite the $\mathbf{x}'\mathbf{H}\mathbf{x}$ term in D (7.23) using biconjugacy:

$$\mathsf{D}(\mathbf{x}) = -\frac{1}{2}||\mathbf{K}\mathbf{x}||_4^4 + \mathbf{x}'\mathbf{H}\mathbf{x} \tag{7.25}$$

$$= -\frac{1}{2}||\mathbf{K}\mathbf{x}||_4^4 + ||\mathbf{F}\mathbf{x}||_2^2 \tag{7.26}$$

$$= -\frac{1}{2}||\mathbf{K}\mathbf{x}||_4^4 + \sup_{\mathbf{z}}\ \mathbf{z}'\mathbf{F}\mathbf{x} - \frac{1}{4}||\mathbf{z}||_2^2, \tag{7.27}$$

because the convex conjugate of $||\cdot||_2^2$ is $\frac{1}{4}||\cdot||_2^2$. Combine (7.27) with (7.23):

$$\widehat{\mathbf{x}} = \underset{\mathbf{x}}{\text{argmax}} \ \sup_{\mathbf{z}} \ \mathsf{Q}(\mathbf{x}, \mathbf{z}) \tag{7.28}$$

$$\mathsf{Q}(\mathbf{x}, \mathbf{z}) = -\frac{1}{2}||\mathbf{K}\mathbf{x}||_4^4 + \mathbf{z}'\mathbf{F}\mathbf{x} - \frac{1}{4}||\mathbf{z}||_2^2. \tag{7.29}$$

We alternately maximize Q over $\mathbf{z}$ and $\mathbf{x}$. Holding $\mathbf{x} = \mathbf{x}^{(n)}$ fixed and maximizing over $\mathbf{z}$ is trivial:

$$\mathbf{z}^{(n+1)} = 2\mathbf{F}\mathbf{x}^{(n)}. \tag{7.30}$$

The optimization over $\mathbf{x}$ with $\mathbf{z} = \mathbf{z}^{(n+1)}$ fixed is slightly more complicated. We perform a single step of steepest ascent:

$$\mathbf{x}^{(n+1)} = \mathbf{x}^{(n)} + \alpha^{(n)} \nabla_x \mathsf{Q}\big(\mathbf{x}, \mathbf{z}^{(n+1)}\big). \tag{7.31}$$

The gradient of Q with respect to $\mathbf{x}$ is

$$\mathbf{g}^{(n)} = \nabla_{\mathbf{x}} \mathsf{Q}\big(\mathbf{x}, \mathbf{z}^{(n+1)}\big) = -2\mathbf{K}'(\mathbf{K}\mathbf{x})^3 + \mathbf{F}'\mathbf{z}^{(n+1)}. \tag{7.32}$$

The step size $\alpha^{(n)}$ is chosen to maximize Q:

$$\alpha^{(n)} = \underset{\alpha}{\text{argmax}} \ \mathsf{Q}\big(\mathbf{x}^{(n)} + \alpha \mathbf{g}^{(n)}\big) \tag{7.33}$$

$$\begin{aligned} \equiv \underset{\alpha}{\text{argmax}} \ & -\frac{\big|\big|\mathbf{K}\mathbf{g}^{(n)}\big|\big|_4^4}{2}\alpha^4 \\ & - 2\Big[\big(\mathbf{K}\mathbf{g}^{(n)}\big)^3\Big]'\big(\mathbf{K}\mathbf{x}^{(n)}\big)\alpha^3 \\ & - 3\Big[\big(\mathbf{K}\mathbf{g}^{(n)}\big)^2\Big]'\Big[\big(\mathbf{K}\mathbf{x}^{(n)}\big)^2\Big]\alpha^2 \\ & - \Big(2\big(\mathbf{K}\mathbf{g}^{(n)}\big)'\Big[\big(\mathbf{K}\mathbf{x}^{(n)}\big)^3\Big] - \big[\mathbf{d}^{(n)}\big]'\mathbf{F}'\mathbf{z}^{(n+1)}\Big)\alpha. \end{aligned} \tag{7.34}$$

One can use either a root-finding routine or one of the formulae for the roots of a third degree polynomial to perform the line search.

7.2.2 Suboptimality and feasibility

The alternating maximization algorithm in the previous section converges to a local maximum of D. Fortunately, although the local maxima of Q are suboptimal and do not induce majorizers of $\mathbf{H}$, we can use them to find an suboptimal (but feasible) solution to the majorizer design problem (7.5).

Let $\widetilde{\mathbf{x}}$ be a local maximum of D. The Hessian of D at $\widetilde{\mathbf{x}}$ is

$$\nabla^2_{\widetilde{\mathbf{x}}}\mathsf{D} = -6\mathbf{K}'\left[\operatorname*{diag}_j\left\{[\mathbf{K}\widetilde{\mathbf{x}}]_j^2\right\}\right]\mathbf{K} + 2\mathbf{H}. \tag{7.35}$$

Because $\widetilde{\mathbf{x}}$ is a local maximum of D, the Hessian of D must be negative semidefinite at this point. Therefore,

$$3\mathbf{K}'\left[\operatorname*{diag}_j\left\{[\mathbf{K}\widetilde{\mathbf{x}}]_j^2\right\}\right]\mathbf{K} \succeq \mathbf{H}. \tag{7.36}$$

Alternatively, one can run power iteration to determine the maximum eigenvalue

$$\alpha = \lambda_{\max}\left(\left(\mathbf{K}'\left[\operatorname*{diag}_j\left\{[\mathbf{K}\widetilde{\mathbf{x}}]_j^2\right\}\right]\mathbf{K}\right)^{-1/2}\mathbf{H}\left(\mathbf{K}'\left[\operatorname*{diag}_j\left\{[\mathbf{K}\widetilde{\mathbf{x}}]_j^2\right\}\right]\mathbf{K}\right)^{-1/2}\right) \tag{7.37}$$

and use the majorizer

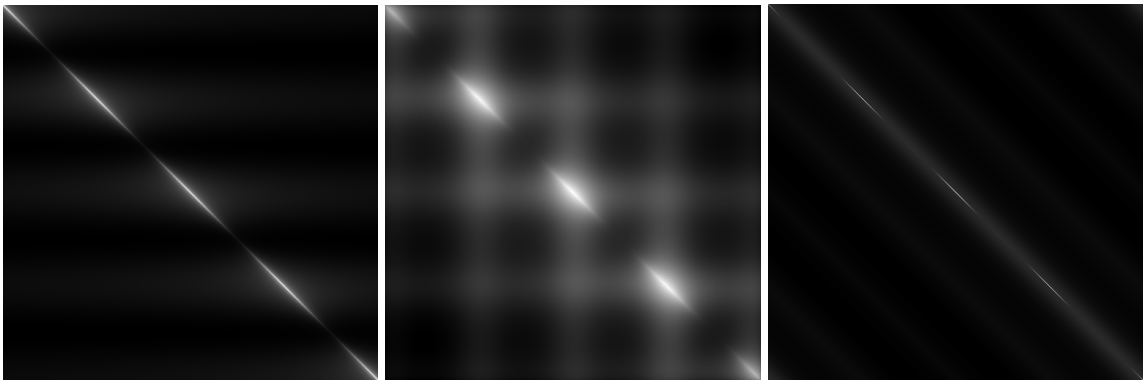
$$\alpha\mathbf{K}'\left[\operatorname*{diag}_j\left\{[\mathbf{K}\widetilde{\mathbf{x}}]_j^2\right\}\right]\mathbf{K} \succeq \mathbf{H}. \tag{7.38}$$

To summarize, our proposed memory-efficient majorizer design procedure is

- Given $\mathbf{H}$ and $\mathbf{K}$, find a local maximum of the dual function D (7.23). The alternating directions procedure in Section 7.2.1 is simple and converges to a local optimum. Let $\widetilde{\mathbf{x}}$ be the local maximizer.
- Set the diagonal matrix $\mathbf{D}$ to

$$\widetilde{\mathbf{D}} = \alpha\left[\operatorname*{diag}_j\left\{[\mathbf{K}\widetilde{\mathbf{x}}]_j^2\right\}\right], \tag{7.39}$$

  where $1 \leq \alpha \leq 3$ is either 3 or determined with power iteration. The resulting



(a) $\mathbf{F}$ (b) $\mathbf{H}$ (c) $\mathbf{M}_{\text{Design-Circ+Diag}}$

Figure 7.1: The nonnegative matrices $\mathbf{F}$ and $\mathbf{H}$ from the experiment in Section 7.3.1, and the matrix $\mathbf{M}_{\text{Design-Circ+Diag}}$ produced by our proposed algorithm.

matrix majorizes $\mathbf{H}$:

$$\mathbf{M} = \mathbf{K}'\widetilde{\mathbf{D}}\mathbf{K} \succeq \mathbf{H}. \tag{7.40}$$

7.3 Applications

The procedure in Section 7.2 does not find an optimal solution to the majorizer design problem (7.5), but it does allow us to design more exotic majorizers than are conventionally available. By selecting $\mathbf{K}$, we can design circulant, block separable, banded, and many other forms of majorizers. This section gives a preliminary experiment for one of these applications.

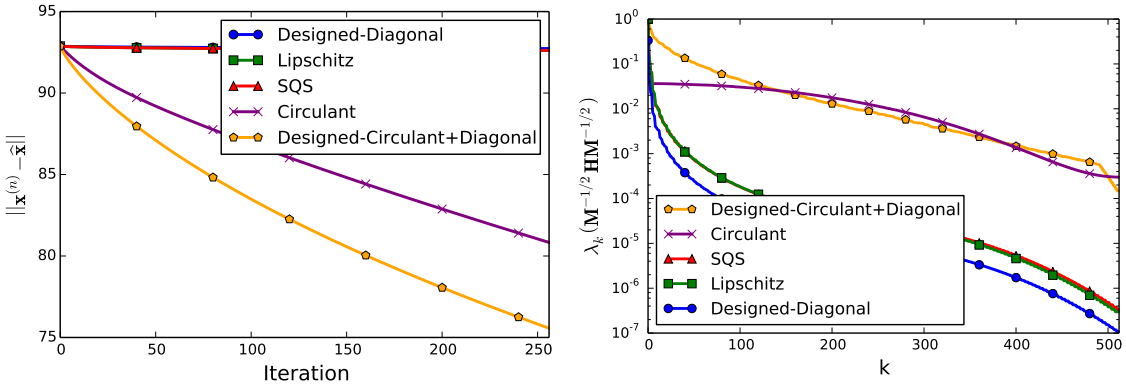
7.3.1 Weighted Toeplitz matrices and circulant majorizers

We generated the $N \times N$ weighted Toeplitz matrix $\mathbf{F}$ with $N = 512$ and entries

$$[\mathbf{F}]_{ij} = \frac{0.1 + \cos^2\left(2\pi \frac{i}{N}\right)}{\sqrt{1 + |i - j|}} \tag{7.41}$$

and set $\mathbf{H} = \mathbf{F}'\mathbf{F}$. This choice is inspired by the $1/r$-like response of the CT system matrix [15]. Figures 7.1a and 7.1b show $\mathbf{F}$ and $\mathbf{H}$, respectively. We generated three diagonal majorizers:

- $\mathbf{M}_{\text{Lipschitz}} = \lambda_{\max}(\mathbf{H})\mathbf{I}$,



(a) Distance to solution

(b) Majorized spectra

Figure 7.2: Convergence plots and majorized spectra for the small Toeplitz experiment in Section 7.3.1. The partially circulant majorizer $\mathbf{M}_{\text{Design-Circ+Diag}}$ acts like both a majorizer and a preconditioner, accelerating convergence of the simple majorize-minimize algorithm. The ideal majorizer inverts the matrix $\mathbf{H}$ and produces a uniform majorized spectrum with value 1.

- $\mathbf{M}_{\text{SQS}}$ using (7.2), and
- $\mathbf{M}_{\text{Design-Diag}}$, using the algorithing proposed in this chapter with $\mathbf{K} = \mathbf{I}$.

We also generated $\mathbf{M}_{\text{Design-Circ+Diag}}$, a combination of circulant and diagonal matrices, using the proposed algorithm with

$$\mathbf{K}_{\text{Circ+Diag}} = \begin{bmatrix} \mathbf{U}_{\text{DFT}} \\ \mathbf{I} \end{bmatrix}. \tag{7.42}$$

Finally, we computed $\mathbf{M}_{\text{Circ}}$

$$\mathbf{M}_{\text{Circ}} = \beta \widehat{\mathbf{C}}, \tag{7.43}$$

where

$$\widehat{\mathbf{C}} = \underset{\mathbf{C} \text{ circulant}}{\text{argmin}} \ ||\mathbf{C} - \mathbf{H}||_{\mathbf{F}}^2, \tag{7.44}$$

and β is chosen with power iteration so $\beta \widehat{\mathbf{C}} \succeq \mathbf{H}$.

We used each of the majorizers to solve the quadratic minimization problem

$$\widehat{\mathbf{x}} = \underset{\mathbf{x}}{\operatorname{argmin}} \ \frac{1}{2}\mathbf{x}'\mathbf{H}\mathbf{x} + \mathbf{x}'\mathbf{g}, \tag{7.45}$$

with $\mathbf{g}$ and $\mathbf{x}^{(0)}$ initialized with zero-mean normal random values. We performed the following simple majorize-minimize (MM) procedure:

$$\mathbf{x}^{(n+1)} = \mathbf{x}^{(n)} - \mathbf{M}^{-1}\big(\mathbf{H}\mathbf{x}^{(n)} + \mathbf{g}\big). \tag{7.46}$$

This experiment explores the relative accelerations that different majorizers provide. If we actually wanted to solve (7.45) quickly with a majorize-minimize algorithm, we would also use some first order acceleration scheme [41, 59, 61]. Figure 7.2a shows how quickly the MM algorithm converged to the solution of (7.45) as a function of iteration with each of the majorizers, and Figure 7.2b shows the eigenvalues of $\mathbf{M}^{-\frac{1}{2}}\mathbf{H}\mathbf{M}^{-\frac{1}{2}}$ for each majorizer $\mathbf{M}$.

The designed diagonal majorizer underperforms the more conventional SQS majorizer, $\mathbf{M}_{\text{SQS}}$. This is possibly due to the proposed algorithm producing a suboptimal solution to the majorizer design dual problem. Regardless, diagonal majorizers are not our primary interest here: the proposed algorithm is more useful for generating majorizers with more exotic structures.

Except for edge conditions, circulant matrices and Toeplitz matrices are very similar. Because $\mathbf{M}_{\text{Circ}}$ and $\mathbf{M}_{\text{Design-Circ+Diag}}$ contain circulant terms, they can approximate $\mathbf{H}$ while majorizing it. When these matrices are inverted in the majorize-minimize procedure (7.46), they act as both preconditioners and majorizers. The preconditioning effect is appears in better-conditioned spectra (i.e., closer to unform 1s) for $\mathbf{M}_{\text{Circ}}$ and $\mathbf{M}_{\text{Design-Circ+Diag}}$ in Figure 7.2b and in faster convergence rates in Figure 7.2a. Because the majorizer design algorithm in this chapter can generate a majorizer with both circulant and diagonal components, it can capture the nonuniform weighting in $\mathbf{H}$ better than $\mathbf{M}_{\text{circ}}$. This results in further improvement over $\mathbf{M}_{\text{circ}}$.

7.4 Conclusions and future work

This chapter proposed an algorithmic approach to designing matrix majorizers. We used a duality-based method to produce an algorithm that is memory efficient: unlike a straight-

forward semidefinite programming approach, our algorithm does not store or manipulate dense $N \times N$ matrices. Hopefully, this makes it possible to use for realistically-sized imaging problems. In a preliminary experiment, we majorized a weighted Toeplitz matrix with a combination of circulant and diagonal matrices. Because the majorizer approximated the Toeplitz matrix better than purely diagonal or circulant matrix could, it provided significant acceleration. Future work will explore other applications of algorithmically-designed majorizers.

CHAPTER 8

Conclusions and Future Work

This thesis contains two types of results: general techniques to algorithmically perform analysis and approximations that are conventionally done by hand (Chapters 3 and 7), and specialized algorithms to quickly solve specific problems for which general techniques are not as well suited (Chapters 4, 5, and 6).

Chapter 3 presents a technique to analyze matrices that may be well-approximated by circulant operators. We had hoped that by analyzing these matrices numerically, we could find useful structures that had escaped our intuition, but instead our results confirmed existing heuristics. Given the relative success of the problem-specialized algorithms in Chapters 4, 5, and 6, perhaps our general approach was "doomed" to be slower than algorithms that used easily exploited structures; e.g., the "checkerboard" groups in Chapter 4. Other problems with more complex structure without obvious inroads for specialized algorithms may be good applications for the analysis in Chapter 3.

The majorizer design work in Chapter 7 is relatively immature but promising. Algorithmic majorizer design may enable algorithm designers to find intuitive majorizers that had been out of reach for computational reasons. These majorizers may be tighter than those accessible by conventional means and may lead to faster-converging algorithms. Our early results on simplified problems are encouraging, and there are many existing majorize-minimize algorithms that may benefit from more exotic majorizers, including a few in this thesis. The majorizer design dual problem involves maximizing a smooth but nonconcave function. We provide only a very basic algorithm for solving this maximization in this thesis that often converges to a suboptimal local maximizer. Future work may find a more effective approach to solving this problem.

Chapters 4, 5, and 6 contain algorithms that exploit the structure of the cost functions they minimize. We use a combination of duality, variable splitting, and group coordinate optimization to produce algorithms that are not only fast but work well on the

GPU. The algorithms often converge more quickly than conventional algorithms on a "per-iteration" basis as well as a "per-time" basis, and we suspect our results may be translatable to non-GPU architectures as well.

Our approach to handling optimization problems involving the CT system matrix in Chapters 5 and 6 – using duality and group coordinate ascent instead of ordered subsets (OS) – yields fast and convergent algorithms with only a modest increase in memory overhead. Heuristically, this approach appears to yield rapid convergence because many of the terms in the tomography data-fit term are "correlated;" this is the same intuition behind OS's subset approximation. However, we do not have a theoretical understanding of why this approach is so effective for CT reconstruction, and further analysis would help us identify in which situations this approach is effective.

The problem-specialized algorithms in this thesis exploit structures in the cost functions they minimize: e.g., the Markov-like property of the edge-preserving regularizer or the high correlation between measurements in the CT data-fit term. Different problems – e.g., sparse-view CT, denoising with nonlocal or patch-based regularization – have very different structures and may require very different approaches. We are slowly getting closer to solving the CT reconstruction problem in this thesis "quickly enough," but this only opens the door to more sophisticated and computationally challenging problems.

APPENDIX A

Matrix permutation

Let $\mathbf{P}_j \in \mathbb{R}^{N\times N}$ be the permutation operator

$$[\mathbf{P}_j\mathbf{v}]_k = v_{1+(|j+k-2| \bmod N)}. \tag{A.1}$$

Define the matrix permutation operator $\mathcal{C}$:

$$\mathcal{C}(\mathbf{H}) = \begin{bmatrix}\mathbf{P}_1\mathbf{h}_1 & \mathbf{P}_2\mathbf{h}_2 & \cdots & \mathbf{P}_N\mathbf{h}_N\end{bmatrix}. \tag{A.2}$$

Let $\{\mathbf{C}_k\}$ and $\{\mathbf{D}_k\}$ contain K $N \times N$ circulant and diagonal matrices, respectively. Apply $\mathcal{C}$ the sum of circulant and diagonal matrices:

$$\mathcal{C}\left(\sum_{k=1}^{K}\mathbf{C}_k\mathbf{D}_k\right) = \sum_{k=1}^{K}\mathcal{C}(\mathbf{C}_k\mathbf{D}_k) \tag{A.3}$$

$$= \sum_{k=1}^{K}\begin{bmatrix}\mathbf{P}_1\mathbf{C}_k\mathbf{e}_1[\mathbf{D}_k]_{1,1} & \mathbf{P}_2\mathbf{C}_k\mathbf{e}_2[\mathbf{D}_k]_{2,2} & \cdots & \mathbf{P}_n\mathbf{C}_k\mathbf{e}_N[\mathbf{D}_k]_{N,N}\end{bmatrix} \tag{A.4}$$

$$= \sum_{k=1}^{K}\begin{bmatrix}\mathbf{P}_1\mathbf{c}_k[\mathbf{D}_k]_{1,1} & \mathbf{c}_k[\mathbf{D}_k]_{2,2} & \cdots & \mathbf{c}_k[\mathbf{D}_k]_{N,N}\end{bmatrix} \tag{A.5}$$

$$= \sum_{k=1}^{K}\mathbf{c}_k\mathbf{d}_k{}', \tag{A.6}$$

where $\mathbf{e}_k$ is the kth elementary basis vector.

APPENDIX B

From diagonal-circulant to diagonal-circulant-diagonal

Let $\mathbf{S} = \frac{1}{2}\sum_{k=1}^{K} \mathbf{C}_k\mathbf{D}_k + \mathbf{D}_k\mathbf{C}_k{}'$ with $\mathbf{D}_k$ diagonal and $\mathbf{C}_k$ circulant, with $\mathbf{C}_k = \mathbf{E}_k + \mathbf{O}_k$ the even-odd decomposition of $\mathbf{C}_k$. That is, $\mathbf{E}_k = \mathbf{E}_k{}'$ and $\mathbf{O}_k = -\mathbf{O}_k{}'$. Let $\mathbf{G}_k$ and $\mathbf{H}_k$ be diagonal matrices derived from the eigenvalue decomposition

$$\sigma_{k,1}\mathbf{g}_k\mathbf{g}_k{}' + \sigma_{k,2}\mathbf{h}_k\mathbf{h}_k{}' = \mathbf{1}\mathbf{d}_k{}' + \mathbf{d}_k\mathbf{1}'. \tag{B.1}$$

Then

$$\mathbf{S} = \frac{1}{2}\sum_{k=1}^{K} \mathbf{C}_k\mathbf{D}_k + \mathbf{D}_k\mathbf{C}_k{}' \tag{B.2}$$

$$= \frac{1}{2}\sum_{k=1}^{K} \sigma_{k,1}\mathbf{G}_k\mathbf{E}_k\mathbf{G}_k + \sigma_{k,2}\mathbf{H}_k\mathbf{E}_k\mathbf{H}_k. \tag{B.3}$$

B.1 Proof

Expand $\mathbf{S}$ using the even-odd decompositions of the $\{\mathbf{C}_k\}$:

$$\mathbf{S} = \frac{1}{2}\sum_{k=1}^{K} \mathbf{C}_k\mathbf{D}_k + \mathbf{D}_k\mathbf{C}_k{}' \tag{B.4}$$

$$= \frac{1}{2}\sum_{k=1}^{K} (\mathbf{E}_k + \mathbf{O}_k)\mathbf{D}_k + \mathbf{D}_k(\mathbf{E}_k - \mathbf{O}_k) \tag{B.5}$$

$$= \frac{1}{2}\sum_{k=1}^{K} \mathbf{E}_k\mathbf{D}_k + \mathbf{D}_k\mathbf{E}_k + \frac{1}{2}\sum_{k=1}^{K} \mathbf{O}_k\mathbf{D}_k - \mathbf{D}_k\mathbf{O}_k. \tag{B.6}$$

The first sum is a symmetric matrix, and the second sum is an antisymmetric matrix. Because $\mathbf{S}$ is also symmetric, the second term must sum to zero:

$$= \frac{1}{2} \sum_{k=1}^{K} \mathbf{E}_k \mathbf{D}_k + \mathbf{D}_k \mathbf{E}_k. \tag{B.7}$$

The (i, j)th entry of one of the terms in this sum is

$$\begin{aligned}
[\mathbf{E}_k \mathbf{D}_k + \mathbf{D}_k \mathbf{E}_k]_{i,j} &= [\mathbf{e}_k]_{1+((i-j) \bmod N)} \Big([\mathbf{d}_k]_i + [\mathbf{d}_k]_j \Big) && \text{(B.8)} \\
&= [\mathbf{e}_k]_{1+((i-j) \bmod N)} [\mathbf{1}\mathbf{d}_k{}' + \mathbf{d}_k \mathbf{1}']_{i,j} && \text{(B.9)} \\
&= [\mathbf{e}_k]_{1+((i-j) \bmod N)} [\sigma_{k,1} \mathbf{g}_k \mathbf{g}_k{}' + \sigma_{k,2} \mathbf{h}_k \mathbf{h}_k{}']_{i,j} && \text{(B.10)} \\
&= [\sigma_{k,1} \mathbf{G}_k \mathbf{E}_k \mathbf{G}_k + \sigma_{k,2} \mathbf{H}_k \mathbf{E}_k \mathbf{H}_k]_{i,j}, && \text{(B.11)}
\end{aligned}$$

where $\mathbf{G}_k$, $\mathbf{H}_k$, $\sigma_{k,1}$ and $\sigma_{k,2}$ come from the eigenvalue decomposition

$$\sigma_{k,1} \mathbf{g}_k \mathbf{g}_k{}' + \sigma_{k,2} \mathbf{h}_k \mathbf{h}_k{}' = \mathbf{1}\mathbf{d}_k{}' + \mathbf{d}_k \mathbf{1}'. \tag{B.12}$$

Thus,

$$\mathbf{S} = \frac{1}{2} \sum_{k=1}^{K} \sigma_{k,1} \mathbf{G}_k \mathbf{E}_k \mathbf{G}_k + \sigma_{k,2} \mathbf{H}_k \mathbf{E}_k \mathbf{H}_k. \tag{B.13}$$

APPENDIX C

Fenchel duality for GPU-based reconstruction algorithm

Proving (6.16) involves a straightforward application of Fenchel's duality theorem, see e.g., [5, Theorem 4.4.3]. Define

$$f(\mathbf{x}) = \frac{\mu}{2}\left|\left|\mathbf{x} - \mathbf{x}^{(n)}\right|\right|_2^2, \tag{C.1}$$

$$\mathbf{K} = \begin{bmatrix} \mathbf{A} \\ \mathbf{C} \\ \mathbf{I} \end{bmatrix}. \tag{C.2}$$

We write the blocks of elements of $\mathbf{Kx}$ as $[\mathbf{Kx}]_{\mathbf{u}}$, $[\mathbf{Kx}]_{\mathbf{v}}$ and $[\mathbf{Kx}]_{\mathbf{z}}$. Define

$$g(\mathbf{Kx}) = \mathsf{L}([\mathbf{Kx}]_{\mathbf{u}}) + \mathsf{R}([\mathbf{Kx}]_{\mathbf{v}}) + \mathsf{I}([\mathbf{Kx}]_{\mathbf{z}}). \tag{C.3}$$

The value attained by the primal update problem (6.5), can be written in this terminology as

$$p = \min_{\mathbf{x}} f(\mathbf{x}) + g(\mathbf{Kx}) \tag{C.4}$$

$$= \min_{\mathbf{x}} \mathsf{J}^{(n)}(\mathbf{x}). \tag{C.5}$$

The convex conjugates of f and g are [8, pg. 95]

$$f^*(\mathbf{x}^*) = \frac{1}{2\mu}||\mathbf{x}^*||_2^2 + (\mathbf{x}^*)'\mathbf{x}^{(n)}, \tag{C.6}$$

$$g^*(\mathbf{q}^*) = \mathsf{L}^*(\mathbf{q}_{\mathbf{u}}^*) + \mathsf{R}^*(\mathbf{q}_{\mathbf{v}}^*) + \mathsf{I}^*(\mathbf{q}_{\mathbf{z}}^*). \tag{C.7}$$

The value attained by maximizing the dual function (6.19) is

$$d = \sup_{\mathbf{q}^*} \; -f^*(-\mathbf{K}'\mathbf{q}^*) - g^*(\mathbf{q}^*) \tag{C.8}$$

$$= \sup_{\mathbf{q}^*} \; \mathsf{D}^{(n)}(\mathbf{q}^*_{\mathbf{u}}, \mathbf{q}^*_{\mathbf{v}}, \mathbf{q}^*_{\mathbf{z}}). \tag{C.9}$$

Note that although (C.8) apparently differs from the statement in [5, Theorem 4.4.2] by a sign, the expressions are equivalent.

The domain of f is

$$\text{dom } f = \mathbb{R}^N, \tag{C.10}$$

and the image of dom f under $\mathbf{K}$ is

$$\mathbf{K} \text{ dom } f = \text{range } \mathbf{K}. \tag{C.11}$$

The set over which g is continuous is

$$\text{cont } g = \{\boldsymbol{\theta} : \boldsymbol{\theta}_{\mathbf{z}} > \mathbf{0}\}. \tag{C.12}$$

Finally, by Fenchel's duality theorem, because

$$\mathbf{K} \text{ dom } f \bigcap \text{cont } g \neq \emptyset, \tag{C.13}$$

and f and g are both convex functions, $p = d$.

APPENDIX D

Equivalence of primal- and dual-based solutions

Let the value of $\mathbf{x}^{(n+1)}$ produced by solving the primal update problem (6.5) be

$$\mathbf{x}_p = \underset{\mathbf{x}}{\operatorname{argmin}} \sup_{\mathbf{u},\mathbf{v},\mathbf{z}} \mathsf{S}^{(n)}(\mathbf{x},\mathbf{u},\mathbf{v},\mathbf{z}). \tag{D.1}$$

The value of $\mathbf{x}^{(n+1)}$ induced by solving the dual problem (6.19) is

$$\mathbf{x}_d = \widetilde{\mathbf{x}}^{(n+1)}\big(\widehat{\mathbf{u}}^{(n+1)},\widehat{\mathbf{v}}^{(n+1)},\widehat{\mathbf{z}}^{(n+1)}\big), \tag{D.2}$$

$$\widetilde{\mathbf{x}}^{(n+1)}(\mathbf{u},\mathbf{v},\mathbf{z}) = \underset{\mathbf{x}}{\operatorname{argmin}}\ \mathsf{S}^{(n)}(\mathbf{x},\mathbf{u},\mathbf{v},\mathbf{z}), \tag{D.3}$$

where

$$\begin{aligned}\widehat{\mathbf{u}}^{(n+1)},\widehat{\mathbf{v}}^{(n+1)},\widehat{\mathbf{z}}^{(n+1)} &= \underset{\mathbf{u},\mathbf{v},\mathbf{z}}{\operatorname{argmax}}\ \mathsf{D}^{(n)}(\mathbf{u},\mathbf{v},\mathbf{z}) \\ &= \mathsf{S}^{(n)}\big(\widetilde{\mathbf{x}}^{(n+1)}(\mathbf{u},\mathbf{v},\mathbf{z}),\mathbf{u},\mathbf{v},\mathbf{z}\big).\end{aligned} \tag{D.4}$$

Our goal is to show $\mathbf{x}_p = \mathbf{x}_d$.

We proceed by contradiction. Suppose $\mathbf{x}_p \neq \mathbf{x}_d$. Because $\mathsf{S}^{(n)}$ is strongly convex and $\mathbf{x}_d$ minimizes $\mathsf{S}^{(n)}$ when the dual variables are fixed at $\big(\widehat{\mathbf{u}}^{(n+1)},\widehat{\mathbf{v}}^{(n+1)},\widehat{\mathbf{z}}^{(n+1)}\big)$ (D.3),

$$d = \mathsf{S}^{(n)}\big(\mathbf{x}_d,\widehat{\mathbf{u}}^{(n+1)},\widehat{\mathbf{v}}^{(n+1)},\widehat{\mathbf{z}}^{(n+1)}\big) \tag{D.5}$$

$$< \mathsf{S}^{(n)}\big(\mathbf{x}_p,\widehat{\mathbf{u}}^{(n+1)},\widehat{\mathbf{v}}^{(n+1)},\widehat{\mathbf{z}}^{(n+1)}\big) \tag{D.6}$$

$$\leq \sup_{\mathbf{u},\mathbf{v},\mathbf{z}} \mathsf{S}^{(n)}(\mathbf{x}_p,\mathbf{u},\mathbf{v},\mathbf{z}) \tag{D.7}$$

$$= p. \tag{D.8}$$

But this is impossible because $p = d$ (see Appendix C). Thus, $\mathbf{x}_p = \mathbf{x}_d$.

APPENDIX E

Convergence for GPU-based reconstruction algorithm with approximate updates

If the maximizing dual variables are found exactly (i.e., if (6.20) holds with equality), then the proposed algorithm is a simple majorize-minimize procedure (6.5) and $\left\{\mathbf{x}^{(n)}\right\} \to \widehat{\mathbf{x}}$. In practice finding the exact maximizers of $\mathsf{D}^{(n)}$ is too computationally expensive, and we settle for approximate optimization. Fortunately, under conditions similar to those for other approximate-update algorithms like ADMM [18], the proposed algorithm converges even with inexact maximization of $\mathsf{D}^{(n)}$.

Let $\epsilon_{\mathbf{u}}^{(n+1)}$, $\epsilon_{\mathbf{v}}^{(n+1)}$ and $\epsilon_{\mathbf{z}}^{(n+1)}$ be the weighted error between the approximate optimizers $\mathbf{u}^{(n+1)}$, $\mathbf{v}^{(n+1)}$ and $\mathbf{z}^{(n+1)}$ of $\mathsf{D}^{(n)}$ and the true optimizers $\widehat{\mathbf{u}}^{(n+1)}$, $\widehat{\mathbf{v}}^{(n+1)}$, $\widehat{\mathbf{z}}^{(n+1)}$:

$$\epsilon_{\mathbf{u}}^{(n)} = \left|\left|\widehat{\mathbf{u}}^{(n)} - \mathbf{u}^{(n)}\right|\right|_{\mathbf{A}\mathbf{A}'}, \tag{E.1}$$

$$\epsilon_{\mathbf{v}}^{(n)} = \left|\left|\widehat{\mathbf{v}}^{(n)} - \mathbf{v}^{(n)}\right|\right|_{\mathbf{C}\mathbf{C}'}, \tag{E.2}$$

$$\epsilon_{\mathbf{z}}^{(n)} = \left|\left|\widehat{\mathbf{z}}^{(n)} - \mathbf{z}^{(n)}\right|\right|. \tag{E.3}$$

Assume that we solve the dual maximization subproblem (6.20) well enough that these errors are summable:

$$\sum_{n=1}^{\infty} \epsilon_{\mathbf{v}}^{(n)} < \infty, \qquad \sum_{n=1}^{\infty} \epsilon_{\mathbf{u}}^{(n)} < \infty, \qquad \sum_{n=1}^{\infty} \epsilon_{\mathbf{z}}^{(n)} < \infty. \tag{E.4}$$

Let $\widehat{\mathbf{x}}^{(n+1)}$ be the exact solution to the primal update problem (6.5). The error between the

approximate update $\mathbf{x}^{(n+1)}$ and the exact update, $\widehat{\mathbf{x}}^{(n+1)}$, is

$$\begin{aligned}
\epsilon_{\mathbf{x}}^{(n)} &= \left|\left|\mathbf{x}^{(n+1)} - \widehat{\mathbf{x}}^{(n+1)}\right|\right| && \text{(E.5)} \\
&= \left|\left|\widetilde{\mathbf{x}}^{(n+1)}\left(\mathbf{u}^{(n+1)}, \mathbf{v}^{(n+1)}, \mathbf{z}^{(n+1)}\right)\right.\right. \\
&\qquad\qquad \left.\left. - \widetilde{\mathbf{x}}^{(n+1)}\left(\widehat{\mathbf{u}}^{(n+1)}, \widehat{\mathbf{v}}^{(n+1)}, \widehat{\mathbf{z}}^{(n+1)}\right)\right|\right| && \text{(E.6)} \\
&\leq \frac{1}{\mu}\left(\epsilon_v^{(n)} + \epsilon_u^{(n)} + \epsilon_{\mathbf{z}}^{(n)}\right), && \text{(E.7)}
\end{aligned}$$

using the form of the dual-induced primal solution (6.17) and the triangle inequality. Because the dual update errors are summable (E.4), the primal update errors $\left\{\epsilon_x^{(n)}\right\}$ are also summable. Then, by [18, Theorem 3], the proposed algorithm is a convergent "generalized proximal point algorithm" and produces a sequence of iterates $\left\{\mathbf{x}^{(n)}\right\}$ that converge to $\widehat{\mathbf{x}}$.

In practice, it may be difficult to verify numerically that the conditions (E.4) hold, but at least this analysis provides some sufficient conditions for convergence. In contrast, the popular ordered subsets with separable quadratic surrogates (OS-SQS) algorithm [1] has no convergence theory (and can diverge even for well-conditioned problems).

APPENDIX F

Minimax result for algorithmic majorizer design

Let $\mathbf{H} \succ \mathbf{0}$ be a given positive semidefinite matrix, $\mathbf{K} \in \mathbb{R}^{K \times N}$, $K \geq N$ have full column rank and define Ω to be

$$\Omega = \{\mathbf{d} : \mathbf{K}'\mathbf{D}\mathbf{K} \succeq \mathbf{H}\}. \tag{F.1}$$

Consider the function

$$\mathsf{S}(\mathbf{d}, \mathbf{x}) = \frac{1}{2}||\mathbf{d}||^2 - \mathbf{d}'(\mathbf{K}\mathbf{x})^2 + \mathbf{x}'\mathbf{H}\mathbf{x}. \tag{F.2}$$

The primal function is

$$\mathsf{J}(\mathbf{d}) = \sup_{\mathbf{x}} \ \mathsf{S}(\mathbf{d}, \mathbf{x}) \tag{F.3}$$

$$= \frac{1}{2}||\mathbf{d}||^2 + \mathsf{I}(\mathbf{d}). \tag{F.4}$$

and the dual function is

$$\mathsf{D}(\mathbf{x}) = \inf_{\mathbf{d}} \ \mathsf{S}(\mathbf{d}, \mathbf{x}) \tag{F.5}$$

$$= -\frac{1}{2}||\mathbf{K}\mathbf{x}||_4^4 + \mathbf{x}'\mathbf{H}\mathbf{x}. \tag{F.6}$$

In this section we show the minimum of the primal function is equal to the maximum of the dual function:

$$p = \min_{\mathbf{d}} \mathsf{J}(\mathbf{d}) = \sup_{\mathbf{x}} \ \mathsf{D}(\mathbf{x}) = d. \tag{F.7}$$

F.1 Proof

Because J is a strongly convex function and Ω is a convex set, there exists a unique minimizer of J over Ω. Let $\widehat{\mathbf{d}}$ be this minimizer:

$$\widehat{\mathbf{d}} = \underset{\mathbf{d}\in\Omega}{\operatorname{argmin}}\ \mathsf{J}(\mathbf{d}). \tag{F.8}$$

Because $\mathbf{H} \succ \mathbf{0}$ the unconstrained minimizer of $\frac{1}{2}||\mathbf{d}||^2$, $\mathbf{d}_{\text{unconstrained}} = \mathbf{0}$, does not lie in Ω. Therefore, $\widehat{\mathbf{d}}$ is on the boundary of Ω and $-\nabla\mathsf{J}\Big(\widehat{\mathbf{d}}\Big) = \widehat{\mathbf{d}}$ is normal to Ω.

We can characterize the feasible set Ω as an intersection of half-spaces:

$$\Omega = \bigcap_{\mathbf{x}} \{\mathbf{d} : \mathbf{x}'(\mathbf{K}'\mathbf{D}\mathbf{K} - \mathbf{H}) \geq 0\} \tag{F.9}$$

$$= \bigcap_{\mathbf{x}} \big\{\mathbf{d} : \mathbf{d}'(\mathbf{K}\mathbf{x})^2 \geq \mathbf{x}'\mathbf{H}\mathbf{x}\big\}. \tag{F.10}$$

Because $-\nabla\mathsf{J}\Big(\widehat{\mathbf{d}}\Big) = \widehat{\mathbf{d}}$ is normal to Ω and on the boundary of Ω, there exists an $\widehat{\mathbf{x}}$ for which one of the above inequalities holds with equality. That is,

$$-\nabla\mathsf{J}\Big(\widehat{\mathbf{d}}\Big) = \widehat{\mathbf{d}} = \alpha(\mathbf{K}\widehat{\mathbf{x}})^2 \tag{F.11}$$

and

$$\widehat{\mathbf{x}}'\mathbf{H}\widehat{\mathbf{x}} = \widehat{\mathbf{d}}'(\mathbf{K}\widehat{\mathbf{x}})^2 = \alpha\big((\mathbf{K}\widehat{\mathbf{x}})^2\big)'(\mathbf{K}\widehat{\mathbf{x}})^2 = \alpha||\mathbf{K}\widehat{\mathbf{x}}||_4^4. \tag{F.12}$$

We use (F.11) to find the minimum of the primal function:

$$\mathsf{J}\Big(\widehat{\mathbf{d}}\Big) = \frac{1}{2}\Big|\Big|\widehat{\mathbf{d}}\Big|\Big|_2^2 \tag{F.13}$$

$$= \frac{1}{2}\Big|\Big|\alpha(\mathbf{K}\widehat{\mathbf{x}})^2\Big|\Big|_2^2 \tag{F.14}$$

$$= \frac{\alpha^2}{2}||\mathbf{K}\widehat{\mathbf{x}}||_4^4 \tag{F.15}$$

$$= p. \tag{F.16}$$

It is widely known that $p \geq d$. Therefore, to show $p = d$ it suffices to show that

$\mathsf{D}(\mathbf{x}) = p$ for some $\mathbf{x}$. We try $\mathsf{D}(\beta\widehat{\mathbf{x}})$, with $\beta^2 = \alpha$:

$$\mathsf{D}(\beta\widehat{\mathbf{x}}) = -\frac{1}{2}||\beta\mathbf{K}\widehat{\mathbf{x}}||_4^4 + \beta^2\widehat{\mathbf{x}}'\mathbf{H}\widehat{\mathbf{x}} \tag{F.17}$$

$$= -\frac{\beta^4}{2}||\mathbf{K}\widehat{\mathbf{x}}||_4^4 + \beta^2\widehat{\mathbf{x}}'\mathbf{H}\widehat{\mathbf{x}} \tag{F.18}$$

via (F.12),

$$= -\frac{\beta^4}{2}||\mathbf{K}\widehat{\mathbf{x}}||_4^4 + \beta^2\alpha||\mathbf{K}\widehat{\mathbf{x}}||_4^4 \tag{F.19}$$

$$= \frac{\alpha^2}{2}||\mathbf{K}\widehat{\mathbf{x}}||_4^4 = p, \tag{F.20}$$

completing the proof.

BIBLIOGRAPHY

[1] H. Erdogan and J. A. Fessler. Ordered subsets algorithms for transmission tomography. *Phys. Med. Biol.*, 44(11):2835–51, November 1999. 11, 12, 35, 36, 49, 51, 54, 55, 58, 63, 65, 70, 72, 77, 89, 93, 99, 118

[2] M. V. Afonso, José M Bioucas-Dias, and Marió A T Figueiredo. Fast image recovery using variable splitting and constrained optimization. *IEEE Trans. Im. Proc.*, 19(9):2345–56, September 2010. 72

[3] T. Baran, D. Wei, and A. V. Oppenheim. Linear programming algorithms for sparse filter design. *IEEE Trans. Sig. Proc.*, 58(3):1605–17, March 2010. 26

[4] D. P. Bertsekas. *Nonlinear programming*. Athena Scientific, Belmont, 2 edition, 1999. 5

[5] J. M. Borwein and Q. J. Zhu. *Techniques of variational analysis*. Springer, 2005. 5, 8, 114, 115

[6] C. Boutsidis, M. W. Mahoney, and P. Drineas. An improved approximation algorithm for the column subset selection problem, 2010. arxiv 0812.4293. 17

[7] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Found. & Trends in Machine Learning*, 3(1):1–122, 2010. 87, 97

[8] S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge, UK, 2004. 5, 100, 114

[9] NASA Jet Propulsion Laboratory / Caltech. PIA05600: Eyeing "Eagle Crater", 2004. 48

[10] A. Chambolle and T. Pock. A first-order primal-dual algorithm for convex problems with applications to imaging. *J. Math. Im. Vision*, 40(1):120–145, 2011. 37, 44, 49

[11] T. F. Chan, G. H. Golub, and P. Mulet. A nonlinear primal-dual method for total variation-based image restoration. *SIAM J. Sci. Comp.*, 20(6):1964–77, 1999. 44

[12] P. Charbonnier, L. Blanc-Feraud, G. Aubert, and M. Barlaud. Deterministic edge-preserving regularization in computed imaging. *IEEE Trans. Im. Proc.*, 6(2):298–311, February 1997. 41

[13] P. Chatterjee and P. Milanfar. Is denoising dead? *IEEE Trans. Im. Proc.*, 19(4):985–911, April 2010. 35

[14] J. H. Cho and J. A. Fessler. Regularization designs for uniform spatial resolution and noise properties in statistical image reconstruction for 3D X-ray CT. *IEEE Trans. Med. Imag.*, 34(2):678–89, February 2015. 71

[15] N. H. Clinthorne, T. S. Pan, P. C. Chiao, W. L. Rogers, and J. A. Stamos. Preconditioning methods for improved convergence rates in iterative reconstructions. *IEEE Trans. Med. Imag.*, 12(1):78–83, March 1993. 15, 19, 69, 105

[16] J. Cui, G. Pratx, B. Meng, and C. S. Levin. Distributed MLEM: an iterative tomographic image reconstruction algorithm for distributed memory architectures. *IEEE Trans. Med. Imag.*, 32(5):957–67, May 2013. 87

[17] J. de Leeuw and K. Lange. Sharp quadratic majorization in one dimension. *Comp. Stat. Data Anal.*, 53(7):2471–84, May 2009. 98

[18] J. Eckstein and D. P. Bertsekas. On the Douglas-Rachford splitting method and the proximal point algorithm for maximal monotone operators. *Mathematical Programming*, 55(1-3):293–318, April 1992. 74, 100, 117, 118

[19] M. Elad and M. Aharon. Image denoising via sparse and redundant representations over learned dictionaries. *IEEE Trans. Im. Proc.*, 15(12):3736–45, December 2006. 1

[20] I. A. Elbakri and J. A. Fessler. Statistical image reconstruction for polyenergetic X-ray computed tomography. *IEEE Trans. Med. Imag.*, 21(2):89–99, February 2002. 69

[21] L. A. Feldkamp, L. C. Davis, and J. W. Kress. Practical cone beam algorithm. *J. Opt. Soc. Am. A*, 1(6):612–9, June 1984. 10

[22] J. A. Fessler. Statistical image reconstruction methods for transmission tomography. In M. Sonka and J. Michael Fitzpatrick, editors, *Handbook of Medical Imaging, Volume 2. Medical Image Processing and Analysis*, pages 1–70. SPIE, Bellingham, 2000. 69

[23] J. A. Fessler and S. D. Booth. Conjugate-gradient preconditioning methods for shift-variant PET image reconstruction. *IEEE Trans. Im. Proc.*, 8(5):688–99, May 1999. 11, 15, 19

[24] J. A. Fessler, N. H. Clinthorne, and W. L. Rogers. On complete data spaces for PET reconstruction algorithms. *IEEE Trans. Nuc. Sci.*, 40(4):1055–61, August 1993. 98

[25] J. A. Fessler, E. P. Ficaro, N. H. Clinthorne, and K. Lange. Grouped-coordinate ascent algorithms for penalized-likelihood transmission image reconstruction. *IEEE Trans. Med. Imag.*, 16(2):166–75, April 1997. 70

[26] J. A. Fessler and W. L. Rogers. Spatial resolution properties of penalized-likelihood image reconstruction methods: Space-invariant tomographs. *IEEE Trans. Im. Proc.*, 5(9):1346–58, September 1996. 38

[27] L. Fu, Z. Yu, J-B. Thibault, B. D. Man, M. G. McGaffin, and J. A. Fessler. Space-variant channelized preconditioner design for 3D iterative CT reconstruction. In *Proc. Intl. Mtg. on Fully 3D Image Recon. in Rad. and Nuc. Med*, pages 205–8, 2013. 15, 69

[28] D. Geman and G. Reynolds. Constrained restoration and the recovery of discontinuities. *IEEE Trans. Patt. Anal. Mach. Int.*, 14(3):367–83, March 1992. 41

[29] P. Grangeat. Mathematical framework of cone beam 3D reconstruction via the first derivative of the Radon transform. In 1990 Oberwolfach, editor, *Mathematical methods in tomography*, pages 66–95. Springer, Berlin, 1991. 10

[30] H. Guan and R. Gordon. A projection access order for speedy convergence of ART (algebraic reconstruction technique): a multilevel scheme for computed tomography. *Phys. Med. Biol.*, 39(11):2005–22, November 1994. 60

[31] H. Guan and R. Gordon. Computed tomography using algebraic reconstruction techniques (ARTs) with different projection access schemes: a comparison study under practical situations. *Phys. Med. Biol.*, 41(9):1727–43, September 1996. 60

[32] N. Halko, P. Martinsson, and J. Tropp. Finding structure with randomness: probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review*, 53(2):217–88, 2011. 17

[33] P. J. Huber. *Robust statistics*. Wiley, New York, 1981. 42

[34] D. R. Hunter and K. Lange. A tutorial on MM algorithms. *American Statistician*, 58(1):30–7, February 2004. 37

[35] M. W. Jacobson and J. A. Fessler. An expanded theoretical treatment of iteration-dependent majorize-minimize algorithms. *IEEE Trans. Im. Proc.*, 16(10):2411–22, October 2007. 6, 37, 41, 42, 77

[36] S. T. Jensen, S. Johansen, and S. L. Lauritzen. Globally convergent algorithms for maximizing a likelihood function. *Biometrika*, 78(4):867–77, December 1991. 41

[37] A. Katsevich. Theoretically exact filtered backprojection-type inversion algorithm for spiral CT. *SIAM J. Appl. Math.*, 62(6):2012–26, 2002. 10

[38] D. Kim and J. A. Fessler. Accelerated ordered-subsets algorithm based on separable quadratic surrogates for regularized image reconstruction in X-ray CT. In *Proc. IEEE Intl. Symp. Biomed. Imag.*, pages 1134–7, 2011. 13

[39] D. Kim and J. A. Fessler. Optimized momentum steps for accelerating X-ray CT ordered subsets image reconstruction. In *Proc. 3rd Intl. Mtg. on image formation in X-ray CT*, pages 103–6, 2014. 70, 89, 93

[40] D. Kim and J. A. Fessler. Distributed block-separable ordered subsets for helical X-ray CT image reconstruction. In *Proc. Intl. Mtg. on Fully 3D Image Recon. in Rad. and Nuc. Med*, 2015. To appear. 87, 97

[41] D. Kim and J. A. Fessler. Optimized gradient methods for smooth convex minimization. In *Intl. Symp. Math. Prog.*, 2015. Submitted. 107

[42] D. Kim, D. Pal, J-B. Thibault, and J. A. Fessler. Accelerating ordered subsets image reconstruction for X-ray CT using spatially non-uniform optimization transfer. *IEEE Trans. Med. Imag.*, 32(11):1965–78, November 2013. 36, 51, 54, 55, 59

[43] D. Kim, S. Ramani, and J. A. Fessler. Accelerating X-ray CT ordered subsets image reconstruction with Nesterov's first-order methods. In *Proc. Intl. Mtg. on Fully 3D Image Recon. in Rad. and Nuc. Med*, pages 22–5, 2013. 63

[44] D. Kim, S. Ramani, and J. A. Fessler. Combining ordered subsets and momentum for accelerated X-ray CT image reconstruction. *IEEE Trans. Med. Imag.*, 34(1):167–78, January 2015. 11, 12, 13, 35, 70, 72, 89, 93

[45] D. H. Kim. *Accelerated optimization algorithms for statistical 3D X-ray computed tomography image reconstruction*. PhD thesis, Univ. of Michigan, Ann Arbor, MI, 48109-2122, Ann Arbor, MI, 2014. 10

[46] K. Kohno, M. Kawamoto, and Y. Inouye. A matrix pseudoinversion lemma and its application to block-based adaptive blind deconvolution for MIMO systems. *IEEE Trans. Circ. Sys. I, Fundamental theory and applications*, 57(7):1499–1512, July 2010. 45

[47] K. Lange. Convergence of EM image reconstruction algorithms with Gibbs smoothing. *IEEE Trans. Med. Imag.*, 9(4):439–46, December 1990. Corrections, T-MI, 10:2(288), June 1991. 38, 43

[48] K. Lange, D. R. Hunter, and I. Yang. Optimization transfer using surrogate objective functions. *J. Computational and Graphical Stat.*, 9(1):1–20, March 2000. 77

[49] Y. Long, J. A. Fessler, and J. M. Balter. 3D forward and back-projection for X-ray CT using separable footprints. *IEEE Trans. Med. Imag.*, 29(11):1839–50, November 2010. 59, 89

[50] D. Matenine, S. Hissoiny, and P. Després. GPU-accelerated few-view CT reconstruction using the OSC and TV techniques. In *Proc. Intl. Mtg. on Fully 3D Image Recon. in Rad. and Nuc. Med*, pages 76–9, 2011. 84

[51] M. McGaffin and J. A. Fessler. Sparse shift-varying FIR preconditioners for fast volume denoising. In *Proc. Intl. Mtg. on Fully 3D Image Recon. in Rad. and Nuc. Med*, pages 284–7, 2013. 2, 14, 30

[52] M. McGaffin and J. A. Fessler. Edge-preserving image denoising via group coordinate descent on the GPU. *IEEE Trans. Im. Proc.*, 24(4):1273–81, April 2015. 2, 35, 64

[53] M. G. McGaffin and J. A. Fessler. Duality-based projection-domain tomography solver for splitting-based X-ray CT reconstruction. In *Proc. 3rd Intl. Mtg. on image formation in X-ray CT*, pages 359–62, 2014. 2, 54, 58, 72

[54] M. G. McGaffin and J. A. Fessler. Fast edge-preserving image denoising via group coordinate descent on the GPU. In *Proc. SPIE 9020 Computational Imaging XII*, page 90200P, 2014. 2, 35, 44, 64

[55] M. G. McGaffin and J. A. Fessler. Alternating dual updates algorithm for X-ray CT reconstruction on the GPU. *IEEE Trans. Comp. Imag.*, 2015. Submitted. 3, 69

[56] M. G. McGaffin and J. A. Fessler. Fast GPU-driven model-based X-ray CT image reconstruction via alternating dual updates. In *Proc. Intl. Mtg. on Fully 3D Image Recon. in Rad. and Nuc. Med*, 2015. To appear. 3, 69, 72

[57] M. G. McGaffin, S. Ramani, and J. A. Fessler. Reduced memory augmented Lagrangian algorithm for 3D iterative X-ray CT image reconstruction. In *Proc. SPIE 8313 Medical Imaging 2012: Phys. Med. Im.*, page 831327, 2012. 2, 14, 19, 35, 51, 54, 55

[58] M. J. Muckley, D. C. Noll, and J. A. Fessler. Fast parallel MR image reconstruction via B1-based, adaptive restart, iterative soft thresholding algorithms (BARISTA). *IEEE Trans. Med. Imag.*, 34(2):578–88, February 2015. 35, 98, 99

[59] Y. Nesterov. A method for unconstrained convex minimization problem with the rate of convergence $O(1/k^2)$. *Dokl. Akad. Nauk. USSR*, 269(3):543–7, 1983. 107

[60] Y. Nesterov. A method of solving a convex programming problem with convergence rate $O(1/k^2)$. *Soviet Math. Dokl.*, 27(2):372–76, 1983. 48

[61] Y. Nesterov. Smooth minimization of non-smooth functions. *Mathematical Programming*, 103(1):127–52, May 2005. 107

[62] H. Nien. *Model-based X-ray CT image and light field reconstruction using variable splitting methods*. PhD thesis, Univ. of Michigan, Ann Arbor, MI, 48109-2122, Ann Arbor, MI, 2014. 1, 10

[63] H. Nien and J. A. Fessler. Combining augmented Lagrangian method with ordered subsets for X-ray CT image reconstruction. In *Proc. Intl. Mtg. on Fully 3D Image Recon. in Rad. and Nuc. Med*, pages 280–3, 2013. 54, 55

[64] H. Nien and J. A. Fessler. Splitting-based statistical X-ray CT image reconstruction with blind gain correction. In *Proc. SPIE 8668 Medical Imaging 2013: Phys. Med. Im.*, page 86681J, 2013. 54, 55

[65] H. Nien and J. A. Fessler. Accelerating ordered-subsets X-ray CT image reconstruction using the linearized augmented Lagrangian framework. In *Proc. SPIE 9033 Medical Imaging 2014: Phys. Med. Im.*, page 903332, 2014. 54, 55, 70

[66] H. Nien and J. A. Fessler. Fast splitting-based ordered-subsets X-ray CT image reconstruction. In *Proc. 3rd Intl. Mtg. on image formation in X-ray CT*, pages 291–4, 2014. 54, 55

[67] H. Nien and J. A. Fessler. Fast X-ray CT image reconstruction using a linearized augmented Lagrangian method with ordered subsets. *IEEE Trans. Med. Imag.*, 34(2):388–99, February 2015. 11, 35, 70, 72, 97

[68] H. Nien, V. Sick, and J. A. Fessler. Model-based image reconstruction of chemiluminescence using plenoptic 2.0 camera. In *Proc. IEEE Intl. Conf. on Image Processing*, 2015. Submitted. 1

[69] M. Nikolova, M. K. Ng, and C-P. Tam. Fast nonconvex nonsmooth minimization methods for image restoration and reconstruction. *IEEE Trans. Im. Proc.*, 19(12):3073–88, December 2010. 38

[70] J. Nocedal and S. J. Wright. *Numerical optimization*. Springer, New York, 1999. 39

[71] J. P. Oliveira, J. M. Bioucas-Dias, and M. A. T. Figueiredo. Adaptive total variation image deblurring: A majorization-minimization approach. *Signal Processing*, 89(9):1683–93, September 2009. 43

[72] T. W. Parks and J. H. McClellan. Chebyshev approximation for nonrecursive digital filters with linear phase. *IEEE Trans. Circ. Theory*, 19(2):189–99, March 1972. 24

[73] L. Pfister and Y. Bresler. Adaptive sparsifying transforms for iterative tomographic reconstruction. In *Proc. 3rd Intl. Mtg. on image formation in X-ray CT*, pages 107–10, 2014. 70

[74] S. Ramani and J. A. Fessler. A splitting-based iterative algorithm for accelerated statistical X-ray CT reconstruction. *IEEE Trans. Med. Imag.*, 31(3):677–88, March 2012. 11, 15, 19, 35, 54, 55, 70, 72

[75] B. Recht and C. R. Parallel stochastic gradient algorithms for large-scale matrix completion. *Mathematical Programming Computation*, 5(2):201–26, June 2013. 17

[76] J. M. Rosen, J. Wu, T. F. Wenisch, and J. A. Fessler. Iterative helical CT reconstruction in the cloud for ten dollars in five minutes. In *Proc. Intl. Mtg. on Fully 3D Image Recon. in Rad. and Nuc. Med*, pages 241–4, 2013. 87

[77] L. I. Rudin, S. Osher, and E. Fatemi. Nonlinear total variation based noise removal algorithm. *Physica D*, 60(1-4):259–68, November 1992. 38

[78] W. P. Segars, M. Mahesh, T. J. Beck, E. C. Frey, and B. M. W. Tsui. Realistic CT simulation using the 4D XCAT phantom. *Med. Phys.*, 35(8):3800–8, August 2008. 19, 94

[79] S. Shalev-Shwartz and T. Zhang. Stochastic dual coordinate ascent methods for regularized loss minimization. *J. Mach. Learning Res.*, 14:567–99, February 2013. 75

[80] M. Sion. On general minimax theorems. *Pacific J. Math.*, 8(1):171–6, 1958. 7, 45

[81] J. W. Stayman and J. A. Fessler. Regularization for uniform spatial resolution properties in penalized-likelihood image reconstruction. *IEEE Trans. Med. Imag.*, 19(6):601–15, June 2000. 27, 51, 71

[82] H. Talebi and P. Milanfar. Global image denoising. *IEEE Trans. Im. Proc.*, 23(2):755–68, February 2014. 1

[83] J-B. Thibault, K. Sauer, C. Bouman, and J. Hsieh. A three-dimensional statistical approach to improved image quality for multi-slice helical CT. *Med. Phys.*, 34(11):4526–44, November 2007. 10, 37, 43, 51, 52, 69, 70

[84] T. Tsiligkaridis and A. O. Hero. Covariance estimation in high dimensions via kronecker product expansions. *IEEE Trans. Sig. Proc.*, 61(21):5347 – 5360, November 2013. 15

[85] A. S. Wang, J. W. Stayman, . Y. Otake, G. Kleinszig, S. Vogt, and J. H. Siewerdsen. Nesterov's method for accelerated penalized-likelihood statistical reconstruction for C-arm cone-beam CT. In *Proc. 3rd Intl. Mtg. on image formation in X-ray CT*, pages 409–13, 2014. 84

[86] J. Wang, T. Li, H. Lu, and Z. Liang. Penalized weighted least-squares approach to sinogram noise reduction and image reconstruction for low-dose X-ray computed tomography. *IEEE Trans. Med. Imag.*, 25(10):1272–83, October 2006. 64

[87] M. Wu and J. A. Fessler. GPU acceleration of 3D forward and backward projection using separable footprints for X-ray CT image reconstruction. In *Proc. Intl. Mtg. on Fully 3D Image Recon. in Rad. and Nuc. Med*, pages 56–9, 2011. 84

[88] J. Xu. *Modeling and development of iterative reconstruction algorithms in emerging X-ray imaging technologies*. PhD thesis, Washington University, St. Louis, May 2014. 87

[89] Z. Yu, L. Fu, D. Pal, J-B. Thibault, C. A. Bouman, and K. D. Sauer. Nested loop algorithm for parallel model based iterative reconstruction. In *Proc. Intl. Mtg. on Fully 3D Image Recon. in Rad. and Nuc. Med*, pages 197–200, 2013. 19

[90] Z. Yu, J-B. Thibault, C. A. Bouman, K. D. Sauer, and J. Hsieh. Fast model-based X-ray CT reconstruction using spatially non-homogeneous ICD optimization. *IEEE Trans. Im. Proc.*, 20(1):161–75, January 2011. 70, 90

[91] M. Zhu, S. Wright, and T. Chan. Duality-based algorithms for total-variation-regularized image restoration. *Comput. Optim. Appl.*, 47(3):377–400, 2010. 44