Distribution Agreement

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

Signature:

_____ _____

Abigail K. Julian Date

Automatic Hyperparameter Tuning in Physics-Based Distortion Correction for Diffusion Tensor Imaging

By

Abigail K. Julian
Doctor of Philosophy
Computer Science and Informatics

Lars Ruthotto, Ph.D.
Advisor

Joyce C Ho, Ph.D.
Committee Member

Deqiang Qiu, Ph.D.
Committee Member

Andreas Züfle, Ph.D.
Committee Member

Accepted:

Kimberly Jacob Arriola, Ph.D., MPH
Dean of the James T. Laney School of Graduate Studies

Date

Automatic Hyperparameter Tuning in Physics-Based Distortion Correction for Diffusion Tensor Imaging

By

Abigail K. Julian
B.A., Vanderbilt University, TN, 2019

Advisor: Lars Ruthotto, Ph.D.

An abstract of
A dissertation submitted to the Faculty of the
James T. Laney School of Graduate Studies of Emory University
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy
in Computer Science and Informatics
2024

Abstract

Automatic Hyperparameter Tuning in Physics-Based Distortion Correction for Diffusion Tensor Imaging
By Abigail K. Julian

Correction of susceptibility artifacts in Echo-Planar Imaging (EPI) is a computationally challenging problem due to its size, non-linearity, and scarcity of ground truth data. Although several post-processing tools for three-dimensional EPI distortion correction are available, these tools require choosing hyperparameters and are often slow, taking several minutes per image volume. In this dissertation, we develop a dependable, physics-based correction with automatic hyperparameter tuning in the context of Diffusion Tensor Imaging (DTI).

First, we build upon existing tools to solve the three-dimensional distortion correction problem in a matter of seconds using a separable optimization setup and highly parallelizable implementation. We implement an early one-dimensional correction approach as an initialization scheme using one-dimensional optimal transport, and implement an easy-to-use PyTorch tool that enables multi-threading and efficient use of graphics processing units (GPUs). Our extensive numerical validation using 3T and 7T data from the Human Connectome Project suggests that our tool achieves accuracy comparable to that of leading distortion correction tools at a fraction of the cost. We also validate the initialization scheme, compare different optimization algorithms, and test the algorithm on different hardware and arithmetic precision.

Second, we expand distortion correction to four-dimensional DTI volumes. In this setting, the three-dimensional brain volume is repeatedly imaged with a different diffusion gradient applied. Using the scalable initialization and optimization setup from the three-dimensional setting, we optimize in 4D using a parameterization of the additional diffusion dimension, leveraging the additional information offered by the associated directions of diffusion. We test a variety of parameterizations that relate the diffusion directions and introduce smoothness in the diffusion dimension. We also employ clustering to choose a subset of directions for optimization, and use the parameterization to interpolate on the original volume containing all of the diffusion directions.

Third and finally, we enable automatic hyperparameter tuning, possible because of the efficiency of four-dimensional distortion correction. We set up a bilevel optimization using metrics of DTI to tune the hyperparameters of the distortion correction problem. The resulting tool runs in times comparable to existing correction tools while not requiring the user to select any hyperparameters. Furthermore, the correction is dependable, since it is based on the interpretable physics of the distortion, and the correction simultaneously optimizes the metrics of the downstream task, diffusion tensor fit.

Automatic Hyperparameter Tuning in Physics-Based Distortion Correction for Diffusion Tensor Imaging

By

Abigail K. Julian
B.A., Vanderbilt University, TN, 2019

Advisor: Lars Ruthotto, Ph.D.

A dissertation submitted to the Faculty of the
James T. Laney School of Graduate Studies of Emory University
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy
in Computer Science and Informatics
2024

Acknowledgments

I owe many people immense thanks for their support, encouragement, and help during the last five years. First and foremost, thank you to my advisor Lars Ruthotto for his guidance throughout this process and always pushing me to grow. I have learned from him computational, writing, presentation, and life skills that I will take with me for the rest of my career. Thank you to the rest of my committee Joyce Ho, Deqiang Qiu, and Andreas Züfle; their varied expertise and perspectives have greatly enriched this work. Thank you to Siawoosh Mohammadi and Deqiang Qiu for their practical feedback from a radiology perspective.

Thank you to my academic family for their advice and camaraderie: Samy Wu Fung, Kelvin Kan, Xingjian Li, Malvern Madondo (the best office mate), Elizabeth Newman, Derek Onken, Haley Rosso, Deepanshu Verma, and Nicole Yang. A special thanks to Ramraj Chandradevan, Malvern Madondo, Vishwanath Seshagiri, Gary Vestal, and Ben Yellin for receiving my venting and celebrating over the years.

Off campus, I am indebted to the friends who have kept me grounded during graduate school: Heidi Barringer, Ethan and Elly Case, Josh and Lauren Casillas, Shelby and Kyle (and David) Langford, Lana McNair, Madison and Christian (and Nathaniel) Sanchez, Keaton and Delaney Scherpereel, Kent and Lynne Simpson, Cassie Taylor, and Wesley Toler.

My parents, Phil and Chris Roberts, have consistently helped me to pursue my gifts and passions. Thank you for always being a place of safety and encouragement.

I am incredibly grateful for my dear husband, Luke Julian. This season is full of sweet memories - moving to Atlanta a few weeks into marriage, enduring the pandemic, graduating law school and passing the bar, and building our life here. But I will cherish most the strength and support Luke has given me every single day, from listening to me vent about a stubborn bug in the code to celebrating each research breakthrough and milestone. He is the best.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Magnetic Resonance Imaging (MRI) is an imaging technique useful in clinical settings for patient diagnosis and care. MRI is also commonly used in research settings to better understand the human body and its functionality. One particular acquisition technique, called Echo-Planar Imaging (EPI), is promising because of its fast scan times, allowing for less time and strain for a patient, increased scan efficiency for healthcare systems, and the ability to quickly acquire a lot of data [58]. However, in trade-off for this speed of acquisition, EPI is known to leave distortion artifacts in the resulting images, distortions which are caused by inhomogeneities in the magnetic field due to susceptibility variations in the human body [15].

A highly efficient and accurate method of distortion correction would make EPI more attractive as an MR acquisition technique for both clinical and research settings. For example, in brain surgery to remove a tumor, a quick EPI-MRI scan during surgery could indicate to the surgical team if all of the tumor had been successfully removed before ending the surgery. In general, a faster MRI acquisition is easier on a patient. For research, being able to quickly acquire many highly-detailed images can facilitate larger-scale studies leveraging this ability to gather more imaging data. Increasing the field of view, the size of the region being captured in the image, of

the acquired image in EPI also increases the distortions in the resulting image, but effective distortion correction can allow for more highly-detailed images to be acquired and studied.

A common application of EPI-MRI using spin-echo sequences is Diffusion Tensor Imaging (DTI), a methodology to study how water moves throughout the brain. This can provide information on the structure and composition of the underlying tissues, which in turn can be used in a variety of neurological studies and treatments [39]. The data for DTI is a set of diffusion-weighted images acquired along at least six gradient directions. Then the elements of the diffusion tensor - an object giving the correlations between molecular movement in perpendicular directions - are estimated. The diffusion tensor can be used to calculate diffusion information such as fractional anisotropy and mean diffusivity, which describe the shape and amount of the diffusion. Because the diffusion tensor is estimated from the acquired image data, accurate images lead to a more accurate diffusion tensor and therefore more accurate diffusion analysis.

Susceptibility artifacts in EPI-MRI can be corrected by estimating the field map that gives the intensity of the magnetic field at scan time [15]. Despite the promise of the speed of EPI acquisition, the computation required for distortion correction makes it often a time-consuming step of MRI processing pipelines [14], and existing correction tools rely on the selection of hyperparameters.

Fast distortion correction without manually tuning hyperparameters would enable online distortion correction in applications where real-time decisions are necessary. For example, the speed of EPI acquisition along with fast distortion correction would enable real-time distortion-free imaging useful for intra-operative guidance (see, e.g., [30, 50, 66]). Additionally, fast distortion correction can be important for furthering emerging fields such as fetal and neonatal imaging (see, e.g., [43, 1, 17]). In this application, EPI is popular to reduce the effects of uncontrollable subject motion,

and fast distortion correction can enable faster intervention if necessary.

1.1 Contributions

In this dissertation, we develop a dependable, easy-to-use, physics-based correction of EPI distortions in DTI with automatic hyperparameter tuning. We focus on improving the speed and efficiency of EPI distortion correction and leverage DTI metrics to automate hyperparameter selection.

1.1.1 GPU-Enabled 3D Distortion Correction in Seconds

We begin by developing a tool for fast GPU-enabled distortion correction on three-dimensional EPI volumes. We introduce PyHySCO, an update to a well-known, reliable, and generalizable formulation, HySCO [52], that offers EPI distortion correction through a GPU-enabled and command line accessible Python tool. The physics-based field map estimation problem defined in HySCO has an interpretable formulation and separable structure [42] acknowledging the physics of the distortions. PyHySCO takes advantage of the separability of the problem formulation to speed up correction on both CPU and GPU with PyTorch multithreading [47]. The Python implementation allows for a simple command line interface compatible with existing MRI postprocessing pipelines. We additionally implement a parallelized one-dimensional initialization scheme, based on the correction of [15] interpreted through optimal transport, to speed up and improve the accuracy of distortion correction. Distortion correction with PyHySCO takes about 10 seconds on the GPU, including loading and saving the data.

1.1.2 Parameterized Four-Dimensional DTI Correction

Given the speed and efficiency of PyHySCO, we can extend distortion correction to the four-dimensional DTI setting where the three-dimensional brain volume is captured over a set of applied diffusion gradients [39]. Because the distortions occur only in the phase encoding dimension [15], the parallelization of computations in estimating the field map is unchanged with the addition of the diffusion dimension. We develop a parameterization of the four-dimensional field map to promote smoothness in the diffusion dimension and leverage the similarity of field maps for related diffusion directions. We explore different parameterizations and metrics of similarity, including using a nearest neighbors graph, spherical linear interpolation, and radial basis functions. Furthermore, we use the parameterized field map to reduce computation by optimizing over a subset of diffusion directions and interpolating the full field map from the optimized parameterization coefficients. A parameterized four-dimensional field map can improve both correction and DTI metrics when compared to the current approaches to DTI distortion correction that use a three-dimensional field map.

1.1.3 Automatic Hyperparameter Tuning

Because solving the distortion correction problem is fast, we can easily solve the the problem multiple times to tune the hyperparameters of distortion correction. We setup a bilevel optimization in the four-dimensional setting using DTI metrics as the outer problem and distortion correction as the inner problem. Using derivative-free optimization, we can improve distortion correction by finding the hyperparameters minimizing the loss value in the diffusion tensor fit problem and related metrics such as the standard deviation of fractional anisotropy maps between corrected image pairs. This results in a physics-based black-box correction, solving a reliable mathematical formulation of the correction problem while directly optimizing diffusion tensor metrics and not requiring the user to select any hyperparameters.

1.2 Overview

The dissertation is organized as follows. Chapter 2 introduces the notation and mathematical formulation for susceptibility artifact distortion correction. We define the reverse gradient polarity correction paradigm and the related separable optimization problem that we solve. We additionally define the diffusion tensor fit problem used in the four-dimensional setting and the derivative-free optimization used in the bilevel setup. Chapter 3 describes PyHySCO, our GPU-enabled PyTorch tool for distortion correction. We implement a parallelized one-dimensional correction as an initialization scheme using tools from optimal transport. We describe the easy-to-use and modular PyTorch implementation of PyHySCO, and we demonstrate the quality and speed of correction on CPU and GPU across several datasets of real and simulated EPI data. Chapter 4 describes the extension of the correction problem to the four-dimensional DTI setting. We develop a class of field map parameterizations that introduce smoothness and improve the distortion correction by leveraging diffusion information. We demonstrate the speed and quality of the four-dimensional parameterized distortion correction on a variety of diffusion-weighted EPI volumes. Chapter 5 proposes a bilevel optimization setup for four-dimensional distortion correction using DTI metrics to tune hyperparameters of the correction problem. We demonstrate the quality of the bilevel optimization in improving metrics of both the correction problem and DTI. In Chapter 6 we provide a conclusion and discussion of future directions for this work.

Chapter 2

Background and Preliminaries

In this chapter, we define the notation and mathematical formulation of field map estimation and distortion correction. In 2.1 we define Reversed Gradient Polarity Correction and review related approaches to the distortion correction problem. In 2.2, we introduce the physical model of EPI distortion correction and in 2.3 define the separable optimization problem that we solve. In 2.4, we describe several optimization schemes used in distortion correction. We additionally define in 2.5 the diffusion tensor problem used in the four-dimensional setting and in 2.6 bilevel and derivative-free optimization.

2.1 Reversed Gradient Polarity Correction

Reversed Gradient Polarity (RGP) methods are commonly used to correct susceptibility artifacts in Echo-Planar Imaging (EPI) [58]. RGP methods acquire a pair of images with opposite phase encoding directions, which leads to a minimal increase in scan time due to the speed of EPI. In a post-processing step, RGP approaches use the fact that the distortion in both images has an equal magnitude but acts in opposite directions to estimate the field map (see Figure 2.1) [15, 11]. The field map is then used to estimate a distortion-free image.

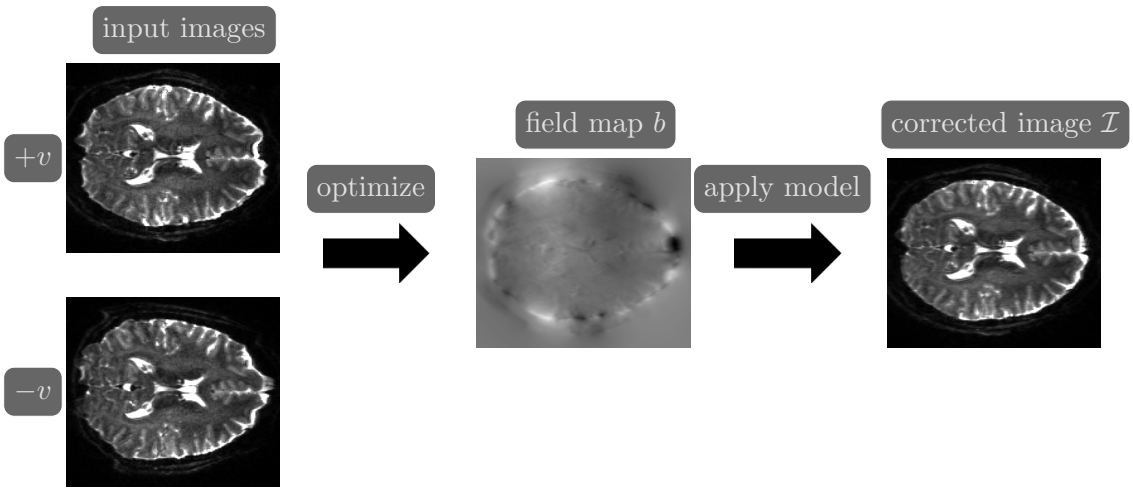


Figure 2.1: The Reverse Gradient Polarity correction paradigm. Two images are acquired with opposite phase encoding directions, $+v$ and $-v$. These two images are used to estimate the field map b, and the distortion correction model [15] is applied to obtain a corrected image $\mathcal{I}$.

Compared to other correction approaches such as field map acquisition, point-spread function map acquisition, and anatomical registration, RGP methods generally achieve comparable or superior accuracy while being more robust to noise and motion; see, e.g., [60, 28, 26, 65]. These advantages make RGP correction a popular choice. For example, the widely-used MRI database from the Human Connectome Project (HCP) [62] used the RGP correction tool TOPUP [4] in the preprocessing of released diffusion MRI from EPI scans.

The original RGP distortion correction approaches in [15, 11] are one-dimensional, treating each image column separately in the phase encoding direction. This leads to a non-smooth field map estimate and corrections. TOPUP addresses this non-smoothness with a 3D spline-based approach and the introduction of regularization [4]. TOPUP has limited support for hyperthreading and is often a time-consuming step of MRI processing pipelines [14]. Running TOPUP on a standard CPU took over 60 minutes on average per HCP subject in our experiments.

Although less widely used than TOPUP, other iterative methods have proposed implementations of RGP correction employing various optimization schemes, dis-

cretizations, and regularization terms to speed up the correction. EPIC [34] introduces correction using a nonlinear image registration framework. The tool was developed specifically for Anterior-Posterior distortions and can be less effective for Left-Right distortions [29]. DR-BUDDI [36] and TISAC [25] regularize the optimization using a T2-weighted or T1-weighted image, respectively. Including undistorted anatomical information can improve the quality of distortion correction [29], but complicates the choice of an effective distance measure and, depending on the protocol, may require additional scan time. HySCO introduces hyper-elastic registration regularization and a novel separable discretization [51, 52, 42]. HySCO can accurately correct real and simulated data varying in phase encoding direction, anatomy, and field of view [29, 57, 60]. In our experiments, on average, HySCO runs on the CPU for one to two minutes per HCP subject. While HySCO is a Statistical Parametric Mapping (SPM) [48] plugin and has been integrated into several SPM-based DTI processing pipelines; see, e.g., [21, 19], its dependency on a MATLAB license may limit its wider use.

Motivated by the long processing times of the above RGP tools, several deep learning approaches for susceptibility artifact correction have been proposed recently; see, e.g., [35, 24, 23, 67, 3]. A recurrent theme is to train a correction operator in an offline stage in a supervised way using training data, which enables fast evaluations in the online step. For example, training S-Net on 150 volumes took over 5 days, while correcting an image pair on a CPU took an average of 2.8 seconds (0.96 seconds on a GPU) [24]. However, the dramatic reduction of correction time comes at the cost of losing the robustness and generalizability that existing RGP approaches obtain from the physical distortion model. For example, while RGP approaches can handle images from different scanners, anatomies, resolutions, and other acquisition parameters, deep learning models perform poorly when applied outside the training distribution, [16]. Furthermore, deep learning models are highly sensitive to noise and adversarial

attacks in other contexts [6].

2.2 Susceptibility Artifact Distortion Correction Model

We now review the physical forward model defined in [15] for three-dimensional images. The field map estimation and distortion correction is based on this model. Let $v \in \mathbb{R}^3$ be the phase encoding direction for the distorted observation $\mathcal{I} : \Omega \to \mathbb{R}$, and let $\Omega \subset \mathbb{R}^3$ be the image domain of interest. The mass-preserving transformation operator that, given the field map $b : \Omega \to \mathbb{R}$, corrects the distortions of an image $\mathcal{I}$ acquired with phase encoding direction v reads

$$\mathcal{T}[\mathcal{I}, b, v](x) = (\mathcal{I}(x + b(x)v) \cdot (1 + \partial_v b))(x) \quad \forall x \in \Omega, \tag{2.1}$$

where $\partial_v b$ is the directional derivative of b in the direction of v. The operator has two parts, the first handling the geometric deformation in the direction of v and the second an intensity modulation term, which should always be positive.

2.3 Separable Optimization Problem

We estimate the field map using the formulation of HySCO [51, 52, 42], so we review the formulation here. The inverse problem, as defined in [51], estimates the field map b based on two observations, $\mathcal{I}_{+v}$ and $\mathcal{I}_{-v}$, acquired with phase encoding directions $\pm v$, respectively. To this end, we estimate the field map b by minimizing the distance of the corrected images

$$\mathcal{D}(b) = \frac{1}{2} \int_\Omega \left(\mathcal{T}[\mathcal{I}_{+v}, b, v](x) - \mathcal{T}[\mathcal{I}_{-v}, b, -v](x)\right)^2 \, \mathrm{d}x.$$

The distance term is additionally regularized to enforce smoothness and the intensity modulation constraint. The smoothness regularization term,

$$\mathcal{S}(b) = \frac{1}{2}\int_{\Omega} ||\nabla b(x)||^2 \mathrm{d}x,$$

penalizes large values of the gradient of b to ensure smoothness in all directions.

The intensity modulation constraint of the physical model requires that $-1 < \partial_v b(x) < 1$ for almost all $x \in \Omega$. This is enforced by the barrier term

$$\mathcal{P}(b) = \frac{1}{2}\int_{\Omega} \phi(\partial_v b(x))\mathrm{d}x, \text{ where } \phi(z) = \frac{z^4}{1-z^2}. \tag{2.2}$$

All together, this gives the optimization problem

$$\min_b \mathcal{J}(b) = \mathcal{D}(b) + \alpha\mathcal{S}(b) + \beta\mathcal{P}(b), \tag{2.3}$$

where the importance of the regularization terms is weighted with non-negative scalars α and β. Higher values of α can promote a smoother field map, while lower values of α promote reduced distance between corrected images at the expense of smoothness in the field map. Any positive value for β ensures the intensity modulation constraint is satisfied, but lower values can lead to more ill-conditioned problems.

Our work follows the discretize-then-optimize paradigm commonly used in image registration; see, e.g., [45]. We discretize the variational problem (2.3) as in [42] to obtain a finite-dimensional optimization problem almost entirely separable in the phase encoding direction. Specifically, coupling is only introduced in the smoothness regularization term when calculating the gradient in the frequency encoding and slice selection directions.

Our convention is to permute the dimensions of the input image such that the phase encoding direction is aligned with the third unit vector $e_3 = [0, 0, 1]^T$. The

field map is discretized on an e_3-staggered grid; that is, we discretize its values in the cell centers along the first two dimensions and on the nodes in the third dimension. The integrals in (2.3) are approximated by a midpoint quadrature rule. The input images are modeled by a one-dimensional piecewise linear interpolation function in the phase encoding direction. The geometric transformation is estimated in the cell centers with an averaging operator, and the intensity modulation is estimated in the cell centers with a finite difference operator.

The discretized smoothness regularization term is computed for the discretized field map $\mathbf{b}$ via

$$S(\mathbf{b}) = \frac{h_1 \cdot h_2 \cdot h_3}{2} \mathbf{b}^\top H \mathbf{b} = \frac{h_1 \cdot h_2 \cdot h_3}{2} ||\mathbf{b}||_H^2, \tag{2.4}$$

where h_1, h_2, h_3 are the voxel sizes and H is a standard five-point discretization of the negative Laplacian and thus is a positive semi-definite operator. The discretized intensity modulation constraint term applies ϕ as defined in (2.2) element-wise to the result of a finite difference operator applied to the discretized field map. This gives the discretized optimization problem to solve as

$$\min_{\mathbf{b}} J(\mathbf{b}) = D(\mathbf{b}) + \alpha S(\mathbf{b}) + \beta P(\mathbf{b}). \tag{2.5}$$

This problem is challenging to solve because it is high-dimensional and non-convex, but we can exploit the structure and separability to efficiently solve the problem using parallelization. The implementation of this optimization problem in a parallelizable way, as described in 3.3, includes choices of image interpolation, linear operators for averaging and finite difference, and regularization terms S and P.

2.4 Optimization Schemes

We describe here three optimization schemes used in this dissertation to solve (2.5): Gauss Newton, ADMM, and LBFGS. The optimizers are described here in the context of this optimization problem and as implemented in previous work. In 3.2 we describe modifications and details of our implementation of the optimizers in this work.

Gauss Newton

The optimization scheme in [52, 51] is Gauss Newton (Algorithm 1) with a preconditioned conjugate gradient (PCG) linear solver (Algorithm 2) [46, Ch. 7 p. 168-170]. Following the general idea of Gauss-Newton, we linearize the (nonlinear) distortion correction operator (2.1) about the k-th iterate $\mathbf{b}_k$, obtain a quadratic model for the objective function by using a second-order Taylor approximation, and update the field map estimate with its approximate solution obtained with a few iterations of the PCG method.

More precisely, let ∇J be the gradient and H_J be a positive definite approximation of the Hessian of the optimization problem (2.5) about $\mathbf{b}_k$. Gauss Newton iteratively updates the current field map estimate via

$$\mathbf{b}_{k+1} = \mathbf{b}_k + \gamma_k \mathbf{q}_k,$$

where the step size γ_k is determined with a line search method such as Armijo [46, Ch. 3 p. 33-36] and the search direction $\mathbf{q}_k$ approximately satisfies

$$H_J(\mathbf{b}_k)\mathbf{q}_k = -\nabla J(\mathbf{b}_k). \tag{2.6}$$

The Newton system (2.6) is solved using the preconditioned conjugate gradient (PCG) method [33, 53]. Conjugate gradient iteratively updates the solution along a set of

Algorithm 1 Gauss Newton

Input: $\mathbf{b}_0$; maximum iterations N
Output: $\mathbf{b}$, solution to (2.5)
$k \leftarrow 0$
while $k < N$ **do**
  Compute $\mathbf{q}_k = \arg\min_{\mathbf{q}} ||H_J(\mathbf{b}_k)\mathbf{q} + \nabla J(\mathbf{b}_k)||^2$
  Compute step size γ_k using line search
  $\mathbf{b}_{k+1} \leftarrow \mathbf{b}_k + \gamma_k \mathbf{q}_k$
  $k \leftarrow k + 1$
end while

conjugate directions until the residual $\mathbf{r}_j$ of (2.6) reaches a specified tolerance. At iteration j, the solution $\mathbf{q}_j$ and conjugate direction $\mathbf{p}_j$ are updated as

$$\mathbf{q}_{j+1} = \mathbf{q}_j + \tau_j \mathbf{p}_j, \quad \text{and} \quad \mathbf{p}_{j+1} = \mathbf{r}_{j+1} + \nu_{j+1}\mathbf{p}_j, \tag{2.7}$$

where τ_j is a step size from exact line search and ν_{j+1} ensures the conjugacy of the directions $\mathbf{p}$. The performance of PCG crucially depends on the clustering of the eigenvalues, which a suitable preconditioner M can often improve. As a computationally inexpensive and often effective option, we implement a Jacobi preconditioner, which approximates the inverse of H_J by the inverse of its diagonal entries. While the diagonal preconditioner works well in our examples, we note that a more accurate (yet also more expensive) block-diagonal preconditioner has been proposed in [42].

ADMM

The optimization problem (2.5) is solved in [42] using an Alternating Direction Method of Multipliers (ADMM) [12] setup (Algorithm 3). To take advantage of the separability of the objective function, the idea is to split the optimization problem into two subproblems. Split (2.5), removing for now the intensity modulation

Algorithm 2 Preconditioned Conjugate Gradient

Input: tolerance $\epsilon > 0$; maximum iterations N
Output: $\mathbf{q}$, solution to (2.6) at iterate $\mathbf{b}_k$
$j \leftarrow 0$; $\mathbf{r}_0 \leftarrow -\nabla J(\mathbf{b}_k)$; $\mathbf{q}_0 \leftarrow \mathbf{0}$
Solve $M\mathbf{z}_0 = \mathbf{r}_0$; $\mathbf{p}_0 \leftarrow \mathbf{z}_0$
while $j < N$ or $\mathbf{r}_j/\mathbf{r}_0 < \epsilon$ **do**
  $\tau_j \leftarrow (\mathbf{r}_j^T \mathbf{z}_j)/(\mathbf{p}_j^T H \mathbf{p}_j)$
  $\mathbf{q}_{j+1} \leftarrow \mathbf{q}_j + \tau_j \mathbf{p}_j$
  $\mathbf{r}_{j+1} \leftarrow \mathbf{r}_j - \tau_j H \mathbf{p}_j$
  Solve $M\mathbf{z}_{j+1} = \mathbf{r}_{j+1}$
  $\nu_{j+1} =\leftarrow (\mathbf{z}_{j+1}^T \mathbf{r}_{j+1})/(\mathbf{r}_j^T \mathbf{z}_j)$
  $\mathbf{p}_{j+1} \leftarrow \mathbf{z}_{j+1} + \nu_{j+1}\mathbf{p}_j$
  $j \leftarrow j + 1$
end while

constraint term, into

$$F(\mathbf{b}) = D(\mathbf{b}) + \alpha S_3(\mathbf{b}), \quad \text{and} \quad G(\mathbf{z}) = \alpha S_1(\mathbf{z}) + \alpha S_2(\mathbf{z}), \tag{2.8}$$

where S_3 is the part of the smoothness regularization term S corresponding to the phase encoding direction, and S_1 and S_2 are the remaining terms corresponding to the other directions. Define the indicator function ι_C for the intensity modulation constraint with $C = \{\mathbf{b} \ : -1 \leq \mathbf{D}_3\mathbf{b} \leq 1\}$ and returning 0 whenever $\mathbf{b} \in C$, and ∞ otherwise. This gives rise to the following optimization problem, equivalent to (2.5):

$$\min_{\mathbf{b},\mathbf{z}} \iota_C(\mathbf{b}) + F(\mathbf{b}) + G(\mathbf{z}) \ \text{ s.t. } \ \mathbf{b} = \mathbf{z}.$$

With the corresponding augmented Lagrangian

$$L(\mathbf{b}, \mathbf{z}, \mathbf{y}) = \iota_C(\mathbf{b}) + F(\mathbf{b}) + G(\mathbf{z}) + \mathbf{y}^T(\mathbf{b} - \mathbf{z}) + \frac{\rho h^3}{2}||\mathbf{b} - \mathbf{z}||^2,$$

where $\mathbf{y}$ is the Lagrange multiplier for the equality constraint $\mathbf{b} = \mathbf{z}$, h is the voxel size, and ρ is a scalar augmentation parameter. Using scaled dual variable $\mathbf{u} = \frac{\mathbf{y}}{\rho h^3}$,

each iteration has the updates

$$\mathbf{b}_{k+1} = \arg\min_{\mathbf{b}} \iota_C(\mathbf{b}) + F(\mathbf{b}) + \frac{\rho h^3}{2}||\mathbf{b} - \mathbf{z}_k + \mathbf{u}_k||^2 \tag{2.9}$$

$$\mathbf{z}_{k+1} = \arg\min_{\mathbf{z}} G(\mathbf{z}) + \frac{\rho h^3}{2}||\mathbf{b}_{k+1} - \mathbf{z} + \mathbf{u}_k||^2 \tag{2.10}$$

$$\mathbf{u}_{k+1} = \mathbf{u}_k + \mathbf{b}_{k+1} - \mathbf{z}_{k+1}. \tag{2.11}$$

The $\mathbf{b}$ update in (2.9) is approximately solved with Sequential Quadratic Programming [10], a quasi-Newton method for constrained nonlinear optimization. At each iteration, the current iterate is updated to the solution of solving the quadratic approximation of the objective function subject to the intensity modulation constraint. This is done using the active set method with Schur complement solver described in [46, Ch. 16, p. 455-480]. Due to the separability of this update and its block diagonal Hessian, solving the linear system in the Schur complement involving the Hessian can be done in parallel.

The $\mathbf{z}$ update is computed by solving (2.10) directly. The closed form solution is

$$\mathbf{z}_{k+1} = \mathbf{K}^{-1}(\rho h^3(\mathbf{b}_{k+1} + \mathbf{u}_k)),$$

where $\mathbf{K}$ is the matrix $\alpha\mathbf{L}^T\mathbf{L} + \rho\mathbf{I}$ with $\mathbf{L}$ the sum of the Laplacian matrices of S_1 and S_2 and $\mathbf{I}$ the identity matrix. This inverse and product is efficiently computed using the factorization

$$\mathbf{K} = \mathbf{Q}^H\mathbf{D}\mathbf{Q}, \tag{2.12}$$

where $\mathbf{Q}$ is a discrete Fourier transform matrix with periodic boundary conditions, and $\mathbf{D}$ is diagonal with the eigenvalues of $\mathbf{K}$ as its elements [31].

The augmentation parameter ρ is updated adaptively as described in [12] to keep the relative primal and dual residuals close.

Algorithm 3 ADMM

Input: $\mathbf{b}_0$; maximum iterations N
Output: $\mathbf{b}$, solution to (2.5)
$k \leftarrow 0$; $\mathbf{z}_0 \leftarrow \mathbf{b}_0$; $\mathbf{u}_0 \leftarrow \mathbf{0}$
while $k < N$ **do**
  Compute $\mathbf{b}_{k+1} = \arg\min_{\mathbf{b}} L_{\mathbf{b}}(\mathbf{b}, \mathbf{z}_k, \mathbf{u}_k)$
  Compute $\mathbf{z}_{k+1} = \arg\min_{\mathbf{z}} L_{\mathbf{z}}(\mathbf{b}_{k+1}, \mathbf{z}, \mathbf{u}_k)$
  $\mathbf{u}_{k+1} \leftarrow \mathbf{u}_k + \mathbf{b}_{k+1} - \mathbf{z}_{k+1}$
  $k \leftarrow k + 1$
end while

Algorithm 4 LBFGS

Input: $\mathbf{b}_0$; maximum iterations N; history size m
Output: $\mathbf{b}$, solution to (2.5)
$k \leftarrow 0$
while $k < N$ **do**
  Compute $\mathbf{p}_k$ using Two-Loop Recursion
  Compute step size γ_k using line search
  $\mathbf{b}_{k+1} = \mathbf{b}_k + \alpha_k \mathbf{p}_k$
  if $k > m$ **then**
    Discard $\mathbf{s}_{k-m}$ and $\mathbf{y}_{k-m}$ from history
  end if
  $\mathbf{s}_{k+1} \leftarrow \mathbf{b}_{k+1} - \mathbf{b}$; $\mathbf{y}_{k+1} \leftarrow \nabla J(\mathbf{b}_{k+1}) - \nabla J(\mathbf{b}_k)$
  $k \leftarrow k + 1$
end while

LBFGS

The final optimization scheme we describe for solving (2.5) is a limited-memory Broyden–Fletcher–Goldfarb–Shanno (LBFGS) algorithm [40] (Algorithm 4). LBFGS is a quasi-Newton method that uses an estimate of the inverse of the objective function's Hessian based on a limited number of previous iterations in solving for the search direction. Using a step size γ_k computed using line search, the update at iteration k is

$$\mathbf{b}_{k+1} = \mathbf{b}_k + \gamma_k \mathbf{p}_k,$$

where the search direction $\mathbf{p}_k$ is solved for using a two-loop recursion (Algorithm 5) that recursively estimates the inverse Hessian.

Algorithm 5 LBFGS Two-Loop Recursion

$\mathbf{p} \leftarrow -\nabla J(\mathbf{b}_k)$
for $i = k-1, k-2, \cdots k-m$ **do**
  $\tau_i \leftarrow (\mathbf{s}_i^T \mathbf{p})/(\mathbf{s}_i^T \mathbf{y}_i)$
  $\mathbf{p} \leftarrow \mathbf{p} - \tau_i \mathbf{y}_i$
end for
$\mathbf{p} \leftarrow (\mathbf{s}_{k-1}^T \mathbf{y}_{k-1})/(\mathbf{y}_{k-1}^T \mathbf{y}_{k-1})\mathbf{p}$
for $i = k-m, k-m+1, \cdots, k-1$ **do**
  $\nu \leftarrow (\mathbf{y}_i^T \mathbf{p})/(\mathbf{s}_i^T \mathbf{y}_i)$
  $\mathbf{p} \leftarrow \mathbf{p} + (\tau_i - \nu)\mathbf{s}_i$
end for

2.5 Diffusion Tensor Imaging

In diffusion weighted imaging, the scan of the three-dimensional volume of interest, such as the human brain, is repeated with varying diffusion sensitization gradients applied [39]. These gradients are defined by the direction $\mathbf{g}$ and weighting parameter, called a b-value, γ. Together, call the directions defining the diffusion sensitization gradients as $\mathbf{d} = \gamma \mathbf{g}$.

One model to study diffusion information is the Stejskal-Tanner equation [59], which models how the intensity at an image voxel decays in the presence of a diffusion gradient. Let $\mathcal{I}^{(0)}$ be one or more images without diffusion weighting and $\mathcal{I}^{(k)}$ be an image acquired with diffusion gradient $\mathbf{g_k}$ and b-value γ_k. For a given voxel x in the image domain $\Omega \subset \mathbb{R}^3$ and given the symmetric 3x3 diffusion tensor for this voxel $\mathcal{D}(x)$, the Stejskal-Tanner equation [59] is

$$\mathcal{I}^{(k)}(x) = \mathcal{I}^{(0)}(x) \exp(-\gamma_k \cdot \mathbf{g}_k^T \mathcal{D}(x) \mathbf{g}_k). \tag{2.13}$$

The symmetric 3x3 diffusion tensor $\mathcal{D}$ is unknown, but the six independent values of the tensor can be estimated for each voxel using (2.13). In particular, given at least six diffusion-weighted images along with their diffusion gradient information, estimating $\mathcal{D}$ can be considered an overdetermined least squares problem [61]. Suppose there are

$n_d > 6$ diffusion directions $\{\mathbf{g}_k\}_{k=1}^{n_d}$. Let A_k of size $n_d \times m$ be n_d diffusion weighted images of size m, and A_0 be a non-diffusion weighted image. Let $\gamma \in \mathbb{R}^{1\times n_d}$ be all of the associated b-values and $\mathbf{g} \in \mathbb{R}^{3\times n_d}$ the n_d diffusion directions. Taking the logarithm of both sides of (2.13), the problem becomes

$$\log A_0(x) - \log A_k(x) = \gamma \cdot \mathbf{g}^T \mathcal{D}(x)\mathbf{g}.$$

Represent the six independent values of $\mathcal{D}(x)$ as

$$\mathcal{D}_6(x) = [\mathcal{D}(x)[1,1], \mathcal{D}(x)[1,2], \mathcal{D}(x)[1,3], \mathcal{D}(x)[2,2], \mathcal{D}(x)[2,3], \mathcal{D}(x)[3,3]],$$

and using the symmetry of $\mathcal{D}$, the right hand side can be expressed as

$$\gamma \cdot \mathbf{g}^T \mathcal{D}(x)\mathbf{g} = \gamma \cdot [\mathbf{g}[1]^2, 2\mathbf{g}[1]\mathbf{g}[2], 2\mathbf{g}[1]\mathbf{g}[3], \mathbf{g}[2]^2, 2\mathbf{g}[2]\mathbf{g}[3], \mathbf{g}[3]^2] \cdot \mathcal{D}_6(x),$$

where $\mathbf{g}[1]$, $\mathbf{g}[2]$, and $\mathbf{g}[3]$ give the first, second, and third rows of $\mathbf{g}$, respectively. This least squares formulation can be solved in parallel for each voxel x.

One metric of correction quality in the diffusion setting is the diffusion tensor fit loss where A_k and A_0 include the corrected images from the input pair with phase encoding directions $+v$ and $-v$. The quality of the diffusion tensor fit will improve when the corrected images are closer, giving more consistent information about how the diffusion gradient has affected the image. Therefore a metric of both distortion correction quality and diffusion tensor quality is the diffusion tensor loss defined as

$$L_{dti}(\mathcal{D}(x)) = \frac{1}{2}||\gamma \cdot \mathbf{g}^T \mathcal{D}(x)\mathbf{g} - (\log A_0(x) - \log A_k(x))||^2. \tag{2.14}$$

As with the least squares solution, the diffusion tensor loss is computed in parallel for each voxel x.

Another common metric of quality in the diffusion tensor imaging setting is the median standard deviation of the fractional anisotropy (FA) maps for the pair of corrected images with phase encoding directions $+v$ and $-v$ [29, 65]. Fractional anisotropy is a measure computed from the eigenvalues of $\mathcal{D}$ that indicates the anisotropy (non-uniformity in different directions) of the diffusion at each voxel. Let $\lambda_1, \lambda_2, \lambda_3$ be the eigenvalues of $\mathcal{D}(x)$, computed for a single phase encoding direction input volume. Then the fractional anisotropy value at voxel x is

$$fa(x) = \sqrt{\frac{(\lambda_1 - \lambda_2)^2 + (\lambda_2 - \lambda_3)^2 + (\lambda_3 - \lambda_1)^2}{2(\lambda_1^2 + \lambda_2^2 + \lambda_3^2)}}. \tag{2.15}$$

The fractional anisotropy map is computed in parallel for each voxel x, using the diffusion tensor for each input volume. A lower median standard deviation between the fractional anisotropy maps for images with phase encoding directions $+v$ and $-v$ indicates more consistency and therefore confidence in the diffusion tensor information. In Chapters 4 and 5, we use both the diffusion tensor loss (2.14) and median standard deviation of the fractional anisotropy in (2.15) as metrics in evaluating four-dimensional correction quality.

2.6 Bilevel and Derivative-Free Optimization

We conclude the chapter by describing bilevel optimization and an approach for solving the bilevel problem with derivative-free optimization. Consider the following general bilevel optimization problem

$$\min_x f(x, y^*(x)) \quad \text{s.t.} \quad y^*(x) \in \arg\min_y g(x, y).$$

The objective f is the outer objective, and the objective g is the inner objective. The bilevel structure couples the two optimization problems and indicates a dependency

between the variables x and y. See [22] for a survey of approaches and applications in bilevel optimization.

We consider the case where the relationship between x and y is not explicitly given, i.e. the derivative of $f(x, y^*(x))$ with respect to x cannot be easily computed because of the dependence of x on y^*. In this case, a derivative-free (sometimes called black-box) optimization is necessary. A popular family of derivative-free methods is Bayesian optimization [44].

In Bayesian optimization, the objective f is treated as a random function with an associated prior distribution. As the objective is evaluated, the prior is updated to form a posterior distribution, and the next value of the optimization parameter is chosen by a sampling method informed by the posterior. Specifically, Bayesian optimization requires a method to model the probability of an objective function value z given the input x and a set of observed function evaluations $\mathcal{Z}$, notated as

$$p(z|x, \mathcal{Z}).$$

One such model of $p(z|x, \mathcal{Z})$ is the Tree-structured Parzen Estimator (TPE) [7] (Algorithm 6). TPE splits the N observed objective function values into a 'low' group $\mathcal{Z}^{(l)} = \{(x_n, z_n)\}_{n=1}^{N^{(l)}}$ and a 'high' group $\mathcal{Z}^{(h)} = \{(x_n, z_n)\}_{n=N^{(l)}}^{N}$ such that

$$p(z|x, \mathcal{Z}) = \begin{cases} p(x|\mathcal{Z}^{(l)}), & z \leq z_q \\ p(x|\mathcal{Z}^{(h)}), & z > z_q, \end{cases}$$

where z_q is the function value defining the boundary between the two sets. The two

Algorithm 6 Tree-structured Parzen Estimator

Input: observations $\mathcal{Z}$
Output: solution x
Compute z_q and split $\mathcal{Z}$ into $\mathcal{Z}^{(l)}, \mathcal{Z}^{(h)}$
Compute weights $\{w_n\}_{n=1}^{N}$ and bandwidth $b^{(l)}, b^{(h)}$
Build $p(x|\mathcal{Z}^{(l)}), p(x|\mathcal{Z}^{(h)})$
Sample $\mathcal{S} = \{x_s\}_{s=1}^{N_s}$ $p(x|\mathcal{Z}^{(l)})$
Pick $x = \arg\max_{x \in \mathcal{S}}(p(x|\mathcal{Z}^{(l)})/p(x|\mathcal{Z}^{(h)}))$

kernel density estimators are computed as

$$p(x|\mathcal{Z}^{(l)}) = w_0^{(l)} p_0(x) + \sum_{n=1}^{N^{(l)}} w_n k(x, x_n|b^{(l)}),$$
$$p(x|\mathcal{Z}^{(h)}) = w_0^{(h)} p_0(x) + \sum_{n=N^{(l)}+1}^{N} w_n k(x, x_n|b^{(h)}),$$

where $\{w_n\}$ are weights, k is a kernel function, $b^{(l)}$ and $b^{(h)}$ are bandwidth parameters, and p_0 is a prior. See [64] for a tutorial on TPE including discussion of how the choice of these parameters in the TPE model trade off between exploration and exploitation. After sampling N_s times from $p(x|\mathcal{Z}^{(l)})$, the solution is taken as the sampled x maximizing the quotient $p(x|\mathcal{Z}^{(l)})/p(x|\mathcal{Z}^{(h)})$.

TPE can be easily applied to a bilevel optimization problem, and is a popular choice for hyperparameter optimization algorithms; see e.g. [2, 8].

Chapter 3

GPU-Enabled 3D Distortion Correction in Seconds

In this chapter, we describe a software tool, PyHySCO [37], developed as an update to HySCO [52] to take advantage of modern GPU hardware and proposing an improved initialization scheme. PyHySCO is implemented in PyTorch [47] and leverages the separability of the optimization problem to enable multithreading in PyTorch to speed up computation.

In Section 3.1, we introduce an initialization scheme based on the correction of Chang and Fitzpatrick [15] and implemented using parallelized one-dimensional optimal transport maps. In Section 3.2, we provide additional details on the optimization schemes implemented in PyHySCO, particularly how they may vary from the traditional setups described in Chapter 2. In Section 3.3, we describe the PyTorch implementation of PyHySCO and the workflow of using the software tool. In Section 3.4, we present extensive numerical results on real and simulated data of varying field strengths. In Section 3.5, we provide a summary of the 3D correction offered by PyHySCO.

3.1 Parallelized One-Dimensional Initialization

Due to the non-convexity of the optimization problem (2.5), an effective initialization strategy for the field map is critical. To this end, PyHySCO initializes the correction with the result of the one-dimensional correction of [15], which can be derived from optimal transport (OT) theory [49]. The key idea is to compute the 'halfway' point of the oppositely distorted images in the Wasserstein space (as opposed to Euclidean space, which would simply average the images). To render this problem feasible, we treat each image column separately, use the closed-form solutions of 1D OT problems, and then apply a smoothing filter. Implementing the [15] correction using optimal transport provides a mathematical understanding of their algorithm and a highly accurate and parallelizable initialization.

We calculate the initial transformations as optimal transport maps [49]. More specifically, because the distortions only occur in the phase encoding direction, these transformations are a set of one-dimensional maps calculated in parallel across the distortion dimension. One-dimensional optimal transport has a closed-form solution arising from considering the one-dimensional signal as a positive measure and constructing a cumulative distribution function [49].

We describe the computation of the one-dimensional optimal transport maps in the distortion correction setting. In practice, the computation is parallelized in the distortion dimension to compute the entire initial field map simultaneously.

Let $i_{+v} \in \mathbb{R}^m$ be the image data from an entry in the phase encoding dimension of $\mathcal{I}_{+v}$, and let $i_{-v} \in \mathbb{R}^m$ be the image data from the corresponding entry in the phase encoding dimension of $\mathcal{I}_{-v}$. Consider i_{half} the sequence of image intensity values from the corresponding entry of the undistorted image $\mathcal{I}$. We numerically ensure i_{+v} and i_{-v} can be considered positive measures by applying a small shift to the image values, which does not change the relative distance between elements.

We initialize the field map using the optimal transport maps T_+ from i_{+v} to i_{half}

and T_- from i_{-v} to i_{half}. These maps can be directly computed using the closed-form one-dimensional optimal transport formula, which depends on a cumulative distribution function and its pseudoinverse [49].

Define the discretized cumulative distribution function $C_i : \{0, \ldots, m\} \to [0, 1]$ of a measure i as the cumulative sum

$$\forall x \in \{0, \ldots, m\} \;\; C_i(x) = \sum_{j=0}^{x} i(j),$$

where $i(j)$ returns the pixel intensity value at index j of i. The pseudoinverse $C_i^{-1} : [0, 1] \to \{0, \ldots, m\}$ is defined as

$$\forall r \in [0, 1] \;\; C_i^{-1}(r) = \min_x \{x \in \{0, \ldots, m\} \mid C_i(x) \geq r\}.$$

In practice, C_i^{-1} is computed using a linear spline interpolation.

Returning to the measures arising from the input images, the closed-form solution for one-dimensional optimal transport gives the optimal transport map from i_{+v} to i_{half} as

$$T_+ = C_{i_{\text{half}}}^{-1} \circ C_{i_{+v}},$$

and the optimal transport map from i_{-v} to i_{half} as

$$T_- = C_{i_{\text{half}}}^{-1} \circ C_{i_{-v}},$$

where $C_{i_{\text{half}}}^{-1}$ is calculated as $(C_{i_{+v}}^{-1} + C_{i_{-v}}^{-1})/2$. Figure 3.1 visualizes the computation of the one-dimensional transport maps, and the parallelized computation and resulting field maps are visualized in Figure 3.2. We thus compute the initial guess for the field map as the average of the maps T_+ and $-T_-$, computed in parallel. To introduce smoothness in the field map in the frequency encoding and slice selection dimensions, we apply a smoothing filter to the initial field map before optimization.

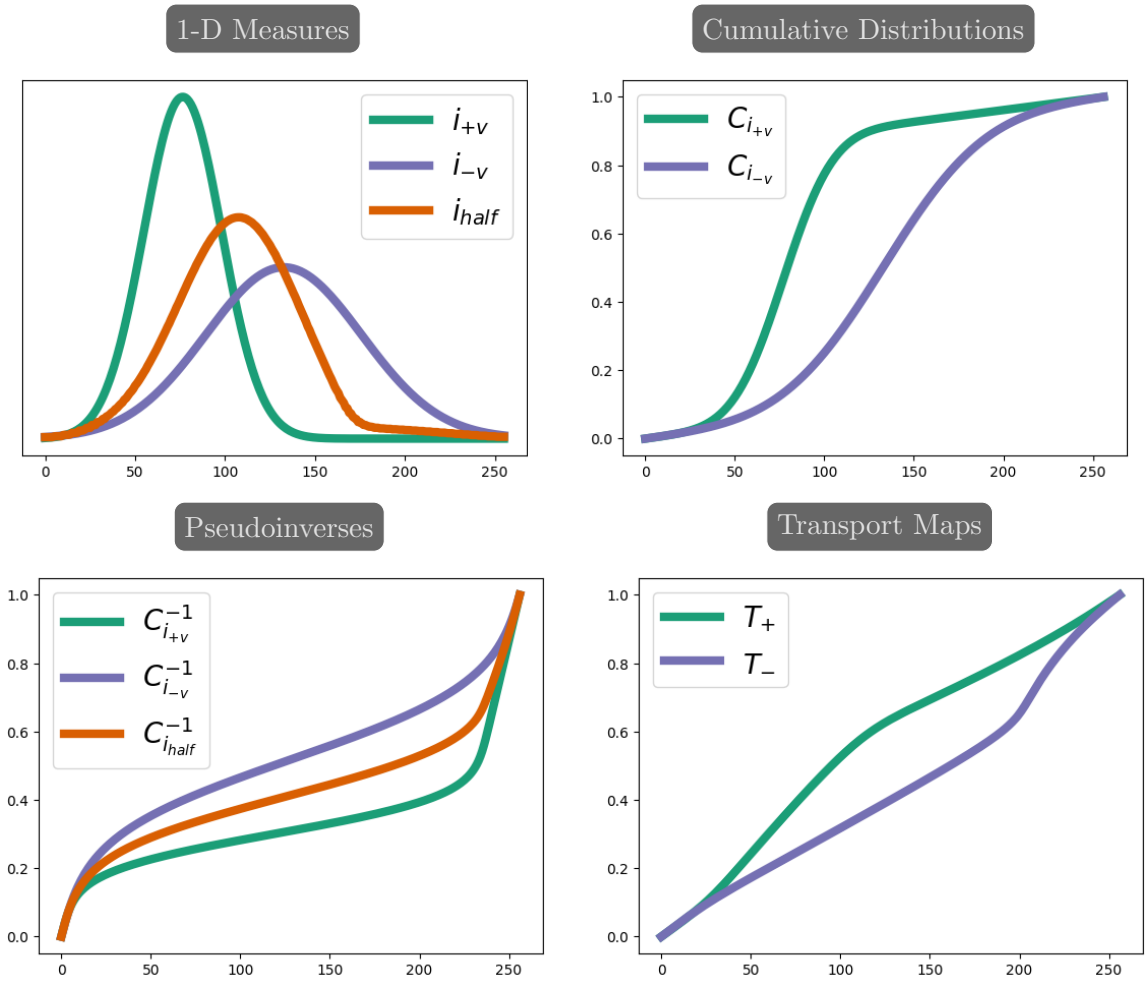
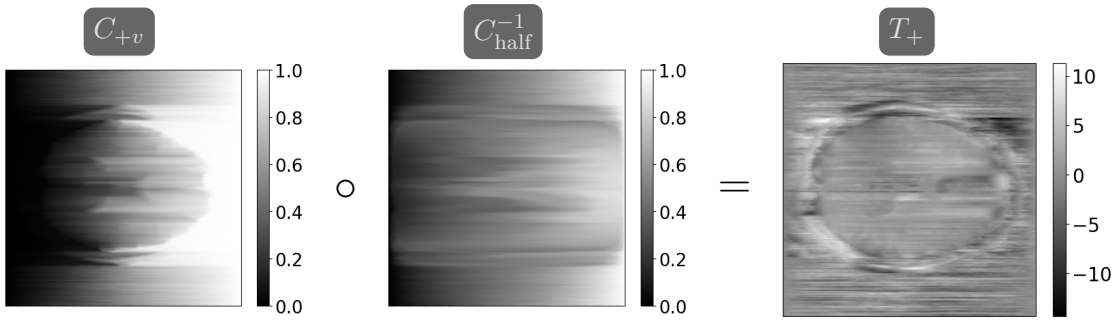
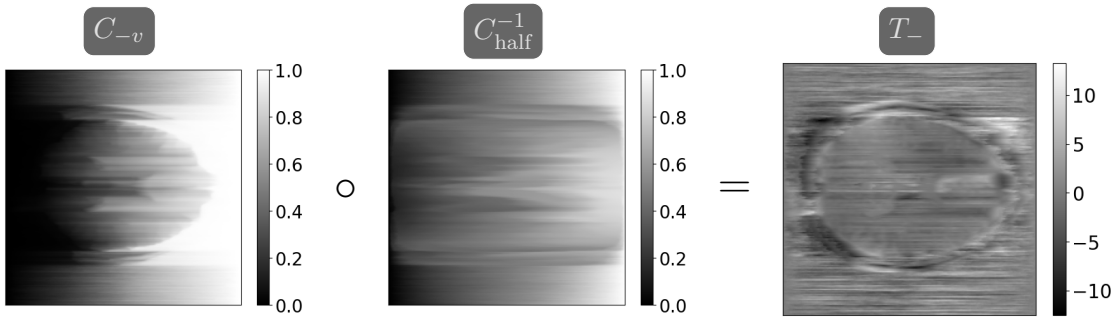


Figure 3.1: Example of one-dimensional optimal transport maps. Top left shows example one-dimensional measures. The green signal, i_{+v}, corresponds to an intensity pileup in $\mathcal{I}_{+v}$, while the purple signal i_{-v} corresponds to an intensity dispersion in $\mathcal{I}_{-v}$. The red signal corresponds to the intensity of the true image. Top right shows the cumulative distributions for the measures i_{+v} and i_{-v}. Bottom left shows the pseudoinverses for i_{+v} and i_{-v} along with the pseudoinverse $C^{-1}_{i_{\text{half}}}$ used in calculating the transport maps $T_+ = C^{-1}_{i_{\text{half}}} \circ C_{i_{+v}}$ and $T_- = C^{-1}_{i_{\text{half}}} \circ C_{i_{-v}}$, shown bottom right.



(a) The map T_+ mapping from $\mathcal{I}_{+v}$ halfway to $\mathcal{I}_{-v}$ is calculated as the composition of the cumulative distribution function C_{+v} from $\mathcal{I}_{+v}$ and the interpolated pseudoinverse C_{half}^{-1}.



(b) The map T_- mapping from $\mathcal{I}_{-v}$ halfway to $\mathcal{I}_{+v}$ is calculated as the composition of the cumulative distribution function C_{-v} from $\mathcal{I}_{-v}$ and the interpolated pseudoinverse C_{half}^{-1}.

Figure 3.2: The maps T_+ and T_- are calculated using the closed-form one-dimensional optimal transport solution, parallelized in the distortion dimension [49]. Note the inverted coloring between T_+ and T_- as the map T_- corrects a distortion in the opposite direction as T_+.

3.2 Parallelized Optimization

We implement in PyTorch three optimization schemes to solve (2.5): Gauss Newton, ADMM, and LBFGS. We implement in PyTorch Gauss Newton PCG as described in Section 2.4. In our implementation, to solve the Newton system in (2.6) we apply up to 10 iterations of the preconditioned conjugate gradient (PCG) method and stop early if the relative residual is less than 0.1. As a computationally inexpensive preconditioner, we implement a Jacobi preconditioner, which approximates the inverse of H_J by the inverse of its diagonal entries. Instead of constructing the matrix H_J, which is

computationally expensive, we provide efficient algorithms to compute matrix-vector products and extract its diagonal.

In our LBFGS implementation, we provide an explicitly calculated derivative to an LBFGS solver[1]. In computing the objective function, we precompute parts of the derivative which allows for faster optimization than relying on automatic differentiation.

We implement an unconstrained PyTorch version of the ADMM setup in [42] described in Section 2.4. In contrast to [42], which uses a hard constraint to ensure positivity of the intensity modulation and employs Sequential Quadratic Programming, we implement this as a soft constraint with the barrier term (2.2). In our case, we split the objective in (2.5) into

$$F(\mathbf{b}) = D(\mathbf{b}) + \alpha S_3(\mathbf{b}) + \beta P(\mathbf{b}), \quad \text{and} \quad G(\mathbf{z}) = \alpha S_1(\mathbf{z}) + \alpha S_2(\mathbf{z}), \tag{3.1}$$

where S_3 is the part of the smoothness regularization term S corresponding to the phase encoding direction, and S_1 and S_2 are the remaining terms corresponding to the other directions. This gives rise to the following optimization problem, equivalent to (2.5):

$$\min_{\mathbf{b},\mathbf{z}} F(\mathbf{b}) + G(\mathbf{z}) \;\; \text{s.t.} \;\; \mathbf{b} = \mathbf{z}.$$

With the corresponding augmented Lagrangian

$$L(\mathbf{b}, \mathbf{z}, \mathbf{y}) = F(\mathbf{b}) + G(\mathbf{z}) + \mathbf{y}^T(\mathbf{b} - \mathbf{z}) + \frac{\rho h^3}{2}||\mathbf{b} - \mathbf{z}||^2,$$

where $\mathbf{y}$ is the Langrange multiplier for the equality constraint $\mathbf{b} = \mathbf{z}$ and ρ is a scalar augmentation parameter, and using scaled dual variable $\mathbf{u} = \frac{\mathbf{y}}{\rho h^3}$, each iteration has

[1] https://github.com/hjmshi/PyTorch-LBFGS

the updates

$$\mathbf{b}_{k+1} = \arg\min_{\mathbf{b}} F(\mathbf{b}) + \frac{\rho h^3}{2}||\mathbf{b} - \mathbf{z}_k + \mathbf{u}_k||^2 \tag{3.2}$$

$$\mathbf{z}_{k+1} = \arg\min_{\mathbf{z}} G(\mathbf{z}) + \frac{\rho h^3}{2}||\mathbf{b}_{k+1} - \mathbf{z} + \mathbf{u}_k||^2 \tag{3.3}$$

$$\mathbf{u}_{k+1} = \mathbf{u}_k + \mathbf{b}_{k+1} - \mathbf{z}_{k+1}. \tag{3.4}$$

The $\mathbf{b}$ update computed in (3.2) involves a separable optimization problem that can be solved independently for each image column along the phase-encoding direction. In PyHySCO we use a modified version of the GN-PCG scheme described in 2.4. The only change is the computation of the search direction, which can now be parallelized across the different image columns. To exploit this structure, we implement a PCG method that solves the system for each image column in parallel. In addition to more parallelism, we observe an increase in efficiency since the scheme uses different step sizes and stopping criteria for each image column.

The $\mathbf{z}$ update in (3.3) is computed directly as described in Section 2.4.

3.3 PyHySCO: A GPU-Enabled, Command Line Compatible, PyTorch Correction Tool

3.3.1 PyHySCO Implementation Details

PyHySCO is a GPU-friendly PyTorch [47] implementation of the distortion correction approach described above. The overall code structure is visualized in the diagrams in Figures 3.3a and 3.3b for the objective function and optimization, respectively. The main classes of PyHySCO are the loss function, implemented in `EPIMRIDistortionCorrection`, and the optimization, defined in `EPIOptimize`. The other classes and methods, described in detail below, implement the components of

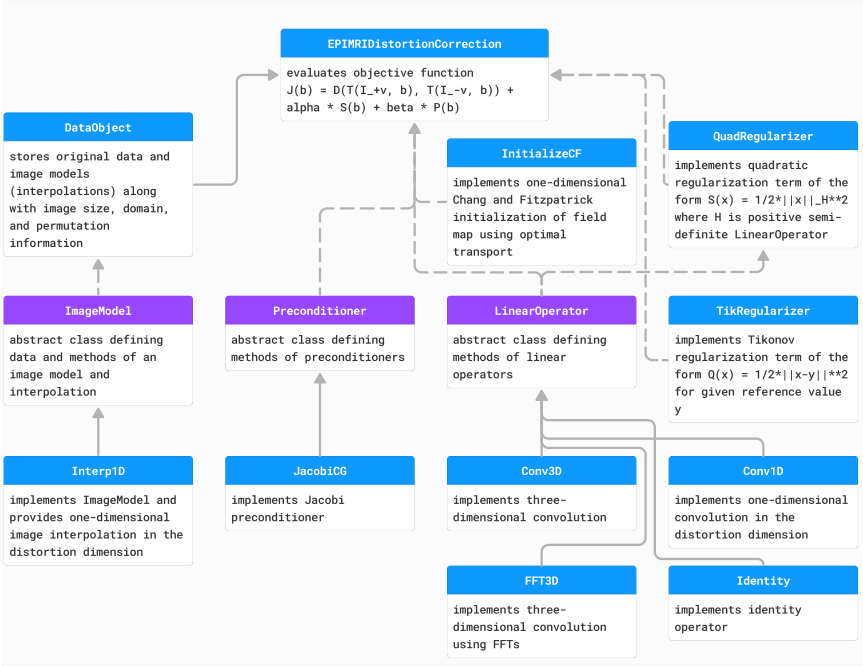
the loss function evaluation and optimization schemes.

Data Storage and Image Model

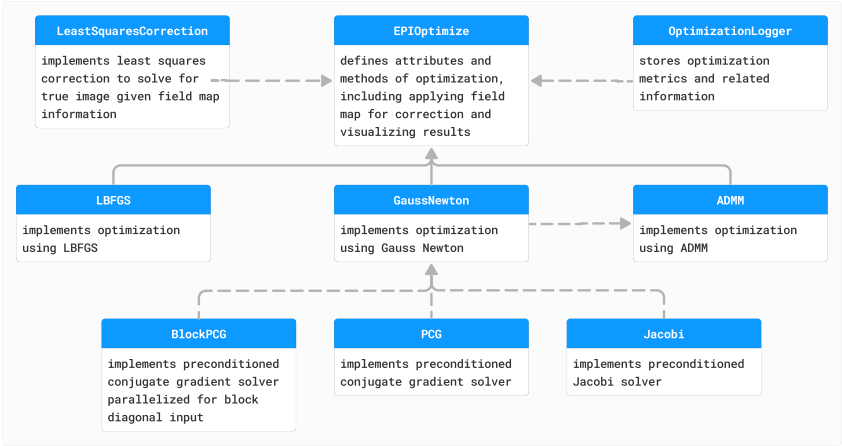
The input pair of images with opposite phase encoding directions are loaded and permuted such that the distortion dimension is the last dimension, as this is where PyTorch expects the batch dimension for parallelizing operations. Information on the input images is stored in an object of type `DataObject`. This class stores information on the image size, image domain, cell size, how to permute the data back to the input order, and the `ImageModel` for each input image. The `ImageModel` abstract class defines the structure and required methods for an image model, including storing the original data and providing a method `eval` that returns the data interpolated on the given points. We provide the default implementation `Interp1D` which is a linear one-dimensional interpolation, parallelized in the last dimension. The `DataObject` for a given input pair is then stored in the `EPIMRIDistortionCorrection` object.

Correction Model

The mass-preserving correction model (2.1) is implemented in the method `mp_transform`, a class method of `EPIMRIDistortionCorrection`. The method takes as input an `ImageModel` and a field map. The geometric deformation is computed by using an averaging `LinearOperator` to compute the field map values in the cell-centers and adding this to a cell-centered grid to obtain the deformed grid defined by this field map. Using the `ImageModel`, the image is interpolated on this deformed grid. The intensity modulation term is computed by applying a finite difference `LinearOperator` to the field map. The two terms are multiplied together element-wise before returning the corrected image. Default implementation of the `LinearOperator` objects for averaging and finite difference are given as one-dimensional convolutions, parallelized in the last dimension.



(a) Class structure of PyHySCO loss function. The main class representing the loss function is `EPIMRIDistortionCorrection`. Purple classes are abstract, and blue classes are concrete. Solid arrows indicate inheritance. Dashed arrows indicate dependencies and class objects that are attributes.



(b) Class structure of PyHySCO optimization. The main class defining optimization is `EPIOptimize`. Solid arrows indicate inheritance. Dashed arrows indicate dependencies and class objects that are attributes.

Figure 3.3: UML diagram of PyHySCO showing the classes and relationships for the (3.3a) loss function and (3.3b) optimization. A `EPIMRIDistortionCorrection` object defining the loss function is an attribute of every `EPIOptimize` object defining the optimization scheme.

Regularization Terms

The intensity regularization term is computed within the `EPIMRIDistortionCorrection` class in the method `phi_EPI` which computes the result of applying ϕ as defined in (2.2) element-wise to the result of applying the finite difference operator to the field map, as computed in the correction model. This function acts as a barrier term penalizing values of the derivative of the field map in the distortion dimension that violate the intensity constraint, that is values outside of the range (-1, 1).

The smoothness regularization term is implemented in a `QuadRegularizer` object, which defines the evaluation of a quadratic regularization term of the form of (2.4) using a positive semi-definite `LinearOperator` as H. By default, H is a discretized negative Laplacian applied via a three-dimensional convolution.

In the ADMM optimizer, the regularizer structure is different to account for the splitting in (3.1). The objective function for the $\mathbf{b}$ update in (3.2) is computed in `EPIMRIDistortionCorrection` where the computation of S_3 is a one-dimensional Laplacian in the distortion dimension applied via a one-dimensional convolution. The proximal term is computed through a `TikRegularizer` object, a Tikhonov regularizer structure. The objective function for the $\mathbf{z}$ update in (3.3) is a `QuadRegularizer` object where the `LinearOperator` H is a two-dimensional Laplacian corresponding to S_2 and S_3. This operator is implemented in `FFT3D`, which defines an operator applying a convolution kernel in Fourier space diagonalized as in (2.12) [20]. This implementation allows for easily inverting the kernel in solving for $\mathbf{z}$.

Hessian and Preconditioning

For the Gauss Newton and ADMM optimizers, an approximate Hessian and preconditioner are additionally computed. Parts of the Hessian are computed during objective function evaluation in `EPIMRIDistortionCorrection`, and the Hessian can be applied through a matrix-vector product. Similarly, a `Preconditioner` can be

computed during objective function evaluation and is accessible through a returned function applying the preconditioner to its input. By default, we provide a Jacobi preconditioner in the class `JacobiCG`.

Initialization

The `EPIMRIDistortionCorrection` class has a method `initialize`, returning an initial guess for the field map using some `InitializationMethod`. We provide an implementation of the proposed parallelized Chang and Fitzpatrick initialization in `InitializeCF`. The implementation computes the one-dimensional transport maps in parallel using a linear spline interpolation. In practice, the parallelized initialization gives a highly non-smooth initial field map, so the method optionally applies a Gaussian blur using the fast FFT convolution operator `FFT3D` to promote a more smooth optimized field map.

Optimization

The minimization of the objective function defined in a `EPIMRIDistortionCorrection` object happens in a subclass of `EPIOptimize`, which takes the objective function object as input. During optimization, the `OptimizationLogger` class is used to track iteration history, saving it to a log file and optionally printing this information to standard output. PyHySCO includes implementations of the LBFGS, Gauss Newton, and ADMM solvers described previously. Each of the classes `LBFGS`, `GaussNewton`, and `ADMM` provide a `run_correction` method which minimizes the objective function using the indicated optimization scheme. The `LBFGS` implementation uses the explicitly computed derivative from `EPIMRIDistortionCorrection`. For LBFGS we use as stopping criteria the norm of the gradient reaching a given tolerance, or the change in loss function or field map between iterations falling below a given tolerance. The `GaussNewton` implementation uses a conjugate gradient solver implemented in

the class `PCG`. Our Gauss Newton implementation uses the same stopping criteria as LBFGS. The `ADMM` implementation solves the **b** update in (3.2) using `GaussNewton` with a parallelized conjugate gradient solver in `BlockPCG`. The **z** update in (3.3) is solved directly through the inverse method `inv` of the operator used to define the `QuadRegularizer` for this term, efficiently implemented using FFTs in `FFT3D`. As stopping criteria, the ADMM iterations will terminate if the change in all of **b**, **z**, and **u** from the previous iteration falls below a given tolerance.

Image Correction

The optimal field map, stored as `Bc` in the `EPIOptimize` object after `run_correction` is completed, can be used to produce a corrected image or pair of images. The `apply_correction` method of `EPIOptimize` implements both a Jacobian modulation correction and a least squares correction. The Jacobian modulation correction is based on the model of [15] as implemented in the `mp_transform` method of `EPIMRIDistortionCorrection`. This correction method computes and saves two corrected images, one for each input image.

The field map can also be used in a least squares correction similar to the correction in [4], implemented in `LeastSquaresCorrection`. In this correction, the estimated field map is used to determine a push forward matrix that transforms the true image to the distorted image given as input. This gives rise to a least squares problem to solve for the true image given the input images and push forward matrix.

3.3.2 PyHySCO Usage and Workflow

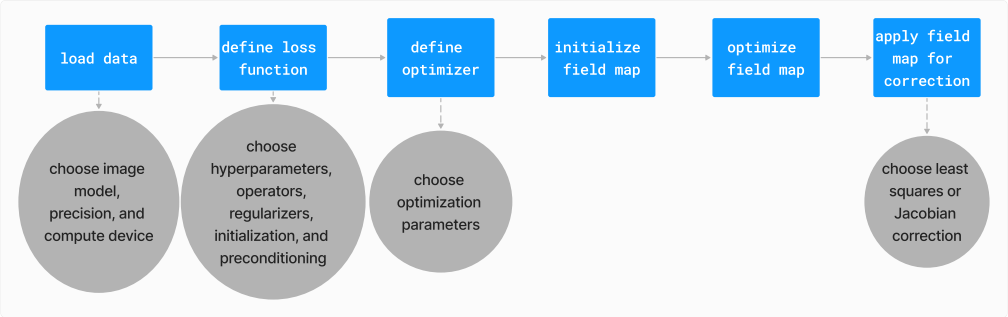
The workflow of PyHySCO is illustrated in Figure 3.4a alongside examples of using PyHySCO in a Python script (Figure 3.4b) and through the command line (Figure 3.4c). Running PyHySCO from a user-defined Python script allows more specific control of the inputs and outputs from PyHySCO methods. The command line in-

terface allows the user to pass configuration options directly from the command-line, which enables our EPI distortion correction tool to be easily used as a part of existing command-line based MRI post-processing pipelines such as the FSL toolbox [56]. Executing PyHySCO requires the user to provide at a minimum the file paths for the input pair of images with opposite phase encoding directions and which dimension (1, 2, or 3) is aligned with the phase encoding direction. The modularity of PyHySCO additionally allows for configuring options such as the scalar hyperparameters in (2.5), implementation of operators, regularizers, and interpolation, optimizer and associated optimization parameters, and image correction method.

Regardless of execution through a script or the command line, PyHySCO stores the input images in a `DataObject` object, the loss function and how to evaluate it in a `EPIMRIDistortionCorrection` object, and the optimizer in an object of a subclass of `EPIOptimize`. The field map is initialized from the method `initialize` in `EPIMRIDistortionCorrection`, and the field map is optimized by calling the method `run_correction` in the optimizer object. Finally, the method `apply_correction` in `EPIOptimize` applies the field map to correct the input images and saves the result to one or more NIFTI file(s).

3.4 Results

We demonstrate PyHySCO's effectiveness through extensive experiments using real and simulated data from the Human Connectome Project [62] and validate the initialization scheme and implementation of optimization algorithms. Section 3.4.1 describes the datasets and Section 3.4.2 introduces our evaluation metrics. In Section 3.4.3, we demonstrate the Chang and Fitzpatrick initialization scheme. The experiments in 3.4.4 compare the performance of the three optimization algorithms implemented in PyHySCO on CPU and GPU hardware. In Section 3.4.5, we compare



(a) The workflow of the PyHySCO toolbox from setup through optimization and distortion correction.

```python
# load images with phase encoding direction in first dimension
data = DataObject('im1.nii.gz', 'im2.nii.gz', 1, device='cuda:0', dtype=torch.float32)
# define loss function
loss_func = EPIMRIDistortionCorrection(data, 300, 1e-4, averaging_operator=myAvg1D,
                                       derivative_operator=myDiff1D, regularizer=myLaplacian3D, PC=JacobiCG)
# define optimizer using Gauss Newton
opt = GaussNewton(loss_func, max_iter=500, verbose=True, path='results/')
# initialize field map
B0 = loss_func.initialize()
# optimize field map
opt.run_correction(B0)
# apply correction
opt.apply_correction(method='lstsq')
```

(b) An example using the PyHySCO toolbox from a Python script.

```
root $ pyhysco --help
usage: pyhysco   [-h] [--output_dir OUTPUT_DIR] [--alpha ALPHA] [--beta BETA] [--rho RHO]
                 [--optimizer OPTIMIZER] [--max_iter MAX_ITER] [--verbose] [--precision {single,double}]
                 [--correction {jac,lstsq}] [--averaging AVERAGING] [--derivative DERIVATIVE]
                 [--initialization INITIALIZATION] [--regularizer REGULARIZER] [--PC PC]
                 file_1 file_2 {1,2,3}

PyHySCO: EPI-MRI Distortion Correction.

positional arguments:
  file_1                Path to the input 1 data file (NIfTI format .nii.gz)
  file_2                Path to the input 2 data file (NIfTI format .nii.gz)
  {1,2,3}               Dimension of phase encoding direction

optional arguments:
  -h, --help                      show this help message and exit
  --output_dir OUTPUT_DIR         Directory to save the corrected images and reports (default=cwd)
  --alpha ALPHA                   Smoothness regularization parameter (default=300)
  --beta BETA                     Intensity modulation constraint parameter (default=1e-4)
  --rho RHO                       Initial Lagrangian parameter (ADMM only) (default=1e3)
  --optimizer OPTIMIZER           Optimizer to use (default=GaussNewton)
  --max_iter MAX_ITER             Maximum number of iterations (default=50)
  --verbose                       Print details of optimization (default=True)
  --precision {single,double}     Use (single/double) precision (default=single)
  --correction {jac,lstsq}        Use (Jacobian ['jac']/ Least Squares ['lstsq']) correction (default=lstsq)
  --averaging AVERAGING           LinearOperator to use as averaging operator (default=myAvg1D)
  --derivative DERIVATIVE         LinearOperator to use as derivative operator (default=myDiff1D)
  --initialization INITIALIZATION Initialization method to use (default=InitializeCF)
  --regularizer REGULARIZER       LinearOperator to use for smoothness regularization term (default=myLaplacian3D)
  --PC PC                         Preconditioner to use (default=JacobiCG)
```

(c) The help message for the PyHySCO command line interface. This interface allows for easily using PyHySCO as part of existing MRI post-processing pipelines.

Figure 3.4: The usage and workflow of PyHySCO.

Dataset	No. of Subjects	Image Size	Resolution	PE directions
3T	20	$168 \times 144 \times 111$	$1.25 \times 1.25 \times 1.25\ mm^3$	LR/ RL
7T	20	$200 \times 200 \times 132$	$1.05 \times 1.05 \times 1.05\ mm^3$	AP/ PA
Simulated	20	$320 \times 320 \times 256$	$0.7 \times 0.7 \times 0.7\ mm^3$	AP/ PA

Table 3.1: Details of data used in validation. LR/RL is left-to-right and right-to-left phase encoding, and AP/PA is anterior-to-posterior and posterior-to-anterior phase encoding. Further details of acquisition parameters are in [62].

the performance of PyHySCO in single and double precision arithmetic on CPU and GPU hardware. Section 3.4.6 compares PyHySCO with existing tools HySCO and TOPUP [52, 4].

3.4.1 Validation Datasets

The data used in the following experiments is from the Human Connectome Project [62]. We validate our methods and tool on 3T and 7T diffusion-weighted imaging data from the HCP 1200 Subjects Release, with 20 subjects randomly chosen for each field strength. Table 3.1 provides details of the datasets.

We also evaluate our methods on simulated data. This data only contains susceptibility artifact distortions, so it shows how our tool performs without the influence of other factors, e.g., patient movement between scans. To simulate the distortions, we use a pair of magnitude and phase images for a subject in HCP and generate the field map using FSL's FLIRT and PRELUDE tools [56]. Considering the physical model of [15], the field map b can be used to define the push-forward matrices that give how the intensity value at x is pushed forward to $x + b(x)$ in the distortion direction $+v$ as well as the opposite direction $-v$. Applying the push-forward matrices to a T2-weighted image for the subject, we generate a pair of distorted images. For the simulated data, we then have a reference value for the field map and an undistorted, true image.

3.4.2 Metrics

The quality of correction results is measured using the relative improvement of the distance between a pair of corrected images. Particularly, we calculate the sum-of-squares distance (SSD) of the corrected image pair relative to the SSD of the input pair. This metric is a useful surrogate for the correctness of the field map in the absence of a ground truth [28]. Additionally, we take the value of the smoothness regularization term $S(\mathbf{b})$ as a measure of how smooth the resulting field map is, with lower values being better.

We report the runtime in seconds of PyHySCO. The runtime is measured as the wall clock time using the Linux `time` command when calling the correction method from the command line. This time, therefore, includes the time taken to load and save the image data. In some cases, we also report the optimization time only, without loading and saving data, as measured by Python's `time` module.

3.4.3 Validity of Chang and Fitzpatrick Initialization

We compare the results of PyHySCO using the one-dimensional parallelized Chang and Fitzpatrick initialization to those of the multi-level initialization used in HySCO [52] both at initialization and after optimization with Gauss-Newton. The multi-level optimization of HySCO solves the optimization problem on a coarse grid and uses the result as the initialization of optimization on a finer grid, continuing until reaching the original image resolution; this follows the guidelines of [45, Chapter 9.4]. In our experiments, we use five levels in the initialization. The multi-level initialization gives a field map that is smooth by construction and improves the distance reduction as the grid becomes more fine. The field map from the PyHySCO Chang and Fitzpatrick initialization drastically lowers the relative error between the input images, a relative improvement of over 96% on real data and 94% on simulated data. However, the parallelized one-dimensional computations lead to a lack of smoothness in the resulting

field map. The smoothness can be improved by applying a Gaussian blur to the field map from the Chang and Fitzpatrick initialization. This field map is smoother after initialization and gives a smoother field map after optimization. These results are comparable in relative error and smoothness to the field map optimized from the multilevel initialization of HySCO. Our one-dimensional parallelized initialization, even with the additional Gaussian blur, is much faster to compute than the multilevel initial field map given the ability to parallelize computations. PyHySCO initialization on a GPU with the additional blur takes less than 1 second on real data and about 3 seconds on simulated data. In comparison, the multi-level initialization on a CPU takes 30 to 40 seconds on real data and over 2 minutes on simulated data. The mean and standard deviation relative improvement, smoothness value, loss function value, and runtime are reported in Table 3.2 across all datasets. Examples of these field maps before and after optimization are shown in Figure 3.5.

3.4.4 Comparison of PyHySCO Optimizers on GPU and CPU

We compare the results of PyHySCO using GN-PCG, ADMM, and LBFGS on both GPU and CPU architectures. Table 3.3 shows the runtimes and correction quality of each optimizer on CPU and GPU. All optimizers achieve a similar correction quality with respect to relative improvement of image distance, loss value, and smoothness regularizer value, but GN-PCG has faster runtime on both CPU and GPU. On real data, GN-PCG took 10-13 seconds on average on GPU and 27-31 seconds on average on CPU, while ADMM took 11-15 seconds on GPU and 98-158 seconds on CPU, and LBFGS took 23-36 seconds on GPU and 104-141 seconds on CPU. Table 3.4 shows optimization metrics, including the number of iterations, stopping criteria, number of function evaluations, number of Hessian evaluations, and number of inner iterations if applicable. Consistent with its faster runtime, optimization with GN-PCG achieves a similar loss value with less computation as measured by function

		Chang & Fitzpatrick		Chang & Fitzpatrick (blur)		Multilevel	
		initial	after opt	initial	after opt	initial	after opt
3T	Runtime (s)	5.78 ±1.26	11.43 ±1.46	6.31 ±0.60	15.36 ±3.90	41.69 ±1.71	55.34 ±2.84
	Opt. Time (s)	0.27 ±0.01	4.34 ±0.67	0.28 ±0.02	6.78 ±0.67	42.43 ±4.04	48.65 ±3.95
	Relative Improvement	96.44 ±1.13	83.90 ±3.43	79.71 ±3.43	82.75 ±3.49	67.04 ±5.15	81.96 ±3.51
	Loss Value	$1.05e09$ $\pm 2.66e08$	$2.84e07$ $\pm 7.49e06$	$1.76e08$ $\pm 5.20e07$	$2.56e07$ $\pm 7.66e06$	$4.82e07$ $\pm 1.70e07$	$2.51e07$ $\pm 7.54e06$
	Smoothness Reg. Value	$3.50e06$ $\pm 8.81e05$	$5.08e04$ $\pm 1.23e04$	$5.28e05$ $\pm 1.54e05$	$3.85e04$ $\pm 1.21e04$	$6.89e04$ $\pm 2.98e04$	$3.47e04$ $\pm 1.08e04$
7T	Runtime (s)	7.61 ±1.99	13.55 ±2.04	8.32 ±2.79	19.72 ±2.91	58.73 ±6.24	77.79 ±5.72
	Opt. Time (s)	0.61 ±0.02	5.09 ±1.23	0.63 ±0.02	10.16 ±0.85	30.38 ±2.63	40.50 ±3.87
	Relative Improvement	96.53 ±1.47	86.01 ±5.15	75.09 ±3.97	85.76 ±5.10	69.12 ±8.28	85.42 ±5.08
	Loss Value	$3.48e09$ $\pm 1.15e09$	$5.28e07$ $\pm 2.01e07$	$4.50e08$ $\pm 2.47e08$	$4.14e07$ $\pm 1.95e07$	$7.77e07$ $\pm 3.01e07$	$4.02e07$ $\pm 1.82e07$
	Smoothness Reg. Value	$1.16e07$ $\pm 3.83e06$	$9.52e04$ $\pm 2.64e04$	$1.36e06$ $\pm 7.74e05$	$5.63e04$ $\pm 1.91e04$	$8.21e04$ $\pm 4.07e04$	$5.03e04$ $\pm 1.48e04$
Simulated	Runtime (s)	10.62 ±0.57	80.29 ±9.96	16.59 ±0.64	106.47 ±11.21	173.20 ±27.06	47.98 ±8.38
	Opt. Time (s)	3.51 ±0.03	64.45 ±10.02	3.61 ±0.15	89.17 ±11.48	125.35 ±24.88	157.95 ±28.79
	Relative Improvement	94.64 ±1.26	76.82 ±5.09	75.34 ±3.44	76.27 ±5.18	55.01 ±5.66	73.63 ±5.39
	Loss Value	$5.10e08$ $\pm 9.51e07$	$6.31e07$ $\pm 1.46e07$	$2.11e08$ $\pm 4.30e07$	$6.07e07$ $\pm 1.39e07$	$8.17e07$ $\pm 2.08e07$	$5.83e07$ $\pm 1.33e07$
	Smoothness Reg. Value	$1.67e06$ $\pm 3.14e05$	$1.06e05$ $\pm 2.94e04$	$5.84e05$ $\pm 1.24e05$	$9.53e04$ $\pm 2.71e04$	$6.18e04$ $\pm 1.70e04$	$7.50e04$ $\pm 2.20e04$

Table 3.2: Validation of the parallelized Chang & Fitzpatrick initialization. We compare the runtime, relative improvement, smoothness value, and loss function value at initialization and after optimization with Gauss Newton for the proposed parallelized initialization, the proposed initialization with an additional Gaussian blur, and the multilevel initialization used in HySCO [52]. For each metric we report the mean and standard deviation in the 3T, 7T, and simulated datasets. The multilevel initialization is timed on CPU in Matlab, and the PyHySCO initializations and all optimizations are timed on GPU in Python. The Chang & Fitzpatrick based initializations provide a comparable quality while decreasing runtime compared to the multilevel initialization, and the initialization with Gaussian blur promotes a more smooth field map.

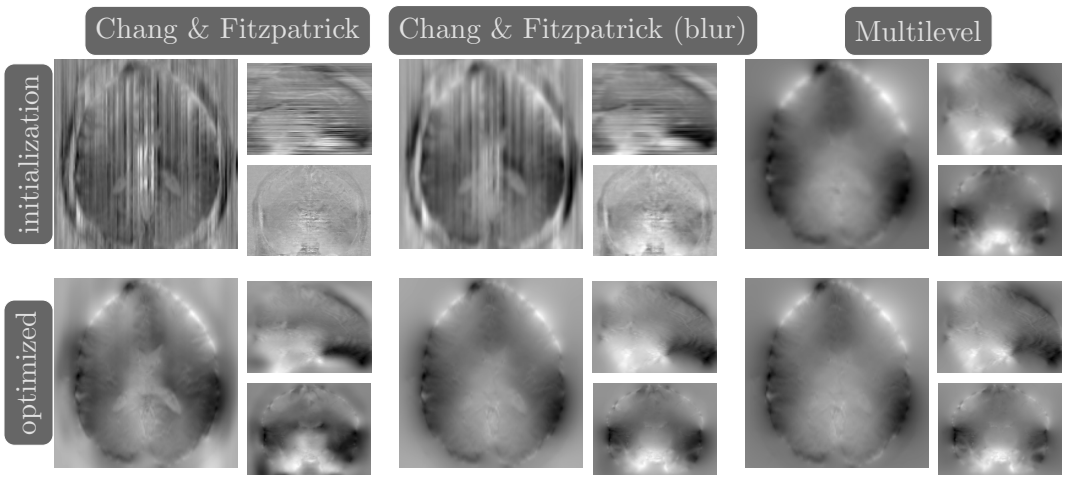


Figure 3.5: Example field maps (Subject ID 826353) at initialization (top row) and after optimization with Gauss-Newton (bottom row). The first column uses the proposed initialization scheme. The middle column uses the same scheme with an additional Gaussian blur to promote smoothness. The right column uses the coarse-to-fine multilevel initialization scheme from HySCO with five levels, and the final field map is optimized at the original image resolution. The multilevel initialized field map is smooth by construction and further optimized to improve the relative image distance at the full resolution. The PyHySCO initialization accurately corrects the distortions but is not smooth in the non-distortion dimensions unless blurred with a Gaussian. After the fine-level optimization all field maps are visually similar.

and Hessian evaluations. Figures 3.6, 3.7, and 3.8 show the field map and corrected images for each optimizer for one example subject from each dataset. The field maps and corrected images are visually similar across optimizers.

3.4.5 Single Precision vs Double Precision on GPU and CPU

We show the validity of PyHySCO using the proposed initialization and GN-PCG in both double precision (64 bit) and single precision (32 bit) arithmetic on three different GPU architectures and a CPU architecture. Since GPU architectures are optimized for the speed of lower precision calculations, we see a significant speedup when using single precision instead of double precision. Calculations in single precision, however, have the risk of lower accuracy or propagating errors due to using fewer bits to approximate floating point values. Empirically, we see that the quality of

our results is not significantly impacted by using single-precision arithmetic. We also see consistent results across different GPU architectures: a Quadro RTX 8000, Titan RTX, and RTX A6000. Because PyHySCO is optimized to parallelize computations on GPU, the runtimes are faster on the GPUs compared to the Intel Xeon E5-4627 CPU.

3.4.6 Comparison of PyHySCO with HySCO and TOPUP

We compare the runtime, relative improvement, and resulting images after correction using PyHySCO against those given by TOPUP [4] as implemented in FSL [56] using the default configuration[2], and HySCO [52] as implemented in the ACID toolbox for SPM using the default parameters. HySCO is also based on the optimization problem (2.5), while TOPUP uses a slightly different objective function. This makes it difficult to compute smoothness and loss function values for TOPUP.

Table 3.6 reports the runtime and correction quality for PyHySCO using GN-PCG, HySCO, and TOPUP. On real 3T and 7T data, PyHySCO achieves lower loss and higher relative improvement between corrected images than HySCO, and higher relative improvement than TOPUP. The runtime on CPU for real data is 1-2 minutes for HySCO and over 1 hour for TOPUP, while PyHySCO on GPU has runtimes of 10-13 seconds. For the simulated dataset, PyHySCO requires an average of 1 minute on GPU, HySCO an average of 12.6 minutes on CPU, and TOPUP an average of 8.5 hours on CPU. Using the ground truth field maps from the simulated dataset, PyHySCO achieves the lowest average field map relative error, 14.48%, compared to 19.70% for HySCO and 16.36% for TOPUP. PyHySCO also achieves the highest structural similarity (SSIM) [63] with the ground truth field map, 91.80, compared to 86.91 for HySCO and 80.15 for TOPUP. All three methods average a structural

[2]The default TOPUP configuration performs upsampling requiring the dimensions to be a multiple of 2. The configuration for TOPUP with images of the 3T data set does not perform upsampling due to the odd number of slices in the image volumes.

similarity of over 99 with the ground truth T2-weighted image. Figures 3.6, 3.7, and 3.8 show the field map and corrected images for one example subject from each dataset. The results of the methods are similar, and the resulting field maps are comparable to those of the existing tools, HySCO and TOPUP, while PyHySCO is considerably faster.

3.5 Summary

The PyHySCO toolbox accurately and robustly corrects susceptibility artifacts in Spin-Echo EPIs acquired using the Reverse Gradient Polarity acquisition. In numerous experiments with real and simulated data, it achieves similar correction quality to the leading RGP toolboxes TOPUP and HySCO while having a time-to-solution in the order of timings reported for pre-trained deep learning approaches. Compared to the latter class of methods, it is important to highlight that PyHySCO does not require any training and is based on a physical distortion model, which helps generalize to different scanners, image acquisition parameters, and anatomies.

PyHySCO's modular design invites improvements and contributions. The toolbox is based on PyTorch, which provides hardware support and other functionality, including automatic differentiation. In our experiments, correction quality is hardware and precision-independent, but a considerable speedup is realized on GPUs with single precision (32-bit) arithmetic. The reduced computational time is mostly attributed to the effective use of multithreading and parallelism on modern hardware.

PyHySCO uses the one-dimensional correction of [15] to initialize the nonlinear optimization. In our numerical experiments, the scheme is fast and effective and we provide further insights through optimal transport theory. The initial estimate of the field map already substantially reduces the distance between the images with opposite phase encoding directions. In our experiments, the non-smoothness of the initial field

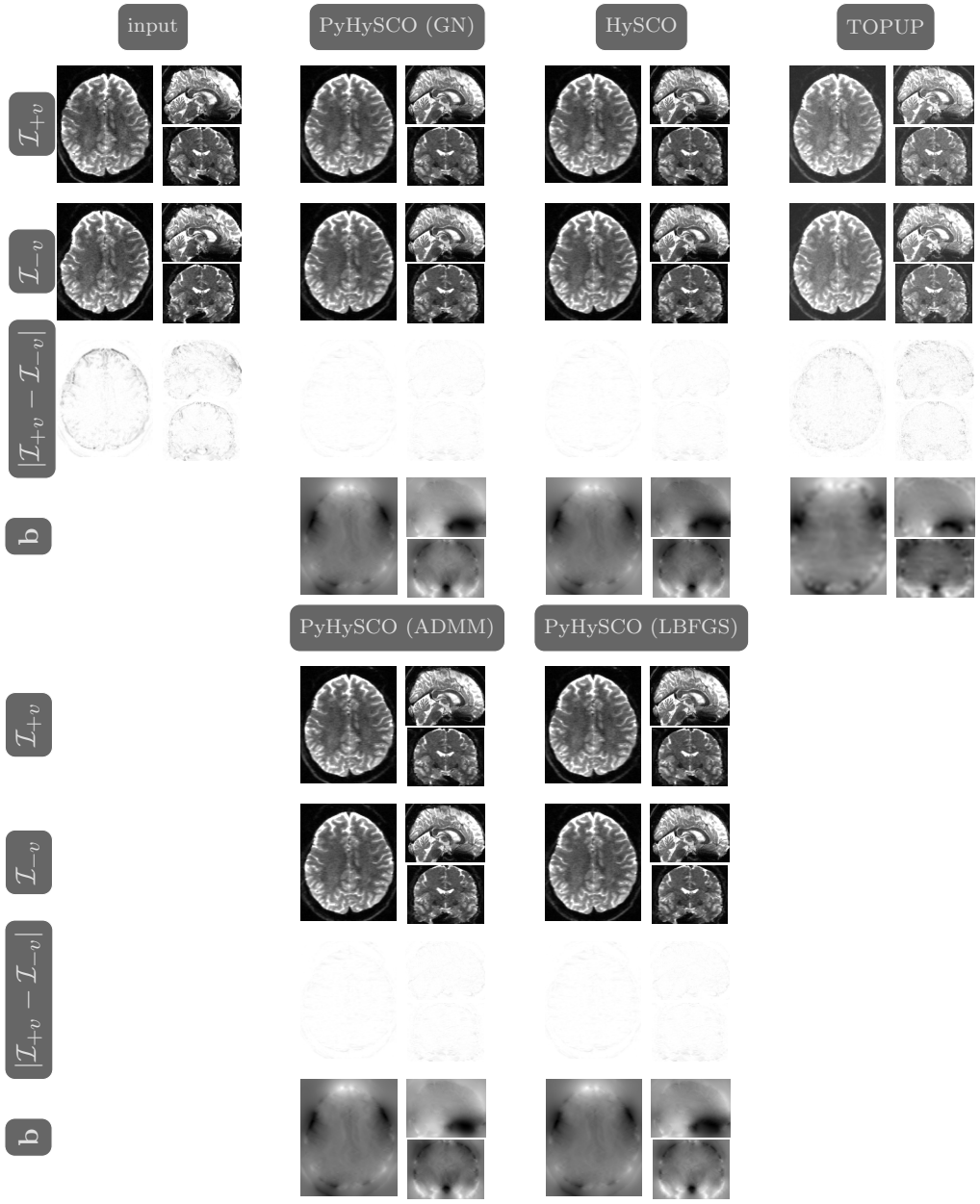


Figure 3.6: Visualization of resulting field maps and images for one subject from the 3T dataset (Subject ID 211619). The first column in the top half shows the input data, and the remaining columns show the results from PyHySCO using LBFGS, Gauss Newton, and ADMM, TOPUP, and HySCO. For each optimization, the top two rows are the pair of images with opposite phase encoding directions, and the third row shows the absolute difference (with inverted color) between the pair of images. The bottom row shows the field maps estimated for each method. PyHySCO gives a reduction in image distance and a field map comparable in smoothness to existing methods.

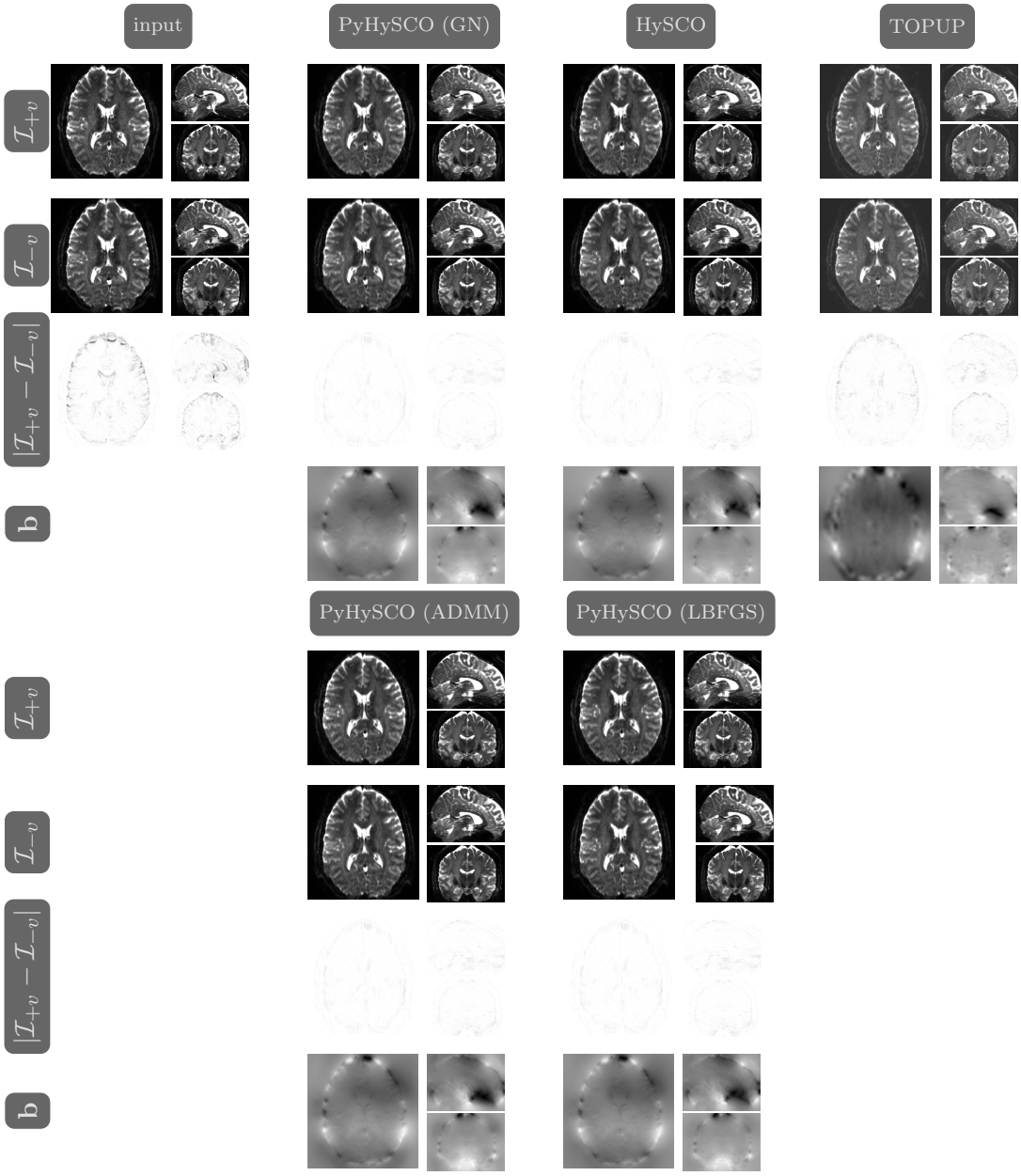


Figure 3.7: Visualization of resulting field maps and images for one subject from the 7T dataset (Subject ID 825048). The first column in the top half shows the input data, and the remaining columns show the results from PyHySCO using LBFGS, Gauss Newton, and ADMM, TOPUP, and HySCO. For each optimization, the top two rows are the pair of images with opposite phase encoding directions, and the third row shows the absolute difference (with inverted color) between the pair of images. The bottom row shows the field maps estimated for each method. PyHySCO gives a reduction in image distance and a field map comparable in smoothness to existing methods.

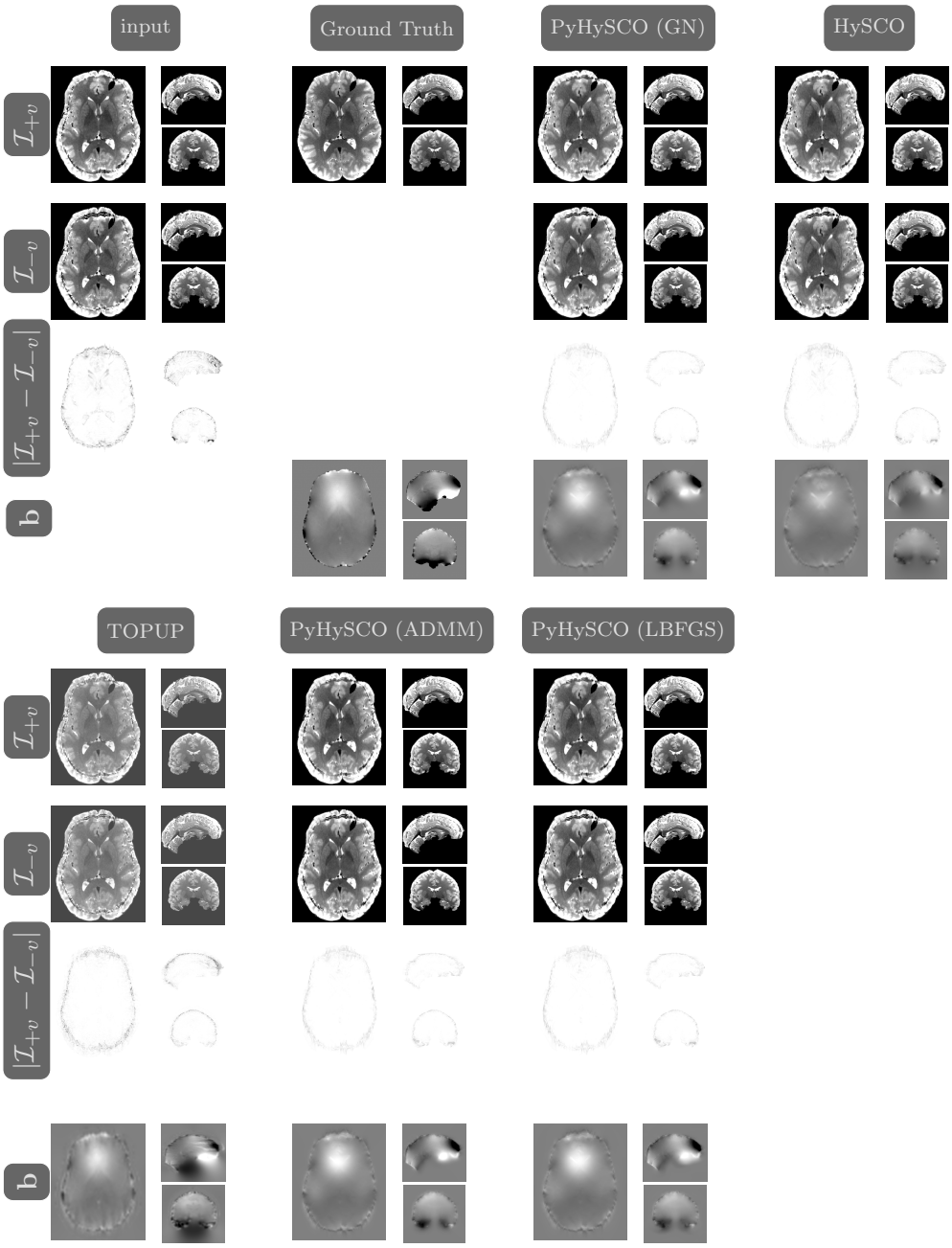


Figure 3.8: Visualization of resulting field maps and images for one subject from the simulated dataset (Subject ID 105014). The first column in the top half shows the input data, and the second column shows the ground truth T2w image and field map. The remaining columns show the results from PyHySCO using LBFGS, Gauss Newton, and ADMM, TOPUP, and HySCO. For each optimization, the top two rows are the pair of images with opposite phase encoding directions, and the third row shows the absolute difference (with inverted color) between the pair of images. The bottom row shows the field maps estimated for each method. PyHySCO gives a reduction in image distance and a field map comparable in smoothness to existing methods.

map can be corrected by applying a Gaussian blur and a few optimization steps to the full image resolution.

PyHySCO's three optimization algorithms achieve comparable correction results but have different computational costs. The ADMM algorithm takes advantage of the separable structure of the optimization problem to enhance parallelism but requires more iterations than GN-PCG. While this results in longer runtimes in our examples, the method could be more scalable for datasets of considerably higher resolution. For the relatively standard image sizes of about $200 \times 200 \times 132$, the default GN-PCG algorithm is most effective. Both customized optimization algorithms are more efficient than our comparison, LBFGS.

PyHySCO can be interfaced directly in Python or run in batch mode via the command line. The latter makes it a drop-in replacement for other RGP tools in MRI post-processing pipelines.

The speed of PyHySCO relative to existing tools makes it uniquely positioned to enable online distortion correction in applications where real-time decisions are necessary. For example, the speed of EPI acquisition along with the speed of PyHySCO distortion correction enables real-time distortion-free imaging useful for intra-operative guidance (see, e.g., [30, 50, 66]). Additionally, PyHySCO can be important for furthering emerging fields such as fetal and neonatal imaging (see, e.g., [43, 1, 17]). In this application, EPI is popular to reduce the effects of uncontrollable subject motion, and fast distortion correction using PyHySCO can enable faster intervention if necessary.

PyHySCO offers RGP-based correction with high accuracy at the cost similar to pre-trained learning-based methods. Our implementation is based on PyTorch and makes efficient use of modern hardware accelerators such as GPUs. We show the accuracy and efficiency of PyHySCO on real and simulated three-dimensional volumes of various field strengths and phase encoding axes. Our results show that PyHySCO

achieves a correction of comparable quality to leading physics-based methods in a fraction of the time.

The source code, examples, and documentation for PyHySCO are available at the following repository: https://github.com/EmoryMLIP/PyHySCO. The Python package for PyHySCO can be installed via pip and be downloaded from: https://pypi.org/project/PyHySCO/.

		LBFGS		GN-PCG		ADMM	
		CPU	GPU	CPU	GPU	CPU	GPU
3T	Runtime (s)	104.45 ±70.74	23.13 ±4.61	27.37 ±4.53	10.37 ±0.87	98.54 ±30.15	11.58 ±2.23
	Opt. Time (s)	100.28 ±70.82	16.70 ±4.49	23.13 ±4.53	4.38 ±0.68	94.53 ±30.20	5.63 ±2.15
	Relative Improvement	81.47 ±3.71	82.32 ±3.40	82.74 ±3.50	82.74 ±3.50	82.76 ±3.31	82.77 ±3.30
	Loss Value	7.90e07 ±7.99e07	2.56e07 ±7.72e06	2.56e07 ±7.69e06	2.56e07 ±7.69e06	3.09e07 ±8.51e96	3.10e07 ±8.56e06
	Smoothness Reg. Value	2.13e05 ±2.56e05	3.72e04 ±1.18e04	3.85e04 ±1.21e04	3.85e04 ±1.21e04	5.62e04 ±1.65e04	5.65e04 ±1.71e04
7T	Runtime (s)	141.44 ±117.38	36.23 ±7.76	31.71 ±3.18	13.62 ±2.38	158.64 ±46.99	15.25 ±3.15
	Opt. Time (s)	135.72 ±116.29	29.23 ±7.88	26.84 ±3.15	6.57 ±2.30	152.69 ±46.64	8.34 ±2.91
	Relative Improvement	80.75 ±6.91	85.74 ±4.99	85.76 ±5.10	85.76 ±5.10	85.87 ±4.99	85.85 ±4.99
	Loss Value	2.25e08 ±2.22e08	4.25e07 ±2.00e07	4.14e07 ±1.95e07	4.14e07 ±1.95e07	4.43e07 ±1.99e07	4.43e07 ±1.95e07
	Smoothness Reg. Value	6.38e05 ±7.01e05	6.00e04 ±2.18e04	5.63e04 ±1.91e04	5.63e04 ±1.91e04	6.68e04 ±2.18e04	6.66e04 ±3.68e04
Sim.	Runtime (s)	6344.93 ±649.21	143.77 ±6.47	1094.96 ±135.20	55.26 ±3.86	7687.28 ±4596.31	52.72 ±18.01
	Opt. Time (s)	6320.43 ±649.01	125.95 ±6.40	1070.65 ±135.69	37.60 ±4.54	7662.55 ±4596.38	35.15 ±17.92
	Relative Improvement	75.45 ±5.40	75.44 ±5.35	76.28 ±5.19	76.28 ±5.18	74.93 ±5.59	75.00 ±5.34
	Loss Value	6.03e07 ±1.44e07	6.00e07 ±1.41e07	6.08e07 ±1.40e07	6.08e07 ±1.40e07	6.08e07 ±1.40e07	6.12e07 ±1.43e07
	Smoothness Reg. Value	9.06e04 ±2.94e04	8.94e04 ±2.74e04	9.56e04 ±2.74e04	9.56e04 ±2.72e04	8.97e04 ±2.79e04	9.12e04 ±2.77e04

Table 3.3: The speed and quality of optimization in PyHySCO on GPU and CPU with LBFGS, Gauss Newton, and ADMM. We report for each dataset and optimizer the mean and standard deviation total runtime (including loading and saving data), optimization time, improvement in distance between corrected images relative to input image, loss value, and smoothness regularizer value. Gauss Newton achieves a similar correction quality in less time than LBFGS or ADMM on both CPU and GPU.

		LBFGS	GN-PCG	ADMM
3T	Iterations	455.30 ±52.80	8.400 ±0.92	36.05 ±10.37
	Stopping Criteria (grad/loss/field map/max iter)	9/3/0/8	0/20/0/0	0/0/20/0
	Func. Evals	463.30 ±54.12	9.40 ±0.92	37.05 ±10.37
	Hessian Evals	N/A	92.40 ±10.08	437.50 ±140.30
	Inner Iterations	N/A	10.0000 ±0.00	11.0269 ±1.02
	Loss Value	$2.56e07$ $\pm 7.72e06$	$2.56e07$ $\pm 7.69e06$	$3.10e07$ $\pm 8.56e06$
7T	Iterations	405.00 ±65.61	7.50 ±0.87	56.75 ±17.01
	Stopping Criteria (grad/loss/field map)	14/3/0/3	0/20/0/0	0/0/20/0
	Func. Evals	415.35 ±68.00	8.50 ±0.87	57.75 ±17.01
	Hessian Evals	N/A	82.25 ±9.15	339.05 ±101.55
	Inner Iterations	N/A	9.9722 ±0.12	4.9771 ±0.08
	Loss Value	$4.25e07$ $\pm 2.00e07$	$4.14e07$ $\pm 1.95e07$	$4.43e07$ $\pm 1.95e07$
Simulated	Iterations	497.65 ±5.88	20.05 ±1.83	109.35 ±64.52
	Stopping Criteria (grad/loss/field map)	1/0/0/19	0/18/2/0	0/0/20/0
	Func. Evals	532.35 ±28.27	21.05 ±1.83	110.35 ±64.52
	Hessian Evals	N/A	220.55 ±20.13	1872.15 ±1417.11
	Inner Iterations	N/A	10.0000 ±0.00	15.1681 ±3.69
	Loss Value	$6.00e07$ $\pm 1.41e07$	$6.08e07$ $\pm 1.40e07$	$6.12e07$ $\pm 1.43e07$

Table 3.4: Details of optimization for PyHySCO optimizers LBFGS, Gauss Newton, and ADMM. For each dataset we report the average and standard deviation number of iterations, count of stopping criteria used (gradient tolerance/ loss function change tolerance/ field map change tolerance/ maximum iterations), average and standard deviation number of function evaluations, average and standard deviation number of Hessian evaluations, average and standard deviation number of inner iterations, and average and standard deviation loss value. Gauss Newton achieves a similar quality of correction with less computation than LBFGS or ADMM.

		RTX A6000 (GPU)		Quadro RTX 8000 (GPU)		Titan RTX (GPU)		Intel Xeon E5-4627 (CPU)	
		double	single	double	single	double	single	double	single
3T	Runtime (s)	12.7262 ±0.68	9.5750 ±0.58	13.4800 ±1.23	7.8854 ±0.91	13.1178 ±1.31	7.5820 ±0.98	34.3328 ±4.26	27.1305 ±3.09
	Optimization Time (s)	6.7947 ±0.51	4.1562 ±0.39	7.0133 ±1.25	2.1065 ±0.90	6.7862 ±1.25	1.9327 ±0.62	27.8682 ±3.10	23.2062 ±3.07
	Relative Improvement	82.7486 ±3.49	82.7393 ±3.50	82.7486 ±3.49	82.7393 ±3.50	82.7486 ±3.49	82.7393 ±3.50	82.486 ±3.49	82.7393 ±3.50
	Loss Value	2.560e07 ±7.66e06	2.562e07 ±7.69e06	2.560e07 ±7.66e06	2.562e07 ±7.69e06	2.560e07 ±7.66e06	2.562e07 ±7.69e06	2.560e07 ±7.66e06	2.562e07 ±7.69e06
	Smoothness Reg. Value	3.848e04 ±1.21e04	3.851e04 ±1.21e04	3.848e04 ±1.21e04	3.851e04 ±1.21e04	3.848e04 ±1.21e04	3.851e04 ±1.21e04	3.848e04 ±1.21e04	3.851e04 ±1.21e04
7T	Runtime (s)	17.2494 ±0.94	11.9028 ±0.44	21.0102 ±6.31	9.3140 ±0.99	18.6522 ±2.84	9.4680 ±2.19	82.1059 ±7.64	33.4020 ±4.15
	Optimization Time (s)	10.1298 ±0.95	5.7460 ±0.42	11.9937 ±3.55	2.2579 ±0.64	10.9617 ±2.85	2.9775 ±2.11	72.1380 ±6.82	28.5530 ±4.09
	Relative Improvement	85.7618 ±5.10	85.7641 ±5.10	85.7618 ±5.10	85.7642 ±5.10	85.7618 ±5.10	85.7642 ±5.10	85.7618 ±5.10	85.7638 ±5.10
	Loss Value	4.143e07 ±1.95e07	4.140e07 ±1.95e07	4.143e07 ±1.95e07	4.140e07 ±1.95e07	4.143e07 ±1.95e07	4.140e07 ±1.95e07	4.1432e07 ±1.95e07	4.1410e07 ±1.95e07
	Smoothness Reg. Value	5.634e04 ±1.91e04	5.628e04 ±1.91e04	5.634e04 ±1.91e04	5.628e04 ±1.91e04	5.634e04 ±1.91e04	5.628e04 ±1.91e04	5.634e04 ±1.91e04	5.629e04 ±1.91e04
Sim.	Runtime (s)	106.92 ±11.42	50.26 ±3.52	127.38 ±13.42	24.17 ±1.31	125.25 ±14.87	23.60 ±3.78	851.18 ±107.41	417.56 ±55.33
	Optimization Time (s)	89.53 ±11.56	35.53 ±4.13	104.47 ±13.88	7.93 ±0.92	105.78 ±14.84	9.32 ±3.81	827.06 ±108.91	402.04 ±56.58
	Relative Improvement	76.27 ±5.18	76.28 ±5.18	76.27 ±5.18	76.28 ±5.18	76.27 ±5.18	76.28 ±5.18	76.27 ±5.18	76.28 ±5.18
	Loss Value	6.07e07 ±1.39e07	6.08e07 ±1.40e07	6.07e07 ±1.39e07	6.08e07 ±1.40e07	6.07e07 ±1.39e07	6.08e07 ±1.40e07	6.07e07 ±1.39e07	6.08e07 ±1.40e07
	Smoothness Reg. Value	9.53e04 ±2.71e04	9.56e04 ±2.72e04	9.53e04 ±2.71e04	9.56e04 ±2.73e04	9.53e04 ±2.71e04	9.56e04 ±2.73e04	9.54e04 ±2.71e04	9.56e04 ±2.73e04

Table 3.5: The speed and quality of PyHySCO optimization with Gauss Newton on three different GPUs and a CPU in both single (float 32) and double (float 64) precision arithmetic. The relative improvement, loss value, and smoothness value are evaluated in double precision in all cases. Results are shown for both 3T and 7T data from the Human Connectome Project [62] and simulated data. There is a great speedup when calculating in single precision without losing the quality of correction, and the speedup of PyHySCO using a GPU is clear compared to the CPU.

		PyHySCO	HySCO	TOPUP
3T	Runtime (s)	10.37 ±0.87	65.06 ±8.64	4022.56 ±73.11
	Relative Improvement	82.74 ±3.50	78.98 ±6.39	54.36 ±17.08
	Loss Value	2.56e07 ±7.69e06	4.13e07 ±1.38e07	N/A
	Smoothness Reg. Value	3.85e04 ±1.21e04	7.84e04 ±3.01e04	N/A
7T	Runtime (s)	13.62 ±2.38	120.92 ±19.61	3713.51 ±63.04
	Relative Improvement	85.76 ±5.10	80.43 ±10.46	74.51 ±9.13
	Loss Value	4.14e07 ±1.95e07	5.87e07 ±2.48e07	N/A
	Smoothness Reg. Value	5.63e04 ±1.91e04	8.03e04 ±3.68e04	N/A
Simulated	Runtime (s)	55.26 ±3.86	757.65 ±96.26	30854.18 ±568.11
	Relative Improvement	76.28 ±5.18	69.53 ±5.10	17.56 ±28.14
	Loss Value	6.08e07 ±1.40e07	6.07e07 ±1.51e07	N/A
	Smoothness Reg. Value	9.56e04 ±2.72e04	6.10e04 ±1.60e04	N/A
	Relative Error (Field Map)	14.48 ±7.71	19.70 ±11.70	16.37 ±3.60
	SSIM (Field Map)	91.80 ±0.03	86.91 ±0.05	80.15 ±0.08
	SSIM (T2w Image)	99.87 ±0.0017	99.95 ±0.0003	99.96 ±0.0002

Table 3.6: The speed and quality of optimization for TOPUP, HySCO, and PyHySCO. PyHySCO uses Gauss Newton and optimizes in single precision on GPU. HySCO and TOPUP optimize on CPU using the default configurations. Results are reported for 3T and 7T data from the Human Connectome Project [62] and the simulated distortion data.

Chapter 4

Parameterized Four-Dimensional DTI Correction

In this chapter, we extend the distortion correction model of the previous chapters to the four-dimensional setting including diffusion information. In the case of diffusion tensor imaging, the acquired images are four-dimensional with the fourth dimension representing the applied diffusion gradient. The usual practice for distortion correction is to use the field map estimated from a non-diffusion weighted ($\gamma = 0$) pair to then correct each diffusion-weighted volume [60]. However, this has been shown to lead to poor correction in the presence of a large amount of subject motion [5], and the quality of diffusion tensor analysis is adversely affected in areas of uniform contrast in the $\gamma = 0$ image [60].

To improve the quality of both distortion correction and diffusion tensor analysis, we propose to estimate a four-dimensional field map, leveraging the structure of the applied diffusion gradients while allowing for differences due to motion and contrast. We do this using a parameterization of the four-dimensional field map and optimizing the coefficients of the parameterization. Our results show that this structure, using different parameterizations, can yield improved correction and diffusion tensor analy-

sis metrics. We additionally show how the parameterization can be used to decrease computation by optimizing on a subset of diffusion directions and interpolating to the full set of diffusion directions.

In Section 4.1, we introduce the additional notation required for the four-dimensional problem and a four-dimensional field map parameterized in the diffusion direction. In Section 4.2, we define five possible parameterizations of the four-dimensional field map. In Section 4.3, we describe the setup for choosing a subset of diffusion directions for which to optimize a field map and how to interpolate to the full field map. In Section 4.4, we present results of experiments using the various parameterizations in optimization as well as interpolation. In Section 4.5, we provide a summary of this chapter.

4.1 Four-Dimensional Optimization Problem

In the four-dimensional diffusion setting, each image is four-dimensional with three spatial dimensions and one diffusion dimension. Let $S^2 \subset \mathbb{R}^3$ be the sphere containing the set of diffusion directions $\{d_i : i = 0, \ldots, n_d\}$ for a diffusion-weighted observation. See Figure 4.1 for a visualization of these directions. Then the observed, distorted, diffusion weighted image is $\mathcal{I} : S^2 \times \Omega \to \mathbb{R}$.

The goal of optimization is to find the field map $\mathbf{b} : S^2 \times \Omega \to \mathbb{R}$ minimizing (2.5) for a pair of four-dimensional input images. Because the distortions happen only in the phase encoding direction, the additional diffusion dimension increases the size of the problem but does not change the physics of the distortion correction. In particular, the computations that are parallelized in the three-dimensional correction remain completely parallelizable in the distortion dimension.

However, this optimization does not promote smoothness in the diffusion dimension, and it does not incorporate any of the additional information provided by the

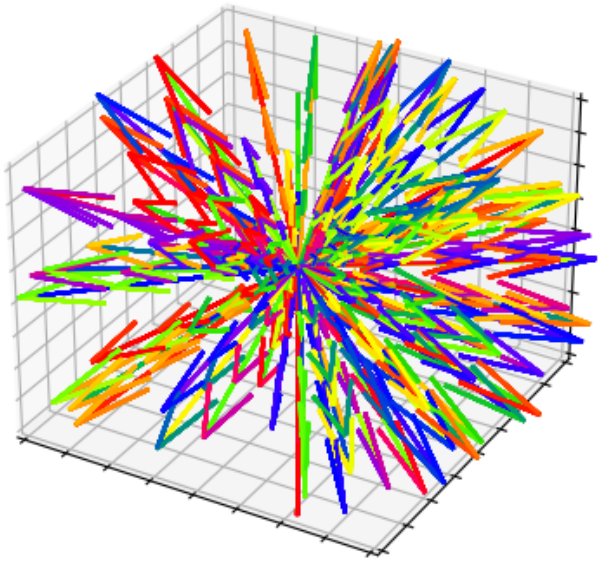


Figure 4.1: The 95 directions of diffusion sensitization gradients for an example volume from HCP.

diffusion directions. We define a parameterization of the field map in the diffusion dimension. This parameterization promotes smoothness among the field maps for similar diffusion directions. Additionally, computation can be lessened by optimizing over a subset of diffusion directions and parameterizing the field map such that for unseen diffusion directions we can quickly interpolate a field map and apply the correction model. Let $\mathbf{Q} \in \mathbb{R}^{n_d \times n_c}$ be the parameterization matrix with n_d the number of diffusion directions and n_c a chosen number of coefficients, and let the coefficients be in $\mathbf{c} \in \mathbb{R}^{n_c \cdot m}$ where m is the three-dimensional image size. Then the field map $\mathbf{b} \in \mathbb{R}^{n_d \cdot m}$ is

$$\mathbf{b} = (\mathbf{Q} \otimes \mathbf{I}_m) \cdot \mathbf{c},$$

where $\mathbf{I}_m$ is the identity matrix in $\mathbb{R}^m$ and $\otimes$ is a Kronecker product. For the complete set of diffusion directions $\{d_i\}_{i=1}^{n_d}$, the objective function is

$$\min_{\mathbf{c}} J_{\mathbf{Q}}(c) = D(\mathbf{c}) + \alpha S(\mathbf{c}) + \beta P(\mathbf{c}), \tag{4.1}$$

where we minimize over the coefficients in $\mathbf{c}$ and after optimization compute the field map $\mathbf{b} = (\mathbf{Q} \otimes \mathbf{I}_m) \cdot \mathbf{c}$.

In implementation, much of the optimization problem is the same as the three-dimensional implementation of Chapter 3, with parallelized computations accounting for the additional dimension. The parameterization $\mathbf{Q}$ is implemented as a `LinearOperator` storing the matrix and allowing for multiplication and transpose multiplication operations. We use Gauss-Newton PCG to optimize (4.1), with fixed hyperparameters $\alpha = 10.0$ and $\beta = 1e-4$.

4.2 Field Map Parameterizations

We consider a collection of field map parameterizations that relate the associated diffusion directions. These parameterizations seek to promote smoothness in the diffusion dimension and leverage the information provided by similar diffusion directions by using the structure of the sphere in which the diffusion directions lie.

4.2.1 Nearest Neighbor Graph Laplacian

A graph parameterization is computed from a nearest-neighbors graph that connects diffusion directions that are close on the sphere. Let $G(V, E)$ be the nearest neighbors graph with the set of diffusion directions $\{d_i\}_{i=1}^{n_d}$ as the vertices of the set V, and edge set E built by including an edge between a direction and the closest k directions using Euclidean distance, weighted by the inverse distance value. The nearest neighbor graphs are visualized for $k = 2$, 5, 10, and 20 neighbors in Figure 4.2. We compute the graph Laplacian $L = D - A$ from the degree matrix D and adjacency matrix A of $G(V, E)$. Let $\lambda_1, \lambda_2, \ldots \lambda_{n_d}$ be the eigenvalues of L, ordered from smallest to largest. Let $\mathcal{E}$ be the set of associated eigenfunctions $\phi_1, \phi_2, \ldots \phi_{n_c}$. We then define

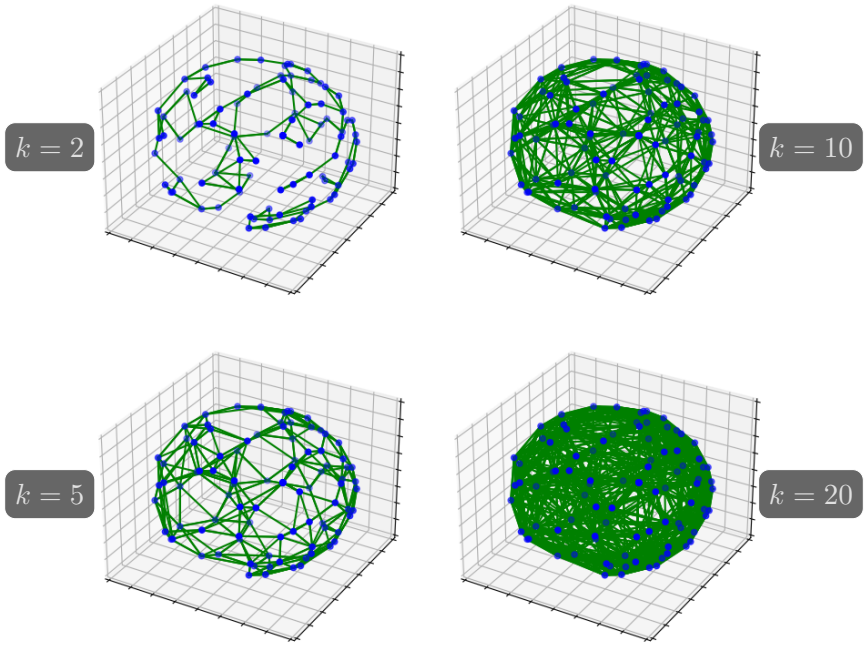


Figure 4.2: Nearest neighbor graph for k = 2, 5, 10, and 20 neighbors.

the parameterization

$$\mathbf{Q}_{graph} = \mathcal{E}_{n_c} = \{\phi_i\}_{i=1}^{n_c},$$

with $n_c \leq n_d$. By taking the n_c eigenfunctions associated with the smallest n_c eigenvalues of the graph Laplacian, we aim to capture the largest patterns of connectedness in the graph Laplacian [18].

The tunable parameters of the Graph Laplacian parameterization are the number of neighbors k in the graph and the number of eigenfunctions n_c taken as the parameterization. The matrix $\mathbf{Q}_{graph}$ is visualized in Figure 4.3 for different parameter options.

4.2.2 Spherical Linear Interpolation

Spherical Linear Interpolation (SLERP) arose in computer graphics as a method to smoothly interpolate between two points on a sphere [55]. Let v_0 and v_1 be vectors

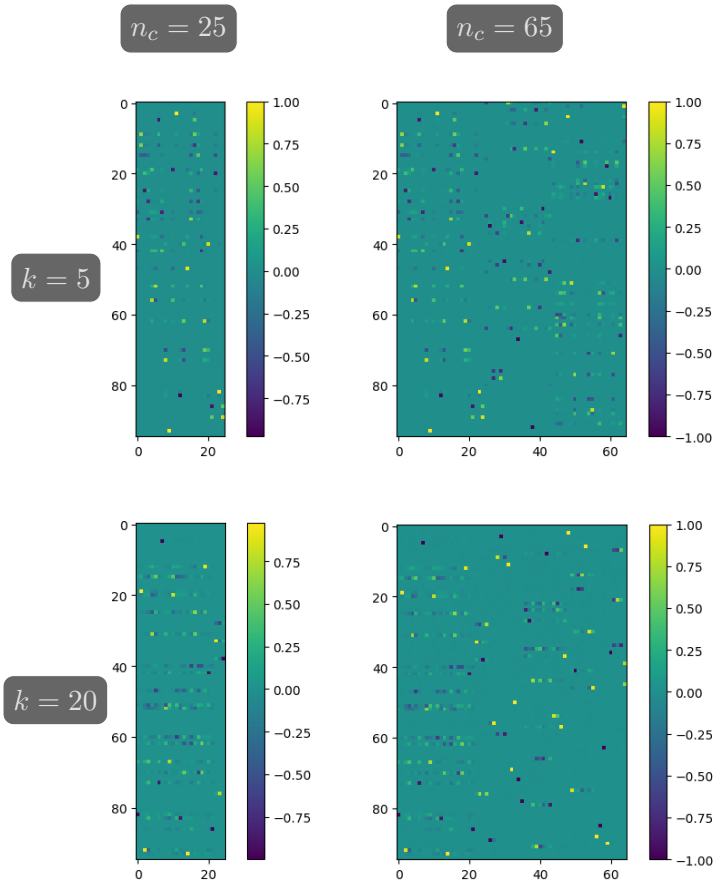


Figure 4.3: Parameterization matrix $\mathbf{Q}_{graph}$ for different parameter options. The top row uses $k = 5$ neighbors in building the graph, and the bottom row uses $k = 20$ neighbors in building the graph. The first column uses $n_c = 25$ eigenfunctions, and the second column uses $n_c = 65$ eigenfunctions.

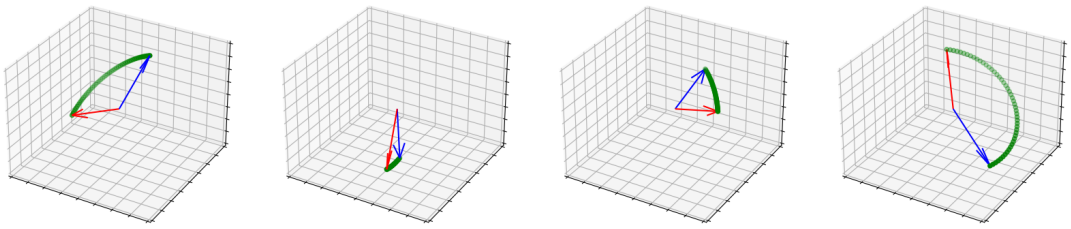


Figure 4.4: Example interpolation paths using spherical linear interpolation (SLERP).

in $\mathbb{R}^3$. Define $\theta = \arccos(v_0 \cdot v_1)$. The spherical linear interpolation between v_0 and v_1, parameterized by t, is

$$v_t = \frac{\sin((1-t)\theta)}{\sin(\theta)} v_0 + \frac{\sin(t\theta)}{\sin(\theta)} v_1.$$

Examples of spherical linear interpolations in $\mathbb{R}^3$ are shown in Figure 4.4. Let $\mathcal{B}$ be a set of n_c basis vectors in $\mathbb{R}^3$. For each diffusion direction d_i, we compute the SLERP weights for the two closest basis vectors v_0, $v_1 \in \mathcal{B}$. We compute t as the projection of d_i onto v_0, normalized by the relative orientations of v_0 and v_1 as

$$t = \frac{d_i \cdot v_0}{v_0 \cdot v_1}.$$

We define the entries in row i of the parameterization matrix $\mathbf{Q}_{slerp}$ as

$$\mathbf{Q}_{slerp}[i,j] = \begin{cases} \frac{\sin((1-t)\theta)}{\sin(\theta)} v_0, & \mathcal{B}(j) = v_0 \\ \frac{\sin(t\theta)}{\sin(\theta)} v_1, & \mathcal{B}(j) = v_1 \\ 0, & \text{otherwise.} \end{cases}$$

This defines a parameterization $\mathbf{Q}_{slerp} \in \mathbb{R}^{n_d \times n_c}$. The tunable parameter of this parameterization is the number of basis vectors n_c. Example parameterization matrices $\mathbf{Q}_{slerp}$ with different number of basis vectors are visualized in Figure 4.5.

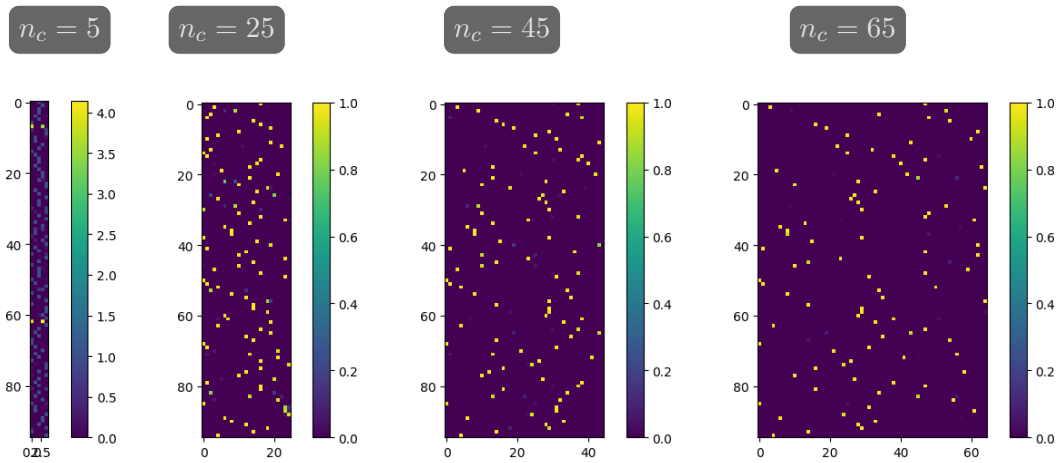


Figure 4.5: Parameterization matrix $\mathbf{Q}_{slerp}$ for different number of basis vectors $n_c = 5, 25, 45, 65, 95$.

4.2.3 Spherical Harmonics

Spherical Harmonics are a set of functions defined on the surface of a sphere that form an orthonormal basis for all functions defined on the surface of a sphere [9]. These functions are computed by solving Laplace's equation on the sphere [38, Ch 1, Appendix B, page 171]. The complex spherical harmonic function with order m and degree l at azimuthal coordinate θ and polar coordinate ϕ is

$$Y_l^m(\theta, \phi) = \sqrt{\frac{2l+1}{4\pi}\frac{(l-m)!}{(l+m)!}} e^{im\theta} P_l^m(cos(\phi)),$$

where P_l^m is the associated Legendre function of order m and degree l. The real spherical harmonic functions $Y_{l,m}$ as defined in [9] are

$$Y_{l,m} = \begin{cases} \frac{(-1)^m}{\sqrt{2}}\mathcal{R}(Y_l^m), & m > 0 \\ Y_l^m, & m = 0 \\ \frac{(-1)^m}{\sqrt{2}}\mathcal{I}(Y_l^m), & m < 0, \end{cases}$$

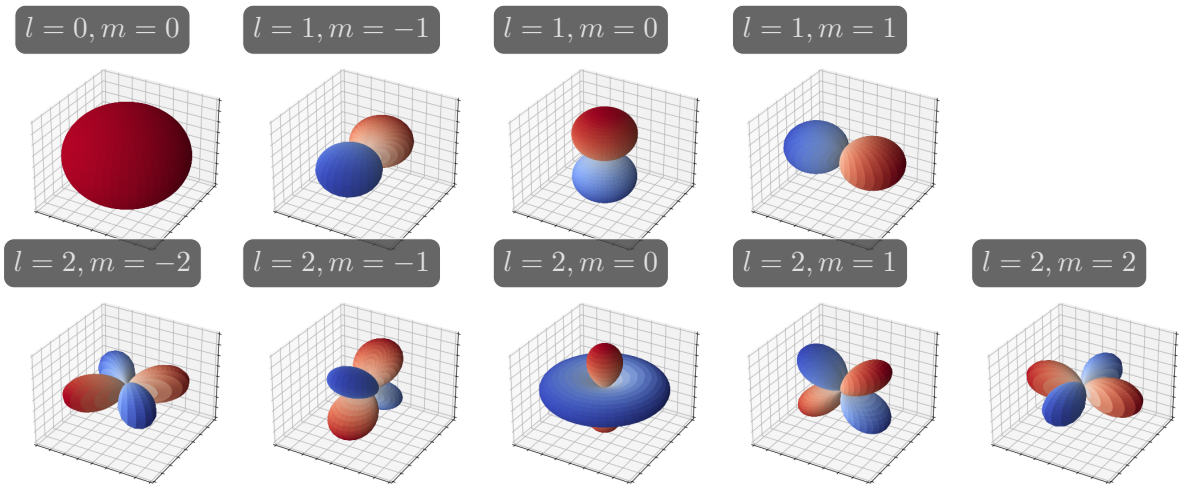


Figure 4.6: Spherical harmonic functions for $l = 0, 1, 2$.

where $\mathcal{R}$ and $\mathcal{I}$ take the real and imaginary parts of the input, respectively. The real spherical harmonic functions of degree $l = 0, 1, 2$ are visualized in Figure 4.6. By computing the spherical harmonic coefficients for each diffusion direction for degree up to n_c and order varying for each degree value l between $-l$ and l, we construct $\mathbf{Q}_{sph} \in \mathbb{R}^{n_d \times n_c^2}$. In particular, for direction d_i with spherical coordinate (r, θ, ϕ), row i of $\mathbf{Q}_{sph}$ is

$$\mathbf{Q}_{sph}[i, :] = [Y_{0,0}(\theta, \phi), Y_{1,-1}(\theta, \phi), Y_{1,0}(\theta, \phi), Y_{1,1}(\theta, \phi), \ldots, Y_{n_c,n_c}(\theta, \phi)].$$

The tunable parameter of this parameterization is n_c, the maximum degree of spherical harmonic function used. Figure 4.7 shows the parameterization matrix $\mathbf{Q}_{sph}$ for maximum degrees of $n_c = 1, 2, 3$, and 4.

4.2.4 Inverse Distance Weighting

Inverse distance weighting was first defined as an interpolation method with application to computing mapping in Geographical Information Systems [54]. For a distance

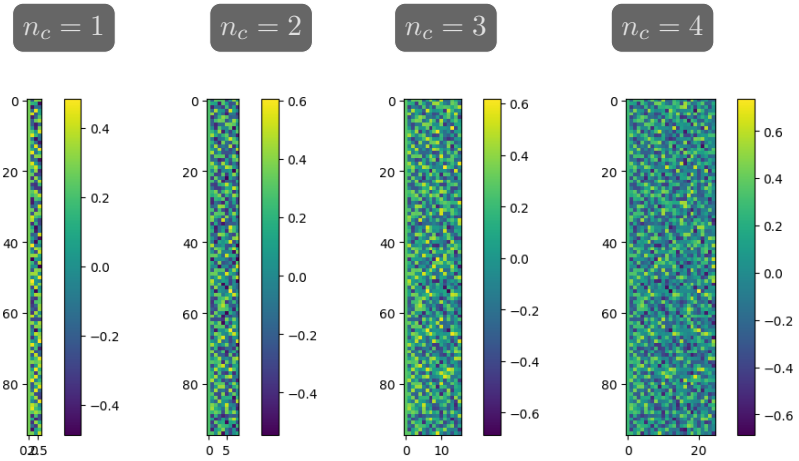


Figure 4.7: Parameterization matrix $\mathbf{Q}_{sph}$ for different values of maximum degree $n_c = 1, 2, 3, 4$.

function δ, define the weight between two vectors in $\mathbb{R}^3$ as

$$w(x, y) = \frac{1}{\delta(x, y)^2}.$$

This weighting favors closer vectors, with the weight decreasing as distance increases. For a set of n_c basis vectors $\mathcal{B} = \{v_1, v_2, \cdots, v_{n_c}\} \subset \mathbb{R}^3$, we define the entries in a row i of the parameterization matrix $\mathbf{Q}_{idw}$ as

$$\mathbf{Q}_{idw}[i, j] = \frac{w(d_i, v_j)}{\sum_{k=1}^{n_c} w(d_i, v_k)}.$$

This defines a parameterization $\mathbf{Q}_{idw} \in \mathbb{R}^{n_d \times n_c}$. The tunable parameters in this parameterization are the number of basis vectors n_c and the choice of distance function δ. Figure 4.8 shows the parameterization matrix $\mathbf{Q}_{idw}$ for different numbers of basis vectors and choice of distance function.

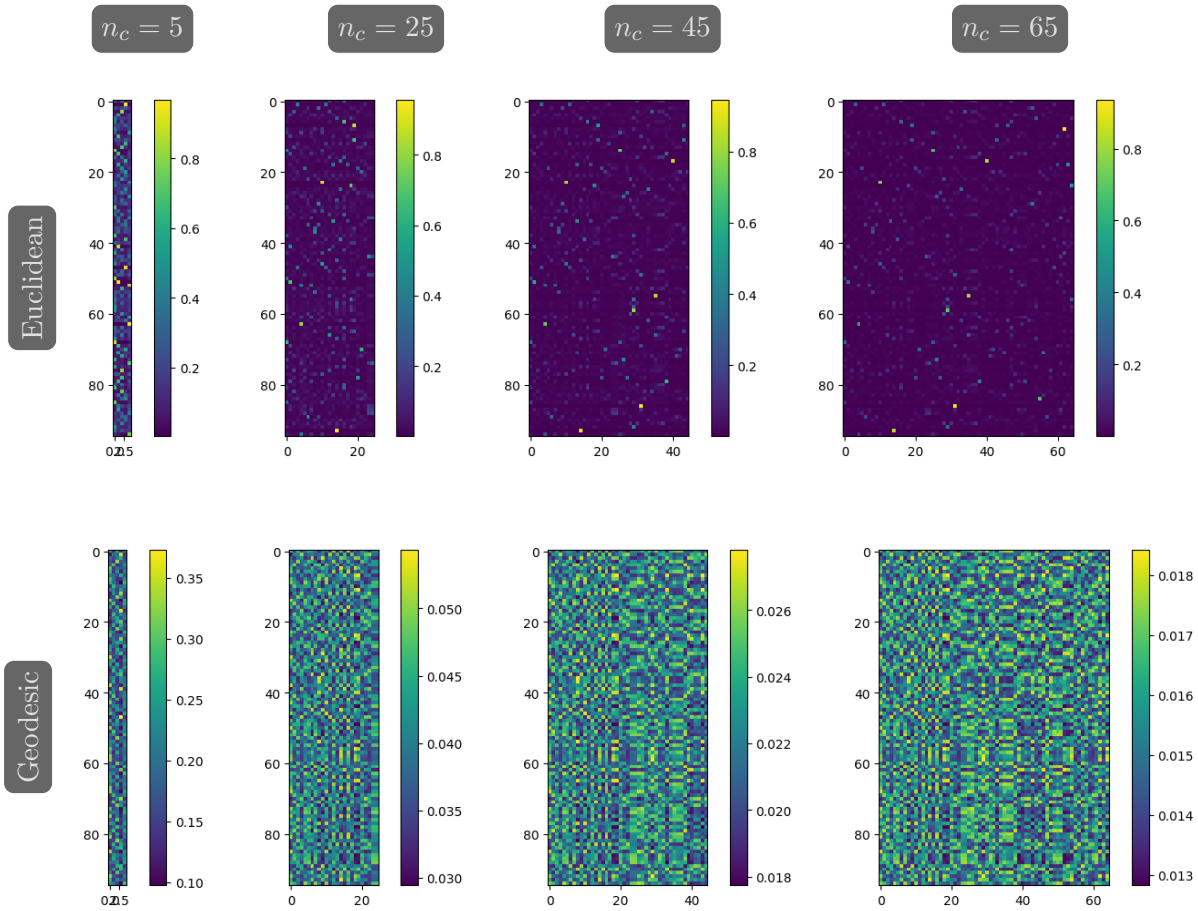


Figure 4.8: Parameterization matrix $\mathbf{Q}_{idw}$ for different number of basis vectors $n_c = 5, 25, 45, 65$ and using Euclidean vs. Geodesic distance.

4.2.5 Radial Basis Functions

A radial basis function parameterization is defined by applying a kernel function to relate the diffusion directions and the vectors of a basis. These functions have been studied as powerful interpolations on spherical data [13, 27].

The Gaussian kernel function for a distance function δ has the form

$$\phi_{gaussian}(\delta, x, y) = e^{\frac{-\delta(x,y)^2}{2\sigma^2}},$$

where the width of the kernel is controlled by the parameter σ. A thin plate spline kernel for a distance function δ is

$$\phi_{tps}(\delta, x, y) = \delta(x, y)^2 \ln \delta(x, y).$$

For a set of n_c basis vectors $\mathcal{B} = \{v_1, v_2, \cdots, v_{n_c}\} \subset \mathbb{R}^3$, we define the entries in a row i of the parameterization matrix $\mathbf{Q}_{rbf}$ as

$$\mathbf{Q}_{rbf}[i, j] = \phi(\delta, d_i, v_j).$$

The parameters of this parameterization are the number of basis vectors n_c, which kernel function ϕ is used, and the distance function δ. Figure 4.9 shows the parameterization matrix $\mathbf{Q}_{rbf}$ for different choices of distance and kernel function.

4.3 Cluster, Reduce, Optimize, and Interpolate

Beyond introducing smoothness in the diffusion dimension, the parameterizations described in the previous section can be used to define an interpolation that allows for optimizing on a subset of diffusion direction volumes and using the interpolation to apply the correction to the full 4D volume. This is done by clustering to choose a

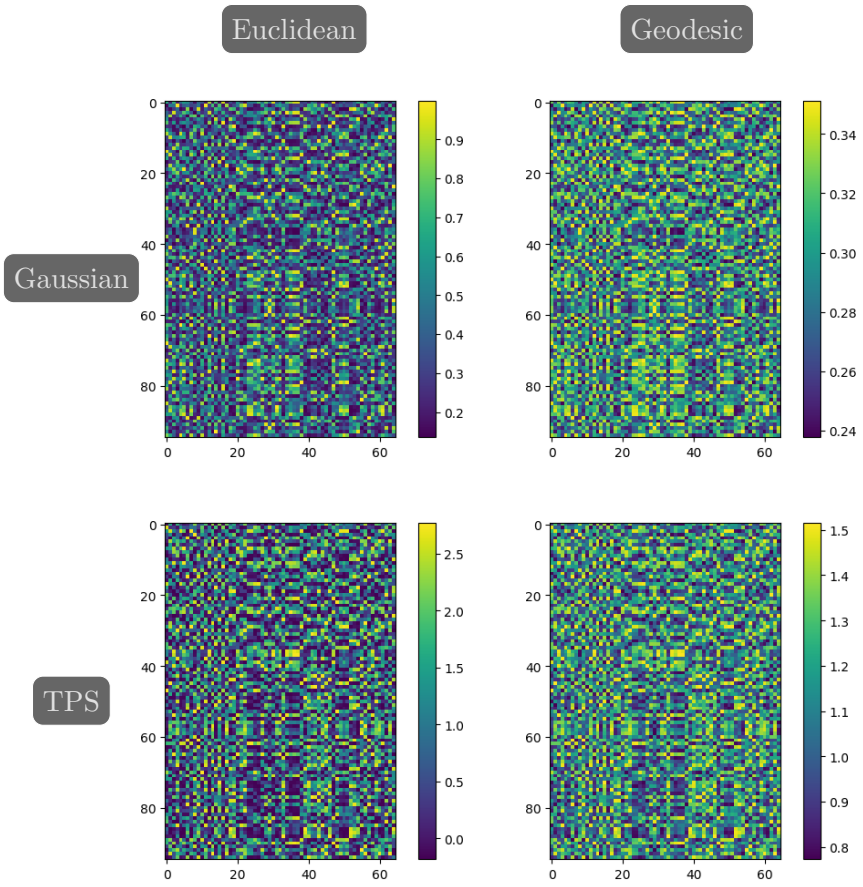


Figure 4.9: Parameterization matrix $\mathbf{Q}_{rbf}$ for $n_c = 65$ basis vectors, using Euclidean vs. Geodesic distance, and using Gaussian vs. Thin Plate Spline kernels.

subset of representative diffusion directions, reducing the optimization inputs to those chosen directions, optimizing on the subset, and interpolating back to the full volume. This significantly reduces the cost of optimization as the memory and compute time requirements of the full four-dimensional optimization could be prohibitive, especially if the image size is large.

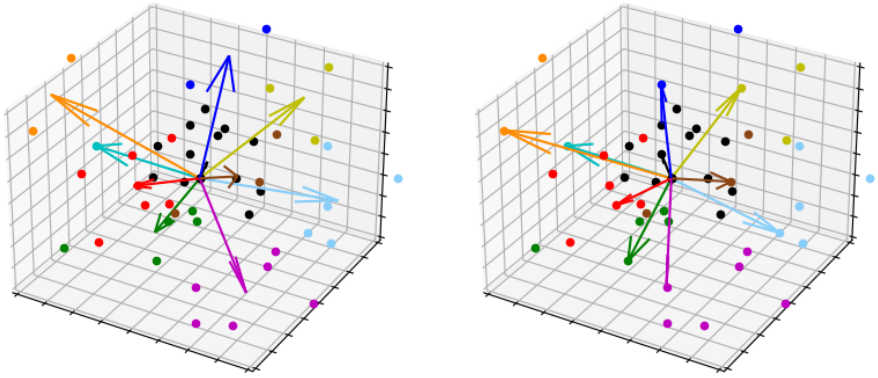
First, we cluster the diffusion directions into groups of related diffusion directions using k-means clustering [41]. The algorithm begins by randomly assigning k diffusion directions as the centroids of each cluster. At each step, the diffusion directions are assigned to the cluster associated with the closest centroid, and the centroids are updated to be the average of all of the directions in the cluster. This repeats until the centroids stop significantly changing. An example clustering of directions is shown in Figure 4.10a.

To reduce the size of the volume for optimization, we select a subset of diffusion directions by taking the diffusion direction closest to the centroid for each cluster found in the previous step. This is illustrated in Figure 4.10b. This gives a subset of diffusion directions of size $n_{interp} \leq k << n_d$.

We then solve (4.1) for the smaller four-dimensional volume corresponding to the subset of diffusion directions from the previous step using a parameterization $\mathbf{Q} \in \mathbb{R}^{n_{interp} \times n_c}$ defined on the subset of diffusion directions. After optimization, the optimized coefficients $\mathbf{c}$ are used along with a parameterization $\mathbf{Q}^{full} \in \mathbb{R}^{n_d \times n_c}$ defined on the full subset of diffusion directions to compute a four-dimensional field map corresponding to the full four-dimensional volume.

4.4 Results

In this section, we report the quality of optimization in 4D using the different parameterizations described in Section 4.2. The data used in these experiments are



(a) Clustering of diffusion directions. The arrows show the centroids of each cluster.

(b) The same clusters as in (a) with the arrows showing the chosen direction for each cluster.

Figure 4.10: The clusters and centroids (a) and clusters and chosen directions (b) using $k = 10$ and 50 diffusion directions.

7 four-dimensional diffusion-weighted images from the Human Connectome Project [62]. These images are 3T strength, with 95 diffusion directions and left-right phase encoding. In all cases, the hyperparameters of (4.1) are $\alpha = 10$ and $\beta = 1\text{e} - 4$.

In 4.4.1, we describe the metrics used to evaluate the four-dimensional parameterized correction. In 4.4.2, we report the results of optimization with different choices of parameterization. In 4.4.3, we report the results of optimization on a subset of diffusion directions and interpolating the field map for the full set of diffusion directions.

4.4.1 Metrics

We evaluate correction in the four-dimensional setting using both metrics of the correction quality and metrics from diffusion tensor imaging. We report the optimization time in seconds, measured using Python's `time` module. We compute the loss function value, distance value, and smoothness regularization value using the four-dimensional field map and four-dimensional optimization problem (4.1). For diffusion tensor met-

rics, we report the diffusion tensor fit loss (2.14) and the median standard deviation of the fractional anisotropy maps (2.15) computed for each input phase encoding direction [65].

4.4.2 Comparison of Parameterizations

In this section, we experiment with the different parameterizations and the tunable parameters for each parameterization. In each case, we report the results of the 4D correction using the parameterization in the optimization, and we compare these results with correction using the $\gamma = 0$ field map to correct for all diffusion directions. The input, baseline $\gamma = 0$, and representative examples for each parameterization are shown in Figures 4.11 (no diffusion weighting) and 4.12 (with diffusion weighting) for an example subject.

For the graph parameterization, we vary the number of neighbors k used in the graph between 2, 5, 10, and 20, and we vary the number of eigenfunctions n_c between 5, 25, 45, 65, and 95. These results are reported in Table 4.1. Compared to applying the $\gamma = 0$ field map, the 4D graph parameterizations always improve the smoothness. The best results are with $n_c = 95$ eigenfunctions, achieving an average relative improvement, relative to applying the $\gamma = 0$ field map, of 76.43% in distance, 78.56% in smoothness, and 5.15% in median standard deviation of fractional anisotropy. However, the graph parameterization is unable to improve the diffusion tensor loss.

For the SLERP parameterization, we vary the number of basis vectors n_c between 5, 25, 45, 65, and 95. These results are reported in Table 4.2. Relative to applying the $\gamma = 0$ field map, the 4D SLERP parameterizations have an average relative improvement of 41.37% in distance, 83.30% in smoothness, 2.32% in diffusion tensor loss, and 2.13% in median standard deviation of fractional anisotropy.

For the spherical harmonics parameterization, we vary the value of n_c, the max-

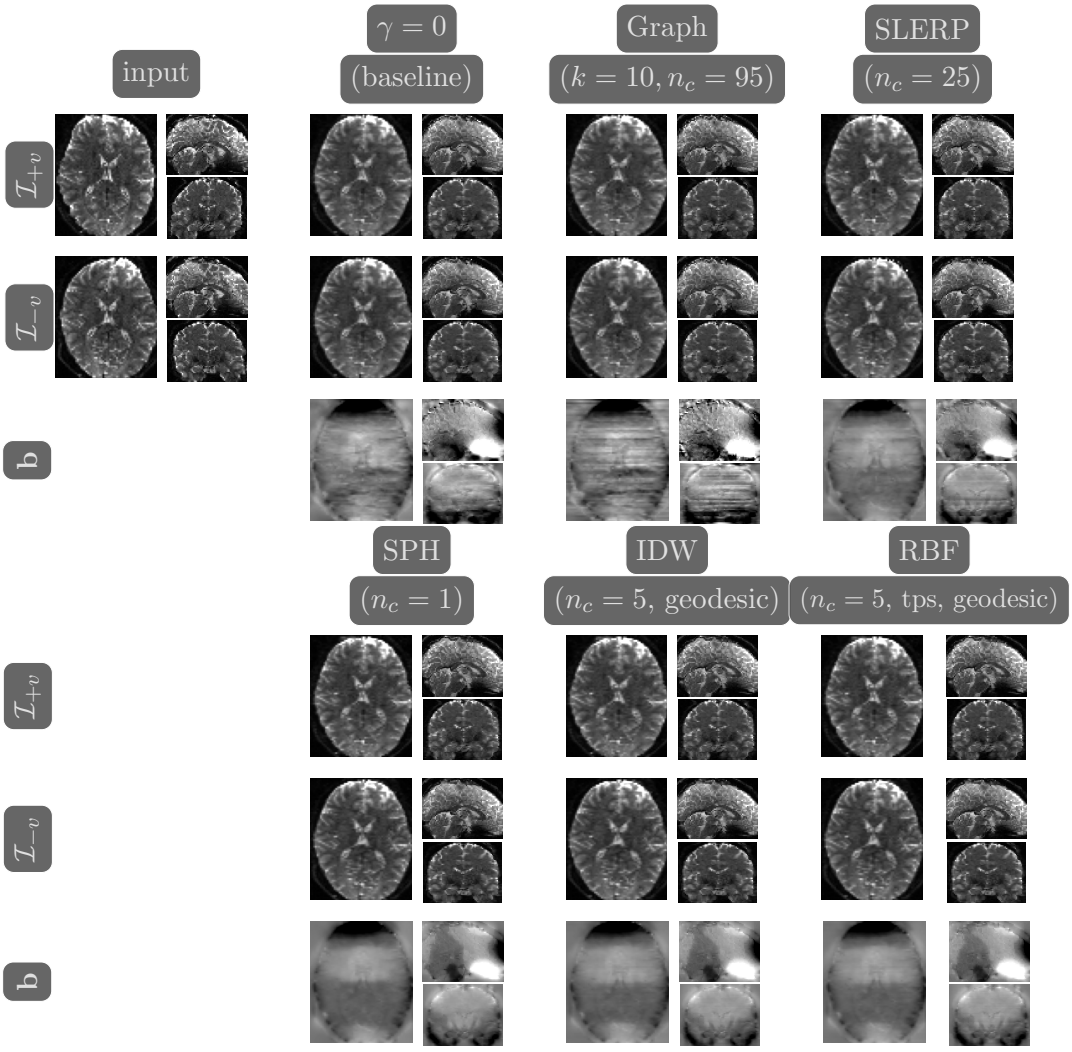


Figure 4.11: Visualization of images and field maps for one subject (Subject ID 100206) with no diffusion weighting. The first column in the top half shows the input data, the second column shows the $\gamma = 0$ baseline correction, and the remaining columns show the results using a graph parameterization with $k = 10$ and $n_c = 95$, a spherical linear interpolation parameterization using $n_c = 25$, a spherical harmonic parameterization using $n_c = 1$, an inverse distance weighting parameterization using $n_c = 5$ and geodesic distance, and a radial basis function parameterization using $n_c = 5$, geodesic distance, and thin plate spline kernel. For each optimization, the corrected images with opposite phase encoding direction and field map are shown. The corrected images are of similar quality, and spherical harmonic, inverse distance weighting, and radial basis function parameterizations have more smooth field maps.

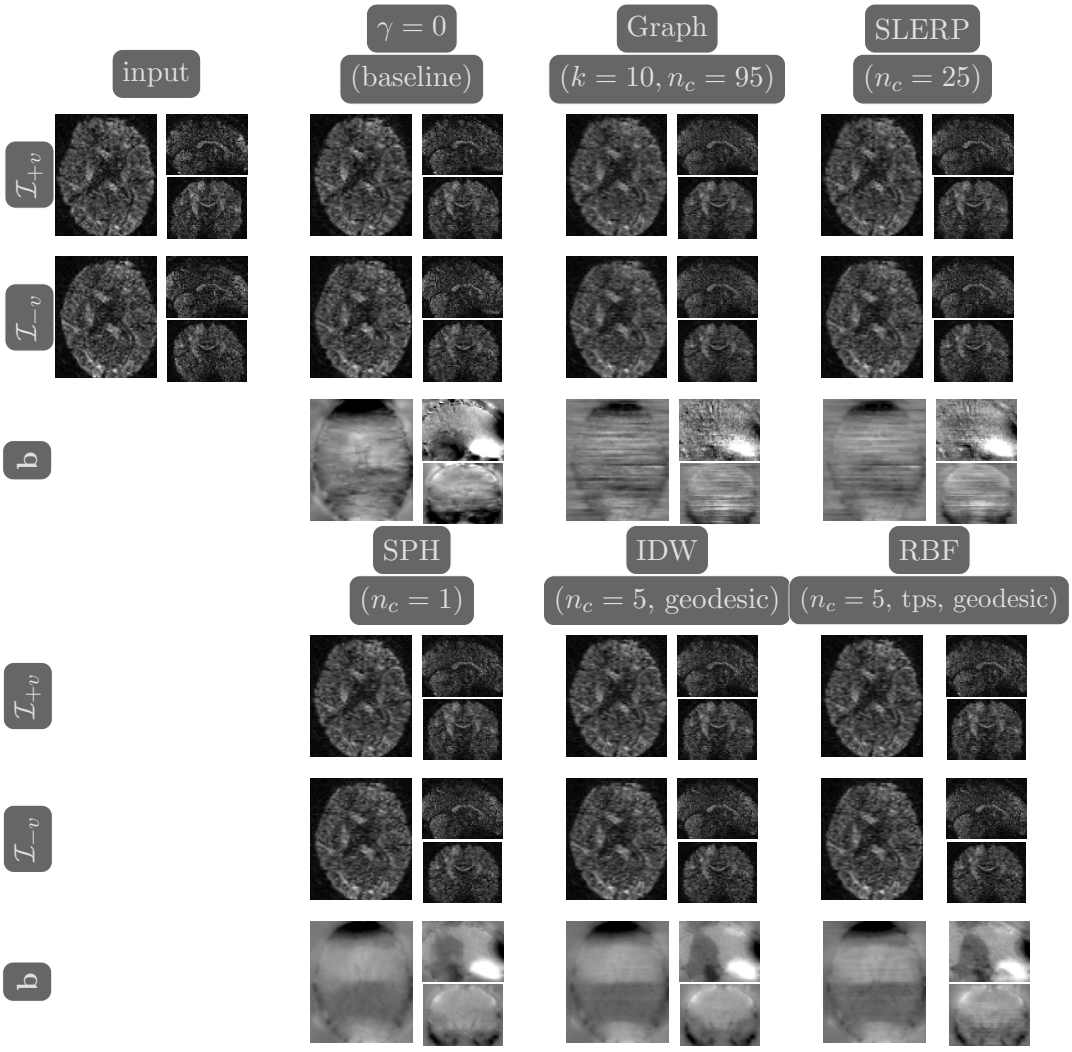


Figure 4.12: Visualization of images and field maps for one subject (Subject ID 100206) with diffusion weighting. The first column in the top half shows the input data, the second column shows the $\gamma = 0$ baseline correction, and the remaining columns show the results using a graph parameterization with $k = 10$ and $n_c = 95$, a spherical linear interpolation parameterization using $n_c = 25$, a spherical harmonic parameterization using $n_c = 1$, an inverse distance weighting parameterization using $n_c = 5$ and geodesic distance, and a radial basis function parameterization using $n_c = 5$, geodesic distance, and thin plate spline kernel. For each optimization, the corrected images with opposite phase encoding direction and field map are shown. The corrected images are of similar quality, and spherical harmonic, inverse distance weighting, and radial basis function parameterizations have more smooth field maps.

imum degree of the spherical harmonic functions, between 1, 2, 3, and 4. These results are reported in Table 4.3. Compared to applying the $\gamma = 0$ field map, all of the spherical harmonic parameterizations improve the distance, smoothness, diffusion tensor loss, and median standard deviation of fractional anisotropy. The 4D spherical harmonic parameterizations have an average relative improvement of 48.12% in distance, 81.99% in smoothness, 2.55% in diffusion tensor loss, and 4.42% in median standard deviation of fractional anisotropy.

For the inverse distance weighting parameterization, we vary the number of basis vectors between $n_c = 5, 25, 45, 65$, and 95, and the distance function between euclidean and geodesic distance. These results are reported in Table 4.4. Compared to applying the $\gamma = 0$ field map, the inverse distance weighting parameterizations improve the distance and smoothness. With euclidean distance, the 4D inverse distance weighting parameterizations have average relative improvement of 54.87% in distance, 74.49% in smoothness, 3.37% in diffusion tensor loss, and 5.64% in median standard deviation of fractional anisotropy. With geodesic distance, the 4D inverse distance weighting parameterizations have average relative improvement of 49.76% in distance, 42.70% in smoothness, 1.52% in diffusion tensor loss, and 3.81% in median standard deviation of fractional anisotropy.

For the radial basis function parameterization, we vary the number of basis vectors between $n_c = 5, 25, 45, 65$, and 95, the distance function between euclidean and geodesic distance, and the kernel function between Gaussian and thin plate spline. These results are reported in Table 4.5. Using a Gaussian kernel and euclidean distance, the 4D radial basis function parameterizations have an average relative improvement of 52.29% in distance, 47.08% in smoothness, 1.91% in diffusion tensor loss, and 4.50% in median standard deviation of fractional anisotropy. Using a Gaussian kernel and geodesic distance, the 4D radial basis function parameterizations have an average relative improvement of 48.22% in distance, 47.44% in smoothness,

	Opt. Time (s)	Loss Value	Distance Value	Smoothness Reg. Value	DTI Loss	std FA
input		1.4981e+09	1.4981e+09		7.6348e+07	0.0797
$\gamma = 0$ (baseline)		9.9229e+08	5.5542e+08	2.7707e+06	3.9761e+07	0.0582
$k = 2, n_c = 5$	475.4539	1.1333e+09	1.0846e+09	**3.0922e+05**	5.2757e+07	0.0712
$k = 2, n_c = 25$	535.8321	**6.2507e+08**	5.5859e+08	**4.2165e+05**	4.9718e+07	0.06352
$k = 2, n_c = 45$	429.7580	**4.3078e+08**	**3.5440e+08**	**4.8444e+05**	4.0825e+07	0.0590
$k = 2, n_c = 65$	528.4616	**3.2658e+08**	**2.4202e+08**	**5.3635e+05**	4.0782e+07	**0.0576**
$k = 2, n_c = 95$	522.0227	**2.2436e+08**	**1.3082e+08**	**5.9326e+05**	4.0185e+07	**0.0551**
$k = 5, n_c = 5$	431.8051	**9.6550e+08**	9.1174e+08	**3.4093e+05**	5.3547e+07	0.0691
$k = 5, n_c = 25$	501.8586	**6.3213e+08**	5.6462e+08	**4.2815e+05**	5.0411e+07	0.0643
$k = 5, n_c = 45$	440.1367	**4.3740e+08**	**3.5891e+08**	**4.9779e+05**	4.1042e+07	0.0597
$k = 5, n_c = 65$	450.2267	**3.2924e+08**	**2.4406e+08**	**5.4026e+05**	4.1170e+07	**0.0581**
$k = 5, n_c = 95$	449.4204	**2.2703e+08**	**1.3115e+08**	**6.0811e+05**	4.0587e+07	**0.0555**
$k = 10, n_c = 5$	421.5404	1.1624e+09	1.1130e+09	**3.1349e+05**	4.4568e+07	0.0696
$k = 10, n_c = 25$	430.8503	**5.9663e+08**	**5.2628e+08**	**4.4613e+05**	4.1363e+07	0.0616
$k = 10, n_c = 45$	484.8360	**4.3966e+08**	**3.6428e+08**	**4.7807e+05**	4.0761e+07	0.0590
$k = 10, n_c = 65$	501.5646	**3.2532e+08**	**2.4321e+08**	**5.2073e+05**	4.0558e+07	**0.0572**
$k = 10, n_c = 95$	348.8677	**2.2305e+08**	**1.3060e+08**	**5.8637e+05**	4.0048e+07	**0.0550**
$k = 20, n_c = 5$	434.4051	1.2636e+09	1.2260e+09	2.3875e+05	4.9499e+07	0.0742
$k = 20, n_c = 25$	466.1244	**6.8500e+08**	6.1287e+08	**4.5746e+05**	4.2344e+07	0.0629
$k = 20, n_c = 45$	471.2274	**5.3714e+08**	**4.5741e+08**	**5.0566e+05**	4.1648e+07	0.0613
$k = 20, n_c = 65$	508.0774	**3.4726e+08**	**2.6513e+08**	**5.2089e+05**	4.1416e+07	**0.0580**
$k = 20, n_c = 95$	318.2721	**2.2384e+08**	**1.3101e+08**	**5.8879e+05**	4.0237e+07	**0.0552**

Table 4.1: Results using the graph parameterization with different number of neighbors and chosen eigenfunctions. We report the optimization time, loss value, distance value, smoothness regularizer value, diffusion tensor imaging loss, and the median standard deviation of fractional anisotropy. For all metrics, lower is better. Values in bold are better than the value for that metric for applying the $\gamma = 0$ field map to all diffusion directions. The best results use $n_c = 65$ or 95 eigenfunctions, though no graph parameterization improves diffusion tensor loss.

0.95% in diffusion tensor loss, and 3.13% in median standard deviation of fractional anisotropy. Using a thin plate spline kernel and euclidean distance, the 4D radial basis function parameterizations have an average relative improvement of 53.03% in distance, 51.40% in smoothness, 2.39% in diffusion tensor loss, and 4.67% in median standard deviation of fractional anisotropy. Using a thin plate spline kernel and geodesic distance, the 4D radial basis function parameterizations have an average relative improvement of 49.66% in distance, 44.29% in smoothness, 0.69% in diffusion tensor loss, and 2.92% in median standard deviation of fractional anisotropy.

	Opt. Time (s)	Loss Value	Distance Value	Smoothness Reg. Value	DTI Loss	std FA
input			1.4981e+09		7.6348e+07	0.0797
$\gamma = 0$ (baseline)		9.9229e+08	5.5542e+08	2.7707e+06	3.9761e+07	0.0582
$n_c = 5$	317.6409	**5.9724e+08**	**5.3836e+08**	**3.7345e+05**	**3.9429e+07**	0.0595
$n_c = 25$	328.7794	**4.3465e+08**	**3.6568e+08**	**4.3746e+05**	**3.8847e+07**	**0.0574**
$n_c = 45$	373.8476	**3.5281e+08**	**2.7487e+08**	**4.9436e+05**	**3.8742e+07**	**0.0567**
$n_c = 65$	401.9963	**3.1396e+08**	**2.3591e+08**	**4.9503e+05**	**3.8567e+07**	**0.0557**
$n_c = 95$	403.9413	**2.9415e+08**	**2.1326e+08**	**5.1301e+05**	**3.8609e+07**	**0.0555**

Table 4.2: Results using the spherical linear interpolation parameterization with different size basis. We report the optimization time, loss value, distance value, smoothness regularizer value, diffusion tensor imaging loss, and the median standard deviation of fractional anisotropy. For all metrics, lower is better. Values in bold are better than the value for that metric for applying the $\gamma = 0$ field map to all diffusion directions. Using the SLERP parameterization, all metrics improve except for median standard deviation of fractional anisotropy for $n_c = 5$.

	Opt. Time (s)	Loss Value	Distance Value	Smoothness Reg. Value	DTI Loss	std FA
input			1.4981e+09		7.6348e+07	0.0797
$\gamma = 0$ (baseline)		9.9229e+08	5.5542e+08	2.7707e+06	3.9761e+07	0.0582
$n_c = 1$	359.9761	**4.0404e+08**	**3.2395e+08**	**5.0793e+05**	**3.8400e+07**	**0.0559**
$n_c = 2$	377.7681	**3.7151e+08**	**2.9394e+08**	**4.9198e+05**	**3.8581e+07**	**0.0556**
$n_c = 3$	471.3437	**3.5281e+08**	**2.7513e+08**	**4.9265e+05**	**3.8844e+07**	**0.0555**
$n_c = 4$	453.0226	**3.3885e+08**	**2.5948e+08**	**5.0341e+05**	**3.9166e+07**	**0.0555**

Table 4.3: Results using the spherical harmonics parameterization with different maximum degree. We report the optimization time, loss value, distance value, smoothness regularizer value, diffusion tensor imaging loss, and the median standard deviation of fractional anisotropy. For all metrics, lower is better. Values in bold are better than the value for that metric for applying the $\gamma = 0$ field map to all diffusion directions. In all cases, the distance, smoothness, diffusion tensor loss, and median standard deviation of fractional anisotropy improves using the parameterization.

	Opt. Time (s)	Loss Value	Distance Value	Smoothness Reg. Value	DTI Loss	std FA
input			1.4981e+09		7.6348e+07	0.0797
$\gamma = 0$ (baseline)		9.9229e+08	5.5542e+08	2.7707e+06	3.9761e+07	0.0582
$n_c = 5$, euclidean	305.9190	**4.3102e+08**	**3.4698e+08**	**5.3298e+05**	**3.8226e+07**	**0.0561**
$n_c = 5$, geodesic	402.6946	**4.0513e+08**	**3.2285e+08**	**5.2183e+05**	**3.8188e+07**	**0.0557**
$n_c = 25$, euclidean	398.1210	**3.6163e+08**	**2.7833e+08**	**5.2826e+05**	**3.8295e+07**	**0.0553**
$n_c = 25$, geodesic	35.8501	**4.8012e+08**	**3.0351e+08**	**1.1201e+06**	**3.9070e+07**	**0.0562**
$n_c = 45$, euclidean	323.3606	**3.2532e+08**	**2.3352e+08**	**5.8219e+05**	**3.8405e+07**	**0.0545**
$n_c = 45$, geodesic	21.5846	**5.4160e+08**	**2.8555e+08**	**1.6239e+06**	**3.9624e+07**	**0.0566**
$n_c = 65$, euclidean	505.5000	**3.2869e+08**	**2.0944e+08**	**7.5635e+05**	**3.8376e+07**	**0.0543**
$n_c = 65$, geodesic	22.9066	**5.8554e+08**	**2.5496e+08**	**2.0966e+06**	**3.9472e+07**	**0.0559**
$n_c = 95$, euclidean	163.8176	**3.6378e+08**	**1.8494e+08**	**1.1343e+06**	**3.8798e+07**	**0.0544**
$n_c = 95$, geodesic	23.6751	**6.3437e+08**	**2.2832e+08**	**2.5753e+06**	**3.9424e+07**	**0.0555**

Table 4.4: Results using the inverse distance weighting parameterization with different size basis and distance function. We report the optimization time, loss value, distance value, smoothness regularizer value, diffusion tensor imaging loss, and the median standard deviation of fractional anisotropy. For all metrics, lower is better. Values in bold are better than the value for that metric for applying the $\gamma = 0$ field map from the identity parameterization for all diffusion directions. All metrics improve, regardless of basis size or distance function.

4.4.3 Clustering and Interpolation

In this section, we experiment with the interpolation setup described in Section 4.3. In all cases, the maximum number of clusters and therefore chosen directions for optimization is 30. The input data, $\gamma = 0$ baseline correction, and corrected images for a representative example of each parameterization are shown in Figures 4.13 (non-diffusion weighted) and 4.14 (diffusion weighted and not in the optimization subset of directions).

For the graph parameterization, we vary the number of neighbors between $k = 2, 5,$ and 10, and the number of selected eigenfunctions n_c between 5, 10, 15, and 25. Table 4.6 shows the results of using the field map optimized on a subset of directions and the graph parameterization to interpolate a field map for the entire 4D volume including all diffusion directions. In the case of the graph parameterization interpolation, we use a weighted combination of the corresponding field map for the nearest neighbors of each diffusion direction as defined in the graph used in the parameterization. While

	Opt. Time (s)	Loss Value	Distance Value	Smoothness Reg. Value	DTI Loss	std FA
input			1.4981e+09		7.6348e+07	0.0797
$\gamma = 0$ (baseline)		9.9229e+08	5.5542e+08	2.7707e+06	3.9761e+07	0.0582
$n_c = 5$, gaussian, euclidean	360.0573	**4.0616e+08**	**3.2708e+08**	**5.0157e+05**	**3.8353e+07**	**0.0558**
$n_c = 5$, gaussian, geodesic	465.5387	**4.0372e+08**	**3.2036e+08**	**5.2870e+05**	**3.8261e+07**	**0.0557**
$n_c = 5$, tps, euclidean	309.9186	**4.2201e+08**	**3.4602e+08**	**4.8199e+05**	**3.8839e+07**	**0.0568**
$n_c = 5$, tps, geodesic	440.1784	**4.0724e+08**	**3.2731e+08**	**5.0692e+05**	**3.8419e+07**	**0.0562**
$n_c = 25$, gaussian, euclidean	394.5604	**4.1378e+08**	**2.7635e+08**	**8.7162e+05**	**3.8399e+07**	**0.0552**
$n_c = 25$, gaussian, geodesic	21.2722	**4.9152e+08**	**3.1203e+08**	**1.1384e+06**	3.9900e+07	**0.0574**
$n_c = 25$, tps, euclidean	522.2779	**3.9396e+08**	**2.7227e+08**	**7.7180e+05**	**3.8389e+07**	**0.0553**
$n_c = 25$, tps, geodesic	22.0514	**4.8429e+08**	**3.0827e+08**	**1.1164e+06**	3.9876e+07	**0.0574**
$n_c = 45$, gaussian, euclidean	25.3961	**5.0522e+08**	**2.7047e+08**	**1.4889e+06**	**3.9642e+07**	**0.0563**
$n_c = 45$, gaussian, geodesic	22.8905	**5.3742e+08**	**2.8443e+08**	**1.6046e+06**	**3.9636e+07**	**0.0565**
$n_c = 45$, tps, euclidean	333.1521	**4.4473e+08**	**2.4795e+08**	**1.2480e+06**	**3.8515e+07**	**0.0549**
$n_c = 45$, tps, geodesic	21.7468	**5.3340e+08**	**2.8292e+08**	**1.5886e+06**	3.9997e+07	**0.0571**
$n_c = 65$, gaussian, euclidean	28.5884	**5.4845e+08**	**2.4213e+08**	**1.9428e+06**	**3.9281e+07**	**0.0555**
$n_c = 65$, gaussian, geodesic	24.3005	**5.7486e+08**	**2.5908e+08**	**2.0028e+06**	**3.9628e+07**	**0.0562**
$n_c = 65$, tps, euclidean	200.2656	**5.1024e+08**	**2.3144e+08**	**1.7682e+06**	**3.8901e+07**	**0.0551**
$n_c = 65$, tps, geodesic	25.7477	**5.7610e+08**	**2.5062e+08**	**2.0643e+06**	**3.9699e+07**	**0.0562**
$n_c = 95$, gaussian, euclidean	30.4266	**6.0727e+08**	**2.0897e+08**	**2.5262e+06**	**3.9341e+07**	**0.0551**
$n_c = 95$, gaussian, geodesic	23.6331	**5.7836e+08**	**2.6196e+08**	**2.0067e+06**	**3.9486e+07**	**0.0561**
$n_c = 95$, tps, euclidean	30.3989	**5.9504e+08**	**2.0666e+08**	**2.4632e+06**	**3.9413e+07**	**0.0553**
$n_c = 95$, tps, geodesic	28.2751	**6.1400e+08**	**2.2891e+08**	**2.4424e+06**	**3.9452e+07**	**0.0556**

Table 4.5: Results using the radial basis function parameterization with different size basis and combinations of kernel and distance functions. We report the optimization time, loss value, distance value, smoothness regularizer value, diffusion tensor imaging loss, and the median standard deviation of fractional anisotropy. For all metrics, lower is better. Values in bold are better than the value for that metric for applying the $\gamma = 0$ field map to all diffusion directions. In all cases, the distance, smoothness, and standard deviation of fractional anisotropy improves using the parameterization. In almost all cases, the median value of the improves.

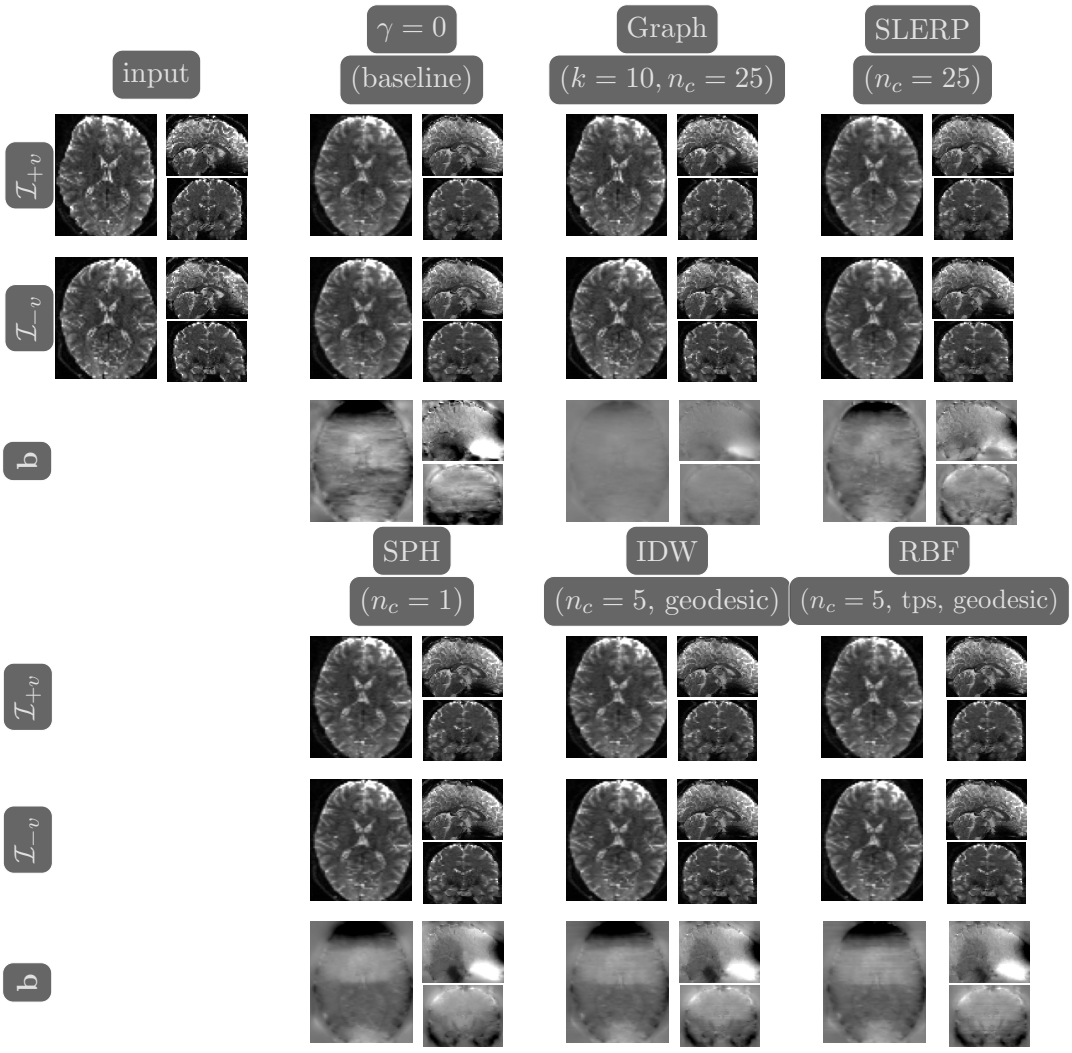


Figure 4.13: Visualization of images and field maps for one subject (Subject ID 100206) with no diffusion weighting. The first column in the top half shows the input data, the second column shows the $\gamma = 0$ baseline correction, and the remaining columns show the results using a graph parameterization with $k = 10$ and $n_c = 25$, a spherical linear interpolation parameterization using $n_c = 25$, a spherical harmonic parameterization using $n_c = 1$, an inverse distance weighting parameterization using $n_c = 5$ and geodesic distance, and a radial basis function parameterization using $n_c = 5$, geodesic distance, and thin plate spline kernel. For each optimization, the corrected images with opposite phase encoding direction and field map are shown. With the exception of the graph parameterization, the corrected images are of similar quality, and spherical harmonic, inverse distance weighting, and radial basis function parameterizations have more smooth field maps.

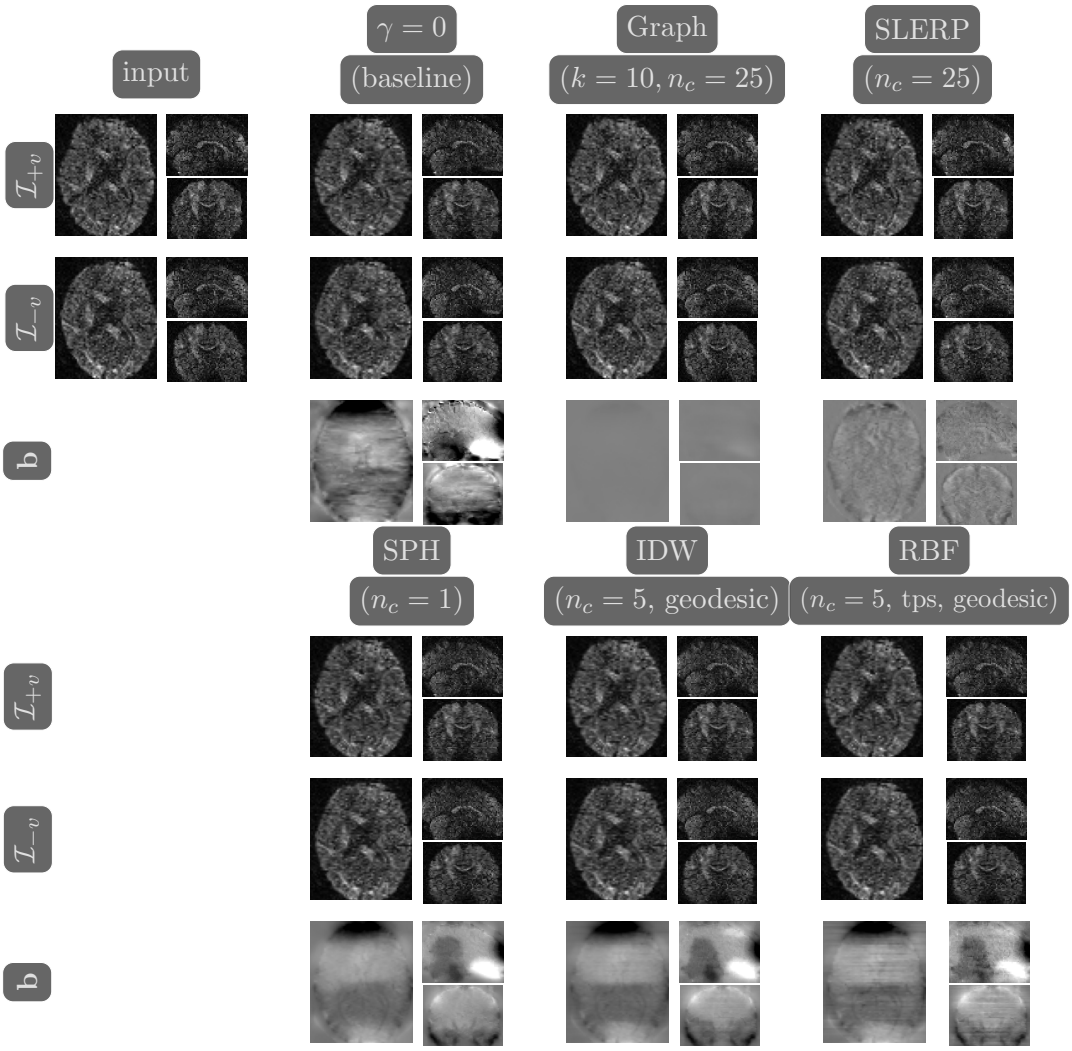


Figure 4.14: Visualization of images and field maps for one subject (Subject ID 100206) with diffusion weighting. This diffusion directions is not in the subset for optimization. The first column in the top half shows the input data, the second column shows the $\gamma = 0$ baseline correction, and the remaining columns show the results using a graph parameterization with $k = 10$ and $n_c = 25$, a spherical linear interpolation parameterization using $n_c = 25$, a spherical harmonic parameterization using $n_c = 1$, an inverse distance weighting parameterization using $n_c = 5$ and geodesic distance, and a radial basis function parameterization using $n_c = 5$, geodesic distance, and thin plate spline kernel. For each optimization, the corrected images with opposite phase encoding direction and field map are shown. The graph and SLERP parameterizations do not interpolate well. The spherical harmonic, inverse distance weighting, and radial basis function parameterizations interpolate well and have more smooth field maps.

the smoothness improves by over 99% relative to applying the $\gamma = 0$ field map, none of the 4D graph parameterizations improve distance, diffusion tensor loss, or median standard deviation of fractional anisotropy. Optimization with a graph parameterization empirically is better with a larger value for n_c, but in this setting n_c is limited by the smaller size of the subset of diffusion directions used in optimization.

Using the spherical linear interpolation, we vary the number of basis vectors between $n_c = 5, 25, 45, 65$, and 95. Table 4.7 shows the results of using the optimized coefficients from a subset of diffusion directions and full parameterization to interpolate a field map for the entire 4D volume. We build a full parameterization using the same basis vectors and the full set of diffusion directions, and interpolate a field map using the optimized coefficients. The only interpolated SLERP parameterization improving diffusion tensor loss is the parameterization using $n_c = 5$. None of the SLERP parameterizations improve distance or median standard deviation of fractional anisotropy.

For the spherical harmonic parameterization, we vary the maximum degree as $n_c = 1, 2, 3$, or 4. Table 4.8 shows the results of using the optimized coefficients for a subset of diffusion directions and full parameterization to interpolate a field map for the entire 4D volume. We build a full parameterization by computing the spherical harmonic coefficients for the full set of diffusion directions. While $n_c = 4$ does not improve any metrics, $n_c = 1, 2$, and 3 successfully interpolate to the full dataset and improve the relative distance compared to applying the $\gamma = 0$ field map by an average of 22.84%, smoothness by 79.10%, diffusion tensor loss by 2.00%, and median standard deviation of fractional anisotropy by 2.29%.

For the inverse distance weighting parameterization, we vary the number of basis vectors between $n_c = 5, 25, 45, 65$, and 95, and the distance function between euclidean and geodesic. Table 4.9 reports the results of using the coefficients optimized on a subset of directions and full parameterization to interpolate a field map. The full

parameterization uses the same basis vectors as the parameterization for optimization. The best results use $n_c = 5$. Using euclidean distance, the relative improvement, compared to applying the $\gamma = 0$ field map, is 19.34% in distance, 79.57% in smoothness, and 3.45% in diffusion tensor loss, and 2.06% in median standard deviation of fractional anisotropy. Using geodesic distance, the relative improvement is 24.77% in distance, 80.72% in smoothness, 3.75% in diffusion tensor loss, and 2.92% in median standard deviation of fractional anisotropy.

For the radial basis function parameterization, we vary the number of basis vectors between $n_c = 5, 25, 45, 65$, and 95, the distance function between euclidean and geodesic, and the kernel function between Gaussian and thin plate spline. Table 4.10 reports the results of using the coefficients optimized on a subset of directions and full parameterization to interpolate a field map. The full parameterization uses the same basis vectors as the parameterization for optimization. The best results use $n_c = 5$. Using a Gaussian kernel and euclidean distance, the relative improvement is 23.82% in distance, 81.32% in smoothness, 3.38% in diffusion tensor loss, and 2.75% in median standard deviation of fractional anisotropy. Using a Gaussian kernel and geodesic distance, the relative improvement is 25.48% in distance, 79.54% in smoothness, 3.49% in diffusion tensor loss, and 2.92% in median standard deviation of fractional anisotropy. Using a thin plate spline kernel and euclidean distance, there is a relative improvement of 20.19% in distance, 82.11% in smoothness, 2.19% in diffusion tensor loss, and 1.03% in median standard deviation of fractional anisotropy. Using a thin plate spline kernel and geodesic distance, the relative improvement is 24.03% in distance, 81.23% in smoothness, 3.09% in diffusion tensor loss, and 2.23% in median standard deviation of fractional anisotropy.

	Opt. Time (s)	Loss Value	Distance Value	Smoothness Reg. Value	DTI Loss	std FA
baseline		1.4981e+09	1.4981e+09		7.6348e+07	0.0797
$\gamma = 0$ (baseline)		9.9229e+08	5.5542e+08	2.7707e+06	3.9761e+07	0.0582
$k = 2, n_c = 5$	126.0622	1.3440e+09	1.3405e+09	**2.2459e+04**	5.3437e+07	0.0767
$k = 2, n_c = 10$	91.7613	1.3114e+09	1.3074e+09	**2.5904e+04**	5.2900e+07	0.0757
$k = 2, n_c = 15$	118.6506	1.2864e+09	1.2820e+09	**2.8232e+04**	5.2453e+07	0.0751
$k = 2, n_c = 25$	110.4325	1.2594e+09	1.2536e+09	**3.6677e+04**	5.1164e+07	0.0740
$k = 5, n_c = 5$	124.0593	1.3757e+09	1.3708e+09	**3.0642e+04**	5.4219e+07	0.0778
$k = 5, n_c = 10$	86.6808	1.3362e+09	1.3310e+09	**3.2471e+04**	5.3061e+07	0.0766
$k = 5, n_c = 15$	105.0814	1.2960e+09	1.2914e+09	**2.9124e+04**	5.2514e+07	0.0755
$k = 5, n_c = 25$	109.7916	1.2608e+09	1.2551e+09	**3.6322e+04**	5.0932e+07	0.0739
$k = 10, n_c = 5$	105.6295	1.3768e+09	1.3722e+09	**2.8919e+04**	5.4223e+07	0.0782
$k = 10, n_c = 10$	119.9717	1.3403e+09	1.3351e+09	**3.2725e+04**	5.3026e+07	0.0767
$k = 10, n_c = 15$	124.3314	1.2925e+09	1.2867e+09	**3.6338e+04**	5.2351e+07	0.0751
$k = 10, n_c = 25$	92.1436	1.2592e+09	1.2535e+09	**3.6203e+04**	5.1007e+07	0.0739

Table 4.6: Results using the field map from optimization with a graph parameterization on a subset of diffusion directions interpolated and applied to the full 4D volume. We report the optimization time on the subset, loss value, distance value, smoothness regularizer value, diffusion tensor imaging loss, and the median standard deviation of fractional anisotropy. For all metrics, lower is better. Values in bold are better than the value for that metric for applying the $\gamma = 0$ field map to all diffusion directions. While the smoothness improves, no other metric does.

	Opt. Time (s)	Loss Value	Distance Value	Smoothness Reg. Value	DTI Loss	std FA
input		1.4981e+09	1.4981e+09		7.6348e+07	0.0797
$\gamma = 0$ (baseline)		9.9229e+08	5.5542e+08	2.7707e+06	3.9761e+07	0.0582
$n_c = 5$	100.1722	**7.3644e+08**	6.7339e+08	**3.9987e+05**	**3.9676e+07**	0.0608
$n_c = 25$	112.9367	1.8842e+09	1.1407e+09	4.7153e+06	4.0517e+07	0.0617
$n_c = 45$	109.6724	1.0903e+09	8.8763e+08	**1.2856e+06**	4.3061e+07	0.0615
$n_c = 65$	113.6391	**8.1025e+08**	7.0928e+08	**6.4037e+05**	4.6613e+07	0.0619
$n_c = 95$	105.4426	**7.9403e+08**	7.0567e+08	**5.6042e+05**	4.9877e+07	0.0628

Table 4.7: Results using the field map from optimization with spherical linear interpolation parameterization on a subset of diffusion directions interpolated and applied to the full 4D volume. We report the optimization time on the subset, loss value, distance value, smoothness regularizer value, diffusion tensor imaging loss, and the median standard deviation of fractional anisotropy. For all metrics, lower is better. Values in bold are better than the value for that metric for applying the $\gamma = 0$ field map to all diffusion directions. While the smoothness improves for many of the SLERP parameterizations and the diffusion tensor loss improves for $n_c = 5$, none of the parameterizations improve distance or median standard deviation of fractional anisotropy.

	Opt. Time (s)	Loss Value	Distance Value	Smoothness Reg. Value	DTI Loss	std FA
input		1.4981e+09	1.4981e+09		7.6348e+07	0.0797
$\gamma = 0$ (baseline)		9.9229e+08	5.5542e+08	2.7707e+06	3.9761e+07	0.0582
$n_c = 1$	89.8742	**4.9172e+08**	**4.1388e+08**	**4.9365e+05**	**3.8478e+07**	**0.0567**
$n_c = 2$	95.8029	**4.9561e+08**	**4.1113e+08**	**5.3583e+05**	**3.8873e+07**	**0.0566**
$n_c = 3$	114.1452	**5.7229e+08**	**4.6072e+08**	**7.0761e+05**	**3.9542e+07**	**0.0573**
$n_c = 4$	144.7307	1.1217e+10	2.3440e+09	5.6273e+07	4.1528e+07	0.0629

Table 4.8: Results using the field map from optimization with spherical harmonic parameterization on a subset of diffusion directions interpolated and applied to the full 4D volume. We report the optimization time on the subset, loss value, distance value, smoothness regularizer value, diffusion tensor imaging loss, and the median standard deviation of fractional anisotropy. For all metrics, lower is better. Values in bold are better than the value for that metric for applying the $\gamma = 0$ field map to all diffusion directions. For $n_c = 1, 2$, and 3 the 4D parameterizations improve all of the metrics and are able to interpolate to the full dataset.

	Opt. Time (s)	Loss Value	Distance Value	Smoothness Reg. Value	DTI Loss	std FA
input		1.4981e+09	1.4981e+09		7.6348e+07	0.0797
$\gamma = 0$ (baseline)		9.9229e+08	5.5542e+08	2.7707e+06	3.9761e+07	0.0582
$n_c = 5$, euclidean	94.9214	**5.3723e+08**	**4.4798e+08**	**5.6607e+05**	**3.8390e+07**	**0.0570**
$n_c = 5$, geodesic	122.4786	**5.0207e+08**	**4.1786e+08**	**5.3411e+05**	**3.8270e+07**	**0.0565**
$n_c = 25$, euclidean	128.0795	2.7260e+13	1.8362e+11	1.7173e+11	4.7187e+07	0.0755
$n_c = 25$, geodesic	7.5889	2.0604e+10	2.3574e+09	1.1573e+08	4.2098e+07	0.0637
$n_c = 45$, euclidean	140.0086	7.2792e+11	1.4272e+10	4.5262e+09	4.1325e+07	0.0633
$n_c = 45$, geodesic	8.4888	1.1729e+10	1.6887e+09	6.3678e+07	4.1446e+07	0.0627
$n_c = 65$, euclidean	115.8576	8.3451e+10	3.9628e+09	5.0414e+08	3.9785e+07	0.0602
$n_c = 65$, geodesic	8.1930	1.1264e+10	1.6484e+09	6.0986e+07	4.1545e+07	0.0629
$n_c = 95$, euclidean	121.7975	2.1383e+09	9.0781e+08	7.8042e+06	**3.8767e+07**	**0.0580**
$n_c = 95$, geodesic	7.3624	1.1276e+10	1.6813e+09	6.0856e+07	4.1709e+07	0.0632

Table 4.9: Results using the field map from optimization with inverse distance weighting parameterization on a subset of diffusion directions interpolated and applied to the full 4D volume. We report the optimization time on the subset, loss value, distance value, smoothness regularizer value, diffusion tensor imaging loss, and the median standard deviation of fractional anisotropy. For all metrics, lower is better. Values in bold are better than the value for that metric for applying the $\gamma = 0$ field map to all diffusion directions. For $n_c = 5$, the metrics improve using the parameterization.

	Opt. Time (s)	Loss Value	Distance Value	Smoothness Reg. Value	DTI Loss	std FA
input		1.4981e+09	1.4981e+09		7.6348e+07	0.0797
$\gamma = 0$ (baseline)		9.9229e+08	5.5542e+08	2.7707e+06	3.9761e+07	0.0582
$n_c = 5$, gaussian, euclidean	121.6855	**5.0470e+08**	**4.2310e+08**	**5.1759e+05**	**3.8418e+07**	**0.0566**
$n_c = 5$, gaussian, geodesic	142.7804	**5.0325e+08**	**4.1388e+08**	**5.6679e+05**	**3.8373e+07**	**0.0565**
$n_c = 5$, tps, euclidean	88.7181	**5.2146e+08**	**4.4330e+08**	**4.9569e+05**	**3.8891e+07**	**0.0576**
$n_c = 5$, tps, geodesic	94.1101	**5.0397e+08**	**4.2198e+08**	**5.1998e+05**	**3.8534e+07**	**0.0569**
$n_c = 25$, gaussian, euclidean	28.9617	2.1381e+10	2.4137e+09	1.2029e+08	4.1628e+07	0.0631
$n_c = 25$, gaussian, geodesic	7.7958	1.2351e+10	1.8186e+09	6.6798e+07	4.1991e+07	0.0635
$n_c = 25$, tps, euclidean	89.8935	3.7916e+11	1.7869e+10	2.2914e+09	4.4683e+07	0.0686
$n_c = 25$, tps, geodesic	7.5928	1.1946e+11	9.1953e+09	6.9936e+08	4.3422e+07	0.0658
$n_c = 45$, gaussian, euclidean	117.4294	1.0989e+10	1.6215e+09	5.9412e+07	4.0818e+07	0.0618
$n_c = 45$, gaussian, geodesic	8.1800	1.7457e+10	2.3571e+09	9.5770e+07	4.2544e+07	0.0643
$n_c = 45$, tps, euclidean	147.6760	1.8085e+10	1.7688e+09	1.0348e+08	4.0274e+07	0.0601
$n_c = 45$, tps, geodesic	8.1497	1.5263e+10	2.0984e+09	8.3491e+07	4.2540e+07	0.0644
$n_c = 65$, gaussian, euclidean	116.0101	9.2442e+09	1.4280e+09	4.9573e+07	4.0800e+07	0.0618
$n_c = 65$, gaussian, geodesic	8.4753	1.6590e+10	2.2865e+09	9.0720e+07	4.2433e+07	0.0640
$n_c = 65$, tps, euclidean	151.2570	3.0200e+09	6.9008e+08	1.4777e+07	**3.9439e+07**	0.0585
$n_c = 65$, tps, geodesic	88.5076	1.4939e+10	2.0876e+09	8.1507e+07	4.2631e+07	0.0644
$n_c = 95$, gaussian, euclidean	137.4680	8.3756e+09	1.3445e+09	4.4593e+07	4.0676e+07	0.0616
$n_c = 95$, gaussian, geodesic	8.8320	1.5886e+10	2.2282e+09	8.6624e+07	4.2214e+07	0.0637
$n_c = 95$, tps, euclidean	121.5010	1.8396e+09	5.6834e+08	8.0629e+06	**3.9161e+07**	**0.0577**
$n_c = 95$, tps, geodesic	8.6731	1.4574e+10	2.0696e+09	7.9306e+07	4.2459e+07	0.0641

Table 4.10: Results using the field map from optimization with radial basis function parameterization on a subset of diffusion directions interpolated and applied to the full 4D volume. We report the optimization time on the subset, loss value, distance value, smoothness regularizer value, diffusion tensor imaging loss, and the median standard deviation of fractional anisotropy. For all metrics, lower is better. Values in bold are better than the value for that metric for applying the $\gamma = 0$ field map to all diffusion directions. For $n_c = 5$, the metrics improve using the parameterization and interpolation.

4.5 Summary

In this chapter, we introduce a parameterization of a four-dimensional field map in the diffusion tensor setting. Using this parameterization and a four-dimensional correction can improve correction quality and diffusion tensor imaging metrics, compared to the current best practice of applying the field map for a non-diffusion weighted $\gamma = 0$ image to all diffusion directions. We also show the potential for using a parameterization as an interpolation of the field map, allowing for faster optimization on a subset of diffusion directions and interpolating the full size field map based on the optimized coefficients.

Based on our experiments, the most promising parameterizations are inverse distance weighting and radial basis function parameterizations. Particularly, parameterizations using geodesic distance can increase the relative improvement of diffusion tensor imaging metrics compared to the $\gamma = 0$ baseline. Optimization with these parameterizations can also converge quickly, with optimization times of around 30 seconds.

Using the parameterization to optimize coefficients using a subset of diffusion directions and interpolating the full field map shows promising results. In our experiments, inverse distance weighting and radial basis function parameterizations also perform the best. However, the improvements are most consistent with a basis size of $n_c = 5$, the smallest number used in our experiments.

Across all experiments and parameterizations, the smoothness value is the most often improved among all of the metrics. While the parameterization promotes smoothness in the diffusion dimension, the smoothness value only measures the smoothness in the three spatial dimensions of the field map. The optimized coefficients can, using the parameterization, result in a more smooth field map overall.

Our experiments in this chapter indicate that a four-dimensional optimization using a parameterization such as inverse distance weighting or radial basis function

can improve the quality of both distortion correction and diffusion tensor analysis compared to using a $\gamma = 0$ field map for correction across all diffusion dimensions. This can overcome the downsides of $\gamma = 0$ field map correction in the presence of motion and uniform contrast. The parameterization promotes smoothness in all dimensions, and can be used to decrease computation by optimizing on a subset of diffusion directions and interpolating to the full set of diffusion directions.

This framework to model a four-dimensional field map allows for further exploration of distortion correction in the DTI setting. In the next chapter, we use the four-dimensional parameterization to enable automatic tuning of hyperparameters of the distortion correction problem. Other explorations, left as future work, include analysis of parameterizations and their associated parameter choices, additional parameterization options, and analysis of which diffusion directions to choose at scan time to most improve distortion correction.

Chapter 5

Automatic Hyperparameter Tuning

In this chapter, we evaluate the influence on distortion correction of the hyperparameters in both three-dimensional (2.5) and four-dimensional (4.1) correction. Existing tools offer default parameter values or suggested configurations of parameter values, but require manual tuning of these parameters to achieve the best results. In experiments of the preceding chapters, the hyperparameter values were fixed for all examples in each experiment. In this chapter, we vary the hyperparameter values and propose a structure for the automatic tuning of hyperparameters in four-dimensional DTI using a bilevel optimization.

In 5.1, we describe the L-curve analysis used to evaluate the influence of α, the scalar weighting the smoothness regularization term in the optimization, on both distance and smoothness. In 5.2, we propose a bilevel optimization for the automatic tuning of hyperparameters in the four-dimensional setting. In 5.3, we present results of the L-curve analysis and bilevel optimization. In 5.4, we summarize the chapter.

5.1 L-curve Analysis

A classical approach to selecting an optimal hyperparameter value is through L-curve analysis [32]. For regularized problems of the form

$$\min_x ||Ax - b||^2 + \lambda R(x),$$

it is often the case that for the optimal x, larger values of λ increase the value of the data fit term, and smaller values of λ decrease the value of data fit term. In L-curve analysis, the optimization problem is solved for values of λ in the range $[\lambda_{min}, \lambda_{max}]$ and the corresponding values of the data fit term and regularization term are plotted on logarithmic axes. In some applications, the resulting curve is L-shaped, with the optimal trade-off between data fit and regularization at the bottom left corner of the 'L'.

In both the three-dimensional optimization problem (2.5) and four-dimensional optimization problem (4.1), the parameter α weighting the importance of the smoothness regularization term is balancing the smoothness term and the distance term. Using L-curve analysis can indicate the optimal choice for the smoothness parameter α. Of note, the speed of optimization allows for solving the distortion correction problem with varying α quickly; in our experiments using 100 different values for α in 3D correction takes about 10 minutes.

Because the optimization problem is non-convex, the L-curve is not likely to have a well-defined corner of the 'L' shape [32]. However, the point of the L-curve with the largest curvature can be taken as the best α value for the optimization. To ensure a smooth curve in this setting, we begin with the largest value of α and for each decreasing value, we use the result of optimization from the previous value as the initial guess for the optimization with the new value of α.

5.2 Bilevel Optimization

For four-dimensional correction, the same L-curve analysis can be used to select a good choice for the smoothness parameter α. However, the lack of a distinct corner of the 'L' makes the results of this approach inconsistent. The additional information of diffusion tensor metrics can be used to setup a bilevel optimization and improve both correction and diffusion tensor metrics through the choice of α and other hyperparameters. Let $f(\mathbf{b})$ be a diffusion tensor metric, either diffusion tensor loss (2.14) or median standard deviation of fractional anisotropy (2.15), evaluated using the field map $\mathbf{b}$. Then we have the following bilevel optimization

$$\min_{\alpha\in\mathbb{R},\alpha>0} f(\mathbf{b}) \quad \text{s.t.} \quad \mathbf{b} = (\mathbf{Q}\otimes\mathbf{I}_m)\cdot\mathbf{c}^* \tag{5.1a}$$

$$\text{and} \quad \mathbf{c}^* \in \arg\min_{\mathbf{c}} J_{\mathbf{Q}}(\mathbf{c}) = D(\mathbf{c}) + \alpha S(\mathbf{c}) + \beta P(\mathbf{c}). \tag{5.1b}$$

The inner optimization (5.1b) and computation of $\mathbf{b}$ is as in Chapter 4. Because α is not explicitly a part of the outer optimization (5.1a), computing the derivative with respect to α is difficult. To avoid this, solving the outer problem is a good candidate for derivative-free optimization such as Bayesian optimization [44]. Additionally, this setup can allow for simultaneously optimizing over a set of multiple hyperparameters such as the parameters of the Gauss Newton optimization.

In Bayesian optimization, the objective f is treated as a random function with an associated prior distribution. As the objective is evaluated, the prior is updated to form a posterior distribution, and the next value of the optimization parameter is chosen by a sampling method informed by the posterior. The algorithm we use is a Tree-structured Parzen Estimator (TPE) [7], in which the sampling chooses the hyperparameter(s) maximizing the quotient of two Gaussian Mixture Models (GMM) evaluated at the hyperparameter(s), where the numerator GMM is fit to the hyperparameter values from the best f values seen so far, and the denominator GMM is fit

to the remaining hyperparameter values. The details of the Tree-structured Parzen Estimator algorithm are in Section 2.6. In our experiments, we use the Python TPE implementation in the Optuna library [2].

5.3 Results

5.3.1 L-curve Analysis in 3D

In this section, we evaluate the L-curve analysis on a set of three-dimensional data from the Human Connectome Project [62] and demonstrate its limitations. These images are 3T strength with left-right phase encoding. In all cases, we use $\beta = 1\text{e}-4$ as the scalar parameter weighting the intensity regularization term. The values for α vary in the range $[1, 1\text{e}3]$, and we use 100 values for α in constructing the L-curve.

Results of the L-curve analysis for 7 example subjects is shown in Table 5.1. The average optimization time, solving the optimization problem (2.5) 100 times with different values of α, is 9.74 minutes. The largest value of α used for each volume results in lower smoothness, but higher distance. The smallest value of α used for each volume results in higher smoothness and lower distance. The best value of α, chosen automatically as the point of highest curvature on the L-curve, attempts to balance the distance and smoothness. Plots of distance increasing as α increases, smoothness penalty decreasing as α increases, curvature changing as α increases, and the resulting L-curve for an example subject are shown in Figure 5.1. For the same subject, the resulting images and field maps are shown in Figure 5.2 for the input, smallest α, largest α, and chosen best α. In this application, the L-curve appears more linear than ‘L‘ shaped, making choosing a corner value difficult.

	Opt. Time (s)		Input	Best	Min	Max
107422	682.5	α	1.3858e+08	28.48	1.0	1000.0
		distance		1.1806e+07	8.2603e+06	3.6597e+07
		smoothness		1.5182e+05	4.2197e+05	2.7264e+04
123117	578.21	α	7.1424e+07	53.37	1.0	1000.0
		distance		5.1776e+06	1.7871e+06	1.9908e+07
		smoothness		7.1671e+04	3.3723e+05	1.3512e+04
172534	588.93	α	1.7121e+08	7.56	1.0	1000.0
		distance		9.8199e+06	8.3788e+06	4.3057e+07
		smoothness		2.8880e+05	4.5867e+05	3.7820e+04
176744	707.44	α	9.6376e+07	13.22	1.0	1000.0
		distance		6.7261e+06	5.0160e+06	2.8979e+07
		smoothness		1.8021e+05	3.9075e+05	1.9920e+04
192843	592.52	α	8.4872e+07	403.7	1.0	1000.0
		distance		1.4071e+07	1.9852e+06	2.1555e+07
		smoothness		2.8105e+04	3.0693e+05	1.6586e+04
211619	477.63	α	1.1660e+08	2.01	1.0	1000.0
		distance		2.0217e+06	1.8103e+06	2.4817e+07
		smoothness		3.4624e+05	4.3202e+05	2.1250e+04
333330	462.3	α	8.7475e+07	2.66	1.0	1000.0
		distance		2.0790e+06	1.8549e+06	2.1780e+07
		smoothness		3.1542e+05	3.9170e+05	1.9189e+04

Table 5.1: Results of using L-curve analysis to choose the best α for three-dimensional optimization. For 7 example subjects, we report the optimization time, and the α value, distance term value, and smoothness term value for the input data and results using the best, minimum, and maximum α values.

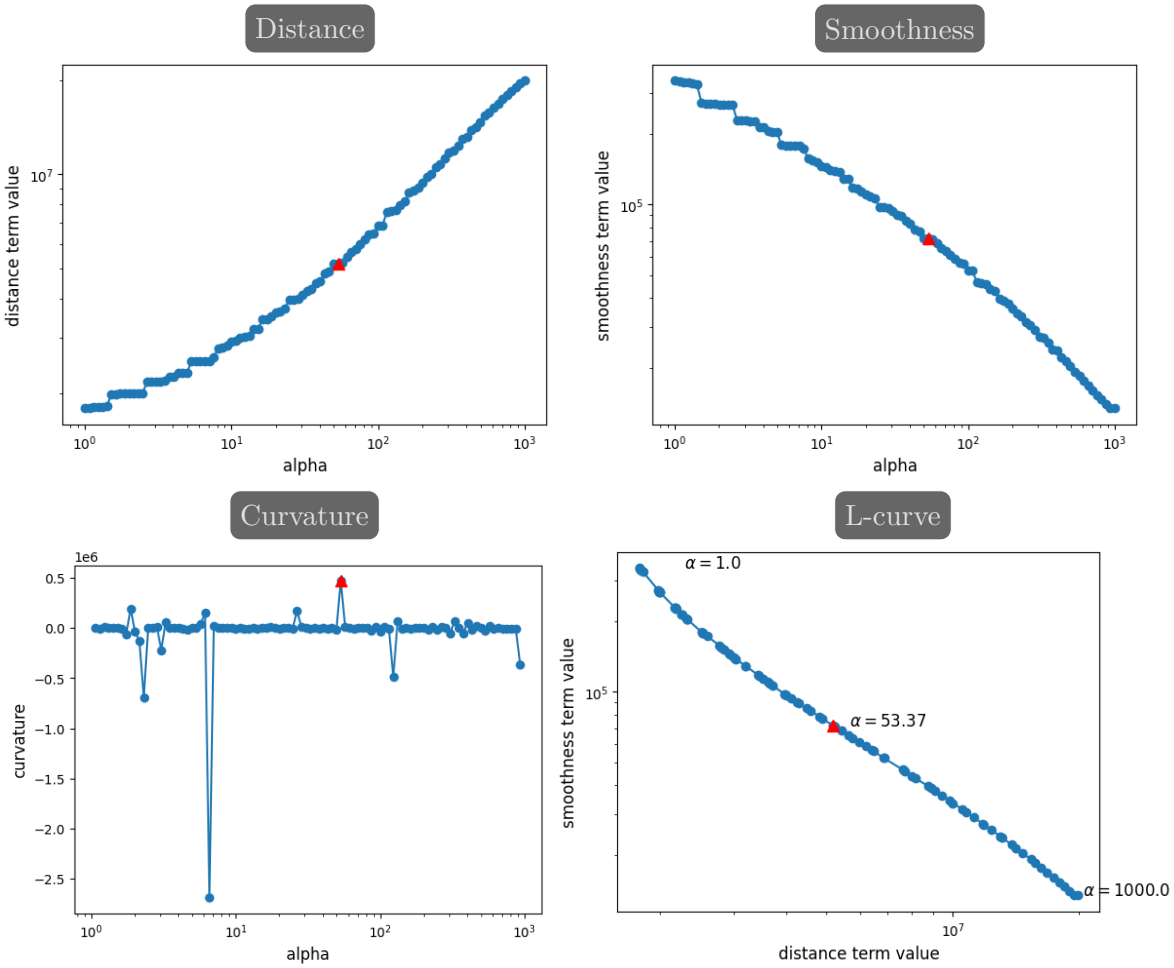


Figure 5.1: Results relating distance and smoothness for an example subject (Subject ID 123117). Top left shows how the distance value increases as α increases. Top right shows how the smoothness penalty decreases as α increases. Bottom left shows the curvature at each α. Bottom right shows the resulting L-curve. The best α, chosen as the point of highest curvature, is indicated with the red triangle in each plot.

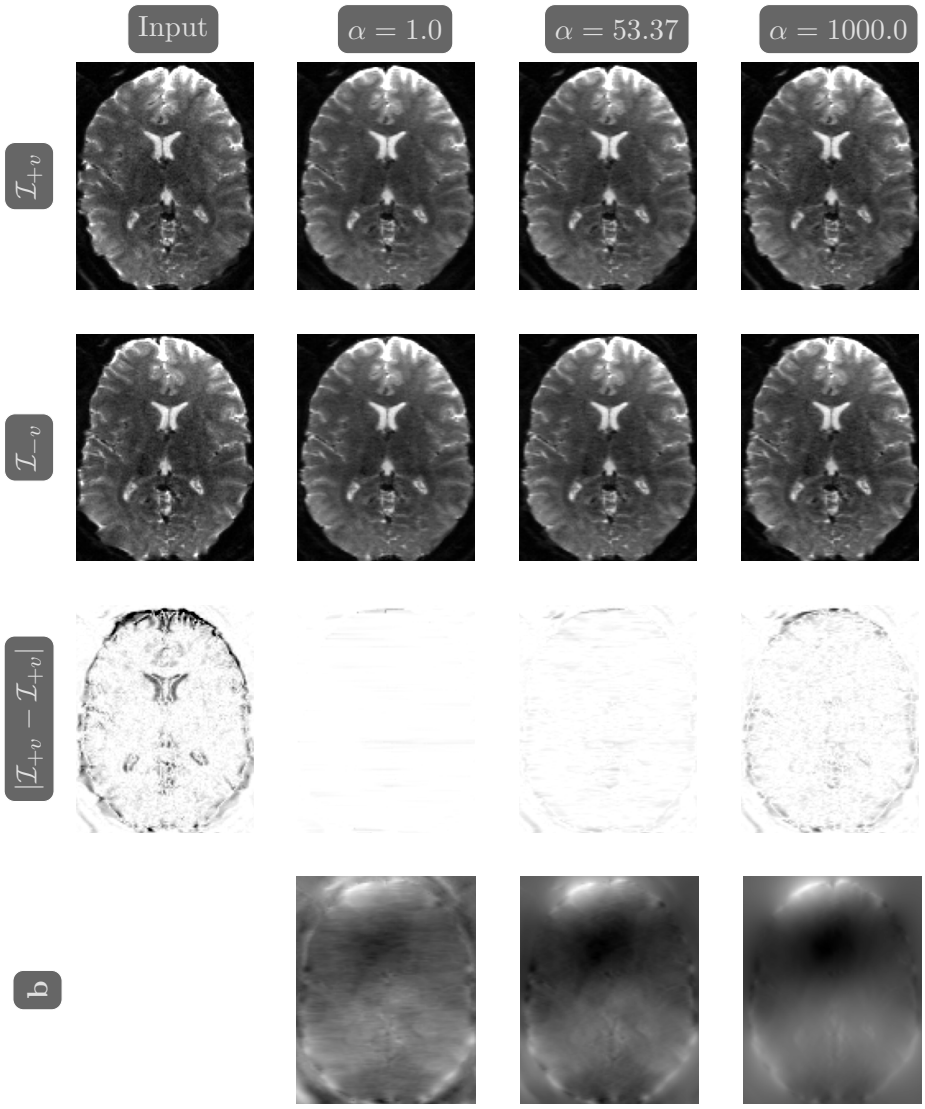


Figure 5.2: Resulting images and field maps using L-curve analysis for an example subject (Subject ID 123117). The top two rows show the pair of images with opposite phase encoding direction, the third row shows the (color inverted) absolute difference between the images, and the bottom row shows the field map. The first column is the pair of input images, the second column is the minimum value of α, the third column is the best value of α, and the last column is the maximum value of α.

5.3.2 Bilevel Optimization in 4D

In this section, we demonstrate the bilevel optimization described in 5.2. The data used in these experiments are 6 four-dimensional diffusion-weighted images from the Human Connectome Project [62]. These images are 3T strength, with 95 diffusion directions and left-right phase encoding. In all cases, we use as $\mathbf{Q}$ a radial basis function parameterization with $n_c = 65$ basis vectors, a thin plate spline kernel, and geodesic distance. In the optimization, the number of outer iterations is 20. The smoothness parameter α varies within the range $[1\text{e}-1, 1\text{e}3]$, the intensity modulation constraint parameter β varies within the range $[1\text{e}-6, 1\text{e}1]$, the maximum number of Gauss Newton iterations varies within the range $[3, 100]$, the stopping criteria tolerance for Gauss Newton varies within the range $[1\text{e}-6, 1\text{e}-2]$, the maximum number of PCG iterations varies in the range $[1, 50]$, and the stopping tolerance for PCG varies in the range $[1\text{e}-2, 1]$.

Figure 5.3 shows resulting L-curves from bilevel optimization only tuning the value of α. The best value of α balances the lowest distance and smoothness values.

In Table 5.2, we report the results of bilevel optimization over all hyperparameters using median standard deviation of fractional anisotropy as the outer objective. Over the 6 four-dimensional volumes, the average optimization time for the bilevel optimization is 9.33 minutes. Compared to the baseline using default hyperparameter values as in Chapter 4, the bilevel optimization results in lower distance, diffusion tensor loss, and median standard deviation of fractional anisotropy.

In Table 5.3, we report the results of bilevel optimization using diffusion tensor loss as the outer objective. Over the 6 four-dimensional volumes, the average optimization time for the bilevel optimization is 8.51 minutes. These results are similar to those using median standard deviation of fractional anisotropy as the metric. Compared to the baseline using default hyperparameter values as in Chapter 4, the bilevel optimization results in lower distance, diffusion tensor loss, and median standard

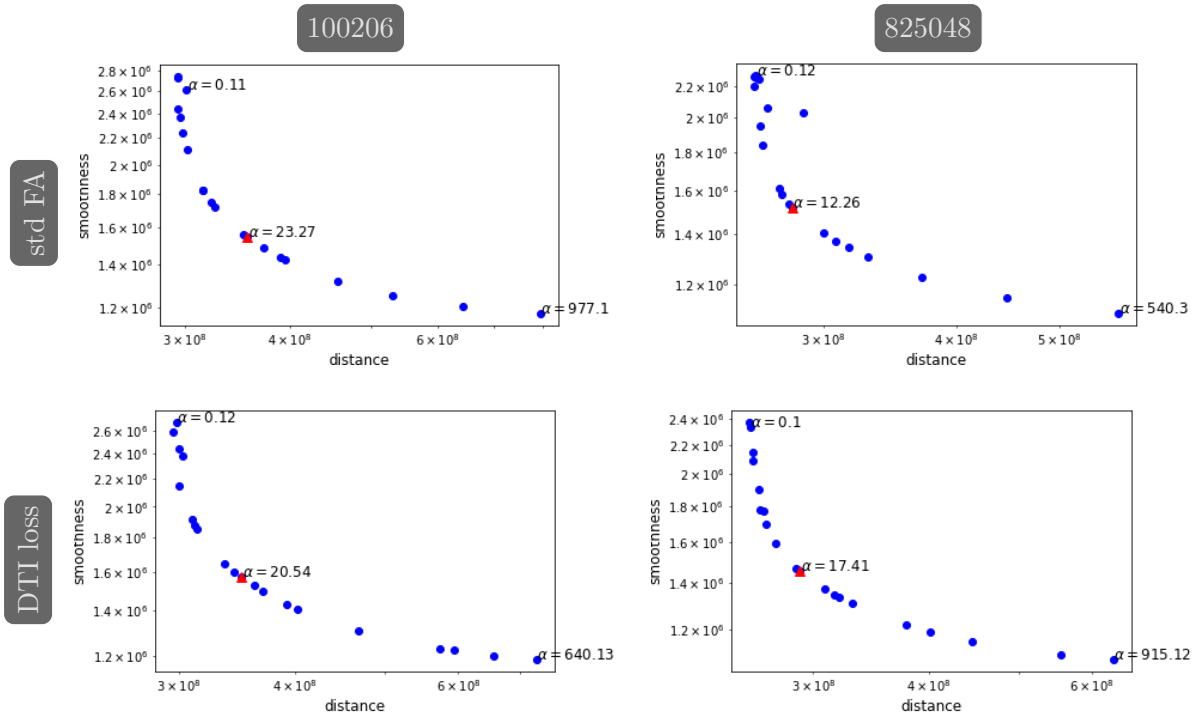


Figure 5.3: The L-curves resulting from bilevel optimization tuning α for two example subjects, Subject ID 100206 (first column) and Subject ID 825048 (second column). The best α chosen in bilevel optimization is indicated by the red triangle. The top row uses median standard deviation of fractional anisotropy as the objective, and the bottom row uses diffusion tensor loss as the objective.

	Opt. Time (s)		Input	Baseline	Best
100206	474.6030	α		10.0000	0.6739
		β		1.0000e-04	2.0348e-03
		GN iterations		500	22
		GN tolerance		1.0000e-05	1.1043e-06
		PCG iterations		20	10
		GN tolerance		1.0000e-01	1.0653e-02
		distance	1.7395e+09	2.7386e+08	2.4353e+08
		smoothness		2.1410e+06	3.0325e+06
		DTI loss	8.9731e+07	4.1730e+07	3.8634e+07
		std FA	7.8084e-02	4.8563e-02	4.4194e-02
120414	481.9603	α		10.0000	0.2360
		β		1.0000e-04	6.7991e-01
		GN iterations		500	30
		GN tolerance		1.0000e-05	2.9998e-05
		PCG iterations		20	9
		GN tolerance		1.0000e-01	1.8818e-02
		distance	1.6304e+09	3.3369e+08	3.0311e+08
		smoothness		2.3272e+06	4.0240e+06
		DTI loss	3.6817e+07	3.6262e+07	3.3641e+07
		std FA	7.5952e-02	6.5179e-02	6.0434e-02
456346	613.2917	α		10.0000	0.1193
		β		1.0000e-04	1.3793e+00
		GN iterations		500	70
		GN tolerance		1.0000e-05	1.6034e-04
		PCG iterations		20	41
		GN tolerance		1.0000e-01	2.2377e-02
		distance	1.4136e+09	2.3173e+08	1.9558e+08
		smoothness		2.1536e+06	4.6290e+06
		DTI loss	9.0470e+07	4.0569e+07	3.7559e+07
		std FA	8.5311e-02	5.7807e-02	5.2334e-02
531940	638.6675	α		10.0000	0.1018
		β		1.0000e-04	1.4414e-02
		GN iterations		500	4
		GN tolerance		1.0000e-05	2.6473e-03
		PCG iterations		20	50
		GN tolerance		1.0000e-01	2.7153e-02
		distance	1.2104e+09	2.3074e+08	1.9634e+08
		smoothness		2.1080e+06	4.7404e+06
		DTI loss	5.7857e+07	3.8570e+07	3.6657e+07
		std FA	7.3497e-02	5.6388e-02	5.2634e-02
825048	516.7515	α		10.0000	0.1063
		β		1.0000e-04	6.8372e-04
		GN iterations		500	65
		GN tolerance		1.0000e-05	1.8828e-04
		PCG iterations		20	25
		GN tolerance		1.0000e-01	1.0556e-02
		distance	1.3860e+09	2.4091e+08	2.1006e+08
		smoothness		1.9000e+06	3.6520e+06
		DTI loss	8.0125e+07	4.0713e+07	3.6826e+07
		std FA	8.2772e-02	5.9019e-02	5.1862e-02
972566	632.7956	α		10.0000	0.1026
		β		1.0000e-04	8.8572e+00
		GN iterations		500	100
		GN tolerance		1.0000e-05	2.5574e-04
		PCG iterations		20	38
		GN tolerance		1.0000e-01	2.0786e-02
		distance	1.2774e+09	2.2226e+08	1.9544e+08
		smoothness		1.7716e+06	3.2740e+06
		DTI loss	7.9139e+07	3.8971e+07	3.6428e+07
		std FA	7.9778e-02	5.6820e-02	5.1582e-02

Table 5.2: Results of using bilevel optimization to choose the best hyperparameters minimizing median standard deviation of fractional anisotropy. For 6 example subjects, we report the optimization time, and the hyperparameters, distance term value, smoothness term value, diffusion tensor loss value, and median standard deviation of fractional anisotropy for the input data, baseline using default values, and the best hyperparameters from bilevel optimization.

deviation of fractional anisotropy.

5.4 Summary

In this chapter, we enable automatic tuning of the hyperparameters in four-dimensional DTI distortion correction. Because the optimization problem is fast to solve, distortion correction while automatically tuning the values of the hyperparameters can be done in less than 10 minutes. The additional metrics provided by diffusion tensor analysis allow for a bilevel optimization using the diffusion metric as the outer objective and the correction as the inner objective. In our experiments, the automatic hyperparameter tuning yields superior results in terms of distance, diffusion tensor loss, and median standard deviation of fractional anisotropy when compared to the baseline using fixed default values.

This bilevel optimization provides a black box correction of DTI distortions, meaning the user does not need to manually tune any hyperparameters to achieve quality results. Additionally, we are directly optimizing the metrics that are ultimately of importance to the user, the metrics of DTI.

The bilevel optimization is a framework that will allow further exploration of distortion correction in DTI, including the influence of various hyperparameters and the relationship between distortion correction and various diffusion tensor metrics. The experiments here are limited to data from the Human Connectome Project [62], but comparing the hyperparameters optimized for data from different scan configurations could provide more insight. Additionally, the outer optimization could be improved with better heuristics on the bounds of the search space or a different choice of derivative-free optimization. The modularity of the bilevel optimization allows for changing the outer metric to suit the downstream task and application of the DTI analysis. These explorations are left as future work.

	Opt. Time (s)		Input	Baseline	Best
100206	556.2285	α		10.0000	0.1555
		β		1.0000e-04	3.4197e+00
		GN iterations		500	23
		GN tolerance		1.0000e-05	2.9182e-04
		PCG iterations		20	19
		GN tolerance		1.0000e-01	4.7504e-02
		distance	1.7395e+09	2.7386e+08	2.3883e+08
		smoothness		2.1410e+06	3.6359e+06
		DTI loss	8.9731e+07	4.1730e+07	3.9161e+07
		std FA	7.8084e-02	4.8563e-02	4.4839e-02
120414	417.9174	α		10.0000	0.1222
		β		1.0000e-04	7.0743e+00
		GN iterations		500	44
		GN tolerance		1.0000e-05	2.4091e-06
		PCG iterations		20	12
		GN tolerance		1.0000e-01	8.8208e-02
		distance	1.6304e+09	3.3369e+08	3.0145e+08
		smoothness		2.3272e+06	4.3049e+06
		DTI loss	3.6817e+07	3.6262e+07	3.3483e+07
		std FA	7.5952e-02	6.5179e-02	5.9944e-02
456346	492.6395	α		10.0000	0.1409
		β		1.0000e-04	1.2362e-03
		GN iterations		500	7
		GN tolerance		1.0000e-05	2.3661e-05
		PCG iterations		20	8
		GN tolerance		1.0000e-01	2.6851e-02
		distance	1.4136e+09	2.3173e+08	2.0018e+08
		smoothness		2.1536e+06	4.5383e+06
		DTI loss	9.0470e+07	4.0569e+07	3.7214e+07
		std FA	8.5311e-02	5.7807e-02	5.2146e-02
531940	515.4201	α		10.0000	0.3123
		β		1.0000e-04	1.1627e+00
		GN iterations		500	83
		GN tolerance		1.0000e-05	1.2876e-03
		PCG iterations		20	40
		GN tolerance		1.0000e-01	1.4505e-01
		distance	1.2104e+09	2.3074e+08	2.0467e+08
		smoothness		2.1080e+06	3.6088e+06
		DTI loss	5.7857e+07	3.8570e+07	3.6284e+07
		std FA	7.3497e-02	5.6388e-02	5.2935e-02
825048	548.3280	α		10.0000	0.1278
		β		1.0000e-04	8.2633e+00
		GN iterations		500	47
		GN tolerance		1.0000e-05	3.5358e-05
		PCG iterations		20	50
		GN tolerance		1.0000e-01	5.1081e-02
		distance	1.3860e+09	2.4091e+08	2.1326e+08
		smoothness		1.9000e+06	3.1940e+06
		DTI loss	8.0125e+07	4.0713e+07	3.6474e+07
		std FA	8.2772e-02	5.9019e-02	5.2040e-02
972566	534.6187	α		10.0000	0.1151
		β		1.0000e-04	6.0215e+00
		GN iterations		500	7
		GN tolerance		1.0000e-05	1.4136e-05
		PCG iterations		20	50
		GN tolerance		1.0000e-01	1.1849e-01
		distance	1.2774e+09	2.2226e+08	2.0177e+08
		smoothness		1.7716e+06	3.2708e+06
		DTI loss	7.9139e+07	3.8971e+07	3.5994e+07
		std FA	7.9778e-02	5.6820e-02	5.1379e-02

Table 5.3: Results of using bilevel optimization to choose the best hyperparameters minimizing diffusion tensor loss. For 6 example subjects, we report the optimization time, and the hyperparameters, distance term value, smoothness term value, diffusion tensor loss value, and median standard deviation of fractional anisotropy for the input data, baseline using default values, and the best hyperparameters from bilevel optimization.

Chapter 6

Conclusion

This thesis develops a dependable physics-based distortion correction with automatic hyperparameter tuning in the setting of Diffusion Tensor Imaging. The fast acquisition of EPI-MRI is promising for increased usage in both clinical and research applications, but the distortions in the resulting images and slow distortion correction pipelines limit the popularity of EPI-MRI. The contributions of this thesis seek to improve the speed, quality, and ease-of-use of EPI-MRI distortion correction in DTI.

In Chapter 3, we offer an improved distortion correction software tool for three-dimensional images. By leveraging the physics of the distortions and the separable structure of the optimization problem, we parallelize many of the computations and demonstrate efficient optimization on both CPU and GPU architectures. We also implement an initialization scheme based on the one-dimensional nature of the distortions. The resulting software tool, called PyHySCO, is easy to use, offering a simple command line interface and publicly available code.

In Chapter 4, we consider distortion correction in the four-dimensional setting of Diffusion Tensor Imaging. The computational advantages of the separable optimization in 3D extend to the 4D setting. We additionally introduce a parameterization of the field map that improves optimization by introducing smoothness in the diffu-

sion dimension and leveraging the additional information offered by the structure of the applied diffusion gradients. We demonstrate the ability of this parameterization to improve distortion correction and DTI metrics compared to the current practice of applying the three-dimensional field map from a non-diffusion weighted volume to correct all diffusion directions. We additionally demonstrate how the parameterization can be used to reduce computation and optimize over a subset of diffusion directions, and then interpolate the full field map.

In Chapter 5, we make optimization easier for the user by enabling automatic hyperparameter tuning. Specifically, we define a bilevel optimization in 4D to tune the hyperparameters of the correction problem and optimization scheme. The bilevel optimization uses Bayesian optimization to choose the hyperparameter values minimizing a metric of Diffusion Tensor Imaging. We demonstrate the effectiveness of the bilevel optimization in improving metrics of both distortion correction and DTI. This provides an easy-to-use, dependable correction that simultaneously optimizes for the quality of DTI analysis without requiring the user to manually tune any hyperparameters.

The contributions of this thesis offer opportunities for further research utilizing EPI-MRI for data acquisition. Additionally, there is potential for further exploration in the four-dimensional setting utilizing different parameterizations and seeking to better understand how the spherical nature of the diffusion directions relate the resulting optimal field map. Finally, we hope that this thesis and the publicly available code will further research into the computational aspects of DTI and EPI-MRI postprocessing, including applications such as functional-MRI, anatomies other than the human brain, and other postprocessing steps such as motion correction.

Bibliography

[1] Onur Afacan, Judy A Estroff, Edward Yang, Carol E Barnewolt, Susan A Connolly, Richard B Parad, Robert V Mulkern, Simon K Warfield, and Ali Gholipour. Fetal echoplanar imaging: promises and challenges. *Topics in Magnetic Resonance Imaging*, 28(5):245–254, 2019.

[2] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[3] Abdallah Zaid Alkilani, Tolga Çukur, and Emine Ulku Saritas. Fd-net: An unsupervised deep forward-distortion model for susceptibility artifact correction in epi. *arXiv preprint arXiv:2303.10436*, 2023.

[4] Jesper L R Andersson, Stefan Skare, and John Ashburner. How to correct susceptibility distortions in spin-echo echo-planar images: application to diffusion tensor imaging. *NeuroImage*, 20(2):870–888, October 2003.

[5] Jesper L.R. Andersson, Mark S. Graham, Ivana Drobnjak, Hui Zhang, and Jon Campbell. Susceptibility-induced distortion that varies due to motion: Correction in diffusion mr without acquiring additional data. *NeuroImage*, 171: 277–295, 2018. ISSN 1053-8119. doi: https://doi.org/10.1016/j.neuroimage.

2017.12.040. URL https://www.sciencedirect.com/science/article/pii/S1053811917310601.

[6] Vegard Antun, Francesco Renna, Clarice Poon, Ben Adcock, and Anders C. Hansen. On instabilities of deep learning in image reconstruction and the potential costs of AI. *Proceedings of the National Academy of Sciences*, 117(48): 30088–30095, 2020. ISSN 0027-8424. doi: 10.1073/pnas.1907377117.

[7] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. *Advances in neural information processing systems*, 24, 2011.

[8] James Bergstra, Daniel Yamins, and David Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *International conference on machine learning*, pages 115–123. PMLR, 2013.

[9] Miguel A. Blanco, M. Flórez, and M. Bermejo. Evaluation of the rotation matrices in the basis of real spherical harmonics. *Journal of Molecular Structure: THEOCHEM*, 419(1):19–27, 1997. ISSN 0166-1280. doi: https://doi.org/10.1016/S0166-1280(97)00185-1. URL https://www.sciencedirect.com/science/article/pii/S0166128097001851.

[10] Paul T Boggs and Jon W Tolle. Sequential quadratic programming. *Acta numerica*, 4:1–51, 1995.

[11] R Bowtell, DJO McIntyre, MJ Commandre, PM Glover, and P Mansfield. Correction of geometric distortion in echo planar images. In *Soc. Magn. Res. Abstr*, volume 2, page 411, 1994.

[12] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, Jonathan Eckstein, et al. Distributed optimization and statistical learning via the alternating direction

method of multipliers. *Foundations and Trends® in Machine learning*, 3(1): 1–122, 2011.

[13] Martin Dietrich Buhmann. Radial basis functions. *Acta numerica*, 9:1–38, 2000.

[14] Leon Y Cai, Qi Yang, Colin B Hansen, Vishwesh Nath, Karthik Ramadass, Graham W Johnson, Benjamin N Conrad, Brian D Boyd, John P Begnoche, Lori L Beason-Held, et al. Prequal: An automated pipeline for integrated preprocessing and quality assurance of diffusion weighted mri images. *Magnetic resonance in medicine*, 86(1):456–470, 2021.

[15] H Chang and J M Fitzpatrick. A Technique for Accurate Magnetic-Resonance-Imaging in the Presence of Field Inhomogeneities. *Medical Imaging, IEEE Transactions on*, 11(3):319–329, September 1992.

[16] Zhaolin Chen, Kamlesh Pawar, Mevan Ekanayake, Cameron Pain, Shenjun Zhong, and Gary F Egan. Deep learning for image enhancement and correction in magnetic resonance imaging—state-of-the-art and challenges. *Journal of Digital Imaging*, pages 1–27, 2022.

[17] Daan Christiaens, Paddy J Slator, Lucilio Cordero-Grande, Anthony N Price, Maria Deprez, Daniel C Alexander, Mary Rutherford, Joseph V Hajnal, and Jana Hutter. In utero diffusion mri: challenges, advances, and applications. *Topics in Magnetic Resonance Imaging*, 28(5):255–264, 2019.

[18] F.R.K. Chung. *Spectral Graph Theory*. Number no. 92 in CBMS Regional Conference Series. Conference Board of the Mathematical Sciences, 1994. ISBN 9780821889367. URL `https://books.google.com/books?id=YUc38_MCuhAC`.

[19] Ian A. Clark, Martina F. Callaghan, Nikolaus Weiskopf, Eleanor A. Maguire, and Siawoosh Mohammadi. Reducing susceptibility distortion related image

blurring in diffusion mri epi data. *Frontiers in Neuroscience*, 15, 2021. URL https://api.semanticscholar.org/CorpusID:235382378.

[20] James W Cooley, Peter AW Lewis, and Peter D Welch. The fast fourier transform and its applications. *IEEE Transactions on Education*, 12(1):27–34, 1969.

[21] Gergely Dávid, Björn Fricke, Jan Malte Oeschger, Lars Ruthotto, Francisco J. Fritz, Ora Ohana, Thomas Sauvigny, Patrick Freund, Karsten Tabelow, and Siawoosh Mohammadi. Acid: A comprehensive toolbox for image processing and modeling of brain, spinal cord, and ex vivo diffusion mri data. *bioRxiv*, 2024. URL https://api.semanticscholar.org/CorpusID:264307573.

[22] Stephan Dempe and Alain Zemkoho. Bilevel optimization. In *Springer optimization and its applications*, volume 161. Springer, 2020.

[23] Soan T. M. Duong, Son Lam Phung, Abdesselam Bouzerdoum, Sui Paul Ang, and Mark M. Schira. Correcting susceptibility artifacts of mri sensors in brain scanning: A 3d anatomy-guided deep learning approach. *Sensors*, 21(7), 2021.

[24] Soan TM Duong, Son L Phung, Abdesselam Bouzerdoum, and Mark M Schira. An unsupervised deep learning technique for susceptibility artifact correction in reversed phase-encoding epi images. *Magnetic Resonance Imaging*, 71:1–10, 2020.

[25] STM Duong, Son Lam Phung, Abdesselam Bouzerdoum, HG Boyd Taylor, AM Puckett, and Mark M Schira. Susceptibility artifact correction for sub-millimeter fmri using inverse phase encoding registration and t1 weighted regularization. *Journal of Neuroscience Methods*, 336:108625, 2020.

[26] Oscar Esteban, Alessandro Daducci, Emmanuel Caruyer, Kieran O'Brien, María J Ledesma-Carbayo, Meritxell Bach-Cuadra, and Andrés Santos.

Simulation-based evaluation of susceptibility distortion correction methods in diffusion mri for connectivity analysis. In *2014 IEEE 11th International Symposium on Biomedical Imaging (ISBI)*, pages 738–741. IEEE, 2014.

[27] W Freeden and W Törnig. On spherical spline interpolation and approximation. *Mathematical Methods in the Applied Sciences*, 3(1):551–575, 1981.

[28] Mark S Graham, Ivana Drobnjak, Mark Jenkinson, and Hui Zhang. Quantitative assessment of the susceptibility artefact and its interaction with motion in diffusion mri. *PloS one*, 12(10):e0185647, 2017.

[29] Xuan Gu and Anders Eklund. Evaluation of six phase encoding based susceptibility distortion correction methods for diffusion mri. *Frontiers in neuroinformatics*, 13:76, 2019.

[30] Walter A Hall and Charles L Truwit. Intraoperative mr-guided neurosurgery. *Journal of Magnetic Resonance Imaging: An Official Journal of the International Society for Magnetic Resonance in Medicine*, 27(2):368–375, 2008.

[31] P.C. Hansen, J.G. Nagy, and D.P. O'Leary. *Deblurring Images: Matrices, Spectra, and Filtering*. Fundamentals of Algorithms. Society for Industrial and Applied Mathematics, 2006. ISBN 9780898716184. URL `https://books.google.com/books?id=JjbSoRR9T-0C`.

[32] Per Christian Hansen. The l-curve and its use in the numerical treatment of inverse problems. In *Computational Inverse Problems in Electrocardiology*. WIT Press, 1999.

[33] Magnus Rudolph Hestenes and Eduard Stiefel. Methods of Conjugate Gradients for Solving Linear Systems. *Journal of Research of the National Bureau of Standards*, 49(6):409 436, 1952.

[34] Dominic Holland, Joshua M Kuperman, and Anders M Dale. Efficient correction of inhomogeneous static magnetic field-induced distortion in Echo Planar Imaging. *NeuroImage*, 50(1):175–183, March 2010.

[35] Zhangxuan Hu, Yishi Wang, Zhe Zhang, Jieying Zhang, Huimao Zhang, Chunjie Guo, Yuejiao Sun, and Hua Guo. Distortion correction of single-shot epi enabled by deep-learning. *NeuroImage*, 221:117–170, 2020.

[36] M Okan Irfanoglu, Pooja Modi, Amritha Nayak, Elizabeth B Hutchinson, Joelle Sarlls, and Carlo Pierpaoli. Dr-buddi (diffeomorphic registration for blip-up blip-down diffusion imaging) method for correcting echo planar imaging distortions. *Neuroimage*, 106:284–299, 2015.

[37] Abigail Julian and Lars Ruthotto. Pyhysco: Gpu-enabled susceptibility artifact distortion correction in seconds. *Frontiers in Nueroscience*, 18, 2024.

[38] W.T. Kelvin, P.G. Tait, and G.H. Darwin. *Treatise on Natural Philosophy*. Number v. 1, no. 1 in Treatise on Natural Philosophy. At the University Press, 1879. URL `https://books.google.com/books?id=D9dJAAAAMAAJ`.

[39] Denis Le Bihan, Jean-François Mangin, Cyril Poupon, Chris A Clark, Sabina Pappata, Nicolas Molko, and Hughes Chabriat. Diffusion tensor imaging: concepts and applications. *Journal of Magnetic Resonance Imaging: An Official Journal of the International Society for Magnetic Resonance in Medicine*, 13(4): 534–546, 2001.

[40] Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1):503–528, 1989.

[41] S. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982. doi: 10.1109/TIT.1982.1056489.

[42] Jan Macdonald and Lars Ruthotto. Improved Susceptibility Artifact Correction of Echo Planar MRI using the Alternating Direction Method of Multipliers. *Journal of Mathematical Imaging and Vision*, 60(2):268–282, 2017.

[43] C Malamateniou, SJ Malik, SJ Counsell, JM Allsop, AK McGuinness, T Hayat, Kathryn Broadhouse, RG Nunes, AM Ederies, JV Hajnal, et al. Motion-compensation techniques in neonatal and fetal mr imaging. *American Journal of Neuroradiology*, 34(6):1124–1136, 2013.

[44] Jonas Mockus. *Bayesian Approach to Global Optimization: Theory and Applications*. Springer Dordrecht, 1989. URL `https://api.semanticscholar.org/CorpusID:120322743`.

[45] Jan Modersitzki. *FAIR: flexible algorithms for image registration*, volume 6 of *Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2009. ISBN 978-0-898716-90-0. doi: 10.1137/1.9780898718843.

[46] Jorge Nocedal and Stephen J Wright. *Numerical optimization*. Springer, 1999.

[47] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL `https://proceedings.neurips.cc/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf`.

[48] William D Penny, Karl J Friston, John T Ashburner, Stefan J Kiebel, and Thomas E Nichols. *Statistical parametric mapping: the analysis of functional brain images*. Elsevier, 2007.

[49] Gabriel Peyré, Marco Cuturi, et al. Computational optimal transport. *Center for Research in Economics and Statistics Working Papers*, 2017-86, 2017.

[50] Constantin Roder, Patrick Haas, Marcos Tatagiba, Ulrike Ernemann, and Benjamin Bender. Technical limitations and pitfalls of diffusion-weighted imaging in intraoperative high-field mri. *Neurosurgical Review*, 44:327–334, 2021.

[51] Lars Ruthotto, H Kugel, J Olesch, B Fischer, J Modersitzki, M Burger, and C H Wolters. Diffeomorphic susceptibility artifact correction of diffusion-weighted magnetic resonance images. *Physics in Medicine and Biology*, 57(18):5715–5731, September 2012.

[52] Lars Ruthotto, Siawoosh Mohammadi, Constantin Heck, Jan Modersitzki, and Nikolaus Weiskopf. Hyperelastic Susceptibility Artifact Correction of DTI in SPM. In *Bildverarbeitung fuer die Medizin*, pages 344–349, Berlin, Heidelberg, 2013. Springer, Berlin, Heidelberg.

[53] Yousef Saad. *Iterative Methods for Sparse Linear Systems*. SIAM. SIAM, 04 2003. ISBN 0898715342.

[54] Donald Shepard. A two-dimensional interpolation function for irregularly-spaced data. In *Proceedings of the 1968 23rd ACM National Conference*, ACM '68, page 517–524, New York, NY, USA, 1968. Association for Computing Machinery. ISBN 9781450374866. doi: 10.1145/800186.810616. URL `https://doi.org/10.1145/800186.810616`.

[55] Ken Shoemake. Animating rotation with quaternion curves. *SIGGRAPH Com-*

put. Graph., 19(3):245–254, jul 1985. ISSN 0097-8930. doi: 10.1145/325165.325242. URL https://doi.org/10.1145/325165.325242.

[56] Stephen M Smith, Mark Jenkinson, Mark W Woolrich, Christian F Beckmann, Timothy EJ Behrens, Heidi Johansen-Berg, Peter R Bannister, Marilena De Luca, Ivana Drobnjak, David E Flitney, et al. Advances in functional and structural mr image analysis and implementation as fsl. *Neuroimage*, 23:S208–S219, 2004.

[57] Haykel Snoussi, Julien Cohen-Adad, Olivier Commowick, Benoit Combes, Elise Bannier, Soizic Leguy, Anne Kerbrat, Christian Barillot, and Emmanuel Caruyer. Evaluation of distortion correction methods in diffusion mri of the spinal cord, 2021.

[58] Michael K. Stehling, Robert Turner, and Peter Mansfield. Echo-planar imaging: Magnetic resonance imaging in a fraction of a second. *Science*, 254(5028):43–50, 1991. doi: 10.1126/science.1925560. URL https://www.science.org/doi/abs/10.1126/science.1925560.

[59] Edward O Stejskal and John E Tanner. Spin diffusion measurements: spin echoes in the presence of a time-dependent field gradient. *The journal of chemical physics*, 42(1):288–292, 1965.

[60] Chantal MW Tax, Matteo Bastiani, Jelle Veraart, Eleftherios Garyfallidis, and M Okan Irfanoglu. What's new and what's next in diffusion mri preprocessing. *NeuroImage*, 249:118830, 2022.

[61] Antonio Tristán-Vega, Santiago Aja-Fernández, and Carl-Fredrik Westin. Least squares for diffusion tensor estimation revisited: Propagation of uncertainty with rician and non-rician signals. *NeuroImage*, 59(4):4032–4043, 2012. ISSN 1053-

8119. doi: https://doi.org/10.1016/j.neuroimage.2011.09.074. URL https://www.sciencedirect.com/science/article/pii/S1053811911011499.

[62] David C Van Essen, Kamil Ugurbil, Edward Auerbach, Deanna Barch, Timothy EJ Behrens, Richard Bucholz, Acer Chang, Liyong Chen, Maurizio Corbetta, Sandra W Curtiss, et al. The human connectome project: a data acquisition perspective. *Neuroimage*, 62(4):2222–2231, 2012.

[63] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[64] Shuhei Watanabe. Tree-structured parzen estimator: Understanding its algorithm components and their roles for better empirical performance, 2023.

[65] Minjie Wu, Lin-Ching Chang, Lindsay Walker, Herve Lemaitre, Alan S Barnett, Stefano Marenco, and Carlo Pierpaoli. Comparison of epi distortion correction methods in diffusion tensor mri using a novel framework. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 321–329. Springer, 2008.

[66] Joseph Yuan-Mou Yang, Jian Chen, Bonnie Alexander, Kurt Schilling, Michael Kean, Alison Wray, Marc Seal, Wirginia Maixner, and Richard Beare. Assessment of intraoperative diffusion epi distortion and its impact on estimation of supratentorial white matter tract positions in pediatric epilepsy surgery. *NeuroImage: Clinical*, 35:103097, 2022.

[67] Benjamin Zahneisen, Kathrin Baeumler, Greg Zaharchuk, Dominik Fleischmann, and Michael Zeineh. Deep flow-net for epi distortion estimation. *Neuroimage*, 217:116886, 2020.