Applications of Automatic Differentiation in Image Registration

Warin Watson[†], Cash Cherry[‡], and Rachelle Lang[§]

Project advisor: Lars Ruthotto[¶]

Abstract. We demonstrate that automatic differentiation (AD), which has become commonly available in machine learning frameworks, is an efficient way to explore ideas that lead to algorithmic improvement in multi-scale affine image registration and affine super-resolution problems. In our first experiment on multi-scale registration, we implement an ODE predictor-corrector method involving a derivative with respect to the scale parameter and the Hessian of an image registration objective function, both of which would be difficult to compute without AD. Our findings indicate that exact Hessians are necessary for the method to provide any benefits over a traditional multi-scale method; a Gauss-Newton Hessian approximation fails to provide such benefits. In our second experiment, we implement a variable projected Gauss-Newton method for super-resolution and use AD to differentiate through the iteratively computed projection, a method previously unaddressed in the literature. We show that Jacobians obtained without differentiating through the projection are poor approximations to the true Jacobians of the variable projected forward map and explore the performance of other approximations in the problem of super-resolution. By addressing these problems, this work contributes to the application of AD in image registration and sets a precedent for further use of machine learning tools in this field.

1. Introduction. In the context of medical imaging, image registration is the problem of finding a reasonable transformation $\vec{y}$ to align a template image $\mathcal{T}$ with a reference image $\mathcal{R}$. In the optimization framework, we find a transformation by minimizing the sum of a distance function $\mathcal{D}[\mathcal{T} \circ \vec{y}, \mathcal{R}]$ and a regularization term [15]. Approaches to find optimal transformations in a parameterized set of admissible transformations can be found either by numerical optimization techniques, or by solving a nonlinear PDE derived from the optimality condition [7]. Our strategy, following [15], is to parameterize the transformations by a vector and numerically solve for the optimal transformation as a finite-dimensional unconstrained optimization problem.

The problem of super-resolution, which we properly introduce in §4, seeks to reconstruct an unknown high resolution reference image from a sequence of unregistered low resolution templates [3]. Solution techniques for this problem implement on algorithms for image registration while simultaneously reconstructing the high resolution reference image. We use the same strategy as in registration to solve the super-resolution problem by using a finite dimensional approximation to the problem. However, due to the simultaneous reconstruction of the reference image and the registration of the templates, the objective has a separable structure amenable to a variable projection [9] approach.

Leveraging the PyTorch [18] library, we consider two novel applications of automatic differentiation (AD) to this framework. We apply a predictor-corrector method to perform multi-scale image registration and improve existing variable projection methodology to solve

[†]Department of Mathematics and Statistics, Colorado Mesa University (wdwatson2@mavs.coloradomesa.edu).
[‡]Department of Applied Mathematics and Statistics, Colorado School of Mines (ccherry@mines.edu).
[§]Department of Mathematics, University of Wisconsin-Madison (rklang@wisc.edu).
[¶]Department of Mathematics, Emory University

a super-resolution problem. The prediction step of the predictor-corrector method requires a derivative with respect to a scaling parameter in the interpolation, and variable projection involves differentiation through an inexact iterative least squares solve. Much of the theory for these problems exists in the literature, but perhaps due to the difficulty of some required derivatives, they do not exist in standard image registration packages like FAIR [15]. The code and data required generate the figures used in this text and to reproduce results can be found at our GitHub repository [22].

2. Image Registration Background.

We provide background on the optimization setting used in our work, and the capabilities of Automatic Differentiation.

2.1. Numerical Optimization Framework.

Assume our images are supported on a box-shaped subset $\Omega \subset \mathbb{R}^d$ for $d = 2$ or 3. We consider an enumerated set $\boldsymbol{\omega} = \{\vec{x}_1, \ldots, \vec{x}_n\} \subset \Omega$ of points with even spacing h_x and h_y in each dimension, on which we have evaluations of the reference and template images. (We do *not* have evaluations for points not on the grid, which we will have to address.) Using this data, we would like to be able to compute the distance $\mathcal{D}[\mathcal{T} \circ \vec{y}, \mathcal{R}]$ of our choice, which in this work will be the least-squares distance metric

$$\mathcal{D}[\mathcal{T} \circ \vec{y}, \mathcal{R}] = \int_\Omega (\mathcal{T}(\vec{y}(\vec{x})) - \mathcal{R}(\vec{x}))^2 dV. \tag{2.1}$$

As an approximation to the integral (2.1), we use the quadrature rule $\|\mathcal{T}(\vec{y}(\boldsymbol{\omega})) - \mathcal{R}(\boldsymbol{\omega})\|_2^2 h_x h_y$, where $\mathcal{T}(\vec{y}(\boldsymbol{\omega}))$ and $\mathcal{R}(\boldsymbol{\omega})$ are taken to be vectors of evaluations on $\boldsymbol{\omega}$ of $\mathcal{T} \circ \vec{y}$ and $\mathcal{R}$, respectively. To compute the evaluations $\mathcal{T}(\vec{y}(\boldsymbol{\omega}))$, we must evaluate the template image at points $\vec{y}(\vec{x}_i)$ which will usually not be contained in the grid $\boldsymbol{\omega}$. Further, as we want to use first and second order optimization techniques to minimize a function containing this term, we need a twice continuously differentiable interpolant. To accomplish this, we use the spline interpolation scheme as described in [15]. (This hypothetically allows for *exact* quadrature of the interpolants; however, it would be more difficult to code and it's not clear it would make a difference in the registration quality.)

The final thing to discretize is the transformation $\vec{y}$. The most expressive "non-parametric" approach would be to solve for every value in $\vec{y}(\boldsymbol{\omega})$ separately. However, to avoid such a high dimensional optimization problem, we use a parameterization in many fewer parameters $\mathbf{w} \in \mathbb{R}^p$, henceforth writing $\vec{y}(\vec{\cdot}; \mathbf{w})$. A number of standard parameterizations are covered in [15]. In this work, we consider only affine transformations for simplicity. Our goal is to evaluate and demonstrate the behavior of the proposed methods, rather than to solve difficult image registration problems.

Our discretized image registration loss function has the general form

$$J(\mathbf{w}) = \|\mathcal{T}(\vec{y}(\boldsymbol{\omega}; \mathbf{w})) - \mathcal{R}(\boldsymbol{\omega})\|_2^2 h_x h_y + \lambda^2 S(\mathbf{w}), \tag{2.2}$$

where $S(\mathbf{w})$ is a regularization term, and $\lambda > 0$ is the corresponding regularization parameter. Note that cubic splines are twice continuously differentiable, so the least squares distance $\|\mathcal{T}(\vec{y}(\boldsymbol{\omega}; \mathbf{w})) - \mathcal{R}(\boldsymbol{\omega})\|_2^2 h_x h_y$ is twice continuously differentiable with respect to $\mathbf{w}$ as long as $\vec{y}(\boldsymbol{\omega}; \mathbf{w})$ is. Note also that the third derivative of a cubic spline with fixed data exists and is bounded *almost everywhere* (as cubic splines are piecewise cubic polynomials on intervals);

so as long as $\vec{y}(\boldsymbol{\omega}; \mathbf{w})$ also has bounded third derivatives with respect to $\mathbf{w}$, the Hessian of the least squares loss is also Lipschitz continuous. If we furthermore choose regularization term satisfying this condition, then (2.2) is twice continuously differentiable function on $\mathbb{R}^p$ with a Lipschitz continuous Hessian, thus we can rigorously expect Newton's method, Gauss-Newton, and gradient descent to converge to local minimizers with their respective rates of convergence [17].

2.2. Automatic Differentiation. In the past, a major difficulty in implementing first and second order numerical optimization schemes on complicated objective functions has been the analytic computation of derivatives. However, with automatic differentiation (AD), the implementation of such methods are simplified by no longer having to hand code analytical derivatives, which is known to be time-consuming and error-prone [14].

To demonstrate the practical efficiency of automatic differentiation (AD), Table 1 reports average evaluation times for computing various derivatives under two transformation models: an affine transformation with 6 parameters, and a neural ODE-based transformation with 50 parameters [1, 20, 23].

We define the *forward function* as the map $\mathbf{w} \mapsto \mathcal{T}(\vec{y}(\boldsymbol{\omega}; \mathbf{w}))$, which evaluates the template image at coordinates determined by $\mathbf{w}$, prior to loss computation. The full loss function $J(\mathbf{w})$ follows (2.2), incorporating squared differences and quadrature. All derivatives are computed using Functorch [12], which provides composable AD transforms similar to those in JAX [19].

While the theoretical cost of computing gradients and Hessians scales as $\mathcal{O}(n)$ and $\mathcal{O}(n^2)$, respectively, modern AD frameworks such as Functorch exploit graph-level optimizations, vectorization, and batched computation to reduce overhead [12]. We observe that while the gradient and Jacobian costs scale moderately, the Hessian cost increases substantially for the 50-parameter case. This supports the practical distinction in cost between Newton and quasi-Newton methods for higher-dimensional transformations.

Table 1

Empirical cost of computing derivatives using AD. "Factor" is the ratio of runtime relative to the corresponding loss or forward function evaluation. For example, computing the gradient took 2.5× the time of a loss evaluation in the affine case. Each measurement was averaged over 100 runs. Reported values are mean runtime (μ) ± standard deviation (σ) in milliseconds.

	6 parameters		50 parameters	
	Time (ms) ($\mu \pm \sigma$)	Factor	Time (ms) ($\mu \pm \sigma$)	Factor
Loss Function	2.171 ± 0.259	1×	5.012 ± 2.168	1×
Gradient	5.422 ± 0.409	2.5×	13.65 ± 1.018	2.7×
Hessian	12.44 ± 1.798	5.7×	120.0 ± 8.737	24×
Forward Function	1.054 ± 0.076	1×	3.204 ± 0.149	1×
Jacobian	11.59 ± 1.518	11×	54.31 ± 3.240	17×

Many (real-valued) linear operators associated with these problems (particularly for the Super Resolution problem) are cheap to apply to a given vector, but expensive to store explicitly. For the purpose of solving normal equations associated with them, it is necessary to compute products with the adjoints of these operators. This is done easily using an explicitly

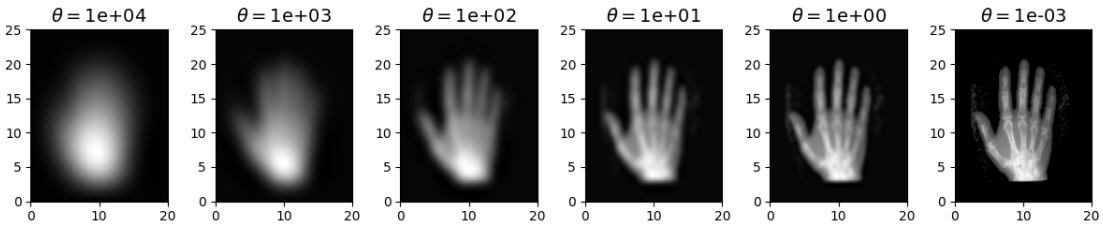

Figure 1. *$\theta = 0$ corresponds to a regularly interpolated image with no blurring, while increasing θ increases the amount of blurring.*

stored matrix form by taking the transpose, but is not straightforward when we cannot store the matrix.

Forward-mode automatic differentiation computes the derivative $\frac{\partial f(\mathbf{x}+h\mathbf{v})}{\partial h}$ with respect to any one dimensional perturbation $\mathbf{v}$ of the input space at the cost of a single forward evaluation. For a matrix-vector product $f(\mathbf{x}) = \mathsf{A}\mathbf{x}$, such a derivative gives the product $\mathbf{v}^\top \mathsf{A}$, for which $\mathsf{A}^\top \mathbf{v} = (\mathbf{v}^\top \mathsf{A})^\top$, the desired adjoint-vector product. Crucially, the matrix A need not be explicitly constructed, as long as the function for computing its action is written in an automatically differentiable way.

3. Multi-scale Methods.

One of the main difficulties of solving image registration problems is that the objective functions (2.2) resulting from real images are often non-convex and contain many local minima due to fine-scale details. Thus, by solving a very down-sampled or blurred registration problem, the minimum obtained is closer to the global minimizer of the desired problem [15]. This leads to an iterative approach, where the solutions to easier versions of the registration problem are iteratively used as initializations for harder versions. There are two such commonly used approaches in registration: The *multi-level* approach (which we don't consider in this work) down-samples the images, and the *multi-scale* approach blurs the images. Figure 1 illustrates the blurring effect that the *scaling parameter*, θ, has on the interpolation of an image. For further explanation and examples of the multi-level and multi-scale techniques, the reader is directed to [15].

An ODE, derived in Section 3.1, describes how the optimal set of transformation parameters changes with respect to θ. One could follow the path of a minimizer from a large value of θ down to a small value of θ purely by numerically solving the ODE, but in our experience, this approach is computationally expensive. Instead, we opt for a predictor-corrector method, where the ODE is used to refine the initial guess at each scale, and local minimization is employed to correct errors at that scale. We distinguish these multi-scale methods from traditional multi-scale methods due to the utilization of the ODE. Additionally, we distinguish these methods from homotopy methods [5] since the scaling is applied to the interpolation of the images rather than directly to the objective function.

3.1. Derivation of Multi-scale ODE. Consider a minimizer $\mathbf{w}^*$ for a fixed θ image registration problem, defined by:

$$\mathbf{w}^*(\theta) \in \arg\min_{\mathbf{w}\in\mathbb{R}^p} J(\mathbf{w};\theta).$$

As J is twice continuously differentiable, and $\mathbf{w}^*$ is a local minimizer, it must satisfy the first-order optimality condition

$$\nabla_{\mathbf{w}} J(\mathbf{w}^*(\theta);\theta) = 0. \tag{3.1}$$

Differentiating (3.1) with respect to θ,

$$\frac{d}{d\theta}\nabla_{\mathbf{w}} J(\mathbf{w}^*(\theta);\theta) = \nabla^2_{\mathbf{w}} J(\mathbf{w}^*(\theta);\theta)\frac{d\mathbf{w}^*}{d\theta} + \frac{d}{d\theta}\nabla_{\mathbf{w}} J(\mathbf{w}^*(\theta);\theta) = 0.$$

It follows that

$$\frac{d}{d\theta}\mathbf{w}^*(\theta) = -\left(\nabla^2_{\mathbf{w}} J(\mathbf{w}^*(\theta);\theta)\right)^{-1}\left(\frac{d}{d\theta}\nabla_{\mathbf{w}} J(\mathbf{w}^*(\theta);\theta)\right). \tag{3.2}$$

Each step of this ODE solve requires an inverse Hessian-vector product, so it is as feasible to implement as Newton's method for optimization.

3.2. Predictor-Corrector Method. The approach involves taking larger steps in θ when solving (3.2) (i.e., using coarser discretizations for the ODE). The error introduced by these larger steps is then corrected through local minimization. In practice, Newton's method for the minimization provides rapid local convergence [17], and allows us the ability to re-use computed Hessians when taking a step in the ODE solve.

To take a step from a coarse scaling parameter θ_n to a finer scaling parameter $\theta_{n+1} < \theta_n$ starting from a minimizer $\mathbf{w}^*(\theta_{n+1})$, a step

$$\mathbf{w}^*_{\text{pred}} = \mathbf{w}^*(\theta_n) + (\theta_n - \theta_{n+1})\left(\frac{d}{d\theta}\mathbf{w}^*(\theta_n)\right) \tag{3.3}$$

of forward Euler's method is taken to predict the location of $\mathbf{w}^*(\theta_{n+1})$, and is then corrected by a local minimization procedure to obtain the true $\mathbf{w}^*(\theta_{n+1})$.

Even with the predictor-corrector method, negative curvature in the local minimization problem still poses a problem. The forward Euler predictor step (3.3) explicitly uses the Hessian inverse, which can lead to computational instability if the Hessian is indefinite. Pure Newton steps also struggle in this scenario, since directly using indefinite Hessians can produce non-descent directions. To address this, we opt for using SciPy's implementation of an exact trust-region Newton solver [4] for the local minimization problem, which avoids explicit Hessian inversion by solving a trust-region subproblem through eigen-decomposition [21]. A line-search Newton method is another viable alternative [17]. After convergence of the local minimization, we reuse the Hessian computed at the accepted iterate in the subsequent forward Euler predictor step, avoiding additional Hessian evaluations.

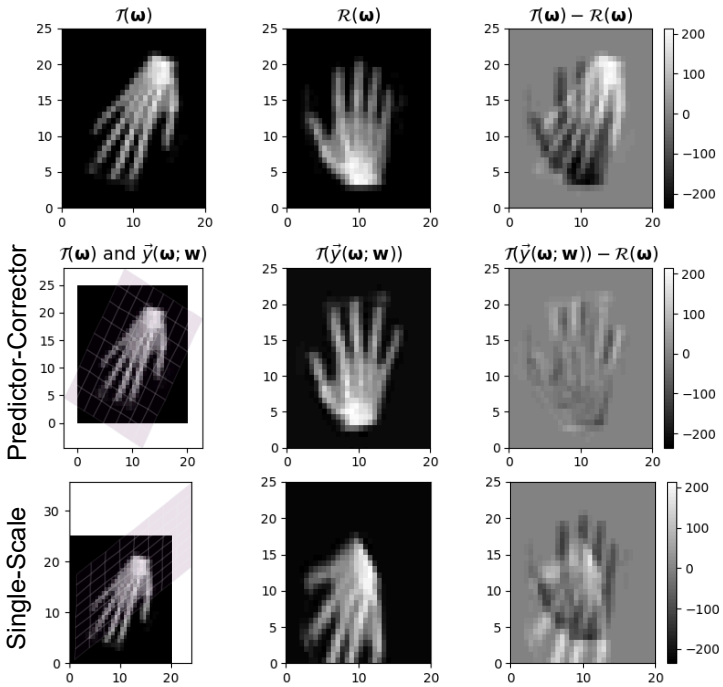

Figure 2. *Template and Reference are shown in the first row. The predictor-corrector method is used to register the images in the second row. Attempting to register at the finest scale alone results in a bad local minimizer as shown in the third row.*

Note that the predictor-corrector method reduces to a traditional multi-scale method when the forward Euler step of the ODE (3.3) is not taken. This suggests that to prefer the predictor-corrector method over the traditional multi-scale method, you would need to ensure the cost of computing (3.3) does not outweigh the cost of performing minimization to get the same $\mathbf{w}^*$. This would vary based on the problem and behavior of the optimization algorithm and its implementation.

3.3. Results. Two results are shown in this section. First, the predictor-corrector method is shown to work for a non-regularized problem where registering from a single scale does not. This motivates the method as having a similar use case as the traditional multi-scale method. Then, for the predictor-corrector method, the prediction is shown to fail when using approximated Hessians via Gauss Newton, but succeeds when using Hessians computed using AD, which are exact with respect to the discretized objective (2.2). An implementation of Levenberg-Marquardt is used for Gauss Newton, while SciPy's Exact Trust Region [21] is used for Newton.

3.3.1. Single Scale vs Predictor-Corrector. Consider an affine, non-regularized image registration problem with template and reference shown in the first row of Figure 2. As shown in the figure, some image registration problems benefit from using the predictor-corrector method, as opposed to optimizing from a single scale, in order to avoid getting stuck in a local minimum.

3.3.2. Approximated Hessian vs Exact Hessian. Gauss-Newton is often implemented over a Newton method to optimize non-linear least squares problems as a way to avoid expensive and/or complicated Hessian computations. Let us define the residual function as $r(\mathbf{w}) = \mathcal{T}(\vec{y}(\boldsymbol{\omega}; \mathbf{w})) - \mathcal{R}(\boldsymbol{\omega})$, and consider the un-regularized least squares loss function

$$J(\mathbf{w}) = \|r(\mathbf{w})\|_2^2.$$

The exact Hessian of such a function is given by

$$\nabla^2 J(\mathbf{w}) = \nabla r(\mathbf{w})^\top \nabla r(\mathbf{w}) + \sum_{i=1}^{n} r_i(x) \nabla^2 r_i(x),$$

where $\nabla r(\mathbf{w})$ is the Jacobian of $r(\mathbf{w})$. Utilizing the exact Hessian in Newton's method yields a quadratic rate of convergence, but can be expensive. At the cost of relaxing the guarantee to only sub-linear convergence, Gauss Newton avoids the computational expense of computing second derivatives by approximating the Hessian using solely the term $\nabla r(\mathbf{w})^\top \nabla r(\mathbf{w})$. However, the method can approach quadratic convergence and often does in practice [17]. With AD, both methods are simple to implement, though the computational expense of computing the Hessian for Newton's method is still greater.

To evaluate the effectiveness of the prediction at each scale, i.e., the step of forward Euler (3.3) , we compare the loss and gradient at the predicted point to the loss and gradient without the prediction. This comparison effectively compares the predictor-corrector method with the traditional multi-scale method. We define the relative loss difference as $(J(\mathbf{w}^*) - J(\mathbf{w}^*_{\text{pred}}))/J(\mathbf{w}^*)$, and the relative gradient norm difference as $(\|\nabla J(\mathbf{w}^*)\| - \|\nabla J(\mathbf{w}^*_{\text{pred}})\|)/\|\nabla J(\mathbf{w}^*)\|$ to give measures of how helpful the predictions are.

This experiment is illustrated in Figure 3. A positive value indicates a successful reduction in loss or gradient, whereas a negative value indicates an increase. The results show that using exact Hessians via Newton's method tends to decrease the loss and gradient, while using approximated Hessians via Gauss-Newton tends to increase the loss and gradient. Therefore, using exact Hessians is necessary for the predictor-corrector method to provide any benefits over a traditional multi-scale method.

4. Coupled Methods for Super-Resolution. We follow the framework developed in [3]. Interpolation of the reference is *linear* in the reference intensity vector $\mathbf{f}^{(0)}$, thus, we can mathematically model our transformations $\vec{y}^{(j)}$ using matrices $\mathsf{I}^{(j)}(\mathbf{w})$. The notation I with no argument is used for the identity matrix, which is consistent with viewing the identity as an application of a trivial transformation. Note that the interpolation matrix is *non-linear* in its dependence on the transformation parameter vector $\mathbf{w}$. Our high resolution transformed templates $\mathbf{f}^{(j)} \in \mathbb{R}^n, j = 1, \ldots, q$ are thus given by the transformations $\mathsf{I}^{(j)}(\mathbf{w})\mathbf{f}^{(0)}$ of an initial reference image $\mathbf{f}^{(0)}$. Our observed template images $\mathbf{d}^{(j)} \in \mathbb{R}^m, j = 0, 1 \ldots, q$ are modeled as $\mathbf{d}^{(j)} = \mathsf{K}\mathbf{f}^{(j)}$, where $\mathsf{K} \in \mathbb{R}^{m \times n}$ is a linear operator which lowers the image resolution ($m < n$), in our case by pooling, that is, averaging the pixels in $k \times k$ regions to reduce resolution by k times in each dimension. Note that our modeling assumes that the template $\mathbf{d}^{(0)}$ is registered with the unknown reference $\mathbf{f}^{(0)}$, as the data can only ever constrain $\mathbf{f}^{(0)}$ up to a transformation; registering $\mathbf{d}^{(0)}$ with $\mathbf{f}^{(0)}$ eliminates this non-uniqueness and saves some

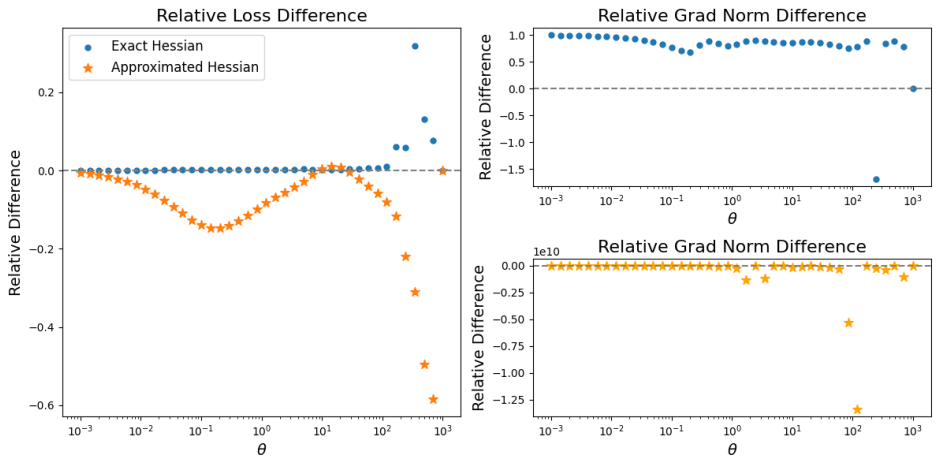

Figure 3. *Relative loss and gradient norm, both plotted against θ on a logarithmic scale. Exact Hessians are used for blue, approximated Hessians are used for orange. All relative gradient norm difference values using approximated Hessians are less than -31000.*

computational overhead. We further let $h_x^{\mathbf{f}}, h_y^{\mathbf{f}}, h_x^{\mathbf{d}}$ and $h_y^{\mathbf{d}}$ be the corresponding grid sizes for the template and reference grids.

In the problem of super-resolution, we aim to reconstruct a high-resolution reference image, $\mathbf{f}^{(0)}$ from a vector of low resolution template images $\mathbf{d} = [{\mathbf{d}^{(0)}}^\top, \ldots, {\mathbf{d}^{(q)}}^\top]^\top$, which are assumed to be unregistered, down-sampled versions of $\mathbf{f}^{(0)}$. Our aim in super-resolution is to register multiple templates $\mathbf{d}^{(j)}$ to an unknown reference $\mathbf{f}^{(0)}$, while simultaneously reconstructing $\mathbf{f}^{(0)}$ from the unregistered templates.

4.1. Super-Resolution Framework. Representing the template intensities in a single vector $\mathbf{d}$, our full model for the super resolution data is given by

$$\mathbf{d} = (\mathsf{I} \otimes \mathsf{K})\mathsf{I}(\mathbf{w})\mathbf{f}^{(0)}, \qquad \mathsf{I}(\mathbf{w}) = \begin{bmatrix} \mathsf{I} \\ \mathsf{I}^{(1)}(\mathbf{w}) \\ \vdots \\ \mathsf{I}^{(q)}(\mathbf{w}) \end{bmatrix}, \qquad \mathbf{d} = \begin{bmatrix} \mathbf{d}^{(0)} \\ \vdots \\ \mathbf{d}^{(q)} \end{bmatrix}.$$

We could approach this as a regularized least squares problem in the two variables $\mathbf{f}^{(0)}$ and $\mathbf{w}$:

$$J(\mathbf{w}, \mathbf{f}^{(0)}) = \|(\mathsf{I} \otimes \mathsf{K})\mathsf{I}(\mathbf{w})\mathbf{f}^{(0)} - \mathbf{d}\|^2 h_x^{\mathbf{d}} h_y^{\mathbf{d}} + \lambda_{\mathbf{f}}^2 S_{\mathbf{f}^{(0)}}(\mathbf{f}^{(0)}).$$

As before, we don't regularize the transformation parameters $\mathbf{w}$. Our regularization term on the reference image $\mathbf{f}^{(0)}$ is a finite difference approximation to the semi-norm $\int_\Omega \|\nabla \mathcal{R}^{(0)}(\vec{x})\|^2 dV$, where $\mathcal{R}^{(0)}$ denotes the continuous reference image approximated by the vector $\mathbf{f}^{(0)}$. This is given by

$$S_{\mathbf{f}^{(0)}}(\mathbf{f}^{(0)}) = \|\mathsf{L}\mathbf{f}^{(0)}\|^2 h_x^{\mathbf{f}} h_y^{\mathbf{f}}, \qquad \mathsf{L} = \begin{bmatrix} h_x^{\mathbf{f}} \mathsf{D} \otimes \mathsf{I} \\ \mathsf{I} \otimes h_y^{\mathbf{f}} \mathsf{D} \end{bmatrix}, \qquad \mathsf{D} = \begin{bmatrix} -1 & 1 & & \\ & \ddots & \ddots & \\ & & -1 & 1 \end{bmatrix}.$$

Our joint super resolution objective function is written

$$J(\mathbf{w}, \mathbf{f}^{(0)}) = \|(\mathsf{I} \otimes \mathsf{K})\mathsf{I}(\mathbf{w})\mathbf{f}^{(0)} - \mathbf{d}\|^2 + \lambda_{\mathbf{f}}^2 \|\mathsf{L}\mathbf{f}^{(0)}\|^2. \tag{4.1}$$

It should be noted that this quadratic regularizer is not ideal, as it penalizes high wavenumber content in the reconstruction that may be part of the true reference image. This shows up in our examples as "checkerboard" patterns in the residual. To move beyond this, nonlinear regularization methods such as total variation can be used with a "Linearize-And-Project" approach [11], however, like in [3], we stick to a quadratic regularization term as it simplifies both the theory and practice of variable projection.

4.2. Variable Projection Theory. We begin by noticing that (4.1) is in the separable form addressed by the variable projection method [9, 8, 6]. To make this even more clear, we write it in the stacked form

$$J(\mathbf{w}, \mathbf{f}^{(0)}) = \left\| \begin{bmatrix} \sqrt{h_x^{\mathbf{d}} h_y^{\mathbf{d}}}\ (\mathsf{I} \otimes \mathsf{K})\mathsf{I}(\mathbf{w}) \\ \lambda_{\mathbf{f}} \sqrt{h_x^{\mathbf{f}} h_y^{\mathbf{f}}}\ \mathsf{L} \end{bmatrix} \mathbf{f} - \begin{bmatrix} \sqrt{h_x^{\mathbf{d}} h_y^{\mathbf{d}}}\ \mathbf{d} \\ \mathsf{O} \end{bmatrix} \right\|^2 = \|\mathsf{A}_{\lambda_{\mathbf{f}}}(\mathbf{w})\mathbf{f}^{(0)} - \mathbf{b}\|^2.$$

In this case, the optimal $\mathbf{f}^{(0)}$ as a function of $\mathbf{w}$ is given by $\mathbf{f}^{(0)}(\mathbf{w}) = \mathsf{A}_{\lambda_{\mathbf{f}}}(\mathbf{w})^{\dagger}\mathbf{b}$. Thus, our variable projected objective function is given by

$$\tilde{J}(\mathbf{w}) = \|\mathsf{A}_{\lambda_{\mathbf{f}}}(\mathbf{w})\mathbf{f}^{(0)}(\mathbf{w}) - \mathbf{b}\|^2 \tag{4.2}$$

A natural approach to minimize (4.2) is Gauss-Newton, in which case we need to compute the Jacobian of the term $\mathsf{A}_{\lambda_{\mathbf{f}}}(\mathbf{w})\mathbf{f}^{(0)}(\mathbf{w})$. Applying the product rule,

$$\frac{\partial}{\partial \mathbf{w}} \left(\mathsf{A}_{\lambda_{\mathbf{f}}}(\mathbf{w})\mathbf{f}^{(0)}(\mathbf{w}) \right) = \frac{\partial \mathsf{A}_{\lambda_{\mathbf{f}}}(\mathbf{w})}{\partial \mathbf{w}} \mathbf{f}^{(0)}(\mathbf{w}) + \mathsf{A}_{\lambda_{\mathbf{f}}}(\mathbf{w}) \frac{\partial \mathbf{f}^{(0)}(\mathbf{w})}{\partial \mathbf{w}}. \tag{4.3}$$

Computing the Jacobian $\frac{\partial \mathbf{f}^{(0)}(\mathbf{w})}{\partial \mathbf{w}}$ of the projection $\mathbf{f}^{(0)}(\mathbf{w})$ requires differentiation through the least squares solution. An exact formula for such a derivative based on implicit differentiation through an exact solution exists [9], but it is expensive to compute. Previous work in variable-projected Gauss-Newton has employed approximations to avoid this cost—either through low-rank SVD approximations [16] or by entirely neglecting the Jacobian term (effectively setting $\frac{\partial \mathbf{f}^{(0)}(\mathbf{w})}{\partial \mathbf{w}} = 0$) as done in [3].

In an automatic differentiation framework, we can ignore the projection term by turning gradient tracking off during the least squares solve. However, by choosing to track gradients through the least squares solution, we can include this term at the cost of increased memory overhead. It has been shown that automatic differentiation through the Conjugate Gradient (CG) algorithm gives correct gradients in the case of an inexact solve [10], and in the case of an exact solve [2], which we will use in our solutions. A recent preprint [13] explores automatic differentiation through other least-squares algorithms.

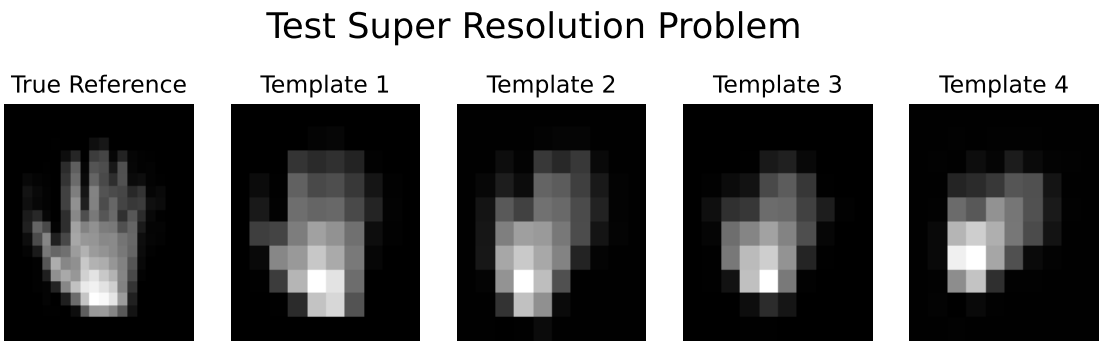

Figure 4. *Our test problem for super resolution. The reference image is* 20×20 *pixels, and the* 4 *templates are* 10×10 *pixels. The problem is well-determined, as we are to infer* 400 *parameters from* 400 *pixels.*

4.3. Implementation and Results. We consider the small super-resolution problem shown in Figure 4 and note a significant benefit in reconstruction quality and convergence rate of the Gauss-Newton method when using AD to differentiate through all iterations of CG (see Figure 5 and Figure 6). There are 18 registration parameters to estimate in this problem, and we found that computing the Jacobian of the variable-projected objective function (Equation (4.2)) using forward-mode automatic differentiation takes approximately six to ten times longer than evaluating the objective function itself, which is in line with our expectations. The final relative reconstruction error when differentiating all iterations of CG is 10.7%, while ignoring the Jacobian of the projection results in a higher relative reconstruction error of 25.61%.

At each optimization step, the Jacobian (4.3) is computed using forward mode AD. Its important to note that in forward mode, memory consumption scales with the size of the input rather than the number of operations in a forward pass. For the variable projected super-resolution problem, the memory consumption due to forward mode should remain constant regardless the number of CG iterations. In our experiments we observed that memory usage increases with the number of optimization steps which can be attributed to an implementation issue in PyTorch.

The computational overhead of using CG to compute the projection still poses a difficulty and its of great interest to reduce the number of iterations while still obtaining quality reconstructions. We explore two options to reduce computational overhead: differentiating through only a portion of CG iterations, but solving the system near exactly, and differentiating through every iteration of CG, but not iterating CG to convergence when computing the projection.

4.3.1. Differentiation of a Portion of CG Iterations. Differentiating through the later iterations of CG proves essential for accurate computation. In these experiments, a total of 200 iterations of CG are used in each optimization step, but the percentage of the final iterations that are differentiated through vary. Figure 7 presents convergence plots for Jacobians obtained via AD, comparing various percentages of CG iterations through which differentiation is applied. Interestingly, differentiating through the final 70% of iterations leads to poorer

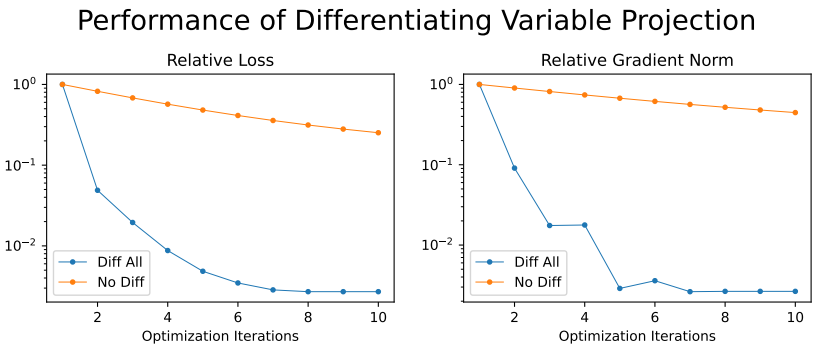

Figure 5. *The performance of 10 iterations of variable-projected Gauss-Newton on the problem in Figure 4. Differentiating through all CG iterations provides a faster reduction in relative loss and relative gradient norm.*

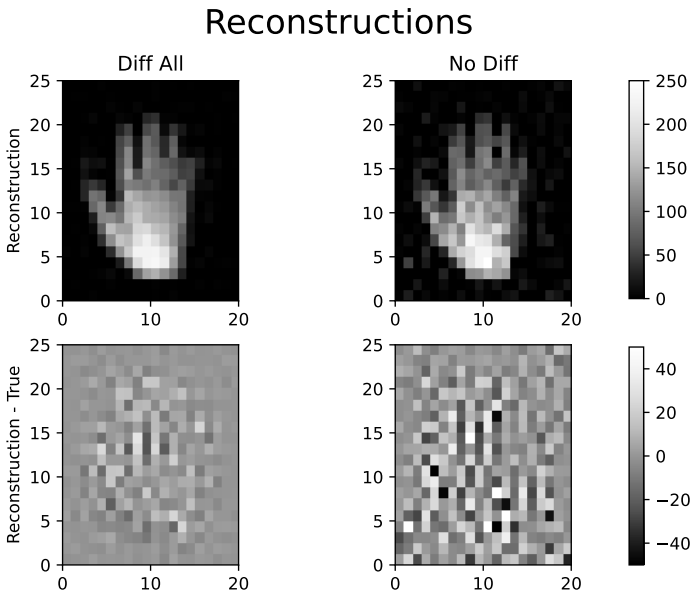

Figure 6. *The final reconstructions from the performance test comparing no differentiation against full differentiation through every iteration of CG.*

convergence than not differentiating through any iterations at all. However, when differentiating through the entire set of iterations (100%), the convergence rate surpasses that of the 90% and 95% cases.

Our results shown in Figure 8 demonstrate that differentiating through more than 70% is necessary. Differentiating through 100% of the iterations of CG is the best in terms of relative loss and gradient norm reduction. The final reconstruction qualities additionally indicate that 100% is the best, but the visual difference in error is small after 90% as seen in Figure 9 . We also tested differentiating through the initial iterations of CG, but the performance was worse than using the later iterations. Therefore, these results are not included in this work.

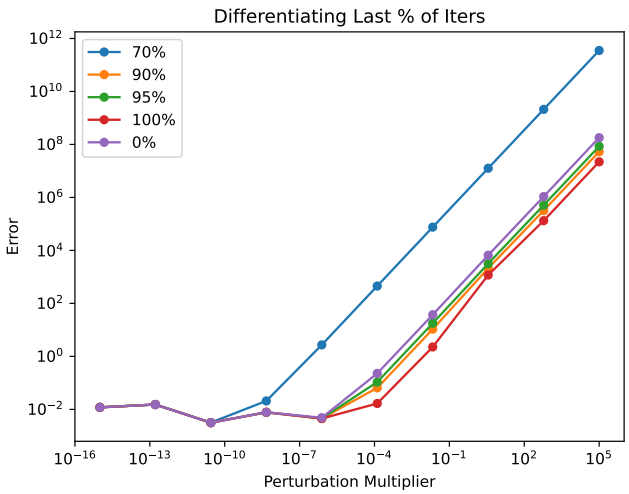

Figure 7. *Jacobian convergence plot for the residual function* $\mathbf{r}(\mathbf{w}) = \mathsf{A}_{\lambda_\mathbf{f}}(\mathbf{w})\mathbf{f}^{(0)}(\mathbf{w}) - \mathbf{b}$. *In this case,* $Error = \|\mathbf{r}(\mathbf{w} + h\mathbf{v}) - \mathbf{r}(\mathbf{w}) - h\frac{\partial bfr(\mathbf{w})}{\partial \mathbf{w}}\mathbf{v}\|$, *where* h *is the perturbation multiplier,* v *is the direction of the perturbation, and our Jacobian approximation* $\frac{\partial bfr(\mathbf{w})}{\partial \mathbf{w}}$ *varies in accuracy depending on the percentage of final iterations of CG that are differentiated.*

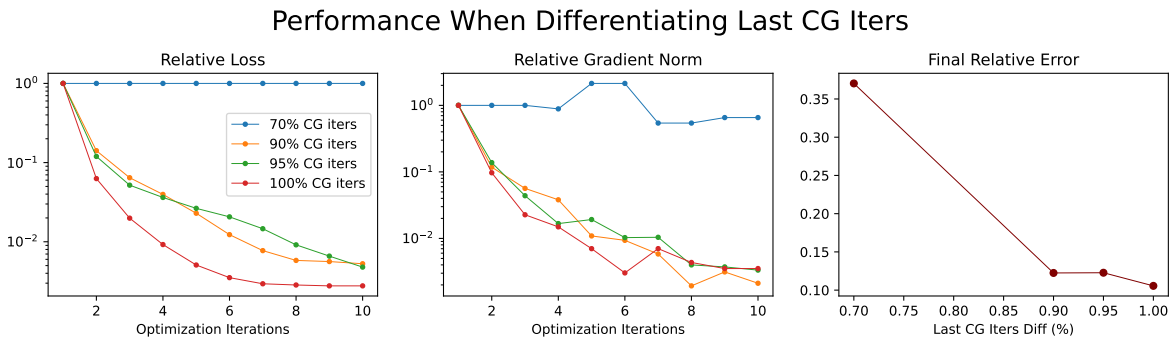

Figure 8. *The performance of 200 iterations of variable-projected Gauss-Newton on the problem from Figure 4 for various percentages of final CG iterations differentiated. The final relative error is computed by solving the least squares problem exactly for the final reference image prediction and measuring the difference from the true reference used to create the data.*

4.3.2. Using Inexact Projections. We test how reconstruction quality varies when using an inexact least squares solution for our projection. We consider the problem shown in Figure 4, which we solve using a variable projected Gauss-Newton method. We automatically differentiate through every iteration of CG, but run a fixed number of CG iterations in each evaluation of the objective function.

Our results (Figure 10) show that there are diminishing returns for using more exact projections in each iteration. Using 50 or 100 iterations performed the best in terms of

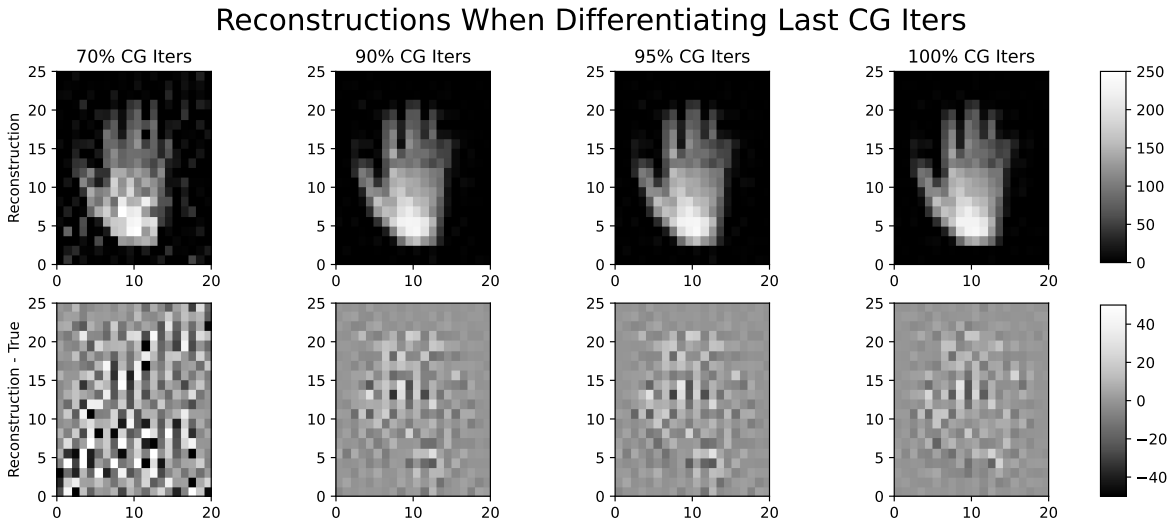

Figure 9. *The final reference images and reconstruction errors of variable projected super resolution with inexact projections. The images are very similar, but some differences in the residuals can be seen.*

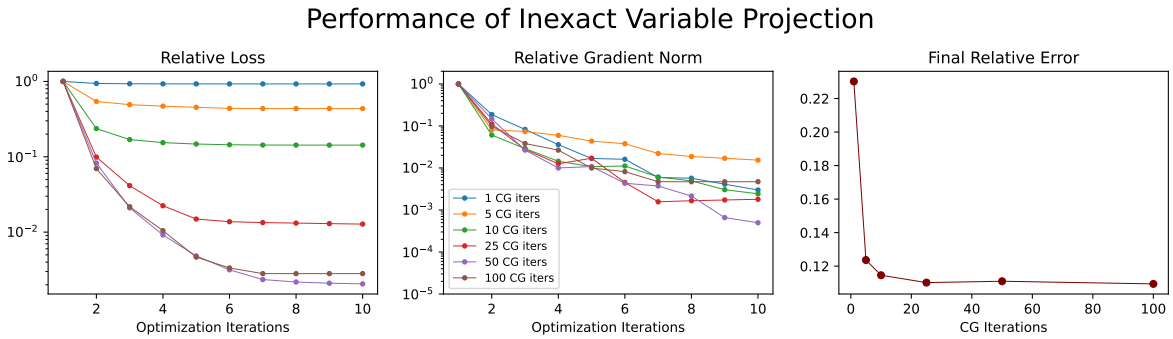

Figure 10. *The performance of* 10 *iterations of variable-projected Gauss-Newton on the problem from Figure 4 for various numbers of Conjugate Gradient Iterations used in each projection. The final relative error is computed by solving the least squares problem exactly for the final reference image prediction and measuring the difference from the true reference used to create the data.*

relative loss and gradient norm reduction. The final recovered reference images (Figure 11) show that the best reconstruction qualitatively is that using 100 CG iterations, but 10, 25, and 50 are close to the same reconstruction quality and the visual difference in reconstructions and error images is small.

5. Conclusions. In the affine registration problem, automatic differentiation has opened up the ability to directly compute Hessians, allowing for the application of Newton's method, which provides faster convergence than the previously used approaches. Furthermore, having access to exact Hessians allows for the implementation of a predictor-corrector method, which

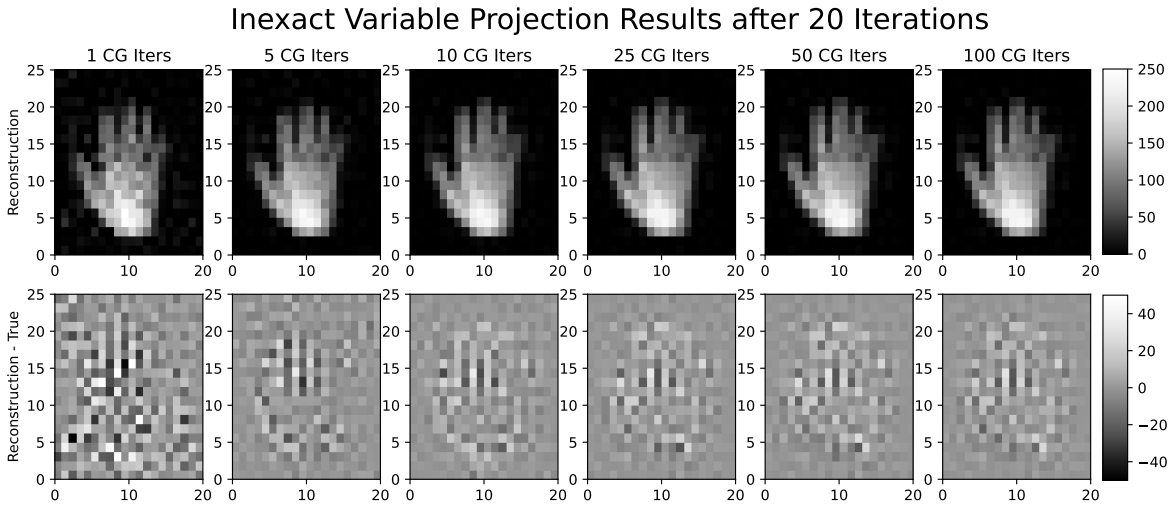

Figure 11. *The final reference images and reconstruction errors of variable projected super resolution with inexact projections. The images are very similar, but some differences in the residuals can be seen.*

we've found to succeed in some problems where the standard multi-scale approach fails.

In the problem of super resolution, we showed that computing the full Jacobian of a variable projected objective function improves both the speed of convergence of the optimization and the final reconstruction quality from the previous approach of not differentiating through the projection, though it introduces computational overhead. Through experiments on a small super resolution problem, we found that tracking the computations of at least 70% of the final iterations of the CG method was necessary to see the benefits of including the Jacobian of the projection. Moreover, we found that using inexact projections during each iteration of optimization maintains high reconstruction quality with fewer CG iterations, providing a balance between computational efficiency and accuracy. These results suggest that leveraging inexact projections can mitigate computational costs while maintaining the advantage in results that the AD computed Jacobian of the projection provide.

Using AD tools, we have been able to simply compute derivatives, opening up methods in image registration that would be difficult to compute manually. The ability to automatically differentiate through complex processes, such as inexact iterative least squares solves, and registration objective functions, has proven its use in developing better super resolution solutions and second order optimization methods. These advancements demonstrate the potential of AD to further progress in image registration. Future work could explore scaling these methods to larger datasets and non-parametric transformations, as well as better optimizing memory usage when computing the derivative of inexact iterative least squares solves.

Acknowledgments. This work was supported by the US National Science Foundation (NSF) award DMS-2349534. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF. We want to especially thank our mentor Dr. Lars Ruthotto and the other mentors

of Emory's 2024 REU for Computational Mathematics and Data Science for making this possible.

REFERENCES

[1] R. T. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, *Neural ordinary differential equations*, Advances in neural information processing systems, 31 (2018).
[2] B. Christianson, *Differentiating through conjugate gradient*, Optimization Methods and Software, 33 (2018), pp. 988–994, https://doi.org/10.1080/10556788.2018.1425862, https://doi.org/10.1080/10556788.2018.1425862, https://arxiv.org/abs/https://doi.org/10.1080/10556788.2018.1425862.
[3] J. Chung, E. Haber, and J. Nagy, *Numerical methods for coupled super-resolution*, Inverse Problems, 22 (2006), p. 1261, https://doi.org/10.1088/0266-5611/22/4/009, https://dx.doi.org/10.1088/0266-5611/22/4/009.
[4] A. R. Conn, N. I. Gould, and P. L. Toint, *Trust region methods*, SIAM, 2000.
[5] D. M. Dunlavy and D. P. O'Leary, *Homotopy optimization methods for global optimization.*, tech. report, Sandia National Laboratories (SNL), Albuquerque, NM, and Livermore, CA (United States), 12 2005, https://doi.org/10.2172/876373, https://www.osti.gov/biblio/876373.
[6] M. I. Español and G. Jeronimo, *Convergence analysis of a variable projection method for regularized separable nonlinear inverse problems*, 2024, https://arxiv.org/abs/2402.08568, https://arxiv.org/abs/2402.08568.
[7] B. Fischer and J. Modersitzki, *Ill-posed medicine—an introduction to image registration*, Inverse Problems, 24 (2008), p. 034008, https://doi.org/10.1088/0266-5611/24/3/034008, https://dx.doi.org/10.1088/0266-5611/24/3/034008.
[8] G. Golub and V. Pereyra, *Separable nonlinear least squares: the variable projection method and its applications*, Inverse problems, 19 (2003), p. R1.
[9] 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 (1973), pp. 413–432.
[10] S. Gratton, D. Titley-Peloquin, P. Toint, and J. T. Ilunga, *Differentiating the method of conjugate gradients*, SIAM Journal on Matrix Analysis and Applications, 35 (2014), pp. 110–126, https://doi.org/10.1137/120889848, https://doi.org/10.1137/120889848, https://arxiv.org/abs/https://doi.org/10.1137/120889848.
[11] J. L. Herring, J. G. Nagy, and L. Ruthotto, *Lap: A linearize and project method for solving inverse problems with coupled variables*, Sampling Theory in Signal and Image Processing, 17 (2018), p. 127–151, https://doi.org/10.1007/bf03549661, http://dx.doi.org/10.1007/BF03549661.
[12] R. Z. Horace He, *functorch: Jax-like composable function transforms for pytorch.* https://github.com/pytorch/functorch, 2021.
[13] P. Hovland and J. Hückelheim, *Differentiating through linear solvers*, 2024, https://arxiv.org/abs/2404.17039, https://arxiv.org/abs/2404.17039.
[14] C. C. Margossian, *A review of automatic differentiation and its efficient implementation*, Wiley interdisciplinary reviews: data mining and knowledge discovery, 9 (2019), p. e1305.
[15] J. Modersitzki, *FAIR: flexible algorithms for image registration*, vol. 6 of Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2009, https://doi.org/10.1137/1.9780898718843.
[16] E. Newman, L. Ruthotto, J. Hart, and B. van Bloemen Waanders, *Train like a (var)pro: Efficient training of neural networks with variable projection*, SIAM Journal on Mathematics of Data Science, 3 (2021), pp. 1041–1066, https://doi.org/10.1137/20M1359511, https://doi.org/10.1137/20M1359511, https://arxiv.org/abs/https://doi.org/10.1137/20M1359511.
[17] J. Nocedal and S. J. Wright, *Numerical Optimization*, Springer, New York, NY, USA, 2e ed., 2006.
[18] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, *Pytorch: An imperative style, high-performance deep learning library*, in Advances in Neural Information Processing Systems 32, Curran Associates, Inc., 2019, pp. 8024–8035, http://papers.neurips.cc/paper/

9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[19] S. S. Schoenholz and E. D. Cubuk, {*JAX*} {*md*}*: End-to-end differentiable, hardware accelerated, molecular dynamics in pure python*, 2020, https://openreview.net/forum?id=r1xMnCNYvB.

[20] S. Sun, K. Han, C. You, H. Tang, D. Kong, J. Naushad, X. Yan, H. Ma, P. Khosravi, J. S. Duncan, and X. Xie, *Medical image registration via neural fields*, Medical Image Analysis, 97 (2024), p. 103249, https://doi.org/https://doi.org/10.1016/j.media.2024.103249, https://www.sciencedirect.com/science/article/pii/S1361841524001749.

[21] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, *SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python*, Nature Methods, 17 (2020), pp. 261–272, https://doi.org/10.1038/s41592-019-0686-2.

[22] W. Watson, C. Cherry, R. Lang, and L. Ruthotto, *Imgregpytorchproject*. https://github.com/wdwatson2/ImgRegPytorchProject, 2024. GitHub repository.

[23] Y. Wu, T. Z. Jiahao, J. Wang, P. A. Yushkevich, M. A. Hsieh, and J. C. Gee, *Nodeo: A neural ordinary differential equation based optimization framework for deformable image registration*, in Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2022, pp. 20804–20813.