

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:

James Lincoln Herring

Date

Efficient Solvers for Nonlinear Problems in Imaging

By

James Lincoln Herring

Doctor of Philosophy

Applied Mathematics

Lars Ruthotto

Advisor

James Nagy

Committee Member

Jun Kong

Committee Member

Accepted:

Lisa A. Tedesco, Ph.D.

Dean of the James T. Laney School of Graduate Studies

Date

Efficient Solvers for Nonlinear Problems in Imaging

By

James Lincoln Herring

B.S., Emory University, 2010

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 Applied Mathematics

2018

Abstract

Efficient Solvers for Nonlinear Problems in Imaging

By James Lincoln Herring

Nonlinear inverse problems arise in numerous imaging applications, and solving them is often difficult due to ill-posedness and high computational cost. In this work, we introduce tailored solvers for several nonlinear inverse problems in imaging within a Gauss–Newton optimization framework.

We develop a linearize and project (LAP) method for a class of nonlinear problems with two (or more) sets of coupled variables. At each iteration of the Gauss–Newton optimization, LAP linearizes the residual around the current iterate, eliminates one block of variables via a projection, and solves the resulting reduced dimensional problem for the Gauss–Newton step. The method is best suited for problems where the subproblem associated with one set of variables is comparatively well-posed or easy to solve. LAP supports iterative, direct, and hybrid regularization and supports element-wise bound constraints on all the blocks of variables. This offers various options for incorporating prior knowledge of a desired solution. We demonstrate the advantages of these characteristics with several numerical experiments. We test LAP for two and three dimensional problems in super-resolution and MRI motion correction, two separable nonlinear least-squares problems that are linear in one block of variables and nonlinear in the other. We also use LAP for image registration subject to local rigidity constraints, a problem that is nonlinear in all sets of variables. These two classes of problems demonstrate the utility and flexibility of the LAP method.

We also implement an efficient Gauss–Newton optimization scheme for the problem of phase recovery in bispectral imaging, a univariate nonlinear inverse problem. Using a fixed approximate Hessian, matrix-reordering, and stored matrix factors, we accelerate the Gauss–Newton step solve, resulting in a second-order optimization

method which outperforms first-order methods in terms of cost per iteration and solution quality.

Efficient Solvers for Nonlinear Problems in Imaging

By

James Lincoln Herring

B.S., Emory University, 2010

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 Applied Mathematics

2018

Acknowledgements

I want to thank the many people who have supported me during my five years of doctoral studies.

First and foremost, thank you to my wife, Soojeong Herring, who left the comforts of home to come to the U.S.A. and take a chance on marrying me. Her love, encouragement, and sacrifice spurs me on. I also want to include a special note to our dog, Gouda. His tail wags and smiling face make every day brighter.

Thank you also to my family: Paul, Kate, Rachel, and Lehman. They have been a constant source of strength, love, and security for my whole life. They also set the best example a youngest family member could ever hope to have. A special note of thanks also goes to Aunt Kristin and Uncle Steve who have taken care of me in Atlanta for more years than they bargained for.

Thank you to my many teachers at Emory, especially my advisors Lars Ruthotto and Jim Nagy. They have invested many hours in making me a better scholar and person. I am grateful for their sage advice and friendship.

Lastly, thank you to the Lord who works all things for good and supplies the peace that passes all understanding. I lean on those promises daily.

Contents

| | | |
|----------|--|-----------|
| 1 | Introduction | 1 |
| 2 | Background | 6 |
| 2.1 | Nonlinear Inverse Problems | 8 |
| 2.2 | Gauss–Newton Method | 9 |
| 2.2.1 | Gauss–Newton Step | 10 |
| 2.2.2 | Armijo Line Search | 14 |
| 2.2.3 | Stopping Criteria | 16 |
| 2.2.4 | Gauss–Newton Algorithm | 17 |
| 2.2.5 | Convergence | 18 |
| 2.3 | Projected Gauss–Newton | 20 |
| 2.4 | Regularization | 25 |
| 2.4.1 | Regularization for Nonlinear Problems | 26 |
| 2.4.2 | Regularization for the Gauss–Newton Step | 29 |
| 2.5 | Imaging | 30 |
| 3 | Linearize and Project Method for Coupled Problems | 33 |
| 3.1 | Coupled Nonlinear Inverse Problems | 34 |
| 3.2 | LAP Method | 35 |
| 3.2.1 | LAP with Projected Gauss–Newton | 39 |
| 3.3 | Alternatives Methods for Coupled Problems | 42 |

| | | |
|----------|---|------------|
| 3.3.1 | Block Coordinate Descent for Coupled Problems | 42 |
| 3.3.2 | Variable Projection for Separable Least-squares | 43 |
| 4 | Motion Correction Problems | 47 |
| 4.1 | Motion Correction Imaging Problem | 48 |
| 4.2 | Super-Resolution | 51 |
| 4.2.1 | Two Dimensional Super-Resolution | 52 |
| 4.2.2 | Three Dimensional Super-Resolution | 57 |
| 4.3 | Motion Correction for Magnetic Resonance Imaging (MRI) | 59 |
| 4.4 | Discussion | 67 |
| 5 | Locally Rigid Image Registration | 69 |
| 5.1 | Locally Rigid Registration Problem | 70 |
| 5.1.1 | Eulerian and Lagrangian Frameworks | 72 |
| 5.1.2 | Hyperelastic Regularizer | 77 |
| 5.1.3 | Numerical Implementation | 79 |
| 5.2 | Numerical Results | 90 |
| 5.2.1 | 2D Knee Registration | 90 |
| 5.2.2 | 3D Block Registration | 95 |
| 5.2.3 | Discussion | 99 |
| 6 | Phase Recovery from the Bispectrum | 103 |
| 6.1 | Bispectral Imaging Problem | 104 |
| 6.1.1 | Calculating the Object Power Spectrum | 105 |
| 6.1.2 | Accumulating the Object Bispectrum | 108 |
| 6.1.3 | Phase Recovery Problem | 111 |
| 6.2 | Numerical Results | 117 |

| | | |
|----------|--|------------|
| 7 | Conclusions and Future Work | 128 |
| | Appendices | 131 |
| A.1 | Linear Interpolation in Higher Dimensions | 131 |
| A.1.1 | Bilinear Interpolation | 131 |
| A.1.2 | Trilinear Interpolation | 133 |
| A.1.3 | Bilinear and Trilinear Interpolation for Multiple Points | 135 |
| A.2 | Rigid Transformations | 138 |
| A.2.1 | Rigid Transformations in 2D | 139 |
| A.2.2 | Rigid Transformations in 3D | 140 |
| | Bibliography | 143 |

Chapter 1

Introduction

Models are used to describe many real world phenomena across a broad range of disciplines, and data collected for a given phenomenon can be viewed as transformation of some ground truth by an appropriate model. Inconveniently, the data collected about this ground truth is almost always obscured by the presence of noise and inaccuracy arising in the data collection process. This presents a challenge for scientists and mathematicians: how to best recover a close approximation to the ground truth given a set of inexact data and some idea of the mathematical model that relates the two. Problems of this kind are widely studied and are known as inverse problems. Often, the model relating the ground truth and collected data is nonlinear resulting in nonlinear inverse problems. Efficient solutions to problems of this form are necessary in a number of fields including medicine, engineering, research science, technology, and many others.

One common approach to solving nonlinear inverse problems is numerical optimization. In this dissertation, we develop tailored solvers within a Gauss–Newton optimization framework for solving nonlinear inverse problems with a focus on applications in medical and astronomical imaging. The contributions of this dissertation can be summarized into two categories, both under the larger umbrella of nonlinear

inverse problems. The first of these is the Linearize And Project (LAP) method for nonlinear inverse problems with two or more sets of nontrivially coupled variables representing different physical quantities. The method is applicable to a broad range of coupled inverse problems, and we demonstrate its utility through numerical experiments on several problems arising in the field of imaging including super-resolution, motion correction for magnetic resonance imaging, and locally rigid image registration. The second category develops tailored solvers for a specific astronomical imaging application: the univariate nonlinear inverse problem of phase recovery in bispectral imaging.

We summarize the contributions of LAP as follows:

- We propose an efficient iterative method called LAP for coupled nonlinear problems that computes the Gauss–Newton step at each iteration by eliminating one block of variables through projection and solving the reduced problem iteratively. Since projection is performed after linearization, any block can be eliminated. This strategy makes the LAP framework more flexible than existing projection-based approaches, e.g., by supporting various types of regularization strategies and the option to impose constraints for all variables.
- We test LAP for separable least-squares problems, a specific class of nonlinear coupled problems. Here, the problem is linear in one set of variables and nonlinear in the other. Our test problems are 2D and 3D motion correction problems in imaging: super-resolution (Sect. 4.2) and magnetic resonance imaging (Sect 4.3). We compare LAP with variable projection (VarPro) and block coordinate descent (BCD), two common approaches for this type of problem. We show that LAP outperforms these methods for these problems in terms of solution quality, CPU time, and computational cost.
- We show LAP is applicable to general nonlinear coupled problems by applying

it to the problem of 2D and 3D image registration subject to rigid motion constraints on some portion of the image domain (Sect. 5.1). This coupled problem is nonlinear in both sets of variables. LAP results in better optimization behavior for the problem than solving for the Gauss–Newton step using a fully coupled, unprojected approach. This is most observable for 3D examples, where the fully coupled strategy can result in poor registration results.

- We demonstrate LAP’s flexibility for different regularization strategies. For separable least-squares examples, this includes Tikhonov regularization using the gradient operator with a fixed regularization parameter and a hybrid regularization approach proposed by [9] that simultaneously computes the search direction and selects an appropriate regularization parameter at each iteration. For the locally rigid image registration problem, we couple LAP with a nonlinear, hyperelastic regularizer from [5] that enforces invertibility of the solution transform while modeling highly nonlinear deformations.
- We use projected Gauss–Newton to implement element-wise lower and upper bound constraints with LAP on the optimization variables for the 2D and 3D super-resolution problems. These constraints allow us to introduce prior knowledge of a solution into our optimization, a desirable property when solving ill-posed problems. This is a distinct advantage over VarPro, where previously proposed methods resort to approximate gradients for incorporating inequality constraints on some blocks of variables.

For the problem of phase recovery in bispectral imaging, we make the following contributions:

- We develop tailored solvers for the Gauss–Newton step for the univariate nonlinear problem of phase recovery in bispectral imaging (Sect. 6.1.3). This problem is poly-convex due to a modulus in the objective function. We implement an

efficient solver for the step in the Gauss–Newton optimization for four existing objective functions proposed for solving the phase recovery problem. Using approximations to the Hessians, matrix reordering, and incomplete factorization, we increase the efficiency of Gauss–Newton optimization schemes for this problem.

- Numerical experiments show our methods produce superior solution images in less time than commonly used first-order optimization schemes from the literature for solving the phase recovery problem. We compare our scheme with gradient descent and nonlinear conjugate gradient (NLCG) for example problems generated using simulated speckle data. We show our tailored Gauss–Newton solver is faster than its competitors in terms of time per iteration and iterations to convergence, and that the resulting solution images are comparable or better.

We present the work in the following order. Ch. 2 presents the background material for the dissertation. It introduces the following: the nonlinear forward model and its associated inverse problem; Gauss–Newton optimization for nonlinear inverse problems and the extension to projected Gauss–Newton; the ill-posedness of many nonlinear inverse problems and need for regularization; and the mathematical description of images and their transformation. Ch. 3 extends the general nonlinear forward model to coupled nonlinear problems and presents the linearize and project (LAP) method for coupled nonlinear problems within the Gauss–Newton framework. It also includes a brief introduction of variable projection (VarPro) and block coordinate descent (BCD), two methods for solving coupled nonlinear least-squares problems, a specific sub-class of coupled nonlinear problems. Ch. 4 shows the results of numerical experiments comparing LAP with VarPro and BCD for coupled nonlinear least-squares problems in super-resolution and motion correction for magnetic resonance imaging. Ch. 5 tests LAP for the fully nonlinear coupled problem of image

registration subject to local rigidity constraints. Results are compared with registration using a fully coupled solver for the Gauss–Newton step. Ch. 6 introduces the univariate nonlinear problem of phase recovery in bispectral imaging and presents a tailored Gauss–Newton approach that we apply to four different optimization formulations for solving the problem. Lastly, we conclude in Ch. 7 with some closing remarks and a view to future work.

Chapter 2

Background

In this chapter, we introduce the mathematical framework necessary for the methods and problems discussed in the subsequent chapters. We begin by presenting the general nonlinear forward model of interest with a brief discussion of the numerous applications in which problems of this type arise. We then introduce the nonlinear inverse problem corresponding to the forward model. Next, we discuss our numerical optimization strategy for solving the nonlinear inverse problems of interest. We present the Gauss–Newton method, a common second-order method for solving nonlinear inverse problems. This discussion includes our choice of line search, the convergence properties of the method for exact and inexact Gauss–Newton, and a discussion of stopping criteria. We then extend the method to projected Gauss–Newton, which allows for the imposition of element-wise bound constraints on the variables in the resulting solution. After ending the discussion about optimization, we introduce the idea of regularization as a means to treat the ill-posedness of our problems of interest and include a broad discussion of the various types of regularization. Lastly, we define the mathematical framework for images that is necessary to pose and implement the imaging examples included in this work.

Before beginning, we make some notes about notation necessary for the presenta-

tion of the material in this and future chapters.

Vectors and matrices are referenced using bold font, e.g. \mathbf{x} denotes a vector and \mathbf{A} a matrix. The lone exception is \mathcal{F} , which is used to denote Fourier transform or block Fourier transform matrices in Chapters 4 and 6. Specific entries of vectors and matrices are denoted by subindices outside of brackets, e.g. $[\mathbf{x}]_j$ denotes the j th entry of the vector \mathbf{x} . Subindices without brackets indicate index in a list or summation, e.g. \mathbf{A}_j is the j th term in the list $\mathbf{A}_1, \dots, \mathbf{A}_N$. Iteration indices will be denoted by a superscripts in parentheses, e.g. $\mathbf{x}^{(k)}$ denotes the vector \mathbf{x} at the k th iteration. Vector superscripts without parentheses are used to indicate dimensions for coordinate pairs or triples stored in vector notation, e.g. for vectors \mathbf{x}^1 and \mathbf{x}^2 , we reference the j th coordinate by the pair $([\mathbf{x}^1]_j, [\mathbf{x}^2]_j)$. No exponential powers are used for matrices in this work.

For continuous variables, we used standard math font, e.g. the function $\Phi(\mathbf{x})$ of the vector \mathbf{x} or an image $T(x) : \mathbb{R}^d \rightarrow \mathbb{R}$ of a coordinate x . We also use standard math font for single variables, points in space, or coordinate pairs, e.g. a regularization parameter α , the point $x \in \mathbb{R}^d$, or coordinate pair (x, y) . When used in each of these ways, context is provided for the given symbol.

Cursive math font is used for sets, function spaces, and functionals, e.g. $\mathbf{x} \in \mathcal{C}_x$ for a vector in the set \mathcal{C}_x , $y \in \mathcal{V}$ for the function in the function space \mathcal{V} , and $\mathcal{D}(y; T, R)$ for a distance functional in terms of a transformation y and continuous image functions T and R . Again, the exception to this rule is \mathcal{F} , which is used for Fourier transform matrices in Ch. 4 and 6. As with standard math font, context is provided when a symbol is introduced.

Lastly, we use $\|\cdot\|$ to indicate the Euclidean 2-norm and omit the subscript in practice, e.g. $\|\mathbf{x}\|$ is the 2-norm of the vector \mathbf{x} . For the few cases when another norm is used, a subscript will be included for clarity, e.g. the infinity norm $\|\cdot\|_\infty$.

2.1 Nonlinear Inverse Problems

We are interested in nonlinear problems with the discrete forward model

$$\mathbf{d} = \mathbf{F}(\mathbf{x}) + \boldsymbol{\eta}, \quad (2.1)$$

where $\mathbf{d} \in \mathbb{F}^m$ is observed data, $\mathbf{F}(\cdot) : \mathbb{F}^n \rightarrow \mathbb{F}^m$ is a nonlinear forward operator subject to $\mathbf{x} \in \mathbb{F}^n$, and $\boldsymbol{\eta} \in \mathbb{F}^m$ is identically distributed Gaussian white noise. \mathbb{F} is a field, either \mathbb{R} or \mathbb{C} depending on the problem. Problems of this type are ubiquitous across the sciences and arise in applications for modeling, statistical regression, inverse problems, etc. [48, 50].

Given the forward model above, we are interested in solving the following nonlinear inverse problem: given a set of noisy data \mathbf{d} and a known forward operator $\mathbf{F}(\cdot)$, recover \mathbf{x} which is unknown. This problem can be formulated mathematically as the nonlinear least-squares optimization problem

$$\min_{\mathbf{x}} \left\{ \Phi(\mathbf{x}) = \frac{1}{2} \|\mathbf{r}(\mathbf{x})\|^2 \right\}. \quad (2.2)$$

Here, $\Phi(\mathbf{x})$ is the objective function we aim to minimize, and $\mathbf{r}(\mathbf{x}) = \mathbf{d} - \mathbf{F}(\mathbf{x}) \in \mathbb{F}^m$ is the residual measuring the data misfit of a given solution \mathbf{x} . The goal then is to find the solution \mathbf{x}^* which minimizes the 2-norm of the residual. We note that other fit to data terms are than the 2-norm are used in other applications, but least-squares is the most common and the focus of our work.

In many applications, inverse problems such as (2.2) are ill-posed. That is, the problems do not satisfy at least one of the criteria in Hadamard's definition of well-posedness [15, 31]:

1. For all possible data realizations, a solution exists
2. For all possible data realizations, the solution is unique

3. The solution depends continuously on the data

Our problems of interest violate one or more of these criteria. The first two are concerned with the existence of a solution. The third criteria is particularly relevant in light of the presence of error in the forward model. If the solution to the problem does not depend continuously on the data, there is no guarantee that a minimizer \mathbf{x}^* to (2.2) for a set of noisy data is close to the true solution to the noise-free problem, \mathbf{x}_{true} . For many applications (particularly imaging), the exact minimizer to a noise-affected problem provides a poor-quality solution. Another concern is non-convexity of the objective function $\Phi(\cdot)$. This means that even if a global minimizer \mathbf{x}^* to (2.2) exists and is a good solution, optimization methods may converge to local minima instead. All of these issues require additional consideration in the setup and solving of our nonlinear problems of interest. Common solutions include the introduction of regularization and constraints. These techniques treat the issues above by incorporating prior knowledge of a desired solution into a problem. This restricts the space where we search for a solution and helps mitigate some of the negative effects of ill-posedness. We discuss both of these options later in Sect. 2.3 and Sect. 2.4. We begin now by introducing the Gauss–Newton method as our optimization approach to solving (2.2).

2.2 Gauss–Newton Method

Optimization problems of the form (2.2) are well studied, and summaries of the numerous methods for solving them can be found in optimization textbooks [2, 50]. All these methods are iterative, starting from an initial guess $\mathbf{x}^{(0)}$ and generating a sequence of iterates $\mathbf{x}^{(0)}, \mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots$ that converges to a solution under certain conditions. At each iterate $\mathbf{x}^{(k)}$, the methods generate the next iterate by adding a step update $\gamma^{(k)}\mathbf{p}^{(k)}$. Here, the parameter $\gamma^{(k)} > 0$ determines the step length and

$\mathbf{p}^{(k)}$ the step direction. The subsequent $(k + 1)$ th iterate is then given by

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \gamma^{(k)}\mathbf{p}^{(k)} \quad \text{for } k = 0, 1, 2, \dots$$

Choices for the step direction and length are based on the derivatives of the objective function $\Phi(\mathbf{x}^{(k)})$ at the current iterate, and both are chosen with the goal of making some sufficient descent towards minimizing the objective function. Different methods for solving (2.2) are characterized by the way they use the derivative information to choose both the direction and length. For some methods, step direction and length are computed simultaneously, e.g. trust-region methods [50]. For others, the direction and step are considered separately, e.g. gradient descent, quasi-Newton methods, Newton’s method, and coordinate descent methods [50]. In these cases, the step direction is calculated and followed by a separate line search problem to determine the optimal step length in the chosen direction. It is beyond the scope of this dissertation to cover the myriad of options available to choose step direction and length. Well known methods for nonlinear problems include the Gauss–Newton method, Levenberg–Marquardt method, Newton and quasi-Newton schemes, trust region methods, etc. For background on the most common methods, we refer the reader to [2, 50].

For the problems in this dissertation, we limit ourselves to Gauss–Newton optimization using a backtracking Armijo linesearch, which we now introduce. We begin with a discussion of the Gauss–Newton step direction, followed by a discussion of the line search.

2.2.1 Gauss–Newton Step

The Gauss–Newton step at the iterate $\mathbf{x}^{(k)}$ of (2.2) requires computing the first and second derivatives of the objective function $\Phi(\mathbf{x}^{(k)})$. Expressions for these derivatives

rely on the Jacobian of the residual $\mathbf{r}^{(k)} = \mathbf{r}(\mathbf{x}^{(k)})$ defined as

$$\mathbf{J}(\mathbf{x}^{(k)}) = \left[\frac{\partial [\mathbf{r}^{(k)}]_i}{\partial [\mathbf{x}^{(k)}]_j} \right]_{ij} \quad \text{for } i = 1, 2, \dots, m \quad \text{and} \quad j = 1, 2, \dots, n.$$

For simplicity of notation, we will refer to this matrix as $\mathbf{J}^{(k)} = \mathbf{J}(\mathbf{x}^{(k)})$ in most cases, but the reader should remain aware of its dependence on the current solution $\mathbf{x}^{(k)}$ at each iterate. For many problems, the entries of this Jacobian matrix are explicitly computed and stored at each iteration. However, for some large-scale problems this may prove infeasible, and the Jacobian and its transpose are available only implicitly as function calls returning their matrix-vector product with a given input vector. Our work presents problems for cases when the Jacobian is available both explicitly and implicitly. Using the above expression for the Jacobian, the first and second derivatives of $\Phi(\mathbf{x}^{(k)})$ are

$$\begin{aligned} \nabla \Phi(\mathbf{x}^{(k)}) &= \mathbf{J}^{(k)\top} \mathbf{r}^{(k)} \\ \nabla^2 \Phi(\mathbf{x}^{(k)}) &= \mathbf{J}^{(k)\top} \mathbf{J}^{(k)} + \sum_{j=1}^m [\mathbf{r}^{(k)}]_j \nabla^2 [\mathbf{r}^{(k)}]_j. \end{aligned} \tag{2.3}$$

Note that computing the Jacobian allows us to compute the gradient $\nabla \Phi(\mathbf{x}^{(k)})$ via a matrix-vector product. Additionally, we know the first term of $\nabla^2 \Phi(\mathbf{x}^{(k)})$ simply by computing the Jacobian. This information is available whether $\mathbf{J}^{(k)}$ is available explicitly or implicitly via a function call to a matrix-vector product. The derivatives above provide the necessary information to compute the step $\mathbf{p}^{(k)}$ for Newton's method for nonlinear least-squares, given by solving

$$\nabla^2 \Phi(\mathbf{x}^{(k)}) \mathbf{p} = -\nabla \Phi(\mathbf{x}^{(k)}).$$

This is equivalent to choosing the update step by exactly optimizing a second-order approximation of the objective function at the current iterate. Newton's method offers

local superlinear convergence under certain conditions, but also presents a number of issues. Firstly, the summation term in the second derivative requires computing the Hessian $\nabla^2[\mathbf{r}^{(k)}]_j$ for each entry of the residual, which is often computationally burdensome. Additionally for some iterates, the Newton step may not result in a reduction of the objective function, i.e., $\mathbf{p} = -(\nabla^2\Phi(\mathbf{x}^{(k)}))^{-1}\nabla\Phi(\mathbf{x}^{(k)})$ is not a descent direction. This occurs if $\nabla^2\Phi(\mathbf{x}^{(k)})$ is not positive definite, which can happen if $\mathbf{J}^{(k)}$ is not full rank or if the spectrum of the summation term $\sum_{j=1}^m[\mathbf{r}^{(k)}]_j\nabla^2[\mathbf{r}^{(k)}]_j$ forces the real part of the spectrum of $\nabla^2\Phi(\mathbf{x}^{(k)})$ to be less than or equal to zero.

Gauss–Newton offers an alternative to the Newton’s method step update. The method uses the Jacobian term of the second derivative in (2.3) as an approximation for the Hessian,

$$\mathbf{H}^{(k)} = \mathbf{J}^{(k)\top}\mathbf{J}^{(k)} \approx \nabla^2\Phi(\mathbf{x}^{(k)}),$$

and omits the second summation term in $\nabla^2\Phi(\mathbf{x}^{(k)})$. The Gauss–Newton optimization step $\mathbf{p}^{(k)}$ is then the solution of the linear system

$$\mathbf{H}^{(k)}\mathbf{p}^{(k)} = -\nabla\Phi(\mathbf{x}^{(k)}). \quad (2.4)$$

We frequently write this as the normal equations,

$$\mathbf{J}^{(k)\top}\mathbf{J}^{(k)}\mathbf{p}^{(k)} = -\mathbf{J}^{(k)\top}\mathbf{r}^{(k)}. \quad (2.5)$$

This corresponds to a linear approximation of the residual (or a quadratic approximation to the objective function) and is reasonable in situations when the Jacobian term contains most of the significant information about the Hessian. Indeed, this is true when the residual $[\mathbf{r}^{(k)}]_j$ is small or nearly linear, i.e., $\nabla^2[\mathbf{r}^{(k)}]_j$ is small. In this case, the Gauss–Newton can be shown to have similar convergence properties to Newton’s method [50]. The Gauss–Newton approximation to the Hessian also has the benefit

that the resulting approximate Hessian is symmetric positive definite provided $\mathbf{J}^{(k)}$ is full rank. Positive definite-ness of the Hessian guarantees that the resulting step update results in a reduction of the objective function. This is straightforward to see noting that if $\mathbf{J}^{(k)\top}\mathbf{J}^{(k)}$ is symmetric positive definite, so is its inverse. Taking the Cholesky factorization of that inverse, $\mathbf{R}^\top\mathbf{R} = (\mathbf{J}^{(k)\top}\mathbf{J}^{(k)})^{-1}$, we then get

$$\begin{aligned} \mathbf{p}^{(k)\top}\nabla\Phi(\mathbf{x}^{(k)}) &= -\nabla\Phi(\mathbf{x}^{(k)})^\top(\mathbf{J}^{(k)\top}\mathbf{J}^{(k)})^{-1}\nabla\Phi(\mathbf{x}^{(k)}) \\ &= -\nabla\Phi(\mathbf{x}^{(k)})^\top\mathbf{R}^\top\mathbf{R}\nabla\Phi(\mathbf{x}^{(k)}) \\ &= -\|\mathbf{R}\nabla\Phi(\mathbf{x}^{(k)})\|^2 \\ &< 0. \end{aligned} \tag{2.6}$$

If the $\mathbf{J}^{(k)}$ is not full rank, the inequality above is not strict. In that case, we lose the guarantee of a descent direction, but the system can still be solved explicitly using an appropriate direct method (e.g pseudo-inverse) or an iterative solver. Lastly, we note that the system (2.5) is the normal equations corresponding to the linear least-squares problem

$$\mathbf{p}^{(k)} = \arg \min_{\mathbf{p}} \|\mathbf{J}^{(k)}\mathbf{p} + \mathbf{r}^{(k)}\|^2. \tag{2.7}$$

The two formulations are equivalent in the case when $\mathbf{J}^{(k)}$ has full rank or when solved iteratively using methods generating iterates from the same subspace. Solving this least-squares formulation avoids the need to explicitly compute the gradient $\mathbf{J}^{(k)\top}\mathbf{r}^{(k)}$ and the approximate Hessian $\mathbf{J}^{(k)\top}\mathbf{J}^{(k)}$. For problems where $\mathbf{J}^{(k)}$ is ill-posed, the least-squares formulation may also have preferable numerical properties to the normal equations approach.

Both (2.5) and (2.7) can be solved using either by direct, matrix-factorization or iterative solvers. Our work uses both approaches for different problems to develop methods for solving the Gauss–Newton step accurately and efficiently. When using an iterative solver, computing the Gauss–Newton step exactly or to a high accuracy

may be impractical or prohibitively expensive. Additionally, recall that the Gauss–Newton step problem assumes a quadratic model of the objective function. This may not be an accurate model of the nonlinear objective function at a given iterate. For these reasons, it is common to solve the step problem in (2.5) or (2.7) to a low accuracy such as 10^{-1} or 10^{-2} when using iterative methods. Such strategies are known as inexact Newton methods [50]. The choice between solving the normal equations and least-squares formulation for the Gauss–Newton step depends on the problem, and we use both formulations on a case by case basis. In some cases, the choice is dictated by the regularizer for a given problem, which determines whether the problem fits conveniently into the least-squares formulation. For others, computational considerations motivate the choice.

2.2.2 Armijo Line Search

Having established in (2.6) that the Gauss–Newton step is a descent direction, we turn our attention to determining step length parameter $\gamma^{(k)} > 0$ in that direction at each iterate. The ideal choice of $\gamma^{(k)}$ for a given iterate $\mathbf{x}^{(k)}$ and step $\mathbf{p}^{(k)}$ is the global minimizer of the problem

$$\min_{\gamma > 0} \{ \Psi(\gamma) = \Phi(\mathbf{x}^{(k)} + \gamma \mathbf{p}^{(k)}) \}.$$

Often, finding this global minimizer is prohibitively expensive. In practice, we compromise by settling for a $\gamma^{(k)}$ parameter that sufficiently decreases the objective function $\Phi(\cdot)$ while incurring minimal cost. One popular condition for defining “sufficient decrease” is the Armijo condition, given by the inequality

$$\Phi(\mathbf{x}^{(k)} + \gamma^{(k)} \mathbf{p}^{(k)}) \leq \Phi(\mathbf{x}^{(k)}) + c_1 \gamma^{(k)} \nabla \Phi(\mathbf{x}^{(k)})^\top \mathbf{p}^{(k)} \quad (2.8)$$

for some chosen constant $c_1 \in (0, 1)$. For implementation, we set $c_1 = 10^{-4}$ per the

recommendation in [50]. In many cases, a further condition is introduced to prevent the step $\gamma^{(k)}$ from being too small. One such condition is the curvature condition that requires the step $\gamma^{(k)}$ to satisfy the following inequality:

$$\nabla\Phi(\mathbf{x}^{(k)} + \gamma^{(k)}\mathbf{p}^{(k)})^\top \mathbf{p}^{(k)} \geq c_2 \nabla\Phi(\mathbf{x}^{(k)})^\top \mathbf{p}^{(k)}, \quad (2.9)$$

for some constant $c_2 \in (c_1, 1)$. This inequality prevents the selection of a $\gamma^{(k)}$ parameter for which the derivative $\Phi'(\mathbf{x}^{(k)} + \gamma\mathbf{p}^{(k)})$ with respect to γ is too negative. This is sensible because if this value is very negative, the objective function could be further minimized by choosing γ larger. Together, the criteria in (2.8) and (2.9) are known as the Wolfe conditions. A line search that satisfies the Wolfe conditions is necessary for the convergence results of the Gauss–Newton method, which we discuss in the next section. Other conditions for ensuring that the step size is not too small include the strong Wolfe conditions and Goldstein conditions [50]. Note that line search methods satisfying both the Armijo and curvature conditions require evaluating the objective function and its gradient during the line search. For large-scale problems, these evaluations can make these line search strategies too expensive.

For our implementation of the Gauss–Newton method, we forego the curvature condition and require only that the Armijo condition in (2.8) be satisfied. We then adopt a backtracking line search strategy to ensure the chosen $\gamma^{(k)}$ is not too small. The backtracking chooses an initial step length γ_0 and a backtracking parameter ρ . It then iteratively backtracks until the chosen parameter results in a sufficient decrease of the objective function. For the Gauss–Newton method, it is logical to choose $\gamma_0 = 1$ following from the step length of Newton’s method. We backtrack by a factor of two using $\rho = 0.5$. Note that this backtracking strategy only requires the evaluation of the objective function, not its gradient. This results in a line search strategy which is comparatively inexpensive and effective in practice. An algorithm

for the backtracking Armijo line search can be seen in Alg. 1.

Algorithm 1 Backtracking Armijo line search algorithm

- 1: Given \mathbf{x} , \mathbf{p} ; choose $\gamma_0 > 0$, ρ , $c_1 \in (0, 1)$; set $\gamma = \gamma_0$
 - 2: **while** $\Phi(\mathbf{x} + \gamma\mathbf{p}) > \Phi(\mathbf{x}) + c_1\gamma\nabla\Phi(\mathbf{x})^\top\mathbf{p}$ **do**
 - 3: $\gamma = \rho\gamma$
 - 4: **end while**
 - 5: **return** γ
-

2.2.3 Stopping Criteria

Another consideration when implementing the Gauss–Newton method is the choice of stopping criteria to judge when the method reaches a suitable solution and should terminate. Ideally, one would simply measure the norm of the distance of the current iterate $\mathbf{x}^{(k)}$ from the true solution \mathbf{x}_{true} and stop the method when this value falls below some tolerance ϵ ,

$$\|\mathbf{x}^{(k)} - \mathbf{x}_{true}\|^2 < \epsilon.$$

However, this approach is impractical in practice because the true solution is typically unavailable. Additionally, convergence to the true solution is usually infeasible due to error and perturbations in the data, i.e., the exact minimizer to the optimization problem, \mathbf{x}^* , may be further than ϵ from the true solution, \mathbf{x}_{true} .

Instead, we look at a number of other potential criteria for stopping the method. One idea is to measure the norm of the gradient at the current iterate and stop the method when this value falls below a chosen tolerance,

$$\|\nabla\Phi(\mathbf{x}^{(k)})\|^2 < \epsilon_1.$$

This choice follows from the fact that the first-order optimality condition for Gauss–Newton dictates that the gradient should converge to the zero vector when the method

reaches a minimum.

Another option is to stop the method when iteration stagnates. This stops the method when the norm of the difference between successive iterates falls below some selected tolerance,

$$\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\|^2 < \epsilon_2.$$

Similarly, one can monitor stagnation in the reduction of the objective function for successive iterates by monitoring

$$|\Phi(\mathbf{x}^{(k+1)}) - \Phi(\mathbf{x}^{(k)})| < \epsilon_3.$$

We note that neither of the criteria using successive iterates guarantees that the method has reached a minimum. For instance, these values may fall below the prescribed tolerance if the step length $\gamma^{(k)}$ is too short, resulting in a small change in successive iterates. This behavior may not necessarily occur at a minimum. Furthermore, none of the of the three criteria above guarantee that any minimum reached is the global minimizer to the problem.

In practice, we use a combination of the criteria above and stop the Gauss–Newton method when any two of the three are fulfilled. The tolerance parameters ϵ_1 , ϵ_2 , and ϵ_3 are chosen distinctly and defined by the user at the initiation of the method. We vary these tolerances depending on the problem. Further details are provided on a case by case basis in the Ch. 4–6.

2.2.4 Gauss–Newton Algorithm

Combining the sections above, we now present an algorithm for the Gauss–Newton method. This can be found below in Alg. 2. The algorithm is presented in pseudo-code and does not include the various input parameters for the line search and stopping criteria discussed in the previous sections.

Algorithm 2 Gauss–Newton Method with Backtracking Armijo Line Search

```

1: Given  $\mathbf{x}^{(0)}$ 
2: for  $k = 0, 1, 2, \dots$  do
3:   Compute  $\Phi(\mathbf{x}^{(k)})$ ,  $\mathbf{J}^{(k)}$ ,  $\mathbf{r}^{(k)}$ 
4:   Solve for step  $\mathbf{p}^{(k)}$  using Eq. 2.5 or Eq. 2.7
5:   Determine step length  $\gamma^{(k)}$  using Backtracking Armijo Line Search
6:   Update  $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \gamma^{(k)}\mathbf{p}^{(k)}$ 
7:   if Stopping Criteria then
8:     return  $\mathbf{x}^{(k+1)}$ 
9:   end if
10: end for

```

2.2.5 Convergence

Lastly, we discuss the convergence behavior of the Gauss–Newton method. The convergence behavior of the method depends on many things including the objective function, its derivatives, the initial guess for $\mathbf{x}^{(0)}$ for the method, and the line search strategy. Also, we must address our use of an inexact solution for the Gauss–Newton step, $\mathbf{p}^{(k)}$, at each iteration. This section recalls some of the convergence results for the Gauss–Newton and inexact Gauss–Newton methods from the literature. We begin by presenting convergence results for the method using an exact solution to the step problem. After, we address the issue of inexact Gauss–Newton steps. For proof of these results and further analysis of the convergence properties of the Gauss–Newton method, we refer the reader again to [2, 50].

Gauss–Newton can be shown to be globally convergent under certain assumptions, i.e., the iteration is guaranteed to find a minimizer (not necessarily global). For the Gauss–Newton method using a line search that satisfies the Wolfe conditions, if the Jacobian $\mathbf{J}(\mathbf{x})$ has singular values bounded away from zero, i.e., has full rank,

for every \mathbf{x} in the region of interest and the residual functions $\mathbf{r}_j(\mathbf{x})$ are Lipschitz continuous for every $j = 1, \dots, m$ within a neighborhood of the level set $\mathcal{L} = \{\mathbf{x} \mid \Phi(\mathbf{x}) \leq \Phi(\mathbf{x}_0)\}$, the method is globally convergent to a minimizer [50]. We note that the Armijo line search outlined in Sec. 2.2.2 does not necessarily fulfill the Wolfe conditions required for this result. We also lose this result if $\mathbf{J}(\mathbf{x})$ is rank deficient for some \mathbf{x} in the region of interest. This may be a problem in practice.

The speed of convergence of the Gauss–Newton method is governed by the proximity of the current iterate $\mathbf{x}^{(k)}$ to the exact minimizer \mathbf{x}^* and by how much the $\mathbf{J}^{(k)\top} \mathbf{J}^{(k)}$ term dominates the summation term of the Hessian in (2.3), i.e., how nonlinear the problem is. If the summation term in the Hessian is the zero matrix, Gauss–Newton converges quadratically in the neighborhood of the true solution, and for problems when the summation term is small, similar superlinear or quadratic-like convergence is observable. Outside the neighborhood of the solution, convergence is linear. This emphasizes the importance of a good initial guess, $\mathbf{x}^{(0)}$, for the Gauss–Newton method. For the numerical experiments in Ch. 4–6, we make note of how each initial guess is chosen on a case by case basis. The near-quadratic convergence behavior of the method in the neighborhood of the solution is also observable in the iteration convergence plots for many of the problems presented in those chapters. Fig. 4.8 shows this clearly for a problem where the initial guess is far from the solution, thus convergence is slow for a number of iterations followed by rapid convergence.

The convergence results above assume exact solutions to the Gauss–Newton step problem in (2.5) and (2.7). However for large-scale problems, exact solutions to these problems may be expensive or infeasible. The proposed solution is to solve the problem for the Gauss–Newton step to a low accuracy using an iterative method. This idea is the basis of inexact Newton methods. Inexact Newton methods can also be shown to be globally convergent to a minimizer under similar assumptions to exact Gauss–Newton given the solution accuracy is high enough. One can prove the con-

vergence properties for inexact Gauss–Newton by viewing the inexact Gauss–Newton step as a perturbation of the exact Gauss–Newton step and bounding the distance between the inexact and exact Gauss–Newton steps with respect to the solution accuracy of the Gauss–Newton step problem. After bounding the distance from the exact solution, the proof for convergence follows the same pattern as the proof for exact Gauss–Newton to guarantee the convergence of inexact Gauss–Newton to a minimizer [43].

2.3 Projected Gauss–Newton

Adding constraints to (2.2) is one way to incorporate prior knowledge of a desired solution. Incorporating such information is particularly useful for our problems of interest where ill-posedness is an issue. In this section, we present the projected Gauss–Newton method as a straightforward way to impose element-wise box constraints on the solution following the descriptions in [27, 38, 63]. The method combines the full Gauss–Newton method, which converges quickly to a solution given an appropriate initial guess, and the projected gradient descent method for which it is easy to implement bound constraints.

To introduce the method, we begin by reformulating (2.2) as the constrained optimization problem

$$\min_{\mathbf{x} \in \mathcal{C}_x} \left\{ \Phi(\mathbf{x}) = \frac{1}{2} \|\mathbf{r}(\mathbf{x})\|^2 \right\}, \quad (2.10)$$

where \mathcal{C}_x is a closed, convex rectangular set imposing element-wise bound constraints on the entries of the solution \mathbf{x} . That is,

$$\mathbf{x} \in \mathcal{C}_x = \{\mathbf{x} \in \mathbb{R}^n \mid l_j \leq [\mathbf{x}]_j \leq u_j \text{ for } j = 1, \dots, n\},$$

where l_j and u_j are lower and upper bounds on the j th entry of solution vector

\mathbf{x} , respectively. Note that the above notation suffices only for real-valued solution vectors. If the solution vector \mathbf{x} is complex-valued, bounds should be implemented bounding both the real and imaginary parts of $[\mathbf{x}]_j$. This is possible, but we do not include bounds on complex-valued solution vectors in this work.

To implement the projected Gauss–Newton method for (2.10), we begin by splitting the solution at each iterate $\mathbf{x}^{(k)}$ into two sets: an active set $\mathcal{A}^{(k)}$ comprised of entries for which the bound constraints are active and an inactive set $\mathcal{I}^{(k)}$ for entries for which the bound constraints are inactive. We denote these two sets by

$$\begin{aligned}\mathcal{A}^{(k)} &= \{[\mathbf{x}^{(k)}]_j \mid ([\mathbf{x}]_j = l_j \parallel [\mathbf{x}^{(k)}]_j = u_j)\} \\ \mathcal{I}^{(k)} &= \{[\mathbf{x}^{(k)}]_j \mid l_j < [\mathbf{x}^{(k)}]_j < u_j\}.\end{aligned}\tag{2.11}$$

Note this division into sets requires that the initial guess for the solution lie in the constraint set, $\mathbf{x}^{(0)} \in \mathcal{C}_x$. The method then forces all further iterates to remain within these bounds. At each iterate $\mathbf{x}^{(k)}$, we separate the iterate into an active subset, $\mathbf{x}_{\mathcal{A}}^{(k)} \subset \mathbf{x}^{(k)}$, and an inactive subset, $\mathbf{x}_{\mathcal{I}}^{(k)} \subset \mathbf{x}^{(k)}$.

On the inactive set, we take the standard Gauss–Newton step. Letting $\mathbf{I}_{\mathcal{I}}^{(k)}$ be the identity matrix with diagonal entries corresponding to active set indices equal to 0, the step on the inactive set is given by solving

$$\left(\mathbf{I}_{\mathcal{I}}^{(k)} \mathbf{J}^{(k)\top} \mathbf{J}^{(k)} \mathbf{I}_{\mathcal{I}}^{(k)}\right) \mathbf{p}_{\mathcal{I}}^{(k)} = -\mathbf{I}_{\mathcal{I}}^{(k)} \mathbf{J}^{(k)\top} \mathbf{r}^{(k)}.\tag{2.12}$$

This system can also be written as

$$\mathbf{p}_{\mathcal{I}}^{(k)} = \underset{\mathbf{p}}{\operatorname{argmin}} \|\mathbf{J}^{(k)} \mathbf{I}_{\mathcal{I}}^{(k)} \mathbf{p} + \mathbf{r}^{(k)}\|^2,\tag{2.13}$$

where the equivalence can be seen by looking at the first-order optimality condition of the least-squares formulation. The first of these expressions is not full rank if

the active set is non-trivial. However, it can be solved iteratively using a slightly modified preconditioned conjugate gradient (PCG) method or directly by truncating the projected matrices and vectors to exclude rows and columns with all zero entries, yielding a reduced, full rank problem. The least-squares problem in (2.13) is also solvable iteratively or directly. For our problems, we typically solve iteratively to a low tolerance as in standard Gauss–Newton. Also, note that for the special case where $u_j = -l_j = \infty$ for all j , the active set is empty, $\mathbf{I}_{\mathcal{I}}^{(k)}$ is the identity, and the problems above revert to their formulations in standard Gauss–Newton.

On the active set, we then perform a gradient descent step given by

$$\mathbf{p}_{\mathcal{A}}^{(k)} = -\mathbf{I}_{\mathcal{A}}^{(k)}(\mathbf{J}^{(k)\top} \mathbf{r}^{(k)}), \quad (2.14)$$

where $\mathbf{I}_{\mathcal{A}}^{(k)} = \mathbf{I} - \mathbf{I}_{\mathcal{I}}^{(k)}$ is the identity matrix with 0’s for diagonal entries with indices in the inactive set. We then combine this step for with the step on the inactive set,

$$\mathbf{p}^{(k)} = \mathbf{p}_{\mathcal{I}}^{(k)} + \mu \mathbf{p}_{\mathcal{A}}^{(k)}. \quad (2.15)$$

Here, the parameter $\mu > 0$ is included to reconcile the difference in scale between the Gauss–Newton and projected gradient descent steps. We follow the recommendation in [27] and use

$$\mu = \frac{\|\mathbf{p}_{\mathcal{I}}^{(k)}\|_{\infty}}{\|\mathbf{p}_{\mathcal{A}}^{(k)}\|_{\infty}}. \quad (2.16)$$

This ensures that step entries on the inactive set are no larger in magnitude than the step entries on the active step. Without this scaling, the method is likely to lose some of its positive, Gauss–Newton-like convergence properties. In the worst case scenario, it reverts to the convergence properties of projected gradient descent.

The next projected Gauss–Newton iterate is then

$$\mathbf{x}^{(k+1)} = \mathbf{Q}(\mathbf{x}^{(k)} + \gamma^{(k)} \mathbf{p}^{(k)}),$$

where \mathbf{Q} is a projection used to ensure that the updated iterate obeys the bound constraints, and $\gamma^{(k)}$ is a line search parameter. Under this projection \mathbf{Q} , variables that would leave the constrained region in the next iterate are projected onto the boundary and join the active set for the next iteration, while other entries update without projection. This projection can be written element-wise as

$$\mathbf{Q}([\mathbf{x}]_j) = \begin{cases} [\mathbf{x}]_j & \text{if } l_j \leq [\mathbf{x}]_j \leq u_j \\ [\mathbf{x}]_j = l_j & \text{if } [\mathbf{x}]_j < l_j \\ [\mathbf{x}]_j = u_j & \text{if } u_j < [\mathbf{x}]_j \end{cases}.$$

The line search parameter $\gamma^{(k)}$ for projected Gauss–Newton is selected using a projected Armijo line search. This follows an identical backtracking approach to the standard Armijo line search when selecting a line search parameter $\gamma^{(k)}$, but it introduces a projection into the Armijo condition in (2.8) to ensure the projected update provides a sufficient decrease in the objective function. This modified Armijo condition is given by

$$\Phi(\mathbf{Q}(\mathbf{x}^{(k)} + \gamma^{(k)}\mathbf{p}^{(k)})) \leq \Phi(\mathbf{x}^{(k)}) + c_1\gamma^{(k)}\mathbf{P}(\nabla\Phi(\mathbf{x}^{(k)}))^\top\mathbf{p}^{(k)}.$$

Here, the constant $c_1 \in (0, 1)$, and $\mathbf{P}(\nabla\Phi(\mathbf{x}^{(k)}))$ is the projected gradient where the projection \mathbf{P} is given element-wise as

$$\mathbf{P}([\nabla\Phi(\mathbf{x}^{(k)})]_j) = \begin{cases} [\nabla\Phi(\mathbf{x}^{(k)})]_j & \text{if } l_j \leq [\mathbf{x}]_j \leq u_j \\ \min\{[\nabla\Phi(\mathbf{x}^{(k)})]_j, 0\} & \text{if } [\mathbf{x}]_j < l_j \\ \max\{[\nabla\Phi(\mathbf{x}^{(k)})]_j, 0\} & \text{if } u_j < [\mathbf{x}]_j. \end{cases}$$

We note that for the special case where the active set is empty for the updated iterate, the projections \mathbf{P} and \mathbf{Q} are equal to the identity, and we revert to the standard Armijo condition and line search.

We end by noting some other necessary considerations for the successful implementation of the method. First, both the inactive set and active sets are expected to change during the projected step update; thus it is necessary to update these sets at every iteration. The projection also affects the stopping criteria for the method. Specifically, the first-order optimality condition of the constrained problem is the projected gradient [2]. The projected gradient sets to zero gradient entries corresponding to entries of the solution vector in the active set that have minima lying outside the feasible region. We cannot expect these entries of the gradient to converge to zero. Instead, these entries in the gradient should be projected to zero when monitoring the convergence of the projected Gauss–Newton method.

As with standard Gauss–Newton, we end our discussion of projected Gauss–Newton by summarizing the material above into Alg. 3, found below.

Algorithm 3 Projected Gauss–Newton Method

- 1: Given a feasible $\mathbf{x}^{(0)}$
 - 2: **for** $k = 0, 1, 2, \dots$ **do**
 - 3: Compute $\Phi(\mathbf{x}^{(k)})$, $\mathbf{J}^{(k)}$, and $\mathbf{r}^{(k)}$
 - 4: Update active set $\mathcal{A}^{(k)}$ and inactive set $\mathcal{I}^{(k)}$
 - 5: Solve for Gauss–Newton step on inactive set $\mathbf{p}_{\mathcal{I}}^{(k)}$ using Eq. 2.12 or Eq. 2.13
 - 6: Compute step on active set $\mathbf{p}_{\mathcal{A}}^{(k)}$ using Eq. 2.14
 - 7: Combine steps $\mathbf{p}^{(k)} = \mathbf{p}_{\mathcal{I}}^{(k)} + \mu\mathbf{p}_{\mathcal{A}}^{(k)}$ with Eq. 2.15 and Eq. 2.16
 - 8: Determine step length $\gamma^{(k)}$ using the projected Armijo Line Search
 - 9: Update $\mathbf{x}^{(k+1)} = \mathbf{Q}(\mathbf{x}^{(k)} + \gamma^{(k)}\mathbf{p}^{(k)})$
 - 10: **if** Stopping Criteria **then**
 - 11: **return** $\mathbf{x}^{(k+1)}$
 - 12: **end if**
 - 13: **end for**
-

2.4 Regularization

We now give an overview of regularization for nonlinear problems. As mentioned previously, the inverse problems corresponding to the nonlinear forward model (2.1) are frequently ill-posed, and their solutions are unstable for noise-affected data. In such cases, regularization is often introduced to the optimization problem (2.2) as a remedy to the various issues arising from ill-posedness and noisy data. The goal of regularization is to modify or change the optimization problem in such a way that a unique solution exists, and if possible, the problem is more stable. This is often done by incorporating prior knowledge or characteristics of a desired solution such as regularity, smoothness, or minimization of some norm by the solution. We also need to consider regularization for the Gauss–Newton step problem (2.5) or (2.7) as the Jacobian $\mathbf{J}(\mathbf{x})$ may be ill-conditioned or not guarantee a unique solution at each iteration.

For the Gauss–Newton method above, we can split our discussion of regularization into two parts, the first on regularization for general nonlinear optimization problems and the second on the options for regularizing the linear problems associated with the Gauss–Newton step. There is significant overlap between the two sections as choices for regularizing the nonlinear optimization problem affect the step update problem and can double as regularization there. Lastly, we note that this discussion of regularization is a brief overview aimed to emphasize its necessity. Regularization is an active field of research, and it is impossible to cover the broad scope of options available here. Specific choices for regularizers are problem dependent, so we leave these until the presentation of the individual problems in Ch. 4–6.

2.4.1 Regularization for Nonlinear Problems

The theory for analyzing regularization methods for nonlinear inverse problems is more difficult than for linear or separable nonlinear problems [8, 15, 16, 48]. Concepts like the singular value decomposition (SVD) that are used to analyze the ill-posedness of linear problems do not always extend to their nonlinear counterparts [12]. One work-around for this issue is to analyze the singular values for a linearization of the nonlinear problem, but it has been shown that an ill-posed nonlinear problem may have a well-posed linearization [16]. Furthermore, analysis for the optimization of nonlinear inverse problems such as (2.2) often requires strict assumptions about the problem that are unrealistic in practice [15, 16]. However, due to the prevalence of nonlinear inverse problems in many applications, much work has been invested into developing effective regularization techniques and theory for nonlinear problems.

One of the most common techniques for regularizing nonlinear optimization problems is iterative regularization. Iterative regularization is based on the early termination of iterative methods when solving ill-posed or noise-affected problems. This is based on the expectation that the early iterates of an iterative method tend to reconstruct low frequency portions of the solution. These iterates are typically less affected by noise in the data, meaning the step updates at those iterates contain more information about the true solution. As iterations continue, noise becomes amplified and is more prevalent in the reconstructed solution. Over a sequence of iterates, this results in the phenomenon of semi-convergence, where the early iterations of a method reduce the relative error between the reconstructed solution $\mathbf{x}^{(k)}$ and the true solution \mathbf{x}_{true} until some optimal iterate, after which the relative error begins to increase for further iterates as noise corrupts the computed solution. The analysis of this behavior for linear optimization problems is based on the singular values and singular vectors of the linear operator for the problem.

In our case, iterative regularization means terminating the projected Gauss–

Newton iteration at an appropriate early iteration before reaching the exact minimizer to the noise-affected problem, \mathbf{x}^* . Iterative regularization is also popular for other nonlinear optimization techniques including the Landweber iteration, nonlinear conjugate gradient method, and quasi-Newton methods [8]. The primary difficulty in iterative regularization is the selection of a suitable stopping iteration. Stop too soon and the method does not reach the best solution possible; stop too late and noise begins to corrupt the recovered solution. Thus to use iterative regularization effectively, one must also find a dependable stopping criteria such as the discrepancy principle [63].

Another regularization strategy is direct regularization, which introduces application specific regularizers to the problem to incorporate prior knowledge of a desired solution. For many applications, properties of the solution are known, for example non-negativity, minimal energy, smoothness, or invertibility. Enforcing such characteristics in the solution treats the ill-posedness of a problem by reducing the space of possible solutions. It can also help alleviate ill-conditioning in the discrete problem. To incorporate these desired properties into a solution, we reformulate (2.10) with a direct regularizer as

$$\min_{\mathbf{x} \in \mathcal{C}_x} \left\{ \Phi(\mathbf{x}) = \frac{1}{2} \|\mathbf{r}(\mathbf{x})\|^2 + \alpha \mathcal{S}(\mathbf{x}) \right\}, \quad (2.17)$$

where $\mathcal{S}(\mathbf{x})$ is a regularizer weighted by a regularization parameter $\alpha > 0$. The introduction of a regularizer to the optimization problem forces the solution to balance between fitting the data and exhibiting desirable characteristics. That is, the introduction of the regularizer changes the problem, i.e., the exact minimizer \mathbf{x}^* changes. The regularization parameter α controls this balance between fitting the data in the residual term and having a highly regularized solution.

One common class of regularizers are quadratic regularizers

$$\mathbf{S}(\mathbf{x}) = \|\mathbf{L}\mathbf{x}\|^2. \quad (2.18)$$

Two examples of this are $\mathbf{L} = \mathbf{I}$, the identity matrix for Tikhonov regularization, and $\mathbf{L} = \nabla_h$, a discretized gradient operator using forward differences. Common non-quadratic regularizers include options such as total variation and p -norm regularizers [15, 48, 63]. These two classes are not unconnected, and non-quadratic regularizers can be addressed by solving a sequence of least-squares problems that include weighted quadratic regularizers [53, 54, 66]. Solving optimization problems with quadratic regularizers is also an essential part of splitting-based methods such as the Split Bregman method for ℓ_1 regularized problems [23]. In this work, we see various choices for $\mathbf{S}(\mathbf{x})$ in later chapters including a nonlinear, hyperelastic regularizer presented in Ch. 5 and a penalty term regularizer in Ch. 6.

Optimal selection of the regularization parameter α is its own challenging problem. Common methods include unbiased predictive risk estimator method (UPRE), generalized cross validation (GCV), the discrepancy principle, and the L-Curve [33, 63]. These methods have limitations: the discrepancy principle requires prior knowledge about the noise level, while the L-curve and GCV methods often require a singular value decomposition (SVD) of the forward operator, which is not available for many nonlinear problems.

Lastly, we make note of statistical methods for nonlinear regularization. These methods maximize an expected likelihood function for the problem after making some underlying assumptions about the statistical distribution of the problem [8]. We do not explore these statistics-based regularization techniques in this dissertation. For further reading on nonlinear regularization, we refer the reader to [15, 16, 33, 63].

2.4.2 Regularization for the Gauss–Newton Step

As mentioned above, regularization is also often needed for the linear problems (2.5) and (2.7) for computing the Gauss–Newton step direction as the Jacobian $\mathbf{J}(\mathbf{x})$ may not guarantee a unique solution to the step problem and may be ill-conditioned. Many of the ideas presented in the previous section extend to regularizing the Gauss–Newton step problem, and direct regularizers introduced to the nonlinear optimization problem affect the Gauss–Newton step problem by changing the derivatives.

The idea of iterative regularization for the linear problems follows the same principle as for nonlinear problems, which we introduced in the previous section. For linear problems, the theory motivating this type of regularization is more clear and follows from the SVD of the Jacobian $\mathbf{J}(\mathbf{x})$ at each iteration [15, 33, 63]. The early iterations of iterative methods (Landweber, Krylov subspace methods, etc.) can be shown to reconstruct portions of the solution corresponding to large singular values and low-frequency singular vectors of the Jacobian while later iterations correspond to smaller singular values and higher-frequency singular vectors. Again, the goal is to stop an iterative method at the optimal iteration where the computed solution most closely matches the true solution rather than the exact minimizer of the noise-affected step equation. In Sect. 2.2.1, we motivated inexact Gauss–Newton as a way to lower the cost associated with computing the Gauss–Newton step, but it also serves as iterative regularization in the case when the step problem is ill-posed.

Direct regularization for the Gauss–Newton step can be inherited through the derivatives of a direct regularization operator for the nonlinear problem as in (2.17) or introduced separately for the Gauss–Newton step problem if the nonlinear problem has no direct regularization term. We discuss the former case, for which the updated Gauss–Newton step problem becomes

$$(\mathbf{J}^{(k)\top} \mathbf{J}^{(k)} + \alpha \nabla^2 \mathbf{S}(\mathbf{x})) \mathbf{p} = -(\mathbf{J}^{(k)\top} \mathbf{r}^{(k)} + \nabla \mathbf{S}(\mathbf{x})), \quad (2.19)$$

where $\nabla \mathbf{S}(\mathbf{x})$ and $\nabla^2 \mathbf{S}(\mathbf{x})$ are the gradient and (approximate) Hessian of the regularization term, respectively. For quadratic regularizers of the form (2.18), this system is symmetric positive definite if \mathbf{L} is full rank, thus ensuring that the exact solution $\mathbf{p}^{(k)}$ is a descent direction. This is helpful when $\mathbf{J}^{(k)}$ is rank deficient and a descent direction for the un-regularized problem is not guaranteed. For quadratic regularizers, we also see that (2.19) is equivalent to the least-squares problem

$$\min_{\mathbf{p}} \left\| \begin{bmatrix} \mathbf{J}^{(k)} \\ \sqrt{\alpha} \mathbf{L} \end{bmatrix} \mathbf{p} + \begin{bmatrix} \mathbf{r}^{(k)} \\ \sqrt{\alpha} \mathbf{L} \mathbf{x}^{(k)} \end{bmatrix} \right\|^2. \quad (2.20)$$

For the case when $\mathbf{S}(\mathbf{x})$ is nonlinear, we typically use a linear, symmetric positive definite approximation of $\nabla^2 \mathbf{S}(\mathbf{x})$ as we did for the Gauss–Newton objective function. The problems in (2.19) and (2.20) can then be solved using direct or iterative methods. For the problems in this dissertation, we typically solve directly regularized problems for the Gauss–Newton step using iterative methods to a low accuracy, thus adding additional iterative regularization to the problem.

We note again that the problems in (2.19) and (2.20) require a choice for the regularization parameter α , which is a separate, challenging problem. For the LAP method and the numerical experiments in Ch. 4, we explore the use of hybrid regularization methods as an option, which seeks to combine the advantages of iterative and direct regularization while automatically selecting a regularization parameter [9, 20, 19, 21].

2.5 Imaging

The nonlinear problems of interest in this work come from applications in imaging. As such, it is necessary to discuss the mathematical framework for imaging that we use to relate continuous reality to the discretized problems of interest. This framework

is used for the presentation of specific problems in Ch. 4–6.

Our notation follows the framework presented by [47]. We consider images as continuously differentiable and compactly supported functions from some domain of interest $\Omega \in \mathbb{R}^d$ where d is the dimension (for our applications, $d = 2$ or 3) to a field \mathbb{F} where the image attains its values or intensities. For our problems, we consider both real-valued images, $\mathbb{F} = \mathbb{R}$, and complex-valued images, $\mathbb{F} = \mathbb{C}$. Using this setup, we then obtain a discrete image $\mathbf{x} \in \mathbb{F}^n$ by evaluating the continuous image at the cell-centers of a rectangular grid with n cells in \mathbb{R}^d .

For the problems in Ch. 4 and 5, we must also consider transformations of images. We define a discrete transformation $\mathbf{y} \in \mathbb{R}^{d \cdot n}$ by evaluating a continuous function $y : \Omega \rightarrow \mathbb{R}^d$ at each of the cell-centers of the rectangular grid defined above. Visually, this maps the original, rectangular grid to a new, transformed grid. An illustration of this can be seen in Fig. 2.1. Many types of transformations are possible within this framework, and in general the number of parameters defining a transformation can be large. Common examples include rigid, affine, and spline based transformations. More complicated parametric transformations and nonparametric transformations are also possible [47, 13]. The problems in Ch. 4 are restricted to rigid, affine transformations consisting of shifts and rotations. Transformations of this kind can be parameterized by small set of variables, which we denote by vectors $\mathbf{w} \in \mathbb{R}^3$ for 2D or \mathbb{R}^6 for 3D. In Ch. 5, we consider more involved transformations that are nonparametric on some portions of the image domain and rigid on others.

Note that in general, the cell-centers of a grid under a given transformation \mathbf{y} do not align to the cell-centers of the original grid, i.e., the known locations of the known image intensities. Consequently to evaluate an image under a transformation \mathbf{y} , we must interpolate from the previously known image intensities. An illustration of this is in Fig. 2.1. For this dissertation, we use bilinear or trilinear interpolation depending on the problem dimension. We represent this interpolation by a sparse matrix

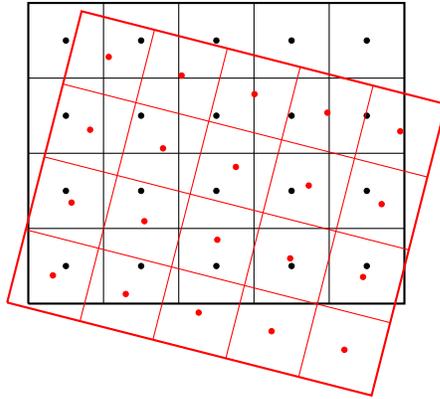


Figure 2.1: An untransformed discrete image is given by evaluating a continuous image at the cell-centers of a rectangular grid (*black*). Under a rigid transformation, the cell-centers of the transformed image (*red*) do not align with previously known image values, necessitating interpolation.

$\mathbf{T}(\mathbf{y}(\mathbf{w})) \in \mathbb{R}^{n \times n}$ dependent on the transformation parameters. The transformed image is then given by a matrix-vector product, $\hat{\mathbf{x}} = \mathbf{T}(\mathbf{y}(\mathbf{w}))\mathbf{x}$. We provide details on constructing the interpolation matrix $\mathbf{T}(\mathbf{y})$, its derivative, and matrix-free implementations in Appendix A.1. For information on linear and other types of interpolation, we refer the reader to [6, 45, 47, 40, 60, 64].

Chapter 3

Linearize and Project Method for Coupled Problems

In this chapter, we introduce the Linearize and Project Method (LAP) for solving the Gauss–Newton step in Ch. 2 for a specific class of nonlinear optimization problems. The problems of interest are characterized by having two sets (or more) of nontrivially coupled variables representing different quantities, for example, different physics. We consider problems that are nonlinear in at least one of these sets, although the method presented is applicable in general to linear problems. Problems of this type arise in numerous applications, and in subsequent chapters we test LAP on several problems of this description arising in imaging. LAP is best suited to problems where the subproblem arising from one of the blocks of variables is well-conditioned and easy to solve. Another strength of the method is its flexibility; LAP allows for various types of regularization and the imposition of bound constraints for enforcing desirable properties in a resulting problem. These options offer multiple ways to deal with the ill-posedness of our problems of interest and introduce prior knowledge of a desired solution.

The chapter is organized in the following way. We begin by reformulating the

forward model of the nonlinear problem from Ch. 2 for problems with two, coupled sets of variables and present the optimization problem for solving the associated inverse problem. We then present the LAP method for solving the Gauss–Newton step. We make note of considerations necessary to regularize and also to use projected Gauss–Newton for problems with coupled sets of variables. Lastly, we discuss two other commonly used methods for problems with coupled variables, block coordinate descent and variable projection, that we use in our numerical experiments in future chapters. Block coordinate descent is the more general of the two approaches and is applicable to general coupled problems. In contrast, variable projection is applicable to a more limited class of separable least-squares problems that are linear in one block of variables and nonlinear in the other.

3.1 Coupled Nonlinear Inverse Problems

We begin by reformulating the discrete forward problem (2.1) for coupled problems. This model is given by

$$\mathbf{d} = \mathbf{F}(\mathbf{x}, \mathbf{w}) + \boldsymbol{\eta},$$

where $\mathbf{d} \in \mathbb{F}^m$ is the observed data, $\mathbf{F}(\cdot, \cdot) : \mathbb{F}^n \times \mathbb{F}^p \rightarrow \mathbb{F}^m$ is a nonlinear forward operator subject to two sets of variables $\mathbf{x} \in \mathbb{F}^n$ and $\mathbf{w} \in \mathbb{F}^p$, and $\boldsymbol{\eta} \in \mathbb{F}^m$ is identically distributed Gaussian white noise. Again, \mathbb{F} is a field, either \mathbb{R} or \mathbb{C} depending on the problem. The only change from the univariate formulation, (2.1), is the dependence on two sets of variables, \mathbf{x} and \mathbf{w} , as opposed to one.

The solution for the inverse problem of recovering \mathbf{x} and \mathbf{w} given a known operator $\mathbf{F}(\cdot, \cdot)$ and a set of observe data \mathbf{d} can then be formulated as the solution of the optimization problem

$$\min_{\mathbf{x} \in \mathcal{C}_x, \mathbf{w} \in \mathcal{C}_w} \left\{ \Phi(\mathbf{x}, \mathbf{w}) = \frac{1}{2} \|\mathbf{r}(\mathbf{x}, \mathbf{w})\|^2 + \alpha \mathbf{S}(\mathbf{x}, \mathbf{w}) \right\}. \quad (3.1)$$

Here, $\mathbf{r} = \mathbf{r}(\mathbf{x}, \mathbf{w}) = \mathbf{F}(\mathbf{x}, \mathbf{w}) - \mathbf{d}$ is the data residual subject to both sets of variables. \mathcal{C}_x and \mathcal{C}_w are closed, convex sets for imposing element-wise bound constraints on the resulting solutions for \mathbf{x} and \mathbf{w} , respectively. Lastly, we introduce a regularizer $\mathbf{S}(\cdot, \cdot)$ weighted by a regularization parameter $\alpha > 0$ that balances the data misfit and regularity of the solution. This is analogous to (2.17) for the univariate case.

We approach the coupled problem using the same projected Gauss–Newton framework from Ch. 2 with special attention paid to the introduction of LAP for determining the Gauss–Newton step. As in the previous chapter, we organize our presentation of the method by introducing LAP for the unconstrained problem with no regularization term using standard Gauss–Newton optimization. After this, we extend it to include the regularization term and the bound constraints within the projected Gauss–Newton framework from Sect. 2.3.

3.2 LAP Method

The LAP method is a tailored solver for the step in (2.5) and (2.7) at each iteration of Gauss–Newton optimization for coupled problems of the form (3.1). LAP is motivated by problems where the subproblem arising from one of the sets of variables is comparatively well-posed and easy to solve. Without loss of generality, we will assume this preferable subset of variables to be \mathbf{w} . Additionally, we assume that the number of variables in \mathbf{w} is considerably less than in \mathbf{x} , i.e., $p \ll n$.

To derive the LAP method, we begin by considering (3.1) without constraints and without the regularization term. This simplified problem is

$$\min_{\mathbf{x}, \mathbf{w}} \left\{ \Phi(\mathbf{x}, \mathbf{w}) = \frac{1}{2} \|\mathbf{r}(\mathbf{x}, \mathbf{w})\|^2 \right\}. \quad (3.2)$$

To set up the Gauss–Newton step for this problem, we first calculate the necessary derivatives. As before, calculating these derivatives requires the Jacobian matrix of

the coupled data residual at each iteration k . For the coupled problem, the Jacobian has a block matrix form with blocks corresponding to the two sets of variables,

$$\mathbf{J}(\mathbf{x}^{(k)}, \mathbf{w}^{(k)}) = \begin{bmatrix} \mathbf{J}_x^{(k)} & \mathbf{J}_w^{(k)} \end{bmatrix}, \quad (3.3)$$

where the blocks are defined as

$$\mathbf{J}_x^{(k)} = \mathbf{J}_x(\mathbf{x}^{(k)}, \mathbf{w}^{(k)}) = \left[\frac{\partial \mathbf{r}_i^{(k)}}{\partial \mathbf{x}_j^{(k)}} \right]_{ij} \quad \text{for } i = 1, 2, \dots, m \quad \text{and} \quad j = 1, 2, \dots, n,$$

and

$$\mathbf{J}_w^{(k)} = \mathbf{J}_w(\mathbf{x}^{(k)}, \mathbf{w}^{(k)}) = \left[\frac{\partial \mathbf{r}_i^{(k)}}{\partial \mathbf{w}_j^{(k)}} \right]_{ij} \quad \text{for } i = 1, 2, \dots, m \quad \text{and} \quad j = 1, 2, \dots, p.$$

Using this block formulation for the Jacobian of the coupled problem, we get a corresponding block structure for the derivatives of the coupled optimization function. These are given by

$$\begin{aligned} \nabla \Phi(\mathbf{x}^{(k)}, \mathbf{w}^{(k)}) &= \begin{bmatrix} \mathbf{J}_x^{(k)\top} \mathbf{r}^{(k)} \\ \mathbf{J}_w^{(k)\top} \mathbf{r}^{(k)} \end{bmatrix} \\ \mathbf{H}^{(k)} &= \begin{bmatrix} \mathbf{J}_x^{(k)\top} \mathbf{J}_x^{(k)} & \mathbf{J}_x^{(k)\top} \mathbf{J}_w^{(k)} \\ \mathbf{J}_w^{(k)\top} \mathbf{J}_x^{(k)} & \mathbf{J}_w^{(k)\top} \mathbf{J}_w^{(k)} \end{bmatrix} \approx \nabla^2 \Phi(\mathbf{x}^{(k)}, \mathbf{w}^{(k)}). \end{aligned} \quad (3.4)$$

Again, we approximate the Hessian for Gauss–Newton using only the Jacobian term and omitting the second, summation term from the full Newton iteration. This is equivalent to a linear approximation of the residual term $\mathbf{r}(\mathbf{x}, \mathbf{w})$, and corresponds to the ‘L’ for linearization in the LAP acronym. The Gauss–Newton step (2.4), at

each iteration is the solution to the block normal equations,

$$\mathbf{H}^{(k)} \begin{bmatrix} \mathbf{p}_x^{(k)} \\ \mathbf{p}_w^{(k)} \end{bmatrix} = - \begin{bmatrix} \mathbf{J}_x^{(k)\top} \mathbf{r}^{(k)} \\ \mathbf{J}_w^{(k)\top} \mathbf{r}^{(k)} \end{bmatrix}. \quad (3.5)$$

If the Jacobian is full rank or the problems are solved iteratively over the same subspace, this is equivalent to the least-squares problem,

$$\min_{\mathbf{x}, \mathbf{w}} \frac{1}{2} \left\| \begin{bmatrix} \mathbf{J}_x^{(k)} & \mathbf{J}_w^{(k)} \end{bmatrix} \begin{bmatrix} \mathbf{p}_x^{(k)} \\ \mathbf{p}_w^{(k)} \end{bmatrix} + \mathbf{r}^{(k)} \right\|^2. \quad (3.6)$$

To solve the block systems above, LAP projects the coupled problem onto a single block of variables. For our examples, we assume the projection is onto the \mathbf{x} block of variables, without loss of generality. This corresponds to the ‘P’ in the LAP acronym. To do the projection, we note that the block system in (3.5) can be written as two, coupled linear systems,

$$\begin{aligned} \mathbf{J}_x^{(k)\top} \mathbf{J}_x^{(k)} \mathbf{p}_x^{(k)} + \mathbf{J}_x^{(k)\top} \mathbf{J}_w^{(k)} \mathbf{p}_w^{(k)} &= -\mathbf{J}_x^{(k)\top} \mathbf{r}^{(k)} \\ \mathbf{J}_w^{(k)\top} \mathbf{J}_x^{(k)} \mathbf{p}_x^{(k)} + \mathbf{J}_w^{(k)\top} \mathbf{J}_w^{(k)} \mathbf{p}_w^{(k)} &= -\mathbf{J}_w^{(k)\top} \mathbf{r}^{(k)}. \end{aligned}$$

The second equation can be solved for $\mathbf{p}_w^{(k)}$ to obtain

$$\mathbf{p}_w^{(k)} = -(\mathbf{J}_w^{(k)\top} \mathbf{J}_w^{(k)})^{-1} (\mathbf{J}_w^{(k)\top} \mathbf{J}_x^{(k)} \mathbf{p}_x^{(k)} + \mathbf{J}_w^{(k)\top} \mathbf{r}^{(k)}). \quad (3.7)$$

Substituting this expression in for $\mathbf{p}_w^{(k)}$ in the first equation and gathering like terms, we then get the projected problem in $\mathbf{p}_x^{(k)}$,

$$\mathbf{J}_x^{(k)\top} \mathbf{P}_{\mathbf{J}_w}^\perp \mathbf{J}_x^{(k)} \mathbf{p}_x^{(k)} = \mathbf{J}_x^{(k)\top} \mathbf{P}_{\mathbf{J}_w}^\perp \mathbf{r}^{(k)}, \quad (3.8)$$

where $\mathbf{P}_{\mathbf{J}_w}^\perp = \mathbf{I} - \mathbf{J}_w^{(k)} (\mathbf{J}_w^{(k)\top} \mathbf{J}_w^{(k)})^{-1} \mathbf{J}_w^{(k)\top}$ is a projector onto the orthogonal comple-

ment of the column space of $\mathbf{J}_w^{(k)}$. This is equivalent to the Schur complement system associated with the $\mathbf{J}_x^{(k)\top} \mathbf{J}_x^{(k)}$ block of the approximate Hessian [24, 57]. For large problems, we solve the projected problem for $\mathbf{p}_x^{(k)}$ using an iterative method with an appropriate preconditioner. Since the problem (3.8) is based on a quadratic approximation of the objective function, we solve it only to a low accuracy following the strategy of inexact Newton methods [50].

We add a few remarks regarding solving (3.8) iteratively. Note that the projection has low rank because $\text{rank}(\mathbf{P}_{\mathbf{J}_w}^\perp) \leq p \ll n$. It follows that a good preconditioner for the $\mathbf{J}_x^{(k)\top} \mathbf{J}_x^{(k)}$ block will also suffice for the projected system. The choice of preconditioner is an important consideration [3, 37, 57]. Matrix-vector multiplications by $\mathbf{J}_x^{(k)}$ (and its transpose) dominate the cost of the iterative solver and also the LAP method. It follows that minimizing the number of iterations (and therefore matrix-vector multiplications) by choosing a suitable preconditioner can improve the efficiency of the method. Also, we will see in Ch. 5 that the choice of preconditioner can significantly impact the numerical performance of the method for some problems.

Also note that if $p \ll n$ is reasonably sized and $\mathbf{J}_w^{(k)}$ is full rank, it is possible to compute the $(\mathbf{J}_w^{(k)\top} \mathbf{J}_w^{(k)})^{-1}$ term in the projection using its Cholesky factors. These factors can be computed once per Gauss–Newton iteration and used repeatedly. Furthermore, the Cholesky factors of $\mathbf{J}_w^{(k)\top} \mathbf{J}_w^{(k)}$ can be computed using a thin QR factorization of the matrix $\mathbf{J}_w^{(k)}$ rather than explicitly forming and factoring $\mathbf{J}_w^{(k)\top} \mathbf{J}_w^{(k)}$. This strategy is particularly attractive for matrix-free implementations of LAP and improves the efficiency of the method significantly. Lastly, we note that for our numerical experiments in Ch. 4 and Ch. 5, we have not observed rank deficiency in $\mathbf{J}_w^{(k)}$.

A similar derivation produces a corresponding least-squares formulation for LAP coming from the coupled Gauss–Newton step given in (3.6). To see this, we look at

the first order optimality condition of (3.6) with respect to \mathbf{p}_w ,

$$0 = \nabla_{\mathbf{p}_w^{(k)}} \left(\frac{1}{2} (\mathbf{J}_x^{(k)} \mathbf{p}_x^{(k)} + \mathbf{J}_w^{(k)} \mathbf{p}_w^{(k)} + \mathbf{r}^{(k)})^\top (\mathbf{J}_x^{(k)} \mathbf{p}_x^{(k)} + \mathbf{J}_w^{(k)} \mathbf{p}_w^{(k)} + \mathbf{r}^{(k)}) \right),$$

which equals

$$0 = (\mathbf{J}_w^{(k)\top} \mathbf{J}_w^{(k)} \mathbf{p}_w^{(k)} + \mathbf{J}_w^{(k)\top} \mathbf{J}_x^{(k)} \mathbf{p}_x^{(k)} + \mathbf{J}_w^{(k)\top} \mathbf{r}^{(k)}).$$

Solving this expression for $\mathbf{p}_w^{(k)}$, we get exactly the expression in (3.7). Substituting this expression into (3.6) gives the projected least-squares problem,

$$\mathbf{p}_x^{(k)} = \underset{\mathbf{p}}{\operatorname{argmin}} \|\mathbf{P}_{\mathbf{J}_w}^\perp (\mathbf{J}_x^{(k)} \mathbf{p} + \mathbf{r}^{(k)})\|^2, \quad (3.9)$$

where $\mathbf{P}_{\mathbf{J}_w}^\perp$ is defined as above. Again, this least-squares problem can be solved iteratively using an appropriate preconditioner. All previous considerations regarding preconditioning and cost of the method still hold, as does the strategy for computing the projection. However, this least-squares problem is better conditioned than the normal equations, which may result in preferable numerical performance for ill-conditioned $\mathbf{J}_x^{(k)}$.

Finally, whether one uses the normal equations or least-squares formulation of LAP to solve for $\mathbf{p}_x^{(k)}$, we calculate $\mathbf{p}_w^{(k)}$ by plugging $\mathbf{p}_x^{(k)}$ into (3.7) and evaluating. Again, if the Cholesky factors for $(\mathbf{J}_w^{(k)\top} \mathbf{J}_w^{(k)})^{-1}$ are computed and stored, this calculation is comparatively very cheap.

3.2.1 LAP with Projected Gauss–Newton

Using the projected Gauss–Newton method framework from Sect. 2.3, we can extend LAP to impose element-wise bound constraints on the solution. While the concept follows directly, we present the extension here in full notation for clarity. To do this, we begin by reformulating problem (3.2) with bounds on the solution, \mathbf{x} and \mathbf{w} , given

by

$$\min_{\mathbf{x} \in \mathcal{C}_x, \mathbf{w} \in \mathcal{C}_w} \left\{ \Phi(\mathbf{x}, \mathbf{w}) = \frac{1}{2} \|\mathbf{r}(\mathbf{x}, \mathbf{w})\|^2 \right\}. \quad (3.10)$$

where \mathcal{C}_x and \mathcal{C}_w are defined as in Sect. 3.1. To extend LAP to projected Gauss–Newton, we begin by partitioning the \mathbf{x} and \mathbf{w} blocks of variables into active and inactive sets as defined in (2.11). We denote by $\mathbf{x}_A^{(k)}$ and $\mathbf{w}_A^{(k)}$ the sets where the bound constraints are active and by $\mathbf{x}_I^{(k)}$ and $\mathbf{w}_I^{(k)}$ the sets where the constraints are inactive.

For the inactive elements of \mathbf{x} , we take the standard Gauss–Newton step solved using LAP. To do this, we first restrict the LAP projection step to the inactive set. Again, this can be done using a normal equations or least-squares approach. The projected problems for $\mathbf{p}_{x,I}^{(k)}$ using each approach are

$$\hat{\mathbf{J}}_w^{(k)\top} \hat{\mathbf{P}}_{\hat{\mathbf{J}}_w}^\perp \hat{\mathbf{J}}_x^{(k)} \mathbf{p}_{x,I}^{(k)} = \hat{\mathbf{J}}_x^{(k)\top} \hat{\mathbf{P}}_{\hat{\mathbf{J}}_w}^\perp \mathbf{r}^{(k)} \quad (3.11)$$

and

$$\mathbf{p}_{x,I}^{(k)} = \underset{\mathbf{p}}{\operatorname{argmin}} \frac{1}{2} \|\hat{\mathbf{P}}_{\hat{\mathbf{J}}_w}^\perp (\hat{\mathbf{J}}_x^{(k)} \mathbf{p} + \mathbf{r}^{(k)})\|^2. \quad (3.12)$$

Here, $\hat{\mathbf{J}}_x^{(k)}$, $\hat{\mathbf{J}}_w^{(k)}$, and $\hat{\mathbf{P}}_{\hat{\mathbf{J}}_w}^\perp$ are $\mathbf{J}_x^{(k)}$, $\mathbf{J}_w^{(k)}$, and $\mathbf{P}_{\mathbf{J}_w}^\perp$ restricted to the inactive set via projection as described in Sect. 2.3. Either of these projected equations can then be solved iteratively using an appropriate choice of preconditioner, noting that the preconditioner will likely also need to be projected to restrict it to the inactive set. The step $\mathbf{p}_{w,I}^{(k)}$ on the inactive set is recovered using the expression

$$\mathbf{p}_{w,I}^{(k)} = -(\hat{\mathbf{J}}_w^{(k)\top} \hat{\mathbf{J}}_w^{(k)})^{-1} (\hat{\mathbf{J}}_w^{(k)\top} \hat{\mathbf{J}}_x^{(k)} \mathbf{p}_{x,I}^{(k)} + \hat{\mathbf{J}}_w^{(k)\top} \mathbf{r}^{(k)}). \quad (3.13)$$

The derivation for the equations is identical to the unrestricted case. Also note that for the case when the upper and lower bounds are ∞ and $-\infty$, i.e., an unconstrained problem, the problem reverts to the LAP framework in Sect. 3.2.

For the elements in the active set, we take the scaled, projected gradient descent step. This projected gradient step is given by

$$\begin{bmatrix} \mathbf{p}_{\mathbf{x},\mathcal{A}}^{(k)} \\ \mathbf{p}_{\mathbf{w},\mathcal{A}}^{(k)} \end{bmatrix} = - \begin{bmatrix} \tilde{\mathbf{J}}_{\mathbf{x}}^{(k)\top} \mathbf{r}^{(k)} \\ \tilde{\mathbf{J}}_{\mathbf{w}}^{(k)\top} \mathbf{r}^{(k)} \end{bmatrix}, \quad (3.14)$$

where $\tilde{\mathbf{J}}_{\mathbf{x}}^{(k)}$ and $\tilde{\mathbf{J}}_{\mathbf{w}}^{(k)}$ are $\mathbf{J}_{\mathbf{x}}^{(k)}$ and $\mathbf{J}_{\mathbf{w}}^{(k)}$ restricted to the active set by projection. We scale this step by a factor $\mu > 0$. Following the suggestion in [27], we set

$$\mu = \frac{\max(\|\mathbf{p}_{\mathbf{x},\mathcal{I}}^{(k)}\|_{\infty}, \|\mathbf{p}_{\mathbf{w},\mathcal{I}}^{(k)}\|_{\infty})}{\max(\|\mathbf{p}_{\mathbf{x},\mathcal{A}}^{(k)}\|_{\infty}, \|\mathbf{p}_{\mathbf{w},\mathcal{A}}^{(k)}\|_{\infty})}. \quad (3.15)$$

This scales the largest entry in the step on the active set so that it is no larger in magnitude than the largest entry on the inactive step with the goal of preserving the convergence properties of the Gauss–Newton method.

We combine the steps on the inactive and active step by

$$\begin{bmatrix} \mathbf{p}_{\mathbf{x}}^{(k)} \\ \mathbf{p}_{\mathbf{w}}^{(k)} \end{bmatrix} = \begin{bmatrix} \mathbf{p}_{\mathbf{x},\mathcal{I}}^{(k)} \\ \mathbf{p}_{\mathbf{w},\mathcal{I}}^{(k)} \end{bmatrix} + \mu \begin{bmatrix} \mathbf{p}_{\mathbf{x},\mathcal{A}}^{(k)} \\ \mathbf{p}_{\mathbf{w},\mathcal{A}}^{(k)} \end{bmatrix}. \quad (3.16)$$

Lastly, we choose a step length for the combined step using the projected Armijo line search described in Sect. 2.3 to ensure that the new iterates, \mathbf{x} and \mathbf{w} , lie within the feasible region. We note again that variables may enter or leave the active and inactive sets, which necessitates updating $\mathcal{A}^{(k)}$ and $\mathcal{I}^{(k)}$ at each iteration.

As with the standard Gauss–Newton approach, most of the computational cost for projected Gauss–Newton is incurred in matrix-vector products by the Jacobian $\mathbf{J}_{\mathbf{x}}^{(k)}$ and its projections onto the active and inactive sets. Barring extreme cases where the number of active boundary variables is of the same order as the total number of variables for the problem, most of the cost for projected Gauss–Newton using LAP is incurred when solving the normal equations or the least-squares problem for the

Gauss–Newton step on the inactive set. Here again, the cost of the iterative solver is dominated by calls to the projected Jacobian, $\hat{\mathbf{J}}_x^{(k)}$, although the projection onto the inactive set is cheap. As before, we solve the step problem iteratively to a low tolerance using an appropriate preconditioner.

3.3 Alternatives Methods for Coupled Problems

To conclude this chapter, this section presents two methods that are used in subsequent chapters as alternatives to LAP for solving problems of the form (3.1), block coordinate descent [50] and variable projection [25]. Both methods are well studied in the literature (see citations below.) Block coordinate descent is an alternating optimization scheme applicable to the general class of coupled problems, whereas variable projection is motivated by the separable least-squares problems in Ch. 4 that are linear in one block of variables and nonlinear in the other.

3.3.1 Block Coordinate Descent for Coupled Problems

Block Coordinate Descent (BCD) represents an uncoupled approach to solving (3.1); see [7, 34, 50]. For the method, the variables are separated into a number of blocks. The method then optimizes over one block while holding the others fixed. This is done sequentially over all the blocks. One iteration is completed after a full cycle of optimizing over every block of variables. The method then iterates until convergence or some other stopping criteria is satisfied.

For our coupled problems, we partition the variables for BCD into the two sets suggested by the problem, \mathbf{x} and \mathbf{w} . The k th iteration then requires solving the following problems. First fixing $\mathbf{w}^{(k)}$, we optimize for $\mathbf{x}^{(k+1)}$ by solving

$$\mathbf{x}^{(k+1)} = \underset{\mathbf{x} \in \mathcal{C}_x}{\operatorname{argmin}} \Phi(\mathbf{x}, \mathbf{w}^{(k)}). \quad (3.17)$$

Then, fixing $\mathbf{x}^{(k+1)}$ we optimize for $\mathbf{w}^{(k+1)}$ by solving

$$\mathbf{w}^{(k+1)} = \underset{\mathbf{w} \in \mathcal{C}_w}{\operatorname{argmin}} \Phi(\mathbf{x}^{(k+1)}, \mathbf{w}). \quad (3.18)$$

Solving these two problems completes a single iteration of the method. Note that BCD is decoupled in that while optimizing over one set of variables, it neglects optimization over the other. For problems with tightly coupled variables, this may result in slow convergence [7, 50]. However, the method offers a number of advantages. It is applicable to a wide class of problems, is straightforward to implement, and extends easily to allow for bound constraints and various types of regularization on either block of variables.

For our implementation, we solve (3.17) using a single iteration of projected Gauss–Newton using an iterative solver to compute the search direction to low accuracy. We solve (3.18) using a single iteration of projected Gauss–Newton with a direct solver on the normal equations to compute the inactive step since $p \ll n$ is moderately sized, although an iterative solver may be used. The total cost of a single BCD iteration is then on the same order of magnitude as a single iteration of LAP. The single projected Gauss–Newton iteration for $\mathbf{x}^{(k+1)}$ requires solving a normal equations or least-squares system corresponding to the matrix $\mathbf{J}_x^{(k)}$, while a direct solver for updating $\mathbf{w}^{(k+1)}$ requires solving a problem corresponding to $\mathbf{J}_w^{(k)}$.

3.3.2 Variable Projection for Separable Least-squares

Variable projection (VarPro) is a popular method for separable least-squares problems, a specific class of coupled nonlinear problems; see [26, 25, 51]. We present numerical results for two problems of this class in Ch. 4. The forward operator for separable least-squares problems is nonlinear in one set of variables and linear in the other. For our problems, we assume the problem to be nonlinear in block \mathbf{w} and

linear in block \mathbf{x} , giving the residual in (3.1) the form

$$\hat{\mathbf{r}}(\mathbf{x}, \mathbf{w}) = \mathbf{F}(\mathbf{w})\mathbf{x} - \mathbf{d},$$

where $\mathbf{F}(\mathbf{w})$ is an operator dependent on \mathbf{w} . The corresponding optimization problem without regularizer is then

$$\min_{\mathbf{x} \in \mathcal{C}_x, \mathbf{w} \in \mathcal{C}_w} \left\{ \hat{\Phi}(\mathbf{x}, \mathbf{w}) = \frac{1}{2} \|\hat{\mathbf{r}}(\mathbf{x}, \mathbf{w})\|^2 \right\}. \quad (3.19)$$

Within a Gauss–Newton framework, VarPro solves a reduced problem for the step by eliminating the linear set of variables via projection. The resulting reduced problem is a nonlinear optimization problem in \mathbf{w} , which can be solved using an optimization scheme of choice. We use Gauss–Newton for the reduced nonlinear problem. We note that VarPro’s strategy of solving a reduced problem after eliminating a block of variables through projection is similar to the LAP method presented in Sect. 3.2. The key difference is that LAP linearizes the problem before projection while VarPro does not. This necessitates linearity in the \mathbf{x} block of variables for VarPro and complicates applying constraints to the linear block of variables.

Projecting the problem onto a reduced subspace in VarPro requires solving a linear least-squares problem in \mathbf{x} using the operator $\mathbf{F}(\mathbf{w}^{(k)})$ that depends on the current $\mathbf{w}^{(k)}$ parameters. VarPro is particularly effective for problems when this projection can be done efficiently and accurately. We define the projection by

$$\mathbf{x}^{(k)}(\mathbf{w}) = \operatorname{argmin}_{\mathbf{x} \in \mathcal{C}_x} \hat{\Phi}(\mathbf{x}, \mathbf{w}^{(k)}). \quad (3.20)$$

For our case where n is large, this can be solved using an iterative method [30]. Substituting the projection into Eq. 3.1 for \mathbf{x} , we get the resulting nonlinear optimization

problem in \mathbf{w} at each iterate,

$$\mathbf{w}^{(k)} = \underset{\mathbf{w} \in \mathcal{C}_w}{\operatorname{argmin}} \hat{\Phi}(\mathbf{x}^{(k)}(\mathbf{w}), \mathbf{w}). \quad (3.21)$$

We then solve this reduced nonlinear optimization problem to update \mathbf{w} with a single Gauss–Newton step, noting that the solution (3.20) also provides an updated guess for $\mathbf{x}^{(k)} = \mathbf{x}^{(k)}(\mathbf{w})$ at each Gauss–Newton iteration.

Imposing constraints on the set of linear variables using VarPro poses an issue. To see this, we note that when \mathbf{w} is unconstrained, i.e., $\mathcal{C}_w = \mathbb{R}^p$, the first order optimality condition for the reduced problem, (3.21), is given by the product rule

$$\mathbf{0} = \nabla_{\mathbf{w}} \hat{\Phi}(\mathbf{x}(\mathbf{w}), \mathbf{w}) + \nabla_{\mathbf{w}} \mathbf{x}(\mathbf{w}) \nabla_{\mathbf{x}} \hat{\Phi}(\mathbf{x}(\mathbf{w}), \mathbf{w}). \quad (3.22)$$

If \mathbf{x} is unconstrained (i.e. $\mathcal{C}_x = \mathbb{F}^n$), the second term in this expression vanishes because $\nabla_{\mathbf{x}} \hat{\Phi}(\mathbf{x}(\mathbf{w}), \mathbf{w}) = \mathbf{0}$ due to the first order optimality condition in (3.20). However, if there are active constraints on the linear variables, this second term may not vanish because $\nabla_{\mathbf{x}} \hat{\Phi} \neq \mathbf{0}$. This issue also arises if the linear problem (3.20) is solved inaccurately and thus does not fulfill the first order optimality condition. In these cases, neglecting this second term may degrade the performance of VarPro.

When $\mathcal{C}_x = \mathbb{R}^n$, the second term in (3.22) can be driven to $\mathbf{0}$ by solving (3.20) to a higher accuracy. This differs from the inexact Newton approaches used in LAP and BCD, where we solve the subproblem associated with the linear block of variables to low accuracy. Solving (3.20) to high accuracy may or may not be possible depending on the conditioning of the problem and the amount of noise in the data. Even if possible, this strategy costs significantly more than the low accuracy solves in LAP and BCD in terms of inner iterations and function calls by the operator $\mathbf{F}(\mathbf{w})$. This consideration is important noting that the operator associated with this system is of the same size as the Jacobian, \mathbf{J}_x , which dominates the cost of the other two methods,

and means that in practice VarPro iterations will be more expensive than the other methods.

If (3.20) has nontrivial constraints, is it necessary to calculate $\nabla_{\mathbf{w}}\mathbf{x}(\mathbf{w})$ for the second term of (3.22) and solve the constrained projection problem to a high accuracy. Computing this gradient term can be as difficult as solving the initial problem (3.19), and the cost for a more accurate solution remains. As such, implementing efficient constraints on the linear variables in VarPro proves challenging. Some progress has been made on the implementation of box constraints for the method using a pseudo-derivative approach to compute approximate gradients for the nonlinear problem [59]. For our numerical experiments in Ch. 4, we omit constraints on the linear block of variables for VarPro. This impacts its performance compared to LAP and BCD for which we use element-wise bounds for some problems.

Chapter 4

Motion Correction Problems

This chapter shows numerical results using LAP for two motion correction problems in super-resolution and magnetic resonance imaging (MRI). Both problems are examples of separable least-squares problems, a specific class of coupled problems that are linear in one set of variables and nonlinear in the other. Problems of this type have received much attention in the research community [26, 25, 51]. For motion correction problems, the goal is to simultaneously recover high-resolution images and motion parameters from noisy, motion-affected data. We show the utility of LAP for solving problems of this class. This includes showing that the convergence rate, cost per iteration, and resulting solutions are preferable to block coordinate descent (BCD) and variable projection (VarPro), two commonly used methods for this class of problems. The chapter begins by presenting a framework for coupled problems in image reconstruction from motion-affected measurements. This framework fits within the more general framework for coupled nonlinear problems presented in Ch. 3. Afterward, we introduce the specific formulation of the super-resolution and MRI motion correction problems in detail and present the results of numerical experiments for LAP, BCD, and VarPro for solving each.

The work in this chapter was organized into a paper by the author, L. Ruthotto,

and J. Nagy and is in revision for publishing. The results and content of this chapter borrow heavily from this paper [36].

4.1 Motion Correction Imaging Problem

From Ch. 2, we recall the definition of images as continuously differentiable and compactly supported functions from some domain of interest $\Omega \in \mathbb{R}^d$ (typically, $d = 2$ or 3) to a field \mathbb{F} where the image attains its values or intensities. The problems in this chapter contain real-valued images, $\mathbb{F} = \mathbb{R}$, and complex-valued images, $\mathbb{F} = \mathbb{C}$. Using this setup, we then obtain a discrete image $\mathbf{x} \in \mathbb{F}^n$ by evaluating the continuous image at the cell-centers of a rectangular grid with n cells in \mathbb{R}^d .

We recall the definitions for discrete transformations presented in Sect. 2.5. The discrete transformation $\mathbf{y} \in \mathbb{R}^{n-d}$ is obtained by evaluating a continuous function $y : \Omega \rightarrow \mathbb{R}^d$ at the cell-centers of the rectangular grid. For the problems in this chapter, we restrict ourselves to the class of rigid transformations consisting of shifts and rotations parameterized by a small set of variables, which we denote by $\mathbf{w} \in \mathbb{R}^{(3,6)}$ for 2D and 3D, respectively. Lastly, recall that the cell-centers of a grid under the transformation $\mathbf{y}(\mathbf{w})$ do not align to the cell-centers of the untransformed grid. To evaluate an image $\hat{\mathbf{x}}$ under the transformation $\mathbf{y}(\mathbf{w})$, we linearly interpolate from the previously known image coefficients \mathbf{x} , although other options are possible, e.g., nearest neighbor or spline interpolation [6, 45, 47, 40, 60, 64]. This interpolation is represented as a sparse matrix $\mathbf{T}(\mathbf{y}(\mathbf{w})) \in \mathbb{R}^{n \times n}$ determined by the transformation. The transformed image is then given by a matrix-vector product, $\hat{\mathbf{x}} = \mathbf{T}(\mathbf{y}(\mathbf{w}))\mathbf{x}$. In practice, this interpolation can be implemented in a matrix-free framework and the interpolated image retrieved via a function call, however, the matrix-vector formulation is conceptually useful. Details on linear interpolation and rigid transformation can be found in Appendices A.1 and A.2.

Using these definitions for discrete images and transformations defined in Ch. 2 and above, we express the forward model for the motion correction imaging problem with N data observations as

$$\mathbf{d}_j = \mathbf{K}_j \mathbf{T}(\mathbf{y}(\mathbf{w}_j)) \mathbf{x} + \boldsymbol{\eta}_j \quad \text{for } j = 1, 2, \dots, N, \quad (4.1)$$

where $\mathbf{d}_j \in \mathbb{F}^{m_j}$ is the j th data observation of the true image \mathbf{x} subject to the rigid transformation parameterized by \mathbf{w}_j , $\mathbf{T}(\mathbf{y}(\mathbf{w}_j))$ is an interpolation as defined above, $\mathbf{K}_j \in \mathbb{F}^{m_j \times n}$ is a problem-specific imaging operator, and $\boldsymbol{\eta}_j$ is image noise. We assume the noise to be independent and identically distributed Gaussian noise. To simplify notation for the problem, we typically concatenate the data measurements over all observations into a single column vector $\mathbf{d} \in \mathbb{F}^m$ where $m = m_j \cdot N$. The same is done for the motion parameters, giving $\mathbf{w} \in \mathbb{R}^p$ where $p = 3N$ or $6N$ for 2D and 3D problems, respectively. Here, we have $p \ll n$ as assumed in Ch. 3. This observation informs several of our choices during optimization.

Using the model in (4.1), the motion correction imaging problem aims to recover both the unknown image coefficients \mathbf{x} and motion parameters \mathbf{w} given a set of data observations $\{\mathbf{d}_1, \mathbf{d}_2, \dots, \mathbf{d}_N\}$. We formulate this as the optimization problem

$$\min_{\mathbf{x} \in \mathcal{C}_x, \mathbf{w} \in \mathcal{C}_w} \left\{ \Phi(\mathbf{x}, \mathbf{w}) = \frac{1}{2} \|\mathbf{K}\mathbf{T}(\mathbf{w})\mathbf{x} - \mathbf{d}\|^2 + \frac{\alpha}{2} \|\mathbf{L}\mathbf{x}\|_2^2 \right\}. \quad (4.2)$$

Here, the matrices \mathbf{K} and $\mathbf{T}(\mathbf{w})$ aggregate the observations from the forward model and have the following block structure

$$\mathbf{K} = \begin{bmatrix} \mathbf{K}_1 & & & \\ & \mathbf{K}_2 & & \\ & & \ddots & \\ & & & \mathbf{K}_N \end{bmatrix} \quad \text{and} \quad \mathbf{T}(\mathbf{w}) = \begin{bmatrix} \mathbf{T}(\mathbf{y}(\mathbf{w}_1)) \\ \mathbf{T}(\mathbf{y}(\mathbf{w}_2)) \\ \vdots \\ \mathbf{T}(\mathbf{y}(\mathbf{w}_N)) \end{bmatrix}.$$

$\mathcal{C}_x \subset \mathbb{F}^n$ and $\mathcal{C}_w \subset \mathbb{R}^p$ are rectangular sets used to impose bound constraints on \mathbf{x} and \mathbf{w} , respectively. Lastly, we introduce a regularization term to balance the misfit of the data and the regularity of the reconstructed image as discussed in Sec. 2.4. This term is subject to the regularization operator, \mathbf{L} , and the regularization parameter $\alpha > 0$. The choice of \mathbf{L} is problem specific, and common choices include the discrete gradient operator or the identity. We discuss choices of \mathbf{L} and α for the super-resolution and MRI motion correction problem in the respective sections. One positive aspect of the LAP solver is, for $\mathbf{L} = \mathbf{I}$, the method allows for hybrid methods that automatically select a new, appropriate regularization parameter α at every Gauss–Newton iterate [9, 19, 20, 21]. In our numerical experiments, we investigate one such hybrid method for automatic regularization parameter selection [9] as well as direct regularization using a fixed α value.

Solving the optimization problem in (4.2) requires the Jacobian of the residual at a given iterate, $\mathbf{r}(\mathbf{x}, \mathbf{w}) = \mathbf{K}\mathbf{T}(\mathbf{w})\mathbf{x} - \mathbf{d}$, with respect to both blocks of variables to calculate the gradient and approximate Hessian. These are given by

$$\mathbf{J}_x = \mathbf{K}\mathbf{T}(\mathbf{w})$$

$$\mathbf{J}_w = \mathbf{K} \text{diag}(\nabla_{\mathbf{w}_1}(\mathbf{T}(\mathbf{y}(\mathbf{w}_1))\mathbf{x}), \dots, \nabla_{\mathbf{w}_N}(\mathbf{T}(\mathbf{y}(\mathbf{w}_N))\mathbf{x})),$$

where each term $\nabla_{\mathbf{w}_j}(\mathbf{T}(\mathbf{y}(\mathbf{w}_j))\mathbf{x})$ for $j = 1, \dots, N$ is the gradient of the transformed image at the transformation $\mathbf{y}(\mathbf{w}_j)$. This can be computed using the chain rule as

$$\nabla_{\mathbf{w}_j}(\mathbf{T}(\mathbf{y}(\mathbf{w}_j))\mathbf{x}) = \nabla_{\mathbf{y}(\mathbf{w}_j)}\mathbf{T}(\mathbf{y}(\mathbf{w}_j))\mathbf{x}\nabla_{\mathbf{w}_j}\mathbf{y}(\mathbf{w}_j).$$

The first of these terms is the derivative of the interpolation with respect to the transformed rectangular grid, and the second term is the derivative of the transformation with respect to the rigid transformation parameters. Details for calculating the two terms in this expression can be found in Appendices A.1 and A.2, respectively. The

derivation for the 2D case can also be found in [7]. Both Jacobians, $\mathbf{J}_x \in \mathbb{F}^{m \times n}$ and $\mathbf{J}_w \in \mathbb{R}^{m \times p}$, are sparse.

Next, we look at numerical experiments for two specific examples of motion correction problems: super-resolution and motion correction for magnetic resonance imaging (MRI). For super-resolution, we show results for both 2D and 3D examples. The super-resolution problems both deal with real-valued images and show LAP’s flexibility with respect to regularization and element-wise bound constraints. Motion correction for MRI gives an example of problem (4.2) where the resulting image is complex-valued. Both motion correction problems are separable least-squares problems of the form (3.1), and the variables are partitioned into two blocks: one corresponding to the linear image variables, \mathbf{x} , and the other the nonlinear motion parameters, \mathbf{w} . As introduced in Sect. 3.3.2 and 3.3.1, VarPro and BCD are two popular methods for solving problems of this form. For our numerical experiments, we compare the results with LAP for solving the super-resolution and MRI motion correction problems with those of VarPro and BCD. Specifically, we look at LAP’s solution quality, convergence, and computational cost when compared to its competitors.

4.2 Super-Resolution

For super-resolution, we present examples in both two and three dimensions. For the 2D problem, the block of image variables, \mathbf{x} , has dimension $n = 16,384$, and the block of motion variables, \mathbf{w} , has size $p = 96$. For the 3D problem, those dimensions are $n = 2,211,840$ and $p = 768$. We separate the section into two parts corresponding to the two problems.

4.2.1 Two Dimensional Super-Resolution

To construct a 2D super-resolution problem with a known ground truth image and motion parameters, we use the real-valued, 2D brain MRI dataset provided in FAIR [47] (original resolution 128×128) to generate 32 frames of low-resolution test data (resolution 32×32) after applying 2D rigid body transformations with randomly chosen parameters and adding varying amounts of Gaussian white noise (see below for details). The resulting total number of parameters in the optimization is 16,528 corresponding to $\mathbf{x} \in \mathbb{R}^{16,384} = \mathbb{R}^{128 \times 128}$ for the image and $\mathbf{w} \in \mathbb{R}^{96}$ for the motion. In the coupled least-squares framework from (4.2), the imaging operator \mathbf{K} is then a block diagonal matrix composed of 32 identical down-sampling matrices $\mathbf{K} = \mathbf{K}_j \in \mathbb{R}^{1024 \times 16384}$ for $j = 1, \dots, 32$ along the diagonal, which relate the high-resolution image to the lower resolution data via block averaging. The block-averaging matrix \mathbf{K} is sparse and can be expressed as a product of Kronecker product matrices. Other downsampling options are also feasible within this framework.

As mentioned in [7], the choice of initial guess is crucial in super-resolution problems, so we perform rigid registration of the low-resolution data onto the first data frame (resulting in 31 separate rigid registration problems) to obtain a starting guess for the motion parameters. The resulting relative error of the parameters is around 2%. Using these parameters, we solve the linear image reconstruction problem to obtain a starting guess for the image with relative error of about 5%.

We then solve the problem using LAP, VarPro, and BCD for noise levels of 1%, 2%, and 3%. For all three approaches, we compare two regularization strategies. All three methods are run using the discretized gradient operator, $\mathbf{L} = \nabla_h$ with a fixed regularization parameter $\alpha = 0.01$. LAP and BCD are run with the Golub–Kahan hybrid regularization approach mentioned in Sect. 2.4 (denoted as HyBR in tables and figures). The hybrid regularization for HyBR cannot be applied directly to the VarPro optimization problem (3.20), so instead we use a fixed $\alpha = 0.01$ and

the identity as our second regularizer when using this method, $\mathbf{L} = \mathbf{I}$. The regularization methods chosen for this problem may not be optimal but were chosen to highlight the generality and flexibility of regularization options allowed within the LAP framework. As mentioned in Sect. 2.4, quadratic regularizers are common and also play an important role in solving problems with other, non-quadratic regularizers [23, 53, 54, 66]. We include HyBR to highlight how hybrid regularizers work within the LAP framework [9, 20, 19, 21]. To allow for comparison between methods and previous super-resolution work, we chose to use the same regularization parameter as in [7]. For more rigorous selection criteria, we refer the reader to the methods mentioned in Sect. 2.4. For LAP and BCD, we add bound constraints on the image space of $[0, 1]$ for both choices of regularizer, with both bounds active in practice. The number of active bounds varies for different noise levels and realizations of the problem, but can include as many as 30 – 35% of the image variables for both LAP and BCD. VarPro is run without bound constraints. Neither constraints nor regularization are imposed on the motion parameters for the three methods.

To compare the results for the three methods, we compare the rate of convergence, quality of the recovered solutions, and the computational cost. For the convergence rate and resulting solutions, we monitor the relative errors of the resultant image and motion parameters. We separate relative errors for the image and motion for plotting, and convergence behavior for these errors against iteration can be seen in Fig. 4.2 for the problem with 2% added noise. The corresponding resulting images for the 2% error case can be seen in Fig. 4.1.

The primary computational cost of the optimization for LAP, VarPro, and BCD comes from solving two different types of linear systems at each outer optimization iteration, one associated with the image variables and another with the motion variables. LAP solves these systems to determine the Gauss–Newton step. We use LSQR [52] with a stopping tolerance of 10^{-2} to solve (3.9) for both regularization

approaches. The motion step (3.7) is computed using the Cholesky factors of $\mathbf{J}_w^\top \mathbf{J}_w$ to invert this matrix. VarPro requires solving the linear system (3.20) in the image parameters for each evaluation of the reduced objective function (3.21). For both choices of regularization, we solved this system by running LSQR for a fixed number of 20 iterations. This provided a sufficiently accurate gradient to perform the Gauss–Newton optimization using the reduced objective function, see Sect. 3.3.2, but required more LSQR iterates than the similarly sized systems in LAP and BCD that were solved to low accuracy. Gauss–Newton on the reduced dimensional VarPro function (3.21) requires solving the reduced linear system in the motion parameters, which we solve using Cholesky factorization on the normal equations. For BCD, coordinate descent requires alternating solutions for the linear system in the image, (3.17), and a nonlinear system in the motion parameters, (3.18). For both of these, we take a single projected Gauss–Newton step. For the image, this is solved using LSQR with a stopping tolerance of 10^{-2} using MATLAB’s `lsqr` function for direct regularization and HyBR’s LSQR for hybrid regularization. For the motion, we use Cholesky factorization on the normal equations.

To compare computational costs during the optimization, we track the number of matrix-vector multiplications by the Jacobian operator associated with the image, \mathbf{J}_x , which has dimension $32,768 \times 16,384$. Matrix-vector multiplications with this operator dominate the optimization cost compared to multiplications by \mathbf{J}_w , which has size $32,768 \times 96$. Multiplications with \mathbf{J}_x and its transpose are required for LAP and BCD when solving the system for the Gauss–Newton step for the image step, while for VarPro, they are necessary for the least-squares solve within the reduced objective function. For all three methods, these multiplications are also required in the Armijo line search. As such, the number of these multiplications provides a fair metric for comparing the computational cost of all three methods for the optimization.

Table 4.1 includes relevant values to compare the convergence, solution quality

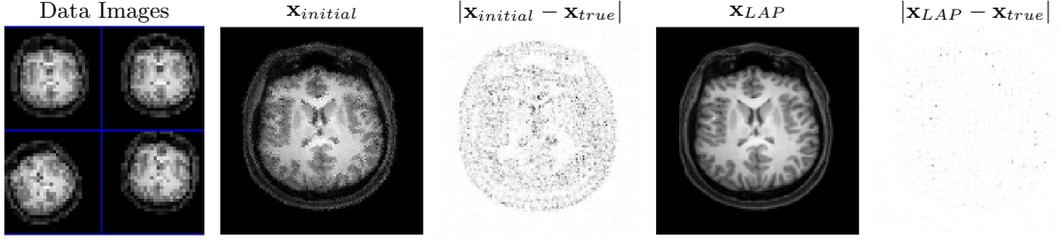


Figure 4.1: The images from left to right show a montage of data frames, the initial guess $\mathbf{x}_{initial}$, the absolute value of the error of the initial guess, the reconstructed image \mathbf{x}_{LAP} for $LAP + \nabla_h$, and the absolute value of its error for the 2D super-resolution problem with 2% noise.

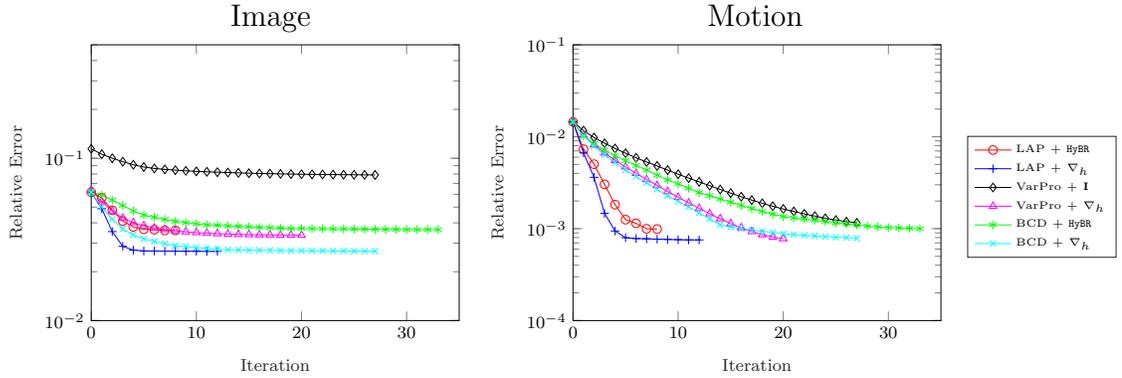


Figure 4.2: These plots show the relative errors for both the reconstructed image and the motion parameters for the 2D super-resolution problem with 2% added noise. We note that methods using the discrete gradient regularizer reach lower minima for the image in this problem, and LAP with the gradient regularizer outperforms both VarPro and BCD in recovering the motion parameters in the early iterations of the method for all noise levels tested.

for both image and motion parameters, and computational cost for all three methods for both regularizers. Values include the average number of optimization iterations, minimum relative errors for image and motion, number of matrix-vector multiplications by \mathbf{J}_x , and CPU timings for the methods taken over 10 different realizations of the problem for all three noise levels is in Table 4.1.

The results show that for direct regularization using the discrete gradient operator, the solutions for all three methods are comparable in terms of the relative error for the motion, with LAP and BCD slightly outperforming VarPro for the relative

| | | | | | | |
|----------|-----------------------|-------------|------------------------|------------------------|-------------|-------------|
| 1% Noise | | Iter. | Rel. Err. \mathbf{x} | Rel. Err. \mathbf{w} | MatVecs. | Time(s) |
| | LAP + HyBR | 11.1 | 5.34e-2 | 1.82e-2 | 93.4 | 14.3 |
| | LAP + ∇_h | 13.1 | 3.76e-2 | 1.78e-2 | 62.2 | 15.4 |
| | VarPro + \mathbf{I} | 25.7 | 8.60e-2 | 1.84e-2 | 554.0 | 60.1 |
| | VarPro + ∇_h | 21.1 | 4.12e-2 | 1.81e-2 | 462.0 | 56.0 |
| | BCD + HyBR | 12.0 | 5.93e-2 | 2.03e-2 | 68.1 | 22.9 |
| | BCD + ∇_h | 29.7 | 3.95e-2 | 1.79e-2 | 89.6 | 55.4 |
| 2% Noise | | Iter. | Rel. Err. \mathbf{x} | Rel. Err. \mathbf{w} | MatVecs. | Time(s) |
| | LAP + HyBR | 12.0 | 6.38e-2 | 1.86e-2 | 77.0 | 12.5 |
| | LAP + ∇_h | 15.9 | 4.38e-2 | 1.81e-2 | 66.4 | 15.9 |
| | VarPro + \mathbf{I} | 29.6 | 1.03e-1 | 1.86e-2 | 632.0 | 64.7 |
| | VarPro + ∇_h | 21.4 | 5.05e-2 | 1.85e-2 | 468.0 | 54.9 |
| | BCD + HyBR | 13.4 | 7.10e-2 | 2.15e-2 | 53.1 | 25.0 |
| | BCD + ∇_h | 29.9 | 4.55e-2 | 1.82e-2 | 91.7 | 57.0 |
| 3% Noise | | Iter. | Rel. Err. \mathbf{x} | Rel. Err. \mathbf{w} | MatVecs. | Time(s) |
| | LAP + HyBR | 12.3 | 7.54e-2 | 2.28e-2 | 68.6 | 16.3 |
| | LAP + ∇_h | 18.5 | 5.34e-2 | 2.22e-2 | 74.2 | 22.0 |
| | VarPro + \mathbf{I} | 34.4 | 1.27e-1 | 2.32e-2 | 728.0 | 86.7 |
| | VarPro + ∇_h | 23.9 | 6.17e-2 | 2.13e-2 | 518.0 | 72.5 |
| | BCD + HyBR | 15.0 | 7.86e-2 | 2.32e-2 | 52.9 | 31.9 |
| | BCD + ∇_h | 29.7 | 5.42e-2 | 2.24e-2 | 83.2 | 63.9 |

Table 4.1: This table shows data for the optimization of the 2D super-resolution problem for multiple values of added Gaussian noise. The columns from left to right give the stopping iteration, relative error of the solution image, relative error of the solution motion, number of matrix-vector multiplications during optimization, and time in seconds using `tic` and `toc` in `MATLAB`. All values are averages taken from 10 instances with different motion parameters, initial guesses, and noise realizations. The best results for each column is bold-faced.

error of the recovered images. This is likely due to the bound constraints on the image implemented for those two methods. Furthermore, these solutions are superior to those for all three methods using hybrid regularization or the identity operator, suggesting that this is a more appropriate regularizer for this problem. LAP with the discrete gradient operator recovers the most accurate reconstructed image of the three methods and achieves better or comparable recovery of the motion parameters. This is observable in the relative error plots for the 2% added noise case in Fig. 4.2, and for the problem over all three noise levels in Table 4.1. We can also see from the relative error plots that LAP tends to recover the correct motion parameters earlier in the Gauss–Newton iterations than either BCD or VarPro. In terms of cost, both the LAP and BCD iterations cost significantly less in terms of time and matrix-vector multiplications than those of VarPro, resulting in faster CPU times and fewer matrix-vector multiplications for the entire optimization. However while BCD is also

relatively cheap in terms of matrix-vector multiplies and CPU time, LAP outperforms it in terms of solution quality. Overall, the LAP approach compares favorably to VarPro and BCD for this example in terms of both the resulting solutions and cost, outperforming both methods.

4.2.2 Three Dimensional Super-Resolution

Next, we compared LAP, BCD, and VarPro on a larger three dimensional super-resolution problem. Again, we use a real-valued, 3D brain MRI dataset provided in FAIR [47] to construct a super-resolution problem with a known ground truth image and motion parameters. The ground truth image (resolution $160 \times 96 \times 144$) is used to generate 128 frames of low-resolution test data (resolution $40 \times 24 \times 32$). Each frame of data is shifted and rotated by a random 3D rigid body transformation, after which it is downsampled using block averaging. Lastly, Gaussian white noise is added. Again, we run the problem for 1%, 2%, and 3% added noise per data frame. The resulting optimization problem has 2,212,608 unknowns, $\mathbf{x} \in \mathbb{R}^{2,211,840}$ for the image and $\mathbf{w} \in \mathbb{R}^{768}$ for the motion parameters. The data has dimension 5,898,240. The formulation of the problem is identical to that of the two dimensional super-resolution problem with appropriate corrections for the change in dimension. The imaging operator \mathbf{K} for the 3D example is block diagonal with 128 identical down-sampling matrices $\mathbf{K}_j \in \mathbb{R}^{12,720 \times 2,211,840}$ that relate the high-resolution image to the low-resolution data by block averaging.

The initial guess for the three dimensional problem is generated using the same strategy as in the two dimensional case: we register all frames onto the first frame (thus solving 127 rigid registration problems.) This gives an initial guess for the motion with approximately 3% relative error. Using this initial guess, we then solve a linear least-squares problem to obtain an initial guess for the image with a relative error around 13%. We note that this is a poorer initial guess, when compared to

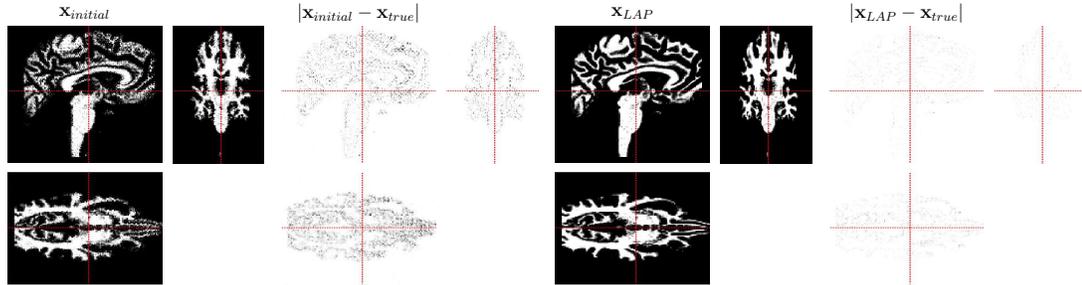


Figure 4.3: This figure shows two dimensional cross-sections of the reconstructed images and image errors for the 3D super-resolution problem with 2% noise. From left to right, the blocks show the initial guess $\mathbf{x}_{initial}$, the absolute value of the error for the initial guess, the solution \mathbf{x}_{LAP} for $LAP + \nabla_h$, and its absolute error. Each cross section was taken by fixing one dimension along its midpoint of the domain with the dotted red lines indicating the cross-sections' relations to the others.

the true solution, than the one obtained for the 2D super-resolution example. This may impact the quality of the solution obtainable for examples with large amounts of noise.

For this example, we test LAP, VarPro, and BCD using only the discrete gradient regularizer due to its better performance in the 2D example. Again, we fix $\alpha = 0.01$ for comparison, noting that this may not be ideal for all three methods. For LAP and BCD, we implement bound constraints on the image variables restricting the solution element-wise to the range $[0, 1]$. These constraints are not applied to VarPro. Also identical to the two dimensional case, all three methods require solving two linear systems, one larger corresponding to the dimension of the image and one smaller in the dimension of the motion. For LAP and BCD, the larger system is solved using LSQR with a tolerance of 10^{-2} while for VarPro we run LSQR for a fixed number of 50 iterations in order to achieve the required accuracy. For all three methods, the reduced system in the motion is solved using Cholesky factorization on the normal equations. As in the 2D case, we use matrix-vector products by \mathbf{J}_x as a metric to compare the computational cost of the three methods.

The resulting images and relative error plots for the images and motion param-

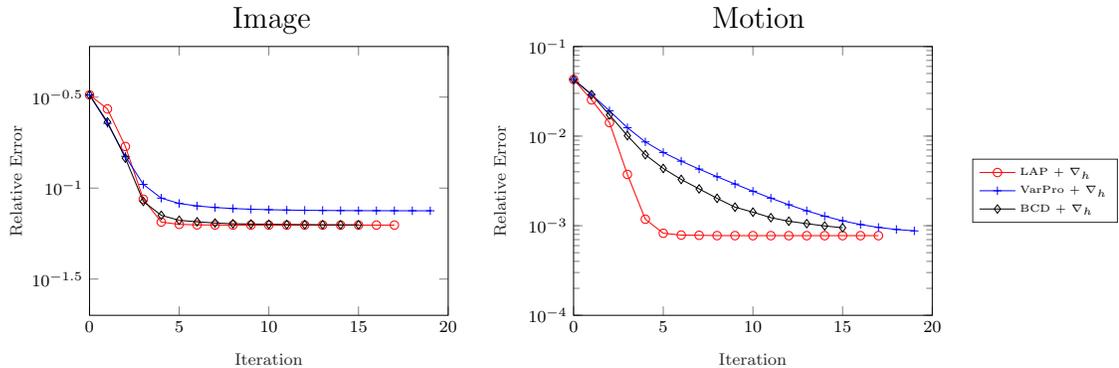


Figure 4.4: This figure plots the relative errors for both the reconstructed image and the motion parameters for the 3D super-resolution problem for 2% added noise. LAP succeeds in capturing the correct motion parameters in fewer iterations than VarPro and BCD and in recovering images of comparable quality.

eters for the problem with 2% noise can be found in Figs. 4.3 and 4.4, respectively. Like the 2D super-resolution example, LAP converges faster to the motion parameters during the early iterations of the optimization than BCD and VarPro and succeeds in reaching lower relative errors for both the recovered image and motion parameters. Again, VarPro’s iterations are far more expensive in terms of matrix-vector multiplications and CPU time as seen in Table 4.2, and the lack of bound constraints results in poorer solution images than LAP and BCD in terms of the relative error. BCD performs similarly with LAP in terms of the the reconstructed image, but it does less well at recovering the motion parameters and is slightly more expensive in terms of CPU time due to a higher number of function calls.

4.3 Motion Correction for Magnetic Resonance Imaging (MRI)

The last separable least-squares problem we look at to compare the methods is a two-dimensional MRI motion correction problem. The goal in this MRI application is to reconstruct a complex-valued MRI image from its Fourier coefficients that are

| | | | | | | |
|----------|---------------------|-------------|------------------------|------------------------|-------------|---------------|
| 1% Noise | | Iter. | Rel. Err. \mathbf{x} | Rel. Err. \mathbf{w} | MatVecs. | Time(s) |
| | LAP + ∇_h | 14.1 | 6.23e-2 | 8.65e-4 | 22.1 | 2.30e3 |
| | VarPro + ∇_h | 16.4 | 7.21e-2 | 9.55e-4 | 1.79e3 | 1.23e4 |
| | BCD + ∇_h | 11.3 | 6.25e-2 | 1.40e-3 | 25.7 | 4.26e3 |
| 2% Noise | | Iter. | Rel. Err. \mathbf{x} | Rel. Err. \mathbf{w} | MatVecs. | Time(s) |
| | LAP + ∇_h | 15.3 | 6.35e-2 | 9.27e-4 | 21.0 | 2.15e3 |
| | VarPro + ∇_h | 18.0 | 7.57e-2 | 1.00e-3 | 1.95e3 | 1.22e4 |
| | BCD + ∇_h | 12.2 | 6.37e-2 | 1.59e-3 | 26.2 | 3.75e3 |
| 3% Noise | | Iter. | Rel. Err. \mathbf{x} | Rel. Err. \mathbf{w} | MatVecs. | Time(s) |
| | LAP + ∇_h | 16.2 | 6.48e-2 | 8.97e-4 | 20.3 | 2.94e3 |
| | VarPro + ∇_h | 17.2 | 8.08e-2 | 9.52e-4 | 1.88e3 | 1.07e4 |
| | BCD + ∇_h | 11.7 | 6.50e-2 | 1.53e-3 | 25.2 | 4.21e3 |

Table 4.2: This table presents data for the solution of the 3D super-resolution for multiple values of added Gaussian noise. The columns from left to right give the stopping iteration, relative error of the solution image, relative error of the solution motion, number of matrix-vector multiplies during optimization, and time in seconds using `tic` and `toc` in `MATLAB`. All values are averages taken from 10 separate problems with different motion parameters, initial guesses, and noise realizations. The best value for each column is bold-faced.

acquired block-wise in a sequence of measurements. Since the measurement process typically requires several seconds or minutes, the object being imaged often moves substantially. Motion renders the Fourier samples inconsistent and — without correction — results in artifacts and blurring in the reconstructed MRI image. To correct for this, one can instead view the collected MRI data as a set of distinct, complex-valued Fourier samplings, each measuring some portion of the Fourier domain and subject to some unknown motion parameters. The problem of recovering the unknown motion parameters for each Fourier sampling and combining them to obtain a single motion-corrected MRI image fits into the coupled imaging framework presented in this chapter.

The forward model for this problem was presented by Batchelor *et al.* [1]. In their formulation, the imaging operator \mathbf{K} in (4.2) is again block diagonal with diagonal blocks \mathbf{K}_j for $j = 1, 2, \dots, N$ given by

$$\mathbf{K}_j = \mathbf{A}_j \mathcal{F} \mathbf{C}.$$

Here, \mathbf{C} is a complex-valued block rectangular matrix with diagonal blocks containing the given coil sensitivities of the MRI machine, \mathcal{F} is a complex-valued block diagonal matrix where each block is a two-dimensional Fourier transform (2D FFT), and \mathbf{A}_k is a real-valued block diagonal matrix with rectangular blocks containing selected rows of the identity corresponding to the Fourier sampling pattern (encoding scheme) for the k th data observation. We note here that the choice of sampling pattern and number of samplings N impacts the resulting problem significantly [11]. For 2D problems, common sampling patterns are Cartesian sequential, radial sequential Cartesian parallel 1D, Cartesian parallel 2D, and random. Image representations of these sampling patterns for a problem with four samples can be seen in Fig. 4.5. Cordero *et al.* noted that parallel samplings performed better than sequential samplings, with Cartesian parallel 2D outperforming both Cartesian parallel 1D and random samplings for very motion-affected data [11].

The superior performance of the parallel and random sampling patterns is likely due the fact that the Cartesian parallel 2D encoding scheme ensures that every sample has both high and low-frequency Fourier data, whereas the sequential samplings do not. We illustrate this with a small example where we plot the objective functions for a single sampling \mathbf{A}_j over a grid for a reduced size motion correction problem of the form (4.2) using a single Fourier sample. First, we fix the number of samples, $N = 16$, and compute a single sampling for each of the sampling strategies listed above. These samples correspond to 1/16 of Fourier space, and we can choose $j = 1, \dots, 16$ corresponding to any of the samples. We use the chosen sampling to create some ground truth data from a known image with no motion, i.e., $\mathbf{w} = \mathbf{0}$ using the forward operator $\mathbf{d}_j = \mathbf{A}_j \mathcal{FCT}(\mathbf{0})\mathbf{x} + \eta$ where \mathbf{A}_j corresponds to our sampling of choice. We add 1% noise. We then test the effect of motion for a given sampling by evaluating the objective function

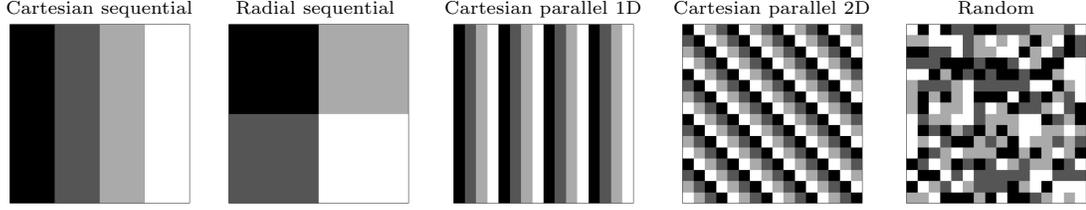


Figure 4.5: This figure illustrates five common sampling patterns for the MRI motion correction problem: Cartesian sequential, radial sequential, Cartesian parallel 1D, Cartesian parallel 2D, and random. The pictures are constructed for a 16×16 Fourier domain with 4 samplings illustrated by the 4 different color shades.

$$\Phi(\mathbf{x}, \mathbf{w}) = \frac{1}{2} \|\mathbf{A}_j \mathcal{FCT}(\mathbf{w}_{var}) \mathbf{x} - \mathbf{d}_j\|_2^2$$

on a grid where the motion \mathbf{w}_{var} is incrementally subjected to a rotation $\theta \in [-\pi/4, \pi/4]$ and shift along the line $b_1 = -b_2$ where $b_1 \in [-1/4, 1/4]$. By plotting the surface of the objective function on a grid corresponding to these motion parameters for each sampling strategy for different samplings \mathbf{A}_j , we can observe how the sampling strategy impacts the difficulty of the optimization. Fig. 4.6 shows the surface plots of the objective function for all four sampling strategies for $j = 1$ and 5. For $j = 1$, all sampling strategies produce relatively smooth objective functions. This is due to the fact that this sampling contains large amounts of low-frequency Fourier data for all four strategies. However for $j = 5$, the samplings for Cartesian sequential, radial sequential, and Cartesian 1D parallel contain less low-frequency information, and the resulting objective functions are highly oscillatory. This increases the difficulty of the optimization. In contrast, the objective functions for the Cartesian 2D parallel and random samplings are relatively smooth for all samplings $j = 1, \dots, N$, making optimization comparatively easier. This makes these methods preferable in the presence of large amounts of motion in the data samplings, and corroborates the results of Cordero *et al.* Following this, we use a Cartesian parallel 2D sampling strategy for the numerical results in this dissertation.

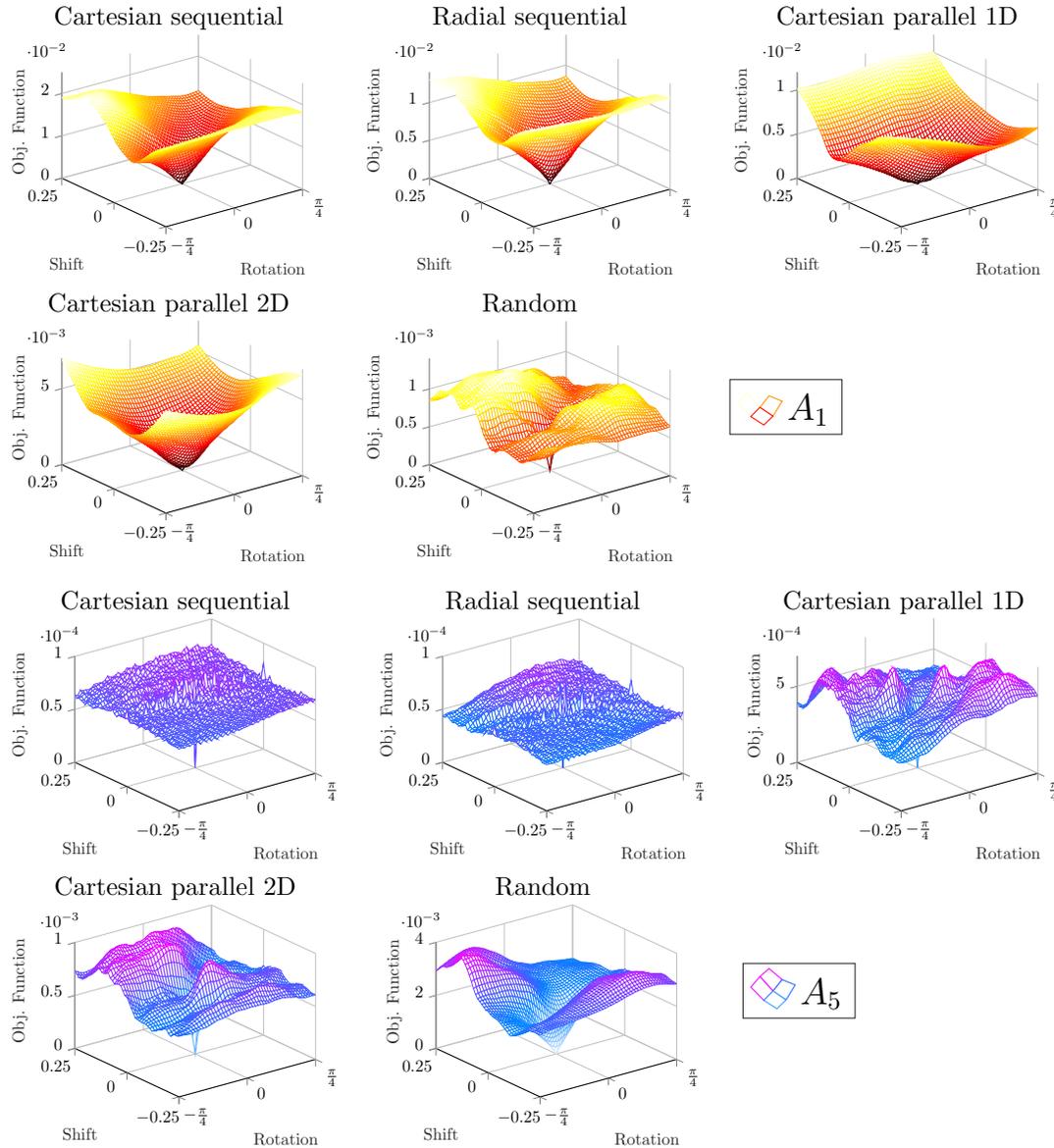


Figure 4.6: This figure shows how the choice of sampling pattern can affect the shape of the objective function. The top two rows (*red*) plot the objective function of sample \mathbf{A}_1 for all five sampling strategies under rotations $\theta \in [-\pi/4, \pi/4]$ and shifts along the line $b_1 = -b_2$ where $b_1 \in [-1/4, 1/4]$. The bottom two rows (*blue*) plot the objective function for sample \mathbf{A}_5 for the same motion parameters. Notice how the lack of low frequency information for some sampling strategies leads to highly oscillatory behavior for the \mathbf{A}_5 objective function.

As with the other examples, the imaging operator \mathbf{K} is multiplied on the right by the block rectangular matrix \mathbf{T} with $\mathbf{T}(\mathbf{y}(\mathbf{w}_k))$ blocks modeling the motion parameters of each Fourier sampling. We note that the cost of matrix-vector multiplications by this imaging operator is dominated by the 2D FFTs in block \mathcal{F} . For our problem, we use 32 receiver coils per sampling, so the resulting cost of a single matrix-vector multiplication will require $32N$ 2D FFTs of size 128×128 where N is the number of Fourier samplings in the data set. Additionally, we note that the presence of these FFT matrices prevents us from explicitly storing the matrix and necessitates passing it as a function call for all of the methods. This also applies to the Jacobian with respect to the image, \mathbf{J}_x . However, because it is relatively small in size, \mathbf{J}_w can still be computed and stored explicitly.

We use the data set provided in the `alignedSENSE` package [11] to set up an MRI motion correction with a known true image and known motion parameters. This data set includes a ground truth complex-valued MRI image and the coil sensitivities for 32 coils. To this end, we generate noisy data by using the forward problem (4.1). The ground truth image with resolution 128×128 is rotated and shifted by a random 2D rigid body transformation. The motion affected image is then observed on 32 sensors with known coil sensitivities. Each of these 32 observations is then sampled in Fourier space. For our problem, each sampling corresponds to $1/16$ of the Fourier domain, meaning that $N = 16$ samplings (each with unknown motion parameters) are needed to have a full sampling of the whole space. To sample, we use the Cartesian parallel 2D sampling pattern shown above. The resulting data has dimension $\frac{128 \times 128 \times 32}{16} \times 16 = 524,288$, and the resulting optimization problem has 16,432 unknowns corresponding to $\mathbf{x} \in \mathbb{C}^{16,384}$ parameters for the image and $\mathbf{w} \in \mathbb{R}^{48}$ for the motion. Gaussian noise is then added to each data sampling. We run the problem for 5%, 10%, and 15% added noise.

As with the super-resolution problem, we compare LAP with VarPro and BCD.

The MRI motion correction problem differs from the super-resolution examples in that the image data and resulting image are complex-valued while the motion parameters are real-valued. This requires consideration during the optimization when taking gradients or projecting between the space of the real and complex-valued variables. For regularization, we test all three methods using the discrete gradient regularizer for a fixed $\alpha = 0.01$. For LAP and BCD, we test using hybrid regularization with HyBR, and we run VarPro with the identity for a regularizer and $\alpha = 0.01$. As with the super-resolution problems, the least-squares problem in LAP and the least-squares problem for the imaging step for BCD are solved using LSQR with a tolerance of 10^{-2} for all choices of regularization. For VarPro, we use LSQR with a tolerance of 10^{-8} or a maximum of 100 iterations to maintain accuracy in the gradient. As with previous examples, Cholesky factorization on the normal equations is used for the lower-dimensional solves with \mathbf{J}_w . No bound constraints were applied to any of the methods for this example because element-wise bound constraints on the real or imaginary parts of the complex-valued image variables will impact the angle, i.e. phase, of the complex-valued solution, which is undesirable.

For an initial guess for the motion parameters, we start with the zero vector, $\mathbf{w} = \mathbf{0}$ (corresponding to a relative error of 100% for the motion). Using this initialization, we solve a linear least-squares problem to get an initial guess for the image. For 10% added Gaussian noise in the data, the initial guess for the image has a relative error of around 35%. This is significantly higher than for the super-resolution examples and has an impact on the convergence of the Gauss–Newton optimization. We show the initial guess in Fig. 4.7.

LAP, VarPro, and BCD all manage to recover fairly accurate reconstructions of both the image and motion parameters for quite large values of noise using either HyBR or the identity as a regularizer; see Figs. 4.7 and 4.8. This is likely due to the fact that the problem is not severely ill-posed and is highly over determined (32 sensor readings

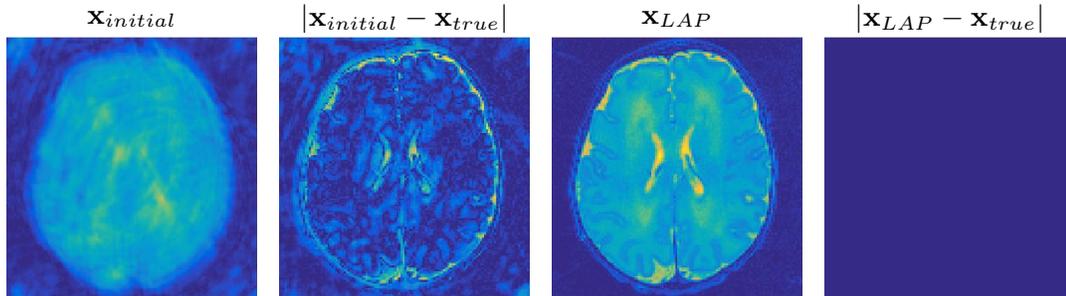


Figure 4.7: From left to right, the images are the initial guess, the modulus of the error of the initial guess, the reconstructed image using LAP + HyBR, and image errors for the reconstructed image. All results are for the MRI motion correction problem with 10% noise. Note that the MRI images and difference images are the modulus of the complex-valued images recovered.

for each point in Fourier space.) For this example, the hybrid regularization approach for LAP and BCD produces the best results, with LAP requiring considerably fewer iterations. We remark that the best regularization for this problem differs from the super-resolution problems, and shows the importance of the flexibility that LAP offers for regularizing the resulting image. The comparative speed of LAP is observable for the relative error plots for the problem with 10% noise and further evidenced in Table 4.3 for all noise levels over 10 separate realizations of the problem. For the gradient based regularizer, all three methods do not recover the motion parameters accurately. We also note that the number of iterations and their cost is an important consideration for this problem. Because of the distance of the initial guess from the solution, this problem requires more iterations to converge than the super-resolution examples. This follows from the local quadratic convergence properties of Gauss–Newton for LAP. This is also seen with BCD, which uses Gauss–Newton for the block optimization on both sets of variables, and VarPro, which uses Gauss–Newton on the reduced nonlinear problem in \boldsymbol{w} . Within each iteration, the high number of 2D FFTs required for a single matrix-vector multiplication makes multiplications by the Jacobian \boldsymbol{J}_x expensive. Table 4.3 shows that LAP outperforms VarPro and BCD

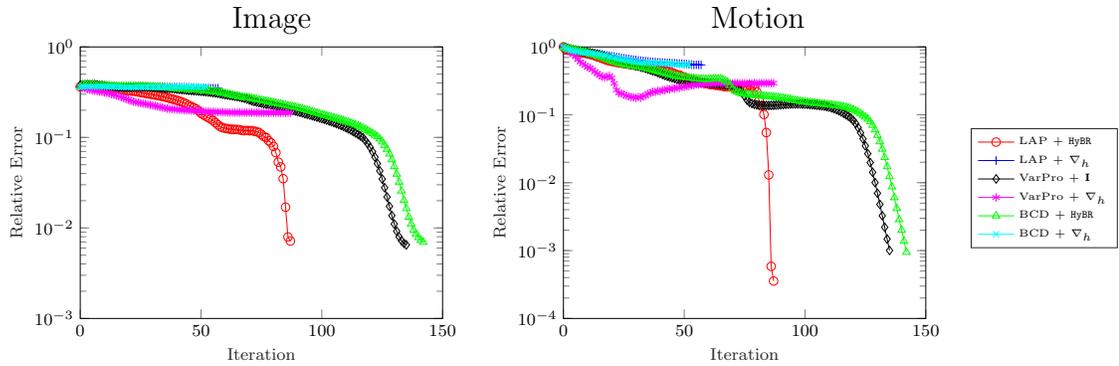


Figure 4.8: This figure shows the relative errors for both the reconstructed image and the motion parameters for the MRI motion correction problem for three levels of noise. LAP with hybrid regularization achieves better reconstructions of the image and motion parameters for fewer iterations than either VarPro and BCD.

for both choices of regularizer by requiring fewer, cheaper iterations in terms of both time and matrix-vector multiplications. The difference in cost is most dramatic when compared with VarPro again due to the large number of FFTs required for a single matrix-vector multiplication and the large number of such multiplications required within each VarPro function call to solve the linear problem associated with the image. For BCD and LAP, the number of matrix-vector multiplications is similar, but BCD requires more iterations for convergence. Overall, we see that LAP outperforms BCD and VarPro for this problem, and that it provides better reconstructions of both the image and motion in fewer, cheaper iterations.

4.4 Discussion

We end the chapter with a brief discussion summarizing the results for both the super-resolution and MRI motion correction examples. For all three examples tested, LAP outperforms BCD and VarPro in terms of solution quality, computational cost, and time to solution. This suggests the method’s utility for separable least-squares problems, especially ones where the subproblem associated with the linear parameters

| | | | | | | |
|-----------|-----------------------|-------------|------------------------|------------------------|---------------|---------------|
| 5% Noise | | Iter. | Rel. Err. \mathbf{x} | Rel. Err. \mathbf{w} | MatVecs. | Time(s) |
| | LAP + HyBR | 76.6 | 3.55e-3 | 1.59e-4 | 3.18e2 | 5.86e2 |
| | LAP + ∇_h | 76.2 | 3.74e-3 | 1.67e-4 | 3.62e2 | 4.32e2 |
| | VarPro + \mathbf{I} | 116.0 | 7.93e-2 | 4.32e-2 | 2.19e4 | 1.20e4 |
| | VarPro + ∇_h | 115.9 | 7.92e-2 | 4.33e-2 | 2.19e4 | 1.52e4 |
| | BCD + HyBR | 128.6 | 4.35e-2 | 2.27e-2 | 4.03e2 | 1.01e3 |
| | BCD + ∇_h | 116.6 | 7.83e-2 | 4.20e-2 | 4.27e2 | 8.59e2 |
| 10% Noise | | Iter. | Rel. Err. \mathbf{x} | Rel. Err. \mathbf{w} | MatVecs. | Time(s) |
| | LAP + HyBR | 76.8 | 6.56e-3 | 3.25e-4 | 3.11e2 | 6.52e2 |
| | LAP + ∇_h | 76.1 | 6.88e-3 | 4.15e-4 | 3.55e2 | 5.19e2 |
| | VarPro + \mathbf{I} | 116.0 | 8.08e-2 | 4.33e-2 | 2.19e4 | 1.12e4 |
| | VarPro + ∇_h | 116.0 | 8.08e-2 | 4.32e-2 | 2.19e4 | 1.27e4 |
| | BCD + HyBR | 128.2 | 4.53e-2 | 2.21e-2 | 3.89e2 | 1.24e3 |
| | BCD + ∇_h | 116.0 | 8.02e-2 | 4.21e-2 | 4.12e2 | 1.05e3 |
| 15% Noise | | Iter. | Rel. Err. \mathbf{x} | Rel. Err. \mathbf{w} | MatVecs. | Time(s) |
| | LAP + HyBR | 77.3 | 9.49e-3 | 4.69e-2 | 3.07e2 | 5.02e2 |
| | LAP + ∇_h | 75.0 | 1.61e-2 | 2.92e-2 | 3.40e2 | 3.61e2 |
| | VarPro + \mathbf{I} | 115.8 | 8.29e-2 | 4.36e-2 | 2.18e4 | 9.98e3 |
| | VarPro + ∇_h | 115.7 | 8.28e-2 | 4.35e-2 | 2.18e4 | 1.27e4 |
| | BCD + HyBR | 127.0 | 4.80e-2 | 2.21e-2 | 3.47e2 | 1.06e3 |
| | BCD + ∇_h | 129.0 | 8.20e-2 | 4.17e-2 | 3.99e2 | 8.17e2 |

Table 4.3: This table shows the results of LAP, VarPro, and BCD for solving the MRI motion correction example for multiple regularizers and varying levels of added noise. Averaged over 10 realizations of the problem, the columns are stopping iteration, relative error of the solution image, relative error of the solution motion, number of matrix-vector multiplies during optimization, and time in seconds using `tic` and `toc` in `MATLAB`. LAP outperforms the other methods in terms of solution quality, computational cost, and CPU time. The best value for each column is bold-faced.

is ill-posed, requires regularization, or benefits from the implementation of bound constraints. The examples in this chapter showed this for both real and complex-valued imaging problems.

One point of interest is that the flexibility in terms of regularization and implementing bound constraints is made possible by LAP’s strategy of linearizing before projecting the problem. In general, the positives of this flexibility may be diminished if the linearization is a poor approximation of the nonlinear objective function. We do not see this for the motion correction problems in this chapter, possibly because separable least-squares problems are linear in the block of \mathbf{x} variables. Thus, the linearization step of LAP loses less information in the linearization step than if the problem was nonlinear in both sets of variables. This likely increases the performance of the method for this class of problem. We compare this with the results in Ch. 5, where the \mathbf{x} block of variables is nonlinear for the problem in question.

Chapter 5

Locally Rigid Image Registration

This chapter explores the utility of LAP for image registration subject to rigidity constraints on some portion of image domain, i.e., locally rigid registration. The locally rigid registration problem is nonlinear in two blocks of variables with the first, larger block corresponding to a nonlinear transformation of the image on the unconstrained portions of the image domain and the second, smaller block corresponding to a rigid transformation on the constrained regions of the domain. This problem fits within the LAP framework presented in Ch. 3. We show the utility and limitations of LAP for this highly nonlinear problem compared to a fully coupled approach by comparing the convergence of the two approaches.

The chapter is organized as follows. We begin by posing the locally rigid registration problem. In the continuous setting, we introduce the Lagrangian framework as a means for tracking the rigidly constrained regions of the image domain and a hyperelastic regularizer that ensures the smoothness and invertibility of the registration map while allowing highly nonlinear deformations. This is followed by a section detailing the numerical implementation for the discretized problem. Finally, we run numerical experiments to examine the utility of LAP as a solver for the resulting coupled problem that is nonlinear in both the block of variables corresponding to the

rigidly constrained regions and also the block of unconstrained, non-parametric variables. We show results for locally rigid registration on a 2D MRI image of a knee and a 3D example using blocks, and discuss potential and limitations of LAP for highly nonlinear problems of this form.

5.1 Locally Rigid Registration Problem

This section formulates the locally rigid registration problem. We introduce the problem in the continuous setting, following the framework given by [28]. The problem in d -dimensions (for our purposes $d = 2$ or 3) can be stated as follows: given a template image $T : \mathbb{R}^d \rightarrow \mathbb{R}$ and a reference image $R : \mathbb{R}^d \rightarrow \mathbb{R}$ that are continuously differentiable and compactly supported on some reference domain $\Omega \in \mathbb{R}^d$, the registration problem aims to find a transformation y in the space of transformations \mathcal{V} such that transformed template image $T(y)$ is *close* or *similar* to the reference image on the domain Ω . As is common in image registration, we limit ourselves to maps y that are diffeomorphic, i.e., smooth and invertible, to prevent folding or cracks in the deformed template image [18]. For our problem, we also impose that the map y is subject to constraints on some subset of the domain, $\Sigma \subset \Omega$. For general constraints, this can be written as

$$\mathcal{C}(y)(x) = 0 \text{ for all } x \in \Sigma, \quad (5.1)$$

where the constraint on y is active whenever we transform any point $x \in \Sigma$ within the constrained region. For this chapter, we consider the specific case of constraints imposing rigid transformations on y in the region Σ . Extending the notation, rigid constraints for N distinct regions can be written as

$$\begin{aligned} \exists \boldsymbol{\theta}_k, \mathbf{b}_k \text{ such that } \mathcal{C}(y)(x) = y(x) - (\mathbf{Q}(\boldsymbol{\theta}_k)x + \mathbf{b}_k) = 0 \\ \text{for all } x \in \Sigma_k \text{ and } k = 1, \dots, N, \end{aligned}$$

or equivalently,

$$y(x) = \mathbf{Q}(\boldsymbol{\theta}_k)x + \mathbf{b}_k \text{ for all } x \in \Sigma_k \text{ and } k = 1, \dots, N.$$

Here, $\mathbf{Q}(\boldsymbol{\theta}_k)$ is a rotation matrix subject to angle(s) $\boldsymbol{\theta}_k$ and \mathbf{b}_k is the shift for the k th constrained region. For the two dimensional case, $\boldsymbol{\theta}_k \in \mathbb{R}$ is scalar and $\mathbf{b}_k \in \mathbb{R}^2$ for a total of three parameters per region, whereas for three dimensions, we have $\boldsymbol{\theta}_k \in \mathbb{R}^3$ and $\mathbf{b}_k \in \mathbb{R}^3$ for six parameters per region. In the optimization and to maintain consistency with Ch. 4, we concatenate the parameters associated with a given rigid transformation in the vector

$$\mathbf{w}_k = \begin{bmatrix} \boldsymbol{\theta}_k \\ \mathbf{b}_k \end{bmatrix} \in \mathbb{R}^{(3,6)}$$

for the k th constrained region. We use \mathbf{w} to refer to the concatenation of all \mathbf{w}_k into a single vector containing the rigid motion parameters for all constrained regions. While introducing the problem, it helps to have condensed notation for the rigidity constraints, so letting $c(\mathbf{w}_k, x) = \mathbf{Q}(\boldsymbol{\theta}_k)x + \mathbf{b}_k$, we condense the notation for local rigidity constraints to the following expression,

$$y(x) = c(\mathbf{w}_k, x) \text{ for all } x \in \Sigma_k \text{ and } k = 1, \dots, N. \quad (5.2)$$

We use this notation for the duration of the introduction for the problem. For more information on the rigid transformations, we refer the reader to appendix A.2 or [47].

Rigid constraints are useful when modeling the motion of regions in the template image that are known to not change shape. One example of this is bones in medical images, which we present in the numerical experiments for this chapter. Typically, such constrained regions are identified prior to registration and can be obtained by segmenting the template image into a number of rigid and non-rigid regions.

To solve the constrained registration problem, we pose it as an optimization problem where the goal is to find the transformation y that minimizes a functional comprised of two terms. The first term is a distance functional $\mathcal{D}(y; T, R)$ that measures the similarity of the transformed template image to the reference image. The second term is regularizer $\mathcal{S}(y)$ weighted by a regularization parameter $\alpha > 0$ that forces the solution y to be *reasonable*, i.e., smooth and invertible. This is necessary as image registration problems are known to be ill-posed [31, 14, 18, 29]. For this work, we consider the sum of squared distances (SSD) for our distance measure and consider a hyperelastic regularizer introduced by [5]. To further introduce both, we now split the conversation into two sections. The first section provides an overview of the Eulerian and Lagrangian way to model transformations. We introduce and compare the frameworks, motivating our choice of the Lagrangian model for the constrained image registration problem. We then introduce a hyperelastic regularizer on the transformation y that allows for modeling highly nonlinear deformations while enforcing smoothness and invertibility.

5.1.1 Eulerian and Lagrangian Frameworks

In this section, we review the Eulerian and Lagrangian transformation models as two options for posing the optimization problem for locally rigid registration problem. We begin by describing the difference between the two approaches for a single point $x \in \Omega$ with emphasis on the way each framework handles constraints on the subdomain $\Sigma \in \Omega$. We then detail the effects of each approach on the SSD measure and constraints for the optimization problem described above. We use this as motivation for using the Lagrangian framework for our problems.

The Eulerian and Lagrangian models represent two ways of describing the same transformation. The difference between the two approaches is the difference between looking backward and looking forward. Consider a transformation y that maps a

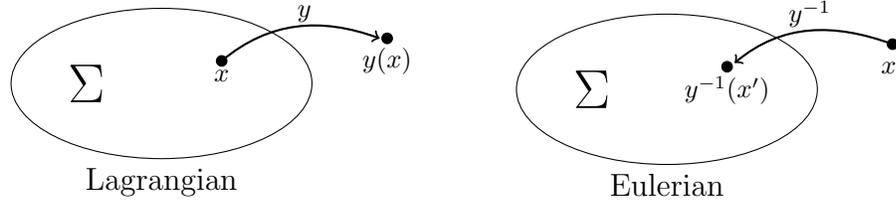


Figure 5.1: The Lagrangian approach (*left*) maps a point $x \in \Sigma$ to a point $x' = y(x)$ outside the constrained region. In the Eulerian approach (*right*), a point x' is mapped from a point in the constrained region $y^{-1}(x') = x$.

point $x \in \Omega$ to the point $x' = y(x)$, and let us pair that point with a function value (image intensity) $T(x)$. The Lagrangian framework starts at this point $(x, T(x))$ in the original image and pushes forward to the transformed pair $(x', T(x)) = (y(x), T(x))$ in the transformed image. This corresponds to transporting the fixed image intensity $T(x)$ forward to a new location $x' \in \Omega$. In contrast, the Eulerian framework considers the endpoint $(x', T(x'))$ for $x' \in \Omega$ and looks at the pair $(x', T(x)) = (x', T(y^{-1}(x')))$ from which it was mapped. This is equivalent to fixing the point $x' \in \Omega$ in the transformed image and looking backwards to find the image intensity at the point from which x' arrived, i.e., the image intensity $T(y^{-1}(x'))$ arriving from the point $x = y^{-1}(x')$. An illustration of the two approaches can be seen in Fig. 5.1.

The choice of the Eulerian or Lagrangian transformation model impacts the formulation of both the distance measure and the constraints for the locally rigid registration. Typically, constraints are introduced in the Lagrangian framework in the context of the forward transformation y , and this is the case in (5.1) and (5.2). However, in the Eulerian framework, the general constraint in (5.1) must be reformulated in terms of the inverse transformation, y^{-1} , as

$$\mathcal{C}_E(y^{-1})(x') = 0 \text{ for all } y^{-1}(x') \in \Sigma. \quad (5.3)$$

A significant issue with this formulation is that the constraints in the Eulerian frame-

work are dependent on the constrained region and the inverse transformation, $\Sigma(y^{-1})$. This poses several problems and is difficult compared to the Lagrangian approach where the constraints rely solely on the constrained region Σ . One strategy to overcome the problems posed by constraints in the Eulerian framework is tracing the indicator function [46]. However, this results in non-differentiable constraints. Intuitively, one can see this by considering two points x' and $x' + \epsilon$ such that $y^{-1}(x') \in \Sigma$ and $y^{-1}(x' + \epsilon) \notin \Sigma$ for an arbitrarily small perturbation ϵ . This implies that the constraint is non-differentiable and makes the Eulerian approach less attractive when dealing with constrained registration problems. A more formal explanation is provided by rewriting the constraint in (5.3) as

$$\chi_{y^{-1}(\Sigma)}(x')\mathcal{C}(y^{-1})(x') = 0 \text{ for all } x' \in \Omega,$$

where $\chi_{y^{-1}(\Sigma)}(x')$ is the characteristic function of the constrained region Σ under the map y^{-1} . Here the non-differentiability of the constraint follows directly from the non-differentiability of the characteristic function.

The Eulerian framework is commonly preferred for unconstrained problems, see Ch. 4 and [47]. One reason for this is a straightforward SSD measure. In the Eulerian framework, we compare the similarity of the registered images by comparing a fixed point in the reference image ($x', R(x')$) to a point in the template image ($x', T(y^{-1}(x'))$) for all $x' \in \Omega$. Thus, the SSD measure is given by

$$\mathcal{D}_E(y^{-1}; T, R) = \frac{1}{2} \int_{\Omega} (T(y^{-1}(x')) - R(x'))^2 dx',$$

where the measure depends only on the fixed domain Ω and the map y^{-1} . Combining the SSD measure with the reformulation of the constraints, the constrained image

registration problem in the Eulerian framework is

$$\min_{y^{-1}} \{ \mathcal{D}_E(y^{-1}; T, R) + \alpha \mathcal{S}(y^{-1}) \}$$

subject to $\mathcal{C}_E(y^{-1})(x') = 0$ for all $y^{-1}(x') \in \Sigma$,

where we solve the minimization problem for the inverse transform y^{-1} . This approach has the positive of a simple distance measure offset by the difficulty in tracking the constraints.

In contrast, constraints in the Lagrangian framework are differentiable and depend solely on the constrained region as seen in (5.1). The tradeoff is that the Lagrangian approach results in a more involved SSD measure. The Lagrangian approach considers the forward transform y . Thus, the point $(x, T(x))$ in the template image is mapped to $(x', T(x)) = (x', T(y^{-1}(x')))$. The SSD must then compare this mapped point $(x', T(y^{-1}(x')))$ in the template image to the fixed point $(x', R(x'))$ in the reference image for every $x' \in y(\Omega)$. The resulting Lagrangian SSD measure is

$$\frac{1}{2} \int_{y(\Omega)} (T(y^{-1}(x')) - R(x'))^2 dx'.$$

This presents two issues. First, the domain of integration $y(\Omega)$ for this measure is subject to changes in the map, meaning that the distance measure reduces or increases if the domain shrinks or expands. This is undesirable but can be fixed by introducing an averaged distance given by

$$\frac{1}{2} \frac{1}{|y(\Omega)|} \int_{y(\Omega)} (T(y^{-1}(x')) - R(x'))^2 dx',$$

where the area or volume of the domain of integration is given by

$$|y(\Omega)| = \int_{y(\Omega)} dx'.$$

The second problem is that this distance measure requires knowledge of both y and its inverse. To avoid this, we make a change of variables by $x = y^{-1}(x')$, which by the transformation rule gives

$$\int_{y(\Omega)} (T(y^{-1}(x')) - R(x'))^2 dx' = \int_{\Omega} (T(x) - R(y(x)))^2 |\det \nabla y(x)| dx$$

and

$$|y(\Omega)| = \int_{\Omega} |\det \nabla y(x)| dx.$$

Combining these two equations, we get a Lagrangian distance dependent only on the forward transform y given by

$$\mathcal{D}_L(y; T, R) = \frac{1}{2} \left(\int_{\Omega} |\det \nabla y(x)| dx \right)^{-1} \int_{\Omega} (T(x) - R(y(x)))^2 |\det \nabla y(x)| dx. \quad (5.4)$$

This distance measure is less simple than the Eulerian SSD due to the presence of additional terms depending on $|\det \nabla y(x)|$. This accounts for its lack of popularity for unconstrained regularization problems. In particular, the absolute value in $|\det \nabla y(x)|$ makes (5.4) non-differentiable when $\det \nabla y(x) = 0$. We avoid this through the introduction of appropriate regularization in the next section. For constrained problems however, dealing with this more complicated distance measure is rewarded with the easier, Lagrangian constraints and a smooth optimization problem. We opt for this framework in our work. The resulting registration problem with local rigidity constraints in the Lagrangian framework is given by

$$\begin{aligned} & \min_y \{ \mathcal{D}_L(y; T, R) + \alpha \mathcal{S}(y) \} \\ & \text{subject to } \mathcal{C}(y)(x) = 0 \text{ for all } x \in \Sigma. \end{aligned} \quad (5.5)$$

Here, it is unnecessary to track the transform of the constrained region as in the Eulerian case, and furthermore the constraints are differentiable provided the map y

is differentiable.

5.1.2 Hyperelastic Regularizer

Next, we focus on the regularizer term $\mathcal{S}(y)$. Image registration problems, including the locally rigid registration problem introduced in the previous section, are known to be ill-posed [31, 14, 18, 29]. As such, it is necessary to introduce a regularizer to ensure the solution map y is *reasonable* by some definition. Typically, these regularizers are based on the gradient of the transformation, ∇y , and a strain tensor based on the displacement, u , where $y(x) = x + u(x)$ [10]. Well known nonlinear regularizers of this form include elastic regularization [4, 10], curvature regularization [17], and hyperelastic regularization [67]. A major limitation of all these methods is that they result in finite regularization energies for $\det \nabla y = 0$, i.e., they do not prevent the solution map y from becoming nondiffeomorphic. Additionally, we recall that the Lagrangian SSD from the previous section is non-differentiable for $\det \nabla y = 0$. Thus, these regularizers are poorly suited to our problem.

We want a regularizer that limits our solution space to diffeomorphic (smooth and one-to-one) transformations, thus ensuring the differentiability of the Lagrangian distance function. We also want to allow for transformations that can model large nonlinear deformations. We opt for a hyperelastic regularizer satisfying both these criteria [5, 14]. This hyperelastic regularizer satisfies

$$\begin{aligned} \mathcal{S}(y) &\rightarrow \infty \quad \text{for} \quad \det \nabla y \rightarrow 0 \\ \mathcal{S}(y) &\geq c_1 \{ \|\nabla y\|^p + \|\text{cof } \nabla y\|^q + (\det \nabla y)^r \} + c_2 \end{aligned}$$

for constants $c_1 > 0$, $c_2 \in \mathbb{R}$, and $p, q, r > 1$. These criteria penalize expansion and shrinkage, respectively, and ensure that the regularizer energy tends to infinity for nondiffeomorphic transformations [10]. Moreover, the regularizer explicitly monitors the value of $\det \nabla y$ to ensure the feasibility of a solution transformation, unlike [67].

One important side effect of the lower bound on the regularization energy is that $\det \nabla y$ is bounded below away from 0. This guarantees that the $|\det \nabla y|$ term in the Lagrangian SSD remains differentiable in the space of permissible transformations.

The hyperelastic regularizer on the transformation y is defined for $d = 3$ as the sum of three terms:

$$\mathcal{S}(y) = \alpha_1 \mathcal{S}_{len}(y) + \alpha_2 \mathcal{S}_{area}(y) + \alpha_3 \mathcal{S}_{vol}(y), \quad (5.6)$$

where α_1 , α_2 , and $\alpha_3 > 0$ are regularization parameters and $\mathcal{S}_{len}(y)$, $\mathcal{S}_{area}(y)$, and $\mathcal{S}_{vol}(y)$ control changes the length, surface area, and volume of the solution transformation. The three terms are defined by

$$\begin{aligned} \mathcal{S}_{len}(y) &= \int_{\Omega} \phi_l(\nabla y) dx && \text{with } \phi_l(\mathbf{X}) = \|\mathbf{X} - \mathbf{I}_d\|_F^2 \\ \mathcal{S}_{area}(y) &= \int_{\Omega} \phi_a(\text{cof } \nabla y) dx && \text{with } \phi_a(\mathbf{X}) = (\|\mathbf{X}\|_F^2 - 3)^2 \\ \mathcal{S}_{vol}(y) &= \int_{\Omega} \phi_v(\det \nabla y) dx && \text{with } \phi_v(x) = ((x - 1)^2/x)^2, \end{aligned}$$

where the cofactor and determinant are given by

$$\text{cof } \nabla y = \begin{bmatrix} \partial_2 y_2 \partial_3 y_3 - \partial_3 y_2 \partial_2 y_3 & \partial_3 y_2 \partial_1 y_3 - \partial_1 y_2 \partial_3 y_3 & \partial_1 y_2 \partial_2 y_3 - \partial_2 y_2 \partial_1 y_3 \\ \partial_3 y_1 \partial_2 y_3 - \partial_2 y_1 \partial_3 y_3 & \partial_1 y_1 \partial_3 y_3 - \partial_3 y_1 \partial_1 y_3 & \partial_2 y_1 \partial_1 y_3 - \partial_1 y_1 \partial_2 y_3 \\ \partial_2 y_1 \partial_3 y_2 - \partial_3 y_1 \partial_2 y_2 & \partial_3 y_1 \partial_1 y_2 - \partial_1 y_1 \partial_3 y_2 & \partial_1 y_1 \partial_2 y_2 - \partial_2 y_1 \partial_1 y_2 \end{bmatrix} \quad (5.7)$$

$$\begin{aligned} \det \nabla y &= \partial_1 y_1 \partial_2 y_2 \partial_3 y_3 + \partial_2 y_1 \partial_3 y_2 \partial_1 y_3 + \partial_3 y_1 \partial_1 y_2 \partial_2 y_3 \\ &\quad - \partial_1 y_3 \partial_2 y_2 \partial_3 y_1 - \partial_2 y_3 \partial_3 y_2 \partial_1 y_1 - \partial_3 y_3 \partial_1 y_2 \partial_2 y_1. \end{aligned}$$

The length term penalizes changes in the length and angle of the transformation via a quadratic penalty for departure from the identity. The area term is based on the interpretation of the columns of the cofactor matrix as normal vectors to the transformed surfaces. The length of these normal vectors corresponds to the area of

the transformed surface, and the area term of the regularizer penalizes changes for lengths different to 1. Lastly, the volume term governs changes in the volume by penalizing changes in the determinant.

A solution transformation $y \in \mathcal{V}$ exists for the hyperelastic regularizer used on unconstrained registration problems with most practical distance measures where

$$\begin{aligned} \mathcal{V} = \{y \in W^{1,2}(\Omega, \mathbb{R}^3) : \text{cof } \nabla y \in L^4(\Omega, \mathbb{R}^{3 \times 3}), \det \nabla y \in L^2(\Omega, \mathbb{R}), \\ \det \nabla y > 0 \text{ a.e.}, |\int y(x) dx| < C_\Omega\}. \end{aligned} \quad (5.8)$$

Proofs for this result can be found in [5] but are not the main focus of this work, so we omit further details here. Instead, we turn our attention to the details necessary for the stable numerical implementation of the locally rigid registration problem using the Lagrangian distance measure and hyperelastic regularizer presented in this section.

5.1.3 Numerical Implementation

For implementation, we follow a discretize then optimize approach with a multilevel optimization strategy as in [47]. This approach starts by discretizing and solving the problem on a coarse level. Once a numerical minimizer is found, this solution is prolonged and used as a starting guess for the problem at a finer discretization. This is done over subsequent levels until a solution at the finest discretization level is reached. The optimization at coarser levels followed by refining helps to avoid local minima during optimization and is cheaper than fine level optimization for the nonparametric portion of the domain. For the optimization at each level, we use Gauss–Newton optimization using LAP for the step solve and a backtracking Armijo line search as described in Ch. 2 and 3. To enact this optimization approach, we now detail our discretization of the continuous problem, and how it fits within the LAP framework.

We begin by restating the Lagrangian formulation of the image registration prob-

lem subject to local rigidity constraints,

$$\begin{aligned} & \min_y \{ \mathcal{D}_L(y; T, R) + \alpha \mathcal{S}(y) \} \\ & \text{subject to } y(x) = c(\mathbf{w}_k, x) \text{ for all } x \in \Sigma_k \text{ and } k = 1, \dots, N. \end{aligned} \quad (5.9)$$

We note that the constraints can be eliminated by rewriting the problem as

$$\min_{x, \mathbf{w}_1, \dots, \mathbf{w}_N} \{ \Phi(y) = \mathcal{D}_L(y(x, \mathbf{w}_1, \dots, \mathbf{w}_N); T, R) + \alpha \mathcal{S}(y(x, \mathbf{w}_1, \dots, \mathbf{w}_N)) \}, \quad (5.10)$$

where the new transformation $y(x, \mathbf{w}_1, \dots, \mathbf{w}_N)$ combines the nonlinear transformations and rigid transformations. We define this new transformation by

$$y(x, \mathbf{w}_1, \dots, \mathbf{w}_N) = \begin{cases} x(u) & \text{if } u \notin \Sigma_k, \forall k = 1, \dots, N \\ c(\mathbf{w}_k, u) & \text{if } u \in \Sigma_k. \end{cases}$$

In practice, the transformation y is obtained by patching together the rigid transformations on the constrained regions and a nonparametric transformation on the remainder of the domain. We then optimize by minimizing the problem (5.10) for x and $\mathbf{w}_1, \dots, \mathbf{w}_N$. Note that the discretization of the nonparametric transformation x varies in size at each level of the optimization, while the rigid transformation has the same number of parameters at every iteration. Here, we also see the origin of the two distinct sets of variables, x corresponding to the nonparametric transformation and $\mathbf{w} = \{\mathbf{w}_1, \dots, \mathbf{w}_N\}$ for the rigid transformation, that make this a coupled problem suitable for LAP.

Having reformulated the problem, we now discuss discretization of the Lagrangian SSD, the hyperelastic regularizer, and their derivatives for an arbitrary transformation $y(x, \mathbf{w}_1, \dots, \mathbf{w}_N)$. To do this, we begin by noting that the parameterized transformation $y(x, \mathbf{w}_1, \dots, \mathbf{w}_N)$ is a specific instance of a general nonlinear trans-

formation y . Thus, we begin by presenting the discretization and derivatives for a general y and then extend that discretization to encompass the specific transformation $y(x, \mathbf{w}_1, \dots, \mathbf{w}_N)$ that parameterizes both the constrained and unconstrained regions appropriately. To simplify notation, we present the framework for the three dimensional case, with some notes on differences for the simpler, two dimensional case. We rely heavily on the work and notation from [56].

Discretization for an unconstrained transformation

To discretize the problem for a general, unconstrained y , we begin as in Ch. 4 by discretizing the continuous images on a regular, cell-centered grid with n cells on the domain Ω . Note that for the multilevel optimization approach we use for this problem, the number of cells in the regular grid is determined by the level of the optimization with coarser discretizations resulting in fewer cells and finer discretizations resulting in more. We assume the discrete reference and template images, \mathbf{T}, \mathbf{R} , for the problem are obtained by evaluating the continuous image functions T and R at the cell-centers of that grid for a given level of the optimization.

To model the nonlinear transformation y , we use piecewise linear, globally continuous finite elements on a tetrahedral mesh built on this discretized grid. To build the mesh for the three dimensional case, each rectangular solid cell is partitioned into 24 tetrahedra at the lowest level of the optimization. For the two dimensional case, this simplifies to each rectangular cell being separated into 4 triangles. This partitioning strategy is chosen over alternatives with 5, 6 tetrahedra or 2 triangles because it is symmetric and avoids directional bias that may affect the quality of the solution [55]. The penalty we pay is increased cost compared to the alternatives. This is particularly acute at fine discretization levels for three dimensional problems. For further details on mesh generation, we refer the reader to Burger *et al.* for details [5]. Note that this generation of the mesh on a regular grid only occurs at the lowest (coarsest)

level of the multilevel optimization. For higher levels, the mesh at the last iteration of the previous level is prolonged to create a finer mesh.

For the generated mesh on the regular cell grid, let $\{v_1, v_2, \dots, v_{n_v}\} \in \mathbb{R}^d$ denote the set of vertices and $\{t_1, t_2, \dots, t_{n_t}\}$ the set of tetrahedra, and assume the volume of all triangles/tetrahedra is positive. For the discrete optimization problem, we then search for a solution transformation y in the space

$$\mathcal{V}_h = \left\{ y \in \mathcal{C}(\Omega, \mathbb{R}^d) \ : \ y|_{t_i} \in \Pi^1(t_i, \mathbb{R}^d) \ \text{for } i = 1, \dots, n_t \right\},$$

where Π^1 is the space of first-order, vector-valued polynomials. We note that this space is a subset of the space for which a minimizer is proven to exist for the continuous problem, i.e., $\mathcal{V}_h \subset \mathcal{V}$ as defined in (5.8). Using standard nodal Lagrange hat functions, we construct a basis for \mathcal{V}_h given by $b_1, b_2, \dots, b_{n_v} : \Omega \rightarrow \mathbb{R}$. This allows us to define a discrete transformation $y_h \in \mathcal{V}_h$ on this mesh by the coefficients with respect to that basis, which we store in the vector $\mathbf{y} \in \mathbb{R}^{3n_v}$. That is,

$$y_h(x) = \sum_{j=1}^{n_v} \begin{bmatrix} \mathbf{y}_j^1 \\ \mathbf{y}_j^2 \\ \mathbf{y}_j^3 \end{bmatrix} b_j(x),$$

where the vector entry $\mathbf{y}_j^i = y_h^i(v_j)$ corresponds to the i th dimensional coordinate of vertex j . We also define the coefficients to an untransformed reference transformation \mathbf{y}_{ref} in a similar manner. We use this reference transformation in the implementation of the regularizer.

Given the above expression for a discrete transformation \mathbf{y} on a mesh, we now develop the notation necessary to evaluate the Lagrangian SSD measure, (5.4), and hyperelastic regularizer, (5.6). To help with this, we define the following: \mathbf{I}_k is the $k \times k$ identity matrix; $\mathbf{1}_k$ is the length k vector of all ones; \otimes is the Kronecker product;

and \odot is the Hadamard product.

We denote the discrete vector gradient operator by

$$\mathbf{B} = \mathbf{I}_3 \otimes \nabla_h, \quad \nabla_h = \begin{bmatrix} \partial_1^h \\ \partial_2^h \\ \partial_3^h \end{bmatrix}, \quad \partial_k^h \in \mathbb{R}^{n_t \times n_v}, \quad \text{and } [\partial_k^h]_{i,j} = \partial_k b_i(V_j).$$

Given this definition, we store the entries of Jacobian matrix ∇y_h in the column vector

$$\mathbf{B}\mathbf{y} = \begin{bmatrix} \partial_1^h \mathbf{y}^1 \\ \partial_2^h \mathbf{y}^1 \\ \partial_3^h \mathbf{y}^1 \\ \partial_1^h \mathbf{y}^2 \\ \partial_2^h \mathbf{y}^2 \\ \partial_3^h \mathbf{y}^2 \\ \partial_1^h \mathbf{y}^3 \\ \partial_2^h \mathbf{y}^3 \\ \partial_3^h \mathbf{y}^3 \end{bmatrix} \in \mathbb{R}^{9n_t}.$$

Interpolation from the vertices to the barycenters of the tetrahedra is done using an averaging matrix

$$\mathbf{A} = \mathbf{I}_3 \otimes \hat{\mathbf{A}} \in \mathbb{R}^{3n_t \times 3n_v} \quad \text{with} \quad [\hat{\mathbf{A}}]_{i,j} = \begin{cases} 1/4 & \text{if } v_j \text{ is a node of } t_i \\ 0 & \text{otherwise} \end{cases}.$$

The volumes of the tetrahedra in the mesh are referred to by the following vector and matrix:

$$\mathbf{v} \in \mathbb{R}^{n_t} \quad \text{with} \quad \mathbf{v}_i = \text{vol}(t_i) \quad \text{and} \quad \mathbf{V} = \text{diag}(\mathbf{v}).$$

Lastly, we define the discretized determinant and cofactor, $\det \mathbf{B}\mathbf{y} \in \mathbb{R}^{n_t}$ and

cof $\mathbf{B}\mathbf{y} \in \mathbb{R}^{9n_t}$, as the entry-wise evaluation of the formulas (5.7) using the appropriate entries of $\mathbf{B}\mathbf{y}$. The derivatives of the determinant and cofactors are also needed during optimization. Letting $\mathbf{D}_i^j = \text{diag}(\partial_i \mathbf{y}^j) \in \mathbb{R}^{n_t \times n_t}$, the derivative of cof $\mathbf{B}\mathbf{y}$ is

$$d \text{ cof } \mathbf{B}\mathbf{y} = \left[\begin{array}{ccc|ccc|ccc} 0 & 0 & 0 & 0 & \mathbf{D}_3^3 & -\mathbf{D}_2^3 & 0 & -\mathbf{D}_3^2 & \mathbf{D}_2^2 \\ 0 & 0 & 0 & -\mathbf{D}_3^3 & 0 & \mathbf{D}_1^3 & \mathbf{D}_3^2 & 0 & -\mathbf{D}_1^2 \\ 0 & 0 & 0 & \mathbf{D}_2^3 & -\mathbf{D}_1^3 & 0 & -\mathbf{D}_2^2 & \mathbf{D}_1^2 & 0 \\ \hline 0 & -\mathbf{D}_3^3 & \mathbf{D}_2^3 & 0 & 0 & 0 & 0 & \mathbf{D}_3^1 & -\mathbf{D}_2^1 \\ \mathbf{D}_3^3 & 0 & -\mathbf{D}_1^3 & 0 & 0 & 0 & -\mathbf{D}_3^1 & 0 & \mathbf{D}_1^1 \\ -\mathbf{D}_2^3 & \mathbf{D}_1^3 & 0 & 0 & 0 & 0 & \mathbf{D}_2^1 & -\mathbf{D}_1^1 & 0 \\ \hline 0 & \mathbf{D}_3^2 & -\mathbf{D}_2^2 & 0 & -\mathbf{D}_3^1 & \mathbf{D}_2^1 & 0 & 0 & 0 \\ -\mathbf{D}_3^2 & 0 & \mathbf{D}_1^2 & \mathbf{D}_3^1 & 0 & -\mathbf{D}_1^1 & 0 & 0 & 0 \\ \mathbf{D}_2^2 & -\mathbf{D}_1^2 & 0 & -\mathbf{D}_2^1 & \mathbf{D}_1^1 & 0 & 0 & 0 & 0 \end{array} \right] \mathbf{B}$$

with $d \text{ cof } \mathbf{B}\mathbf{y} \in \mathbb{R}^{9n_t \times 3n_v}$. This expression for the derivative is quite involved and dominates the cost of optimization with the regularizer term for three dimensional problems for both matrix-based and matrix-free implementations. For 2D problems, the cofactor term in the regularizer is not needed, which significantly lowers the cost.

We can denote the derivative of $\det \mathbf{B}\mathbf{y} \in \mathbb{R}^{n_t}$ by

$$d \det \mathbf{B}\mathbf{y} = \left[\mathbf{C}_1^1 \quad \mathbf{C}_1^2 \quad \mathbf{C}_1^3 \quad \mathbf{C}_2^1 \quad \mathbf{C}_2^2 \quad \mathbf{C}_2^3 \quad \mathbf{C}_3^1 \quad \mathbf{C}_3^2 \quad \mathbf{C}_3^3 \right] \mathbf{B} \in \mathbb{R}^{n_t \times 3n_v},$$

where $\mathbf{C}_i^j = \text{diag}([\text{cof } \mathbf{B}\mathbf{y}]_{i,j}) \in \mathbb{R}^{n_t \times n_t}$.

Using these pieces, we can evaluate the discretized objective function $\Phi(\mathbf{y})$ and its derivatives for a general, unconstrained transformation. Discretizing both the Lagrangian SSD measure (5.4) and the hyperelastic regularizer (5.6) using a midpoint

quadrature rule for the numerical integration, we obtain

$$\Phi(\mathbf{y}) = D_L(\mathbf{y}) + S(\mathbf{y}),$$

where the distance is given by

$$D_L(\mathbf{y}) = \frac{1}{2} \mathbf{r}(\mathbf{y})^\top \mathbf{V} \text{diag}(\det \mathbf{B}\mathbf{y}) \mathbf{r}(\mathbf{y}) \quad \text{for} \quad \mathbf{r}(\mathbf{y}) = \mathbf{T}_b - \mathbf{R}(\mathbf{A}\mathbf{y}).$$

Here, the discrete template image $\mathbf{T}_b \in \mathbb{R}^{nt}$ is evaluated at the barycenters of an undeformed, reference mesh using linear interpolation. Note that this expression looks like the two norm in (2.2), although the dependence of the determinant term on \mathbf{y} changes the problem slightly. The evaluation of the regularizer is given by

$$\begin{aligned} S(\mathbf{y}) &= \alpha_1 S_{len}(\mathbf{y}) + \alpha_2 S_{area}(\mathbf{y}) + \alpha_3 S_{vol}(\mathbf{y}) \\ &= \frac{\alpha_1}{2} (\mathbf{y} - \mathbf{y}_{ref})^\top \mathbf{B}^\top (\mathbf{I}_9 \otimes \mathbf{V}) \mathbf{B} (\mathbf{y} - \mathbf{y}_{ref}) + \alpha_2 \mathbf{v}^\top \phi_w(\text{cof } \mathbf{B}\mathbf{y}) + \alpha_3 \mathbf{v}^\top \phi_v(\det \mathbf{B}\mathbf{y}). \end{aligned}$$

Next, we evaluate the gradient as the sum of the gradients of the SSD and regularizer terms,

$$\nabla \Phi(\mathbf{y}) = \nabla D_L(\mathbf{y}) + \nabla S(\mathbf{y}).$$

Using the product rule, the gradient for the SSD term is given by

$$\nabla D_L(\mathbf{y}) = -\mathbf{r}(\mathbf{y})^\top \mathbf{V} \text{diag}(\det \mathbf{B}\mathbf{y}) \nabla \mathbf{R}(\mathbf{A}\mathbf{y}) \mathbf{A} + \frac{1}{2} (\mathbf{r}(\mathbf{y}) \odot \mathbf{V} \mathbf{r}(\mathbf{y}))^\top d \det \mathbf{B}\mathbf{y}, \quad (5.11)$$

and the gradient for the regularizer is given by

$$\nabla S(\mathbf{y}) = \alpha_1 \nabla S_{len}(\mathbf{y}) + \alpha_2 \nabla S_{area}(\mathbf{y}) + \alpha_3 \nabla S_{vol}(\mathbf{y}),$$

where

$$\begin{aligned}\nabla S_{len}(\mathbf{y}) &= (\mathbf{y} - \mathbf{y}_{ref})^\top \mathbf{B}^\top (\mathbf{I}_9 \otimes \mathbf{V}) \mathbf{B} \\ \nabla S_{area}(\mathbf{y}) &= ((\mathbf{1} \otimes \mathbf{v}) \odot \phi'_w(\text{cof } \mathbf{B}\mathbf{y}))^\top d \text{cof } \mathbf{B}\mathbf{y} \\ \nabla S_{vol}(\mathbf{z}) &= \mathbf{v}^\top \phi'_v(\det \mathbf{B}\mathbf{y}) d \det \mathbf{B}\mathbf{y}.\end{aligned}$$

Lastly, we compute an approximation of the Hessian for the objective function as

$$\mathbf{H}(\mathbf{y}) = \nabla^2 D_L(\mathbf{y}) + \nabla^2 S(\mathbf{y}),$$

where the first term is

$$\nabla^2 D_L(\mathbf{y}) \approx \mathbf{A}^\top (\nabla \mathbf{R}(\mathbf{A}\mathbf{y}))^\top \mathbf{V} \text{diag}(\det \mathbf{B}\mathbf{y}) \nabla \mathbf{R}(\mathbf{A}\mathbf{y}) \mathbf{A},$$

and the second term is

$$\nabla^2 S(\mathbf{y}) = \alpha_1 \nabla^2 S_{len} + \alpha_2 \nabla^2 S_{area}(\mathbf{y}) + \alpha_3 \nabla^2 S_{vol}(\mathbf{y}),$$

with

$$\begin{aligned}\nabla^2 S_{len} &= \mathbf{B}^\top (\mathbf{I}_9 \otimes \mathbf{V}) \mathbf{B} \\ \nabla^2 S_{area}(\mathbf{y}) &= (d \text{cof } \mathbf{B}\mathbf{y})^\top (\mathbf{I}_9 \otimes \mathbf{V}) \phi''_w(\text{cof } \mathbf{B}\mathbf{y}) d \text{cof } \mathbf{B}\mathbf{y} \\ \nabla^2 S_{vol}(\mathbf{y}) &= (d \det \mathbf{B}\mathbf{y})^\top \text{diag}(\mathbf{v} \odot \phi''_v(\det \mathbf{B}\mathbf{y})) d \det \mathbf{B}\mathbf{y}.\end{aligned}$$

Here we reiterate an important observation that the Hessians for the area and volume terms of the regularizer are expected to be ill-conditioned when there are large volume changes in the transformation [56]. This is because $\phi''_v(\det \mathbf{B}\mathbf{y}) \rightarrow \infty$ as $\det \mathbf{B}\mathbf{y} \rightarrow 0^+$ or $\det \mathbf{B}\mathbf{y} \rightarrow \infty$. This requires consideration during the step solve for the Gauss–Newton optimization. We also note that the formulas above are for the

three dimensional case. For 2D problems, the area term of the regularizer is dropped, reducing the complexity and cost of evaluating the regularizer.

Discretization for a constrained transformation

The previous section details the discretization for a general, unconstrained transformation, y , parameterized by the vector \mathbf{y} . We now extend the discretization above to the constrained transformation $y(x, \mathbf{w}_1, \dots, \mathbf{w}_N)$ for the locally rigid registration problem. To discretize the constrained transformation by $\mathbf{y}(\mathbf{x}, \mathbf{w})$, we begin as for the general case and initialize the problem with a discretized finite element transformation $\mathbf{y} \in \mathbb{R}^{3n_v}$ on the regular cell grid associated with the images at the lowest optimization level. Using the masks determining the constrained regions Σ_k for all k at the lowest discretization level, we then partition \mathbf{y} into sets corresponding to the unconstrained vertices of the mesh and the constrained vertices denoted by \mathbf{y}_x and \mathbf{y}_{w_k} for all k , respectively. We set $\mathbf{x} = \mathbf{y}_x \in \mathbb{R}^n$ to be the column vector containing the coefficients of the unrestricted transformation on the unconstrained region with $n < 3n_v$ and n divisible by 3. For \mathbf{y}_{w_k} on each constrained region, we restrict the finite element vertices of the mesh in the region Σ_k to the rigid transformation defined by the parameters \mathbf{w}_k . The transformation and its derivatives on these regions of the domain can be expressed in terms of matrix-vector products, see App. A.2. This reduces the number of parameters for each set of nodes \mathbf{y}_{w_k} to the dimension of \mathbf{w}_k . In practice as in previous chapters, we refer to the concatenation of all the \mathbf{w}_k vectors by $\mathbf{w} \in \mathbb{R}^p$. The constrained transformation $\mathbf{y}(\mathbf{x}, \mathbf{w})$ is completely parameterized by the blocks of variables \mathbf{x} and \mathbf{w} . For higher levels of the optimization, we again prolongate the mesh $\mathbf{y}(\mathbf{x}, \mathbf{w})$, and the new mesh is partitioned at each level into constrained and unconstrained regions using the masks for Σ_k at that level. For the constrained regions, the \mathbf{w} parameters for the final iterate of the optimization on the previous level are carried over.

As in the motion correction problems of the previous chapter, the number of parameters in the \mathbf{x} block of variables is considerably larger than in the \mathbf{w} block. However, for the multilevel optimization strategy in this chapter, the size of the \mathbf{x} block increases at each higher levels of the optimization as the mesh is made more fine, while the size of \mathbf{w} remains fixed at every level.

Using the discretized constrained transformation above, we extend the gradients and Hessians for the SSD and regularizer. These expressions now include an extra step of the chain rule in their differentiation. For example, the gradient of the Lagrangian SSD for $\mathbf{y}(\mathbf{x}, \mathbf{w})$ becomes

$$\nabla D(\mathbf{y}(\mathbf{x}, \mathbf{w})) = \nabla_{\mathbf{x}, \mathbf{w}} \mathbf{y}(\mathbf{x}, \mathbf{w}) \nabla_{\mathbf{y}} D(\mathbf{y}(\mathbf{x}, \mathbf{w})),$$

where the second term of that expression, $\nabla_{\mathbf{y}} D(\mathbf{y}(\mathbf{x}))$, is given in (5.11). Similar expressions follow for the Hessian of the SSD measure and the gradient and Hessian of the regularizer. The remaining term, $\nabla_{\mathbf{x}, \mathbf{w}} \mathbf{y}(\mathbf{x}, \mathbf{w})$ can be partitioned into two blocks corresponding to the unconstrained and constrained regions by

$$\nabla_{\mathbf{x}, \mathbf{w}} \mathbf{y}(\mathbf{x}, \mathbf{w}) = \begin{bmatrix} \nabla_{\mathbf{x}} \mathbf{y}(\mathbf{x}, \mathbf{w}) \\ \nabla_{\mathbf{w}} \mathbf{y}(\mathbf{x}, \mathbf{w}) \end{bmatrix}.$$

The first of these blocks, $\nabla_{\mathbf{x}} \mathbf{y}(\mathbf{x}, \mathbf{w})$, corresponds to an identity matrix as \mathbf{x} is simply a subset of the entries of \mathbf{y} corresponding to the unconstrained portion of the domain. The second requires differentiation of the coefficients of the transformation \mathbf{y} on the constrained vertices with respect to rigid motion parameters. Details on differentiation with respect to rigid motion parameters can be in App. A.2. The block structure in the gradient of the constrained transformation affects the derivatives for the constrained problem, resulting in a block system for the Gauss–Newton step at

each iteration,

$$\begin{bmatrix} \mathbf{H}_{xx} & \mathbf{H}_{xw} \\ \mathbf{H}_{wx} & \mathbf{H}_{ww} \end{bmatrix} \begin{bmatrix} \mathbf{p}_x \\ \mathbf{p}_w \end{bmatrix} = - \begin{bmatrix} \nabla_x \Phi(\mathbf{y}) \\ \nabla_w \Phi(\mathbf{y}) \end{bmatrix},$$

where the block matrices are defined as $\mathbf{H}_{xx} = \nabla_x \mathbf{y}^\top \mathbf{H}(\mathbf{y}) \nabla_x \mathbf{y}$ and so on. This system has the same block structure as the normal equations framework for LAP in (3.5). We do not write the system in terms of Jacobian matrices because neither the hyperelastic regularizer nor the gradient are expressible in those terms. Here, the block Hessian \mathbf{H} is symmetric positive definite if the diagonal blocks are full rank, and the off diagonal blocks have the relation $\mathbf{H}_{xw} = \mathbf{H}_{wx}^\top$. We apply LAP to the system and project out the small block associated with the rigid motion parameters and obtain a reduced problem,

$$(\mathbf{H}_{xx} - \mathbf{H}_{xw} \mathbf{H}_{ww}^{-1} \mathbf{H}_{wx}) \mathbf{p}_x = -\nabla_x \Phi(\mathbf{y}) + \mathbf{H}_{xw} \mathbf{H}_{ww}^{-1} \nabla_w \Phi(\mathbf{y}),$$

with

$$\mathbf{p}_w = -\mathbf{H}_{ww}^{-1} (\nabla_w \Phi(\mathbf{y}) + \mathbf{H}_{wy} \mathbf{p}_x).$$

The first of these we solve iteratively for \mathbf{p}_x using PCG method, and the second yields \mathbf{p}_w directly by substitution. The first system is solved to a low tolerance 10^{-1} or 10^{-2} at each Gauss–Newton iteration. Preconditioning is an important consideration for this problem, and we demonstrate this in the numerical results section below where we consider a number of preconditioning options. For the substitution, we can compute and store the Cholesky factors of \mathbf{H}_{ww} once per Gauss–Newton iteration and use them repeatedly to reduce the computational cost since the dimension of \mathbf{w} is small.

5.2 Numerical Results

In this section, we present the results for two locally rigid registration problems. The first is a two dimensional example registering a real-valued MRI of a human knee. The second is a three dimensional example registering two blocks. For both problems, we explore using LAP for the coupled Gauss–Newton step solve. We compare its performance with solving the fully coupled system (‘Full’ in tables and figures). In particular, we are interested the potential benefits of LAP when the Gauss–Newton step problem is ill-posed and more difficult to solve, i.e., for transformations when $\det \mathbf{B}y$ is very small for some elements. To analyze this, we plot the iteration behavior of preconditioned conjugate gradient (PCG) on the fully coupled system and the projected LAP system for various preconditioners for the iteration with the smallest minimum value for $\det \mathbf{B}y$ at each level of the multilevel registration. Using a matrix-based framework for both problems, we compare diagonal, and symmetric Gauss-Seidel (SGS) preconditioning for both the fully coupled and LAP solve. We also compare the 2-norm of the difference between the registered images to compare any differences in the resulting registrations for the two methods, and we look at the time to solution as a cost-comparison metric.

5.2.1 2D Knee Registration

To set up a 2D registration problem using a knee MRI, we extract 128×128 slices from the template and reference data provided in the 3D knee MRI dataset from FAIR [47]. The same slice is taken from both the template and reference images and can be seen in Fig. 5.2. The two large bones above and below the knee joint are set as the two constrained regions, Σ_1 and Σ_2 , in the template image. Binary masks of dimension 128×128 for these constrained regions are created using manual segmentation using Matlab’s `imageSegmenter` GUI. We set the image domain to

$\Omega = [0, 128] \times [0, 128]$. This high-level data is then down-sampled to create multi-level data using the `getMultilevel` function in the FAIR toolbox. We use 7 as the highest level of registration and 4 as the lowest for this dataset, where 7 indicates the exponent of the square image dimension with respect to base 2, i.e., level 7 signifies 128×128 data as $2^7 = 128$. One issue is that the downsampling uses block average downsampling, making the lower-level masks of the constrained regions non-binary. To fix this, we use a truncation tolerance of 0.25 to set pixels in the mask with intensity less than or equal to the tolerance equal to 0 and round pixel values greater than the tolerance to 1. Note that this strategy of block averaging followed by truncation for the masks may lead to overlapping constraint regions, particularly if the images and masks are downsampled to a low level. This is infeasible, and in practice means that locally rigid registration problems should not downsample over too many levels, particularly if the constrained regions are close to one another as is the case for this problem. In general for other problems, the segmented masks, downsampling strategy, and lowest level for registration should be adapted to ensure that the constrained regions are disjoint.

The downsampled images and masks are created on a regular rectangular grid. We then introduce a discretized transformation $\mathbf{y}(\mathbf{x}, \mathbf{w})$ using the finite element strategy detailed in the previous section and introduced by [5]. For levels 4 through 7, this results in optimization problems of sizes 938, 3,734, 14,980, and 60,196. At each level, $\mathbf{w} \in \mathbb{R}^6$ gives the rigid motion parameters for the two constrained regions, and the remaining parameters constitute \mathbf{x} , the parameterization of transformation for the nonparametric registration on the unconstrained regions of the Ω .

We use an initial guess of $\mathbf{w} = \mathbf{0}$ for the motion parameters of the rigid registration on the constrained region Σ_1 and Σ_2 . This number of parameters is constant over all levels of the optimization. At the lowest level, the zero initial guess necessitates some overlap of the constrained regions in the template image and reference image

2D Knee

| Method | Precond. | $\ \mathbf{T} - \mathbf{R}(\mathbf{y})\ $ | PCG Iters. | | | | Time(s) |
|--------|----------|---|------------|-----------|-----------|-----------|-------------|
| | | | Level 4 | Level 5 | Level 6 | Level 7 | |
| LAP | SGS | 5.25 | 57 | 87 | 92 | 77 | 18.3 |
| | Diagonal | 5.27 | 115 | 201 | 155 | 252 | 16.5 |
| | None | 5.26 | 188 | 303 | 341 | 651 | 18.4 |
| Full | SGS | 5.24 | 74 | 101 | 90 | 167 | 22.1 |
| | Diagonal | 5.27 | 149 | 178 | 197 | 210 | 16.7 |
| | None | 7.87 | 104 | 210 | 119 | 127 | 15.6 |

3D Blocks

| Method | Precond. | $\ \mathbf{T} - \mathbf{R}(\mathbf{y})\ $ | PCG Iters. | | Time(s) |
|--------|----------|---|-------------|-------------|------------|
| | | | Level 5 | Level 6 | |
| LAP | SGS | 5.91 | 2000 | 1344 | 14331 |
| | Diagonal | 6.35 | 2000 | 1453 | 2786 |
| | None | 6.32 | 2000 | 1875 | 3045 |
| Full | SGS | 5.91 | 2000 | 1101 | 11278 |
| | Diagonal | 14.07 | 1038 | N/A | 488 |
| | None | 37.66 | 1473 | 1505 | 2857 |

Table 5.1: This table shows various statistics comparing LAP versus solving the fully coupled system (Full) for registration for registering 2D knee images and 3D blocks images. For both problems, LAP and Full were used for various preconditioners (displayed row-wise). From left to right, the columns show the 2-norm difference of the transformed reference image to the template image, the total number of PCG iterations for the Gauss–Newton step systems at each level of the optimization, and the CPU times for various methods. The lowest value for each column is bold faced for emphasis.

to avoid becoming stuck in local minima. For the unconstrained regions, the initial guess for \mathbf{x} at the lowest level is given by a regular triangular mesh constructed on the rectangular grid of the untransformed template image at that level. For higher levels, the transformation on the unconstrained region from the final iterate of the next lowest level is prolonged to become the initial guess at the new level [5]. The parameters for the rigid transformation on the constrained portions of the domain are passed along as the initial guess at the next level with no prolongation needed. The prolongation on the nonparametric region of the domain results in increasingly expensive optimization problems at each level.

We run the problem using the LAP method and the fully coupled approach with no preconditioner, diagonal preconditioning, and symmetric Gauss-Seidel (SGS) preconditioning with 10 iterations of Gauss–Newton at each optimization level. When solving for the Gauss–Newton step for both strategies, we use PCG with a tolerance

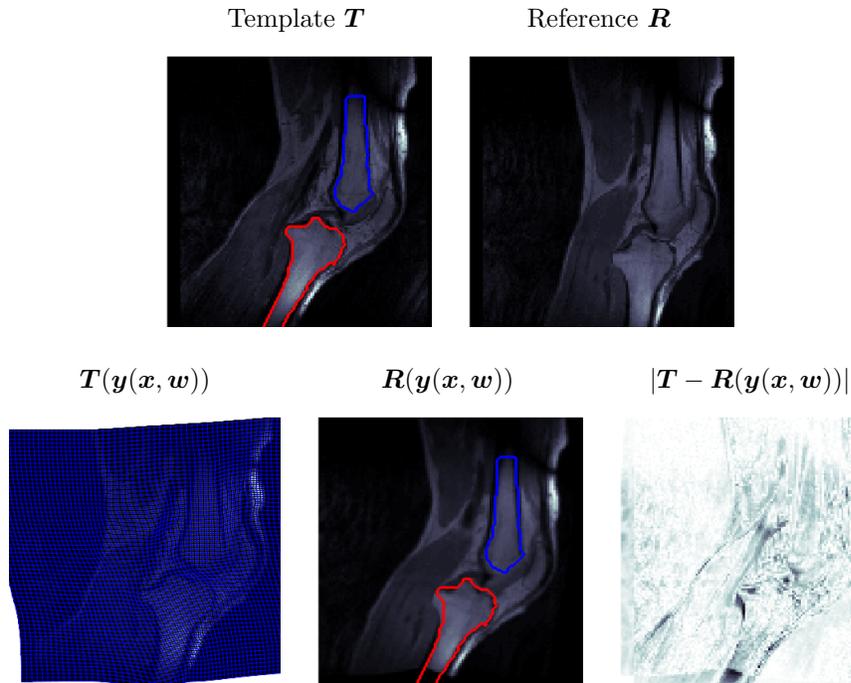


Figure 5.2: Results for locally rigid registration for a 2D knee MRI. The first row shows the original template and reference images for the problem with the constrained regions Σ_1 and Σ_2 outlined in blue and red on the template image. The second row shows the results of the registration. To the *left*, we see the transformed template image, $\mathbf{T}(\mathbf{y}(\mathbf{x}, \mathbf{w}))$ with the solution transform overlaid in a blue grid. The *middle* image shows the transformed reference image used for the Lagrangian distance measure, and the image on the *right* shows the absolute value of the error between the template image and the transformed reference image.

of 10^{-1} and a maximum of 100 iterations. For both methods, we use $\alpha_1 = \alpha_2 = \alpha_3 = 0.01$ as the regularization parameter, noting that this may not be optimal. For this small problem, we form the Hessian matrices explicitly

We run each strategy with no preconditioning, diagonal preconditioning, and symmetric Gauss-Seidel preconditioning. To assess the quality of the registration, we monitor the norm of the difference between the template image and the transformed reference image, $\|\mathbf{T} - \mathbf{R}(\mathbf{y})\|^2$, which is the quantity minimized by the distance term of the optimization. Before the registration, the norm between the two images is 15.24. To quantify cost, we track the total optimization time for the methods over 5 sepa-

rate runs of the problem. We also track the total number of PCG iterations at each level of the optimization. This gives an indication of how the solution strategy affects the solver for the Gauss–Newton step. These values can be found in Table 5.1. We also monitor the impact of the solver at each Gauss–Newton iterate by looking at the relative error for PCG solving for the Gauss–Newton step for both strategies. Fig. 5.4 shows plots of the relative error at every level of the optimization for the iterate with the smallest value for $\min(\det \mathbf{B}\mathbf{y})$, which constitutes the most ill-conditioned step solve at each optimization level. Lastly, we display the registered solution images and solution transformation in Fig. 5.2.

The results of the tests in Table 5.1 show several points of interest. Firstly, the solutions for both the LAP and fully coupled strategies depend heavily on the preconditioner used. This is particularly noticeable in the poor quality of the registration for the fully coupled approach when no preconditioner is used. This results in a poor registration. For the other preconditioners, the solutions are similar in terms of the 2-norm, and the difference between the strategies is in terms of cost and time to solution. Symmetric Gauss-Seidel preconditioning requires the fewest PCG iterations but is slower than diagonal preconditioning due to the cost of applying the preconditioner. Comparing the two strategies for a fixed preconditioners, LAP requires fewer PCG iterations and takes less time than the fully coupled approach for both SGS and diagonal preconditioning for this problem. This trend does not hold without preconditioning, but we disregard this as LAP results in a substantially better registration of the template image. Overall, the reduction in PCG iterations and time to solution for LAP suggests that the numerics are better for the projected system solved in the LAP strategy. For this small 2D example, this results in preferable cost and time to solution.

Interestingly, the superior performance of LAP for the PCG systems is not easily observable in Fig. 5.4 where we plot the relative error of the PCG iterations for the

iteration with the smallest value of $\det \mathbf{B}_y$ for each level of the optimization. The convergence for LAP is not significantly faster or slower than for the fully coupled approach in terms of the relative error. This holds across optimization levels and for all tested preconditioners. This behavior is also observable for other iterations of the optimization for which the relative error plots are not shown. However, note that the relative error plots for LAP versus the fully coupled approach are for two different numerical systems, meaning that a direct comparison between the convergence rates of the two may not be especially meaningful. One useful observation from the PCG plots is that the relative error behavior using LAP is less oscillatory than when solving the fully coupled system, particularly when no preconditioner is used. From Table 5.1, the fully coupled system with no preconditioner results in a poor registration solution, suggesting that the oscillations indicate poor numerical behavior of the unpreconditioned, fully coupled system. We see further evidence of this for the larger 3D problem in the next section.

Overall, the results for this small two dimensional problem suggest that LAP performs slightly better than the fully coupled approach, especially if no preconditioner is used. This is expected for coupled problems. With appropriate preconditioning, the methods both result in good registrations, but LAP is less costly in terms of PCG iterations and reaches a solution more quickly. However, the difference between the methods is not drastic for this small problem. To further explore the differences between the two approaches, we test the two methods in the next section on a larger, more difficult 3D problem.

5.2.2 3D Block Registration

Next, we set up a small 3D registration problem with local rigidity constraints. To do this, we create template and reference images of dimension $64 \times 64 \times 64$. The reference image has two vertically aligned blocks, while the template image subjects

those blocks to two, separate rigid transformations. The two images can be seen in Fig. 5.3. The two blocks are used as the rigidly constrained regions Σ_1 and Σ_2 , and masks at the finest discretization are created using truncation. We set the image domain arbitrarily as $\Omega = [-32, 32] \times [-32, 32] \times [-32, 32]$. Multilevel data for the optimization is again created using FAIR’s `getMultiLevel` function. Again, we use a truncation strategy with a tolerance 0.25 to obtain binary masks from the interpolated masks provided by `getMultiLevel`. To ensure that the rigid regions and masks are disjoint, we choose a coarsest registration level of 5, i.e., images of dimension $32 \times 32 \times 32$ where $2^5 = 32$. The highest level of registration is then 6 for the original images.

The discretized transformation $\mathbf{y}(\mathbf{x}, \mathbf{w})$ results in optimization problems of dimension 500,394 and 3,947,997 for levels 5 and 6, respectively. At each level, 12 parameters in \mathbf{w} correspond to the rigid motion on the two constrained regions, while the remainder of the variables in \mathbf{x} are from the nonparametric registration on the unconstrained region. Note that the finite element discretization for the 3D case divides each rectangular cell from the original image into 24 tetrahedra. This makes the size of the \mathbf{x} block of variables for the nonparametric registration very large for 3D problems.

At the lowest level of the multilevel optimization, we use a zero initial guess, $\mathbf{w} = \mathbf{0}$, for both of the constrained regions. For the unconstrained region, we use a regular tetrahedral mesh on the untransformed template image. When moving to a finer discretization, the rigid motion parameters at the last iterate of the coarse discretization are passed on, and the mesh on the unconstrained region is prolonged to become the initial guess at the finer level [5]. We reiterate that this prolongation results in increasingly expensive problems at finer discretizations.

As in the 2D example, we solve the problem using both LAP and the fully coupled approach, and for each approach solve with no preconditioning, diagonal precondi-

tioning, and symmetric Gauss-Seidel preconditioning. We initially used a tolerance of 10^{-1} and a maximum of 100 iterations for PCG in the Gauss-Newton step solve for both methods, but this gave poor results. This was due to the ill-posedness of the problem, so we changed the tolerance to 10^{-2} with a maximum of 200 iterations and achieved better results for both approaches. While this problem is significantly larger than the 2D knee example, we still form the Hessian matrices explicitly. The regularization parameters are set to $\alpha_1 = \alpha_2 = \alpha_3 = 0.01$ by trial and error. When α_j are chosen too small, insufficient regularization is included, resulting in nondiffeomorphic solution transformations and a failure of the optimization. When α_j are chosen too large, big nonlinear deformations are heavily penalized during the optimization resulting in poor registration results. The α values above were chosen because they result in meaningful registration results and diffeomorphic solution transformations, but they may not be optimal.

The three dimensional problem is more difficult than the two dimensional one, and this results in some stratification between LAP and the fully coupled approach. We also see variation across the regularization approaches. Once again, we look at the 2-norm of the difference between the transformed reference image and the template image as a measure of the quality of the registration. The norm of the difference between the two images before registration is 53.12. We track the total number of PCG iterations for each approach at every optimization level, and we measure the total time used for the registration using Matlab's `tic` and `toc` functions. These results are shown in Table 5.1, and the resulting transformed reference image can be seen in Fig. 5.3.

For this problem, LAP does slightly better in the optimization than the fully coupled approach, particularly when no preconditioner or a diagonal preconditioner is used. With no preconditioning, LAP finds a solution transformation that results in better registration in terms of $\|\mathbf{T} - \mathbf{R}(\mathbf{y})\|$. The solution for LAP with diagonal

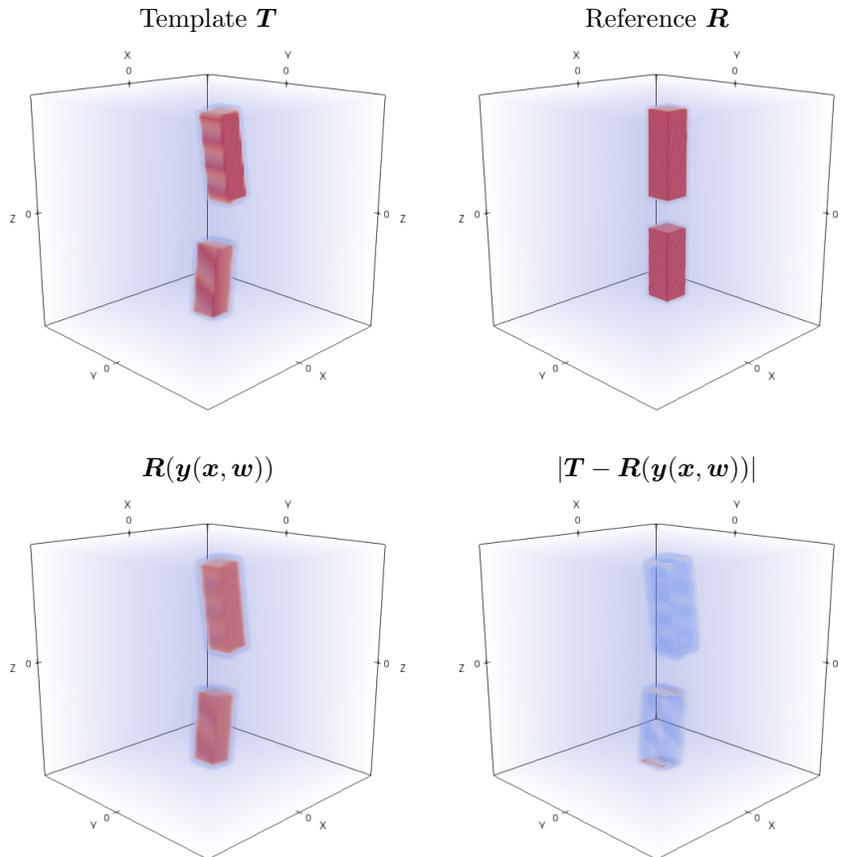


Figure 5.3: Results for locally rigid registration for 3D Blocks using LAP with a symmetric Gauss-Seidel preconditioner. The first row shows the given template image \mathbf{T} (left) and reference image \mathbf{R} (right). The bottom row shows the transformed reference image $\mathbf{R}(\mathbf{y}(\mathbf{x}, \mathbf{w}))$ (left) and the absolute value of the error (right). The SSD measure for the registration problem measures the distance of the template image and the transformed reference image.

preconditioning is similarly better. Here, the fully coupled approach stalled after failing to find an update resulting in a diffeomorphic transformation during the highest level of the optimization. Note that this is not due to the value of the regularization parameters as the optimization problem is identical for LAP and the fully coupled approach regardless of the preconditioning. When a symmetric Gauss-Seidel preconditioner is used, LAP and the fully coupled approach find solution transformations with similar registration results in terms of $\|\mathbf{T} - \mathbf{R}(\mathbf{y})\|$. However, we note that for

larger problems where a matrix-free implementation is required, a direct preconditioner like symmetric Gauss-Seidel may be unavailable. Symmetric Gauss-Seidel is also costly, as we see from the timings using it for both LAP and the fully coupled approach. Contrast this with diagonal preconditioning, which is cheaper to apply and can be implemented in a matrix-free implementation. Across all preconditioners, LAP is slightly more expensive than the fully coupled approach in terms of time and PCG iterations. This is different than the results for the 2D example where LAP was faster and cheaper. Overall, the results show that the optimization behaves better using LAP, but that there may be some increased cost associated with the method. The difference is particularly noticeable when no preconditioning or diagonal preconditioning is used when solving the linear system for the Gauss–Newton step.

We also look at the convergence behavior of PCG when solving for the Gauss–Newton step. Plots of this are in Fig. 5.5 where we plot the relative error of the PCG iterates at each optimization level for the Gauss–Newton system with the smallest value for $\det \mathbf{B}y$. There is some difference in the order of convergence, with the fully coupled approach converging more quickly at the lowest level optimization and LAP converging more quickly at the highest level. However, this behavior is not necessarily indicative of the quality of the resulting registration. More importantly, we again see that the convergence plots for LAP are less oscillatory than those for the fully coupled approach. LAP without a preconditioner does oscillate, but these oscillations are dampened compared to the fully coupled approach. This suggests better numerical behavior for LAP compared to the fully coupled approach but reinforces the need for good preconditioning.

5.2.3 Discussion

We end the section with a brief discussion summarizing the results of the numerical tests for the locally rigid registration problem. First, the chapter and numerical tests

show the applicability of LAP to coupled problems that are nonlinear in both sets of variables. This extends on the work from Ch. 4 where we tested the method on separable least-squares problems that were linear in the larger, \mathbf{x} block of variables. We also showed another example of LAP's flexibility for regularization using the nonlinear, hyperelastic regularizer.

The two examples compared LAP to the fully coupled approach for the multilevel optimization problem of image registration subject to local rigidity constraints. We tested both methods with three different preconditioning options. LAP resulted in similar or better registration results in terms of the 2-norm distance between the reference and transformed template image. This was especially evident when no preconditioner was used in the 3D example. However, there was no significant difference in the registration results between LAP and the fully coupled approach using symmetric Gauss-Seidel preconditioning. Additionally, LAP had the drawback of being slower and taking more iterations for the larger, 3D example. Overall, the results suggest that LAP is somewhat preferable to the fully coupled approach, especially if a suitable preconditioner is unavailable. It is not significantly more or less expensive than the fully coupled approach, but it results in better optimization results for the problem for our examples.

PCG convergence plots for 2D knee

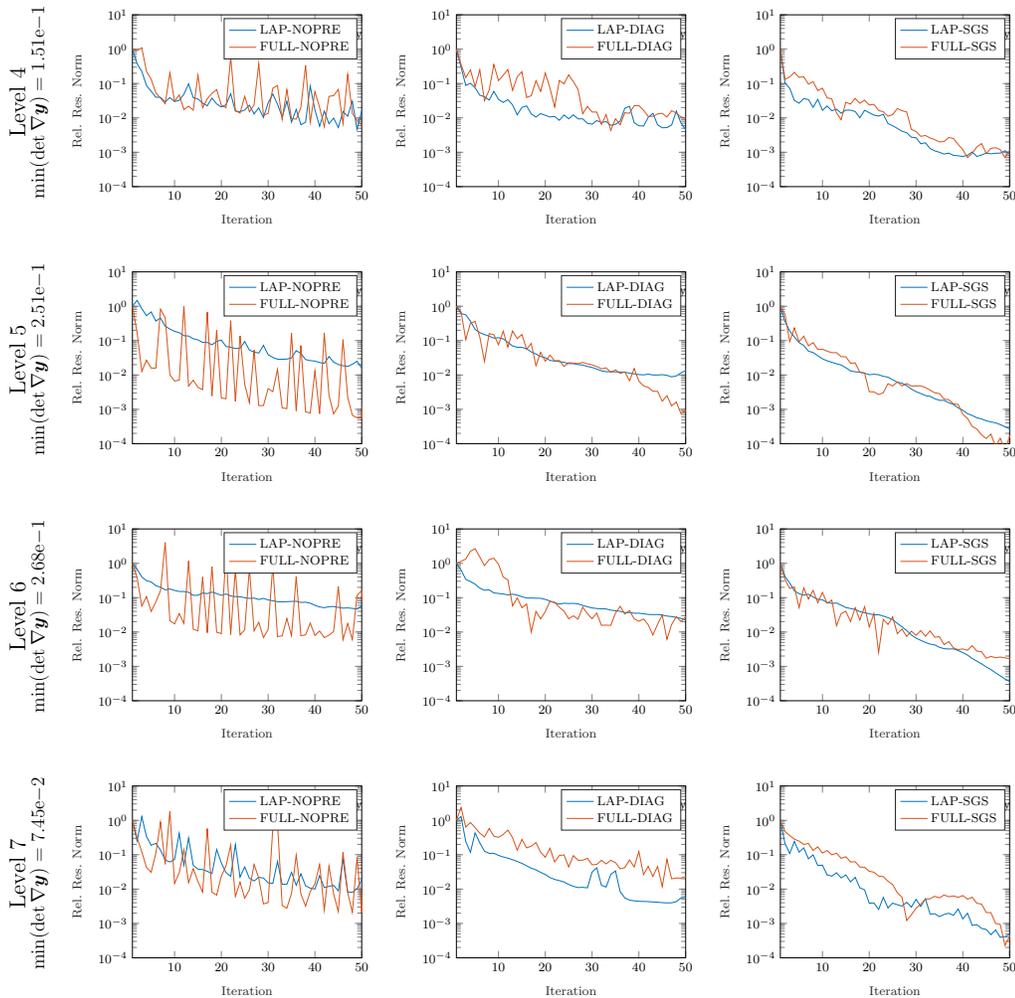


Figure 5.4: These plots compare the PCG convergence for the Gauss–Newton step problem for LAP (*blue*) versus solving the full coupled system (*orange*) for the registration of a 2D knee MRI. Each row corresponds to a different level of the optimization. At each level, we choose the Gauss–Newton iterate with the smallest value for $\min(\det(\nabla \mathbf{y}))$, i.e., the most ill-conditioned system at that level. The columns show PCG convergence behavior for LAP and the fully coupled system for no preconditioning, diagonal preconditioning, and symmetric Gauss–Seidel symmetric preconditioning.

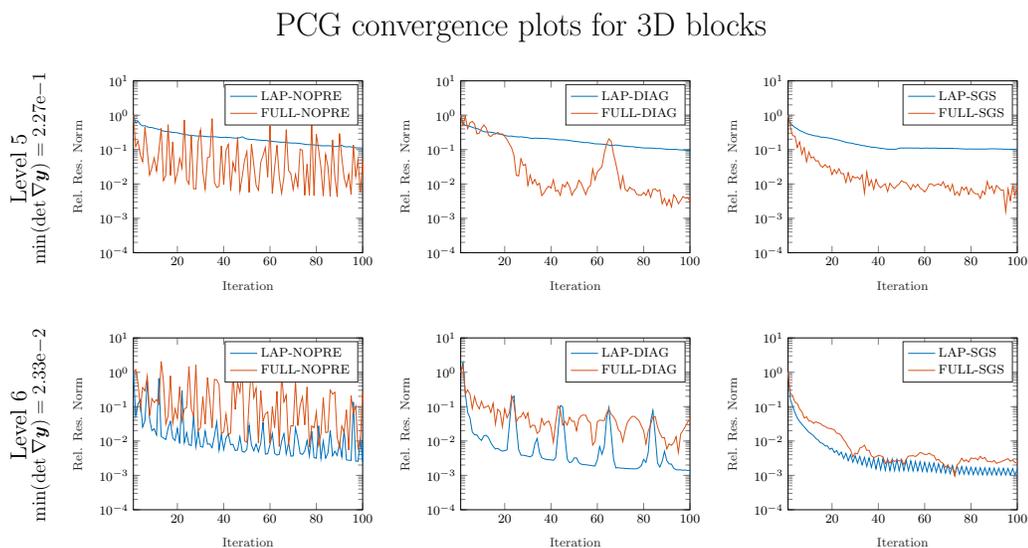


Figure 5.5: These plots compare the PCG convergence for the Gauss–Newton step problem for LAP (*blue*) versus solving the full coupled system (*orange*) for the 3D registration of two blocks. Each row corresponds to a different level of the optimization. At each level, we choose the Gauss–Newton iterate with the smallest value for $\min(\det(\nabla \mathbf{y}))$, i.e., the most ill-conditioned system at that level. The columns show PCG convergence behavior for LAP and the fully coupled system for no preconditioning, diagonal preconditioning, and symmetric Gauss–Seidel symmetric preconditioning.

Chapter 6

Phase Recovery from the Bispectrum

This chapter presents work on phase recovery in bispectral imaging, a univariate, nonlinear optimization problem in astronomical imaging. This problem fits within nonlinear optimization framework of Ch. 2. In this chapter, we extend previous work by applying Gauss–Newton optimization to the univariate nonlinear problem of phase recovery, whereas previous work focused primarily on gradient-based methods. In particular, we show that by using both direct and iterative methods for appropriate sparse Hessians, Gauss–Newton optimization offers faster convergence, better solutions, and lower cost than both gradient descent and the nonlinear conjugate gradient method for this problem.

The results of our work on this topic were prepared and presented for the 2015 AMOS conference by the author and J. Nagy, and this chapter’s material and results borrow heavily from the technical report published for the conference [35].

6.1 Bispectral Imaging Problem

Astronomers have developed a number of techniques to solve the problem of obtaining high-resolution astronomical images in the visual spectrum from ground-based telescopes. Due to the faintness of many such astronomical objects, particular interest is paid to techniques that can produce high-resolution images from photon-limited data. One common method for obtaining high-resolution images under such circumstances is speckle interferometry, which was initiated by Labeyrie’s observation in 1970 that high spatial frequency information could be recovered from low-light, short-exposure images [41]. His method uses two sets of short-exposure image frames, one set of the desired object and the other of a reference star, to obtain a single, composite reconstruction of an object’s energy spectrum, i.e., Fourier modulus. However, while the object’s recovered energy spectrum contains sufficient information about the object to produce images for many simple astronomical objects, it proves insufficient for obtaining high-resolution images for more complicated objects. In the case of a more complicated object, obtaining a high-resolution image also necessitates the reconstruction of the object’s Fourier phase. Several methods have been proposed for this phase reconstruction, most utilizing an object’s triple correlation, its bispectrum, or some subset of these high-order statistical correlation measures that are obtainable from the data [39, 65].

The goal of this chapter is to review and expand on several of the algorithms presented in the literature for phase recovery using an object’s bispectrum. In particular, we focus on the mathematical formulations of these phase retrieval algorithms as non-linear least-squares problems and solve them using the Gauss–Newton optimization framework detailed in Ch. 2. Areas of interest include efficient implementation of Gauss–Newton optimization and appropriate regularization to improve the quality of the resulting images.

The chapter is organized into the following sections. Sections 6.1.1 and 6.1.2

provide a brief description of the speckle interferometry problem and the efficient collection of the object's bispectrum. Note that this chapter only aims to cover this setup in sufficient depth to present the subsequent bispectrum recovery problem in a cogent manner. For a more extensive discussion of the speckle interferometry problem, we recommend the articles listed in the bibliography, particularly the paper of Negrete-Regagnon [49]. Section 6.1.3 introduces the problem of phase recovery from an object's bispectrum including the objective functions, gradients, and (approximate) Hessians used for the optimization. Section 6.2 provides some numerical results for the optimization, and the final section offers some concluding remarks about the results.

6.1.1 Calculating the Object Power Spectrum

Before one can consider Fourier phase recovery using bispectral methods, one must approach the primary problem of speckle interferometry: recovery of the object's Fourier modulus or power spectrum. To do this, we start from the continuous model for describing the blurring of an object $o(x, y)$ by an atmosphere-telescope system resulting in an observed image $i(x, y)$. For a single short-exposure observation, this blurring is expressed in terms of the convolution operator,

$$i(x, y) = \iint_{-\infty}^{\infty} o(x', y') o(x - x', y - y') dx' dy', \quad (6.1)$$

where the convolution kernel $h(x, y)$ is the point spread function (PSF) that models the blurring of a given point of the object as a function of the atmosphere-telescope system. In Fourier space, the convolution property of the Fourier transform allows the equation above to be expressed by

$$I(u, v) = O(u, v)H(u, v), \quad (6.2)$$

where the multiplication is taken component-wise in frequency space. Here, $I(u, v)$, $O(u, v)$, and $H(u, v)$ denote the two dimensional Fourier transforms of their lower-case counterparts in (6.1). We refer to $H(u, v)$ as the optical transfer function (OTF), which can be expressed as a product of fixed aberrations due to telescope optics and random aberrations due to atmospheric turbulence. A more complete discussion of the properties and accurate simulation of $H(u, v)$ can be found in the paper by Negrete-Regagnon and the references therein [49].

The equations above describe a single instance of the blurring of an object, resulting in a single short-exposure image frame. In practice when dealing with photon-limited imaging, multiple frames are needed to provide sufficient data to recreate a single, high-resolution composite image. Following this idea, if we assume a set of N short-exposure images and denote the Fourier transform of the k th image by $I_k(x, y)$, then taking an ensemble average we get

$$\langle I_k(u, v) \rangle = O(u, v) \langle H_k(u, v) \rangle,$$

where $H_k(u, v)$ is the OTF associated with the k th observed image. Here, we define the ensemble average as

$$\langle I_k(u, v) \rangle = \frac{1}{N} \sum_{k=1}^N I_k(u, v)$$

for N realizations of (6.2). This approach is equivalent to producing the Fourier spectrum of a single long-exposure image, but has the drawback of suppressing desirable high-frequency information due to the “averaging out” effect of the ensemble-average operation. In order to avoid this drawback, Labeyrie proposed taking the ensemble average of the modulus squared of the Fourier transform of the data [41],

$$\langle |I_k(u, v)|^2 \rangle = |O(u, v)|^2 \langle |H_k(u, v)|^2 \rangle. \quad (6.3)$$

Here, the modulus squared operation produces strictly non-negative values for the ensemble average, which allows for the retention of high-frequency information about the Fourier spectrum of the object. It follows that simple component-wise operations provide an expression for the recovery of the Fourier modulus of the object,

$$|O(u, v)| = \left[\frac{\langle |I_k(u, v)|^2 \rangle}{\langle |H_k(u, v)|^2 \rangle + \epsilon} \right]^{\frac{1}{2}}, \quad (6.4)$$

where the ϵ in the denominator is a small parameter added to prevent division by zero. The modulus squared OTF, $|H_k(u, v)|^2$, is commonly called the modular transfer function (MTF) and in practice is unknown. Instead, it is replaced by the ensemble average of the Fourier modulus squared of an appropriate reference star, the second set of short-exposure image data collected for speckle interferometry. If we denote the k th frame of this star data by $s_k(x, y)$ and its Fourier transform by $S_k(u, v)$, we have

$$\langle |S_k(u, v)|^2 \rangle \approx \langle |H_k(u, v)|^2 \rangle.$$

This substitution works because the reference star acts as a point source, and the data should be collected concurrently with the object data in order to ensure the same astronomical seeing conditions [49]. Effecting this substitution in (6.4), we can calculate the object's Fourier modulus, also known as its power spectrum, by

$$|O(u, v)| = \left[\frac{\langle |I_k(u, v)|^2 \rangle}{\langle |S_k(u, v)|^2 \rangle + \epsilon} \right]^{\frac{1}{2}}.$$

The above expression can be calculated in the discrete setting using element-wise operations on the collected data and must be multiplied by an appropriate mask to prevent the computed power spectrum from being overwhelmed by noise in the data. Details for this can be found in Sect. 7 of Negrete-Regagnon [49].

The above provides a short summary of both the impetus and the means to

calculate an object's power spectrum. A few other remarks are worth noting. First, an object's power spectrum provides the most significant information relevant to accurate image recovery using speckle interferometry. It is the primary problem in the sense that without an accurate recovery of the object's power spectrum, recovering its Fourier phase does not allow for recovery of a high-quality resultant image. The ability to recover the object's power spectrum accurately is dependent on the data signal-to-noise ratio (SNR) that in turn depends on atmospheric conditions, light levels of the data (or photo-events per data frame), and optics setup among other things. For a more complete discussion of these factors, we again recommend Negrete-Regagnon and its references [49].

6.1.2 Accumulating the Object Bispectrum

Having outlined a method for obtaining an object's power spectrum using speckle interferometry, we now turn to the problem recovering the object's Fourier phase. Most techniques for this focus on the use of correlation techniques, with much of the literature focusing on the use of the object's triple correlation and its Fourier transform, the bispectrum [39, 65]. An object's triple correlation measures the correlation of the object against two independently shifted copies of itself in the spatial domain. In two dimensions, this is expressed by

$$o^{TC}(x_1, y_1, x_2, y_2) = \iint_{-\infty}^{\infty} o^*(x, y) o(x + x_1, y + y_1) o(x + x_2, y + y_2) dx dy.$$

Taking a Fourier transform of the equation, we get the object's bispectrum

$$\begin{aligned} O^{(3)}(u_1, v_1, u_2, v_2) &= O(u_1, v_1)O(u_2, v_2)O^*(u_1 + u_2, v_1 + v_2) \\ &= |O^{(3)}(u_1, v_1, u_2, v_2)| \exp(i\beta(u_1, v_1, u_2, v_2)), \end{aligned} \quad (6.5)$$

where the element-wise relationship comes from an analogous property to the convolution property for the Fourier transform. Noting that we already have a method for obtaining the object's Fourier modulus from the previous section, we shift our focus to the exponential term of the bispectrum and its angle, which we call the phase, $\beta(u_1, v_1, u_2, v_2)$.

First, we establish the relationship between the phase of the bispectrum and the object's phase that we want to recover, ϕ . It has been shown that similarly to (6.3), we can relate the collected ensemble bispectrum of the data to the bispectrum of the object by

$$\langle I_k^{(3)}(u_1, v_1, u_2, v_2) \rangle = O^{(3)}(u_1, v_1, u_2, v_2) \langle H_k^{(3)}(u_1, v_1, u_2, v_2) \rangle.$$

Furthermore, analysis by Lohmann *et al.* showed that the bispectral transfer function, $\langle H_k^{(3)}(u_1, v_1, u_2, v_2) \rangle$ has an expected real value, meaning that its phase is effectively zero [42]. This then gives a direct relationship between the phase of the data's bispectrum, β , and the phase of the object that is to be recovered

$$\begin{aligned} \beta(u_1, v_1, u_2, v_2) &= \phi(u_1, v_1) + \phi(u_2, v_2) - \phi(u_1 + u_2, v_1 + v_2) \\ \beta(\mathbf{u}, \mathbf{v}) &= \phi(\mathbf{u}) + \phi(\mathbf{v}) - \phi(\mathbf{u} + \mathbf{v}), \end{aligned} \quad (6.6)$$

where the second expression is identical to the first but substitutes in the vector notation for the Fourier coordinates, $\mathbf{u} = (u_1, v_1)$ and $\mathbf{v} = (u_2, v_2)$. We will use

this convention for the remainder of the chapter to simplify indexing. This equation provides a direct relation between the elements of the phase of the data bispectrum and the phase of the object, and it serves as the starting point for all of the phase recovery algorithms in Sect. 6.1.3.

In order to use (6.6) to recover the object phase, one must first be able to efficiently accumulate the phase of the collected ensemble bispectrum of the data, $\beta(\mathbf{u}, \mathbf{v})$ for a large number of points \mathbf{u}, \mathbf{v} in the Fourier domain. To do this, we again look at the exponential term in (6.5). Denoting by ψ the phase of a single data frame $I_k(u, v)$, the exponential part of the equation can be written as

$$\exp(i\beta(\mathbf{u}, \mathbf{v})) = \exp(i\psi(\mathbf{u})) \exp(i\psi(\mathbf{v})) \exp(-i\psi(\mathbf{u} + \mathbf{v})).$$

Taken over the set of N data frames, the phase of the bispectrum then becomes

$$\beta(\mathbf{u}, \mathbf{v}) = \text{angle} \left(\sum_{k=1}^N \exp(i\psi_k(\mathbf{u})) \exp(i\psi_k(\mathbf{v})) \exp(-i\psi_k(\mathbf{u} + \mathbf{v})) \right).$$

From this, one can see that efficiently computing the object's bispectrum becomes a question of identifying all the $(\mathbf{u}, \mathbf{v}, \mathbf{u} + \mathbf{v})$ triplets necessary to evaluate and sum the right hand side across for each data frame (the triplets remain constant across all frames.) The most efficient method of doing this is by means of an indexing structure that saves the indices of these triplets and vectorizes computation of the bispectrum accumulation. Additional efficiency can be attained by exploiting the symmetries in the phase of real images in Fourier space. A description of the logic behind such an indexing structure can be found in work by Tyler and Schulze [61].

We note that even with the efficiency afforded by vectorization and symmetry when using an indexing structure, the number of bispectrum elements becomes infeasible if \mathbf{u} and \mathbf{v} span over the whole image when collecting the triplets. In practice, we restrict \mathbf{u} and $\mathbf{u} + \mathbf{v}$ within a larger radius called the Fourier radius. Typically, the

Fourier radius is between $\frac{1}{4}$ and $\frac{1}{2}$ of the pixels in the domain, for example, between 32 and 64 pixels for an image of size 128×128 . We restrict \mathbf{v} to a smaller, second radius between 3 and 5 pixels. Fortunately, this choice is not just a computational compromise, but is driven by considerations inherent to the problem. Both radii are sensitive to astronomical seeing conditions and telescope optics and should be chosen with care [61]. A good choice has the effect not only of ensuring the collected bispectrum has a good signal-to-noise ratio but also has the practical benefit of making the number of bispectrum elements computationally feasible for the phase recovery problem.

6.1.3 Phase Recovery Problem

Objective Functions

Once the data bispectrum has been collected, we are prepared to approach the problem of recovering the object's phase. All the methods for the phase recovery are based on the relationship provided in (6.6). Considering this relationship across all bispectrum phase elements, we can reformulate the expression in matrix-vector form,

$$\boldsymbol{\beta} = \mathbf{A}\boldsymbol{\phi}.$$

Here, the data vector $\boldsymbol{\beta}$ is an $m \times 1$ vector of all the accumulated bispectral elements corresponding to m distinct $(\mathbf{u}, \mathbf{v}, \mathbf{u} + \mathbf{v})$ triplets, $\boldsymbol{\phi}$ is the unknown $n \times 1$ dimensional unknown object phase at the cell-centers of the object's Fourier transform, and \mathbf{A} is an $m \times n$ sparse matrix ($m \gg n$) with three non-zeros entries per row, two 1's and one -1 corresponding the sign of the phase elements in (6.6).

From this expression, fitting the phase to the data is formulated as an overdetermined nonlinear least-squares problem. The nonlinearity arises due to the fact that the bispectrum is collected modulo- 2π (or wrapped). One idea then is to unwrap the

phase to obtain a linear least-squares problem, as proposed by Marron *et al.* [44]. We focus on the alternative option of solving the nonlinear least-squares problem with the collected, wrapped bispectrum. Multiple minimization schemes for solving the nonlinear problem have been proposed, and we build on these. The first scheme we look at was proposed by Haniff [32], given by

$$\min_{\phi} \left\{ E_1(\phi) = \frac{1}{2} \|\mathbf{W}^{\frac{1}{2}} \bmod_{2\pi}(\boldsymbol{\beta} - \mathbf{A}\phi)\|_2^2 \right\},$$

where $\mathbf{W}^{\frac{1}{2}} = \text{diag}(\sqrt{w_1}, \dots, \sqrt{w_m})$ is a diagonal matrix containing weights corresponding to the signal-to-noise ratio of the j th collected bispectrum phase element [49]. We compute the gradient and approximate Hessian for this objective function by

$$\begin{aligned} \nabla_{\phi} E_1 &= -\mathbf{A}^{\top} \mathbf{W} (\bmod_{2\pi}(\boldsymbol{\beta} - \mathbf{A}\phi)) \\ \nabla_{\phi}^2 E_1 &= \mathbf{A}^{\top} \mathbf{W} \mathbf{A}, \end{aligned} \tag{6.7}$$

where $\mathbf{W} = \mathbf{W}^{\frac{1}{2}} \mathbf{W}^{\frac{1}{2}}$. Here, the presence of the modulo- 2π operation introduces a number of considerations for the problem. First, the modulo causes a loss of convexity as there are periodic local minima for each phase element every 2π . This makes optimization methods more likely to be caught in one of these local minima and more sensitive to an educated starting guess. Furthermore, $E_1(\phi)$ is non-differentiable at the many periodic jump-continuities where the modulo- 2π misfit wraps from 0 to 2π . In the derivatives above, we have simply ignored the modulus during differentiation, an idea that is used in the literature and has proven to be effective in our work. However, we should remain aware of it during the optimization.

To avoid the issue of non-differentiability, Haniff's paper also proposes an alternative non-linear least-squares scheme that minimizes over the misfit in the object's

phasor [32]. Here, a phasor is defined as the normalized complex exponential of a phase element, i.e., the phase ϕ corresponds to the phasor $\exp(i\phi) = \cos(\phi) + i \sin(\phi)$. The misfit function for this scheme is given by

$$\min_{\phi} \left\{ E_2(\phi) = \frac{1}{2} \left(\|\mathbf{W}^{\frac{1}{2}}(\cos(\boldsymbol{\beta}) - \cos(\mathbf{A}\phi))\|_2^2 + \frac{1}{2} \|\mathbf{W}^{\frac{1}{2}}(\sin(\boldsymbol{\beta}) - \sin(\mathbf{A}\phi))\|_2^2 \right) \right\},$$

where $\mathbf{W}^{\frac{1}{2}}$ is defined as above. Differentiating with respect to the phase, we derive expressions for the gradient and approximate Hessian,

$$\begin{aligned} \nabla_{\phi} E_2 &= \mathbf{A}^{\top} \mathbf{W} (\cos(\boldsymbol{\beta}) \odot \sin(\mathbf{A}\phi) - \sin(\boldsymbol{\beta}) \odot \cos(\mathbf{A}\phi)) \\ \nabla_{\phi}^2 E_2 &= \mathbf{A}^{\top} \mathbf{W} \mathbf{D} \mathbf{A}. \end{aligned} \tag{6.8}$$

Here, \odot denotes the Hadamard product or component-wise multiplication of vectors and \mathbf{D} is a diagonal matrix defined by $\mathbf{D} = \text{diag}(\cos(\boldsymbol{\beta}) \odot \cos(\mathbf{A}\phi) + \sin(\boldsymbol{\beta}) \odot \sin(\mathbf{A}\phi))$. This formulation is still non-convex with periodic local minima every 2π , but it is differentiable everywhere. The addition of \mathbf{D} creates additional computational considerations, namely the approximate Hessian associated with $E_1(\phi)$ is constant, whereas the Hessian for $E_2(\phi)$, the Hessian must be updated at every iteration of the optimization because of the \mathbf{D} matrix.

In both formulations of the phase-matching problem above, the gradients and Hessians are calculated by differentiating with respect to the phase, ϕ . However, the final goal of the optimization is to combine the matched phase with the object's calculated power spectrum to obtain the best quality image possible from the data. Thus, it would be useful if the least-squares optimization for matching the phase took into account the resulting image. To this end, Glindemann and Dainty proposed a different approach to Haniff's first objective function, $E_1(\phi)$. Rather than differ-

entiating with respect to the phase, they proposed considering the phase retrieval as a function of the resultant image, $E_1(\mathbf{o})$ [22]. To do this one must also differentiate with respect to the resultant image. The resultant image can be expressed as $\mathbf{o} = \mathcal{F}^{-1}(|\mathbf{O}(u, v)| \odot \exp[i\phi])$ where \mathcal{F}^{-1} is an inverse 2D Fourier transform and $|\mathbf{O}(u, v)|$ is the object's calculated Fourier modulus. Differentiating with respect to the resultant image can be done as an extension to (6.7) using the chain rule to differentiate the phase with respect to the image. The gradient and Hessian resulting from this approach are, respectively,

$$\begin{aligned}\nabla_{\mathbf{o}} E_1 &= -\frac{\partial \phi^*}{\partial \mathbf{o}} \mathbf{A}^\top \mathbf{W} [\text{mod}_{2\pi}(\boldsymbol{\beta} - \mathbf{A}\phi)] \\ \nabla_{\mathbf{o}}^2 E_1 &= \frac{\partial \phi^*}{\partial \mathbf{o}} \mathbf{A}^\top \mathbf{W} \mathbf{A} \frac{\partial \phi}{\partial \mathbf{o}}.\end{aligned}\tag{6.9}$$

Here, the gradient is nearly the same as (6.7) with the exception of the additional $\frac{\partial \phi}{\partial \mathbf{o}}$ operator. To derive an expression for this operator and its adjoint, we first note that the object's phase ϕ can be expressed as a function of the current image \mathbf{o} by

$$\phi = \arctan\left(\frac{\text{Im}(\mathcal{F}\mathbf{o})}{\text{Re}(\mathcal{F}\mathbf{o})}\right).$$

From this, it is then possible to calculate the adjoint gradient operator $\frac{\partial \phi^*}{\partial \mathbf{o}}$ via a combination of the chain rule and the quotient rule, as done in the appendix of Glindemann and Dainty [22]. The adjoint operator in the generic direction \mathbf{z} is then given by

$$\begin{aligned}\frac{\partial \phi^*}{\partial \mathbf{o}}(\mathbf{z}) &= \frac{\text{Re}(\mathcal{F}\mathbf{o}) \odot \text{Im}(\mathcal{F}\mathbf{z}) - \text{Im}(\mathcal{F}\mathbf{o}) \odot \text{Re}(\mathcal{F}\mathbf{z})}{|\mathcal{F}\mathbf{o}|^2} \\ &= \frac{\text{Im}(\mathcal{F}\mathbf{z} \odot \overline{\mathcal{F}\mathbf{o}})}{\mathcal{F}\mathbf{o} \odot \overline{\mathcal{F}\mathbf{o}}} \\ &= \text{Im}\left(\frac{\mathcal{F}\mathbf{z}}{\mathcal{F}\mathbf{o}}\right).\end{aligned}\tag{6.10}$$

Here, the Hadamard product, the division, and the squared operation are all taken component-wise, while \mathcal{F} again denotes a two dimensional Fourier transform. Some other considerations are important in computation. In order to avoid division by zero, the gradient at indices where the denominator is equal to zero should be set to zero. Also, the Fourier transforms need to be scaled properly according to the implementation used.

When computing the approximate Hessian for the Newton-based optimization, it is also necessary to compute the forward operator of (6.10) in the direction \mathbf{z} . This is most easily done and verified by an adjoint test, and is expressed as

$$\begin{aligned} \frac{\partial \phi}{\partial \mathbf{o}}(\mathbf{z}) &= \text{Im} \left(\mathcal{F} \left[\frac{\text{Re}(\mathcal{F}\mathbf{o}) \odot \mathbf{z}}{|\mathcal{F}\mathbf{o}|^2} \right] \right) - \text{Re} \left(\mathcal{F} \left[\frac{\text{Im}(\mathcal{F}\mathbf{o}) \odot \mathbf{z}}{|\mathcal{F}\mathbf{o}|^2} \right] \right) \\ &= \text{Im} \left(\mathcal{F} \left[\frac{\mathbf{z} \odot \overline{\mathcal{F}\mathbf{o}}}{\mathcal{F}\mathbf{o} \odot \overline{\mathcal{F}\mathbf{o}}} \right] \right) \\ &= \text{Im} \left(\mathcal{F} \left[\frac{\mathbf{z}}{\mathcal{F}\mathbf{o}} \right] \right). \end{aligned}$$

Here again, division, squared operations, and Hadamard products are component-wise operations; indices corresponding to division by zero should be set to zero; and Fourier transforms should be scaled appropriately.

Having derived $\frac{\partial \phi}{\partial \mathbf{o}}$ and its adjoint, we can efficiently evaluate the gradient and Hessian for E_1 with respect to the resulting image in (6.9). This framework can also be applied to differentiate Haniff's second objective function, E_2 , using Glindemann's idea. To our knowledge, this formulation for the minimization has not been explored in the literature and represents a new extension on the ideas above. The gradient

and Hessian of $E_2(\mathbf{o})$ with respect to the resultant image are given by

$$\begin{aligned}\nabla_{\mathbf{o}}E_2 &= \frac{\partial\phi^*}{\partial\mathbf{o}}\mathbf{A}^\top\mathbf{W}[\cos(\boldsymbol{\beta})\odot\sin(\mathbf{A}\boldsymbol{\phi}) - \sin(\boldsymbol{\beta})\odot\cos(\mathbf{A}\boldsymbol{\phi})] \\ \nabla_{\mathbf{o}}^2E_2 &= \frac{\partial\phi^*}{\partial\mathbf{o}}\mathbf{A}^\top\mathbf{W}\mathbf{D}\mathbf{A}\frac{\partial\phi}{\partial\mathbf{o}}.\end{aligned}\tag{6.11}$$

Differentiating both objective functions with respect to the resultant image has a number of possible advantages. First, we remarked above on the non-convexity of both least-squares formulations and the presence of many local minima. Differentiating with respect to the resultant image has the potential to drive convergence to different minima, potentially resulting in improved image quality. Furthermore, differentiating with respect to the image rather than the phase presents the prospect for regularization with respect to the resultant image. This would allow for the enforcement of certain desirable image traits like non-negativity. With this in mind, we look at a potential regularizer.

Regularization

We look at a regularization scheme proposed by Glindemann and Dainty for enforcing non-negativity [22]. This regularization is used with $E_1(\mathbf{o})$ and $E_2(\mathbf{o})$ when differentiated with respect to the resultant image. The regularization introduces a penalty term to discourage negative intensity values in the resultant image and is written as

$$E_+(\mathbf{o}) = \frac{\alpha}{2} \sum_{\gamma} |\mathbf{o}(\gamma)|^2,$$

where γ is the set of indices corresponding to negative image values at the current iteration. The weighting parameter α should be chosen to drive the solution to positive values while still allowing the optimization to effectively match the object's phase to the data bispectrum. Some discussion for the selection of the regularization

parameter will be included in Sect. 6.2. Next, we must differentiate the regularizer; the element-wise first and second derivative are given by

$$\frac{\partial E_+}{\partial \boldsymbol{o}_j} = \begin{cases} \alpha o_j & \text{if } j \in \gamma \\ 0 & \text{else} \end{cases}$$

$$\frac{\partial^2 E_+}{\partial \boldsymbol{o}_j^2} = \begin{cases} \alpha & \text{if } j \in \gamma \\ 0 & \text{else} \end{cases}.$$

These can then be put into vector and sparse, diagonal matrix form for the gradient and Hessian, respectively. Summing them with the gradient and Hessian from (6.9) and (6.11) then follows.

6.2 Numerical Results

To test and compare the objective functions introduced in the previous section, we first simulate the necessary sets of speckle interferometry data [58, 62]. We generate 50 frames of short-exposure data for both the object and the reference star with Fried parameter $D/r_0 = 30$, using different seeds for the random number generator to ensure independence of the randomness of the two data sets. Each star and data frame has dimension 256×256 . The object data is scaled to include $3e6$ photo-events per frame and zero-mean Gaussian noise with standard deviation $\sigma_{rn} = 5$ is added. The star data is scaled to 5000 photo-events per frame.

From this data, we collect the object's power spectrum and phase of the data bispectrum. The phase of the data bispectrum is collected using a Fourier radius of $\frac{256}{4} = 64$ and a second radius of 5. This results in collected bispectrum of length $m = 496,466$. The unknown phase has dimension $n = 256^2$, but we only edit entries

within the Fourier radius leading to $\tilde{n} = 12,852$ variables. Before testing the various optimization schemes, we note once again that the objective functions $E_1(\boldsymbol{\phi})$, $E_2(\boldsymbol{\phi})$, $E_1(\boldsymbol{o})$, and $E_2(\boldsymbol{o})$ all have frequent local minima. As a consequence, the minimization of both objective functions is highly dependent on a good initial guess for the phase. To generate this guess, we use the recursive phase recovery scheme proposed by Tyler [61].

Using the initial phase collected via the recursive algorithm, we solve the phase recovery problem for each objective functions proposed in the previous section using gradient descent, nonlinear conjugate gradient (NCLG), and Gauss–Newton. Gradient descent is included as a first-order benchmark while NLCG has been preferred in previous implementations due to reluctance to implement the large linear system solves using the Hessian that are necessary for Newton-based optimization [49]. Newton-based methods pose the challenge of solving a linear system with the Hessian when choosing a descent direction, but offer the promise of improved convergence. Thus, we consider whether the Hessian solve can be made efficient enough to offset the additional cost associated with the method.

Several choices are made for methods parameters. For all three optimization methods, we use a backtracking Armijo line search with a maximum step length of 1 for the step length parameter. We note that this does not necessarily satisfy the strong Wolfe conditions necessary for a descent direction in NLCG, but in practice it proves sufficient. The results for each minimization scheme on each of our objective functions can be seen in Fig. 6.1. For stopping criteria, we use the relative change in the objective function per iteration and a tolerance of 10^{-5} . That is, if the objective value fails to decrease more than the set tolerance at each iteration, the method was stopped.

From Fig. 6.1, we see that the methods perform as expected in terms of convergence: NLCG outperforms gradient descent and Gauss–Newton is superior to both

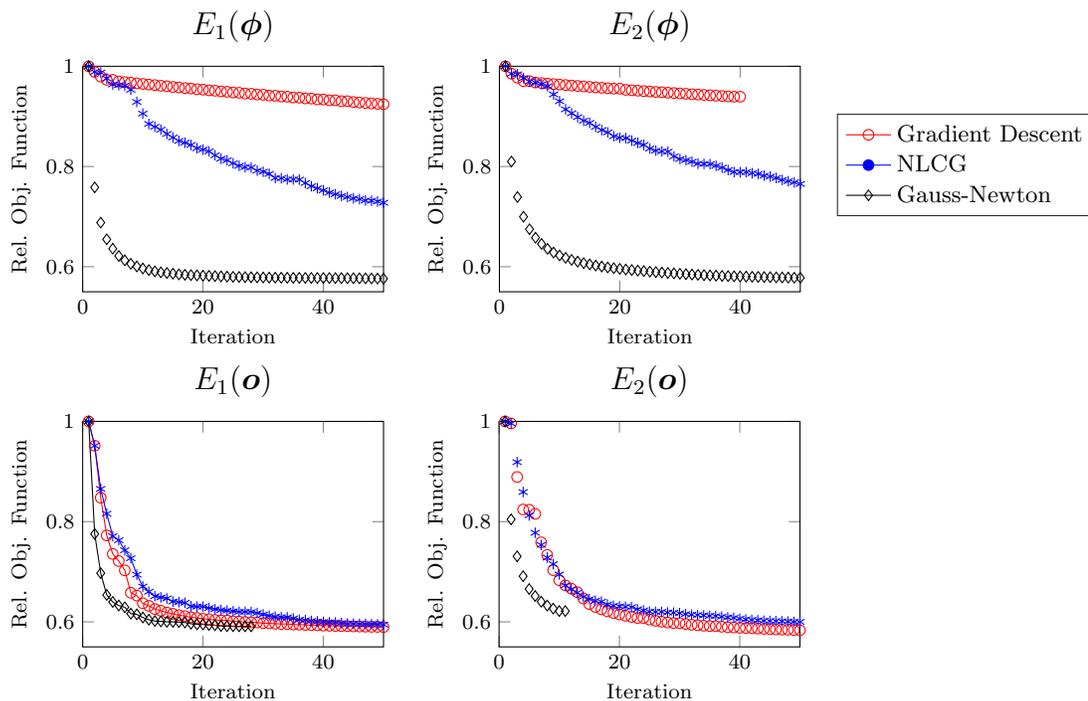


Figure 6.1: The subplots show the convergence of the proposed optimization strategies for matching the object phase ϕ to the phase of the data bispectrum for 50 frames and $D/r_0 = 30$ with stopping tolerance $\text{tol} = 10^{-5}$. Subplots (1, 1) and (1, 2) show, respectively, the results for the objective functions $E_1(\phi)$ and $E_2(\phi)$ optimized with respect to the phase, while subplots (2, 1) and (2, 2) show the results for $E_1(\mathbf{o})$ and $E_2(\mathbf{o})$ when differentiated with respect to the resultant image. When differentiating with respect to the phase, the Gauss–Newton converges faster and to lower minima than the gradient based methods. Differentiating with respect to the image, similar minima are reached but Gauss–Newton meets the convergence criteria in fewer iterations.

in terms of convergence per iteration. These results are more pronounced for $E_1(\phi)$ and $E_2(\phi)$ than for $E_1(\mathbf{o})$ and $E_2(\mathbf{o})$, but the trend holds. However, the results from the figure do not give an indication of the total cost of the iterations in terms of time and computation. If the cost of the Hessian solve in each Gauss–Newton iteration is too costly, the improvement in convergence per iteration may be offset by the expense of each iteration itself. Thus, we must address the cost of the Hessian solve for each objective function before making any comment on the preferability of one method to another.

Handling the Hessian Solve and per Iteration Cost

To approach the Hessian solve for each iteration of Gauss–Newton using $E_1(\phi)$ and $E_2(\phi)$, we first note that the Hessian for $E_1(\phi)$ in (6.7) is independent of ϕ , making it constant for all iterations of the optimization. As such, factorization for a direct, sparse solver is an option because any factorization can be computed once and stored for all iterations of the method. We also recall that due to the windowing for signal-to-noise ratio considerations and the symmetries of real images in Fourier space, the number of phase elements to be recovered for a given image is significantly less than the number of pixels in the image, i.e., $\tilde{n} \ll n$. This means that the $m \times n$ Hessian is highly rank deficient, with many zero columns. Lastly, we note for $E_1(\phi)$, the Hessian is symmetric positive semi-definite because \mathbf{W} has positive entries. Here, the semi-definiteness is a direct result of the rank deficiency.

Bearing these characteristics in mind, we implement the following approach for the Hessian of $E_1(\phi)$. First, we compute and store a symmetric approximate minimum degree permutation to shift the non-zero rows and columns of the Hessian to the upper left-hand corner of the matrix. Next, we extract the resultant sub-matrix corresponding to the non-zero part of the Hessian. This sub-matrix has dimension $\tilde{n} \times \tilde{n}$ and is much smaller than the full Hessian. It is also symmetric positive def-

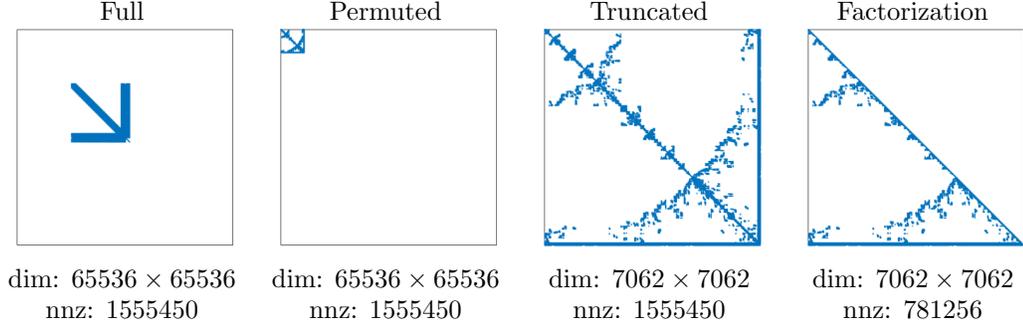


Figure 6.2: This shows the non-zero values for the Hessian $\mathbf{A}^\top \mathbf{W} \mathbf{A}$ and its treatment for the objective functions associated with the Hessians in (6.7) & (6.8) using Matlab’s `spy` command. The Fourier radius for the bispectrum recovery is 64 and the second radius 5 for a 256×256 image. These radii and symmetry limit the number of phase values to be recovered, making the matrix highly singular and sparse. The full Hessian is in the first subplot. A symmetric minimum degree permutation shifts the non-zeros toward the upper left of the matrix in subplot two. In the third subplot, we extract the non-singular sub-matrix in the upper left corner, extracting a truncated Hessian. Finally, we compute an incomplete Cholesky factorization of this truncated Hessian in subplot four, which preserves the sparsity pattern and limits fill in.

inite. As such, we can compute the Cholesky factorization or incomplete Cholesky factorization of the sub-matrix once and store the factors for all subsequent iterations of the method. This reduces the cost of solving the linear system to compute the Gauss–Newton step at each iteration from a potentially $\mathcal{O}(\tilde{n}^3)$ operation to a permutation followed by two smaller $\mathcal{O}(\tilde{n}^2)$ triangular systems, while the $\mathcal{O}(\tilde{n}^3)$ factorization is only done once. Additionally, it ensures that the non-zero components of the direction in the optimization correspond to the phases of elements in the Fourier plane that we wish to recover. In practice, we can save further by using an incomplete Cholesky factorization because it minimizes fill in and maintains the sparsity pattern. This speeds up solving the triangular system while delivering similar results in the optimization. Fig. 6.2 shows the transition for the permutation, truncation, and factorization of the Hessian matrix using MATLAB’s `spy` function.

For the Hessian of $E_2(\phi)$, the strategy used for the Hessian of $E_1(\phi)$ presents a problem: the diagonal matrix \mathbf{D} in the Hessian in (6.8) is dependent on ϕ . Also,

\mathbf{D} may contain non-positive entries causing the extracted permuted matrix to lose both positive definiteness and positive semi-definiteness, thus preventing the use of a Cholesky factorization. To avoid these complications, we omit the matrix \mathbf{D} in practice, making the Hessian for $E_2(\boldsymbol{\phi})$ equal to that of $E_1(\boldsymbol{\phi})$. This then allows us to follow the strategy used above. In practice, this strategy delivers comparable convergence in the optimization while avoiding updates to the factorization and loss of positive definiteness in the permuted sub-matrix (see Fig. 6.1).

The Hessian solves for objective functions $E_1(\boldsymbol{o})$ and $E_2(\boldsymbol{o})$ are more complicated due to the $\frac{\partial \phi}{\partial \boldsymbol{o}}$ operator and its adjoint. These operators are dependent on the resultant image \boldsymbol{o} and must be updated at each iteration. Additionally, the Fourier transforms present in these operators cause a loss of sparsity, meaning that the Hessians for both $E_1(\boldsymbol{o})$ and $E_2(\boldsymbol{o})$ are full matrices of dimension $n \times n$. This makes factorization and direct solvers infeasible. However, the sparse matrix $\mathbf{A}^\top \mathbf{W} \mathbf{A}$ can be computed once offline, and multiplication by $\frac{\partial \phi}{\partial \boldsymbol{o}}$ and its adjoint done efficiently with FFTs. Using this strategy, one can compute matrix-vector products efficiently using these Hessians, so we use MATLAB's `pcg` solver with a low tolerance of 10^{-1} to determine the search direction for the Gauss–Newton step when optimizing $E_1(\boldsymbol{o})$ and $E_2(\boldsymbol{o})$. For $E_2(\boldsymbol{o})$ as for $E_2(\boldsymbol{\phi})$, we omit \mathbf{D} to ensure that the Hessian operator is symmetric positive definite, making it identical to $E_1(\boldsymbol{o})$. This strategy again allows for offline computation of $\mathbf{A}^\top \mathbf{W} \mathbf{A}$. This provides computational savings, and the convergence of the method does not suffer significantly.

In Table 6.1, we compare of the average number of function calls in the line search and the average CPU time per iteration for gradient descent, NLCG, and Gauss–Newton for each of the objective functions. For $E_1(\boldsymbol{\phi})$ and $E_2(\boldsymbol{\phi})$, the Gauss–Newton scheme results in a significant speed up of the method. It converges in fewer iterations, and each iteration is faster than either gradient descent or NLCG. This is not expected for a second-order optimization scheme but is achieved due to a reduced number of

| | | Grad. Descent | NLCG | Gauss-Newton |
|-------------------|------------|---------------|------|--------------|
| $E_1(\phi)$ | LS/Iter. | 14.4 | 14.5 | 1.0 |
| | Sec./Iter. | 0.94 | 0.97 | 0.33 |
| $E_2(\phi)$ | LS/Iter. | 14.4 | 14.5 | 1.0 |
| | Sec./Iter. | 1.52 | 1.63 | 0.27 |
| $E_1(\mathbf{o})$ | LS/Iter. | 16.4 | 15.8 | 1.7 |
| | Sec./Iter. | 1.28 | 1.21 | 1.19 |
| $E_2(\mathbf{o})$ | LS/Iter. | 15.9 | 16.1 | 1.1 |
| | Sec./Iter. | 1.98 | 1.96 | 0.84 |

Table 6.1: This table shows the average number of function calls in the line search per iteration and the average CPU time per iteration for gradient descent, nonlinear conjugate gradient, and Gauss–Newton optimization for each of the four objective functions: $E_1(\phi)$, $E_2(\phi)$, $E_1(\mathbf{o})$, and $E_2(\mathbf{o})$.

function calls in the line search and the efficiency of the Hessian solve due to the strategy outlined above. Some improvement in the comparison for the gradient based methods may be possible using an adaptable maximum step length for the Armijo line search, but the improved convergence rate for the Gauss–Newton method and its scaled step length make it an attractive choice. For $E_1(\mathbf{o})$ and $E_2(\mathbf{o})$, the trends are lessened but remain. The Gauss–Newton method converges in fewer iterations and displays the same advantages in the line search. It reaches similar minima in terms of the relative objective function, although Fig. 6.3 suggests this the relative objective function is not always the best metric for the quality of the resulting image. Omitting \mathbf{D} and computing $\mathbf{A}^\top \mathbf{W} \mathbf{A}$ offline for both objective functions significantly reduces the cost of matrix-vector multiplications using the Hessian, and the reduced number of line search iterations. This makes Gauss–Newton iterations cheaper than gradient descent or NLCG iterations. These results could possibly be improved further by introducing a suitable preconditioner to the conjugate gradient method for calculating the Gauss–Newton step. This represents future work.

Choosing a Regularization Parameter

Another important decision affecting the performance of $E_1(\mathbf{o})$ and $E_2(\mathbf{o})$ is the selection of a regularization parameter α for enforcing non-negativity when using $E_+(\mathbf{o})$. This is complicated by the nonlinearity of the objective functions, which limits the applicability of many standard techniques for regularization parameter selection.

For our tests on this problem, we found using the average mean-squared error to be an effective method of parameter selection. If \mathbf{o}_α is the regularized solution for a given α parameter, we chose the α that minimized the two norm between the \mathbf{o}_α and the true solution \mathbf{o}_{true} . This was tested across a range of α and several different true images to test that the regularization parameter was a realistic choice. We found $\alpha = 100$ to be an effective parameter for all three methods when using $3e6$ photo-events per object data frame and 5000 photo-events per star data frame. Note that in general, α is sensitive to the number of photo-events in the object and star data and should be determined accordingly.

Using the average mean-squared error, we can ensure that the regularization parameter selected is affecting our solution and enforcing non-negativity. While using the true solution to select the regularization parameter is not a realistic expectation for real world data, it does verify that the regularization approach is effective. Finding a method of parameter selection that is independent of knowledge of the true solution represents future work.

Resultant Images

The final goal of the phase recovery is to combine it with the computed object power spectrum to obtain a good resultant image. Below, we display the resultant images from $E_1(\phi)$, $E_2(\phi)$, $E_1(\mathbf{o})$, and $E_2(\mathbf{o})$. For the first two objective functions, the phase is combined with the power spectrum after the iterative methods, where as for

$E_1(\mathbf{o})$ and $E_2(\mathbf{o})$, the power spectrum is included in the initial guess and the resultant images come directly from the iterative method. The resultant images for 50 data frames and $D/r_0 = 30$ can be seen in Fig. 6.3.

From the figure, we can see various results of the image recovery. Combining the power spectrum with the initial guess for the phase makes the object identifiable to the eye. The optimization results for the phase offer varying improvement to the details of the object including sharpening some details and edges. For example, observe the round knob on the satellite’s body and details on the panelling from image to image. The regularization for $E_1(\mathbf{o})$ and $E_2(\mathbf{o})$ helps reduce the ringing around the object, likely due to the introduction of the regularizer. Note that the regularization scheme pushes the negative image values upward toward zeros, but does not impose strict non-negativity. Picking a regularization parameter that balances non-negativity and image quality yields the best results. Larger values for α tended to sacrifice resultant image quality for the sake of forcing background values to be non-negative. Something to this effect can be seen in the image results for NLCG on $E_1(\mathbf{o})$ and $E_2(\mathbf{o})$ where over-regularization compromises image quality. This happens even though the relative error of the objective function is comparable to the other two methods, suggesting this is not always the best metric for evaluating images. Overall, the results for the three methods show that the gains in optimization using Gauss–Newton optimization also result in comparable or better images than the gradient based methods for phase recovery.

We conclude this chapter with some final remarks on the utility of the methods implemented. The image quality obtained from the gradient-based methods and Gauss–Newton were mostly comparable. The main utility of the Newton-based optimization lies in the improved convergence offered by incorporating the Hessian into the optimization. For all four objective functions, the Hessian offers a speed up in terms of iterations to convergence and also cost per iteration when coded efficiently. This

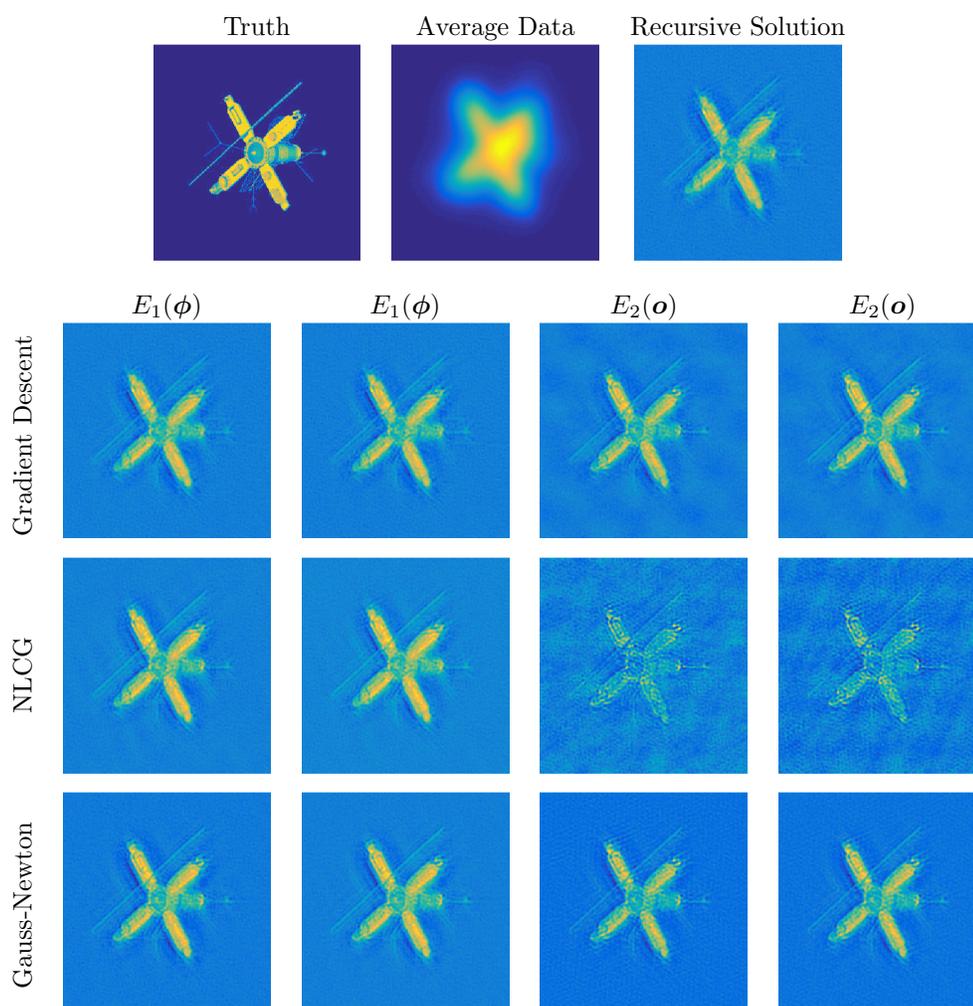


Figure 6.3: This figure shows the necessity of recovering the object's phase from the bispectrum. The first row shows the true solution, average data frame, and initial guess using the phase recovered recursive algorithm. The bottom grid of pictures shows the resultant images for the phases recovered during optimization. From left to right, the columns show the results for $E_1(\phi)$, $E_2(\phi)$, $E_1(\mathbf{o})$, and $E_2(\mathbf{o})$ with the different optimization schemes along the rows. Note that the regularization for $E_1(\mathbf{o})$ and $E_2(\mathbf{o})$ does not enforce complete non-negativity, but by penalizing some non-negativity succeeds in sharpening some of the details of the satellite image and reducing the presence ringing artifacts. This represents a successful compromise in the regularization parameter selection.

makes it an attractive option, especially when a good solution is wanted quickly. An efficient implementation may make bispectral imaging a useful method for obtaining an initial guess for more expensive methods such as multi-frame blind deconvolution if not a stand alone method. Further improvements using $E_1(\mathbf{o})$ and $E_2(\mathbf{o})$ may be possible by introducing an appropriate preconditioner into the Gauss–Newton step solve.

Overall, the work demonstrates the utility of Gauss–Newton for the nonlinear optimization problem of phase recovery in bispectral imaging. In particular, our approach yields speedups in the optimization while not sacrificing image quality compared to gradient-based methods.

Chapter 7

Conclusions and Future Work

Our work focuses on Gauss–Newton optimization for nonlinear inverse problems arising in imaging. Our contributions are concentrated in two main categories. The first is the LAP method for nonlinear inverse problems with coupled sets of variables Ch. 3–5. The second is tailored Gauss–Newton step solvers for the nonlinear, non-convex problem of phase recovery in bispectral imaging in Ch. 6.

LAP exists within a Gauss–Newton optimization framework and optimizes nonlinear problems by linearizing and projecting the problems onto a reduced subspace associated with one of the coupled blocks of variables. LAP is best suited to problems where the subproblem associated with one block of variables is comparatively well-posed. The method also provides a flexible framework for regularization and allows for the imposition of element-wise bound constraints on the solution for all blocks of variables using projected Gauss–Newton. We show this through numerous numerical experiments in Ch. 4 and 5.

We first demonstrate that LAP is applicable to the class of separable least-squares problems that are linear in one block of variables and nonlinear in the other in Ch. 4. Specifically, we show that it outperforms block coordinate descent (BCD) and variable projection (VarPro) for 2D and 3D coupled nonlinear problems in super-resolution

and MRI motion correction. LAP produces better quality optimization results at a lower computational and time cost than either BCD or VarPro for these problems. The experiments for these problems also show LAP's flexibility for regularization using direct, iterative, and hybrid regularization approaches. Lastly, the element-wise bound constraints on the linear image variables for the 2D and 3D super-resolution problems show the benefit of the method compared to VarPro, which does not allow for straightforward constraints on the linear block of variables.

We also test LAP on the fully nonlinear problem of image registration subject to local rigidity constraints. We show this for two and three dimensional examples in Ch. 5. The results show the applicability of LAP for problems that are nonlinear in both blocks of variables. Numerical results for the 2D and 3D registration problems compare LAP with a fully coupled solver for the Gauss–Newton step. While the methods have similar cost, the results show that LAP offers some improvement in the numerical behavior of the optimization. This is particularly evident for the more difficult, 3D problem when no preconditioning is used. Also, these results further demonstrate the regularization flexibility using LAP through the use of a nonlinear, hyperelastic regularizer for this problem.

Future work for LAP should explore the method's utility in other applications. LAP is generally applicable to all linear and nonlinear coupled problems, and it supports various regularization strategies and element-wise bound constraints during the Gauss–Newton optimization. These characteristics make it an attractive option for coupled problems in imaging and in other applications. We are interested in exploring its strengths and weaknesses for specific types of problems. Problems of interest may include blind deconvolution and deep learning. Another way to explore LAP's broader applicability is in depth analysis of the method's convergence properties for certain classes of problems. It would be useful to analyze how nonlinearity in the objective function impacts LAP's performance and to develop some criteria for when

LAP is preferable to alternative approaches for coupled problems.

Phase recovery in bispectral imaging results in a univariate, nonlinear optimization problem. We look at four proposed objective functions from the literature for solving this problem. We develop tailored Gauss–Newton step solvers using approximations to the Hessians, matrix reorderings, and incomplete factorizations. These strategies make second-order optimization schemes efficient. As a result, our tailored solvers are faster in terms of cost per iteration and time to solution than the first-order optimization schemes common in the literature due to clever exploitation of the problem’s structure. In the future, it would be interesting to extend our work to the projected Gauss–Newton framework. This would allow us to enforce non-negativity in the solution image, which is currently enforced via a penalty regularizer. This would then allow us to explore with other types of regularization on the recovered phase or image.

Overall, our work shows the importance of tailored approaches to Gauss–Newton optimization for a selection of nonlinear inverse problems in imaging. LAP provides a flexible framework for the Gauss–Newton optimization of coupled inverse problems. We show this for 2D and 3D examples in super-resolution, MRI motion correction, and constrained image registration. We also develop an efficient Gauss–Newton approach to the nonlinear non-convex problem of phase recovery in bispectral imaging. Both these problems build on and contribute to research for current problems. The work also paves the way for future work. In particular, LAP looks to be applicable to a broad class of coupled problems and should be studied in greater depth in the future.

Appendices

A.1 Linear Interpolation in Higher Dimensions

Piecewise linear interpolation in higher dimensions is used in all the coupled imaging problems in Ch. 4 and 5. In these problems, we want to evaluate a continuous image under transformation. In a discrete setting, this requires us to evaluate the function values (intensities) of the image at transformed coordinates that do not lie directly over the coordinates of known function values (intensities) of a given discrete image. We do this via linear interpolation. It is important to compute the interpolation and its derivatives efficiently as it must be done at every evaluation of the objective function and its derivatives within the optimization. This section outlines the procedure for implementing piecewise linear interpolation and its derivatives in 2 and 3 dimensions using either a matrix-free or sparse matrix implementation. For notation, we use a matrix-oriented axis for figures illustrating the interpolation so that the x -coordinate is vertical from top to bottom and the y -coordinate is horizontal from left to right.

A.1.1 Bilinear Interpolation

We begin by describing piecewise linear interpolation in 2 dimensions, also known as bilinear interpolation, at a single point. We begin with a set of 4 known function values f_{00} , f_{10} , f_{01} , and f_{11} evaluated on a rectangular grid where $f_{ij} = f(x_i, y_j)$ to

simplify notation. The goal of bilinear interpolation is create an interpolating function $\hat{f}(x, y) \approx f(x, y)$ with which we can evaluate at any point $(x, y) \in [x_0, x_1] \times [y_0, y_1]$. We create such a function using two consecutive one dimensional linear interpolations. First, we fix y_0 and y_1 and interpolate along the x -dimension using linear Lagrange polynomials,

$$\begin{aligned}\hat{f}(x, y_0) &= \left(\frac{x_1 - x}{x_1 - x_0}\right) f_{00} + \left(\frac{x - x_0}{x_1 - x_0}\right) f_{10} \\ \hat{f}(x, y_1) &= \left(\frac{x_1 - x}{x_1 - x_0}\right) f_{01} + \left(\frac{x - x_0}{x_1 - x_0}\right) f_{11}.\end{aligned}\tag{1}$$

Next, we interpolate these resulting functions over the y -direction to get

$$\begin{aligned}\hat{f}(x, y) &= \left(\frac{y_1 - y}{y_1 - y_0}\right) \hat{f}(x, y_0) + \left(\frac{y - y_0}{y_1 - y_0}\right) \hat{f}(x, y_1) \\ &= \left(\frac{y_1 - y}{y_1 - y_0}\right) \left(\frac{x_1 - x}{x_1 - x_0}\right) f_{00} + \left(\frac{y_1 - y}{y_1 - y_0}\right) \left(\frac{x - x_0}{x_1 - x_0}\right) f_{10} + \dots \\ &\quad \dots \left(\frac{y - y_0}{y_1 - y_0}\right) \left(\frac{x_1 - x}{x_1 - x_0}\right) f_{01} + \left(\frac{y - y_0}{y_1 - y_0}\right) \left(\frac{x - x_0}{x_1 - x_0}\right) f_{11}.\end{aligned}\tag{2}$$

Rewriting $\Delta x = \frac{x - x_0}{x_1 - x_0}$ and $\Delta y = \frac{y - y_0}{y_1 - y_0}$ and noting that $0 \leq \Delta x, \Delta y \leq 1$, we can express the interpolating function \hat{f} as a function of the displacements Δx and Δy by

$$\hat{f}(\Delta x, \Delta y) = (1 - \Delta x)(1 - \Delta y)f_{00} + \Delta x(1 - \Delta y)f_{10} + (1 - \Delta x)\Delta y f_{01} + \Delta x\Delta y f_{11}.\tag{3}$$

This formulation is equivalent to (2). From this formulation, we can see that the function value $\hat{f}(x, y)$ is simply a sum of the known function values weighted by the distance of (x, y) from each of those points. Note that for the case when (x, y) is

equal to the coordinates of a known function value, $\hat{f}(x, y) = f(x, y)$ interpolates the known, prescribed value. Also, the order of the linear interpolations does not matter for the derivation of the formulas above; one could interpolate in the y direction followed by the x . For a visual representation of bilinear interpolation, we refer the reader to the first subplot in Fig. A.1.1.

Along with the interpolated function value $\hat{f}(x, y)$, we often require its directional derivatives with respect to Δx and Δy . These derivatives are essential for calculating the Jacobians in Ch. 4 and 5. Differentiating (3), the directional derivatives are

$$\begin{aligned}\hat{f}_{\Delta x}(\Delta x, \Delta y) &= (1 - \Delta y)(f_{10} - f_{00}) + \Delta y(f_{11} - f_{01}) \\ \hat{f}_{\Delta y}(\Delta x, \Delta y) &= (1 - \Delta x)(f_{01} - f_{00}) + \Delta x(f_{11} - f_{10}).\end{aligned}\tag{4}$$

A.1.2 Trilinear Interpolation

Using the formulas for bilinear interpolation and its derivatives from the previous section, it is straightforward to extend to the case of 3D linear interpolation by a further linear interpolation along the z -direction. This is known as trilinear interpolation. For a set of eight reference points f_{ijk} , the formula for trilinear interpolation is

$$\begin{aligned}\hat{f}(\Delta x, \Delta y, \Delta z) &= (1 - \Delta x)(1 - \Delta y)(1 - \Delta z)f_{000} + \Delta x(1 - \Delta y)(1 - \Delta z)f_{100} + \\ &\dots (1 - \Delta x)\Delta y(1 - \Delta z)f_{010} + \Delta x\Delta y(1 - \Delta z)f_{110} + \\ &\dots (1 - \Delta x)(1 - \Delta y)\Delta z f_{001} + (1 - \Delta x)\Delta y\Delta z f_{011} + \\ &\dots \Delta x(1 - \Delta y)\Delta z f_{101} + \Delta x\Delta y\Delta z f_{111}.\end{aligned}\tag{5}$$

The weights Δx , Δy , and Δz are defined as in the 2D case with $0 \leq \Delta x, \Delta y, \Delta z \leq 1$.

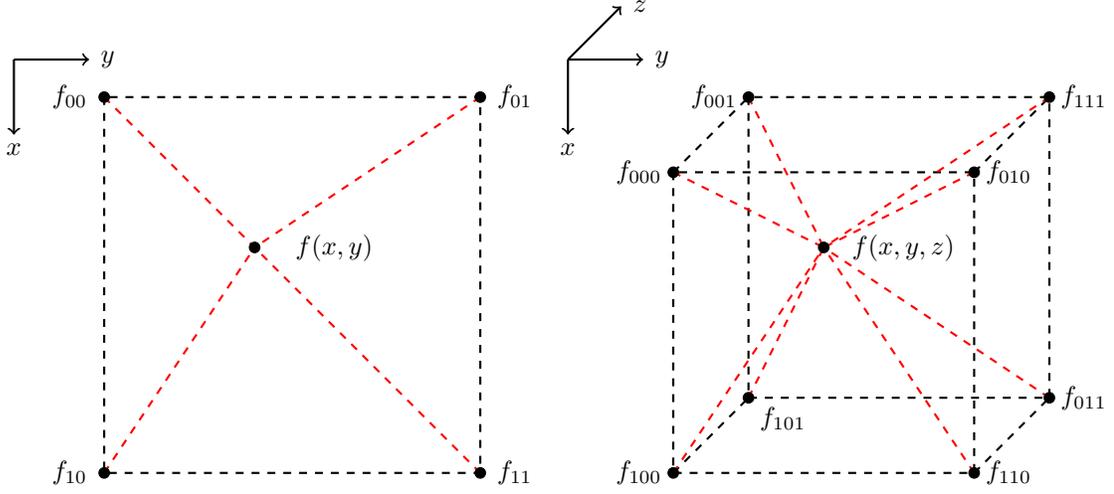


Figure A.1.1: Illustrations for bilinear (*left*) and trilinear (*right*) interpolation. The points f_{ij} and f_{ijk} are the known function values while $\hat{f}(x, y)$ and $\hat{f}(x, y, z)$ are the values to be determined by interpolation in each case. The dashed red lines show the distance of the points to be interpolated from the known function values that correspond to the weights.

Similarly, we also can compute the partial derivatives,

$$\begin{aligned} \hat{f}_{\Delta x}(\Delta x, \Delta y, \Delta z) &= (1 - \Delta y)(1 - \Delta z)(f_{100} - f_{000}) + \Delta y(1 - \Delta z)(f_{110} - f_{010}) + \\ &\quad \dots (1 - \Delta y)\Delta z(f_{011} - f_{001}) + \Delta y\Delta z(f_{111} - f_{011}) \end{aligned}$$

$$\begin{aligned} \hat{f}_{\Delta y}(\Delta x, \Delta y, \Delta z) &= (1 - \Delta x)(1 - \Delta z)(f_{010} - f_{000}) + \Delta x(1 - \Delta z)(f_{110} - f_{100}) + \\ &\quad \dots (1 - \Delta x)\Delta z(f_{011} - f_{001}) + \Delta x\Delta z(f_{111} - f_{101}) \end{aligned}$$

$$\begin{aligned} \hat{f}_{\Delta z}(\Delta x, \Delta y, \Delta z) &= (1 - \Delta x)(1 - \Delta y)(f_{001} - f_{000}) + \Delta x(1 - \Delta y)(f_{101} - f_{100}) + \\ &\quad \dots (1 - \Delta x)\Delta y(f_{011} - f_{010}) + \Delta x\Delta y(f_{111} - f_{110}). \end{aligned}$$

(6)

For a visual representation of trilinear interpolation, we refer the reader to the second subplot of Fig. A.1.1. We also remark that while higher order linear interpolations are beyond the scope of this work, we can continue this pattern of linear interpolation along subsequent dimensions to develop n -dimensional linear interpolation formulas.

A.1.3 Bilinear and Trilinear Interpolation for Multiple Points

The sections above derive linear interpolation formulas for a single point. In practice however, interpolation is needed for a large number of points simultaneously. For example, the coupled imaging problems in Ch. 4 and 5 use the pixel intensities of a reference image as the known data values and require computing a transformed image by interpolating those values at a transformed grid of points. This can be expensive if the dimensions of the data are large and requires additional computational considerations. In this section, we discuss efficient linear interpolation for both matrix-based and matrix-free approaches. We restrict ourselves to the case where the known function values lie on a regular grid. This is the case when interpolating from a reference image, where we consider each given function value to be the intensity of the image evaluated at the cell-center of a pixel. For notation, we change from (x, y, z) coordinates to (x_1, \dots, x_d) to present a general framework for interpolation in d -dimensions.

For our description, we rely heavily on the notation and ideas from FAIR [47]. Let $f(\mathbf{x}) : \Omega \subset \mathbb{R}^d \rightarrow \mathbb{R}$ be a continuous, compactly supported reference image mapping a coordinate vector $\mathbf{x} = (x_1, \dots, x_d) \in \mathbb{R}^d$ to an image intensity $f(\mathbf{x}) \in \mathbb{R}$. Typically, $d = 2$ or 3 , but here we present a general notation for n dimensions. We assume Ω to be rectangular and store it in the vector $\boldsymbol{\omega} = [\omega_1 \dots \omega_{2d}]$ where the k th dimension of Ω is the interval $\omega_{2k} - \omega_{2k-1}$ for $k = 1, \dots, d$. Next, we divide the domain Ω into a regular grid of cells with dimensions $\mathbf{m} = [m_1 \dots m_d]$ where the width of each cell in the k th dimension is given by $\mathbf{h} = [h_1 \dots h_d]$ with $h_k = (\omega_{2k} - \omega_{2k-1})/m_k$ for $k = 1, \dots, d$. We assume that the intensity value of the reference image is known at the cell-centers of the cells in this grid. The coordinates of the cell-centers in the k th dimension can be calculated as $\boldsymbol{\xi}_k \in \mathbb{R}^{m_k}$ where $\boldsymbol{\xi}_k(j) = \omega_{2k-1} + (j_k - 0.5)h_k$ for $j = 1, \dots, m_k$. The coordinates of a any cell-center are then given by $(\boldsymbol{\xi}_1(j_1), \dots, \boldsymbol{\xi}_d(j_d))$ for some indices (j_1, \dots, j_d) , and the known intensity value of the reference image at this cell-center is

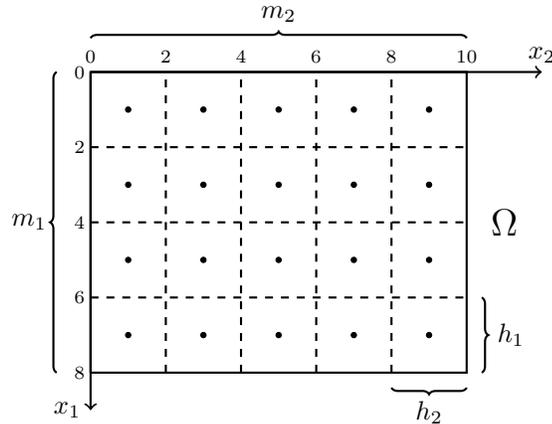


Figure A.1.2: An illustration of a 2D cell-centered grid. The solid lines indicate the compactly supported image domain, $\Omega = [0, 8] \times [0, 10]$. The dashed lines indicate the borders of the cells with $m_1 = 4$ and $m_2 = 5$ and cell widths of $h_1 = h_2 = 2$. The dots indicate the cell-centers where values of the image $f(x_1, x_2)$ are known. The goal of bilinear interpolation is to evaluate the image value for any other point $\mathbf{x} \in \Omega$.

stored as element $\mathbf{f}(j_1, \dots, j_d)$ in the discrete image $\mathbf{f} \in \mathbb{R}^{m_1 \times \dots \times m_d}$. A visualization of a regular, cell-centered grid in two dimensions can be found in Fig. A.1.2.

It follows that to interpolate $f(\mathbf{x})$ for a generic point $\mathbf{x} \in \Omega$, we must be able to efficiently identify the (j_1, \dots, j_d) indices that identify its neighboring cell-centers and the known discrete image intensities \mathbf{f} at those points. We must also evaluate the distance of \mathbf{x} from these cell-centers to determine the weights in the interpolating formula. To this, we introduce following map,

$$\mathbf{x} \rightarrow \mathbf{x}' \quad \text{by} \quad x'_i = (x_i - \omega_{2i})/h_i + 0.5 \quad \text{for} \quad i = 1, \dots, d. \quad (7)$$

This maps the cell-center $(\boldsymbol{\xi}_1(j_1), \dots, \boldsymbol{\xi}_d(j_d))$ to the integer coordinates (j_1, \dots, j_d) , which are the indices referencing the corresponding known function value at the point, $\mathbf{f}(j_1, \dots, j_d)$. For a general point $\mathbf{x} \in \Omega$, the mapped point \mathbf{x}' can be separated into integer and remainder parts,

$$\mathbf{x}' = \mathbf{p} + \mathbf{r} \quad \text{where} \quad \mathbf{p} = \lfloor \mathbf{x}' \rfloor \quad \text{and} \quad \mathbf{r} = \mathbf{x}' - \mathbf{p}. \quad (8)$$

Here, $\lfloor \mathbf{x}' \rfloor$ is the component-wise floor function. The integer vectors \mathbf{p} and $\mathbf{p} + \mathbf{1}$ uniquely identify the indices of the neighboring cell-centers of \mathbf{x}' and, importantly, the indices of the known reference image intensities in the array \mathbf{f} . The remainder vectors \mathbf{r} and $\mathbf{1} - \mathbf{r}$ provide the interpolation weights, i.e. the distance from \mathbf{x} to the cell-centers with known image intensities. Thus, the information in \mathbf{p} and \mathbf{r} provide the information necessary to evaluate the formulas in (3) or (5) and their derivatives. Note that if \mathbf{p} identifies neighboring cell-centers corresponding to cells outside the domain Ω , these points should be ignored or assumed to be zero for interpolation. This is because we assume the reference image to be compactly supported on Ω , i.e., zero boundary conditions. For other applications, additional consideration is required to handle periodic, reflexive, or other appropriate boundary conditions.

The procedure above can be vectorized to determine the indices and weights necessary to interpolate the function $f(x, y)$ (and its derivatives) simultaneously at a large number of points. To introduce this, let $\mathbf{x} = \begin{bmatrix} \mathbf{x}^1 & \mathbf{x}^2 & \mathbf{x}^3 \end{bmatrix} \in \mathbb{R}^{n \times d}$ where the k th column is the vector coordinates in the k th dimension of the n points, and row j is the j th point for interpolation, $([\mathbf{x}]_j^1, [\mathbf{x}]_j^2, [\mathbf{x}]_j^3)$. We then apply the map in (7) row-wise to determine \mathbf{p} and \mathbf{r} for all interpolation points. These values can then be used to evaluate the interpolated image $\hat{\mathbf{f}}$ at the n points described by \mathbf{x} .

For some applications, it is useful to express linear interpolation as a sparse matrix-vector product, $\hat{\mathbf{f}} = \mathbf{T}(\mathbf{x})\mathbf{f}$ where $\mathbf{x} \in \mathbb{R}^{n \times d}$ as above. Here $\mathbf{f} \in \mathbb{R}^m$ is a vectorized version of the array $\mathbf{f} \in \mathbb{R}^{m_1 \times \dots \times m_d}$ with $m = \text{prod}(m_1, \dots, m_d)$. The matrix $\mathbf{T}(\mathbf{x}) \in \mathbb{R}^{n \times m}$ is sparse with the j th row having a maximum of four non-zero entries containing weights in the column indices corresponding to the function values of \mathbf{f} corresponding to the neighboring cell-centers of the point \mathbf{x}_j . The information to construct this matrix is completely contained in the vectors \mathbf{p} and \mathbf{r} defined above. The matrix-vector product $\mathbf{T}(\mathbf{x})\mathbf{f}$ gives $\hat{\mathbf{f}}$, the image intensities evaluated via the interpolation at the n points in \mathbf{x} .

Evaluating the derivatives of the matrix vector expression for interpolation $\mathbf{T}(\mathbf{x})\mathbf{f}$ results in a $1 \times d$ block-diagonal matrix where the diagonal of the k th block for $k = 1, \dots, d$ is the directional derivative with respect to the k th dimension. That is,

$$\nabla_{\mathbf{x}}\mathbf{T}(\mathbf{x})\mathbf{f} = \left[\text{diag}(\nabla_{x^1}\mathbf{T}(\mathbf{x})\mathbf{f}) \quad \dots \quad \text{diag}(\nabla_{x^d}\mathbf{T}(\mathbf{x})\mathbf{f}) \right],$$

where $\nabla_{x^k}\mathbf{T}(\mathbf{x})\mathbf{f} \in \mathbb{R}^n$ is the evaluation of the formula (4) or (6) for the points \mathbf{x} along the k th dimension. As with the interpolation above, these diagonals can be computed efficiently using the map in (7) along with \mathbf{p} and \mathbf{r} .

A.2 Rigid Transformations

In this section, we discuss rigid transformations of points in two and three dimensions. Specifically, we explain how to evaluate points under rotations and shifts parameterized by \mathbf{w} and how to differentiate the resulting points with respect to those motion parameters. These operations are necessary to evaluate the objective functions and Jacobians used for the optimization in Ch. 4 and 5. Much of the notation and material in this appendix is taken from *FAIR*, and we refer the reader there for a more in depth discussion of the concepts described here [47].

We use the following notation. Let

$$\mathbf{y} = \begin{bmatrix} \mathbf{y}^1 \\ \vdots \\ \mathbf{y}^d \end{bmatrix} \in \mathbb{R}^{d \cdot n}$$

be a set of n points in d -dimensions where \mathbf{y}^k is the vector of coordinates in the k th dimension. For the applications in this dissertation, $d = 2$ or 3 , so we limit the discussion to these dimensions. The goal is then to evaluate the new coordinates $\mathbf{y}(\mathbf{w})$, which are the points in \mathbf{y} under the rigid transformation parameterized by the

vector \mathbf{w} . In practice, we want to do this for a large number of points simultaneously as is necessary to transform points in a regular grid in Ch. 4 or the vertices of a tetrahedral mesh in Ch. 5. We separate the discussion into transformations in two and three dimensions.

A.2.1 Rigid Transformations in 2D

For two dimensions, a rigid transformation can be parameterized by three parameters,

$$\mathbf{w} = \begin{bmatrix} \theta & b_1 & b_2 \end{bmatrix}^\top \in \mathbb{R}^3,$$

where θ is a rotation and b_1 and b_2 are shifts in the 1st and 2nd dimensions, respectively. For a single point (y_1, y_2) under transformation \mathbf{w} , we can express the map from $(y_1, y_2) \rightarrow (y_1(\mathbf{w}), y_2(\mathbf{w}))$ by

$$\begin{bmatrix} y_1(\mathbf{w}) \\ y_2(\mathbf{w}) \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}.$$

However, this formula does not extend easily to matrix-vector form to evaluate the transformation of many points simultaneously. For the general array of n points in the vector $\mathbf{y} \in \mathbb{R}^{2n}$, this can be done more efficiently as

$$\mathbf{y}(\mathbf{w}) = \begin{bmatrix} \mathbf{y}^1(\mathbf{w}) \\ \mathbf{y}^2(\mathbf{w}) \end{bmatrix} = \begin{bmatrix} \mathbf{y}^1 & \mathbf{y}^2 & \mathbf{e} & & & \\ & & & \mathbf{y}^1 & \mathbf{y}^2 & \mathbf{e} \end{bmatrix} \begin{bmatrix} \cos \theta \\ -\sin \theta \\ b_1 \\ \sin \theta \\ \cos \theta \\ b_2 \end{bmatrix}, \quad (9)$$

where \mathbf{e} is the vector of ones of length n and $\mathbf{y}(\mathbf{w})$ is the array of points under the transformation \mathbf{w} . Taking the derivative of $\mathbf{y}(\mathbf{w})$ with respect to \mathbf{w} is also straight-

forward using this formulation and is given by the matrix product,

$$\nabla_{\mathbf{w}}\mathbf{y}(\mathbf{w}) = \mathbf{Q}\mathbf{P}, \quad (10)$$

where we define the matrix \mathbf{Q} as

$$\mathbf{Q} = \begin{bmatrix} \mathbf{y}^1 & \mathbf{y}^2 & \mathbf{e} & & & \\ & & & \mathbf{y}^1 & \mathbf{y}^2 & \mathbf{e} \end{bmatrix} \in \mathbb{R}^{2n \times 6}$$

and \mathbf{P} as

$$\mathbf{P} = \nabla_{\mathbf{w}} \begin{bmatrix} \cos \theta \\ -\sin \theta \\ b_1 \\ \sin \theta \\ \cos \theta \\ b_2 \end{bmatrix} = \begin{bmatrix} -\sin \theta & 0 & 0 \\ -\cos \theta & 0 & 0 \\ 0 & 1 & 0 \\ \cos \theta & 0 & 0 \\ -\sin \theta & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \in \mathbb{R}^{6 \times 3}.$$

The formula in (9) provides an efficient, matrix-based framework for evaluating the transformation of an arbitrary number of points under a 2D rigid transformation, while (10) provides the derivative of the transformed points with respect to the transformation parameters.

A.2.2 Rigid Transformations in 3D

Rigid transformations in three dimensions are similar to those in two, but slightly more involved. The transformations are described by 6 parameters,

$$\mathbf{w} = \left[\theta_1 \quad \theta_2 \quad \theta_3 \quad b_1 \quad b_2 \quad b_3 \right]^\top \in \mathbb{R}^6,$$

where θ_k and b_k are a rotation around the k -dimensional axis and a shift in the

k th dimension, respectively. The transformation of a single point (y_1, y_2, y_3) to $(y_1(\mathbf{w}), y_2(\mathbf{w}), y_3(\mathbf{w}))$ is then given by

$$\begin{bmatrix} y_1(\mathbf{w}) \\ y_2(\mathbf{w}) \\ y_3(\mathbf{w}) \end{bmatrix} = \mathbf{R}_3 \mathbf{R}_2 \mathbf{R}_1 \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}.$$

Here, the matrices \mathbf{R}_1 , \mathbf{R}_2 , and \mathbf{R}_3 are 3×3 rotation matrices around the respective axes defined as

$$\mathbf{R}_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \theta_1 & -\sin \theta_1 \\ 0 & \sin \theta_1 & \cos \theta_1 \end{bmatrix} \quad \mathbf{R}_2 = \begin{bmatrix} \cos \theta_2 & 0 & -\sin \theta_2 \\ 0 & 1 & 0 \\ \sin \theta_2 & 0 & \cos \theta_2 \end{bmatrix} \quad \mathbf{R}_3 = \begin{bmatrix} \cos \theta_3 & -\sin \theta_3 & 0 \\ \sin \theta_3 & \cos \theta_3 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

As with two dimensions, we then reformulate the transformation above to make it more efficient for evaluation at n points in the array $\mathbf{y} \in \mathbb{R}^{3n}$. We can write a similar formulation to (9) with the matrix-vector product,

$$\mathbf{y}(\mathbf{w}) = \mathbf{Q}\mathbf{p}, \tag{11}$$

where the matrix \mathbf{Q} is given by

$$\mathbf{Q} = \mathbf{I}_3 \otimes \begin{bmatrix} \mathbf{y}^1 & \mathbf{y}^2 & \mathbf{y}^3 & \mathbf{e} \end{bmatrix} \in \mathbb{R}^{3n \times 12}.$$

Here, \otimes denotes a Kronecker product, and \mathbf{e} is the vector of ones of length n . The entries of the vector \mathbf{p} are given by the matrix product $\mathbf{R}_3 \mathbf{R}_2 \mathbf{R}_1$ and the shifts b_1 , b_2 , and b_3 . The vector is given by

$$\mathbf{p} = \begin{bmatrix} \cos \theta_2 \cos \theta_3 \\ -\sin \theta_1 \sin \theta_2 \cos \theta_3 - \cos \theta_2 \sin \theta_3 \\ -\cos \theta_1 \sin \theta_2 \cos \theta_3 - \sin \theta_1 \sin \theta_3 \\ b_1 \\ \cos \theta_2 \sin \theta_3 \\ -\sin \theta_1 \sin \theta_2 \sin \theta_3 + \cos \theta_1 \cos \theta_3 \\ -\cos \theta_1 \sin \theta_2 \sin \theta_3 - \sin \theta_1 \cos \theta_3 \\ b_2 \\ \sin \theta_2 \\ \sin \theta_1 \cos \theta_2 \\ \cos \theta_1 \cos \theta_2 \\ b_3 \end{bmatrix} \in \mathbb{R}^{12}.$$

As with the 2D case, the derivative of the (11) with respect to the transformation \mathbf{w} is then given by the matrix product

$$\nabla_{\mathbf{w}} \mathbf{y}(\mathbf{w}) = \mathbf{Q} \mathbf{P}, \quad (12)$$

where \mathbf{Q} is defined as above and $\mathbf{P} \in \mathbb{R}^{12 \times 6}$ is defined column-wise as

$$\mathbf{P} = \nabla_{\mathbf{w}} \mathbf{p} = \begin{bmatrix} \frac{d\mathbf{p}}{d\theta_1} & \frac{d\mathbf{p}}{d\theta_2} & \frac{d\mathbf{p}}{d\theta_3} & \frac{d\mathbf{p}}{db_1} & \frac{d\mathbf{p}}{db_2} & \frac{d\mathbf{p}}{db_3} \end{bmatrix}.$$

The formulas above follow the same logic for their computation as in the two dimensional case, although they are slightly more involved. Equations (11) and (12) provide an efficient, matrix-based framework for calculating points under 3D transformations and their derivatives with respect to the transformation parameters.

Bibliography

- [1] P. Batchelor, D. Atkinson, P. Irarrazaval, D. Hill, J. Hajnal, and D. Larkman. Matrix Description of General Motion Correction Applied to Multishot Images. *Magnetic resonance in medicine*, 54(5):1273–1280, 2005.
- [2] A. Beck. *Introduction to Nonlinear Optimization: Theory, Algorithms, and Applications with MATLAB*. SIAM, Philadelphia, PA, 2014.
- [3] Å. Björck. *Numerical methods for least squares problems*. SIAM, Philadelphia, PA, 1996.
- [4] C. Broit. *Optimal registration of deformed images*. PhD thesis, University of Pennsylvania, 1981.
- [5] M. Burger, J. Modersitzki, and L. Ruthotto. A hyperelastic regularization energy for image registration. *SIAM Journal on Scientific Computing*, 35(1):B132–B148, 2013.
- [6] T. F. Chan and J. J. Shen. *Image processing and analysis: variational, PDE, wavelet, and stochastic methods*, volume 94. SIAM, Philadelphia, PA, 2005.
- [7] J. Chung, E. Haber, and J. Nagy. Numerical methods for coupled super-resolution. *Inverse Problems*, 22(4):1261–1272, 2006.

- [8] J. Chung, S. Knepper, and J. G. Nagy. Large-scale inverse problems in imaging. In O. Scherzer, editor, *Handbook of Mathematical Methods in Imaging*, pages 43–86. Springer, New York, 2011.
- [9] J. Chung, J. Nagy, and D. O’Leary. A weighted GCV method for Lanczos hybrid regularization. *Electronic Transactions on Numerical Analysis*, 28:149–167, 2008.
- [10] P. G. Ciarlet. *Mathematical elasticity: Three dimensional elasticity*. Elsevier Science Publishers BV, Amsterdam, 1988.
- [11] L. Cordero-Grande, R. P. Teixeira, E. Hughes, J. Hutter, A. Price, and J. Hajnal. Sensitivity encoding for aligned multishot magnetic resonance reconstruction. *IEEE Transactions on Computational Imaging*, 2(3):266–280, 2016.
- [12] E. de Sturler and M. Kilmer. A regularized Gauss–Newton trust region approach to imaging in diffuse optical tomography. *SIAM Journal on Scientific Computing*, 33(5):3057–3086, 2011.
- [13] P. A. V. den Elsen, E. Pol, and M. A. Viergever. Medical image matching — a review with classification. *IEEE Engineering in Medicine and Biology Magazine*, 12(1):26–39, 1993.
- [14] M. Droske and M. Rumpf. A variational approach to nonrigid morphological image registration. *SIAM Journal on Applied Mathematics*, 64(2):668–687, 2004.
- [15] H. W. Engl, M. Hanke, and A. Neubauer. *Regularization of Inverse Problems*. Kluwer Academic Publishers, Dordrecht, 2000.
- [16] H. W. Engl and P. Kugle. Nonlinear inverse problems: theoretical aspects and some industrial applications. *Mathematics in Industry*, 6:3–47, 2005.
- [17] B. Fischer and J. Modersitzki. Combining landmark and intensity driven registrations. *PAMM*, 3(1):32–35, 2003.

- [18] B. Fischer and J. Modersitzki. Ill-posed medicine — an introduction to image registration. *Inverse Problems*, 24(3):034008, 2008.
- [19] S. Gazzola and J. G. Nagy. Generalized Arnoldi-Tikhonov method for sparse reconstruction. *SIAM J. Sci. Comput.*, 36(2):B225–B247, 2014.
- [20] S. Gazzola and P. Novati. Automatic parameter setting for Arnoldi-Tikhonov methods. *J. Comp. Appl. Math.*, 256(15):180–195, 2014.
- [21] S. Gazzola, P. Novati, and M. R. Russo. Embedded techniques for choosing the parameter in Tikhonov regularization. *Num. Lin. Alg. Appl.*, 21(6):796–812, 2014.
- [22] A. Glindemann and J. C. Dainty. Object fitting to the bispectral phase by using least squares. *JOSA A*, 10(5):1056–1063, 1993.
- [23] T. Goldstein and S. Osher. The Split Bregman method for L1-regularized problems. *SIAM Journal on Imaging Sciences*, 2(2):323–343, 2009.
- [24] G. H. Golub and C. F. V. Loan. *Matrix computations*. JHU Press, Baltimore, MD, 4 edition, 2012.
- [25] G. H. Golub and V. Pereyra. The differentiation of pseudo-inverses and nonlinear least squares problems whose variables separate. *SIAM Journal on Numerical Analysis*, 10(2):413–432, 1973.
- [26] G. H. Golub and V. Pereyra. Separable nonlinear least squares: the variable projection method and its applications. *Inverse Problems*, 19(2):R1, 2003.
- [27] E. Haber. *Computational Methods in Geophysical Electromagnetics*. SIAM, Philadelphia, PA, 2014.

- [28] E. Haber, S. Heldmann, and J. Modersitzki. A computational framework for image-based constrained registration. *Linear Algebra and its Applications*, 431(3-4):459–470, 2009.
- [29] E. Haber and J. Modersitzki. Numerical methods for volume preserving image registration. *Inverse problems*, 20(5):1621–1638, 2004.
- [30] E. Haber and D. Oldenburg. A GCV based method for nonlinear ill-posed problems. *Computational Geosciences*, 4:41–63, 2000.
- [31] J. Hadamard. Sur les problemes aux derive espartielles et leur signification physique. *Bulletin of Princeton University*, 13:1–20, 1902.
- [32] C. A. Haniff. Least-squares Fourier phase estimation from the modulo 2π bispectrum phase. *JOSA A*, 8(1):134–140, 1991.
- [33] P. C. Hansen. *Rank-deficient and discrete ill-posed problems*. SIAM, Philadelphia, PA, 1998.
- [34] R. Hardie, K. Barnard, and E. Armstrong. Joint MAP registration and high-resolution image estimation using a sequence of undersampled images. *IEEE Transactions on Image Processing*, 6(12):1621–1633, 1997.
- [35] J. L. Herring and J. G. Nagy. Fast optimization schemes for phase recovery. Technical report, AMOS Conference Technical Reports, 2015.
- [36] J. L. Herring, J. G. Nagy, and L. Ruthotto. LAP: a linearize and project method for solving inverse problems with coupled variables. 2018. arXiv preprint arXiv:1705.09992 [math.NA].
- [37] M. R. Hestenes and E. Stiefel. *Methods of conjugate gradients for solving linear systems*. NBS, Washington, DC, 1952.

- [38] C. T. Kelley. *Iterative methods for optimization*. SIAM, Philadelphia, PA, 1999.
- [39] K. T. Knox and B. J. Thompson. Recovery of images from atmospherically degraded short-exposure photographs. *Astrophys. J*, 193:L45–L48, 1974.
- [40] L. L. Piegl and W. Tiller. *The NURBS book*. Springer, Berlin, 2012.
- [41] A. Labeyrie. Attainment of diffraction limited resolution in large telescopes by Fourier analyzing speckle patterns in star images. *Astron. Astrophys.*, 6:85–87, 1970.
- [42] A. W. Lohmann, G. Weigelt, and B. Wirnitzer. Speckle masking in astronomy: triple correlation theory and applications. *Applied Optics*, 22(24):4028–4037, 1983.
- [43] J. Macdonald and L. Ruthotto. Improved susceptibility artifact correction of echo planar mri using the alternating direction method of multipliers. *Journal of Mathematical Imaging and Vision*, 2017. pre-print.
- [44] J. C. Marron, P. P. Sanches, and R. C. Sullivan. Unwrapping algorithm for least-squares phase recovery from the modulo 2π bispectrum phase. *JOSA A*, 7(1):14–20, 1990.
- [45] J. Modersitzki. *Numerical methods for image registration*. Oxford University Press on Demand, Oxford, UK, 2004.
- [46] J. Modersitzki. FLIRT with rigidity — image registration with a local non-rigidity penalty. *International Journal of Computer Vision*, 76(2):153–163, 2008.
- [47] J. Modersitzki. *FAIR: flexible algorithms for image registration*, volume 6 of *Fundamentals of Algorithms*. SIAM, Philadelphia, PA, 2009.
- [48] J. Mueller and S. Siltanen. *Linear and nonlinear inverse problems with practical applications*, volume 10. SIAM, Philadelphia, PA, 2012.

- [49] P. Negrete-Regagnon. Practical aspects of image recovery by means of the bispectrum. *JOSA A*, 13(7):1557–1576, 1996.
- [50] J. Nocedal and S. Wright. *Numerical optimization*. Springer, Berlin, 1999.
- [51] D. O’Leary and B. Rust. Variable projection for nonlinear least squares problems. *Computational Optimization and Applications. An International Journal*, 54(3):579–593, 2013.
- [52] C. C. Paige and M. A. Saunders. LSQR: An algorithm for sparse linear equations and sparse least squares. *ACM Transactions on Mathematical Software (TOMS)*, 8(1):43–71, 1982.
- [53] P. Rodríguez and B. Wohlberg. An iteratively reweighted norm algorithm for total variation regularization. In *Proceedings of the 40th Asilomar Conference on Signals, Systems and Computers (ACSSC)*, 2006.
- [54] P. Rodríguez and B. Wohlberg. An efficient algorithm for sparse representations with ℓ^p data fidelity term. In *Proceedings of 4th IEEE Andean Technical Conference (ANDESCON)*, 2008.
- [55] L. Ruthotto. *Hyperelastic image registration*. PhD thesis, University of Lübeck, 2012.
- [56] L. Ruthotto, C. Greif, and J. Modersitzki. A stabilized multigrid solver for hyperelastic image registration. *Numerical Linear Algebra with Applications*, 24(5), 2017.
- [57] Y. Saad. *Iterative methods for sparse linear systems*. SIAM, Philadelphia, PA, 2003.
- [58] J. Schmidt. *Numerical simulation of optical wave propagation with examples in MATLAB*. SPIE Press, Bellingham, WA, 2010.

- [59] D. Sima and S. V. Huffel. Separable nonlinear least squares fitting with linear bound constraints and its application in magnetic resonance spectroscopy data quantification. *Journal of computational and applied mathematics*, 203(1):264–278, 2007.
- [60] P. Thévenaz, T. Blu, and M. Unser. Image interpolation and resampling. *Handbook of medical imaging, processing and analysis*, 1(1):393–420, 2000.
- [61] D. Tyler and K. Schulze. Fast phase spectrum estimation using the parallel part-bispectrum algorithm. *Pub. of the Astron. Soc. of the Pacific*, 116(815):65–76, 2004.
- [62] R. Tyson. *Principles of adaptive optics*. CRC press, San Diego, CA, 2010.
- [63] C. Vogel. *Computational methods for inverse problems*. SIAM, Philadelphia, PA, 2002.
- [64] G. Wahba. *Spline models for observational data*. SIAM, Philadelphia, PA, 1990.
- [65] G. Weigelt. Modified speckle interferometry: speckle masking. *Opt. Commun.*, 21:55–59, 1977.
- [66] B. Wohlberg and P. Rodríguez. An iteratively reweighted norm algorithm for minimization of total variation functionals. *IEEE Signal Processing Letters*, 14:948–951, 2007.
- [67] I. Yanovsky, C. L. Guyader, A. Leow, A. Toga, P. Thompson, and L. Vese. Unbiased volumetric registration via nonlinear elastic regularization. In *2nd MICCAI workshop on mathematical foundations of computational anatomy*, 2008.