Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

Jørgen Nilsen Riseth

Gradient-Based Algorithms in Shape Analysis for Reparametrization of Parametric Curves and Surfaces

Master's thesis in Applied Physics and Mathematics
Supervisor: Elena Celledoni
February 2021

NTNU
Norwegian University of Science and Technology

Jørgen Nilsen Riseth

Gradient-Based Algorithms in Shape Analysis for Reparametrization of Parametric Curves and Surfaces

Master's thesis in Applied Physics and Mathematics
Supervisor: Elena Celledoni
February 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

NTNU
Norwegian University of
Science and Technology

Abstract

In this thesis, we study two gradient-based optimization algorithms in shape analysis for reparametrization of open parametric curves and surfaces. One is a previously known Riemannian gradient descent algorithm on the group of orientation preserving diffeomorphisms. The other is a novel approach, where finding an optimal reparametrization corresponds to the training of a residual neural network. We compare the two algorithms using a few test examples for both curves and surfaces, for which the residual neural network significantly outperforms the gradient descent algorithm.

Sammendrag

I denne oppgaven studerer vi to gradientbaserte optimeringsalgoritmer i formanalyse, for reparametrisering av parametriske kurver og overflater. Den ene algoritmen er en tidligere kjent "gradient descent"-algoritme på gruppen av orienteringsbevarende diffeomorfier. Den andre er en ny tilnærming til reparametriseringsproblemet, der det å finne en optimal reparametrisering, tilsvarer treningen av et restnevralt nettverk. Vi sammenligner ytelsen til de to algoritmene ved hjelp av noen få eksempler for både kurver og overflater. I begge tilfeller presterer det restnevrale nettverket bedre enn "gradient descent"-algoritmen.

Preface

This thesis concludes my studies of applied physics and mathematics at NTNU, specializing in industrial mathematics.

I would like to thank my supervisor, Prof. Elena Celledoni, for her support and guidance through the research process. Your genuine interest in my work, as well as countless hours spent discussing and exploring new ideas, has been invaluable for my work.

Jørgen Nilsen Riseth

Contents

List of Figures

Chapter 1

Introduction

The geometric shape is of major importance when characterizing objects. As toddlers, many children start learning about shapes through various toys and puzzles, and when asked to describe an object, most people will resort to comparing it to familiar shapes. Based on this observation, it seems only natural that objects' shape should be of significant importance when developing algorithms for automatic object recognition, such as in computer vision tasks.

According to the Oxford Learners Dictionary, shape (when used as a noun) is defined as *the form of the outer edges or surfaces of something; an example of something that has a particular form.* While this definition encapsulates the general concepts of the word, we require a more precise mathematical definition to enable automatic object analysis. The field of shape analysis concerns itself with this subject: It provides a formal definition of the shape of an object and tools to compare and measure differences between shapes of objects.

In [29], Kendall describes shapes informally as 'what is left when the differences which can be attributed to translations, rotations, and dilatations have been quotiented out'. This notion of shapes may be formalized by defining shapes as elements of a quotient manifold called *shape space*. The algorithms described in this thesis is developed within this framework.

In [51], the authors identify four main goals for the field of shape analysis.

1. Quantification of shape differences.
2. Building templates for specific shape classes.
3. Modelling shape variations through statistical models.
4. Shape clustering, classification and estimation.

In this thesis, we will mainly concern ourselves with problems related to the quantification of shape differences. Whereas Kendall represented objects by a set of points along its boundary, we will be representing objects as parametric curves or surfaces. This is arguably a more natural representation of real objects. However, this introduces some difficulties. These are elements of infinite-dimensional function spaces, which complicates the construction of shape space. Moreover, the same shape may be outlined by different parametrizations (e.g. two curves tracing out the same path in the plane, but with different velocities). This is often dealt with by defining distances in shape spaces in terms of optimization problems over the space of *reparametrizations*. For parametric curves, this problem is usually solved using a dynamic programming algorithm. However, the introduction of additional constraints to the optimization problem might prevent the application of the algorithm, and to our knowledge, no such algorithm is currently available for surfaces.

This thesis describes two gradient-based approaches for finding optimal reparametrizations, applicable for both curves and surfaces. The first approach is based on the Riemannian gradient descent algorithm over the reparametrization group outlined in [33]. In the second approach, we use ideas from deep learning to rephrase the optimization problem as the process of training a neural network. This leads to a finite-dimensional optimization problem that we solve with the BFGS algorithm. This is to our knowledge a novel approach to solving the problem of finding optimal reparametrizations. We emphasize that even though we are using tools from deep learning, we are not attempting to make predictions regarding unseen instances, which is usually the main goal in

machine learning. The training corresponds to finding an optimal reparametrization of a single curve, and the network will have to be retrained to work for other curves.

We evaluate the performance of the algorithms for both curves and surfaces using a few test examples. Based on the experiments presented, the *deep reparametrization* significantly outperforms the Riemannian gradient descent algorithm.

The rest of the thesis is structured as follows: In chapter 2, we review previous approaches described in the literature on shape analysis for comparing objects. Chapter 3 provides an introduction to the differential geometric theory underlying the algorithms. Chapter 4 considers the problem of finding optimal reparametrization of curves. We define the functions spaces necessary to get a formal notion of the shape of a curve and describe the two gradient-based algorithms. We provide numerical results from the two algorithms and compare their performance. In chapter 5, we extend the problem formulation and algorithms to reparametrization of parametric surfaces and compare the performance of the two algorithms. Chapter 6 concludes the thesis, with a summary of the results presented in the previous chapters and ideas for further work.

Note to the reader

Some of the sections throughout this thesis describe ways to solve very similar problems. Therefore, the contents of these sections look a lot like each other. This is especially true for the extension of the deep reparametrization algorithm from curves to surfaces. While it would be possible to use a higher level of abstraction to put these sections together in a single place, we have decided to keep them apart, such that it is easier to look up parts of particular interest.

Chapter 2

Literature Review

The earliest work regarding the field of shape analysis is generally attributed to D'Arcy Thompson. In the book "On Growth and Form" [58], originally released in 1917, he studied plants and animals and visualized how the shapes of, for example, different types of fish may be made to look similar by applying non-linear transformations to the coordinate system where the objects were represented. Since then, many different efforts have been made towards solving the problems posed in shape analysis, based on a broad range of theoretical approaches; the ability to identify and compare shapes of objects is relevant in numerous branches of science and engineering.

In this section, we review some of the previous work in shape analysis. We do not intend to give a comprehensive description of all available methods, and will focus mainly on the literature taking a differential geometric perspective as outlined in [51]. For a broader picture of tools in shape analysis, we refer to the works of Loncaric [37], and Zhang and Lu [62].

2.1 Shape Analysis of Two-Dimensional Objects

Point Cloud Analysis

One way to represent objects, is by a collection of points $X = \{\mathbf{x}_i\}_{i=1}^k \subset \mathbb{R}^2$ giving the planar coordinates of k points sampled along the objects boundary. In case the point collections are unordered, they are often referred to as point clouds. A popular approach to compare objects in this setting uses variations of the iterated closest point (ICP) algorithm [10, 18, 63]. This algorithm easily extends to the case with three-dimensional surfaces. In [50], the authors describe it as an algorithm for solving the optimization problem

$$\min_{O\in SO(2),\rho\in\mathbb{R}_+,\mathbf{a}\in\mathbb{R}^2,\zeta} \sum_{i=1}^{k} |(\mathbf{a} + \rho O\mathbf{x}_i) - \mathbf{y}_{\zeta(i)}|^2,$$

for two points clouds X, Y. Here O is a rotation matrix, ρ is a scaling constant, $\mathbf{a}$ is a translation vector, while $\zeta : \{1, 2, ..., k\} \to \{1, 2, ..., k\}$ is a mapping called the registration of $\mathbf{y}$ to $\mathbf{x}$, assigning to each $\mathbf{x}_i$ a corresponding point $\mathbf{y}_i$. The optimization problem may be split into two different optimization problems with well-defined solutions: One is the search for optimal *transformation variables*, i.e. the rotation, translation and scaling given a fixed registration, and in the other we want to find an optimal registration for a given set of transformation variables. The ICP algorithm solves these problems iteratively, alternating between the two problems until convergence.

Kendall's Landmark-Based Shape Analysis

Kendall was one of the first to formally define shapes as elements on a manifold [29, 30]. In his works, he represented objects by an ordered set of points, called *landmarks*, from the boundary of an object.

Let $X \in \mathbb{R}^{k\times 2}$ be a matrix where each row $\mathbf{x}_i$ corresponds to one of these landmarks. In the two dimensional case it is often useful to represent each vector as a complex number. Identify $\mathbf{x_i}$ with $z_i = x_{i1} + \mathrm{i}x_{i2} \in \mathbb{C}$, and X with $\mathbf{z} = (z_i)_{i=1}^k \in \mathbb{C}^k$. To remove differences between objects attributed to translations, these objects are standardized by mapping their centroid $\mathbf{z}_0$ to the

origin, and scaled to have unit norm by

$$\mathbf{z} \mapsto \frac{1}{\|\mathbf{z} - \mathbf{z}_0\|} (\mathbf{z} - \mathbf{z}_0), \quad \mathbf{z}_0 = \frac{1}{k} \sum_{i=1}^{k} z_i$$

where $\|\cdot\|$ denotes the standard euclidean norm for complex numbers. The image of $\mathbb{C}^k$ under this standardization is what Kendall calls the *pre-shape space*:

$$\mathcal{C} = \left\{ \mathbf{z} \in \mathbb{C}^k \;\middle|\; \frac{1}{k} \sum_{i=1}^{k} z_i = 0,\ \|\mathbf{z}\| = 1 \right\}.$$

The motivation behind the name *pre*-shape is due to the fact that different points in Cee may represent the same shape, as different rotations of objects have not yet been considered. To identify elements which differ only by a rotation, define the equivalence relation

$$\mathbf{z}_1 \sim \mathbf{z}_2 \iff \mathbf{z}_1 = \mathbf{z}_2 \mathrm{e}^{\mathrm{i}\theta} \text{ for some } \theta \in \mathbb{S}^1.$$

The equivalence classes $[\mathbf{z}] = \{\mathbf{z}^* \in \mathcal{C} \mid \mathbf{z} \sim \mathbf{z}^*\}$ corresponds to all possible rotations of an element in $\mathcal{C}$, and may formally be defined as the *shape* of an object. The set of all shapes $\mathcal{S}$ is then defined as the quotient space

$$\mathcal{S} = \mathcal{C}/\mathbb{S}^1 = \{[\mathbf{z}] \mid \mathbf{z} \in \mathcal{C}\}$$

Now to quantify differences between shapes, we define the distance function in shape space as

$$d_{\mathcal{S}}([\mathbf{z}_1], [\mathbf{z}_2]) = \min_{\theta \in \mathbb{S}^1} d_{\mathcal{C}}(\mathbf{z}_1, \mathbf{z}_2 \mathrm{e}^{\mathrm{i}\theta})$$

where $d_{\mathcal{C}}$ is a distance function on $\mathcal{C}$, defined as the length of *geodesics* in $\mathcal{C}$.

The importance of the constructs given here will be further highlighted through later sections: While most of the recent literature on differential-geometric approaches to shape analysis considers objects represented by curves instead of point samples, they still use the idea of representing shapes as elements on quotient spaces.

Deformation-Based Shape Analysis

Another approach for comparing two-dimensional objects represented as images is based on deformable templates. Pioneered by Grenanders work in e.g. [25], this approach poses the registration problem as a variational problem, where the goal is to find an optimal "warping" of the image domain to match two different images. Paraphrasing the description given in [42], we considers images $I_i : D \to \mathbb{R},\ i = 1, 2$ on some domain D, e.g. $D = [0, 1]^2$. The difference between the images is given by

$$C(I_1, I_2) = \min_{\psi : D \to D} d(I_1, I_2 \circ \psi) + E(\psi)$$

for some distance function d, often chosen as the L^2-distance, and a regularization term $E(\psi)$ penalizing large deformations of the image domain. In many cases, it is useful to impose further restrictions on the set of warping functions ψ, such as requiring them to be diffeomorphisms on the domain D. One big difference between the deformable template-matching as compared to the previously described point-based approaches and the curve-based approaches that will be presented shortly is that it takes into account the contents of a complete image, rather than just the contours of an object. This may both be considered a strength or a weakness, depending on the application. However, the deformable template-matching of images is closely related to the Riemannian framework for matching surfaces that we study in this thesis.

2.1.1 Parametric Approaches

The use of curves or surfaces instead of point collections to represent objects complicates the construction of shape metrics. In [50], however, the authors make a compelling argument for studying objects as continuous parametric curves and surfaces anyways: For one, real-life objects are indeed continuous objects themselves, hence the information loss occurring by sampling objects and representing them as finite-dimensional vectors may be avoided. Moreover, any approach where objects are represented by point vectors, impose a somewhat arbitrary correspondence between

points on the two objects, through the ordering of the vector elements. If the points are not ordered, then we are still faced with matching points between objects, which is also the main difficulty when utilizing parametric curves.

There are two common approaches to represent objects as curves: Either as open curves defined on some interval I (typically $[0,1]$ or $[0,2\pi]$) or as closed curves define on the circle $\mathbb{S}$. The curves considered are typically assumed to be immersions, i.e. smooth with non-zero velocity. In most of the literature, smooth is taken to mean infinitely differentiable. However, some authors also allow functions with lower regularity, (e.g. C^2 [60]).

For simplicity, we will concern ourselves with open curves on the unit interval $I = [0,1]$, and we will take smooth to mean infinitely differentiable. We will refer to the space of parametric curves as $\mathcal{C} \subset C^\infty(I, \mathbb{R}^2)$, but note that the exact definition of the pre-shape space $\mathcal{C}$ varies in the literature.

Riemannian Metrics

When comparing shapes of parametric curves, it is useful to make use of the underlying manifold structure of the space of curves $\mathcal{C}$. By defining equivalence relations between curves which differ only by their velocity, then we may construct the shape space as the quotient manifold under this relation. To define distance functions in such a shape space typically requires two steps: Firstly, we need to equip the underlying space of parametrized curves with a Riemannian metric G. A Riemannian metric gives us a mean of measuring local deformations of curves, which in turn enables the computation of geodesics, i.e. the *shortest* path between curves. Secondly, we need to find the correct representatives for the equivalence classes of the curves we compare. This is done through solving an optimization problem over the group of reparametrizations.

There are multiple ways to define the Riemannian metric on the manifold of curves. When comparing curves, its often tempting to consider the standard L^2-metric. However, the L^2-metric induces a distance which is not invariant to reparametrizations (see section 3). One attempt to alter the L^2-metric to become reparametrization invariant, is to integrate with respect to the arc length of a curve, i.e.

$$G_c(h,k) = \int_I \langle h,k\rangle |c'(t)|dt.$$

This approach was studied by Michor and Mumford in e.g. [41, 40]. However, they showed that given any two curves in $\mathcal{C}$, it is possible to find some geodesic with length zero. This pathology is related to the lack of information regarding derivatives of the curves and motivates the use of higher-order Sobolev-type metrics of the form

$$G^n_c(h,k) = \int_I \sum_{i=0}^{n} a_i \langle D^i_s h, D^i_s k\rangle |c'(t)|dt$$

where $D_s h = h'/|c'(t)|$, and $a_i \geq 0$ are weighting coefficients. Such metrics were studied in e.g. [39], with further investigation of a special case of these, called immersion-Sobolev metrics of the form

$$G^{Imm,n}_c(h,k) = \int_I \langle h,k\rangle + a_n \langle D^n_s, D^n_s\rangle |c'(t)|dt.$$

While Sobolev metrics of higher orders n are nice theoretical generalizations of the L^2-metric, their practicality is limited due to the geodesic equations being partial differential equations of order $2n$. To avoid dealing with higher-order PDEs, much focus has been given to first-order Sobolev metrics, and especially the so-called elastic metric

$$G^{a,b}_c(h,k) = \int_I a^2 \langle D_s h, n\rangle \langle D_s k, n\rangle + b^2 \langle D_s h, v\rangle \langle D_s k, v\rangle |c'(t)|dt$$

where n, v are the unit normal and tangent vectors to c. Whenever $h = k$, the first term in the integrand may be considered a measure of the stretching of the curve in the direction of h, while the second term may be considered a measure of the bending of the curve. The importance of these two features may thus be adjusted by changing the weights a, b. In [61], Younes et al. showed that its possible to find explicit expression for the geodesics in the case $a = b = 1$, by representing the curve by $\sqrt{c'}$ where c is interpreted as a complex-valued function, rather than a planar curve.

An alternative representation of the curves, which has seen much popularity in the literature is the square-root velocity transform (SRVT) of the curve, introduced in [49]. It is given as the map

$$\mathcal{R} : \mathcal{C} \to C^\infty(I, \mathbb{R}^2) \setminus \{0\}, \quad c \mapsto \frac{c'}{\sqrt{|c'|}}.$$

One reason behind its popularity is that under this transformation, the elastic metric with coefficients $a = 1$ and $b = 1/2$ becomes the flat L^2-metric, which greatly simplifies the computation of shape distances and geodesics.

The success of the SRVT has spawned similar approaches to constructing reparametrization invariant metrics by transforming the curves in a way which reduces various Riemannian metrics to the flat L^2-metric. In [5], Bauer et al. study the theoretical properties of some of these metrics: In addition to the SRVT, and the aforementioned transform of Younes et al. [61], they also consider an analogue to the Q-transform which Kurtek et al. developed for surfaces [33]. The pullback of the L^2-metric under this transformation, is a first-order immersion-Sobolev metric on the space of curves. While it lacks some of the nice properties of the SRVT, such as translation invariance and a known inverse, it is more easily generalized to surfaces. Therefore we will put a lot of emphasis on this transform.

Optimizing over the Reparametrization Group

After having endowed the space of parametric curves with some Riemannian metric, we need an approach to find an *optimal* parametrization of the curves we want to compare. To do this, we consider the group of orientation preserving diffeomorphisms, also referred to as the reparametrization group

$$\mathrm{Diff}^+(I) =: \{\gamma \in C^\infty(I, I) \mid \gamma(0) = 0,\, \gamma(1) = 1,\, \gamma'(x) > 0\, \forall x \in I\}\,.$$

The reparametrization group is a Lie group, which acts on $\mathcal{C}$ from the right by composition. The effect of this action on the curve is that its velocity changes. We may identify two objects as having the same shape, if one may be attained through composing the other from the right, by a function from the reparametrization group. If we assume that the Riemannian metric defines a distance function $d_{\mathcal{C}}(c_1, c_2)$ on the space of parametrized curves, then a distance between the chapes c_1 and c_2 may be defined as $\inf_{\gamma \in \mathrm{Diff}^+(I)} d(c_1, c_2 \circ \gamma)$.

There are multiple approaches to finding such an optimal diffeomorphism. The most popular of these is through a dynamic programming algorithm [48], which approximates the diffeomorphism by piecewise linear functions. It entails mapping the unit square $[0, 1]^2$ into a grid $\{(x_i, y_j)\}_{1 \le i,j \le, N}$ of points, where the first coordinate represents the domain of the function, and the second is the range. The graph of the reparametrization is found by drawing lines between points on this grid in an optimal way. Due to the ordering of points in time, it is possible to rewrite the optimization problem as a sum of subproblems which may be solved recursively, to find a globally optimal solution.[1] This algorithm has been developed further in e.g. [20, 9] to significantly reduce the computational cost from $\mathcal{O}(N^4)$ to $\mathcal{O}(N)$, but with reduced precision. In [60], a semi-discretized algorithm was proposed in which only the domain is discretized.

The DP algorithm offers a great alternative to find optimal reparametrizations for computing shape distances when it is applicable. However, in some circumstances further restrictions are imposed on the diffeomorphisms, e.g. when considering closed curves, or if the cost function is augmented with additional terms. These additional constraints may prevent the cost function from being additive over the graph of the diffeomorphism. Moreover, there are to our knowledge currently no similar approach available for the case of surfaces. This motivates the developments of alternative approaches for finding optimal reparametrizations. In [19], the authors considered closed curves parametrized on the unit circle $\mathbb{S}^1$ and found diffeomorphisms generated by flows of vector fields. In [5], the authors expanded upon a gradient-based algorithm proposed in [55], where the optimal diffeomorphism is found by a gradient descent algorithm on the reparametrization group. This is similar to approach described in [49], but while the former samples the diffeomorphism at a grid of point at each iteration, the latter works directly with functions from a subspace of the diffeomorphism group, formed by a truncated Fourier series. We will expand upon this in a later section.

[1] Since the algorithm is restricted to find linear functions passing through a predetermined set of points, it is of course not able to cover the whole reparametrization group.

2.2 Shape Analysis of Three-Dimensional Objects

Many of the methods already described for shape analysis of curves have natural extensions to three-dimensional objects and surfaces. Increasing the dimension of the point cloud analysis approaches poses no challenges (at least theoretically), and the ICP algorithm was developed to work also for this case. The landmark-based tools for Kendall's shape analysis, such as Procrustes methods, are still available, even though we can no longer use the identification with complex numbers. In case of parametrized surfaces, one approach to solving the problem is the SPHARM, and SPHARM-PDM [12, 52] approaches, in which surfaces are approximated by use of spherical harmonics, possibly augmented by shape descriptors on the surface.

Another popular approach is an extension of deformation based approaches. In the method of large deformation diffeomorphism matching metric (LDDMM), studied in e.g. [13, 6], the objects are embedded in e.g. a unit cube $[0,1]^3$. Transformations are then applied to the ambient space, dragging the surfaces along. Similarly as for the two-dimensional case, the shape distance is computed through minimizing a cost functional which measures differences after the transformation, but penalizes large deformations. Metrics based on this approach are often referred to as *outer metrics*.

Contrasting the LDDMM framework are methods based on so-called *inner metrics*. These methods consider parametric surfaces and use Riemannian metrics similar to the ones we described for curves, where deformations are prescribed directly to the surface itself without changing the ambient space. In [33] the Q-transform[2] for surfaces was introduced,

$$Q : \mathrm{Imm}(M, \mathbb{R}^3) \to C^\infty(M, \mathbb{R}^3), \quad f \mapsto \sqrt{|f_x \times f_y|} f(x)$$

where $\mathrm{Imm}(M, \mathbb{R}^3)$ the set of immersions defined on some domain M, and f_x, f_y refer to partial derivatives of the surface. This transform allows us to use the L^2-metric to compute shape distances. The authors also provided a framework to compute the optimal reparametrization of surfaces trough a gradient-based algorithm, allowing the computation of shape distances. This method was extended in [35, 34], to compute the geodesics between surfaces. The Q-transform for surfaces exhibits the same theoretical difficulties as the curves. To find a shape metric independent of translation and scaling requires a pre-processing step similar to the landmark-based approach. Denoting the area scaling factor of a surface f by $a_f = |f_x \times f_y|$, the standardization of surfaces is given by the map

$$f \mapsto \frac{f_c(\mathbf{x})}{\sqrt{\int_M a_{f_c}(\mathbf{x})\, d\mathbf{x}}}, \quad f_c(\mathbf{x}) = f(\mathbf{x}) - \frac{\int_M f(\mathbf{x}) a_f(\mathbf{x})\, d\mathbf{x}}{\int_M a_f(\mathbf{x})\, d\mathbf{x}},$$

which maps the centroid of the surface to the origin, and scales the surfaces to have unit area. The image of this standardization is analogue to Kendall's pre-shape space.

In [28], Jermyn et al. defined a general elastic metric on the space of parametrized surfaces and introduced a transform more closely related to the successful SRVT: The Square Root Normal Field (SRNF), where the surface is represented by its normal vector and divided by the square root of the local area scaling factor. Similarly as for the Q-transform, distances between surfaces under this transform may be computed in terms of the L^2-metric. Using the SRNF, the authors improved upon the Q-transform in clustering tasks, in terms of cluster purity and symmetry of shape distances. They did, however, recognize that the SRNF is not injective; there exists different shapes with the same SRNF-representation, causing problems (at least theoretically) when defining shape distances. In [31], Klassen and Michor investigated the nature of this lack of invertibility and found multiple examples of closed surfaces which have the same image under the SRNF. However, they were also able to show that the SRNF of a strictly convex surface is unique.

The SRNF framework has seen some recent developments. In [53], Su et al. augment the SRNF-distance with another metric. Under this representation, the distance function between shapes is a sum of the L^2-distance of the transformed surfaces and the length of the geodesic under the so-called DeWitt-metric. In [54], the authors define a family of elastic metrics on the space of parametrized three-dimensional surfaces, which includes the elastic metric induced by the SRNF transform.

[2] The original name of the Q-transform was the q-map. However, the authors used this word to refer both to the transform itself, as well as elements in its image. To avoid this ambiguity, we adopt the name used in [5].

Optimal Reparametrizations

Under all of the aforementioned metrics for parametric surfaces, we are still required to find an optimal representative of each shape to compute distances in shape space. One fact that greatly complicates finding optimal reparametrizations of surfaces is the lack of a canonical ordering of points on a surface. The extra dimensions added allows the reparametrization functions to rotate points so that two points switch places in one or both of its coordinates. The reparametrization group for surfaces on some domain $M \subset \mathbb{R}^2$ may be defined as

$$\text{Diff}^+(M) = \left\{\gamma \in C^\infty(M, M) \;\middle|\; J_\gamma > 0,\ \gamma \text{ bijective}, \gamma^{-1} \in C^\infty(M, M)\right\}$$

where J_γ is the Jacobian determinant of γ. Diffeomorphisms in the reparametrization group for surfaces may move points along the boundary, as opposed to the reparametrization group for curves, which necessarily preserves the endpoints.

The dynamical programming algorithm for optimal reparametrization of curves is therefore not easily extended to surface, and to our knowledge, no such global optimization algorithm has been developed in this case. This leaves us with various gradient-based algorithms.

In [33] this was solved through gradient descent on a subset of the group of diffeomorphisms, spanned by a set of basis functions that are tensor products of Fourier series. This will be further elaborated upon in section 4. Other approaches to solving this have been proposed in e.g. [4], where the group of diffeomorphisms are approximated in a linear space, and optimal coefficients found using BFGS. The deep reparametrization algorithm described in sections 4 and 5 may be considered a mixture of the two approaches.

2.3 Shape Analysis in Activity Recognition

Until now, we have mainly considered shape analysis for static object recognition. Another important application for the tools presented is in automatic identification of activities from videos or motion capture data. A typical approach is to extract the shapes as e.g. the silhouettes of objects in each frame of a video and make use of the usual tools for comparing shapes. This, however, requires additional time-series modelling to compare sequences on shape spaces, which may be solved in very different ways. We will settle to describe one of these approaches, which is given as a natural extension of the SRVT framework for comparing objects.

A popular way of capturing realistic motions for use in computer animation is through motion capturing, where the movement of an actor is recorded, and imposed onto a virtual *skeleton*. In [21], Eslitzbichler provided an approach to model character animations from motion capture data as curves on an n-torus. Each point on the curve corresponds to a *pose*, that may be mapped or embedded in $\mathbb{R}^3$. This allows the use of the SRVT of the curves to compare and classify different movements. In [15], Celledoni et al. provided an extension of the SRVT for Lie group valued curves, which in turn made it possible to model these character animations directly as curves in $SO(3)^n$. In [16], the SRVT was further generalized for curves in homogeneous spaces.

Similarly as in object recognition, it is necessary to find an optimal parametrization of the curves to best compare different animations. In [3], the authors describe both a gradient descent algorithm and a dynamic programming algorithm to find optimal reparametrizations when comparing character animations. [3]

2.4 Outro

The contribution of this thesis to the field of shape analysis is twofold. Most importantly, we put forth a novel approach to finding optimal reparametrizations, where we approximate diffeomorphisms using a residual neural network. This approach admits a seemingly effective, unified framework for optimal reparametrization of both curves and surfaces. Secondly, the Riemannian gradient descent algorithm considered within this thesis is based on the framework provided in [33] for reparametrization of surfaces. However, the algorithm in the original paper is stated with a high level of abstraction, and we could not find a detailed explanation of how to implement

[3] These algorithms are really just extensions of the algorithms described for optimal reparametrizations of open curves, but they augment the optimization problem with an additional term matching feature points on the animations to be compared.

the algorithm within available literature. While there are probably better ways to implement the gradient descent algorithm than the one we propose, we hope to provide a sufficient description of the algorithm for interested readers to understand, and that it may serve as a starting point for future improvements.

Chapter 3

Theoretical Framework

This chapter contains the differential geometric theory underlying the algorithms studied in this thesis. Basic theory regarding Riemannian geometry and Lie groups is based on [50], while the introduction to infinite-dimensional manifolds is based on [14]. For the manifold structure of manifolds with corners, and the manifold structure of function spaces we refer to [38], while the description of Riemannian gradient descent is taken from [36].

3.1 Differential Geometry

Smooth Manifolds

Definition 3.1.1 (Smooth Manifold)**.** *A smooth manifold modelled on a topological vector space E, is a Hausdorff topological space M together with a family of charts $(\varphi_i, U_i)_{i\in\mathcal{I}}$ satisfying*

1. $M = \bigcup_{i\in\mathcal{I}} U_i$.
2. U_i *is open in* M *for all* $i \in \mathcal{I}$.
3. $\phi_i : U_i \to \phi(U_i)$ *are homeomorphisms onto open subsets* $\phi(U_i) \subset E$.
4. $\phi_i \circ \phi_j^{-1} : \phi_j(U_i \cap U_j) \to \phi_i(U_i \cap U_j)$ *are smooth (i.e.* C^∞*) for all* $i, j \in \mathcal{I}$.

This definition of manifolds covers both finite and infinite-dimensional manifolds (without boundary) depending on the choice of the vector space E. We are also interested in finite-dimensional manifolds with boundaries and corners.

Definition 3.1.2 (n-dimensional Manifolds)**.** *We say that a smooth manifold with a family of charts $(\varphi_i, U_i)_{i\in\mathcal{I}}$ is an n-dimensional manifold*

1. *without boundary if the open sets $\phi_i(U_i)$ are open in $\mathbb{R}^n$.*
2. *with boundary if the open sets $\phi_i(U_i)$ are open subset of $H^n := [0,\infty) \times \mathbb{R}^{n-1}$.*
3. *with corners if $\phi_i(U_i)$ are open in $\mathbb{R}^n_k = [0,\infty)^k \times \mathbb{R}^{n-k}$ for some $0 \le k \le n$.*

Here $\mathbb{H}^{n-1}, \mathbb{R}^n_k \subset \mathbb{R}^n$ are equipped with the subspace topology.

While the domain of the functions used to represent objects is assumed to be finite-dimensional manifolds, the function spaces themselves may be given an infinite-dimensional manifold structure. Four common modelling spaces for infinite-dimensional manifolds, are

- Hilbert spaces.
- Banach spaces.
- Fréchet spaces.
- Convenient vector spaces.

Hilbert and Banach spaces are relatively nice to work with, as most of multivariate calculus on finite-dimensional spaces generalizes easily to Banach spaces. However, due to Omori [44], we know that a Banach Lie group with the properties we desire of the diffeomorphism group is necessarily finite-dimensional. This means that Banach spaces are not fitting as a modelling spaces for the diffeomorphism group. Therefore, we turn to the other two alternatives.

Definition 3.1.3 (Locally Convex Vector Spaces)**.** *A vector space E is called a locally convex vector space if its topology is induced by a family of semi-norms $\{p_i\}_{i\in\mathcal{I}}$ for some index set $\mathcal{I}$, and the semi-norms are point-separating (i.e. $p_i(x) = 0\ \forall\ i \in \mathcal{I} \iff x = 0$). A locally convex vector space which is sequentially complete, is called a Fréchet space.*

The construction of the manifold structure of relevant functions spaces is a complicated process far beyond the scope of this master thesis. However, it is possible to show that by equipping the space $C^\infty(M, N)$ of smooth functions between smooth manifolds with the so-called compact-open topology, it becomes a Fréchet space. Moreover, it is possible to show that interesting function spaces such as the set of immersions are open subsets of $C^\infty(M, N)$, and are therefore manifolds covered by a single chart. We will be using this setting throughout the thesis. For a thorough exposition of the last alternative; convenient vector spaces, we refer to [32].

Tangent Spaces

While smooth manifolds are inherently non-linear spaces, they are by definition locally homomorphic to vector spaces. By approximating manifolds around a point by linear spaces, the tools of linear algebra, such as inner products, are made available locally. By using the smooth structure of the manifolds, local arguments may be extended to the whole manifold, enabling concepts such as distances on the manifold.

Definition 3.1.4 (Tangent Vectors)**.** *Let M be a manifold modelled on a locally convex vector space E, and let $p \in M$. A curve $\alpha : (a, b) \to M$ is considered smooth if the map $\phi \circ \alpha : (a, b) \to E$ is smooth for some coordinate map ϕ. We say that a smooth curve α passes through p if $\alpha(0) = p$. Given two smooth curves α, β passing through p, we define the equivalence relation*

$$\alpha \sim \beta \iff (\phi \circ \alpha)'(0) = (\phi \circ \beta)'(0)$$

for any[1] *chart ϕ of M around p. The equivalence class $[\alpha]$ is called a tangent vector of M at p. The tangent space of M at p, denoted T_pM, is defined as the set of all tangent vectors at p. The collection of all tangent tangent spaces $TM = \bigcup_{p\in M} T_pM$ is called the tangent bundle.*

The tangent space at a point is locally isomorphic to the modelling space. If ϕ is a coordinate chart of M around p, then there exists a bijective map $T_pM \to E$ defined by

$$[\gamma] \mapsto (\phi \circ \gamma)'(0) = \phi(\gamma(0))$$

which allows us to transfer the operations of addition and scalar multiplication from E to T_pM, turning T_pM into a vector space. Given a curve α passing through 0, we denote the equivalence class of α in T_pM by $\alpha'(0)$.

Since the tangent space T_pM is isomorphic to the modelling space E, then for any manifold M which is an open subset of E, every tangent vector v at a point p may be represented by the line $t \mapsto p + tv$, defined for some interval $(a, b) \ni 0$. This will be the case for most of the function spaces we are working with throughout this thesis.

The tangent vectors of a manifold give us a notion of moving along a manifold in a specific direction, which allow us to define the directional derivative of a map $f : M \to N$ between manifolds.

Definition 3.1.5 (Directional Derivatives)**.** *Given a differentiable map $f : M \to N$ between manifolds M, N and a smooth curve $\alpha : (a, b) \to M$ passing through $p \in M$ with $\alpha'(0) = v$, the directional derivative or differential of f at p is given by*

$$df_p : T_pM \to T_{f(p)}(N), \qquad df_p(v) = (f \circ \alpha)'(0) = \left.\frac{d}{dt}\right|_{t=0} f(\alpha(t)) \tag{3.1}$$

which defines a linear map between tangent spaces at p. If $g : N \to K$ is another map between manifolds, then the directional derivative satisfies the chain rule

$$d(g \circ f)_p = dg_{f(p)} \circ df_x$$

[1] If the equality holds for some chart, then it holds for every chart around p, by the chain rule.

Riemannian Manifolds

Definition 3.1.6 (Riemannian Metrics)**.** *Let M be a smooth manifold modelled on a vector space E. A weak Riemannian metric G is a smooth map G assigning to each $p \in M$ a map G_p satisfying*

1. *$G_p(\cdot,\cdot)$ is symmetric and bilinear for all $p \in M$.*
2. *$G_p(h,h) \geq 0$ for all $h \in T_pM$ with equality only for $h = 0$.*

If a weak Riemannian metric in addition satisfies

3. *The topology of the inner product space (T_pM, G_p) coincides with the topology T_pM inherits from the manifold M.*

then we call it a strong Riemannian metric. A (weak/strong) Riemannian manifold (M, G) is a smooth manifold M endowed with a (weak/strong) Riemannian metric G.

The difference between a weak and strong Riemannian metric is a purely infinite-dimensional phenomenon. It does have some consequences in the field of shape analysis. If we define distances on M in terms of the length of *geodesics* with respect to the metric, then a weak Riemannian metric is not point-separating. This means that there exist distinct points on the manifold, for which there is a curve of arbitrarily short length connecting the two points. Even worse, this might happen for *any* two points on the manifold. We briefly mentioned one example of this degeneracy in section 2.1.1: The arc-length parametrized L^2-metric.

Definition 3.1.7 (Geodesic)**.** *Let $c : [a,b] \to M$ be a piecewise C^1-curve on a Riemannian manifold (M, G). The length of c is*

$$L(c) := \int_a^b |c'(t)|\, dt = \int_a^b \sqrt{G_{c(t)}(c'(t), c'(t))}\, dt \tag{3.2}$$

The geodesic distance between to points $p_1, p_2 \in M$ is given by

$$dist(p_1, p_2) = \inf_c L(c)$$

where the infimum is taken over the set of all piecewise C^1-curves with $c(a) = p_1$, $c(b) = p_2$. We say that a curve c is arc-length parametrized if $|c'| = 1$. A shortest geodesic is an arc-length parametrized curve attaining the infimal length. A geodesic is an arc-length parametrized curve that is locally length minimizing, in the sense that there exist some $\delta > 0$ such that c restricted to any subinterval $I \subset [a,b]$ of length smaller than δ, is a shortest geodesic.

In the algorithms considered in this thesis, we are not computing any of these geodesics. However, the distance functions we will use to compare shapes correspond to the geodesic distance for some Riemannian metric on the spaces of curves and surfaces, and the geodesics play a central role for the theory behind gradient descent on manifolds.

Definition 3.1.8 (Riemannian Gradient)**.** *Let $f : M \to \mathbb{R}$ be a function over a Riemannian manifold (M, G). Denote by $\langle \cdot,\cdot \rangle_p = G_p(\cdot,\cdot)$. The Riemannian gradient $\nabla f : M \to TM$ of f is a vector field assigning to each $p \in M$ the unique tangent vector $\nabla f(p) \in T_pM$ satisfying*

$$df_p(v) = \langle \nabla f(p), v \rangle_p \qquad \forall\, v \in T_pM. \tag{3.3}$$

The gradient of a function defines a direction of largest growth. By taking short steps in the opposite direction of the gradient, we should expect a decrease in the function value. However, due to the non-linearity of manifolds, we need to take care when talking about small steps along a direction. For this purpose we define the Riemannian exponential map.

Definition 3.1.9 (Exponential Map)**.** *Let M be a Riemannian manifold. Given a point $p \in M$ and a vector $v \in T_pM$, there exists for some $\varepsilon > 0$, a unique constant-speed parametrized geodesic $\alpha_v : (-\varepsilon, \varepsilon) \to M$, such that $\alpha_v(0) = p$, $\alpha_v'(0) = v$. The Riemannian exponential map,*

$$\exp_p : U \subset T_pM \to M, \quad \exp_p(v) \mapsto \alpha_v(1),$$

maps elements from an open 0-neighbourhood U in T_pM to the manifold M.

For the exponential map to be well defined, it needs to be restricted to a 0-neighbourhood U for which the integral curve α_v is defined at time 1. However, for any vector v outside of this neighbourhood, there exist some $a > 0$ such that $av \in U$. Hence any tangent vector may be rescaled to a vector such that the exponential map is defined.

Using the Riemannian exponential, one can define Riemannian gradient descent analogously as the Euclidean case:

$$p_{t+1} = \exp_{p_t}(-\eta \nabla f(p_t)) \tag{3.4}$$

On vector spaces equipped with a flat metric (e.g. $\mathbb{R}^n$, $L^2(I)$), geodesics are given as straight lines $t \mapsto p + tv$, which reduces the exponential map to addition $\exp_p(v) = p + v$. On general manifolds, however, the exponential map may be more complicated to compute. Therefore, Riemannian gradient descent algorithms are often defined in terms of retractions.

Definition 3.1.10 (Retraction)**.** *A retraction on a manifold M is a map $r : TM \to M$ that assigns to the tangent space at each point $p \in M$ a map*

$$r_p : T_pM \to M, \quad v \mapsto r_p(v) \tag{3.5}$$

satisfying

$$r_p(0) = p, \quad (dr_p)_0 = Id|_{T_pM}\,.$$

A retraction is a first order approximation of the Riemannian exponential map, and the exponential map is a retraction itself. Using retractions, we may define an alternative update rule for Riemannian gradient descent by

$$p_{t+1} = r_{p_t}\left(-\eta \nabla f(p_t)\right) \tag{3.6}$$

A family of retractions that are especially useful for gradient-based algorithms on manifolds embedded in a vector space E, is given by

$$r_p : T_pM \to M, \quad v \mapsto \pi(p + v) \tag{3.7}$$

where $\pi : E \to M$ is a differentiable projection (i.e. $\pi \circ \pi = \pi$).

Lie Groups

Definition 3.1.11 (Lie Groups)**.** *A Lie Group G is a smooth manifold endowed with a group structure such that the group operations multiplication*

$$\mu_G : G \times G \to G, \quad \mu_G(g, h) = gh$$

and inversion

$$\iota : G \to G,\ \iota(p) \mapsto p^{-1}$$

are smooth maps. We denote by

$$L_g : G \to G, \quad L_g(h) = gh$$

the left-multiplication by g.

Since left multiplication by g is a smooth operation, the neighbourhood around any element $g \in G$ is diffeomorphic to the neighbourhood around the identity element, and the derivative map $d(L_g)_{\mathrm{id}} : T_{\mathrm{id}}G \to T_gG$ defines an isomorphism between tangent spaces. By describing the tangent space at the identity element id, then $d(L_g)_{\mathrm{id}}(T_{\mathrm{id}}G)$ describes the tangent space around any element $g \in G$. Therefore, the tangent space at the identity of a lie group is of special importance.

The set

$$\mathrm{Diff}(M) = \left\{\gamma \in C^\infty(M, M) \;\middle|\; \gamma \text{ bijective}, \gamma^{-1} \in C^\infty(M, M)\right\}$$

of smooth diffeomorphisms on a manifold (possibly with corners) M, forms a Lie group under composition of smooth maps, $\mu_{\mathrm{Diff}(M)}(\gamma, \varphi) = \gamma \circ \varphi$. It is possible to show that the diffeomorphism group forms an open subset of the Fréchet space $C^\infty(M, M)$ [2], hence the manifold structure. The smoothness of composition and inversion is, in turn, inherited from the model space.

[2] If M is a manifold with corners, then the model space is instead $C^\infty_{\mathrm{nice}}(M, M)$, of smooth functions mapping the boundary of M onto itself.

Remark (Lie Group Exponential). When studying Lie groups, it is common to define the Lie group exponential map

$$\exp_G : T_{\mathrm{id}}G \to G$$

which takes elements from the tangent space of the identity element to elements in the group itself. This is similar to the Riemannian exponential map, and if we equip G with a Riemannian metric which is invariant to left- *and* right-multiplication, then the two definitions coincide.

As this is not the case for the diffeomorphism group equipped with the metrics we consider in this thesis, any use of the term exponential map throughout this thesis will refer to the Riemannian exponential map.

Chapter 4

Optimal Reparametrization of Parametric Curves

This chapter describes two algorithms for finding optimal reparametrizations of parametric curves. We start by formally defining shapes, and a distance function on shape space. After that, in section 4.2, we describe the gradient descent algorithm and present numerical results for two test examples. In section 4.3 we describe how the gradient descent algorithm has similarities with a residual neural network, and how we may use this structure to create a new algorithm for reparametrization of curves. We also provide numerical examples for the new algorithm. Finally, in 4.4 we compare the performance of the two algorithms for the given test examples.

4.1 Shape Space Metric

We consider planar curves taken from the space of immersions defined on the unit interval $I = [0, 1]$,

$$\mathcal{C} := \mathrm{Imm}(I, \mathbb{R}^2) = \{c \in C^\infty(I, \mathbb{R}^2) \mid c'(t) \neq 0,\ \forall t \in I\}. \tag{4.1}$$

To be able to identify two curves representing the same shape, we define the *reparametrization group* as the set of orientation preserving diffeomorphisms, which consists of monotonically increasing functions from I onto itself,

$$\Gamma := \mathrm{Diff}^+(I) = \{\gamma \in C^\infty(I, I) \mid \gamma(0) = 0,\ \gamma(1) = 1,\ \ \gamma'(x) > 0,\ \forall x \in I\}. \tag{4.2}$$

Γ is an infinite-dimensional Lie group with a manifold structure modeled on a subspace of the Fréchet space $C^\infty(I, \mathbb{R})$. The reparametrization group Γ has a natural *right group action* on $\mathcal{C}$ by composition

$$\mathcal{C} \times \Gamma :\to \mathcal{C} \qquad (c, \gamma) \mapsto c \circ \gamma,$$

which we will refer to as *reparametrization of c by γ* (or more generally as *reparametrization of c*).

To define the gradient of a function defined on the reparametrization group, we need to characterize the tangent space $T_\gamma\Gamma$. For this purpose consider the curve $\alpha : (-\varepsilon, \varepsilon) \to \Gamma$ passing through γ (i.e. $\alpha(0) = \gamma$), and define the map

$$\alpha^\wedge : (-\varepsilon, \varepsilon) \times I \to I, \quad (t, x) \mapsto \alpha(t)(x).$$

Since α is a curve in Γ which consists of smooth functions with fixed endpoints, then $\alpha^\wedge(t, 0) = 0$, $\alpha^\wedge(t, 1) = 1$. Therefore, the velocity

$$(\alpha)'(0) = \left.\frac{\partial}{\partial t}\right|_{t=0} \alpha^\wedge(t, \cdot)\ \ (\in C^\infty(I, \mathbb{R}))$$

satisfies

$$\left.\frac{\partial}{\partial t}\right|_{t=0} \alpha^\wedge(t, 0) = \left.\frac{\partial}{\partial t}\right|_{t=0} \alpha^\wedge(t, 1) = 0.$$

Hence the tangent space of the reparametrization group at a point γ may be identified by

$$T_\gamma\Gamma = \{v \in C^\infty(I, \mathbb{R}) \mid v(0) = v(1) = 0\}. \tag{4.3}$$

4.1.1 Shape Space

To formally define the *shape* of a parametric curve, we define an equivalence class based on the *orbits of* c w.r.t. reparametrization. Define the orbit of an element $c \in \mathcal{C}$ as the set

$$[c] = \{\hat{c} \in \mathcal{C} \mid \hat{c} = c \circ \gamma \text{ for some } \gamma \in \Gamma\},$$

and the equivalence relation

$$c_1 \sim c_2 \iff c_1 \in [c_2].$$

We say that c and $\hat{c}$ have the same shape if $c \sim \hat{c}$, and define the shape of a curve c as the equivalence class $[c]$ (i.e. the orbit of c). The *shape space* is the collection of all shapes, and corresponds to the quotient space

$$\mathcal{S} = \mathcal{C} \,/\, \Gamma.$$

Our goal is to compute distances between elements in $\mathcal{S}$, defined in terms of a metric $d_{\mathcal{S}}$. Such a metric is typically constructed by defining a metric $d_{\mathcal{C}}$ on the underlying space of curves $\mathcal{C}$, and then defining the shape distance by

$$d_{\mathcal{S}}([c_1], [c_2]) = \inf_{\gamma \in \Gamma} d_{\mathcal{C}}(c_1, c_2 \circ \gamma). \tag{4.4}$$

Hence to compute the distance between two shapes we need to solve an optimization problem. Our goal is to study gradient-based algorithms to solve this problem.

One important property of the shape distance, is that we want it to be independent of the chosen representative for each shape. One way to achieve this, is to use an underlying metric $d_{\mathcal{C}}$ which is *reparametrization invariant*, which is formally defined by the property

$$d_{\mathcal{C}}(c_1 \circ \varphi, c_2 \circ \varphi) = d_{\mathcal{C}}(c_1, c_2), \quad \forall \varphi \in \Gamma. \tag{4.5}$$

Assuming reparametrization invariance of $d_{\mathcal{C}}$, then for $\phi, \varphi \in \Gamma$,

$$\begin{aligned} d_{\mathcal{S}}([c_1 \circ \phi], [c_2 \circ \varphi]) &= \inf_{\gamma \in \Gamma} d_{\mathcal{C}}(c_1 \circ \phi, (c_2 \circ \varphi) \circ \gamma) \\ &= \inf_{\gamma \in \Gamma} d_{\mathcal{C}}(c_1, c_2 \circ \underbrace{\varphi \circ \gamma \circ \phi^{-1}}_{\in \Gamma}) \\ &= \inf_{\gamma \in \Gamma} d_{\mathcal{C}}(c_1, c_2 \circ \gamma) \\ &= d_{\mathcal{S}}([c_1], [c_2]). \end{aligned}$$

which shows that $d_{\mathcal{S}}$ is indeed independent of the chosen representatives from the shapes $[c_1], [c_2]$.

4.1.2 Introducing the Q-transform

One way to define the underlying distance function $d_{\mathcal{C}}$, is by transforming the curves into alternative representations, under which familiar metrics such as the L^2-norm is reparametrization invariant. In section 2.1.1 we mentioned some examples of such transformations used for curves, and we will be using the so-called Q-transform. This transform is closely related to the Q-transform for surfaces, which is used in the Riemannian gradient descent algorithm presented in [33].

Definition 4.1.1. *Define the* ***Q-transform*** *as the map*

$$Q : \mathcal{C} \to C^\infty(I, \mathbb{R}^2), \qquad c(\cdot) \mapsto \sqrt{|c'(\cdot)|}c(\cdot). \tag{4.6}$$

We say that for any $c \in \mathcal{C}$, the curve $q = Q(c)$ is ***the q-map or q-representation of*** *c, and we define* ***the set of q-maps*** *as*

$$\mathcal{Q} := Q(\mathcal{C}) = \{q \in C^\infty(I, \mathbb{R}^2) \mid q = Q(c) \text{ for some } c \in \mathcal{C}\}.$$

Using the Q-transform, we define a metric on $\mathcal{C}$ by

$$d_{\mathcal{C}}(c_1, c_2) := \|Q(c_1) - Q(c_2)\|_{L^2(I,\mathbb{R}^2)} = \left(\int_I |Q(c_1)(t) - Q(c_2)(t)|^2 \, dt \right)^{1/2}.$$

Before proceeding to show that this is a reparametrization invariant metric, we derive the following property for the Q-transform:

$$Q(c \circ \gamma)(t) = \sqrt{|c'(\gamma(t))||\gamma'(t)|}\, c(\gamma(t)) = \sqrt{\gamma'(t)}\, Q(c)(\gamma(t)) = \sqrt{\gamma'(t)}\, (Q(c) \circ \gamma)(t). \tag{4.7}$$

By inserting this into the distance function $d_{\mathcal{C}}$,

$$\begin{aligned}
d_{\mathcal{C}}(c_1 \circ \varphi, c_2 \circ \varphi) &= \left(\int_I |Q(c_1 \circ \varphi)(t) - Q(c_2 \circ \varphi)(t)|^2 \, dt \right)^{1/2} \\
&= \left(\int_I |\sqrt{\varphi'(t)}\, Q(c_1)(\varphi(t)) - \sqrt{\varphi'(t)}\, Q(c_2)(\varphi(t))|^2 \, dt \right)^{1/2} \\
&= \left(\int_I \dot{\varphi}(t) |Q(c_1 \circ \varphi)(\tilde{t}) - Q(c_2 \circ \varphi)(\tilde{t})|^2 \, \frac{1}{\dot{\varphi}(t)} d\tilde{t} \right)^{1/2} \\
&= \left(\int_I |Q(c_1)(\tilde{t}) - Q(c_2)(\tilde{t})|^2 \, d\tilde{t} \right)^{1/2} \\
&= d_{\mathcal{C}}(c_1, c_2),
\end{aligned}$$

which shows that that $d_{\mathcal{C}}$ is reparametrization invariant. In light of (4.7), we also define a right action of the reparametrization group on the set of q-maps, by

$$\mathcal{Q} \times \Gamma \to \mathcal{Q}, \qquad (q, \gamma) \mapsto \sqrt{\gamma'}(q \circ \gamma),$$

and the orbit of $q \in \mathcal{Q}$ by

$$[q] = \{r \in \mathcal{Q} \mid r = \sqrt{\gamma'}(q \circ \gamma) \text{ for some } \gamma \in \Gamma\}.$$

Note that we have defined the right action of Γ on $\mathcal{Q}$ such that for $c \in \mathcal{C}$ and $q = Q(c) \in \mathcal{Q}$, we have $[q] = Q([c])$. Lastly we define the orbit map for a given $r \in \mathcal{Q}$, by

$$\phi^r : \Gamma \to [r], \quad \phi^r(\gamma) = \sqrt{\gamma'}(r \circ \gamma).$$

4.2 Gradient Descent on the Reparametrization Group

In this section, we describe the gradient descent algorithm over the reparametrization group for curves. In short, it works by iteratively reparametrizing one of the curves by small perturbations of the identity-diffeomorphism. It is based on the algorithm described in [33] for surfaces.

Assume that we are given two curves $c_1, c_2 \in \mathcal{C}$, and want to find a reparametrization minimizing the shape distance. Instead of directly minimizing the shape distance, we will be working with its square. Denote by $q = Q(c_1)$ and $r_0 = Q(c_2)$, and define for any $r \in [r_0]$ the cost function

$$E^r : \Gamma \to \mathbb{R}, \quad E^r(\gamma) = \|q - \phi^r(\gamma)\|^2_{L^2(I,\mathbb{R}^2)}$$

Our goal is to minimize the cost function E^{r_0} by iteratively reparametrizing r_0 according to

$$r_n = \phi^{r_{n-1}}(\gamma_n) = \sqrt{\gamma_n'}(r_{n-1} \circ \gamma_n), \quad n = 1, 2, ...,$$

for a sequence of diffeomorphisms

$$\gamma_n = \mathrm{id} - \eta_n \nabla E^{r_{n-1}}(\mathrm{id}).$$

Here $\eta_n > 0$ is the step size in each iteration, and $\nabla E^{r_n}(\mathrm{id})$ is the Riemannian gradient of E^{r_n} at the identity. This procedure is repeated until convergence, and assuming that the algorithm terminates after m iterations, then the optimal reparametrization is given by

$$\bar{\gamma} = \gamma_1 \circ \gamma_2 \circ ... \circ \gamma_m.$$

It might seem unintuitive that we are using the gradient with respect to a sequence of cost functions E^{r_n} that changes at each iteration of the algorithm. However, in section 4.2.5, we relate this approach to a proper Riemannian gradient descent algorithm for the original problem E^{r_0}.

4.2.1 Computing The Gradient of E^r

To simplify the computation of the gradient, define

$$F : C^\infty(I, \mathbb{R}^2) \to \mathbb{R}, \qquad F(s) = \|q - s\|^2_{L^2(I,\mathbb{R}^2)}$$

with differential

$$dF_s : C^\infty(I, \mathbb{R}^2) \to \mathbb{R}, \qquad dF_s(h) = -2\langle q - s, h\rangle_{L^2(I,\mathbb{R}^2)}.$$

Then $E^r = F \circ \phi^r$, such that the directional derivative $dE^r_\gamma : T_\gamma\Gamma \to \mathbb{R}$ is given by the chain rule,

$$dE^r_\gamma(v) = dF_s(d\phi^r_\gamma(v)) = -2\left\langle q - \phi^r(\gamma), d\phi^r_\gamma(v)\right\rangle_{L^2(I,\mathbb{R}^2)}. \tag{4.8}$$

To find the differential of ϕ^r, consider the curve

$$\alpha : (-\varepsilon, \varepsilon) \to \Gamma, \quad \alpha(t) = \gamma + tv$$

passing through $\gamma \in \Gamma$ with $v = \alpha'(0) \in T_\gamma\Gamma$. The differential of the orbit map

$$d\phi^r_\gamma : T_\gamma\Gamma \to T_{\phi^r(\gamma)}[r] \ \left(\subset C^\infty(I, \mathbb{R}^2)\right),$$

is defined by

$$\begin{aligned} d\phi^r_\gamma(v) &= \left.\frac{d}{dt}\right|_{t=0} \sqrt{\alpha'(t)}\,(r \circ \alpha(t)) = \left.\frac{d}{dt}\right|_{t=0} \sqrt{\gamma' + tv'}\,(r \circ (\gamma + tv)) \\ &= \frac{1}{2\sqrt{\gamma'}}\left(\left.\frac{d}{dt}\right|_{t=0} (\gamma' + tv')\right)(r \circ \gamma) + \sqrt{\gamma'}(r' \circ \gamma)\left.\frac{d}{dt}\right|_{t=0} (\gamma + tv) \\ &= \frac{1}{2\sqrt{\gamma'}} v'(r \circ \gamma) + \sqrt{\gamma'}(r' \circ \gamma)v. \end{aligned} \tag{4.9}$$

By letting $\gamma = \mathrm{id}$, (4.9) is reduced to

$$d\phi^r_{\mathrm{id}}(v) = \frac{1}{2}v'r + vr', \tag{4.10}$$

which inserted into (4.8) gives

$$dE^r_{\mathrm{id}}(v) = -2\langle q - r, \frac{1}{2}v'r + vr'\rangle_{L^2(I,\mathbb{R}^2)}. \tag{4.11}$$

The differential $dE^r_{\mathrm{id}}(v)$ is a linear functional from a subspace of the Hilbert space $L^2(I, \mathbb{R})$, and by Riesz representation theorem, there exists a unique element $\delta E^r_{\mathrm{id}} \in L^2(I, \mathbb{R})$ such that

$$dE^r_{\mathrm{id}}(v) = \langle \delta E^r_{\mathrm{id}}, v\rangle_{L^2(I,\mathbb{R})}.$$

By applying integration by parts to (4.11), and using the fact that elements of $T_{\mathrm{id}}\Gamma$ vanish at the boundary, we may show that the element δE^r_{id}, which we call the *functional gradient* is given by

$$\delta E^r_{\mathrm{id}} = r^T q' - q^T r'.$$

At first sight, it is tempting to declare this function as the Riemannian gradient of E^r at the identity. However, it is only guaranteed to be an element of the larger space $L^2(I, \mathbb{R})$, and will generally not be an element of the tangent space $T_{\mathrm{id}}(\Gamma)$. To deal with this, we will need to project the gradient onto the tangent space at the identity of the reparametrization group.

4.2.2 Gradient Projection

Assume that we are given a basis $\{v_i\}_{i\in\mathbb{N}}$ for $T_{\mathrm{id}}\Gamma$, which is orthonormal with respect to an inner product $\langle\langle\cdot,\cdot\rangle\rangle$. By equipping the tangent space with this inner product, the Riemannian gradient at the identity is defined as the element $\nabla E^r(\mathrm{id})$ satisfying

$$dE^r_{\mathrm{id}}(v) = \langle \delta E^r_{\mathrm{id}}, v\rangle_{L^2(I,\mathbb{R})} = \langle\langle \nabla E^r(\mathrm{id}), v\rangle\rangle.$$

Writing $\nabla E^r(\mathrm{id})$ in terms of the basis, we want to find a set of coefficients a_i such that

$$\langle \delta E^r_{\mathrm{id}}, v\rangle_{L^2(I,\mathbb{R})} = \langle\langle \nabla E^r(\mathrm{id}), v\rangle\rangle = \langle\langle \sum_{i\in\mathbb{N}} a_i v_i, v\rangle\rangle = \sum_{i\in\mathbb{N}} a_i \langle\langle v_i, v\rangle\rangle .$$

Specifically, if we let $v = v_k$, then by the orthonormality of $\{v_i\}_{i\in\mathbb{N}}$,

$$\langle \delta E^r_{\mathrm{id}}, v_k\rangle_{L^2(I,\mathbb{R})} = \sum_{i\in\mathbb{N}} a_i \langle\langle v_i, v_k\rangle\rangle = a_k.$$

Thus the Riemannian gradient of E^r with respect to the inner product $\langle\langle\cdot,\cdot\rangle\rangle$, is at the identity given by

$$\nabla E^r(\mathrm{id}) = \sum_{i\in\mathbb{N}} v_i \langle \delta E^r_{\mathrm{id}}, v_i\rangle_{L^2(I,\mathbb{R})} = \sum_{i\in\mathbb{N}} v_i \langle r^T q' - q^T r', v_i\rangle_{L^2(I,\mathbb{R})}.$$

In practice, we need to approximate the infinite sum in the projection step by using a finite orthonormal basis $(v_i)_{i=1}^N$, spanning some subspace $V \subset T_{\mathrm{id}}\Gamma$. In the following, we present three alternative bases.

Truncated Fourier Basis The arguably simplest choice of basis is a truncated Fourier sine series. These basis functions are given on the form

$$v_n = \sqrt{2}\sin(n\pi x), \quad n \in \mathbb{N},$$

and are orthonormal with respect to the L^2 inner product. These functions are clearly smooth, and vanish on the boundaries of the interval. By choosing some $N \in \mathbb{N}$ representing the maximal frequency of the sines, the truncated Fourier sine basis spans a subspace

$$V_F := \mathrm{span}\{(v_i)_{i=1}^N\}.$$

Jacobi Polynomials The Jacobi polynomials $P_n^{(\alpha,\beta)}$ are a family of polynomials that are orthogonal on the interval $[-1,1]$, with respect to a weight function $x \mapsto (1-x)^\alpha(1+x)^\beta$ for $\alpha,\beta > -1$. While the Jacobi polynomials themselves do not vanish at the boundaries of the interval, we may use the weight function to construct a set of polynomials, spanning some subspace V_J of $T_{\mathrm{id}}\Gamma$, that are orthonormal with respect to the L^2 inner product. For simplicity we will only consider the case where $\alpha = \beta = 2$. Define the weight function

$$w(z) = (1-z)^2(1+z)^2,$$

and the polynomials

$$\tilde{p}_n(z) = (1-z)(1+z)P_n^{(2,2)}(z), \quad n \in \mathbb{N}_0. \tag{4.12}$$

By the orthogonality of the Jacobi polynomials with respect to w, then for all $n, m \in \mathbb{N}_0$,

$$\begin{aligned}
\langle P_n^{(2,2)}, P_m^{(2,2)}\rangle_w &:= \int_{-1}^{1} (1-z)^2(1+z)^2 P_n^{(2,2)}(z) P_m^{(2,2)}(z)\, dz \\
&= \int_{-1}^{1} \tilde{p}_n(z)\tilde{p}_m(z)\, dz \\
&= \langle \tilde{p}_n, \tilde{p}_m\rangle_{L^2([-1,1],\mathbb{R})} \\
&= \delta_{n,m}
\end{aligned}$$

where $\delta_{n,m}$ defines the Kronecker delta. Hence the polynomials from (4.12) are orthogonal with respect to the L^2 inner product on the interval $[-1,1]$. Before they may be used to construct an orthonormal set in $T_{\mathrm{id}}\Gamma$, we first need to normalize them by

$$p_n = \frac{\tilde{p}_n}{\|\tilde{p}_n\|_{L^2([-1,1])}},$$

and then map them onto the interval I. For this purpose, define the map

$$\Phi : [0,1] \to [-1,1], \qquad \Phi(x) = 2x - 1,$$

such that

$$\delta_{n,m} = \langle p_n, p_m \rangle_{L^2([-1,1])} = \int_{-1}^{1} p_n(z)p_m(z)\, dz = \int_0^1 p_n(\Phi(x))p_m(\Phi(x))\, 2dx$$
$$= \langle \sqrt{2}(p_n \circ \Phi), \sqrt{2}(p_m \circ \Phi) \rangle_{L^2(I,\mathbb{R})}.$$

Thus the set of polynomials $\{v_n\}_{n\in\mathbb{N}_0}$ defined by

$$v_n(x) = \sqrt{2}(p_n \circ \Phi)(x) = \sqrt{2}\, p_n(2x-1)$$

is an orthonormal set with respect to the L^2 inner product over the interval I. While we will leave out the details in the derivation, it may be shown (see e.g. [57]) that these polynomials may be expressed by

$$v_n(x) = C_n x(1-x) \sum_{m=0}^{n} B_n^m (x-1)^m$$

$$C_n = \sqrt{\frac{2n+5}{(n+1)(n+2)(n+3)(n+4)}}, \qquad B_n^m = \frac{\prod_{k=m+3}^{n+m+4} k}{m!(n-m)!}.$$

Note that these functions vanish at the boundaries of I, and as polynomials they are clearly smooth. Hence the set

$$V_J := \operatorname{span}\{v_n\}_{n=0}^{N}$$

forms a finite dimensional subspace of the tangent space $T_{\mathrm{id}}\Gamma$.

Palais Basis Both of the preceding bases use functions with large oscillations, which means large derivatives. This may cause problems in the gradient descent algorithm, due to the positive derivative constraint on the reparametrization group. As we will see in the next section, this constraint limits the largest allowed step size that can be performed in the direction of the gradient, without stepping outside of the reparametrization group. This motivates the use of orthogonal bases with respect to a Sobolev-type metric, taking derivatives into account. One such basis is a version of the Fourier series which is orthogonal with respect to a first-order *Palais metric* [45], given by

$$\langle u, v \rangle_s = u(0)v(0) + \int_I u'(x)v'(x)\, dx.$$

Forming an orthonormal basis for the tangent space with respect to this metric is simple: Since the basis functions should vanish at the boundary, the first term in the Palais metric will always be zero. Thus we only need to find a set of functions that vanish at the boundaries of I, whose derivatives are orthonormal with respect to the L^2-metric. In [49, 50] such a basis is formed using the trigonometric functions

$$v_n^{(1)}(x) = \frac{1}{\sqrt{2}\pi n}\sin(2\pi n x), \quad v_n^{(2)}(x) = \frac{1}{\sqrt{2}\pi n}\left(\cos(2\pi n x) - 1\right).$$

The presence of n in the denominator prevents large derivatives in the higher-order elements. Once again we fix some $N \in \mathbb{N}$ representing the maximal frequency of the basis functions and form a basis of $2N$ elements spanning a subspace

$$V_P := \operatorname{span}\{v_n^{(1)}, v_n^{(2)}\}_{n=1}^{N}.$$

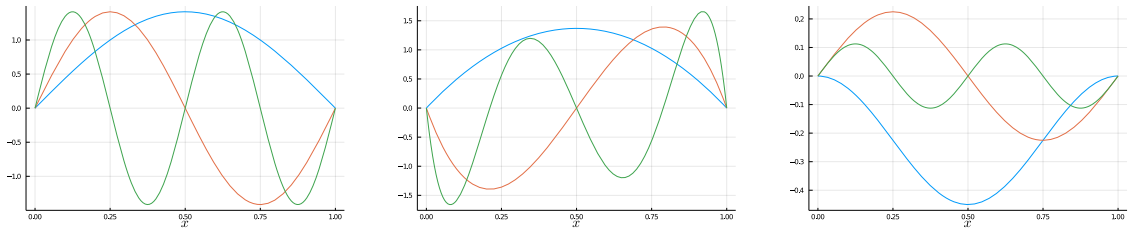
4.2.3 Step Size Bounds

After projection, we have an expression for the Riemannian gradient at the identity, and we want to take a small step in its direction. Since both the reparametrization group and its tangent space form subspaces of $C^\infty(I,\mathbb{R})$, we may define a retraction map at the identity by

$$\rho_{\mathrm{id}} : T_{\mathrm{id}}\Gamma \subset C^\infty(I,\mathbb{R}) \to \Gamma, \qquad \rho_{\mathrm{id}}(v) = \mathrm{id} + v$$

with pointwise addition. Then a single step of a Riemannian gradient descent algorithm for E^r, starting at $\gamma_0 = \mathrm{id}$, is given by

$$\gamma_1 = \rho_{\mathrm{id}}(-\eta \nabla E^r(\mathrm{id})) = \mathrm{id} - \eta \nabla E^r(\mathrm{id}). \tag{4.13}$$



(a) Fourier Basis (b) Jacobi Polynomial Basis (c) Palais Basis

Figure 4.1: Example functions for each of the three bases from section 4.2.2.

However, if the step size η is too large, the positive derivative constraint on the reparametrization group may be violated, causing us to step outside of Γ. To simplify notation, denote by $v = -\nabla E^r(\mathrm{id})$[1]. We want to find some $\bar{\eta} > 0$ such that for any $\eta < \bar{\eta}$

$$\gamma_1'(x) = 1 + \eta v'(x) > 0 \quad \forall x \in I. \tag{4.14}$$

In any point x such that $v'(x) \geq 0$, $\gamma_1'(x)$ is positive for any $\eta > 0$. If we assume that $v \not\equiv 0$, however, there exists some $x \in I$ such that $v'(x) < 0$. To ensure that $\gamma_1'(x)$ is non-negative in such a point, η must satisfy

$$1 - \eta|v'(x)| \geq 0 \implies \eta \leq \frac{1}{|v'(x)|} = \frac{1}{-v'(x)}.$$

The upper bound $\bar{\eta}$ is the smallest, postive step size which satisfies the above relation for *every* $x \in I$. Thus for

$$\bar{\eta} = \frac{-1}{\min_{x \in I} v'(x)}$$

we have $1 + \bar{\eta} v(x) \geq 0 \, \forall x \in I$, and (4.14) is satisfied for every $\eta < \bar{\eta}$.

The upper bound $\bar{\eta}$ is formulated in terms of an optimization problem. We will be satisfied to find a simple estimate $\tilde{\eta}$ for this minimum. Consider a set of linearly spaced points $x_i = i/K$ for $i = 0, ..., K$ on I. Denote by $h = 1/K$, and define intervals around each of the points by

$$I_i = \left[x_i - \frac{h}{2}, x_i + \frac{h}{2} \right].$$

Now assume that the minimizer $\bar{x} = \operatorname{argmin}_{x \in I} v'(x)$ is contained in I_k. Using that $v''(\bar{x}) = 0$ [2], then by Taylors formula with integral remainder,

$$v'(x_k) = v'(\bar{x}) + \frac{1}{2} \int_{\bar{x}}^{x_k} v'''(\xi)(x - \xi)^2 \, d\xi \leq v'(\bar{x}) + \underbrace{\frac{h^3}{16} \max_{x \in I} |v'''(\xi)|}_{=:\varepsilon}.$$

By reordering the elements in the above equation, and using that $\min_{x \in I} v'(x) < 0$,

$$|v'(x_k) - \varepsilon| \leq |v'(\bar{x})| \implies \tilde{\eta} := \frac{1}{|\min_{i=0,...,K} v(x_i) - \varepsilon|} \leq \bar{\eta} \tag{4.15}$$

Thus $\tilde{\eta}$ gives a simple, practical upper bound for the step size η. However, for the purpose of the gradient descent algorithm, we rarely want to take a single step that takes us all the way to the boundary of Γ. Therefore we choose some *relative step size* $\alpha \in (0, 1)$, and use $\eta = \alpha \tilde{\eta}$ as the chosen step size in (4.13).

4.2.4 Gradient Descent Formulation

Putting the previous steps together, we sum up the gradient descent algorithm for finding an optimal reparametrization by the pseudocode in Algorithm 1. After finding the optimal reparametrization $\bar{\gamma}$, the final shape distance may thus be computed by

$$d_{\mathcal{S}}([c_1], [c_2]) = \|Q(c_1) - Q(c_2 \circ \bar{\gamma})\|_{L^2(I, \mathbb{R}^2)}.$$

[1] The minus sign allow us to reuse the argument here in the deep reparametrization chapter.

[2] Here we have assumed that $\bar{x}$ is an internal point of I. If this is not the case, the derived bound will still work, since the endpoints are included in the gridpoints x_i.

Algorithm 1 Gradient Descent for Optimal Reparametrization of Curves

Require: $q, r_0 \in \mathcal{Q}$: q-maps of parametric curves.
Require: $\{v_i\}_{i=1}^N$: Orthonormal basis for $V \subset T_{\mathrm{id}}\Gamma$.
1: **for** $n = 0, 1, 2, \ldots$ **do** ▷ until convergence criterion is met
2: $\quad \delta E^{r_n} \leftarrow r_n^T q' - q^T r_n'$
3: $\quad \nabla E^{r_n}_{\mathrm{id}} \leftarrow \mathrm{project}(\delta E^{r_n}_{\mathrm{id}}, \{v_i\}_{i=1}^N)$
4: $\quad \eta \leftarrow \mathrm{step\_select}(E^{r_n}, -\nabla E^{r_n}_{\mathrm{id}})$ ▷ max step size by (4.15), and backtracking
5: $\quad \gamma_{n+1} \leftarrow \mathrm{id} - \eta \nabla E^{r_n}_{\mathrm{id}}$
6: $\quad r_{n+1} \leftarrow \phi^{r_n}(\gamma_n)$
7: **end for**
8: **return** $\gamma_0 \circ \gamma_1 \circ \ldots$

Now we want to show that the algorithm actually corresponds to a Riemannian gradient descent algorithm for the original problem E^{r_0}.

Theorem 1. *Assume that $\langle \cdot, \cdot \rangle_{T_{id}\Gamma}$ is an inner product on the tangent space $T_{id}\Gamma$, and denote by*

$$L_\gamma : \Gamma \to \Gamma, \quad L_\gamma(\varphi) = \gamma \circ \varphi$$

the left composition by γ. Then Algorithm 1 is equivalent to a Riemannian gradient descent algorithm with update rule

$$\gamma^{(n+1)} = \rho_{\gamma^{(n)}}(-\eta \nabla E^{r_0}(\gamma^{(n)})),$$

where ρ is a retraction which assigns to each $\gamma \in \Gamma$ the map

$$\rho_\gamma : T_\gamma\Gamma \mapsto \Gamma, \quad \rho_\gamma(v_\gamma) = \gamma \circ (id + d\,(L_\gamma)^{-1}_{id}\,(v_\gamma)),$$

and $\nabla E^{r_0}(\gamma)$ is the Riemannian gradient of E^{r_0} with respect to the Riemannian metric defined by

$$\langle u, v \rangle_{T_\gamma\Gamma} = \left\langle d(L_\gamma)^{-1}_{id} u, d(L_\gamma)^{-1}_{id} v \right\rangle_{T_{id}\Gamma}.$$

Proof. Given $q, r_0 \in \mathcal{Q}$, Algorithm 1 corresponds to an iteration on the form on the form

$$\gamma^{(n+1)} = \gamma^{(n)} \circ (\mathrm{id} - \eta \nabla E^{r_n}(\mathrm{id})) \tag{4.16}$$

where $r_n = \phi^{r_{n-1}}(\gamma_n) = \phi^{r_0}(\gamma^{(n)})$, and

$$E^{r_n}(\gamma) = \|q - \phi^{r_n}(\gamma)\|^2_{L^2(I,\mathbb{R})}.$$

First we see that the orbit map on $\mathcal{Q}$ satisfies

$$\phi^{\phi^r(\gamma)}(\varphi) = \sqrt{\varphi'}\sqrt{\gamma' \circ \varphi}\,((r \circ \gamma) \circ \varphi) = \sqrt{(\gamma \circ \varphi)'}(r \circ \gamma \circ \varphi) = \phi^r(\gamma \circ \varphi) = (\phi^r \circ L_\gamma)(\varphi),$$

such that

$$E^{\phi^r(\gamma)}(\varphi) = \|q - \phi^{\phi^r(\gamma)}(\varphi)\|^2_{L^2(I,\mathbb{R}^2)} = \|q - \phi^r(\gamma \circ \varphi)\|^2_{L^2(I,\mathbb{R}^2)} = E^r(\gamma \circ \varphi).$$

Thus the sequence of cost functions is related to the original problem by $E^{r_n}(\gamma) = E^{r_0}(\gamma^{(n)} \circ \gamma)$.

Next, we relate the gradients $\nabla E^{r_n}(\mathrm{id})$, to the Riemannian gradient $\nabla E^{r_0}(\gamma)$ of the original problem. To keep track of which tangent space a vector belongs to, we will use the notation $v_\gamma \in T_\gamma\Gamma$. Consider the derivative of the left multiplication

$$d(L_\gamma)_\varphi : T_\varphi\Gamma \to T_{\gamma\circ\varphi}\Gamma, \quad d(L_\gamma)_\varphi(v_\varphi) = (\gamma' \circ \varphi)v_\varphi$$

and its inverse

$$d(L_\gamma)^{-1}_\varphi : T_{\gamma\circ\varphi}\Gamma \to T_\varphi\Gamma, \quad d(L_\gamma)^{-1}_\varphi(v_\varphi) = \frac{1}{\gamma' \circ \varphi} v_{\gamma\circ\varphi}.$$

This gives for the derivative of the orbit map

$$d\phi^{\phi^r(\gamma)}_\varphi = d\phi^r_\gamma \circ d(L_\gamma)_\varphi \iff d\phi^r_\gamma = d\phi^{\phi^r(\gamma)}_\varphi \circ d(L_\gamma)^{-1}_\varphi.$$

Inserting this into the derivative for E^r we get

$$\begin{aligned} dE^r_\gamma(v_\gamma) &= \left\langle q - \phi^r(\gamma), d\phi^r_\gamma(v_\gamma) \right\rangle_{L^2(I,\mathbb{R})} \\ &= \left\langle q - \phi^{\phi^r(\gamma)}(\mathrm{id}), d\phi^{\phi^r(\gamma)}_{\mathrm{id}} \left(d(L_\gamma)^{-1}_{\mathrm{id}}(v_\gamma) \right) \right\rangle_{L^2(I,\mathbb{R})} \\ &=^3 \left\langle \nabla E^{\phi^r(\gamma)}(\mathrm{id}), d(L_\gamma)^{-1}_{\mathrm{id}} v_\gamma \right\rangle_{T_{\mathrm{id}}\Gamma} \\ &= \left\langle d(L_\gamma)^{-1}_{\mathrm{id}} \left(d(L_\gamma)_{\mathrm{id}} \left(\nabla E^{\phi^r(\gamma)}(\mathrm{id}) \right) \right), d(L_\gamma)^{-1}_{\mathrm{id}} v_\gamma \right\rangle_{T_{\mathrm{id}}\Gamma} \\ &= \left\langle d(L_\gamma)_{\mathrm{id}} \left(\nabla E^{\phi^r(\gamma)}(\mathrm{id}) \right), v_\gamma \right\rangle_{T_\gamma\Gamma}. \end{aligned}$$

Then by definition

$$\nabla E^r(\gamma) = d(L_\gamma)_{\mathrm{id}} \left(\nabla E^{\phi^r(\gamma)}(id) \right) = \gamma' \, \nabla E^{\phi^r(\gamma)}(\mathrm{id}).$$

Finally, to check that the map ρ is indeed a retraction, we write out

$$\rho_\gamma(v_\gamma) = \gamma \circ (\mathrm{id} + d(L_\gamma)^{-1}_{\mathrm{id}}(v_\gamma)) = \gamma \circ \left(\mathrm{id} + \frac{1}{\gamma'} v_\gamma \right),$$

and evaluate ρ_γ at the zero-vector

$$\rho_\gamma(0) = \gamma \circ (\mathrm{id} + 0) = \gamma \circ \mathrm{id} = \gamma.$$

Moreover, since the domain of the retraction map at a point is a *vector space* $T_\gamma\Gamma$, we may define a curve on $T_\gamma\Gamma$ passing through 0 with velocity $u_\gamma \in T_\gamma\Gamma$ by $t \mapsto t u_\gamma$. Then

$$d(\rho(\gamma))_0(u_\gamma) = \left. \frac{d}{dt} \right|_{t=0} \gamma \circ \left(\mathrm{id} + \frac{t}{\gamma'} u_\gamma \right) = (\gamma' \circ \mathrm{id}) \frac{1}{\gamma'} u_\gamma = u_\gamma$$

which means that $d(\rho_\gamma)_0 = \mathrm{id}|_{T_\gamma\Gamma}$, and ρ does indeed define a retraction map. Inserting all of this into the update rule,

$$\begin{aligned} \gamma^{(n+1)} &= \rho_{\gamma^{(n)}}(-\eta \nabla E^r(\gamma^{(n)})) \\ &= \gamma^{(n)} \circ \left(\mathrm{id} - \eta \frac{1}{(\gamma^{(n)})'} (\gamma^{(n)})' \, \nabla E^{\phi^r(\gamma^{(n)})}(\mathrm{id}) \right) \\ &= \gamma^{(n)} \circ \left(\mathrm{id} - \eta \nabla E^{\phi^r(\gamma^{(n)})}(\mathrm{id}) \right) \\ &= \gamma^{(n)} \circ (\mathrm{id} - \eta \nabla E^{r_n}(\mathrm{id})) \end{aligned}$$

which corresponds to the update (4.16). □

4.2.5 Implementation

The algorithm was implemented using the Julia language [11]. The language treats functions as so-called *first-class citizen*, that may be passed as arguments, and returned as a value from other functions. This allows us to implement the algorithm using actual functions rather than finite-dimensional approximations, and makes the implementation as closely related to the described algorithm as possible. It does, however, increase the computational cost, and as the length of the chain of compositions increases, so does the computational cost of each iteration. We will see that this is manageable for curves but will become a problem when we extend the algorithms to surfaces in section 5.2. The source code for the implementation is available on GitHub[4].

Backtracking

Since the computational cost increases with each iteration, we want to choose the step size in a way that ensures sufficient progress with each reparametrization. For this purpose, we will use

[3] This comes from the original computation of the gradient at the identity in section 4.2.2

[4] https://github.com/jorgenriseth/Reparam.jl

a backtracking line search algorithm, which ensures that the step satisfies the *Armijo-Goldstein condition* [2], which in this case corresponds to

$$E^{r_n}(\mathrm{id} - \eta \nabla E^{r_n}(\mathrm{id})) \leq E^{r_n}(\mathrm{id}) - \eta c \|\nabla E^{r_n}(\mathrm{id})\|^2_{L^2(I,\mathbb{R})} \tag{4.17}$$

for some $c \in (0,1)$. To achieve this, we chose an initial step size η as described in 4.2.3. If the Armijo-Goldstein condition is not satisfied, we scale down the step size by a factor $\rho \in (0,1)$ and check again. This process is repeated until we find a step size which satisfies (4.17). If no such step is found within a maximum number of iterations, we terminate the algorithm.[5]

Quadrature

Since the algorithm is implemented using functions, rather than finite-dimensional representations, we need to approximate the integrals occurring in the function norms and inner products by a quadrature rule. For this purpose, we use the Gauss-Legendre quadrature rule defined by

$$\int_0^1 f(x)\, dx \approx \frac{1}{2} \sum_{k=1}^{K} w_k f(x_k)$$

where w_k are *quadrature weights*, and x_k are the *quadrature nodes*, corresponding to affine transformations of the roots of the Legendre polynomials. The nodes and weights were found using the package `FastGaussQuadrature.jl` [59].

Automatic Differentiation

The Q-transform, orbit map ϕ^r, and the expression for the functional gradient, all contain the derivatives of functions and curves. To compute derivatives, we use the `ForwardDiff.jl` package for so-called *forward-mode automatic differentiation*. The package implements a multidimensional version of a *dual number*, which in the one-dimensional case is defined by an element $x + \epsilon y$ whose behaviour satisfies

$$f(x + \epsilon y) = f(x) + f'(x)\epsilon$$

and $\epsilon^2 = 0$, when used as input for a function f. Most functions evaluated by a computer may be broken down into simple functions whose derivatives are easily evaluated. By defining the behaviour of these simple functions when applied to a dual number, we may retrieve the derivative of the original function as the composition of these elementary operations. Due to Julia's support of *multiple dispatch* i.e., that the type of the input values determines a function's behaviour, then most user-defined code automatically works with dual numbers as well. For further details regarding implementation and the extension to multiple dimensions, we refer to the paper [47] by the authors of the package.

Termination

To check for convergence of the algorithm, we check if the relative change in the cost function

$$\frac{E^r(\gamma_n) - E^r(\gamma_{n+1})}{E^r(\gamma_n)}$$

is smaller than some tolerance. In addition, we stop the algorithm after a fixed number of iterations, if the termination citerion is not met.

4.2.6 Numerical Results

This section presents numerical results from using the gradient descent algorithm. Firstly, we will compare two curves representing the same shape, and see if we can reparametrize one curve to match the other. Thereafter we will compare two curves representing different shapes for which we have an analytical solution. The algorithm parameters were chosen heuristically as a set that has shown to perform well for various curves and are listed in Table 4.1.

[5] Under sufficient regularity of the cost function, there is a theoretical guarantee to find a step size satisfying the Armijo condition within a finite number of steps using exact arithmetic. However, due to approximation and rounding errors, we also use a maximal number to ensure termination.

Parameter	Value
Relative Step Size α	0.1
Quadrature Points K	128
Backtracking Coefficient c	0.9
Backtracking Coefficient ρ	0.9
Relative Error Tolerance	10^{-6}
Max Iterations	200

Table 4.1: List of parameters used in the gradient descent algorithm for curves throughout this section.

Same Shape Comparison

In this section we compare two parametric curves representing the same shape. Start out by defining the curve

$$c_2(x) = [\cos(2\pi x), \sin(4\pi x)], \tag{4.18}$$

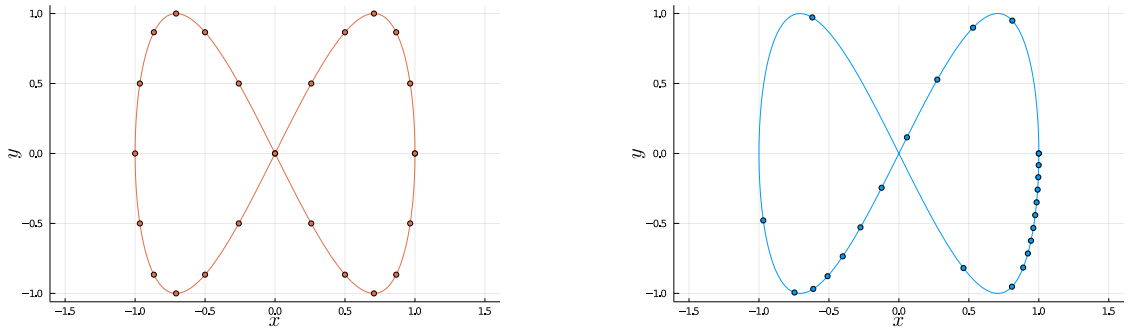
and then reparametrize it by a function ψ to get another curve

$$c_1(x) = (c_2 \circ \psi)(x)$$

from the same equivalence class. We will be using the diffeomorphism

$$\psi(x) = \frac{\log(20x+1)}{2\log(21)} + \frac{1+\tanh(20(x-0.5))}{4\tanh(10)} \tag{4.19}$$

which will present a challenge for the algorithm. The two curves are illustrated in figure 4.2. The dots along the curves represent linearly spaced points in the interval $I = [0, 1]$.



(a) c_2 (b) $c_1 = c_2 \circ \psi$

Figure 4.2: Two curves representing the same shape. *(left)* The curve c_2 (4.18). *(right)* The curve c_2 reparametrized by the diffeomorphism (4.19).

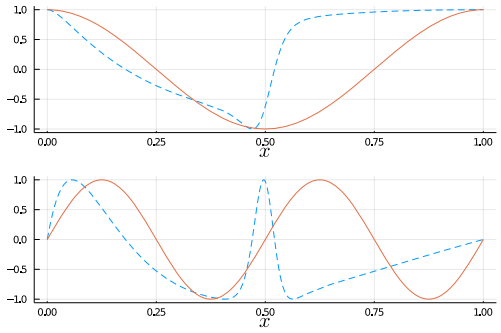
Now let $q = Q(c_1) = Q(c_2 \circ \psi)$ and $r = Q(c_2)$. The optimal reparametrization of r is of course ψ, and by applying the gradient descent algorithm to these curves, the resulting diffeomorphism $\bar{\gamma}$ should ideally be as close to ψ as possible. Moreover, since the shape distance should be zero, we may use the cost function

$$E^r(\bar{\gamma}) = \|q - \sqrt{\bar{\gamma}'}(r \circ \bar{\gamma})\|^2_{L^2(I)}$$

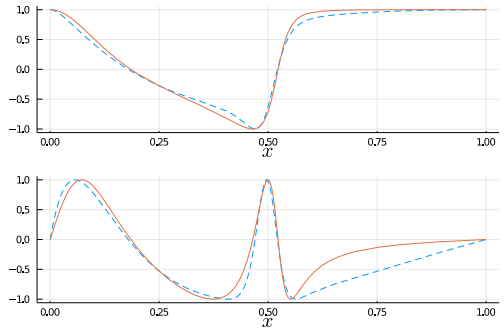
as a measure of the error of the parametrization.

We start by testing the algorithm using the truncated Fourier sine basis, with only three basis functions for the gradient projection. The results are presented in figures 4.3 and 4.4. The algorithm comes a long way towards finding an optimal reparametrization, and significantly reduce the estimated shape distance.

In figure 4.5, we compare the final error when using different types and number of basis functions. In this case, all three basis functions seem to reach its best performance with as few as 3-4 basis elements, and the error increases for a larger number of elements. The performance of the Palais basis seems to be more stable with respect to the number of basis functions used.

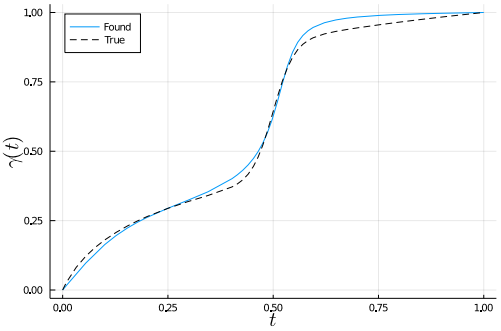


(a) Before Reparametrization

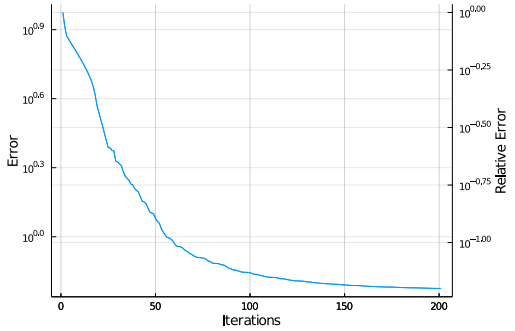


(b) After Reparametrization

Figure 4.3: Comparison of the 1st *(top)* and 2nd *(bottom)* coordinate of the two curves. The dashed blue line represents c_1, while the orange line represents c_2 before and after reparametrization.



(a) Comparison of the true reparametrization ψ and the one found by the reparametrization algorithm.



(b) The cost function E^r versus number of iterations. The initial error is 9.433, while the final error is 0.5956. a reduction to 6.3% of the original error.

Figure 4.4

The increase in the error when adding basis elements may be attributed to the fact that the search space increases significantly by adding dimensions to the problem, which increases the probability that the algorithm falls into a local minimum. For the Fourier and Jacobi bases, the increased number of basis elements also reduce the allowed step size in the parametrization updates, preventing convergence within a reasonable amount of time.

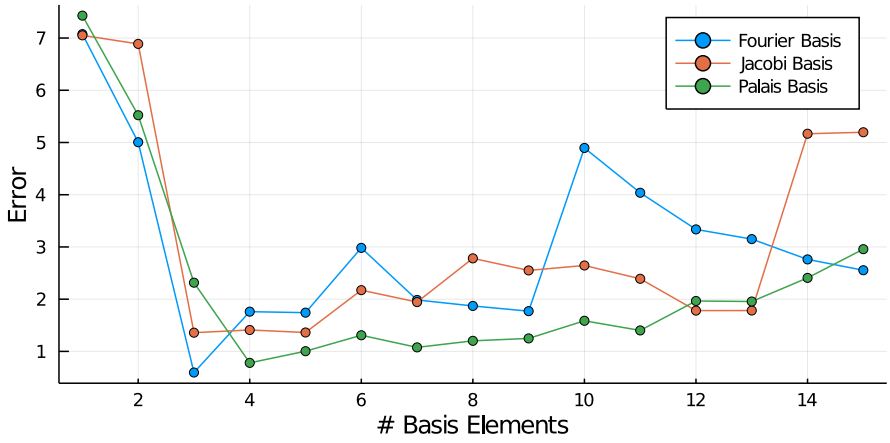


Figure 4.5: The final error of the reparametrization algorithm when used to compare two curves representing the same shape, versus the number of basis elements used in the gradient projection step.

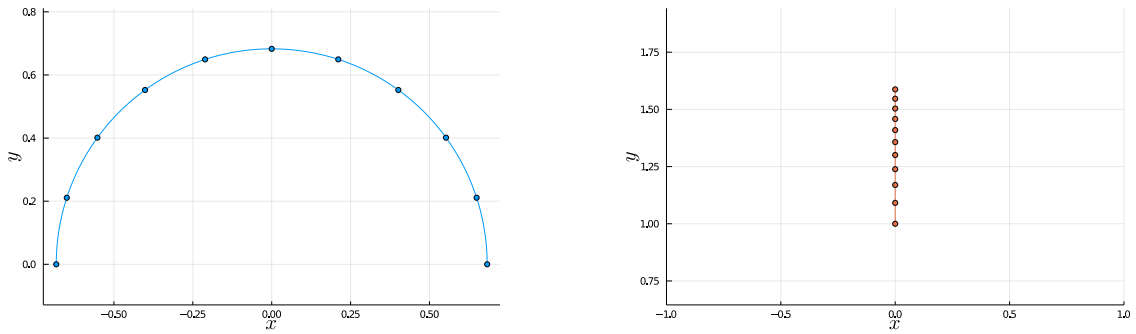
Different Shapes Comparison

In this section we test the performance of the algorithm when comparing curves representing different shapes: A half-circle and a straight line. The examples are constructed such that we have a known analytical solution to the problem. The two curves are given by

$$c_1(x) = \frac{1}{\sqrt[3]{\pi}} \left[\cos(\pi x), \sin(\pi x)\right], \quad c_2(x) = \left[0, \sqrt[3]{3x+1}\right] \tag{4.20}$$

with Q-maps

$$q(x) = [\cos(\pi x), \sin(\pi x)], \quad r(x) = [0, 1].$$



(a) c_1 (b) c_2

Figure 4.6: The two curves defined in (4.20).

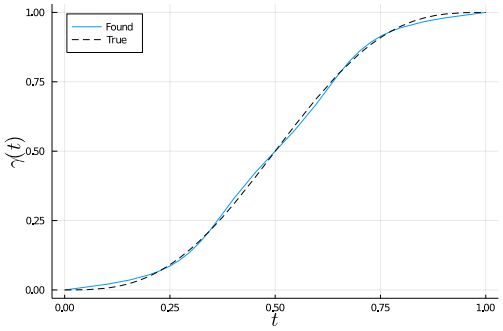
We refer to [60, Appendix A] regarding the derivation of the optimal solution, and will be content to state that the optimal solution is given by

$$\psi(x) = x - \frac{\sin(2\pi x)}{2\pi}. \tag{4.21}$$

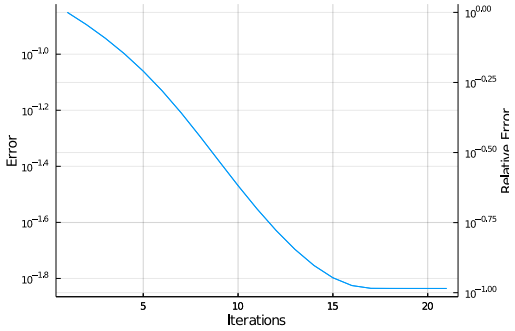
As a measure of the error in the algorithm, we will use the difference between the true value of the cost function $E(\psi)$ and the error found by the algorithm,

$$\Delta E^r(\gamma, \psi) = E(\gamma) - E(\psi).$$

For this example, the true shape distance is given by $E(\psi) = 2 - \sqrt{2} \approx 0.58579$. In figure 4.7 we present the results of the algorithm using 5 Fourier basis functions in the projection step. In this case, we see that the algorithm matches the true parametrization quite close, oscillating slightly around the optimal solution.



(a) Comparison of the true reparametrization ψ and the one found by the reparametrization algorithm.



(b) The error ΔE^r versus number of iterations. The initial error is 0.141, while the final error is 0.0146, a reduction to 10.3% of the original error.

Figure 4.7

4.3 Deep Reparametrization of Curves

This section presents a novel gradient-based approach to optimal reparametrization of curves by using ideas from deep learning. One big advantage of this approach is the ability to implement the algorithm using neural network frameworks such as `pytorch` and `tensorflow`, which allow us to easily produce fast and scalable code. Moreover, the diffeomorphism group is approximated by a set of functions defined on a linear space, such that optimization algorithms on euclidean spaces are made available.

The idea behind the algorithm is inspired by the previously discussed gradient descent approach. In each iteration of the gradient descent algorithm, we use an orthogonal projection to find a set of coefficients $\mathbf{c}$ and define a diffeomorphism

$$\gamma_n(x) = x + \sum_{i=1}^{N} c_i v_i(x).$$

where $\{v_i\}_{i=1}^N$ span some finite-dimensional approximation of the tangent space $T_{\mathrm{id}}\Gamma$. Once the coefficients have been determined, they will remain fixed throughout the whole algorithm. For the algorithm to proceed, we need to find a new weight vector, defining a new diffeomorphism, which we compose with the previous estimate. Assuming that the algorithm terminates after L iterations, the optimal parametrization is given as

$$\bar{\gamma} = \left(\mathrm{id} + \sum_{n=1}^{N} c_n^1 v_n\right) \circ \ldots \circ \left(\mathrm{id} + \sum_{n=1}^{N} c_n^L v_n\right).$$

This chain of compositions may become arbitrarily long, and the computational cost of evaluating the functions increases with each iteration.

If we instead fix the number L of functions that we want to compose, and then consider the weight vectors as coefficients to be optimized, we get a structure similar to the successful *residual neural networks* [27]. In this setting, the task of finding an optimal reparametrization corresponds to what is usually referred to as *training* the neural network.

We emphasize again that we do not attempt to generalize to unseen instances. The loss function that we optimize is given in terms of two fixed curves (or rather their q-maps), and the input of the network will be a fixed vector of points from the domain $I = [0, 1]$, which will be used to approximate the squared shape distance. Even though we use a fixed vector for the optimization process, the network defines a diffeomorphism that accepts as input an arbitrary number of points, from anywhere on the domain.

4.3.1 Problem Formulation

Given $q, r \in \mathcal{Q}$, we want to find a minimizer to the function

$$E : \Gamma \to \mathbb{R}, \quad E(F) = \| q - \sqrt{F'}\,(r \circ F)\,\|^2_{L^2(I,\mathbb{R}^2)}$$

such that $F \in \Gamma$. We will search for functions on the form

$$F = F_L \circ F_{L-1} \circ ... \circ F_1$$

for some $L \in \mathbb{N}$, where each *layer* F_l is given on the form

$$F_l(x) = x + f_l(x), \quad f_l(x) = \sum_{n=1}^{N} c_n^l v_n(x),$$

for a set of basis function $\{v_n\}_{n=1}^N \subset T_\gamma\Gamma$. We denote by $C = \{\mathbf{c}^l\}_{l=1}^L \subset \mathbb{R}^N$ the collection of weight vectors $\mathbf{c}^l$, and use the notation

$$F(x; C) = F_L(x; \mathbf{c}^L) \circ ... \circ F_1(x; \mathbf{c}^1)$$

to highlight the dependence of F and F_l on C and $\mathbf{c}^l$ respectively. Our goal is to find a set of vectors C, such that the function $F(\cdot, C) \in \Gamma$ minimizes the distance between q, r. To approximate the shape distance, we take a vector $\mathbf{x} = (x_k)_{k=1}^K \in I^K$ of linearly spaced points on I, and compute the mean squared error (MSE) between the q-maps sampled at these points. In other words we want to minimize the function

$$E(C; \mathbf{x}) = \frac{1}{K} \sum_{k=1}^{K} \left| q(x_k) - \sqrt{F'(x_k; C)} r(F(x_k; C)) \right|^2.$$

under the constraints that $F(0, C) = 0$, $F(1, C) = 1$, and $F'(\cdot : C) > 0$. We will be solving the problem using the BFGS algorithm, with a projection step in case the algorithm steps outside of the feasible set[6].

To uphold the derivative constraint, we require that the derivative F_l' of each layer F_l is positive. If we denote by

$$F^{(l)} = F_l \circ F_{l-1} \circ ... \circ F_1$$

the composition of the first l layers, then by the chain rule

$$\left(F^{(l)}\right)' = \left(F_l \circ F^{(l-1)}\right)' = \left(F_l' \circ F^{(l-1)}\right) \left(F^{(l-1)}\right)'.$$

Continuing by induction, the derivative of $F = F^{(L)}$ satisfies

$$F' = \prod_{l=1}^{L} F_l' \circ F^{(l-1)}$$

where $F^0 = \mathrm{id}$. Hence, the positivity of F' follows from the positivity of the derivative of each layer.

Basis Functions

Contrasting the gradient descent algorithm, the basis functions are in this case not required to be orthonormal. However, to ensure that the function F preserves the endpoints of I, we still want the basis functions to be vanishing at the boundaries, such that each layer F_l preserves the endpoints as well. In practice, we could easily have avoided this requirement by adding an affine transformation at the end of the network, mapping the output back onto I. However, since there are no simple extensions of this procedure to surfaces, we prefer boundary preserving basis functions.

For the numerical experiments in this thesis, we are reusing the Fourier and Palais basis functions from 4.2.2, which have proven to work well. In the future, it would be interesting to adopt the standard approach from deep learning of letting the network "learn its own basis functions". One possible way to achieve this, is by choosing

$$v_n(x) = \zeta(x)\sigma(w_n x + b_n)$$

where σ is a so-called *activation function* such as tanh or the sigmoid function, and ζ is some function satisfying $\zeta(0) = \zeta(1) = 0$, that ensures that the network is boundary preserving. w_n and b_n represent coefficients to be optimized. This will increase the dimensionality of the optimization problem, but will allow more flexibility in each layer.

[6] This is a makeshift solution which has performed well for the experiments in this thesis. A large scale application of the reparametrization algorithm will probably gain from changing to a constrained optimization routine, or other projected quasi-Newton algorithms with a better theoretical foundation.

4.3.2 Single-Layer Network

In this section, we will concern ourselves with a network consisting of a single layer, i.e. $L = 1$. For simplicity, we will suppress the sub- and superscripts l denoting the layer number, as we only have a single layer. Let $x_k \in I$, and denote by

$$\begin{aligned} z_k = F(x_k; \mathbf{c}) &= x_k + \sum_{n=1}^{N} c_n v_n(x_k) =: x_k + \mathbf{v}(x_k)^T \mathbf{c}, \\ y_k = F'(x_k; \mathbf{c}) &= 1 + \sum_{n=1}^{N} c_n v_n'(x_k) =: x_k + \mathbf{u}(x_k)^T \mathbf{c}. \end{aligned} \tag{4.22}$$

The cost function may then be expressed as

$$E(\mathbf{c}; \mathbf{x}, \mathbf{z}, \mathbf{y}) = \frac{1}{K} \sum_{k=1}^{K} \left| q(x_k) - \sqrt{y_k} r(z_k) \right|^2, \qquad \text{subject to (4.22).}$$

By a slight abuse of notation we will overload the use of F to allow vector inputs: Let $F : I^K \to I^K$ be defined by

$$\begin{aligned} \mathbf{z} = F(\mathbf{x}) &= \mathbf{x} + V(\mathbf{x})\mathbf{c} \\ \mathbf{y} = F'(\mathbf{x}) &= \mathbf{1} + U(\mathbf{x})\mathbf{c} \end{aligned}$$

where $V(\mathbf{x})_{kn} = v_n(x_k)$ and $U(\mathbf{x})_{kn} = v_n'(x_k)$. The above expression shows that the diffeomorphism layers are affine transformations *in the weights* $\mathbf{c}$ (but not in $\mathbf{x}$, as the basis functions are nonlinear).

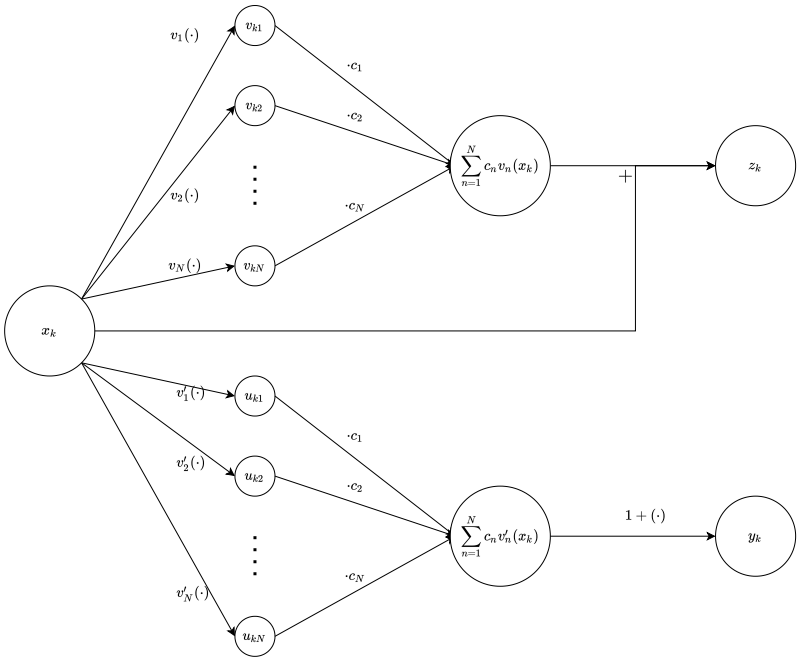


Figure 4.8: An illustration of how a single layer maps a point $x_k \in I$ to both $z_k = F(x_k)$ and $y_k = F'(x_k)$ simultaneously. The weights c_n are shared between the two paths.

Invertibility Constraints

The positive derivative constraint on the layers in the network, is to ensure that the network is invertible, as is required of a diffeomorphism. In [7] the authors enforces invertibility of residual networks by ensuring that the Lipschitz constants of f are smaller than one. This is achieved through a projection step at each iteration of training, as proposed in [24]. The projection step entails restricting the norm of the weight vector $\|\mathbf{c}\|_p$ for $p \in [1, \infty]$ (typically $p = 1, 2$ or ∞) to be lower than some threshold depending on the specific function.

Such a constraint is unnecessarily strict for the diffeomorphism group: While a Lipschitz constant smaller than one does indeed ensure that the derivatives of F are strictly positive, they also impose an upper bound of $F'(x) < 2$. No such upper bound exist for the diffeomorphism group. In figure 4.9, we have visualized the feasible sets under such constraints, as opposed to the actual feasible set, when using a Fourier sine basis with $N = 2$.

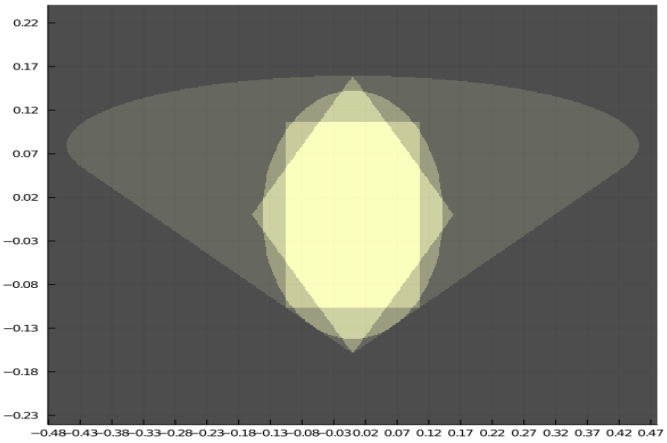


Figure 4.9: Feasible sets under various constraints ensuring that the derivative of F_l stays positive for $N = 2$, with $v_n(x) = \sin(\pi n x)$. The outermost shape corresponds to the actual sets of vectors $\mathbf{c} \in \mathbb{R}^2$ for which $F_l'(x) > 0$ for all $x \in I$. The square, circle and diamond corresponds to Lipschitz constraints on the norm of $\mathbf{c}$ for $p = 1$, $p = 2$ and $p = \infty$ respectively.

To avoid imposing unnecessarily strict constraints on the weight vectors, we make use of the linearity of f with respect to $\mathbf{c}$. Assume that for the current weight vector $\mathbf{c}$, there exist some $x \in I$ such that

$$y = F'(x; \mathbf{c}) = 1 + f'(x; \mathbf{c}) = 1 + \mathbf{u}(x)^T \mathbf{c} \leq 0$$

We want to scale down the vector $\mathbf{c}$ by a constant $k \in (0, 1)$ such that $k\mathbf{c}$ is a feasible weight vector, i.e.

$$F'(x; k\mathbf{c}) = 1 + f'(x, k\mathbf{c}) = 1 + kf'(x, \mathbf{c}) > 0.$$

This corresponds to the problem of finding a step size bound discussed in section 4.2.3, and we may reuse the expression (4.15) to find k. Let $(x_i)_{i=0}^K$ be a grid of points on I, and denote by $y_{\min} = \min_{i=0,\dots K} F'(x_i)$. Using the relation $-f'(x; \mathbf{c}) = 1 - y$, then

$$k = \frac{-1}{\min_{i=1,\dots,K} f'(x_k; \mathbf{c}) - \varepsilon} = \frac{1}{1 - (y_{\min} - \varepsilon)} \implies F'(x; k\mathbf{c}) \geq 0$$

where ε is an error term depending on the basis functions v_i and $\mathbf{c}$. It is possible to find a tight bound for ε, but as we want F' to be strictly positive, we will in practice add a small constant to this bound. This constant will also work as a numeric stabilizer that might be necessary when evaluating the gradient, as we will see in section 4.3.4.

This projection onto the feasible set should be performed after each update of the weight vectors. Of course it should only be done if $\mathbf{c}$ is outside of the feasible set (or rather our approximation of the feasible set), which correspond to the condition $y_{\min} \geq \varepsilon$. In general the projection may be written as

$$\pi(\mathbf{c}, y_{min}) = \frac{1}{1 - \min\{0, y_{\min} - \varepsilon\}} \mathbf{c}. \tag{4.23}$$

such that no scaling occurs if $\mathbf{c}$ is feasible.

4.3.3 Multilayer Network

Extending the concepts from the single-layer network to a multilayer network is simple. Since all layers in the multilayer network are assumed to have the same structure as the single layer

network, the notation and constraints provided in the previous sections easily lends themselves to the multilayer case. Let $x_k \in I$, and set $z_k^0 = x_k$ and $y_k^0 = 1$. Then we denote by

$$\begin{aligned} z_k^l &= F^{(l)}(x_k) = F_l(z_k^{l-1}) = z_k^{l-1} + \mathbf{v}(z_k^{l-1})^T \mathbf{c}^l \\ y_k^l &= \left(F^{(l)}\right)'(x_k) = F_l'(z_k^{l-1}) \left(F^{(l-1)}\right)'(x_k) = \left(1 + \mathbf{u}(z_k^{l-1})^T \mathbf{c}^l\right) y_k^{l-1} \end{aligned} \tag{4.24}$$

for $l = 1, ..., L$, or in vector notation:

$$\begin{aligned} \mathbf{z}^l &= \mathbf{z}^{l-1} + V(\mathbf{z}^{l-1})\mathbf{c}^l \\ \mathbf{y}^l &= \left(\mathbf{1} + U(\mathbf{z}^{l-1})\mathbf{c}^l\right) \odot \mathbf{y}^{l-1}. \end{aligned}$$

Here $\odot$ denotes the Hadamard (entrywise) product. The loss function will be on the same form as in the single-layer case, but with the output of the last layer as its inputs

$$E(C; \mathbf{x}, \mathbf{z}^L, \mathbf{y}^L) = \frac{1}{2} \sum_{k=1}^{K} w_k \left(q(x_k) - \sqrt{y_k^L} r(z_k^L) \right)^2, \quad \text{subject to (4.24)}$$

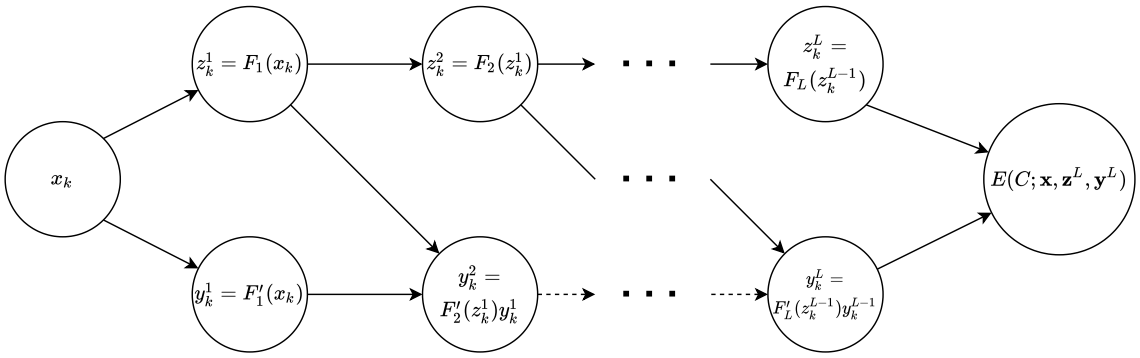


Figure 4.10: Illustration of the information flow through a multilayer reparametrization network. The upper path corresponds to the diffeomorphism itself, while the lower path corresponds to the derivative of the diffeomorphism.

To ensure the derivative of the network is positive, we apply to each layer the same procedure as we did in the single-layer: Choose a grid $\mathbf{x} \in I^K$, evaluate $\mathbf{y}^l = F_l(\mathbf{x}; \mathbf{c}^l)$, and scale the weight vector $\mathbf{c}^l$ according to (4.23). Pseudocode for the projected gradient method to be used while training a network with multiple layers is provided in Algorithm 2.

Algorithm 2 Optimization of weight vectors in a multilayer network for reparametrization of curves, with intermediate projection.

Require: $C_0 = \{\mathbf{c}_0^l\}_{l=1}^L \subset \mathbb{R}^N$: Initial weights.
Require: $\mathbf{x} = (x_k)_{k=1}^K \in I^K$: Vector of points on I.
$C \leftarrow C_0$
while $\mathbf{c}$ not converged **do**
  $\mathbf{z} \leftarrow F(\mathbf{x}; C)$
  $\mathbf{y} \leftarrow F'(\mathbf{x}; C)$
  $g \leftarrow \nabla_C E(C; \mathbf{x}, \mathbf{z}, \mathbf{y})$
  $\hat{C} \leftarrow \text{update}(C, g)$
  for $l = 1$ to L **do**
    $\mathbf{c}^l \leftarrow \pi(\hat{\mathbf{c}}^l, y_{\min}^l)$
  end for
end while
return C

4.3.4 Weight Optimization

Optimizing the weights in a neural network is usually done by an iterative line search method, where the weights are updated according to

$$\mathbf{c}_{t+1}^l = \mathbf{c}_t^l + \eta \mathbf{p}_t$$

where $\eta > 0$ is the step size, and $\mathbf{p}_t$ is the search direction, typically related to the gradient of the cost function. For example the gradient descent algorithm corresponds to $\mathbf{p}_t = -\nabla_{\mathbf{c}^l} E$, and in various *momentum* algorithms (see e.g. [56]) the search direction is chosen as $\mathbf{p}_t = -\nabla_{\mathbf{c}^l} E + \alpha \mathbf{v}_t$, for some constant $\alpha \in (0, 1)$, and $\mathbf{v}_t$ a vector depending on previous search directions $\mathbf{p}_\tau$ for $\tau < t$.

Due to the stochasticity that is usually present in deep learning, second-order optimization algorithms such as Newton's method or other Quasi-newton algorithms are not often used. However, since we are dealing with a deterministic problem, we may use the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm. The BFGS algorithm is a Quasi-Newton line-search method. The search direction $\mathbf{p}_t$ at stage t of the algorithm, is the solution to the problem

$$B_t \mathbf{p}_t = -\nabla_{\mathbf{c}^l} E(\mathbf{c}_t^l),$$

where B_t is an approximation of the Hessian of the objective function. The Hessian approximation is constructed such that it satisfies

$$B_{t+1}(\mathbf{c}_{t+1}^l - \mathbf{c}_t^l) = \nabla_{\mathbf{c}^l} E(\mathbf{c}_{t+1}^l) - \nabla_{\mathbf{c}^l} E(\mathbf{c}_t^l).$$

We refer to e.g. [43, 22] for further details on the Hessian approximation B_t, but note that it requires the computation of the gradient of E at each iteration.

To compute the gradient of the E with respect to the weights, we make use of a *reverse-mode automatic differentiation* algorithm called *back-propagation*. Back-propagation differs from forward-mode automatic differentiation (AD) which was discussed in section 4.2.5. Both algorithms break larger function down into smaller elementary operations for which its easy to compute the derivative, and then use the chain rule to compute the derivative of the function as a whole. However, the derivatives in forward-mode AD is computed simultaneously as the function is evaluated, while the back-propagation algorithm works in two steps: First it takes a *forward pass*, where the cost function is evaluated. During the forward pass, the AD software builds a computational graphs, keeping track of all operations applied to the input data. Then back-propagation computes gradients of the cost function with respect to each of the weight vectors, by applying the chain rule in the reverse direction of the computational graph.

For the reparametrization network, the gradients are computed according to the relations

$$\begin{aligned}
\nabla_{\mathbf{c}^l} E &= \sum_{k=1}^K \frac{\partial E}{\partial z_k^l} \nabla_{\mathbf{c}^l} z_k^l + \frac{\partial E}{\partial y_k^l} \nabla_{\mathbf{c}^l} y_k^l && = \sum_{k=1}^K \frac{\partial E}{\partial z_k^l} \mathbf{v}(x_k) + \frac{\partial E}{\partial y_k^l} \mathbf{u}(x_k) \\
\frac{\partial E}{\partial z_k^l} &= \frac{\partial E}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1} k}{\partial z_k^l} && = \frac{\partial E}{\partial z_k^{l+1}} F_{l+1}(z_k^l) \\
\frac{\partial E}{\partial y_k^l} &= \frac{\partial E}{\partial y_k^{l+1}} \frac{\partial y^{l+1}}{\partial y_k^l} + \frac{\partial E}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial y_k^l} && = \frac{\partial E}{\partial y_k^{l+1}} F_{l+1}''(z_k^l) + \frac{\partial E}{\partial z_k^{l+1}} F_{l+1}'(z_k^l) \\
\frac{\partial E}{\partial z_k^L} &= -\frac{2}{K} \left(q(x_k) - \sqrt{y_k^L} r(z_k^L) \right)^T r'(z_k^L) \\
\frac{\partial E}{\partial y_k^L} &= -\frac{1}{K} \left(q(x_k) - \sqrt{y_k^L} r(z_k^L) \right)^T \frac{r(z_k^L)}{\sqrt{y_k^L}}.
\end{aligned} \tag{4.25}$$

Hence by first computing the derivatives of the loss function with respect to the elements of the output vectors $\mathbf{y}^L$ and $\mathbf{z}^L$ of the network, then the above relations allow us to recursively compute the gradients with respect to each of the weight vectors $\mathbf{c}^l$.

We have implemented the network using the PyTorch-library[46] for Python, which takes care of the computation of gradients, and updates the weight vectors . The source code for our implementation is available on GitHub.[7]

[7] https://github.com/jorgenriseth/deepshape

From the last expression of (4.25), we see that the gradients of the cost function includes a division by $\sqrt{y_k^L}$. Since the y_k^L is meant to represent the derivative of a diffeomorphism at a point, we ideally want to allow it to be arbitrarily close to zero. This means that the gradients are not Lipschitz continuous, which may cause problems for the convergence of the algorithm. However, by letting the "buffer" ε in the projection step be sufficiently large, the derivative y_k^L should never be so small that it causes a problem in practice.

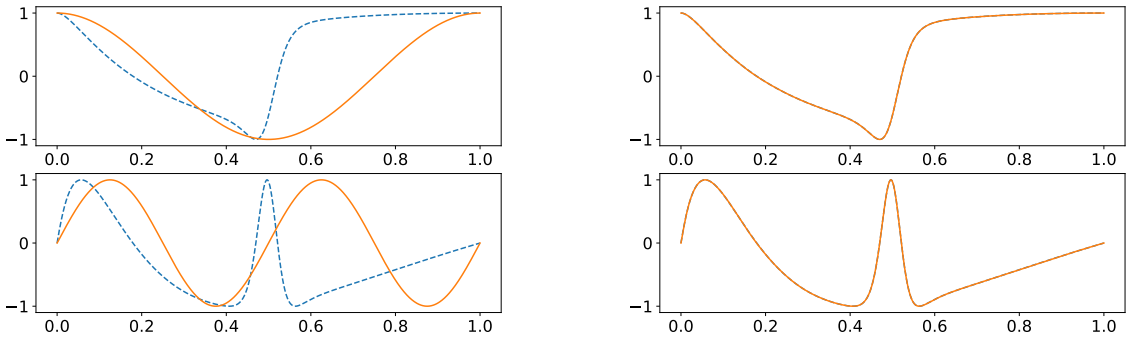
Imposing such a limit is not ideal. To ensure the robustness of the algorithm, further investigation should be put into possible alterations to either the optimization algorithm, or to the problem formulation, to avoid this degeneracy.

4.3.5 Numerical Results

In this section, we present numerical results using the deep reparametrization algorithm. We will be using the same examples as those presented in section 4.2.6, such that we may easily compare the two algorithms.

Same Shape Comparison

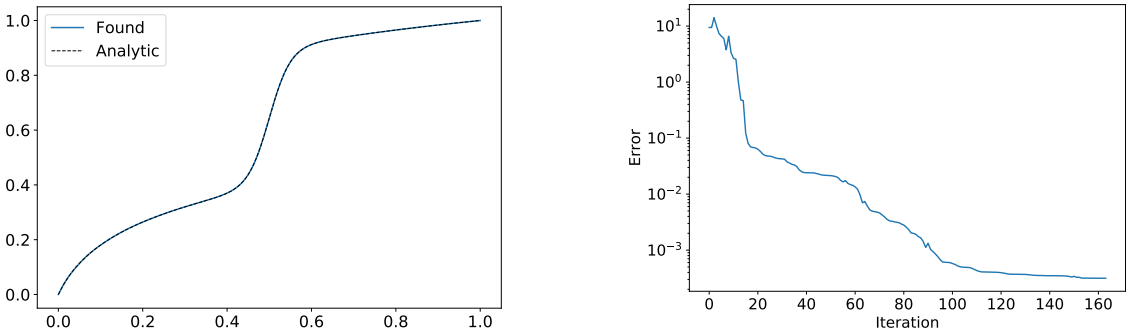
We start out by comparing the two curves c_2 and $c_1 = c_2 \circ \psi$ with c_2 and ψ given in (4.18) and (4.19) respectively. In figure 4.11 and 4.12 we present the results of the reparametrization using a network with 5 layers, each with 5 Fourier basis functions. The results show that the



(a) Before reparametrization.

(b) After reparametrization.

Figure 4.11: Comparison of the 1st *(top)* and 2nd *(bottom)* coordinate of the two curves defined by (4.18) and (4.19). The dashed blue line represents c_1, while the orange line represents c_2 before and after reparametrization.



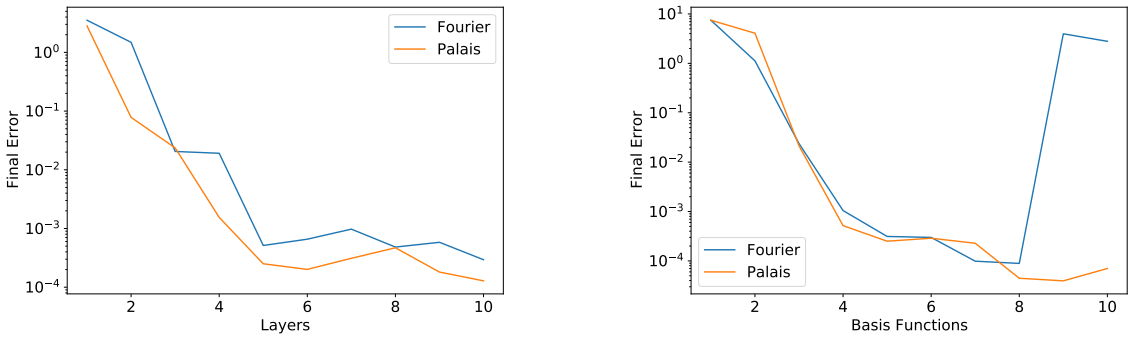
(a) Comparison of the true reparametrization ψ and the one found by the reparametrization algorithm.

(b) The cost function E^r versus number of iterations. The initial error is 9.44, while the final error is 0.000175, a reduction to 0.0019% of the original error.

Figure 4.12

deep reparametrization algorithm does a very good job of finding the optimal reparametrization. Both the resulting diffeomorphism, and coordinate functions are visually indistinguishable after reparametrization. In figure 4.13 we compare the performance of the algorithm for a varying number of layers and basis functions. Similarly as for the gradient descent algorithm, the error may increase when adding Fourier basis functions. However, when using Palais basis functions, the performance only seems to improve by adding more. A plausible explanation for this behaviour is that the constraints for the Fourier basis is *poorly scaled*. Since the derivatives of higher order basis elements have large derivatives, the size of the feasible set in some directions is smaller than in others. This makes it difficult to find a learning rate/step size which works well for the given problem. This is not the case for the Palais basis functions.

On the other hand, increasing the depth (i.e. the number of layers) of the network seems to improve the reparametrization for both bases. It is, however, still desirable to keep the number of layers low, as it lowers the dimensionality of the problem.



(a) Error vs. layers.

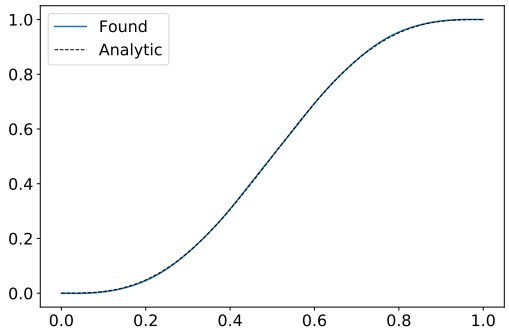
(b) Error vs. basis functions per layer.

Figure 4.13: The final error of the deep reparametrization algorithm when comparing two curves representing different shapes, versus *(left)* the number of layers in the network and a fixed number of 5 basis functions per layer, and *(right)* number of basis functions per layer with a fixed number of 5 layers.

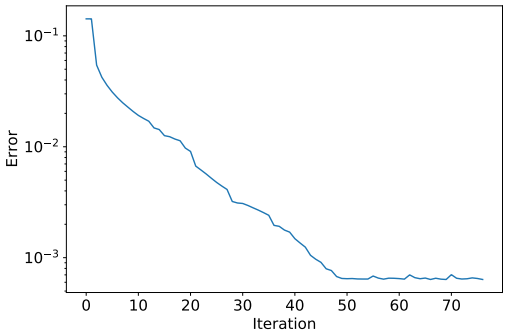
Different Shape Comparison

Applied to the half-circle and line defined in (4.20), the algorithm once again performs very well, and the diffeomorphism found by the algorithm is visually indistinguishable from the optimal solution.

When comparing the performance of the algorithm for a varying number of basis functions (figure 4.14b), there seems to be little to gain from using more than two basis functions. However, upon inspecting the optimal solution 4.21, this has a simple explanation: The optimal solution may be represented exactly by a single-layer network with only two basis functions. Hence by setting all other weights close to zero, the network finds a good approximation to the optimal reparametrization.

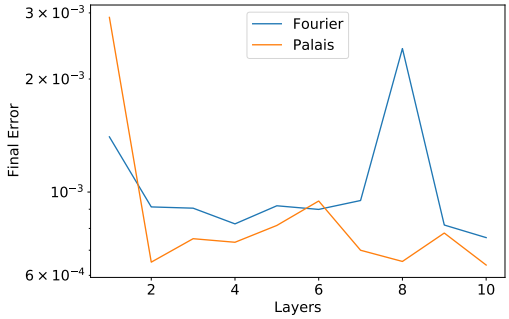


(a) Comparison of the true reparametrization ψ and the one found by the reparametrization algorithm.

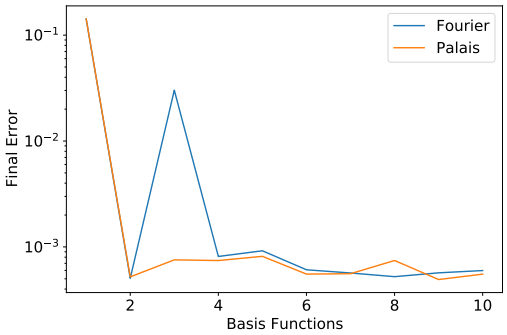


(b) The error ΔE versus number of iterations. The initial error is 0.142, while the final error is 6.3×10^{-4}, a reduction to 0.0044% of the original error.

Figure 4.14



(a) Error vs. layers.



(b) Error vs. Basis Functions

Figure 4.15: The final error of the deep reparametrization algorithm when comparing two curves representing the different shape, versus *(left)* the number of layers in the network and a fixed number of 5 basis functions per layer, and *(right)* number of basis functions per layer with a fixed number of 5 layers.

4.4 Summary

Throughout this chapter we have presented two gradient based algorithms for finding optimal reparametrizations of curves, and tested their performance on two test examples. To compare their performance we will be using two metrics:

$$\|\psi - \bar{\gamma}\|_\infty \approx \max_{i,j=1,\dots,256} |\psi(x_i, x_j) - \bar{\gamma}(x_i, x_j)|, \qquad \Delta E(\bar{\gamma}, \psi) = E(\bar{\gamma}) - E(\psi), \tag{4.26}$$

where ψ denotes the true optimal reparametrization, $\bar{\gamma}$ denotes the reparametrization found by the algorithms, and the points x_i are sampled on an equidistant grid on I. The results are taken from the examples in section 4.2.6 and 4.3.5. While the parameter settings (e.g. type and number of basis functions) may be adjusted for both the algorithms to improve the performance, the given settings performed reasonably well compared to alternative parameter settings.

It should be noted that while the cost function E for the two algorithms are approximations of the squared shape distance, the approximation differs in the two algorithms, so one should be careful when using ΔE to compare the algorithms. However, the difference between the final results is sufficiently large, that we allow ourselves to make a conclusion based on them. The results are summarized in table 4.2.

For the given test examples, the gradient descent algorithm typically needed between 1 to 10 seconds until termination, while the deep reparametrization algorithm terminated in less than 1 second, i.e. about one order of magnitude faster[8]. However, since the algorithms are implemented using two different languages, we are careful to put too much emphasis on the speed of

[8]Timings are taken with an Intel® Core™ i7-10875H CPU @ 2.30GHz × 16

Algorithm	Test Case	$\|\psi - \bar{\gamma}\|_\infty$	$\Delta E(\bar{\gamma}, \psi)$
Gradient Descent	Same Shape	2.4×10^{-2}	5.9×10^{-1}
Deep Reparametrization	Same Shape	6.0×10^{-4}	1.7×10^{-4}
Gradient Descent	Different Shapes	2.1×10^{-2}	1.0×10^{-1}
Deep Reparametrization	Different Shapes	3.7×10^{-3}	6.3×10^{-4}

Table 4.2: Comparison of the results achieved by the two algorithms for the two test cases described in previous sections.

the algorithms. Even though Julia should be a high-performance language in terms of speed, the `pytorch`-library is a highly optimized framework used both by researchers and industry, and is likely to be faster than our implementation of the gradient descent algorithm.

Based on the two test examples, the deep reparametrization algorithm far outperforms the gradient descent algorithm for optimal reparametrization of curves.

Chapter 5

Optimal Reparametrization of Parametric Surfaces

5.1 Shapes of Parametric Surfaces

We will consider parametric surfaces embedded in $\mathbb{R}^3$, represented by immersions defined on the unit square $M = [0,1]^2$,

$$\mathcal{F} = \{f : C^\infty(M, \mathbb{R}) \mid |f_x \times f_y| > 0\}. \tag{5.1}$$

Here f_x, f_y denote the partial derivatives of f. The reparametrization group on $\mathcal{F}$ is the group of orientation preserving diffeomorphisms, which consists of smooth invertible maps from M onto itself, such that the Jacobian determinant J_γ is positive,

$$\Gamma = \{\gamma \in C^\infty(M, M) \mid J_\gamma > 0,\ \gamma(0,y) = \gamma(x,0) = 0,\ \gamma(1,y) = \gamma(x,1) = 1\}$$

The tangent space $T_\gamma\Gamma$ consists of smooth *boundary preserving* vector fields on M,

$$T_\gamma\Gamma = \left\{\gamma \in C^\infty(M, \mathbb{R}^2) \;\middle|\; v|_{\partial M} \cdot \nu = 0,\ v \text{ smooth}\right\} \tag{5.2}$$

where ν denotes the outwards normal vector along the boundary of M (see e.g. [38]). Writing out the set condition explicitly gives for $v \in T_\gamma\Gamma$

$$v^1(0,y) = v^1(1,y) = v^2(x,0) = v^2(x,1) = 0.$$

Once again, the reparametrization group Γ acts on $\mathcal{F}$ from the right by composition;

$$(\cdot,\cdot) : \mathcal{F} \times \Gamma, \quad (c, \gamma) \mapsto c \circ \gamma.$$

Remark. For a function $\gamma \in C^\infty(M, M)$ to have a positive Jacobian determinant in a point, then the eigenvalues of the Jacobian matrix is either both positive, or both negative. However since we are working with boundary preserving functions on a manifold with corners, the eigenvalues of $D\gamma(\mathbf{x}^*)$ are both 1 (i.e. positive) in any cornerpoint $\mathbf{x}^*$. Since $D\gamma$ varies smoothly over M this means that the eigenvalues of the Jacobian matrix for any $\gamma \in \Gamma$ must be positive everywhere in M.

5.1.1 Shape Space

The construction of shape space for surfaces is completely analogue to the case for curves: We define an equivalence relation on $\mathcal{F}$ by

$$f_1 \sim f_2 \iff f_1 = f_2 \circ \gamma \text{ for some } \gamma \in \Gamma,$$

and define shape space as the quotient space

$$\mathcal{S} = \mathcal{F}/\Gamma,$$

where each equivalence class $[f]$ represents the shape of a surface. To define a shape space metric, we require a reparametrization invariant distance function $d_{\mathcal{F}}$ on the underlying space of parametric surfaces, and define the shape distance by

$$d_{\mathcal{S}}([f_1],[f_2]) = \inf_{\gamma\in\Gamma} d_{\mathcal{F}}(f_1, f_2 \circ \gamma). \tag{5.3}$$

5.1.2 The Q-transform for Surfaces

The Q-transform for surfaces serves to transform parametric surfaces into a representation that is reparametrization invariant under the L^2-metric. We will be using the same name for the Q-transform for curves and surfaces, as the distinction should be clear from the context.

Definition 5.1.1. *Define the* ***Q-transform*** *as the map*

$$Q : \mathcal{F} \to C^\infty(M, \mathbb{R}^3), \quad f(\cdot) \mapsto \sqrt{a_f(\cdot)} f(\cdot)$$

where

$$a_f(\mathbf{x}) = |f_x(\mathbf{x}) \times f_y(\mathbf{x})|$$

is the area scaling factor of the surface. We say that for any $f \in \mathcal{C}$, the surface $q = Q(f)$ is ***the q-map*** *or* ***q-representation of*** *f, and we define* ***the set of q-maps*** *as*

$$\mathcal{Q} := Q(\mathcal{C}) = \{q \in C^\infty(M, \mathbb{R}^3) \mid q = Q(c) \text{ for some } c \in \mathcal{C}\}.$$

The q-map of a surface after reparametrization satisfies the relation

$$Q(f \circ \gamma) = \sqrt{J_\gamma}(Q(f) \circ \gamma).$$

See e.g. [34] for details regarding the derivation of this property. Now, we define the underlying shape distance by

$$d_{\mathcal{F}}(f_1, f_2) = \|Q(f_1) - Q(f_2)\|_{L^2(M,\mathbb{R}^3)}$$

which satisfies the reparametrization invariance property by

$$\begin{aligned}
d_{\mathcal{F}}(f_1 \circ \gamma, f_2 \circ \gamma) &= \|Q(f_1 \circ \gamma) - Q(f_2 \circ \gamma)\|_{L^2(M,\mathbb{R}^3)} \\
&= \int_M \left| \sqrt{J_\gamma(\mathbf{x})} Q(f_1)(\gamma(\mathbf{x})) - \sqrt{J_\gamma(\mathbf{x})} Q(f_2)(\gamma(\mathbf{x})) \right|^2 d\mathbf{x} \\
&= \int_M J_\gamma(\mathbf{x}) |Q(f_1)(\mathbf{s}) - Q(f_2)(\mathbf{s})|^2 \left(J_\gamma(\mathbf{x})^{-1} \, d\mathbf{s} \right) \\
&= \int_M |Q(f_1)(\mathbf{s}) - Q(f_2)(\mathbf{s})|^2 \, d\mathbf{s} \\
&= d_{\mathcal{F}}(f_1, f_2)
\end{aligned}$$

We define a right action of the reparametrization group on the set of q-maps, by

$$\mathcal{Q} \times \Gamma \to \mathcal{Q}, \qquad (q, \gamma) \mapsto \sqrt{J_\gamma}(q \circ \gamma)$$

and the orbit of any $q \in \mathcal{Q}$ by

$$[q] = \{r \in \mathcal{Q} \mid r = J_\gamma(q \circ \gamma) \text{ for some } \gamma \in \Gamma\}.$$

Finally, for a given $r \in \mathcal{Q}$ we define the orbit map

$$\phi^r : \Gamma \to [r], \quad \phi^r(\gamma) = \sqrt{J_\gamma}(r \circ \gamma).$$

5.2 Gradient Descent on the Reparametrization Group

This section describes the Riemannian gradient descent for optimal reparametrization of surfaces. It is very similar to the gradient descent algorithm for curves, and many of the expressions involved are natural higher-dimensional extensions of the expressions found in section 4.2. The two main challenges when extending the algorithm to surfaces, lies in the nature of the tangent space $T_{\text{id}}\Gamma$ and in finding a step-size bound.

Assume that we are given two surfaces $f_1, f_2 \in \mathcal{F}$, and we want to compute their shape distance. Denote by $q = Q(f_1)$, $r_0 = Q(f_2)$ the q-maps of the curves, and define for any $r \in [r_0]$ the cost function

$$E^r : \Gamma \to \mathbb{R}, \quad E^r(\gamma) = \|q - \phi^r(\gamma)\|^2_{L^2(M,\mathbb{R}^3)}$$

We want to find a minimizer for the squared shape distance E^{r_0} by iteratively reparametrizing r_0 according to

$$r_{n+1} = \phi^{r_n}(\gamma_n) = \sqrt{J_{\gamma_n}}(r_n \circ \gamma_n), \quad n = 0, 1, 2, ...,$$

by a sequence of diffeomorphisms

$$\gamma_n = \mathrm{id} - \eta_n \nabla E^{r_n}(\mathrm{id}).$$

Here $\eta_n > 0$ is the step size in each iteration, and $\nabla E^{r_n}(\mathrm{id})$ is the Riemannian gradient of E^{r_n} at the identity.

5.2.1 Computing The Gradient of E^r

Let

$$F : C^\infty(M, \mathbb{R}^3) \to \mathbb{R}, \quad F(s) = \|q - s\|^2_{L^2(M,\mathbb{R}^3)}$$

with directional derivative

$$dF_s : C^\infty(M, \mathbb{R}^3) \to \mathbb{R}, \quad dF_s(v) = -2\langle q - s, v\rangle_{L^2(M,\mathbb{R}^3)}.$$

Then $E^r = F \circ \phi^r$ and $dE^r_\gamma(v) = dF_{\phi^r(\gamma)}\left(d\phi^r_\gamma(v)\right)$. The orbit map ϕ^r has three components $[\phi^{r^1}, \phi^{r^2}, \phi^{r^3}]^T$, so that we may differentiate each component separately. We denote by Df the Jacobian matrix of any function f. Let $v \in T_{\mathrm{id}}\Gamma$, and assume that $\alpha : (-\varepsilon, \varepsilon) \to \Gamma$ is some curve such that $\alpha(0) = \mathrm{id}$ and $\alpha'(0) = v$. Then

$$\begin{aligned}
d\phi^{r^j}_{\mathrm{id}}(v) &= \left.\frac{d}{dt}\right|_{t=0} \phi^{r^j}(\alpha(t)) = \left.\frac{d}{dt}\right|_{t=0} \left(\sqrt{J_{\alpha(t)}}(r^j \circ \alpha(t))\right) \\
&= \left(\frac{1}{2\sqrt{J_{\alpha(0)}}}\left(\left.\frac{d}{dt}\right|_{t=0} J_{\alpha(t)}\right)(r^j \circ \alpha(0)) + \sqrt{J_{\alpha(0)}}(\nabla r^j \circ \alpha(0))^T \alpha'(0)\right) \\
&= \left(\frac{1}{2\sqrt{J_{\mathrm{id}}}}\left(\left.\frac{d}{dt}\right|_{t=0} J_{\alpha(t)}\right)(r^j \circ \mathrm{id}) + \sqrt{J_{\mathrm{id}}}(\nabla r^j \circ \mathrm{id})^T v\right) \\
&= \frac{1}{2} div(v) r^j + (\nabla r^j)^T v,
\end{aligned}$$

where we used that

$$\begin{aligned}
\left.\frac{d}{dt}\right|_{t=0} J_{\alpha(t)} &= \left.\frac{d}{dt}\right|_{t=0} \det(D\alpha(t)) = d(\det)_{D\mathrm{id}}(\underbrace{[D\mathrm{id}]}_{=I} Dv) \\
&= d(\det)_I(Dv) = \mathrm{tr}(Dv) = div(v).
\end{aligned}$$

Putting the components together as a vector, gives the expression

$$d\phi^r_{\mathrm{id}} = \frac{1}{2} div(v) r + (Dr)v.$$

To help us derive an expression for the Riemannian gradient of E^r, we will require the adjoint of the divergence operator. Assuming $\varphi \in C^\infty(M, \mathbb{R})$ and $v \in T_{\mathrm{id}}(\Gamma)$, then integration by parts gives

$$\langle \varphi, div(v)\rangle_{L^2(M,\mathbb{R})} = \int_M div(v)\varphi \, dx = \underbrace{\int_{\partial M} \varphi v \cdot \nu \, ds}_{=0 \text{ since } v\cdot\nu = 0} - \int_M \nabla\varphi \cdot v \, dx = \langle -\nabla\varphi, v\rangle_{L^2(M,\mathbb{R}^2)}.$$

Inserting this into the differential of E^r gives

$$\begin{aligned}
dE_{\mathrm{id}}(v) &= -2\langle q - r, d\phi^r_{\mathrm{id}}(v)\rangle_{L^2(M,\mathbb{R}^3)} = -2\sum_{j=1}^{3}\langle q - r, d\phi^{r^j}_{\mathrm{id}}(v)\rangle_{L^2(M,\mathbb{R}^2)} \\
&= -2\left(\sum_{j=1}^{3}\left\langle \frac{1}{2}(q^j - r^j)r^j, div(v)\right\rangle_{L^2(M,\mathbb{R}^2)} + \langle (q^j - r^j)\nabla r^j, v\rangle_{L^2(M,\mathbb{R}^2)}\right) \\
&= -2\left(\sum_{j=1}^{3}\left\langle -\nabla\left(\frac{1}{2}(q^j - r^j)r^j\right), v\right\rangle_{L^2(M.\mathbb{R}^2)} + \langle (q^j - r^j)\nabla r^j, v\rangle_{L^2(M,\mathbb{R}^2)}\right) \\
&= \left\langle \sum_{j=1}^{3}(r^j\nabla q^j - q^j\nabla r^j), v\right\rangle_{L^2(M,\mathbb{R}^2)} \\
&= \left\langle (Dq)^T r - (Dr)^T q, v\right\rangle_{L^2(M,\mathbb{R}^2)}
\end{aligned}$$

Hence we define the functional gradient

$$\delta E^r_{\mathrm{id}} = (Dq)^T r - (Dr)^T q$$

Similarly as for curves, the functional gradient is generally not boundary preserving and needs to be projected onto the tangent space $T_{\mathrm{id}}\Gamma$.

5.2.2 Gradient Projection

The tangent space of the diffeomorphism group on M consists of smooth vector fields such that the vectors along the boundary ∂M are purely tangential. We will construct the basis component-wise, as tensor products of Fourier series.

We start by finding a basis $\mathcal{B}^1$ for the space

$$\{v \in C^\infty(M,\mathbb{R}) \mid v(0,y) = v(1,y) = 0\}$$

of functions on M that vanish at the boundaries in the x-direction. We will form a basis consisting of three families of functions:

$$\begin{aligned}
\xi_k(x,y) &= \sqrt{2}\sin(\pi k x) \\
\eta_{k,l}(x,y) &= 2\sin(\pi k x)\cos(2\pi l y) \\
\varphi_{k,l}(x,y) &= 2\sin(\pi k x)\sin(2\pi l y),
\end{aligned} \tag{5.4}$$

and define $\mathcal{B}^1 := \{\xi_k\}_{k=1}^\infty \cup \{\eta_{k,l}\}_{k,l=1}^\infty \cup \{\varphi_{k,l}\}_{k,l=1}^\infty$.

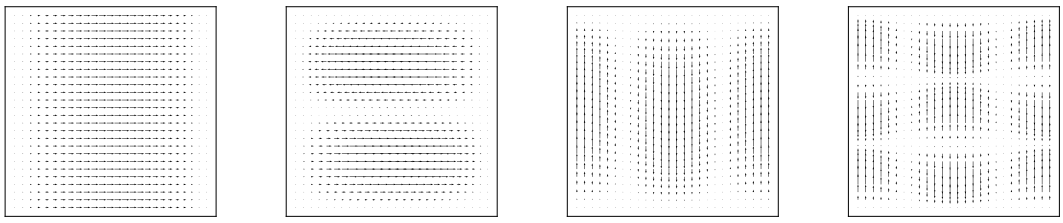
For any $v_n \in \mathcal{B}^1$, define the function $\tilde{v}_n(x,y) = v_n(y,x)$, such that $\tilde{v}_n$ is a smooth function that vanish on the boundaries in the y-direction. Denote by $\mathcal{B}^2 := \{\tilde{\xi}_k\}_{k=1}^\infty \cup \{\tilde{\eta}_k\}_{k,l=1}^\infty \cup \{\tilde{\varphi}_k\}_{k,l=1}^\infty$. A full basis for $T_\gamma\Gamma$ is then given by

$$\mathcal{B} = \left\{ \begin{bmatrix} v_n \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ \tilde{v}_n \end{bmatrix} \;\middle|\; v_n \in \mathcal{B}^1,\ \tilde{v}_n \in \mathcal{B}^2 \right\} \tag{5.5}$$

Finally, choosing some $N \in \mathbb{N}$ representing the maximal frequency of the basis functions, we form a basis for a subspace $V \subset T_\gamma\Gamma$ by truncating the bases

$$\begin{aligned}
\mathcal{B}^1_N &= \{\xi_k\}_{k=1}^N \cup \{\eta_{k,l}\}_{k,l=1}^N \cup \{\varphi_{k,l}\}_{k,l=1}^N \subset \mathcal{B}^1 \\
\mathcal{B}^2_N &= \{\tilde{\xi}_k\}_{k=1}^N \cup \{\tilde{\eta}_{k,l}\}_{k,l=1}^N \cup \{\tilde{\varphi}_{k,l}\}_{k,l=1}^N \subset \mathcal{B}^2
\end{aligned}$$

and define the basis $\mathcal{B}_N$ similarly as in (5.5). The total number of basis functions in $\mathcal{B}_N$ is $2(2N^2 + N)$, which means that there is a significant computational cost to increasing the number of basis functions used in the algorithm.



(a) ξ_1 (b) $\varphi_{1,1}$ (c) $\tilde{\eta}_{1,1}$ (d) $\tilde{\eta}_{1,3}$

Figure 5.1: Four examples of basis elements of $T_{\mathrm{id}}(\Gamma)$.

5.2.3 Step Size Bounds

After computing the Riemannian gradient $\nabla E^r(\mathrm{id})$, we define a new diffeomorphism by taking a step

$$\gamma = \mathrm{id} - \eta \nabla E^r(\mathrm{id}).$$

To ensure that γ is a diffeomorphism, we require that the step size η is sufficiently small such that the Jacobian determinant J_γ is positive. To find an upper bound for the allowed step size, fix a point $\mathbf{x} \in M$, and denote by $A(\mathbf{x}) = D(\nabla E^r(\mathrm{id}))(\mathbf{x})$ the Jacobian matrix of the vector field $\nabla E^r(\mathrm{id})$ at $\mathbf{x}$. The Jacobian determinant J_γ satisfies

$$J_\gamma(\mathbf{x}) = \det(D\gamma(\mathbf{x})) = \det(I - \eta A(\mathbf{x})) = 1 - \eta \,\mathrm{trace}(A(\mathbf{x})) + \eta^2 \det(A(\mathbf{x})) =: P_{\mathbf{x}}(\eta)$$

which is a quadratic polynomial in η, intersecting the y-axis at 1. Depending on $A(\mathbf{x})$, this polynomial has zero, one, or two roots. The upper bound for η corresponds to the smallest positive root. If no positive roots exists then $J_\gamma(\mathbf{x}) > 0$ for all $\eta > 0$. To determine the smallest positive root η, denote by $a = \det(A(\mathbf{x}))$, $b = -\mathrm{trace}(A(\mathbf{x}))$,[1] and consider the following cases:

1. $\mathbf{a < 0}$: If $a < 0$, then $P_{\mathbf{x}}(\eta)$ will always have one positive and one negative root. Since a is negative, the positive root is given by

$$\eta = \frac{-b - \sqrt{b^2 - 4a}}{2a}.$$

2. $\mathbf{a > 0}$: In this case we might have 0, 1 or two valid roots. If $b > 0$, then any real root will be negative. If $b < 0$ and $b^2 - 4a < 0 \implies b > -\sqrt{4a}$, $P_{\mathbf{x}}(\eta)$ has no real roots. Finally, for $b \geq -\sqrt{4a}$ there are two positive roots (or one with multiplicity 2 if the equality holds). We are interested in the smallest of the two roots,

$$\eta = \frac{-b - \sqrt{b^2 - 4a}}{2a}.$$

   which coincides with the formula in case 1.

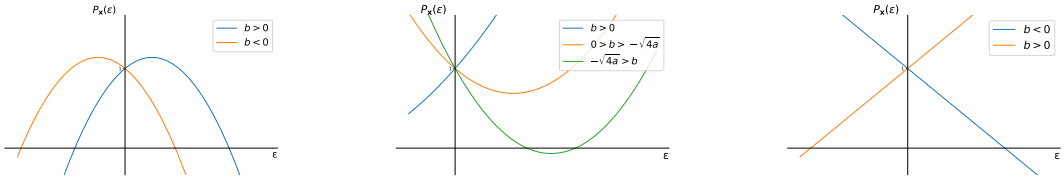
3. $\mathbf{a = 0}$: In the case $a = 0$, $P_{\mathbf{x}}(\eta)$ is reduced to a linear function which will have a positive root if and only if $b < 0$. In this case the root is given as

$$\eta = -\frac{1}{b}.$$

   Of course, a will rarely be exactly 0. However, for $a \approx 0$ the quadratic formula may become numerically unstable. Hence for $|a|$ sufficiently small, we will use the linear approximation to determine the bound.

The different cases are illustrated in figure 5.2.

[1] By choosing b to be negative, then the analysis presented here translates to a similar argument for the projection step required for deep reparametrization of surfaces



(a) Case 1: $\det(A) < 0$. (b) Case 2: $\det(A) > 0$ (c) Case 3: $\det(A) \approx 0$

Figure 5.2: Different cases to consider when determining the smallest positive root of $P_{\mathbf{x}}(\eta)$, determining the largest possible step size η allowed in the parametrization updates.

We sum up the different cases by the operator ϵ_δ defined as

$$\epsilon_\delta(a,b) = \begin{cases} -1/b & |a| < \delta \text{ and } b < 0 \\ \frac{-b-\sqrt{b^2-4a}}{2a} & (a < -\delta) \text{ or } \left(a > \delta \text{ and } b < -\sqrt{4a}\right) \\ \infty & \text{otherwise} \end{cases} \tag{5.6}$$

where δ is chosen as a small positive constant. ϵ_δ may be used to find an upper bound for the allowed step size, such that $J_\gamma(\mathbf{x})$ remains positive for a fixed $\mathbf{x} \in M$. To ensure that this holds everywhere, we therefore require

$$\eta < \bar{\eta} = \min_{\mathbf{x} \in K} \epsilon_\delta(\det(A(\mathbf{x})), \operatorname{trace}(A(\mathbf{x}))).$$

To find an approximation of the above upper bound, consider a set of points $X = \{\mathbf{x}_i\}_{i=1}^K$ from a tight grid on M, define

$$\tilde{\eta} = \min_{i=1,\dots,K} \epsilon_\delta(\det(A(\mathbf{x}_i)), \operatorname{trace}(A(\mathbf{x}_i))),$$

and then choose $\eta = \alpha\tilde{\eta}$ for some relative step size $\alpha \in (0,1)$. It is important to note that $\tilde{\eta}$ is just a rough approximation of $\bar{\eta}$, and by choosing α too close to 1, the step size may be too large. However, for the purposes of the numerical examples in this thesis, we will be satisfied to choose α heuristically.

5.2.4 Summary and Implementation

We sum up the algorithm by the pseudocode in Algorithm 3. To show that this algorithm actually corresponds to a Riemannian gradient descent algorithm, the argument from Theorem 1 for curves, translates almost one-to-one to the case for surfaces. The only difference lies in the definition of differential of the left multiplication by γ,

$$d\left(L_\gamma\right)_\varphi : T_\varphi\Gamma \to T_{\gamma\circ\varphi}\Gamma, \quad d\left(L_\gamma\right)_\varphi(v_\varphi) = (D\gamma \circ \varphi)v_\varphi$$

and its inverse

$$d\left(L_\gamma\right)_\varphi^{-1} : T_{\gamma\circ\varphi}\Gamma \to T_\varphi\Gamma, \quad d\left(L_\gamma\right)_\varphi(v_\varphi) = (D\gamma \circ \varphi)^{-1}v_\varphi.$$

Otherwise, the definitions of the retraction map and Riemannian metric translates directly to work for surfaces.

Implementation The implementation of the gradient descent algorithm for reparametrization of surfaces is also completely analogue to the algorithm for reparametrization of curves. We therefore refer to section 4.2.5 for details regarding the implementation. We do, however, note that the Gauss-Legendre quadrature is extended to the two-dimensional case, by

$$\int_0^1 \int_0^1 f(x,y)\,dx\,dy \approx \int_0^1 \frac{1}{2}\sum_{i=1}^n w_i f(x_i,y)\,dy \approx \frac{1}{4}\sum_{i=1}^n\sum_{j}^n w_i w_j f(x_i,x_j).$$

Algorithm 3 Gradient Descent for Optimal Reparametrization of Surfaces

Require: $q, r \in \mathcal{Q}$: q-maps of parametric surfaces.
Require: $\mathcal{B}_N$: Orthonormal basis for $T_{\mathrm{id}}\Gamma$.

$r_0 = r$
for $n = 0, 1, 2, ...$ **do** ▷ until convergence criterion is met
  $\delta E^{r_n} \leftarrow (Dr_n)^T q - (Dq)^T r_n$
  $\nabla E^{r_n}_{\mathrm{id}} \leftarrow \mathrm{project}(\delta E^{r_n}_{\mathrm{id}}, \mathcal{B}_N)$
  $\eta \leftarrow \mathrm{step\_select}(E^{r_n}, \nabla E^{r_n}_{\mathrm{id}})$
  $\gamma_n \leftarrow \mathrm{id} - \eta \nabla E^{r_n}_{\mathrm{id}}$
  $r_{n+1} \leftarrow \phi^{r_n}(\gamma_n)$
end for
return $\gamma_0 \circ \gamma_1 \circ ...$

5.2.5 Numerical Results

In this section, we present numerical results of Algorithm 3. Since we do not have available two different shapes for which we have a known optimal reparametrization, we will only consider surfaces representing the same shapes.

Before presenting the results, we note that the computational cost for the gradient descent algorithm for surfaces is significantly larger than for curves. For one, the number of basis functions increases by $2(2N^2 + N)$, where N is the maximal frequency of the basis functions. For $N = 1$ this corresponds to 6 basis functions and for $N = 2$ we get 20 basis functions. In addition, the computation of double integrals using Gauss-Legendre quadrature with K quadrature nodes per dimension requires $\mathcal{O}(K^2)$ evaluations. Already at $N = 2$, the computational cost is so large that it may pose a problem in large scale applications such as clustering tasks. Therefore, in the numerical experiments presented here, we have chosen N to be low. In the first experiment, the algorithm performed best with $N = 1$, and in the second we have used $N = 2$.

Rotation Diffeomorphism

Define the surface f_2 by

$$f_2(x, y) = [x, y, x^2 - y^2]^T, \tag{5.7}$$

and $f_1 = f_2 \circ \psi$, for

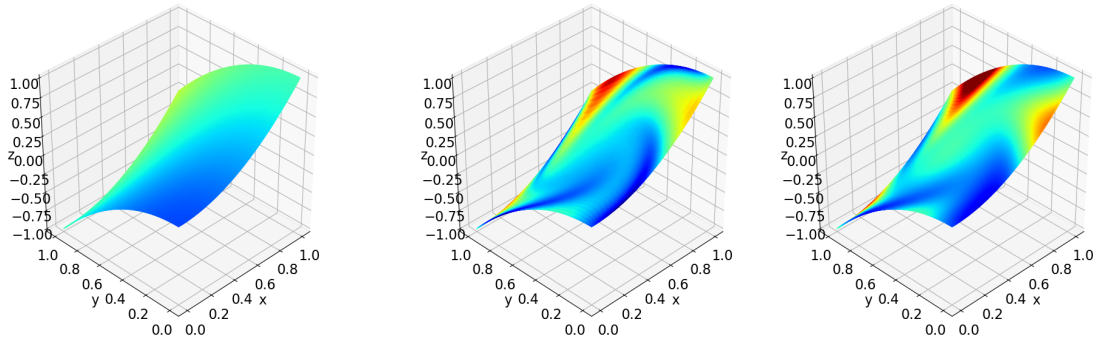
$$\psi(x, y) = \begin{bmatrix} (x - 0.5)\cos(\theta(x, y)) - (y - 0.5)\sin(\theta(x, y)) + 0.5 \\ (x - 0.5)\sin(\theta(x, y)) + (y - 0.5)\cos(\theta(x, y)) + 0.5 \end{bmatrix}, \tag{5.8}$$

$$\theta(x, y) = \frac{\pi}{2}\sin(\pi x)\sin(\pi y).$$

By applying the reparametrization algorithm to f_2, we want to find a diffeomorphism $\bar{\gamma}$ as close to ψ as possible. Since the correct shape distance for these two surfaces is zero, we use

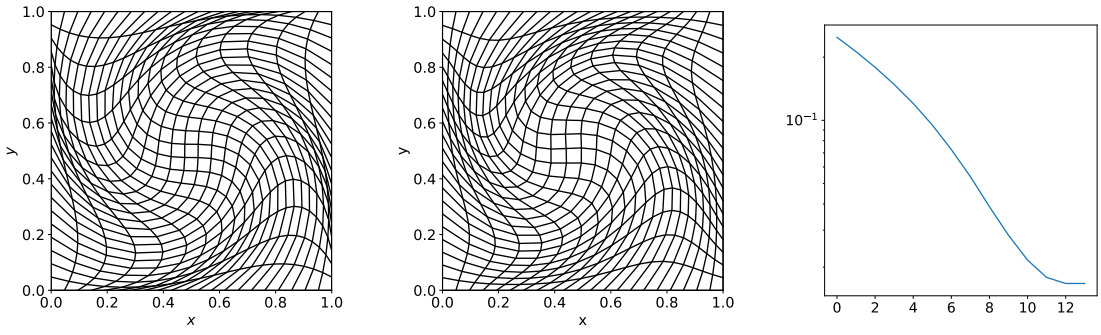
$$E^r(\bar{\gamma}) = \|q - \sqrt{J_{\bar{\gamma}}}(r \circ \bar{\gamma})\|^2_{L^2(M,\mathbb{R}^3)}$$

as a measure of the error. The results of the reparametrization are shown in figure 5.3 and 5.4, where we have used the colour of the surface to represent the local area scaling factor. The reparametrized surface has a rotation resembling that of g, but with somewhat erroneous area scaling factors. The algorithm terminated after 13 iterations, due to a relative error change below the threshold of 10^{-6}.



(a) f_2 (b) $f_1 = f_2 \circ \psi$ (c) $f_2 \circ \bar{\gamma}$

Figure 5.3: Two surfaces representing the same shape, and the reparametrized surface. *(left)* The surface f_2 (5.7). *(middle)* The surface f_2 reparametrized by the diffeomorphism ψ(5.8). *(right)* The surface f_2 reparametrized by $\bar{\gamma}$ found by the deep reparametrization algorithm.



(a) ψ (b) $\bar{\gamma}$ (c) Error vs. Iteration

Figure 5.4: Results from the reparametrization algorithm applied to $f_1 = f_2 \circ \psi$ and f_2, defined in equation (5.7). *(left)* Correct reparametrization ψ defined in (5.8). *(middle)* Reparametrization $\bar{\gamma}$ found by the algorithm. *(right)* Convergence plot. Initial error is 0.248 and final error is 0.0167, which is a reduction to 6.7% of original error.

Wrapped Cylinder

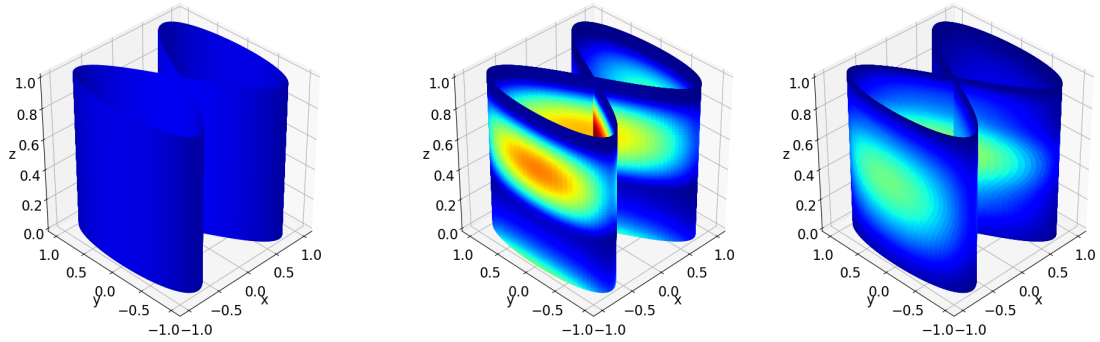
As a stress test for the algorithm, we compare another set of surfaces representing the same shape, but with a surface and reparametrization that is less "well-behaved" than previously. Define the surface f_2 by

$$f_2(x, y) = [\sin(2\pi), \sin(4\pi x), y]^T, \tag{5.9}$$

and reparametrize it to get a new surface $f_1 = f_2 \circ \psi$, where the reparametrization ψ is given by

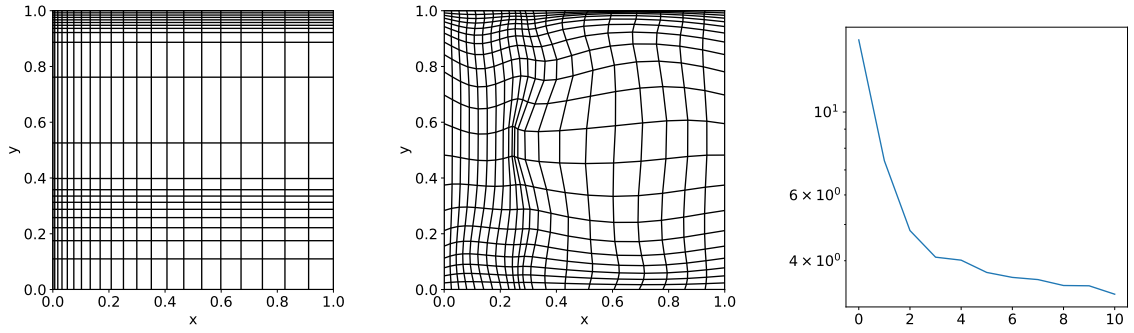
$$\psi(x, y) = \begin{bmatrix} 0.9x^2 + 0.1x \\ \frac{\log(20y+1)}{2\log(21)} + \frac{1+\tanh(20(y-0.5))}{4\tanh(10)} \end{bmatrix} \tag{5.10}$$

The surfaces and the result of the reparametrization are presented in figures 5.5 and 5.6. In this case we see the limitations of the gradient descent algorithm: While there are some improvements, there is still a significant difference between the reparametrized surface and the target.



(a) f_2 (b) $f_1 = f_2 \circ \psi$ (c) $f_2 \circ \bar{\gamma}$

Figure 5.5: Two surfaces representing the same shape, and the reparametrized surface. *(left)* The surface f_2 (5.9). *(middle)* The surface f_2 reparametrized by the diffeomorphism ψ(5.10). *(right)* The surface f_2 reparametrized by $\bar{\gamma}$ found by the deep reparametrization algorithm.



(a) ψ (b) $\bar{\gamma}$ (c) Error vs. Iteration

Figure 5.6: Results from the reparametrization algorithm applied to $f_1 = f_2 \circ \psi$ and f_2, defined in equation (5.9). *(left)* Correct reparametrization ψ defined in (5.10). *(middle)* Reparametrization $\bar{\gamma}$ found by the algorithm. *(right)* Convergence plot. Initial error is 15.6 and final error is 3.3, which is a reduction to 21% of original error.

5.3 Deep Reparametrization of Surfaces

This section extends the deep reparametrization framework to surfaces. Similarly as for gradient descent, the main difference from the algorithm for curves lies in the nature of the tangent space $T_\gamma\Gamma$ and the projection step.

5.3.1 Problem Formulation

Given $q, r \in \mathcal{Q}$, we want to find a minimizer to the function

$$E : \Gamma \to \mathbb{R}, \quad E(F) = \|q - \sqrt{J_F}\,(r \circ F)\,\|^2_{L^2(M,\mathbb{R}^3)}$$

such that $F \in \Gamma$. We will search for solutions on the form

$$F = F_L \circ F_{L-1} \circ ... \circ F_1$$

for some $L \in \mathbb{N}$, where each layer F_l is given on the form

$$F_l(\mathbf{x}) = \mathbf{x} + f_l(\mathbf{x}), \quad f_l(\mathbf{x}) = \sum_{n=1}^{N} c_n^l v_n(\mathbf{x}),$$

for a set of basis function $\{v_n\}_{n=1}^N \subset T_\gamma\Gamma$. We denote by $C = \{\mathbf{c}^l\}_{l=1}^L \subset \mathbb{R}^N$ the collection of weight vectors $\mathbf{c}^l$, and use the notation

$$F(\mathbf{x}; C) = F_L(\mathbf{x}; \mathbf{c}^L) \circ ... \circ F_1(\mathbf{x}; \mathbf{c}^1)$$

to highlight the dependence of F and F_l on C and $\mathbf{c}^l$ respectively. Our goal is to find a set of vectors C, such that the function $F(\cdot, C) \in \Gamma$ minimizes the shape distance between q, r. To approximate the shape distance, we sample the Q-maps q and $\phi^r(F)$ on a set of points $X = \{\mathbf{x}\}_{k=1}^K \subset M$, and compute the mean squared error (MSE) between the points. In other words we want to minimize the function

$$E(C; X) = \frac{1}{K} \sum_{k=1}^{K} \left| q(\mathbf{x}_k) - \sqrt{J_F(\mathbf{x}_k; C)} r(F(\mathbf{x}_k; C) \right|^2 .$$

under the constraints that $J_F(\cdot : C) > 0$ and $F(\partial M; C) = \partial M$. To uphold the constraint on the Jacobian determinant J_F, we require that the Jacobian determinant J_{F_l} of each F_l is positive. Denote by

$$F^{(l)} = F_l \circ F_{l-1} \circ ... \circ F_1$$

the composition of the first l layers. Then by the chain rule,

$$\begin{aligned} J_{F^{(l)}} &= \det(DF^{(l)}) = \det((DF_l \circ F^{(l-1)})DF^{(l-1)}) \\ &= \det(DF_l \circ F^{(l-1)}) \det(DF^{(l-1)}) = \sqrt{J_{F_l} \circ F^{(l-1)}} \sqrt{J_{F^{(l-1)}}}. \end{aligned}$$

Continuing by induction, the Jacobian of $F = F^{(L)}$ satisfies

$$J_F = \prod_{l=1}^{L} J_{F_l} \circ F^{(l-1)}$$

where $F^0 = \mathrm{id}$.

Basis Functions

The basis functions are no longer required to be orthogonal, which offers more flexibility in choosing the basis functions than in the gradient descent algorithm. For the numerical experiments in this chapter, we have based the basis functions on the vector fields presented in section 5.2.2. However, to avoid that higher-order basis elements are too sensitive to changes in the weights, we choose basis functions from one of the three families

$$\xi_k(x, y) = \frac{\sin(\pi k x)}{k}, \quad \eta_{k,l}(x, y) = \frac{\sin(\pi k x)\cos(2\pi l y)}{kl}, \quad \varphi_{k,l}(x, y) = \frac{\sin(\pi k x)\sin(2\pi l y)}{kl}. \quad (5.11)$$

These are just scaled versions of the original vector fields, and chosen such that the derivatives of higher-order elements does not become too large. We refer to 5.2.2 for further details on how to construct a full basis from these functions.

5.3.2 Single-Layer Network

To begin with, we consider a network with only a single layer. Since we only have one layer, we suppress the sub- and superscripts l in the notation. Consider a point $\mathbf{x} \in M$. Denote by

$$\begin{aligned} \mathbf{z}_k = F(\mathbf{x}_k) &= \mathbf{x}_k + \sum_{n=1}^{N} c_n v_n(\mathbf{x}_k) =: \mathbf{x}_k + V(\mathbf{x}_k)^T \mathbf{c} \\ y_k = J_F(\mathbf{z}_k) &= \det(I + \sum_{n=1}^{N} c_n Dv_n(\mathbf{x}_k)) =: \det(I + \mathsf{D}(\mathbf{x}_k)\mathbf{c}) \end{aligned} \tag{5.12}$$

where $V(\mathbf{x})$ is a matrix defined by

$$V(\mathbf{x})_{in} = v_n^i(\mathbf{x}), \quad i = 1, 2 \quad n = 1, ..., N,$$

and

$$\mathsf{D}(\mathbf{x})_{i,j,n} = \frac{\partial v_n^i}{\partial x^j}(\mathbf{x}), \quad i = 1, 2, \quad j = 1, 2, \quad n = 1, ..., , M$$

is a three-dimensional array. We denote by

$$\mathsf{D}(\mathbf{x})\mathbf{c} = \sum_{n+1}^{N} \mathsf{D}(\mathbf{x})_{:,:,n} c_n = \sum_{n+1}^{N} Db_n(\mathbf{x}_k) c_n$$

the tensor-vector product[2] between the three dimensional array and the weight vector. Using this notation, we rewrite the cost function as

$$E(\mathbf{c}; X, Z, \mathbf{y}) = \frac{1}{K} \sum_{k=1}^{K} |q(\mathbf{x}_k) - \sqrt{y_k} r(\mathbf{z}_k)|^2 \quad \text{subject to (5.12).}$$

Note that the matrix V here differs from the one in section 4.3: Where we previously used matrices in the vectorized notation for applying the layers to a collection of point in our domain, it is here used in the setting of a single point $\mathbf{x} \in M \subset \mathbb{R}^2$. We avoid introducing the vectorized notation for each of the layers, as it becomes rather cumbersome, and will be content to introduce the notation $Z = \{F(\mathbf{x}_k)\}_{k=1}^K$ for the collection of points after reparametrization, and $\mathbf{y} = (J_F(x_k))_{k=1}^K$ as the vector of Jacobian determinants evaluated at the same collection of points. However, the implementation uses 4-dimensional arrays, and vectorized operations available in `pytorch` to increase performance.

Invertibility Constraints

To ensure that the network upholds the invertibility constraints, i.e. that the Jacobian determinant is positive everywhere, we will use a projection step between each parameter update. Assume that for the current estimate of the weight vector $\mathbf{c}$, there exists some $\mathbf{x} \in M$ such that $J_F(\mathbf{x}; \mathbf{c}) < 0$. We want to find a scaling constant $k \in (0, 1)$ such that $J_F(\mathbf{x}; k\mathbf{c}) > 0, \forall \mathbf{x} \in M$. Following the analysis in section 5.2.3, the Jacobian determinant of $F(\mathbf{x}; k\mathbf{c}) = \mathbf{x} + kf(\mathbf{x}; \mathbf{c})$ at $\mathbf{x}$ becomes a quadratic polynomial in k

$$P_{\mathbf{x}}(k) = 1 + k \operatorname{trace}(Df(\mathbf{x})) + k^2 \det(Df(\mathbf{x})).$$

To ensure that $J_F(\mathbf{x}; k\mathbf{c}) \geq 0 \; \forall \; \mathbf{x} \in M$, we want to find the smallest positive root of $P_{\mathbf{x}}(k)$ (if it exists) for all $\mathbf{x} \in M$, and then set k to the smallest of these roots. To find an approximation of this k, we start by adding a *buffer* ε to the polynomial:

$$P_{\mathbf{x}}^{\varepsilon}(k) = (1 - \varepsilon) + k \operatorname{trace}(A(\mathbf{x})) + k^2 \det(A(\mathbf{x})).$$

Altering the operator defined in (5.6) to take into account the buffer, we define

$$\epsilon_{\delta}^{\varepsilon}(a, b) = \begin{cases} -(1 - \varepsilon)/b & |a| < \delta \text{ and } b < 0 \\ \frac{-b - \sqrt{b^2 - 4a(1-\varepsilon)}}{2a} & (a < -\delta) \text{ or } \left(a > 0 \text{ and } b < -\sqrt{4a}\right) \\ \infty & \text{otherwise} \end{cases} \tag{5.13}$$

[2] Here we use the words *tensor* and *tensor-vector product* in the rather crude interpretation as a multi-dimensional array, often found used in Deep Learning (see e.g. [23])

for some small number δ, and then choose

$$k = \min_{i=1,\dots,K} \epsilon_\delta^\varepsilon \left(\det(A(\mathbf{x}_i)), \text{trace}A(\mathbf{x}_i)\right),$$

for $\{\mathbf{x}_k\}_{k=1}^K$ a collection of points sampled from a grid on M. Of course, if there is no $\mathbf{x}$ such that $J_F(\mathbf{x};\mathbf{c}) < 0$, we do not want to change the weight vector. We therefore define the projection operator by

$$\pi(\mathbf{c};\{\mathbf{x}_i\}_{i=1}^K) = \left(\min\left\{1, \min_{i=1,\dots,K} \epsilon_\delta^\varepsilon \left(\det(A(\mathbf{x}_i)), \text{trace}A(\mathbf{x}_i)\right)\right\}\right)\mathbf{c}. \tag{5.14}$$

5.3.3 Multi-Layer Network

The extension to a multi-layer network is straight forward. We denote by

$$\begin{aligned}\mathbf{z}_k^l &= F^l(\mathbf{z}_k^l) = \mathbf{z}_k^{l-1} + U(\mathbf{z}_k^l)\mathbf{c}^l \\ y_k^l &= \det\left(DF^l(\mathbf{z}_k^{l-1})\right) = \det\left(1 + \mathsf{D}(\mathbf{z}_k^{l-1})\mathbf{c}^l\right) y_k^{l-1}\end{aligned}$$

with $\mathbf{z}_k^0 = \mathbf{x}_k$ and $y_k^0 = 1$. Then the cost function for a network with L layers, is given by

$$E(C;X,Z,\mathbf{y}) = \frac{1}{K}\sum_{k=1}^K \left| q(\mathbf{x}_k) - \sqrt{y_k^L}\, r\left(\mathbf{z}_k^L\right)\right|^2.$$

The optimization procedure is the exact same as described in 4.3.4, using a BFGS optimizer with intermediate projection steps. Since `pytorch` takes care of the computation of gradients and updating the weight vectors, we may reuse most of the code for reparametrization of curves. The only major changes are in the implementation of the projection procedure for surfaces, and the new basis functions.

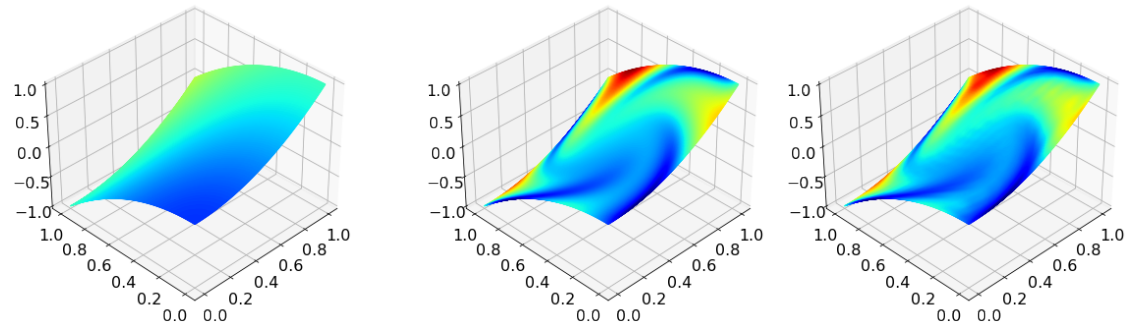
Algorithm 4 Optimization of weight vectors in a multilayer network for reparametrization of surfaces, with intermediate projection.

Require: $C_0 = \{\mathbf{c}_0^l\}_{l=1}^L \subset \mathbb{R}^N$: Initial weights.
Require: $X = \{\mathbf{x}_k\}_{k=1}^K \subset M$: Collection of points in M.

```
C ← C_0
while c not converged do
    Z ← F(X; C)
    y ← J_F(X; C)
    g ← ∇_C E(C; X, Z, y)
    Ĉ ← update(C, g)
    for l = 1 to L do
        c^l ← π(ĉ^l, X)
    end for
end while
return C
```

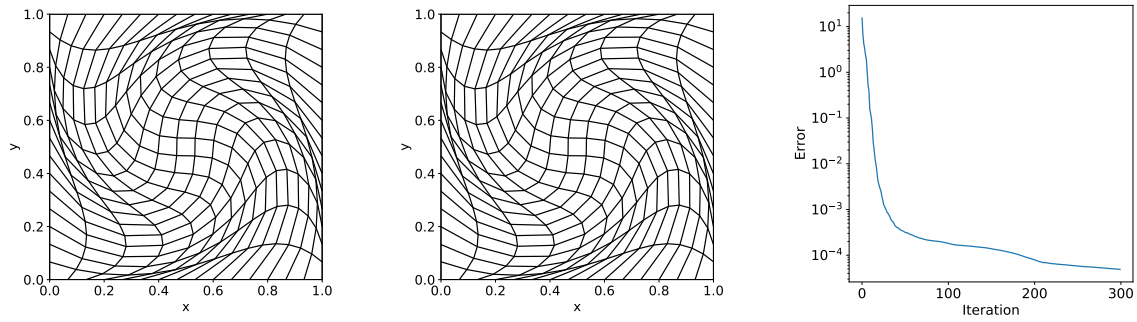
5.3.4 Numerical Results

This section presents test results from using the deep reparametrization algorithm for the test examples introduced in section 5.2. The results are based on a three-layer network with 930 basis functions per layer, which corresponds to a maximal frequency of $N = 15$. In both test cases, the deep reparametrization algorithm is able to approximate the correct solution very closely.



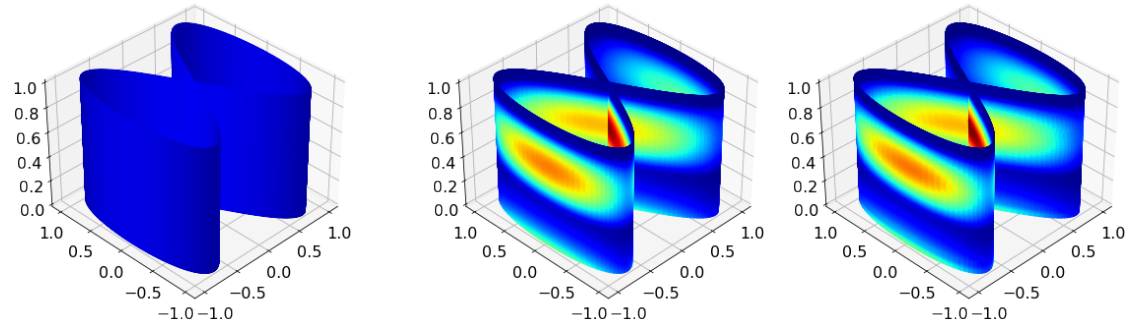
(a) f_2 (b) $f_1 = f_2 \circ \psi$ (c) $f_2 \circ \bar{\gamma}$

Figure 5.7: Two surfaces representing the same shape, and surface reparametrized by deep reparametrization. *(left)* The surface f_2 (5.7). *(middle)* The surface f_2 reparametrized by the diffeomorphism ψ(5.8). *(right)* The surface f_2 reparametrized by $\bar{\gamma}$ found by the deep reparametrization algorithm.



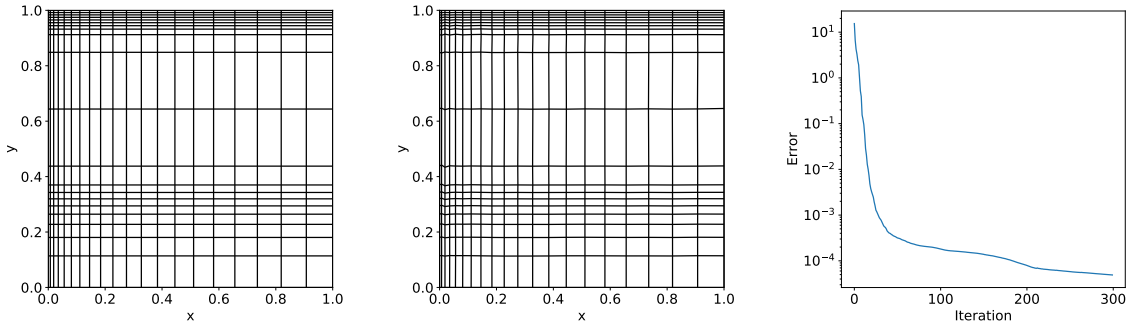
(a) ψ (b) $\bar{\gamma}$ (c) Error vs. Iteration

Figure 5.8: Results from the reparametrization algorithm applied to $f_1 = f_2 \circ \psi$ and f_2, defined in equation (5.7). *(left)* Correct reparametrization ψ defined in (5.8). *(middle)* Reparametrization $\bar{\gamma}$ found by the algorithm. *(right)* Convergence plot. Initial error is 0.26 and final error is 7.2×10^{-7}.



(a) f_2 (b) $f_1 = f_2 \circ \psi$ (c) $f_2 \circ \bar{\gamma}$

Figure 5.9: Two surfaces representing the same shape, and surface reparametrized by deep reparametrization. *(left)* The surface f_2 (5.9). *(middle)* The surface f_2 reparametrized by the diffeomorphism ψ(5.10). *(right)* The surface f_2 reparametrized by $\bar{\gamma}$ found by the deep reparametrization algorithm.



(a) ψ (b) $\bar{\gamma}$ (c) Error vs. Iteration

Figure 5.10: Results from the reparametrization algorithm applied to $f_1 = f_2 \circ \psi$ and f_2, defined in equation (5.9). *(left)* Correct reparametrization ψ defined in (5.10). *(middle)* Reparametrization $\bar{\gamma}$ found by the algorithm. *(right)* Convergence plot. Initial error is 15.4 and final error is 4.9×10^{-5}.

5.4 Summary

Throughout this chapter we have presented two gradient based algorithms for finding optimal reparametrizations of surfaces, and tested their performance on two test examples. To compare their performance we will be using two metrics:

$$\|\psi - \bar{\gamma}\|_\infty \approx \max_{i,j=1,\dots,256} |\psi(x_i, x_j) - \bar{\gamma}(x_i, x_j)|, \qquad \Delta E(\bar{\gamma}, \psi) = E(\bar{\gamma}) - E(\psi), \tag{5.15}$$

where ψ denotes the true optimal reparametrization, $\bar{\gamma}$ denotes the reparametrization found by the algorithms, and the points x_i are sampled on an equidistant grid on $[0, 1]$. The results are taken from the first examples in section 5.2.5. Since these examples concern surfaces representing the same shape, $E(\psi) = 0$ and $\Delta E(\bar{\gamma}, \psi) = E(\bar{\gamma})$.

The results are presented in table 5.1. In the first test case with the rotation diffeomorphism, the difference between the two algorithms is not too large in terms of the max-norm error of the diffeomorphisms. However, we see that this difference still has a large impact on the computed shape distances. For the wrapped cylinder test case on the other hand, the deep reparametrization algorithm significantly outperforms the gradient descent algorithm in both metrics.

Algorithm	Test Case	$\|\psi - \bar{\gamma}\|_\infty$	$\Delta E(\bar{\gamma}, \psi)$
Gradient Descent	Rotation	0.0842	1.7×10^{-1}
Deep Reparametrization	Rotation	0.0113	7.2×10^{-7}
Gradient Descent	Wrapped Cylinder	0.1772	3.3×10^{0}
Deep Reparametrization	Wrapped Cylinder	0.0054	4.9×10^{-5}

Table 5.1: Comparison of the results achieved by the two algorithms for reparametrization of surfaces, in the two test cases described in previous sections.

For the given test examples, the deep reparametrization algorithm was able to achieve the given results in between 10 and 20 seconds, whereas the gradient descent algorithm needed from 1 to 3 minutes. The significant increase in computational cost when increasing the number of basis functions, prevents the gradient descent algorithm from reaching its full potential within a reasonable amount of time. It might therefore be of interest to improve the implementation of the gradient descent algorithm, by for example vectorizing functions, before making a final conclusion regarding its performance relative to the deep reparametrization algorithm.

However, in the comparison between the two algorithms for reparametrization of curves (section 4.4) where the computational cost did not restrict the gradient descent algorithm, the deep reparametrization algorithm still performed significantly better than gradient descent. It therefore seems improbable that the gradient descent for surfaces will be able to achieve the same level of precision as the deep reparametrization algorithm, even with improvements to reduce computational cost.

Chapter 6

Conclusion

In this thesis, we have studied two gradient-based algorithms for solving the registration problem for parametric curves and surfaces. The first approach is a Riemannian gradient descent algorithm based on the framework proposed in [33], while the other is a novel approach in which optimal reparametrizations are found by training a residual neural network. Testing the two algorithms on a few test examples, the residual neural network performs significantly better that the gradient descent algorithm. This difference in performance is especially large when comparing parametric surfaces, where the gradient descent algorithm is limited by computational cost. Even though alterations to the gradient descent algorithm may reduce the computational cost, the deep reparametrization algorithm still beats gradient descent when both are allowed to run until convergence. Based on this, we recommend that future work focuses on improvements to the deep reparametrization algorithm.

Ideas for Future Work

The deep reparametrization algorithm seems very promising based on the numerical examples presented here, and should be tested against real data to evaluate its performance in practical problems. Moreover, several questions regarding the theoretical aspects of the method deserve further investigation.

Optimal Control The convergence properties of the algorithm should be further investigated: Due to the square root in the cost function, the gradient with respect to the weights in the network is not Lipschitz continuous, which may cause problems. In [26, 17], the authors propose an interpretation of residual neural networks as discrete optimal control problems. In [8] this interpretation is used to derive optimality conditions for the system, and to develop algorithms which ensure that the conditions are fulfilled. Adopting such an interpretation of the deep reparametrization algorithm may help in understanding the convergence properties of the algorithm, and provide insight into possible alterations to ensure convergence, if necessary.

Controllability The interpretation of the algorithm as a control system may also help to understand the representation abilities of the residual neural network. In [1], the authors study control systems on the group of diffeomorphisms. They show that any orientation preserving diffeomorphism may be represented as the composition of a finite number of exponentials of vector fields rescaled by smooth functions, provided the vector fields are chosen from a bracket generating family of vector fields. Interpreting the layers of the network as approximations of the exponential map, and choosing basis functions that satisfy these properties, may improve the network's ability to represent functions on the diffeomorphism group.

Bibliography

[1] A. A. Agrachev and M. Caponigro. "Controllability on the group of diffeomorphisms". en. In: *Annales de l'Institut Henri Poincaré C, Analyse non linéaire* 26.6 (Nov. 2009), pp. 2503–2509. ISSN: 0294-1449. DOI: 10.1016/j.anihpc.2009.07.003. URL: http://www.sciencedirect.com/science/article/pii/S0294144909000687 (visited on 01/24/2021).

[2] Larry Armijo. "Minimization of functions having Lipschitz continuous first partial derivatives." en. In: *Pacific Journal of Mathematics* 16.1 (1966), pp. 1–3. ISSN: 0030-8730. URL: https://projecteuclid.org/euclid.pjm/1102995080 (visited on 01/23/2021).

[3] Martin Bauer, Markus Eslitzbichler, and Markus Grasmair. "Landmark-Guided Elastic Shape Analysis of Human Character Motions". In: *arXiv:1502.07666 [cs]* (Feb. 2015). arXiv: 1502.07666. URL: http://arxiv.org/abs/1502.07666 (visited on 12/31/2020).

[4] Martin Bauer et al. "A numerical framework for elastic surface matching, comparison, and interpolation". In: *arXiv:2006.11652 [cs, math]* (June 2020). arXiv: 2006.11652. URL: http://arxiv.org/abs/2006.11652 (visited on 12/29/2020).

[5] Martin Bauer et al. "Constructing reparameterization invariant metrics on spaces of plane curves". en. In: *Differential Geometry and its Applications* 34 (June 2014), pp. 139–165. ISSN: 0926-2245. DOI: 10.1016/j.difgeo.2014.04.008. URL: http://www.sciencedirect.com/science/article/pii/S092622451400062X (visited on 12/27/2020).

[6] M. Faisal Beg et al. "Computing Large Deformation Metric Mappings via Geodesic Flows of Diffeomorphisms". en. In: *International Journal of Computer Vision* 61.2 (Feb. 2005), pp. 139–157. ISSN: 1573-1405. DOI: 10.1023/B:VISI.0000043755.93987.aa. URL: https://doi.org/10.1023/B:VISI.0000043755.93987.aa (visited on 12/28/2020).

[7] Jens Behrmann et al. "Invertible Residual Networks". en. In: *International Conference on Machine Learning*. PMLR, May 2019, pp. 573–582. URL: http://proceedings.mlr.press/v97/behrmann19a.html (visited on 01/07/2021).

[8] Martin Benning et al. "Deep learning as optimal control problems: models and numerical methods". In: *arXiv:1904.05657 [cs, math]* (Sept. 2019). arXiv: 1904.05657. URL: http://arxiv.org/abs/1904.05657 (visited on 01/24/2021).

[9] J. Bernal, G. Dogan, and C. R. Hagwood. "Fast Dynamic Programming for Elastic Registration of Curves". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. ISSN: 2160-7516. June 2016, pp. 1066–1073. DOI: 10.1109/CVPRW.2016.137.

[10] Paul J. Besl and Neil D. McKay. "Method for registration of 3-D shapes". In: *Sensor Fusion IV: Control Paradigms and Data Structures*. Vol. 1611. International Society for Optics and Photonics, Apr. 1992, pp. 586–606. DOI: 10.1117/12.57955. URL: https://www.spiedigitallibrary.org/conference-proceedings-of-spie/1611/0000/Method-for-registration-of-3-D-shapes/10.1117/12.57955.short (visited on 12/22/2020).

[11] Jeff Bezanson et al. "Julia: A Fresh Approach to Numerical Computing". In: *arXiv:1411.1607 [cs]* (July 2015). arXiv: 1411.1607. URL: http://arxiv.org/abs/1411.1607 (visited on 01/23/2021).

[12] Ch. Brechbühler, G. Gerig, and O. Kübler. "Parametrization of Closed Surfaces for 3-D Shape Description". en. In: *Computer Vision and Image Understanding* 61.2 (Mar. 1995), pp. 154–170. ISSN: 1077-3142. DOI: 10.1006/cviu.1995.1013. URL: http://www.sciencedirect.com/science/article/pii/S1077314285710132 (visited on 12/29/2020).

[13] M. Bruveris et al. "The Momentum Map Representation of Images". en. In: *Journal of Nonlinear Science* 21.1 (Feb. 2011), pp. 115–150. ISSN: 1432-1467. DOI: 10.1007/s00332-010-9079-5. URL: https://doi.org/10.1007/s00332-010-9079-5 (visited on 12/28/2020).

[14] Martins Bruveris et al. "Moser's theorem on manifolds with corners". In: *Proceedings of the American Mathematical Society* 146.11 (Aug. 2018). arXiv: 1604.07787, pp. 4889–4897. ISSN: 0002-9939, 1088-6826. DOI: 10.1090/proc/14130. URL: http://arxiv.org/abs/1604.07787 (visited on 01/15/2021).

[15] Elena Celledoni, Markus Eslitzbichler, and Alexander Schmeding. "Shape Analysis on Lie Groups with Applications in Computer Animation". In: *Journal of Geometric Mechanics* 8.3 (Sept. 2016). arXiv: 1506.00783, pp. 273–304. ISSN: 1941-4889. DOI: 10.3934/jgm.2016008. URL: http://arxiv.org/abs/1506.00783 (visited on 12/14/2020).

[16] Elena Celledoni et al. "Shape Analysis on Lie Groups and Homogeneous Spaces". en. In: *Geometric Science of Information*. Ed. by Frank Nielsen and Frédéric Barbaresco. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2017, pp. 49–56. ISBN: 9783319684451. DOI: 10.1007/978-3-319-68445-1_6.

[17] Bo Chang et al. "Reversible Architectures for Arbitrarily Deep Residual Neural Networks". en. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 32.1 (Apr. 2018). ISSN: 2374-3468. URL: https://ojs.aaai.org/index.php/AAAI/article/view/11668 (visited on 01/24/2021).

[18] Yang Chen and Gérard Medioni. "Object modelling by registration of multiple range images". en. In: *Image and Vision Computing*. Range Image Understanding 10.3 (Apr. 1992), pp. 145–155. ISSN: 0262-8856. DOI: 10.1016/0262-8856(92)90066-C. URL: http://www.sciencedirect.com/science/article/pii/026288569290066C (visited on 12/22/2020).

[19] Colin J. Cotter, Allan Clark, and Joaquim Peiró. "A Reparameterisation Based Approach to Geodesic Constrained Solvers for Curve Matching". en. In: *International Journal of Computer Vision* 99.1 (Aug. 2012), pp. 103–121. ISSN: 1573-1405. DOI: 10.1007/s11263-012-0520-0. URL: https://doi.org/10.1007/s11263-012-0520-0 (visited on 01/02/2021).

[20] G. Doğan, J. Bernal, and C. R. Hagwood. "A fast algorithm for elastic shape distances between closed planar curves". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. ISSN: 1063-6919. June 2015, pp. 4222–4230. DOI: 10.1109/CVPR.2015.7299050.

[21] Markus Eslitzbichler. "Modelling character motions on infinite-dimensional manifolds". en. In: *The Visual Computer* 31.9 (Sept. 2015), pp. 1179–1190. ISSN: 1432-2315. DOI: 10.1007/s00371-014-1001-y. URL: https://doi.org/10.1007/s00371-014-1001-y (visited on 01/03/2021).

[22] Roger Fletcher. *Practical methods of optimization*. John Wiley & Sons, 2013.

[23] Ian Goodfellow et al. *Deep learning*. Vol. 1. 2. MIT press Cambridge, 2016.

[24] Henry Gouk et al. "Regularisation of neural networks by enforcing Lipschitz continuity". en. In: *Machine Learning* (Dec. 2020). ISSN: 1573-0565. DOI: 10.1007/s10994-020-05929-w. URL: https://doi.org/10.1007/s10994-020-05929-w (visited on 01/07/2021).

[25] Ulf Grenander and Michael I. Miller. "Computational anatomy: an emerging discipline". en. In: *Quarterly of Applied Mathematics* 56.4 (1998), pp. 617–694. ISSN: 0033-569X, 1552-4485. DOI: 10.1090/qam/1668732. URL: https://www.ams.org/qam/1998-56-04/S0033-569X-1998-1668732-7/ (visited on 12/23/2020).

[26] Eldad Haber and Lars Ruthotto. "Stable architectures for deep neural networks". en. In: *Inverse Problems* 34.1 (Dec. 2017), p. 014004. ISSN: 0266-5611. DOI: 10.1088/1361-6420/aa9a90. URL: https://doi.org/10.1088/1361-6420/aa9a90 (visited on 01/24/2021).

[27] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *arXiv:1512.03385 [cs]* (Dec. 2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385 (visited on 01/20/2021).

[28] Ian H. Jermyn et al. "Elastic Shape Matching of Parameterized Surfaces Using Square Root Normal Fields". en. In: *Computer Vision – ECCV 2012*. Ed. by Andrew Fitzgibbon et al. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2012, pp. 804–817. ISBN: 9783642337154. DOI: 10.1007/978-3-642-33715-4_58.

[29] David G. Kendall. "Shape Manifolds, Procrustean Metrics, and Complex Projective Spaces". en. In: *Bulletin of the London Mathematical Society* 16.2 (1984), pp. 81–121. ISSN: 1469-2120. DOI: https://doi.org/10.1112/blms/16.2.81. URL: https://londmathsoc.onlinelibrary.wiley.com/doi/abs/10.1112/blms/16.2.81 (visited on 12/15/2020).

[30] David George Kendall et al. *Shape and shape theory*. Vol. 500. John Wiley & Sons, 2009.

[31] Eric Klassen and Peter W. Michor. "Closed surfaces with different shapes that are indistinguishable by the SRNF". In: *arXiv:1910.10804 [math]* (Oct. 2019). arXiv: 1910.10804. URL: http://arxiv.org/abs/1910.10804 (visited on 12/29/2020).

[32] Andreas Kriegl and Peter Michor. *The Convenient Setting of Global Analysis*. Vol. 53. Oct. 1996. DOI: 10.1090/surv/053.

[33] S. Kurtek et al. "A novel riemannian framework for shape analysis of 3D objects". In: *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. ISSN: 1063-6919. June 2010, pp. 1625–1632. DOI: 10.1109/CVPR.2010.5539778.

[34] S. Kurtek et al. "Elastic Geodesic Paths in Shape Space of Parameterized Surfaces". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34.9 (Sept. 2012), pp. 1717–1730. ISSN: 1939-3539. DOI: 10.1109/TPAMI.2011.233.

[35] S. Kurtek et al. "Parameterization-Invariant Shape Comparisons of Anatomical Surfaces". In: *IEEE Transactions on Medical Imaging* 30.3 (Mar. 2011), pp. 849–858. ISSN: 1558-254X. DOI: 10.1109/TMI.2010.2099130.

[36] Mario Lezcano-Casado and David Martínez-Rubio. "Cheap Orthogonal Constraints in Neural Networks: A Simple Parametrization of the Orthogonal and Unitary Group". In: *arXiv:1901.08428 [cs, stat]* (May 2019). arXiv: 1901.08428. URL: http://arxiv.org/abs/1901.08428 (visited on 01/11/2021).

[37] Sven Loncaric. "A survey of shape analysis techniques". en. In: *Pattern Recognition* 31.8 (Aug. 1998), pp. 983–1001. ISSN: 0031-3203. DOI: 10.1016/S0031-2023(97)00122-2. URL: http://www.sciencedirect.com/science/article/pii/S0031202397001222 (visited on 12/15/2020).

[38] Peter W Michor. *Manifolds of differentiable mappings*. Vol. 3. Birkhauser, 1980.

[39] Peter W. Michor and David Mumford. "An overview of the Riemannian metrics on spaces of curves using the Hamiltonian approach". en. In: *Applied and Computational Harmonic Analysis*. Special Issue on Mathematical Imaging 23.1 (July 2007), pp. 74–113. ISSN: 1063-5203. DOI: 10.1016/j.acha.2006.07.004. URL: http://www.sciencedirect.com/science/article/pii/S1063520307000243 (visited on 12/27/2020).

[40] Peter W. Michor and David Mumford. "Riemannian geometries on spaces of plane curves". In: *arXiv:math/0312384* (Feb. 2006). arXiv: math/0312384. URL: http://arxiv.org/abs/math/0312384 (visited on 12/27/2020).

[41] Peter W. Michor and David Mumford. "Vanishing geodesic distance on spaces of submanifolds and diffeomorphisms". In: *arXiv:math/0409303* (May 2005). arXiv: math/0409303. URL: http://arxiv.org/abs/math/0409303 (visited on 12/27/2020).

[42] David Mumford. "Pattern Theory: A Unifying Perspective". en. In: ed. by Anthony Joseph et al. Progress in Mathematics. Basel: Birkhäuser, 1994, pp. 187–224. ISBN: 9783034891103. DOI: 10.1007/978-3-0348-9110-3_6. URL: https://doi.org/10.1007/978-3-0348-9110-3_6 (visited on 12/23/2020).

[43] Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

[44] Hideki Omori. "On Banach-Lie groups acting on finite dimensional manifolds". EN. In: *Tohoku Mathematical Journal* 30.2 (1978), pp. 223–250. ISSN: 0040-8735, 2186-585X. DOI: 10.2748/tmj/1178230027. URL: https://projecteuclid.org/euclid.tmj/1178230027 (visited on 01/04/2021).

[45] Richard S. Palais. "Morse theory on Hilbert manifolds". en. In: *Topology* 2.4 (Jan. 1963), pp. 299–340. ISSN: 0040-9383. DOI: 10.1016/0040-9383(63)90013-2. URL: http://www.sciencedirect.com/science/article/pii/0040938363900132 (visited on 01/09/2021).

[46] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *arXiv:1912.01703 [cs, stat]* (Dec. 2019). arXiv: 1912.01703. URL: http://arxiv.org/abs/1912.01703 (visited on 01/26/2021).

[47] J. Revels, M. Lubin, and T. Papamarkou. "Forward-Mode Automatic Differentiation in Julia". In: *arXiv:1607.07892 [cs.MS]* (2016). URL: https://arxiv.org/abs/1607.07892.

[48] T. B. Sebastian, P. N. Klein, and B. B. Kimia. "On aligning curves". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 25.1 (Jan. 2003), pp. 116–125. ISSN: 1939-3539. DOI: 10.1109/TPAMI.2003.1159951.

[49] A. Srivastava et al. "Shape Analysis of Elastic Curves in Euclidean Spaces". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33.7 (July 2011), pp. 1415–1428. ISSN: 1939-3539. DOI: 10.1109/TPAMI.2010.184.

[50] Anuj Srivastava and Eric P Klassen. *Functional and shape data analysis*. Vol. 1. Springer, 2016.

[51] Anuj Srivastava, Pavan Turaga, and Sebastian Kurtek. "On advances in differential - geometric approaches for 2D and 3D shape analyses and activity recognition". In: *Image and Vision Computing* 30.6 (June 2012), pp. 398–416. ISSN: 0262-8856. DOI: 10.1016/j.imavis.2012.03.006. URL: http://www.sciencedirect.com/science/article/pii/S0262885612000492 (visited on 12/22/2020).

[52] Martin Styner et al. "Framework for the Statistical Shape Analysis of Brain Structures using SPHARM-PDM". In: *The insight journal* 1071 (2006), pp. 242–250. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3062073/ (visited on 12/29/2020).

[53] Z. Su et al. "Simplifying Transformations for a Family of Elastic Metrics on the Space of Surfaces". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. ISSN: 2160-7516. June 2020, pp. 3705–3714. DOI: 10.1109/CVPRW50498.2020.00432.

[54] Zhe Su et al. "Shape Analysis of Surfaces Using General Elastic Metrics". In: *Journal of Mathematical Imaging and Vision* 62.8 (Oct. 2020), pp. 1087–1106. ISSN: 1573-7683. DOI: 10.1007/s10851-020-00959-4. URL: https://doi.org/10.1007/s10851-020-00959-4 (visited on 12/29/2020).

[55] Ganesh Sundaramoorthi et al. "A New Geometric Metric in the Space of Curves, and Applications to Tracking Deforming Objects by Prediction and Filtering". In: *SIAM Journal on Imaging Sciences* 4.1 (Jan. 2011), pp. 109–145. DOI: 10.1137/090781139. URL: https://epubs.siam.org/doi/abs/10.1137/090781139 (visited on 01/02/2021).

[56] Ilya Sutskever et al. "On the importance of initialization and momentum in deep learning". en. In: *International Conference on Machine Learning*. PMLR, May 2013, pp. 1139–1147. URL: http://proceedings.mlr.press/v28/sutskever13.html (visited on 01/26/2021).

[57] Gabor Szego. *Orthogonal Polynomials*. en. Google-Books-ID: RemVAwAAQBAJ. American Mathematical Soc., Dec. 1939. ISBN: 9780821810231.

[58] D. W. Thompson. "On growth and form." English. In: *On growth and form.* (1942). URL: https://www.cabdirect.org/cabdirect/abstract/19431401837 (visited on 01/03/2021).

[59] Alex Townsend. *FastGaussQuadrature.jl*. original-date: 2014-08-31T14:35:16Z. Jan. 2021. URL: https://github.com/JuliaApproximation/FastGaussQuadrature.jl (visited on 01/23/2021).

[60] Esten Nicolai Wøien. "A Semi-Discretized Method for Optimal Reparametrization of Curves". In: (2019). URL: https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2624611 (visited on 12/16/2020).

[61] Laurent Younes et al. "A Metric on Shape Space with Explicit Geodesics". In: *arXiv: 0706.4299 [math]* (May 2008). arXiv: 0706.4299. URL: http://arxiv.org/abs/0706.4299 (visited on 12/28/2020).

[62] Dengsheng Zhang and Guojun Lu. "Review of shape representation and description techniques". en. In: *Pattern Recognition* 37.1 (Jan. 2004), pp. 1–19. ISSN: 0031-3203. DOI: 10.1016/j.patcog.2003.07.008. URL: http://www.sciencedirect.com/science/article/pii/S0031320303002759 (visited on 12/15/2020).

[63] Zhengyou Zhang. “Iterative point matching for registration of free-form curves and surfaces”. en. In: *International Journal of Computer Vision* 13.2 (Oct. 1994), pp. 119–152. ISSN: 1573-1405. DOI: 10.1007/BF01427149. URL: https://doi.org/10.1007/BF01427149 (visited on 12/22/2020).

NTNU

Norwegian University of
Science and Technology