Mémoire d'Habilitation à Diriger les Recherches

Méthodes et outils computationnels pour l'analyse de données géométriques

Computational Methods and Tools for Geometric Data Analysis

Benjamin Charlier

Présenté le **11 Décembre 2025**

Devant le jury composé de

Mohamed Daoudi — Professeur, IMT Nord Europe	Rapporteur
Xavier Pennec — Directeur de recherche, INRIA	Rapporteur
Frédéric Richard — Professeur, Aix-Marseille Université	Rapporteur
Lorenzo Rosasco — Associate Professor, Università di Genova, MIT	Examinateur
Joseph Salmon — Professeur, Université de Montpellier, INRIA	Examinateur
Carola–Bibiane Schönlieb — Full professor, University of Cambridge	Examinatrice
Alain Trouvé — Professeur, ENS Paris-Saclay	Examinateur

Charte d'intégrité scientifique

Je déclare avoir respecté, dans la conception et la rédaction de ce mémoire d'HDR, les valeurs et principes d'intégrité scientifique destinés à garantir le caractère honnête et scientifiquement rigoureux de tout travail de recherche, visés à l'article L.211–2 du Code de la recherche et énoncés par la Charte nationale de déontologie des métiers de la recherche et la Charte d'intégrité scientifique de l'Université de Montpellier. Je m'engage à les promouvoir dans le cadre de mes activités futures d'encadrement de recherche.

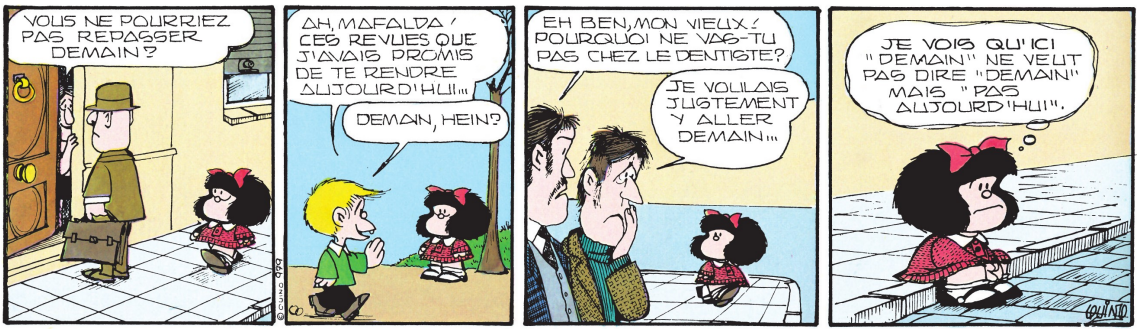
Benjamin Charlier

benjamin.charlier@inrae.fr

Unité de Mathématiques et Informatique Appliquées de Toulouse
INRAE Occitanie-Toulouse
24 Chemin de Borde Rouge
31320 Auzeville–Tolosane — France

Contents

Résumé



Écrire son Habilitation à diriger les recherches, par Quino dans [67]

Avant-Propos

Ce mémoire présente les développements des méthodes et outils computationnels en analyse de formes réalisés au cours de ces dix dernières années, en tant que Maître de Conférences à l'Université de Montpellier. À partir de 2013 et pendant douze ans, en parallèle de mes activités d'enseignement à la Faculté des Sciences, j'ai été membre de l'Institut Montpelliérain Alexander Grothendieck, au sein de l'équipe Statistique et Probabilités.

Au cours ces années, j'ai eu la chance de passer près d'un quart de mon temps — ce qui correspond officiellement à la moitié de mon temps alloué à la recherche — à travailler au sein d'autres laboratoires. De 2016 à 2019, j'ai été membre à temps partiel de l'équipe-projet Aramis (INRIA) à l'Institut du Cerveau et de la Moëlle Épinière (ICM), hébergé à l'Hôpital de la Pitié-Salpêtrière. Durant le premier semestre de 2023, j'ai passé quelques mois au Laboratoire Jacques-Louis Lions (LJLL). De plus, j'ai régulièrement fréquenté le Centre de Mathématiques et Leurs Applications (CMLA) à l'École Normale Supérieure de Cachan, qui est devenu par la suite le Centre Borelli à l'ENS Paris-Saclay.

Depuis peu, je suis en détachement dans l'unité Mathématiques et Informatique appliquées de Toulouse à l'Institut national de recherche pour l'agriculture, l'alimentation et l'environnement (INRAE).

Les différents laboratoires que j'ai fréquentés reflètent la diversité de mes centres d'intérêt scientifiques. Mes domaines de spécialité se situent à l'intersection de plusieurs champs disciplinaires : géométrie, statistique, informatique graphique, optimisation, imagerie, programmation, etc. Il est donc parfois difficile de résumer mes thématiques de recherche en quelques mots-clés. En définitive, mon intérêt se porte avant tout sur la résolution de problèmes concrets par la modélisation mathématique, laquelle mobilise souvent des outils issus de plusieurs domaines.

J'ai constaté, au fil de mes premières années de carrière, que les applications constituent une source inépuisable de développements méthodologiques originaux et féconds — pour peu que l'on reste curieux.

Si l'on inclut les années de doctorat, cela fait maintenant quinze ans que j'exerce un métier dont la finalité est de développer des outils et des méthodes pour l'analyse de données. À partir de l'ensemble des travaux menés durant cette période, il m'a fallu trouver un angle d'attaque pour construire un récit cohérent. J'ai choisi de centrer le propos sur mes contributions en analyse de forme, en structurant le document selon une unité de lieu, de temps et d'action. La première partie se déroule au CMLA, auprès d'A. Trouvé, autour des formes fonctionnelles ; la deuxième à l'ICM avec S. Durrleman, sur l'analyse de données longitudinales ; enfin, une dernière partie dans le cloud, avec J. Glaunès et J. Feydy, autour du développement de KeOps.

Je termine en mentionnant deux autres domaines de recherche. Le premier regroupe une série de travaux menés en collaboration avec B. Gris, portant sur la théorie des modules de déformation appliquée à l'analyse de formes structurées. Le second, développé plus récemment à Montpellier avec J. Salmon, concerne le traitement des données issues du crowdsourcing. Ces thématiques ne sont pas abordées dans les chapitres principaux, mais feront l'objet d'une discussion dans le chapitre de conclusion.

Introduction

Contexte

Comprendre la notion de forme pour en étudier la variabilité implique l'utilisation, à plusieurs niveaux, d'outils issus de la géométrie. En premier lieu, il y a les formes géométriques que l'on observe. Dans les applications issues de l'imagerie médicale ou biologique, ce sont souvent des nuages de points, des courbes ou des maillages dans le plan ou l'espace 3D. Définir la notion de forme de manière générale demeure une tâche délicate. On peut néanmoins la concevoir comme ce qui reste d'un objet lorsque l'on quotiente les effets de certaines transformations agissant sur lui[1]. Cette acception nous amène à un second niveau où la géométrie apparaît: l'ensemble des formes est décrit par un espace non-euclidien[2] telle qu'une variété riemannienne. Selon le groupe de tansformations considéré, les espaces de formes seront de dimension finie, comme la sphère des triangles de Kendall [42], ou de dimension infinie avec des géométries plus complexes, tels que les espace de courbes [55] ou des difféomorphismes [7].

Les espaces de Hilbert à noyau reproduisant (RKHS, pour *Reproducing Kernel Hilbert Space*) [8], sont des outils classsiques en analyse qui sont particulièrement pratiques pout définir des métriques sur les espaces de formes. Il s'agit d'espaces de dimension infinie qui se comportent presque comme des espaces de dimension finie, dans le sens où ils sont munis d'un produit scalaire et où l'évaluation en un point est une fonctionnelle linéaire continue. Cette propriété est très utile lorsqu'on traite des données discrètes que l'on peut interpoler pour approximer des objets continus. Les RKHS, qui constituent un concept central de cette thèse, sont également largement utilisés dans de nombreux domaines des mathématiques appliquées [10, 71, 82].

Les nouvelles modalités d'imagerie en biologie ou en médecine permettent désormais de mesurer la valeur d'une variable à des localisations précises de l'espace et/ou de multiplier

[1] D. G. Kendall motive sa définition par: "*Nous définissons ici 'la forme' de manière informelle comme 'ce qui reste lorsque les différences qui peuvent être attribuées aux translations, rotations et dilatations ont été retirées'.*" [41].

[2] D. Mumford a écrit ce que l'on pourrait traduire par "*La notion de forme est en quelque sorte l'expression ultime de la non-linéarité*" [59].

les acquisitions. Les données ayant motivé les travaux sur les formes fonctionnelles, décrites au Chapitre 1, sont composées d'un maillage sur lequel un signal a été observé. Au Chapitre 2, on s'intéresse au suivi de l'évolution temporelle d'un maillage représentant une sous-structure corticale. En conséquence, les modèles théoriques doivent être généralisés, ce qui conduit souvent à des formulations mathématiques plus complexes. Il est alors nécessaire de développer de nouveaux outils computationnels, tels que ceux présentés au Chapitre 3, pour appliquer ces modèles aux ensembles de données modernes.

Dans ce mémoire, je présente une série de contributions théoriques et pratiques dans le domaine de l'analyse de formes, en mettant l'accent sur la notion de formes fonctionnelles et sur l'analyse des données longitudinales, ainsi que leurs applications en imagerie médicale. Les outils computationnels développés sont suffisamment généraux et bas niveau pour intéresser d'autres communautés, notamment en statistique et en apprentissage automatique. Le travail est organisé en trois chapitres principaux, chacun traitant d'un aspect distinct de l'analyse de formes et de ses applications.

Plan du mémoire

Chapitre 1 — Analyse de formes pour données géométriques et fonctionnelles

Dans le premier chapitre, je présente des analyses théoriques et applications numériques des *formes fonctionnelles* (fshape) introduites par [13]. La définition des formes fonctionnelles a été motivée par la nécessité d'analyser la variabilité d'un ensemble de données acquises par tomographie par cohérence optique (OCT), dans le cadre de la recherche sur le glaucome. Les données consistent en des maillages de surface (les supports géométriques), tous distincts, auxquels sont associés un champ scalaire représentant l'épaisseur de la membrane observée. Un des objectifs de l'application était de développer un classificateur capable de détecter les symptômes de la maladie, tout en produisant des cartes d'épaisseur cohérentes entre les patients, fournissant ainsi aux praticiens un indicateur fiable de l'état de chaque patient.

Sur un plan théorique, l'analyse de forme fonctionnelle repose sur l'analyse de formes classique, où les objets géométriques sont étudiés modulo des déformations régulières et non-rigides agissant sur eux. Le principal enjeu réside dans l'adaptation des méthodes existantes — basées sur la théories des Large Deformation Metric Mapping (LDDMM) — pour traiter simultanément la composante géométrique et fonctionnelle.

Sur un plan pratique, le premier verrou est la taille importante des calculs. Les schémas numériques utilisés reposent largement sur les méthodes à noyaux, qui ont une complexité quadratique et sont donc difficiles à appliquer efficacement à des ensembles de données de centaines de milliers de points. Un autre problème crucial est l'instabilité numérique. Ce point a pu être résolu en étudiant les propriétés théoriques des modèles continues, donnant alors des indications pour corriger les modèles discrets.

Je commence par définir le fibré des formes fonctionnelles et introduire une classe de déformations — appelées métamorphoses — qui peuvent modifier à la fois la géométrie et le signal. Je définis ensuite une distance entre les formes fonctionnelles en utilisant un outil de la théorie géométrique de la mesure connu sous le nom de *varifolds*. Il fournit un cadre théorique attractif pour définir une famille de distances entre des objets géométriques muni de signaux (ou structures plus avancées, cela fonctionne toujours). Ces distances possèdent plusieurs atouts: elles sont régulières et n'ont pas besoin de correspondance point-à-point entre formes.

Je présente ensuite des résultats précis d'existence — démontrés dans les articles [Art3, Art6] —

pour le problème d'appariement (estimer la déformation entre un objet source et un objet cible) et le problème d'estimation d'atlas (estimer une forme moyenne et les déformations entre cette forme et chaque observation). Cette analyse a révélé pourquoi les expériences numériques avec des signaux L^2 étaient instables — principalement en raison d'effets de disparition de masse — et a conduit à étendre la cadre aux signaux de régularité plus élevée, dans des espaces de Sobolev H^s. Enfin, je décris le schéma discret utilisé dans les simulations numériques.

Ces travaux correspondent aux articles [Art3, Art6, Art4, Art5, Art11], [Proc2], ainsi qu'au logiciel FshapeTk [Soft1].

Chapitre 2 — Données longitudinales et évolution de formes

Dans le deuxième chapitre, je présente une série de résultats théoriques et pratiques développés pour modéliser l'évolution temporelles des formes géométriques. Ces travaux ont été réalisés durant mon séjour à l'ICM au sein de l'équipe ARAMIS, en collaboration avec S. Durrleman. Les chercheurs de l'ICM s'intéressent à la compréhension des causes, des dynamiques et des traitements potentiels des maladies neurodégénératives complexes telles que la maladie d'Alzheimer. En exploitant la base de données multimodales de l'Alzheimer's Disease Neuroimaging Initiative (ADNI) [39], qui contient des données longitudinales de patients, l'objectif est de comprendre la progression de la maladie d'Alzheimer et de contribuer à la personnalisation du parcours de soins des patients.

Les données longitudinales sont modélisées en représentant chaque observation par un point sur une variété riemannienne — par exemple, la variété des formes dans le cas des structures géométriques comme les maillages. L'évolution de ces structures au cours de la vie d'un patient est alors décrite comme une courbe sur cette variété abstraite, observée à des instants discrets.

Le transport parallèle est un outil central de la géométrie riemannienne qui permet de comparer des vecteurs tangents. C'est pourquoi la première section du chapitre présente un travail sur une méthode d'approximation — le schéma dit, en éventail (*fanning scheme* ou FS) — pour calculer le transport parallèle des vecteurs tangents [Art7]. J'étudis la vitesse de convergence de cette méthode et démontre que l'erreur diminue linéairement en le pas de discrétisation. Un avantage important du FS est qu'il ne nécessite pas le calcul du logarithme riemannien, qui est souvent coûteux en termes de calcul dans les applications réelles. Je fait également le lien avec le travail plus récent [37], qui démontre que des approches alternatives, connues sous le nom de méthodes à échelle (*Ladder*), peuvent offrir de meilleures performances pour l'approximation du transport parallèle.

Dans une seconde partie, je présente brièvement une application du transport parallèle, en montrant comment différents modèles de régression peuvent être utilisés pour prédire l'évolution d'une forme [Proc5]. Ces modèles ont été implémentés dans le logiciel Deformetrica [Soft2]. Enfin, je décris succinctement l'article [Art10], qui analyse la variabilité de la progression de la maladie d'Alzheimer à partir de données multimodales, incluant des images cérébrales, des évaluations clinico-neuropsychologiques et d'autres biomarqueurs.

Dans ce chapitre sont évoqués les travaux [Art7, Art8, Art10], [Proc1, Proc5, Proc7], et le logiciel FshapeTk [Soft2].

Chapitre 3 — Méthodes à noyau en pratique: formules génériques et ensembles de données de grande taille

Dans le dernier chapitre, je présente la bibliothèque KeOps, qui permet d'effectuer simplement des calculs efficaces pour des opérations à noyau à grande échelle. Le parangon de cette famille d'opérations est la convolution, interprétée ici comme un produit matrice-vecteur. Elles constituent le cœur de nombreuses méthodes abordées dans les chapitres précédents. KeOps permet de définir des formules mathématiques arbitraires pour ce type de calculs tout en garantissant de hautes performances et une bonne scalabilité.

Les deux principaux goulets d'étranglement des opérations à noyau sont la complexité computationnelle et l'usage mémoire. KeOps recourt à un calcul "brute force", conservant ainsi une complexité quadratique en temps, mais en la compensant par une exécution fortement parallélisée sur GPU. Cette approche lui permet d'atteindre l'état de l'art en terme de rapidité de calcul sur des problèmes de l'ordre du million de points. Mais surtout, KeOps conserve une empreinte mémoire linéaire, ce qui en fait un outil particulièrement adapté aux applications à grande échelle, où la mémoire constitue souvent un facteur limitant.

KeOps peut être utilisé avec des opérations à noyau définies à partir de formules mathématiques arbitraires, tout en gérant automatiquement leur exécution parallélisée. Pour cela, nous avons introduit le concept de *tenseur symbolique* [Art9], qui distingue trois types de dimensions : *batch*, externe et interne. Les opérations mathématiques peuvent être appliquées de manière différée (*lazy*) sur la dimension interne, c'est-à-dire que les calculs ne sont pas exécutés à la déclaration mais remis à plus tard, quand le résultat est requis. L'exécution effective n'est déclenchée que lors de la réduction de la dimension externe. C'est à ce moment que KeOps applique un schéma de réduction par tuilage optimisé.

Cette conception permet d'utiliser les tenseurs symboliques dans un code existant avec un minimum de code auxiliaire, facilitant ainsi son adoption. J'explique comment notre moteur interne de différentiation automatique est compatible avec la méthode `autograd` de PyTorch, ce qui permet une utilisation aisée dans des pipelines d'optimisation.

Je montre ensuite comment effectuer directement des opérations algébriques de haut niveau avec les tenseurs symboliques — tel que l'inversion de système linéaires basés sur des noyaux, des décompositions en espaces propres, etc. Enfin, je décris l'architecture interne de l'API, en détaillant sa conception modulaire et son extensibilité.

Ce travail a été publié dans [Art9], [Proc6, Proc8], ainsi que dans la documentation officielle disponible à l'adresse: www.kernel-operations.io [Soft3].

Foreword

Writing his Habilitation à diriger les recherches, by Quino in [67]

This document presents the development about computational methods and computational tools in shapes analysis carried out over the past decade during my tenure as a *Maître de Conférences* at the *Université de Montpellier*. From 2013 and for twelve years, alongside my teaching duties at the *Faculté des Sciences*, I was a member of the *Institut Montpelliérain Alexander Grothendieck* (IMAG), within the Statistics and Probability team.

Fortunately, I was able to spend nearly a quarter of this time — which officially corresponds to half of my allocated research time — visiting other laboratories. From 2016 to 2019, I was a part-time member of the Aramis project-team (INRIA) at the *Institut du Cerveau et de la Moëlle Épinière* (ICM), hosted at La Pitié-Salpêtrière Hospital. During the first half of 2023, I spent a few months at the *Laboratoire Jacques-Louis Lions* (LJLL). In the meantime, I regularly visited the *Centre de Mathématiques et Leurs Applications* (CMLA) at *École Normale Supérieure Cachan* (ENS Cachan), which later became the *Centre Borelli* at *ENS Paris-Saclay*.

I recently joined the *Mathématiques et Informatique appliquées de Toulouse* (MIAT) laboratory at the *Institut national de recherche pour l'agriculture, l'alimentation et l'environnement* (INRAE) as a full-time researcher (*Chargé de Recherches*).

This variety of structures reflects my interests, which lie at the intersection of several fields — such as geometry, statistics, computer graphics, optimization, imaging, coding, ... As a result, it is sometimes difficult to encapsulate my research output under a single keyword. It's not that my knowledge is particularly vast, but rather that it is drawn from a wide range of disciplines, in an effort to identify common patterns and address practical problems through modeling. I have come to realize that applied problems can serve as a powerful source of creativity for theoretical development. In any case, curiosity is always a good drive.

After 15 years of developing methods and tools at the frontier between applied mathematics and computer science, it became necessary to find a coherent narrative thread to present this body of work. I chose to center the theme around geometric data analysis through the concept of shape, which encompasses the majority of my contributions. This topic is a natural continuation —

though distinct — from my PhD thesis, which focused on high-dimensional statistics applied to quotiented data. In the spirit of classical theater principles, each chapter loosely follows the rules of the *Unités de lieu, de temps et d'action*. The first part takes place at CMLA with A. Trouvé and focuses on functional shapes; the second part at ICM with S. Durrleman explores longitudinal data analysis; and the third part unfolds in the *cloud* with J. Glaunès, J. Feydy, and the KeOps project.

Two other topics could have been included in this thesis. The first is a direct continuation of shape analysis, focusing on the modular deformation framework developed in collaboration with B. Gris at LJLL. Orthogonally, a more recent line of work with J. Salmon (IMAG) explores statistics and machine learning for the analysis of crowd-sourced data. While not covered in the main chapters, these topics will be discussed in the perspectives section at the conclusion of the thesis.

On a train. Somewhere between Toulouse and Montpellier, May 18th, 2025.

Introduction

Context

Performing shape analysis involves working with geometry at several levels. The first level concerns the observed geometric form itself, which is often represented in applications as point clouds, curves, or meshes in the plane or in 3D space. The notion of shape is delicate to define in full generality, but it can be informally understood as what remains after factoring out the effects of certain transformations acting on the object[3]. This leads to a second level where geometry is useful: the set of shapes is itself a space that can be modeled as non-Euclidean[4] manifolds. These can range from relatively simple finite-dimensional spaces, such as Kendall's shape sphere for triangles [42], to more intricate spaces like the space of curves [55] or the space of diffeomorphisms [7].

To define metrics between shapes, a convenient tool in functional analysis is the Reproducing Kernel Hilbert Space (RKHS) [8]. These are infinite-dimensional spaces that behave almost like finite-dimensional ones, in the sense that they are endowed with an inner product and that evaluation at a point is a continuous linear functional. This property is particularly useful when dealing with discrete data and interpolation schemes to approximate continuous objects. RKHSs, which are a central concept in this thesis, are also widely used in various fields of applied mathematics [10, 71, 82].

Recent advances in biological and medical data acquisition have enriched traditional spatial measurements with additional information and enabled more frequent or repeated acquisitions. This may take the form of features observed at each point — such as a signal value, as discussed in Chapter 1 — or temporal evolutions of the object, as in Chapter 2. As a result, theoretical models must be extended, often leading to more complex mathematical formulations. This, in turn, necessitates the development of new computational tools, such as those presented in Chapter 3, to make these methods scalable for modern datasets.

In this thesis, I present a series of theoretical and practical contributions to the field of shape analysis, with a particular focus on functional shapes and longitudinal data, and their applications in medical imaging. The computational tools developed are sufficiently general and low-level to be of interest to other communities, including statistics and machine learning. The work is organized into three main chapters, each addressing a distinct aspect of shape analysis and its applications.

[3] Citing D. G. Kendall in [41]: "*We here define 'shape' informally to be 'what is left when the differences which can be attributed to translations, rotations, and dilatations have been quotiented out'*".

[4] Citing D. Mumford in [59]: "*Shape is the ultimate non-linear sort of thing*".

Outline

Chapter 1 — Shape Analysis for Geometric and Functional Data

In the first chapter, I present theoretical and numerical developments of the functional shape (fshape) framework, originally introduced by A. Trouvé and N. Charon in [13], shortly before I joined the group at CMLA. The definition of fshapes was motivated by the need to analyze the variability in a dataset acquired through Optical Coherence Tomography of the optic nerve, in the context of glaucoma research. The data consists of distinct surface meshes (geometrical supports), each paired with a scalar signal representing thickness. A key application goal was to develop a classifier capable of detecting the disease, while also producing consistent thickness maps across patients — providing practitioners with a reliable indicator of each patient's condition.

On the theoretical side, functional shape analysis builds upon classical shape analysis, where geometric objects are studied modulo smooth, non-rigid deformations acting on them. The main challenge lies in extending existing methods — based on the Large Deformation Metric Mapping (LDDMM) framework — to handle both geometric and functional components simultaneously.

On the practical side, the first major bottleneck was the computational load. The discrete numerical scheme relies heavily on kernel methods, which have quadratic complexity and are difficult to apply efficiently to datasets of this size (hundreds of thousands of points). Another significant issue was numerical instability, which we were able to address by studying the theoretical properties of the underlying continuous equations.

I begin by defining the fshape bundle and introducing a class of deformations — called metamorphoses — that can modify both geometry and signal. I then define a distance between fshapes using a tool from geometric measure theory known as varifolds. This provides a powerful framework for defining a family of distances between geometric objects carrying features (signals or more advanced structures). These distances have several desirable properties, such as being differentiable and not requiring point-to-point correspondence between shapes.

I recall fine existence results, established in [Art3, Art6], for both the registration problem (estimating the deformation between a source and a target object) and the atlas problem (estimating a mean fshape and the deformations from it to each observation). This analysis revealed why the numerical experiments with L^2 signals were unstable — primarily due to mass cancellation effects — and led us to develop the theory in higher-regularity settings, specifically within Sobolev spaces H^s. Finally, I discuss the discrete scheme used in the experiments.

These works correspond to the papers [Art3, Art6, Art4, Art5, Art11], [Proc4, Proc2], as well as the FshapeTk software [Soft1].

Chapter 2 — Longitudinal Datasets and Shape Evolution

In the second chapter, I present a series of theoretical and practical results developed to model shape evolution. These developments were carried out during my time at ICM within the ARAMIS team, and working with S. Durrleman. Researchers at ICM are interested in understanding the causes, dynamics, and potential treatments for complex neurodegenerative diseases such as Alzheimer's disease and dementia. Leveraging the large, multimodal Alzheimer's Disease Neuroimaging Initiative (ADNI) database [39], which contains longitudinal data from patients, the aim is to explore the progression of Alzheimer's disease and contribute to the personalization of patient care.

It is convenient to model such data by representing each observation as a point on a manifold — for example, a shape manifold in the case of geometric structures like meshes describing subcortical anatomy. The evolution of these structures over the lifetime of a patient can then be described as a path on this abstract manifold, observed at discrete time points.

Since infinitesimal time evolutions are represented by tangent vectors, a central tool in Riemannian geometry for comparing such vectors defined at different points on a manifold is parallel transport. For this reason, the first section presents work on an approximation method — the Fanning Scheme — for computing the parallel transport of tangent vectors [Art7]. I establish the convergence rate of the method and demonstrate that the error decreases linearly. A key advantage of the fanning scheme is that it does not require computation of the Riemannian logarithm, which is often computationally expensive in real-world applications. I also discuss recent developments by [37], who demonstrated that alternative approaches, known as Ladder methods, can offer better performance for approximating parallel transport.

In the second part, I briefly present an application of parallel transport, showing how various regression models can be used to address the task of shape progression prediction [Proc5]. These models were implemented in the Deformetrica software [Soft2]. Finally, I introduce the application paper [Art10], which applies similar methods to analyze variability in Alzheimer's disease progression using multimodal data, including brain imaging, clinical and neuropsychological assessments, and biomarkers.

This chapter correspond to the papers [Art7, Art8, Art10], [Proc1, Proc5, Proc7], as well as the FshapeTk software [Soft2].

Chapter 3 — Kernel Methods in Action: General Formulas and Real-World Scale Datasets

In the final chapter, I introduce the KeOps library — developed in collaboration with J. Feydy and J. Glaunès — which is designed for efficient and user-friendly computation of large-scale kernel operations. A prototypical example of such operations is convolution, viewed as a matrix–vector product. These operations form the computational backbone of many of the methods discussed in the previous chapters. KeOps enables the use of arbitrary mathematical formulas for these kernel computations while maintaining high performance and scalability.

The main bottlenecks in kernel operations are twofold: computational complexity and memory usage. KeOps tackles these challenges by using brute-force computation, maintaining the quadratic time complexity, but offsetting it with highly parallelized execution on GPUs. This approach allows KeOps to achieve state-of-the-art performance on problems involving millions of data points. Most importantly, KeOps maintains a linear memory footprint, making it especially well-suited for large-scale applications where memory is a limiting factor.

KeOps allows users to define custom kernel operations from arbitrary mathematical formulas, while automatically handling their parallelized execution. To enable this, KeOps introduces the concept of symbolic tensors, which distinguish between three types of dimensions: batch, outer, and inner. Users can apply mathematical operations lazily on the inner dimension, meaning that computations are deferred and not immediately executed. Actual computation is only triggered when the outer dimension is reduced, at which point KeOps applies an optimized tiling reduction scheme to ensure both efficiency and scalability.

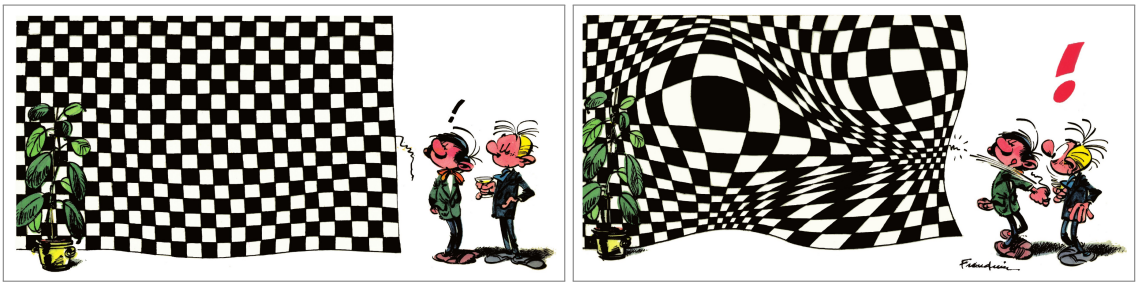
This design allows the symbolic tensor abstraction to be integrated into existing code with minimal boilerplate, making it easy to adopt. I explain how our internal automatic differentiation engine is fully compatible with PyTorch's autograd system, enabling seamless use within

optimization pipelines. I then demonstrate how high-level algebraic routines can be performed directly with symbolic tensors — for example, computing inverse kernel — based matrix–vector products, eigenspace decompositions, and more. Finally, I describe the internal architecture of the API, detailing its modular design and extensibility.

This work was published in [Art9], [Proc6, Proc8], and in the official documentation available at www.kernel-operations.io [Soft3].

CHAPTER 1

Shape Analysis for Geometric and Functional Data



Non rigid deformations by Franquin in [27]

Fshapes This chapter summarizes my work on functional shapes and presents, in a synthetic manner, the results of [Art3, Art6, Art4, Art5, Art11], and [Proc2]. This long-standing series of works was initiated during my postdoctoral position at CMLA, in the team of A. Trouvé, at the time when N. Charon was completing his PhD thesis. Collaboration with both of them subsequently continued, the latter moved to Johns Hopkins University as a faculty member at the Center for Imaging Science (CIS). During this period, it was also an opportunity to work in close collaboration with the team of F. Beg at Simon Fraser University, where the application paper on the retina dataset was part of the PhD thesis of S. Lee.

FshapesTk The fshapes framework is grounded in solid theoretical developments, which were guided and refined through numerical experiments on data. Several unusual numerical behaviors turned out to be indicators of genuine theoretical challenges, such as mass cancellation and gradient renormalization at boundaries. All numerical experiments were implemented in the FshapesTK software [Soft1], which I developed and actively maintained from 2012 to 2019[1]. It was coded in Matlab, with performance-critical components — primarily kernel-related operations — accelerated using C/CUDA code integrated via the Mex API[2]. FshapesTK incorporates numerous algorithmic stabilization techniques, as detailed in Section 9 of [Art3], making it a robust and

[1] Still running as of 2025!

[2] See https://www.mathworks.com/help/matlab/matlab_external/cpp-mex-api.html

reliable tool. Alongside Deformetrica [Soft2], it remains one of the few software packages capable of implementing atlas estimation frameworks for geometric data.

Outline We begin by introducing the Functional Shape framework and provide an application example, using the OCT dataset, that will be referenced throughout the chapter. We then describe the mathematical foundations for analyzing geometric and functional data, extending the classical non-rigid deformation framework traditionally used for geometric data. Finally, we present the discrete framework along with practical algorithms for computing key quantities, such as atlas registration. All of these concepts are defined and detailed in the sections that follow.

1.1 Functional shapes

1.1.1 Informal definition

Spatial data extracted from medical imaging often comes with additional measurements as thickness, pressure, diffusion direction, gene expression, etc. A general framework to model this type of data is to consider that measurement are given as a pair (x, s), where $x \in X$ denotes the geometric position and $s \in \mathcal{M}$ represents a feature attached to x. While X is typically a subset of $\mathbb{R}^2$ or $\mathbb{R}^3$ in practical applications based on medical imaging — as points, curves or surfaces –, the feature space $\mathcal{M}$ can take various forms, depending on both the acquisition modality and the object under study — a real number, a vector, a distribution, etc.

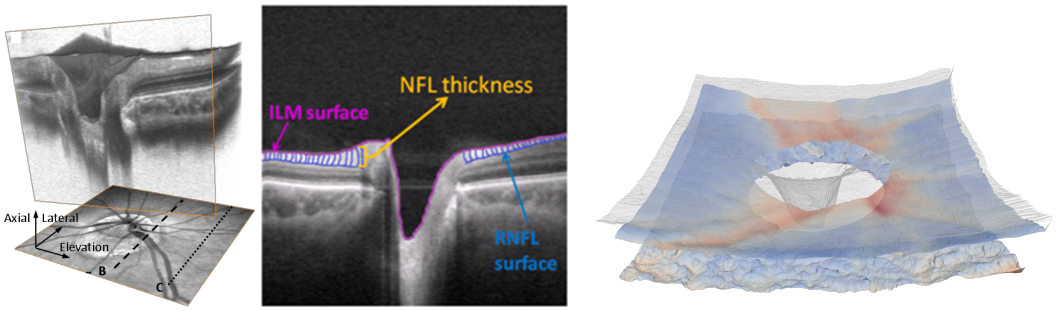
We introduce a framework where the geometric structure and its associated feature field are treated as a single entity and are processed jointly. Accordingly, the object of interest is generally represented as a pair (X, f) where X denotes the geometrical support — typically a manifold of dimension $d \in \mathbb{N}$ in an ambient space $\mathbb{R}^n$ with $d \leq n$ — and $f : X \to \mathcal{M}$ is a function defined on this manifold. We assume that the continuous object (X, f) is observed through a discrete sampling process, possibly involving a mesh, yielding samples $(x_i, s_i = f(x_i))_{i=1,\dots,N}$.

In the **functional shape** (fshape) framework, the feature space corresponds to real-valued signals, i.e. $\mathcal{M} = \mathbb{R}$. In our applications, a fshape consists of a pair (X, f), where X denotes the geometric support given by a surface in a 3D ambient space, and $f : X \to \mathbb{R}$ is a function defined on this surface. This definition is intentionally broad and can be adapted to accommodate different object and ambient space dimensions.

More general feature spaces have recently been explored in the literature. In some of my recent work, this exploration has been motivated by applications in spatial transcriptomics [Art15] and the modeling of structured deformations [PreP1]. This topic is further discussed in the concluding chapter.

1.1.2 Case Study: OCT Dataset

In practical applications conducted in collaboration with F. Beg's team, the geometry-feature pairs represent the retinal layer surface along with the corresponding thickness of that layer, mapped onto the surface. The dataset was acquired using Optical Coherence Tomography (OCT), a high-resolution 3D imaging technique for the posterior segment of the eye, as illustrated in Figure 1.1a. The study included a few dozen scans, each represented as three meshes corresponding to the retinal nerve fiber layer (RNFL), the Inner Limiting Membrane (ILM) and Bruch's membrane (BM) — with approximately 10^5 vertices each, as illustrated in Figure 1.1c.



(a) OCT scan (b) Segmentation (c) Meshes with signal

Figure 1.1: (a) 3D visualization of the Optic Nerve Head; (b) Example of a slice segmentation of ILM (magenta), posterior RNFL boundary (blue), BM (white), and posterior choroidal boundary (cyan); (c) Corresponding meshes color mapped with thickness. Courtesy of M. Sarunic (SFU).

We propose an approach for quantitative shape variability analysis of retinal OCT data using the fshape framework. This method enables the construction of a population-mean template from the geometry-function pairs extracted from each individual scan. Shape variability across multiple retinas is then measured through geometric deformations and functional residuals between this template and each subject's observation. In [Art4], we demonstrate the clinical relevance and practical application of this framework by generating atlases of the RNFL thickness for both glaucomatous and healthy subjects. These atlases reveal detailed spatial patterns of RNFL loss associated with glaucoma, as illustrated by the sample data in Figure 1.2. This work was further extended in [Art5], where we introduce high-level tools designed for clinical applications. The resulting analysis allows for comprehensive visualization and interpretation of morphometric patterns driven by multiple clinical and anatomical factors.

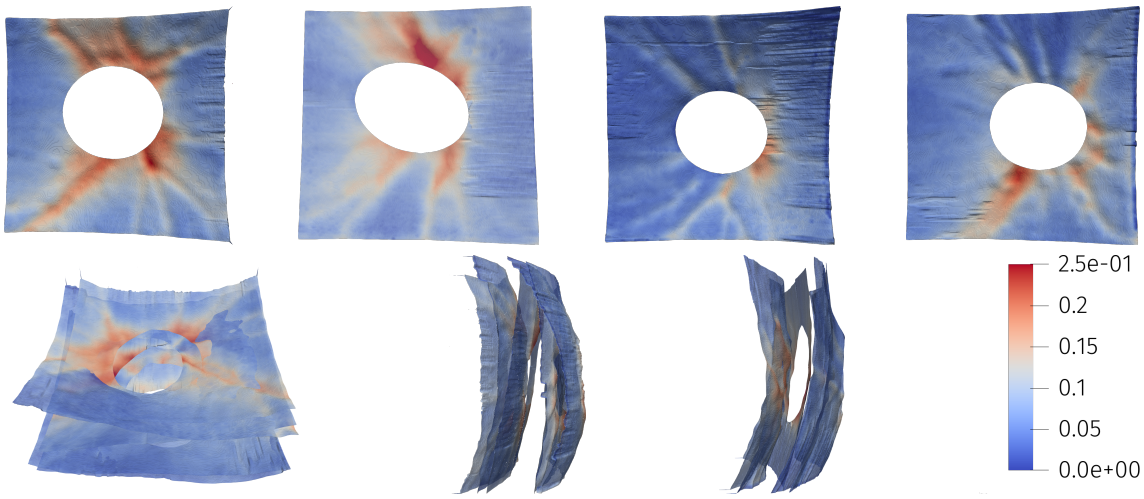


Figure 1.2: A sample from the OCT dataset. The first row depicts four examples of RNFL surfaces along with their corresponding RNFL thickness maps (in mm). The second row illustrates the relative positions of these surfaces from different viewing angles.

1.1.3 From shapes to fshapes analysis

The analysis of variability in a dataset composed of discrete fshapes is conducted in the spirit of classical shape theory. In Grenander's setting [32], shape spaces are modeled as sets of shapes that are homogeneous under the action of a group of spatial transformations. Metrics between shapes are then induced from right-invariant Riemannian metrics on the transformation group itself. The aim of this chapter is to introduce a similar, yet extended, framework tailored to the case of functional shapes. This approach is commonly used in the field of computational anatomy [62] and draws its theoretical foundations from several areas of mathematics, including geometry, metric measure theory, optimal control, and the calculus of variations. The objective of defining a comprehensive non-rigid deformation model for fshapes has progressed through the following steps, with each contribution recalled in the corresponding paper:

- Define an adequate mathematical structure to describe the observed data. This is achieved through the introduction of the fshape bundle in [Art3], which is further enhanced in [Art6].

- Define admissible deformations and their associated cost. We describe the metamorphosis framework, which precisely models how geometry and signal deform jointly. The initial setting introduced in [Art3] was restricted to signals in L^2 spaces. However, numerical experiments revealed the emergence of signal oscillations. The theoretical explanation lies in the fact that the corresponding metrics may be too weak, failing to sufficiently penalize variations in the signal. To address this, stronger metrics were considered — specifically, Sobolev-type metrics in [Art6] and bounded variation norms in [Art11].

- Define a dissimilarity measure to compare objects. This is achieved through the functional varifold distance — a smooth and flexible metric introduced in [Art3]. It is versatile, adaptable to new imaging modalities [56], and scalable to real-world datasets, as demonstrated in [Proc2], thanks to the KeOps library (see Chapter 3).

- Solve the registration problem, which consists in finding a minimum-energy deformation that sends a fshape close to another. Generating these minimum-energy deformations is formulated as an optimal control problem, leading to a Hamiltonian formulation (geodesic shooting). The discrete case is studied in [Art3], and the general continuous case in [Art6]. A coherent discrete framework is established for both the deformation model and the dissimilarity measure: in [Art3] for signals in L^2 space, and in [Art6] for signals in Sobolev spaces. The convergence of the (static) discrete scheme toward its continuous counterpart is proven within the framework of Γ-convergence in [Art11].

- Solve the atlas estimation problem, which consists in estimating a mean fshape from a set of fshape samples, along with the deformations from this central object to each observed data point. We prove the existence of solutions in [Art3], and implement a practical algorithm in [Art3], demonstrating its effectiveness on real datasets in [Art4, Art5].

All these questions are the subject of a vast body of literature, which can be loosely defined under the umbrella of shape analysis [82]. Similar problems have been addressed in other modalities, such as 2D or 3D images, and under different group actions.

1.2 Deformation of fshapes

1.2.1 Fshape Bundle

We consider shapes as geometric objects embedded in a given ambient vector space $\mathbb{R}^n$. These shapes are sub-manifolds $X \subset \mathbb{R}^n$ (with or without boundary) of dimension d, where $1 \leq d \leq n$, and such that both X and its boundary possess regularity of order $s \geq 0$.

The paper [Art3] corresponds to the case $s = 0$, where X is assumed to be a rectifiable set and is endowed with a signal f in the space of square-integrable functions $L^2(X)$, equipped with the norm

$$\|f\|^2_{L^2(X)} = \int_X |f(x)|^2 \, d\mathcal{H}^d(x) < \infty,$$

where $\mathcal{H}^d$ denotes the d-dimensional Hausdorff (or volume) measure. The paper [Art6] addresses the case of higher regularity, with $s \geq 1$. The manifolds X considered are of class C^s, and the associated function spaces are Sobolev spaces $H^s(X)$. The norm $\|f\|_{H^s(X)} = \sum_{k=0}^{s} \|\nabla^k f\|^2_{L^2(X)}$ ensures that all covariant derivatives of f up to order $k \leq s$ are square-integrable.

In the fshape framework, we will consider groups $G = G_V$ of smooth non-rigid deformations of $\mathbb{R}^n$ based on the Large Deformation Diffeomorphic Metric Mapping (LDDMM) setting [9, 82]. These deformations are obtained as flows of time-dependent velocity fields modeled on a Reproducing Kernel Hilbert Space (RKHS) V of smooth vector fields. The space V is chosen so that the induced deformations are compatible with the regularity s of the submanifold X.In this context, shape spaces are defined as orbits of a fixed bounded C^s submanifold X_0 (the **template**) under the action of G_V, that is $\mathcal{S} = \{\phi(X_0) \mid \phi \in G_V\}$ which turns $\mathcal{S}$ into a homogeneous space. The **fshape bundle** of regularity s modeled on the set $\mathcal{S}$ is:

$$\mathcal{F}^s_{\mathcal{S}} = \{(X, f) \mid X \in \mathcal{S}, f \in H^s(X)\}.$$

This defines a vector bundle structure, whose fibers are Sobolev spaces parametrized by the orbit $\mathcal{S}$ of X_0. Indeed, strictly speaking, each functional space $H^s(X)$ is distinct — since the domain of definition of functions in $H^s(X)$ is the shape X itself — yet these spaces are smoothly related through the deformation group action. In the case of the OCT dataset, we can think of each RNFL surface with its associated thickness signal as an image whose support is a different surface embedded in the ambient space $\mathbb{R}^3$.

The geometric deformation extends naturally to fshapes in $\mathcal{F}^s_{\mathcal{S}}$ as follows:

$$\phi \cdot (X, f) = (\phi(X), f \circ \phi^{-1})$$

which corresponds to deforming the geometry by ϕ while pulling the signal back onto the deformed shape $\phi(X)$. Note that in this case, the action on the signal is canonically defined, since the feature space is one-dimensional. However, when dealing with more complex features, the modeling of the deformation action involves non-trivial choices and must be carefully formulated. This point is made explicit in recent work on implicit deformation modules [PreP1].

1.2.2 Metamorphoses

The action of G_V on the fshape bundle considered so far only accounts for the geometrical part of fshape variability corresponding to an horizontal motions in the fshape bundle Figure 1.3. To complete it, we also need to introduce vertical motions in $\mathcal{F}^s_{\mathcal{S}}$ which are essentially variations of signal functions within a given fiber.

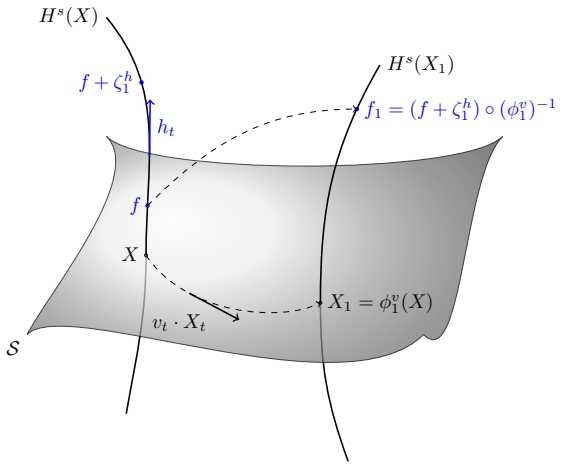


Figure 1.3: Fshape bundle and metamorphosis.

Thus, an fshape transformation is a combination of a geometrical deformation $\phi \in G_V$ and the addition of a residual signal function ζ to the signal part of the fshape. Namely, if $(X, f) \in \mathcal{F}^s_{\mathcal{S}}$ and $(\phi, \zeta) \in G_V \times H^s(X)$, we shall consider the deformation:

$$(\phi, \zeta) \cdot (X, f) = (\phi(X), (f + \zeta) \circ \phi^{-1}). \tag{1.1}$$

Note that unlike the classical setting of shape spaces, without further assumptions, this can be no longer considered as an actual group action since the set of all transformations (ϕ, h) in $\mathcal{F}^s_{\mathcal{S}}$ is not even a group but should be rather thought as a section of the bundle $G_V \times \mathcal{F}^s_{\mathcal{S}}$.

We define a metamorphosis of (X, f) as a couple of a time-varying infinitesimal deformation $v \in L^2([0,1], V)$ and infinitesimal signal variation $h \in L^2([0,1], H^s(X))$. The time integration of $(v, h) \in L^2\big([0,1], V \times H^s(X)\big)$ parametrizes an fshape transformation path (ϕ^v_t, ζ^h_t) with $\phi^v_t \in G_V$ and $\zeta^h_t \in H^s(X)$ through the dynamical equations:

$$\begin{cases} \dot{\phi}^v_t = v_t \circ \phi^v_t \\ \dot{\zeta}^h_t = h_t \\ \phi^v_0 = \mathrm{Id},\ \zeta^h_0 = 0 \end{cases} \tag{1.2}$$

We then define the infinitesimal metric on $V \times H^s(X)$ by $\|(v, h)\|^2_{(X,f)} = \frac{\gamma_V}{2}\|v\|^2_V + \frac{\gamma_f}{2}\|h\|^2_{H^s(X)}$ where $\gamma_V, \gamma_f > 0$ are weighting parameters. The magnitude of these coefficients governs the trade-off between changing the geometry and changing the signal, and should be chosen based on the specific problem at hand. In integrated form, this gives the following energy of the path (ϕ^v_t, ζ^h_t):

$$E_X(v, h) = \frac{\gamma_V}{2} \int_0^1 \|v_t\|^2_V dt + \frac{\gamma_f}{2} \int_0^1 \|h_t \circ (\phi^v_t)^{-1}\|^2_{H^s(X_t)} dt \tag{1.3}$$

with $X_t \doteq \phi^v_t(X)$. Note that the penalty on the signal variation h_t at each time is measured on the deformed submanifold X_t with respect to the metric $\|\cdot\|_{H^s(X_t)}$.

Finally, we can define a distance between two given fshapes (X, f) and (X', f') in the bundle $\mathcal{F}^s_{\mathcal{S}}$ by setting

$$d_{\mathcal{F}^s_{\mathcal{S}}}\big((X, f), (X', f')\big)^2 = \inf\left\{E(v, h) \mid (\phi^v_1, \zeta^h_1) \cdot (X, f) = (X', f')\right\} \tag{1.4}$$

The space $(\mathcal{F}^s_{\mathcal{S}}, d_{\mathcal{F}^s_{\mathcal{S}}})$ is a complete metric space (Property 1 in [Art6]) , and a geodesic exists between any two fshapes (Theorem 2 in [Art3]). Moreover, given a sample $(X^\ell, f^\ell)_{1 \le \ell \le P}$ of fshapes

in $\mathcal{F}_{\mathcal{S}}$, there exist $(X_*, f_*) = \min_{(X,f)\in\mathcal{F}_{\mathcal{S}}} \sum_{\ell=1}^{P} d_{\mathcal{F}_{\mathcal{S}}}\big((X,f),(X^\ell,f^\ell)\big)^2$. This miminizer of the sum of the square distances to each (X^ℓ, f^ℓ) is known as the Karcher mean, and its existence is established in Theorem 3 of [Art3]. These results are satisfactory as they provide a convenient mathematical framework to perform first order (non-linear) statistical analysis in the fshape bundle. However, they have limited practical utility because the fshapes observed in a sample, such as in the OCT dataset, are unlikely to belong to the orbit $\mathcal{S}$ of a common template. Therefore, we require an additional tool to compare arbitrary fshapes, such as the functional varifold norm defined in Section 1.3.

1.2.3 Tangential model

The framework referred to as the **tangential model** [Art3], is obtained by neglecting the metric changes in Equation (1.3) and approximating the signal cost by $\|h_t\|^2_{H^s(X_0)}$ instead. In this formulation, the cost of a signal change depends solely on the source geometry. While this leads to a loss of some of the Riemannian-like properties characteristic of the metamorphosis setting, important practical problems — such as registration — still admit solutions as shown by Theorem 4 in [Art3]. Moreover, this simplification significantly reduces the complexity of the geodesic shooting equations — particularly the adjoint system, which must be integrated backward in time to compute the gradient of the energy (notably, this was before automatic differentiation became standard practice). This allowed for faster computations and more stable numerical scheme [Art4, Art5]. However, it also leads to a partial decoupling between geometry and signal evolution. See also the Section 1.5 below.

Let us consider Figure 1.4 for a comparison of minimum-energy trajectories under the metamorphosis and tangential models. Denoting $\mathbb{S}^2$ the unit sphere in $\mathbb{R}^3$, the initial fshape is $(\mathbb{S}^2, 0)$ and the final fshape is $(2\mathbb{S}^2, 1)$. In other words, we deform a sphere such that both its radius and its signal increase by one unit. Since we are using radial kernels, it turns out that the entire path in fshape bundle remains within the set of spheres with uniform signal throughout the deformation. In the tangential setting (top row), the optimal path corresponds to a linear interpolation of both geometry and signal. However, in the metamorphosis model, the volume measure contributes to the signal variation cost. As a result, geometry and signal must evolve together to find a more efficient (i.e., lower energy) trajectory. The optimal strategy involves first contracting the sphere — note the reduced radius at $t = 0.4$ — which reduces the cost of modifying the signal. Then, the radius expands again to reach the final target geometry at $t = 1$. The amplitude of this bouncing effect depends on the relative size of γ_V and γ_f as explained in Section 3.3.5 of [Art6].

1.3 Distance Between Functional Shapes in the Functional Varifold Framework

1.3.1 Measuring Distances Between Shapes

The previous framework has primarily focused on comparing fshapes within a single, common bundle $\mathcal{F}$. However, this approach proves impractical in most real-world applications, where the fshapes in a dataset cannot reasonably be assumed to lie within the same bundle. This is mainly because the geometric supports of two subjects are not necessarily related by a deformation in the group G.

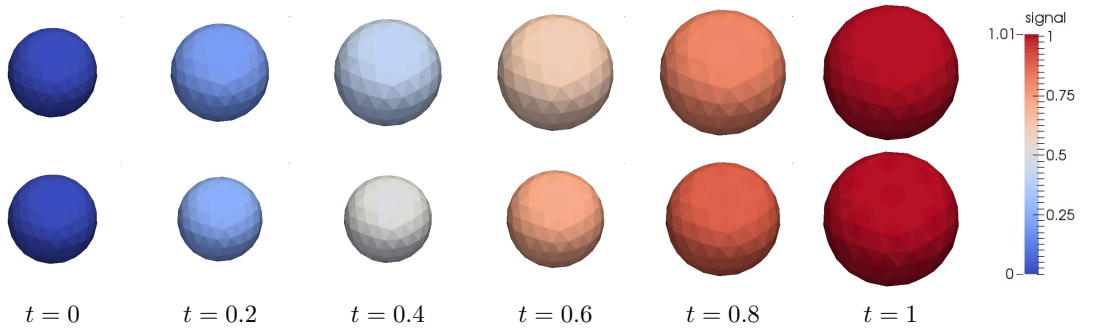


Figure 1.4: Mimimun-Energy path in the tangential model (top) versus metamorphosis (bottom) computed with the FshapeTk.

In registration problems, it is common to introduce additional dissimilarity terms (also called data attachment term or goodness-of-fit) to the deformation cost, which can also be interpreted as modeling noise intensity, see Section 1.4.2. In standard image registration, these terms typically take the form of a simple squared L^2 norm between the deformed template image and the target. However, the case of functional shapes is more complex: given two fshapes, the lack of explicit correspondence between their geometric supports prevents any direct comparison of their signals.

Several distances have been proposed to compare geometric objects, and a review can be found in [Chap1]. The classical Hausdorff distance is theoretically appealing but computationally expensive and sensitive to noise. Optimal transport distances have received renewed interest, particularly with relaxed formulations of the problem [66], but they remain computationally demanding (e.g., due to Sinkhorn iterations). Kernel-based current distances [77] are attractive thanks to their solid mathematical foundations and their adaptability to functional data [13]. However, they require geometries with consistently oriented meshes — which can be tricky to achieve in automatic segmentation and meshing pipelines, as used in Section 2.2.2 — and can suffer from numerical instabilities, such as mass cancellation effects when the geometry contracts.

To address this issue, varifold distances were introduced into the field of computational anatomy by [14], where Theorem 4.1 demonstrates why they are not subject to the aforementioned shortcomings of currents. The mathematical concept of varifolds[3] originates from geometric measure theory [4, 2], initially motivated by the Plateau problem. It is striking to see how, decades later, these abstract theoretical tools have led to practical solutions in applied fields. More recently, the discrete approximation of varifolds has been extensively studied in [12, 11]. In this work, we present the extension to functional varifolds as introduced in [Art3].

A comparison between the current and varifold approaches was carried out in [Proc2] within the unifying framework of oriented varifolds. It was demonstrated that kernel-based varifold distances possess several desirable properties: they are smooth, adaptable to various data modalities, robust to noise and missing data, and scalable to large datasets.

1.3.2 Functional Shapes as Functional Varifolds

Definition

For any integer $1 \leq d \leq n$, we denote by G_d^n the Grassmann manifold of all d-dimensional (non-oriented) subspaces of $\mathbb{R}^n$. An element $V \in G_d^n$ can be identified with the orthogonal projector

[3] Short for variational manifolds.

onto V. In the special cases where $d = 1$ (resp. $d = n - 1$, i.e., co-dimension one), the subspace V can be naturally identified with the range (resp. the null space) of the projector. This is why, up to a sign, when $n = 2, 3$, the orientation of one-dimensional objects such as curves is typically represented using unit tangent vectors, while the orientation of two-dimensional objects such as surfaces is represented using unit normal vectors. This was used in the definition of the **oriented varifold** setting in [Proc2]. In what follows, we will refer to the orientation vector as $t \in \mathbb{S}^{n-1}$.

We say that μ is a d-dimensional **functional varifold** (fvarifold, in short) if μ is a finite Borel measure on the space $\mathbb{R}^n \times G_d^n \times \mathbb{R}$. Note that we consider only the case of real-valued signals here, but this could be extended to other feature spaces [56]. This means that a fshape may be viewed as a distribution over the space (position $\times$ orientation $\times$ feature). For instance, it amounts to representing a functional curve or surface as the distribution of its points, with a unit orientation vector attached, together with the signal value.

A fshape (X, f) of dimension d in $\mathbb{R}^n$ can be represented by a fvarifold $\mu_{(X,f)}$. This measure can integrate any function $\omega \in C_0(\mathbb{R}^n \times G_d^n \times \mathbb{R})$, the space of continuous functions that vanish at infinity. Namely, we have

$$\mu_{(X,f)}(\omega) = \int_X \omega(x, T_x X, f(x)) d\mathcal{H}^d(x), \tag{1.5}$$

where we denoted $T_x X$ the tangent space of X at point x. The identification $X \mapsto \mu_X$ defines an injection from the set of smooth sub-manifolds into the dual space W^* of a suitable space $W \subset C_0(\mathbb{R}^n \times G_d^n \times \mathbb{R})$ of test functions on $\mathbb{R}^n \times G_d^n \times \mathbb{R}$, provided that this test function space is sufficiently rich. Spaces W that are Reproducing Kernel Hilbert Spaces (RKHS) is a convenient setting and will be discussed in more detail below.

Some particularly simple fvarifolds are the *Dirac delta distributions*, which in this context can be written in the form $\delta_{(x,V,s)}$, with $x \in \mathbb{R}^n$, $V \in G_d^n$, and $s \in \mathbb{R}$. These are defined by the relation $\delta_{(x,V,s)}(\omega) = \omega(x, V, s)$ for any test function ω. Such singular measures will be used to approximate polyhedral fshapes as finite sums of Dirac deltas. Each Dirac precisely encodes the location of a face of the fshape, its orientation, and the associated signal value.

Deformation of Fvarifold

We now need to define how to deform fvarifolds in a way that is consistent with deformation of fshapes. Let $\phi : \mathbb{R}^n \to \mathbb{R}^n$ be a diffeomorphism and h a measurable function on $\mathbb{R}^n$, this is given by the following set of equations:

$$\begin{cases} \forall \omega \in C_0(\mathbb{R}^n \times G_d^n \times \mathbb{R}),\ \big((\phi, \zeta) \cdot \mu\big)(\omega) = \mu\big((\phi, \zeta) \cdot \omega\big) \\ \text{where} \\ \big((\phi, \zeta) \cdot \omega\big)(x, V, s) = |d_x \phi_{\restriction V}|\, \omega(\phi(x), d_x\phi(V), s + \zeta) \end{cases}$$

where for $V \in G_d^n$, $|d_x \phi_{\restriction V}|$ denotes the Jacobian of ϕ along subspace V (*i.e.* the volume change along V at point x) and $d_x\phi(V)$ is the image of V by the invertible linear application $d_x\phi$. This yields, by Proposition 1 in [Art3], to the natural relation

$$(\phi, \zeta) \cdot \mu_{(X,f)} = \mu_{(\phi(X),(f+\zeta)\circ\phi^{-1})}.$$

1.3.3 RKHS Based Distances Between Fvarifold

As mentioned above, we focus on a particular class of test functions given by RKHS. In the context of fvarifolds, an admissible space W is a Hilbert space that is continuously embedded in

$C_0(\mathbb{R}^n \times G_d^n \times \mathbb{R})$. According to the standard theory of RKHS [8], such a space can be equivalently described by a positive and continuous kernel k defined on $\mathbb{R}^n \times G_d^n \times \mathbb{R}$.

A natural and convenient way to define kernels on product spaces is to consider tensor products of kernels. Let k_{pos}, k_{or}, and k_{sig} be three positive definite kernels of class C^1, defined on $\mathbb{R}^n$, G_d^n, and $\mathbb{R}$, respectively. The RKHS associated with the kernel, defined for all $x_1, x_2 \in \mathbb{R}^n$, $V_1, V_2 \in G_d^n$, and $s_1, s_2 \in \mathbb{R}$ by

$$k_{\text{pos}} \otimes k_{\text{or}} \otimes k_{\text{sig}}\big((x_1, V_1, s_1), (x_2, V_2, s_2)\big) = k_{\text{pos}}(x_1, x_2)\, k_{\text{or}}(V_1, V_2)\, k_{\text{sig}}(s_1, s_2)$$

is admissible as shown by Proposition 2 of [Art3]. Although these separable kernels do not cover the entire possible set of kernels on $\mathbb{R}^n \times G_d^n \times \mathbb{R}$, the tensor product construction has the advantage of providing a large class of metrics through the various possible choices of k_{pos}, k_{or}, and k_{sig}, while being easy to interpret in terms of the combination of spatial and orientation characteristics.

In particular, choosing kernels with symmetry (i.e., those that are invariant under a group action) can lead to a metric with desirable properties. For instance, let us consider two functions $\rho_{\text{pos}}, \rho_{\text{or}} : \mathbb{R} \to \mathbb{R}$. If $k_{\text{pos}}(x, y) = \rho_{\text{pos}}(\|x - y\|_{\mathbb{R}^n})$ is a radial kernel, then the varifold norm will be translation invariant. In the case $d = 2, 3$, and $k_{\text{or}}(t_1, t_2) = \rho_{\text{or}}(\langle t_1, t_2 \rangle_{\mathbb{R}^n})$ for orientation vectors $t_1, t_2 \in \mathbb{S}^{n-1}$ and an even function $\rho_{\text{or}}(\cdot) = \rho_{\text{or}}(-\cdot)$, the norm will be invariant under reorientation. This is particularly useful when meshes are not properly oriented. We refer to Section 4.3 of [Proc2] for a more detailed discussion on various kernel options.

With these assumptions, W and its dual W^* are Hilbert spaces, and we denote the Hilbert norm on W^* by $\|\cdot\|_{W^*}$. Moreover, for any fshapes (X, f) and (Y, g), we have that $\mu(X, f)$ and $\mu_{(Y,g)}$ belong to W^*, and the resulting inner product is given by:

$$\langle \mu_{(X,f)}, \mu_{(Y,g)} \rangle_{W^*} = \int_X \int_Y k_{\text{pos}}(x, y)\, k_{\text{or}}(T_x X, t_y Y)\, k_{\text{sig}}(f(x), g(y)) d\mathcal{H}^d(x) d\mathcal{H}^d(y), \tag{1.6}$$

where as before $T_x X$ (resp. $T_y Y$) denotes the tangent space to X (resp. Y) at the point x (resp. y). Thus, a dissimilarity measure between any two fshapes can be defined as the norm of the difference in the space of fvarifolds, i.e.,

$$\|\mu_{(X,f)} - \mu_{(Y,g)}\|^2_{W^*} = \langle \mu_{(X,f)}, \mu_{(X,f)} \rangle_{W^*} + \langle \mu_{(Y,g)}, \mu_{(Y,g)} \rangle_{W^*} - 2\langle \mu_{(X,f)}, \mu_{(Y,g)} \rangle_{W^*}, \tag{1.7}$$

which can be computed easily using (1.6).

Not every choice of kernels k_{pos}, k_{or}, and k_{sig} results in a valid distance on the space of varifolds; it is possible for two distinct fshapes to have a distance of zero. A valid distance is guaranteed if and only if the RKHS W is dense in $C_0(\mathbb{R}^n \times G_d^n \times \mathbb{R})$, in which case the kernel k is said to be c_0-universal. For example, Gaussian kernels satisfy this property [73].

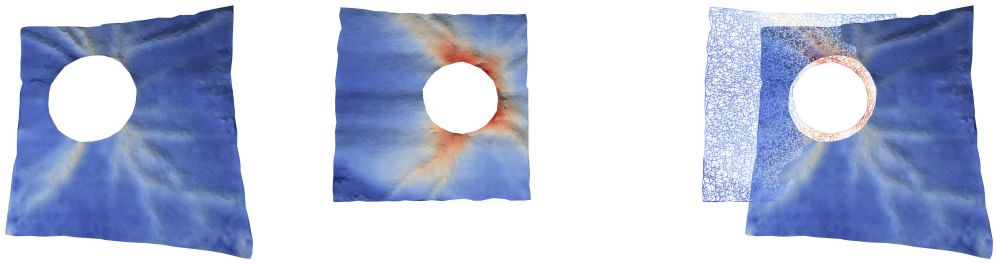
1.4 Registration of Fshapes and Atlas Estimation

1.4.1 Registration (a.k.a. Matching)

Problem Definition

In shape analysis, the **registration** (or inexact matching) problem amount to find a deformation ϕ_* between a source S and a target T objects. It often consists of optimizing a problem of the form

$$\min_{\phi \in \mathcal{D}} \text{energy}(\phi) + \text{discrepancy}(\phi \cdot S, T)$$



(a) Source (b) Target (c) Relative position

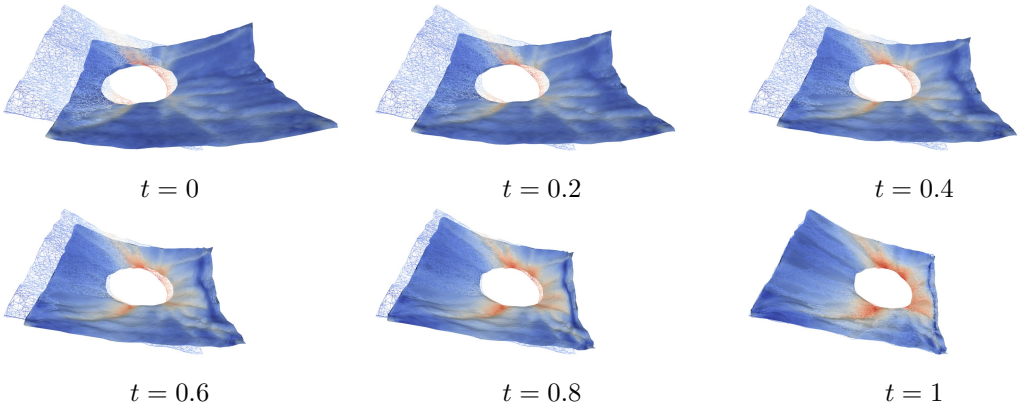
Figure 1.5: Two fshapes from the OCT dataset (en face view).

over a set of admissible deformations $\mathcal{D}$. The deformed source $\phi_* \cdot S$ should then be similar to the target T, while the energy of deformation — somehow measuring the complexity of ϕ_* — remains low. In the setting outlined above, objects are fshapes with metamorphoses acting as deformations, the energy of deformation is given by formula (1.3), and the data attachment term is a fvarifold norm of Equation (1.7). Similarly to the classical LDDMM framework, metamorphoses are generated as the flow up to time $t = 1$ of infinitesimal deformations, but here affecting both geometry and signal. This allows interpolation between the source, since $(\phi_{t=0}, \zeta_{t=0}) = (\mathrm{Id}, 0)$, and the final deformation $\phi = (\phi_{t=1}, \zeta_{t=1})$ as illustrated in Figure 1.6.

Given a template fshape (X^0, f^0) and a target $(X^{\text{tar}}, f^{\text{tar}})$, we will focus on variational problems that have the general form:

$$\begin{cases} (v_*, h_*) = \underset{\substack{v \in L^2([0,1],V) \\ h \in L^2([0,1],H^s(X_0))}}{\mathrm{Arginf}} \{E_X(v,h) + A(X_1, f_1)\} \\ \dot{\phi}^v_t = v_t \circ \phi_t,\ \dot{\zeta}^h_t = h_t \\ \phi^v_0 = \mathrm{Id},\ \zeta^h_0 = 0 \end{cases} \tag{1.8}$$

where $A(X_1, f_1)$ is a data attachment term between the transformed template shape $(X_1, f_1) = \left(\phi^v_1(X^0), (f^0 + \zeta^h_1) \circ (\phi^v_1)^{-1}\right)$ and the target shape $(X^{\text{tar}}, f^{\text{tar}})$. The function $\zeta^h_1 \in H^s(X_0)$ is called **functional residual** and is of particular interest as it encode the signal variation to account difference between the two fshapes. In the following, we consider the case where the data attachment term $A(X_1, f_1) = \|\mu_{(X_1,f_1)} - \mu_{(X^{\text{tar}},f^{\text{tar}})}\|^2_{W^*}$ is a RKHS based fvarifold norm given by



$t = 0$ $t = 0.2$ $t = 0.4$

$t = 0.6$ $t = 0.8$ $t = 1$

Figure 1.6: A registration with metamorphosis ($s = 1$) between the two fshapes shown in Figure 1.5. Both the geometry and signal are evolving along the time.

formula (1.7).

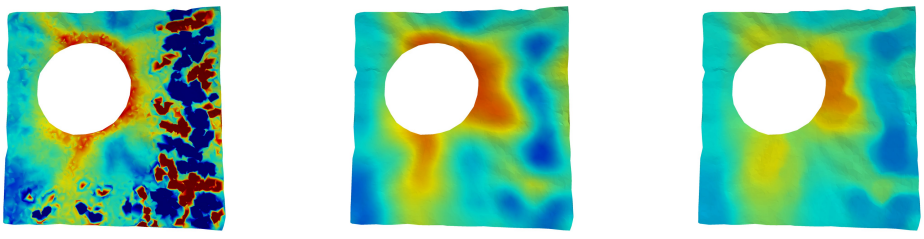
Existence Results

Equation (1.8) defines an optimal control problem, with two controls given by the deformation field v and the variable h of signal transformation. The existence of solutions of registration problem (1.8) have been extensively studied in Section 5 of [Art3] and Section 3.2 of [Art6] for $s \geq 1$. To summarize, the various results

- Theorem 3 in [Art6] shows the existence of solutions to the registration problem (1.8) in the case where the data attachment term A is lower semi-continuous in $L^2([0,1], V \times H^s(X))$. Unfortunately, this result does not cover the case where A is a fvarifold norm with L^2 signal (corresponding to $s = 0$), as fvarifold terms are generally not lower semi-continuous with respect to the weak convergence in $L^2([0,1], L^2(X))$.

- Theorem 7 in [Art3] establishes the existence of solutions to the registration problem (1.8) when A is a fvarifold norm with L^2 signal, under the following assumptions: the kernels k_{pos}, k_{or}, and k_{sig} are sufficiently smooth, and the ratio γ_f/γ_W — the weights appearing in the energy (1.3) — is large enough. This latter condition on the penalty weights has practical implications in numerical experiments. When signal variations are not sufficiently penalized, oscillations appear, as illustrated in Figure 1.7a. Conversely, an excessively large γ_f forces the functional residuals to remain nearly null. These numerical instabilities motivated the subsequent works [Art6, Art11], where norms that explicitly penalize signal variations are considered.

- Theorem 5 in [Art6] addresses the case where A is a fvarifold norm with an H^s signal ($s \geq 1$) in the registration problem (1.8). Existence of solutions is guaranteed provided that the kernels k_{pos}, k_{or}, and k_{sig} are sufficiently smooth. Notably, no additional conditions on the weights are required. Numerical experiments involving such Sobolev norms yield significantly improved functional residuals compared to the L^2 case, as illustrated in Figures 1.7b. Figure 1.7c suggests that the BV norm could be a viable alternative. The trade-off, however, is increased model complexity for these norms and higher computational cost.



(a) L^2 norm (b) H^1 norm (c) BV norm penalized

Figure 1.7: The functional residuals ζ_1 in three different registration involving various functional norms. Data are depicted of Figure 1.5. As theoretical result on existence suggest, penalizing signal variation stabilized the algorithm.

Solution approximations

A full characterization of solutions to the registration problem (1.8), along with the derivation of a conservation law along the minimum energy path via a Hamiltonian formulation, is provided in Section 3.3 of [Art6]. This deep understanding of the properties of the continuous model has played an important role in guiding the design of appropriate discrete models and algorithms. In Section 1.5, we present the discrete formulation adopted in the FshapesTk implementation [Soft1]. The practical computation of a registration between two fshapes, as illustrated in Figures 1.6 and 1.7, is performed by solving a variational problem using an adaptive gradient descent algorithm, as described in Section 1.5.4. Despite the use of highly parallelized code for kernel operations, registration remains computationally intensive due to the reliance on numerical time integration (typically using a second-order Runge–Kutta scheme). The examples presented in this chapter typically require several minutes to compute.

1.4.2 Atlas

Going further the registration problem and keeping notations of Section 1.4.1, the **atlas estimation** problem aims to jointly estimate a template shape S_* and a collection of deformations $\{\phi_*^\ell\}$ aligning this template to a set of observed shapes $\{T^\ell\}_{\ell=1}^P$. It typically involves solving an optimization problem of the form

$$\min_{S\in\mathcal{S},\,\{\phi^\ell\}\subset\mathcal{D}} \text{penalty}(S) + \sum_{\ell=1}^{P} \left\{\text{energy}(\phi^\ell) + \text{discrepancy}(\phi^\ell\cdot S, T^\ell)\right\},$$

where $\mathcal{S}$ denotes the set of admissible shapes. The objective is to identify a shape S^0, referred to as the **hypertemplate**, that is both simple and representative, and that can be accurately deformed to approximate the observed data. In this setting, $\mathcal{S}$ is defined as the fshape bundle over the fixed reference shape S^0, meaning that S_* is a deformation of S^0, see Figure 1.8. In the atlas estimation framework, the penalty term associated with the mean template corresponds to the energy of the deformation mapping S^0 onto S_*. While the hypertemplate may be chosen from among the observations, it is preferably constructed as a simple, smooth prototype with plausible topology, in order to reduce bias toward any particular individual.

Bayesian interpretation

Let us now consider a sample of P observed fshapes $(X^\ell, f^\ell)_{\ell=1,\dots,P}$. We introduce a forward generative model [20] for which observations are noisy geometric-functional transformations of a common unknown template fshape (X, f) plus additional noise terms:

$$(X^\ell, f^\ell) = (\phi^\ell, \zeta^\ell)\cdot(X, f) + \varepsilon^\ell, \qquad \text{for all } \ell = 1,\dots,P$$

As above, ϕ^ℓ is the flow of a vector field $v^\ell \in L^2([0,1], V)$ and (v^ℓ, ζ^ℓ) are regarded as hidden latent variables of the transformations from template to subjects, ε^ℓ 's are noise variables. Considering i.i.d. variables ε^ℓ, we may define a noise model on fshapes based on fvarifold metrics:

$$p(\varepsilon^\ell) = p\left((X^\ell, f^\ell) \mid (X, f), (v^\ell, \zeta^\ell)\right) \propto e^{-\frac{1}{2\sigma_W^2}\|\mu_{(\phi^\ell,\zeta^\ell)\cdot(X,f)} - \mu_{(X^\ell,f^\ell)}\|_{W^*}^2}$$

which is a Gaussian model with respect to the metric $\|\cdot\|_{W^*}$. Note that this is only formal for the infinite dimensional space of fvarifolds but can be given a rigorous sense if restricted to a

[4] See the video at https://miat.inrae.fr/bcharlier/soft/img/atlas_H1.webm

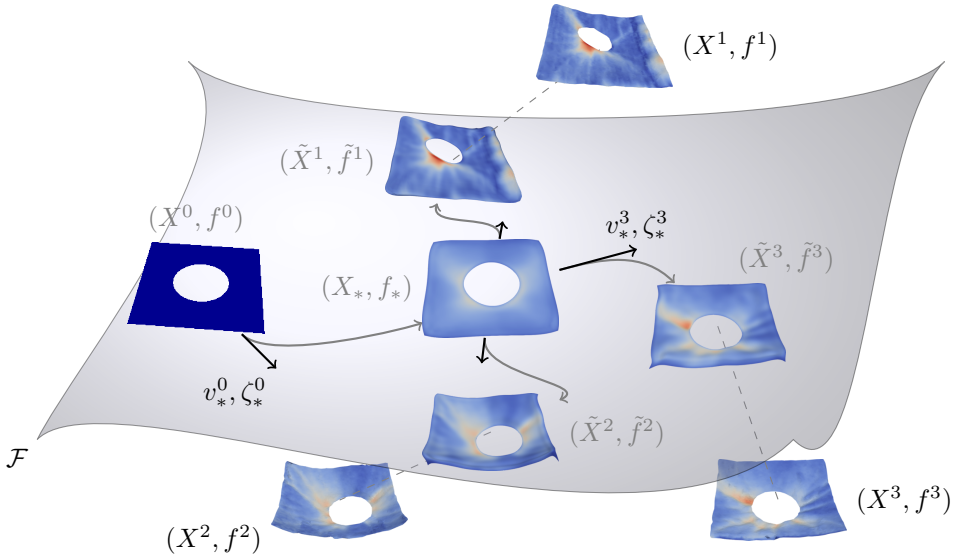


Figure 1.8: Atlas estimation on the OCT dataset performed using the hypertemplate algorithm[4]. We illustrate the case with $P = 3$ observations, where the deformed template for each $\ell = 1, \dots, P$ is denoted by $(\tilde{X}^\ell, \tilde{f}^\ell) = (\phi^\ell_*, \zeta^\ell_*) \cdot (X_*, f_*)$.

predefined discrete grid, similarly to [29]. As for the latent variables $\left(v^\ell, \zeta^\ell\right)_\ell$, we take the following prior deriving from the energy (1.3)

$$p\left((v^\ell, \zeta^\ell) \mid (X, f)\right) \propto e^{-E_X(v^\ell, \zeta^\ell)^2}, \quad \text{for all } \ell = 1, \dots, P$$

which is essentially assuming independent (formal) Gaussian distribution on v and ζ in their respective metric spaces and with variance $\sigma_V^2 = \frac{1}{\gamma_V}$ and $\sigma_f^2 = \frac{1}{\gamma_f}$.

Finally, we also model the template (X, f) as a random variable itself. Inspired from the hypertemplate model for shape atlases of [49], we represent (X, f) as a transformation of a given hypertemplate fshape (X^0, f^0), i.e $(X, f) = (\phi^0, \zeta^0) \cdot (X^0, \zeta^0)$ for a deformation ϕ^0 and a residual $\zeta^0 \in L^2(X^0)$. As previously, the prior on the template is:

$$p\left(v^0, \zeta^0\right) \propto e^{-E_{X_0}(v^0, \zeta^0)^2}.$$

With an hypertemplate (X_0, f_0) fixed by the user, estimating the template then amounts to computing the maximum *a posteriori* (MAP) estimate of (v^0, ζ^0) knowing the observations $(X^\ell, f^\ell)_\ell$. With Bayes rules this leads to minimizing:

$$-\sum_{\ell=1}^{P} \log\left(\int p\left((X^\ell, f^\ell) \mid (v^0, \zeta^0), (v^\ell, \zeta^\ell)\right) p(v^\ell, \zeta^\ell)\right) - \log\left(p(v^0, \zeta^0)\right).$$

The first term involves the integral with respect to the probability distribution of the latent variables (v^ℓ, ζ^ℓ). As there is no closed form expression of this integral, we use the standard Fast Approximation with Modes and replace it by $\max_{(v^\ell, \zeta^\ell)} p\left((X^\ell, f^\ell) \mid (v^0, \zeta^0), (v^\ell, \zeta^\ell)\right) p\left(v^\ell, \zeta^\ell\right)$ leading eventually to the following variational problem:

$$\begin{aligned}\left((v^0_*, \zeta^0_*), (v^\ell_*, \zeta^\ell_*)_\ell\right) = \underset{(v^0, \zeta^0), (v^\ell, \zeta^\ell)_\ell}{\operatorname{Arginf}} & \frac{1}{2\sigma_V^2} \int_0^1 \|v^0_t\|_V^2 dt + \frac{1}{2\sigma_f^2} \|\zeta^0\|^2_{H^s(X^0)} \\ & + \sum_{\ell=1}^{P} \left(\frac{1}{2\sigma_V^2} \int_0^1 \|v^\ell_t\|_V^2 dt + \frac{1}{2\sigma_f^2} \|\zeta^\ell\|^2_{H^s(X^0)} + \frac{1}{2\sigma_W^2} \|\mu_{(\phi^\ell, \zeta^\ell) \cdot (X, f)} - \mu_{(X^\ell, f^\ell)}\|^2_{W^*} \right)\end{aligned} \tag{1.9}$$

where we remind that for $\ell = 0, \dots, P$ deformations ϕ^ℓ are the flows of the v^ℓ and $X = \phi^0(X_0)$. Note that the variances $\sigma_V^2, \sigma_f^2, \sigma_W^2$ serve as weighting coefficients between the different terms of the objective. Here, we treat these variances as algorithm parameters.

Mean template estimation

The previous paragraphs highlight the Bayesian interpretation underlying the variational problem (1.9), for which the existence of solutions under certain conditions is recalled in Section 1.4.1. In practice, computing an approximate mean template requires solving multiple concurrent registration problems. Two different atlas estimation methods are described in detail in Section 7 of [Art3]: the **hypertemplate algorithm** (discussed Section 1.4.2 and used in [Art4] and [Art5]) and the **free template algorithm**. Both approaches rely on the following two steps iterated until convergence:

1. Partially solve P registration problems from the current mean template onto the dataset.
2. Compute the gradient and update the mean template.

The first step is common to both algorithms. In the free template algorithm, the mean template is updated directly using the aggregated gradient from the P registrations. In contrast, the hypertemplate algorithm applies this same gradient through an additional registration step between a fixed hypertemplate and the mean template. This ensures that the updated mean template remains within the same fshape bundle, see Figure 1.8.

Application to OCT dataset

The cohorts of patients in the OCT dataset studies include both healthy and glaucomatous subjects. The first study [Art4] was a methodological paper aimed at demonstrating the feasibility of the proposed approach, and included $P = 53$ observations. In the second, medically oriented study [Art5], the cohort consisted of $P = 38$ acquisitions. Despite the smaller sample size, the cohort was designed with balanced sex and age distributions to ensure the statistical relevance of the conclusions. By estimating a mean template for both geometry and function, individual observations can be mapped into the coordinate system of the template. This alignment enables functional residuals $\boldsymbol{\zeta}$ — representing deviations in the functional signal — to be indexed on a common geometric domain. As a result, statistical analysis can be performed in this common space to study thickness loss patterns associated with glaucoma.

The fshape framework enabled detailed visualization of spatial patterns, offering significantly finer representations compared to classical methods that rely on low-resolution, fixed-sector views as used in [44]. The differences between age-matched normal and glaucomatous retinal nerve fiber layers are thoroughly analyzed in [Art5]. The glaucomatous layers were found to be significantly thinner, especially in the inferior region near Bruch's membrane opening. Additionally, comparisons between younger and older healthy subjects revealed a significant age-related thinning of the choroid, particularly in the nasal and inferior regions.

1.5 Numerical implementation

In the OCT dataset, data results from a complex pipeline involving image acquisition, segmentation, and surface extraction. In this context, the ideal underlying continuous functional surface (X, f) is unknown and is approximated by a textured triangular mesh.

1.5.1 Discrete fshapes

A continuous fshape (X, f) of dimension d embedded in $\mathbb{R}^n$ is assumed to be known through a finite set of $N \geq (d+1)$ points, along with their associated signal values and the connectivity relations between vertices. In the discrete setting, an fshape is therefore described by a triplet of objects $(\boldsymbol{x}, \boldsymbol{f}, C)$, where:

- $\boldsymbol{x} = (x_i)_{i=1,\dots,N}$ is a $N \times n$ matrix of the N vertex coordinates $x_i \in \mathbb{R}^n$,
- $\boldsymbol{f} = (f_i)_{i=1,\dots,N} \in \mathbb{R}^{N\times 1}$ is a column vector of signal values attached to each vertex (in Lagrangian coordinates),
- $\boldsymbol{C} \in \{1,\dots,N\}^{T\times(d+1)}$ is a $T \times (d+1)$ connectivity matrix. The mesh thus consists of $T > 0$ simplices of dimension d (line segments (for curves) or triangles (for meshed surfaces)), where the k-th row of C contains the indices of the $d+1$ vertices forming the simplex $k \in \{1,\dots,T\}$.

In direct analogy with the continuous transport equations (1.1), the transformation of a discrete fshape by a deformation $\phi : \mathbb{R}^n \to \mathbb{R}^n$ and a functional residual $\boldsymbol{\zeta} \in \mathbb{R}^{N\times 1}$ results in the discrete fshape given by:

$$(\phi, \boldsymbol{\zeta}) \cdot (\boldsymbol{x}, \boldsymbol{f}) = \big(\phi(\boldsymbol{x}), \boldsymbol{f} + \boldsymbol{\zeta}\big) = \big(\phi(x_i), f_i + \zeta_i\big)_{i=1,\dots,N},$$

with the connectivity matrix $\boldsymbol{C}$ remaining unchanged.

1.5.2 Discrete functional norms

A discrete shape $(\boldsymbol{x}, \boldsymbol{f}, \boldsymbol{C})$ is a graph equipped with a signal attached to each vertex. From this graph, we construct a piecewise polyhedral domain $\tilde{X} \subset \mathbb{R}^n$, consisting of d-dimensional simplices whose vertices and edges are encoded by the matrices $\boldsymbol{x}$ and $\boldsymbol{C}$. Let $\tilde{f} : \tilde{X} \to \mathbb{R}$ be an interpolating function such that $\tilde{f}(x_i) = f_i$ for all $i = 1,\dots,N$. In this setting, the H^s norm of $\tilde{f}$ on $\tilde{X}$ is denoted by $\|\boldsymbol{f}\|_{H^s(\boldsymbol{x})}$ — omitting the explicit dependence on $\tilde{X}$ and $\tilde{f}$ — and can be generally expressed as:

$$\|\boldsymbol{f}\|^2_{H^s(x)} = \boldsymbol{f}^\top D_s(\boldsymbol{x}) \boldsymbol{f},$$

where, $D_s(\boldsymbol{x})$ is a symmetric positive definite $N \times N$ matrix that depends on the interpolation scheme used to define $\tilde{f}$ over $\tilde{X}$. The entries of $D_s(\boldsymbol{x})$ can be computed from the data in $\boldsymbol{x}$ and $\boldsymbol{C}$, and the matrix is typically sparse, as it is derived from the adjacency structure of the mesh. In our applications, we consider Sobolev norms of order $s = 0, 1$, and the P_1, P_0, and mass lumping interpolation schemes were sufficient for our purposes — i.e., they provide a precise meaning to $\|\boldsymbol{f}\|_{L^2}$ and $\|\nabla \boldsymbol{f}\|_{L^2}$. Further details can be found in Section 4.2 of [Art6] and Section 6.2 of [Art11].

1.5.3 Data attachment term and discrete Varifold norm

Using the notations from Section 1.3.2, the reproducing kernel property implies that, for any $x \in \mathbb{R}^n$, $V \in G^n_d$, and $s \in \mathbb{R}$, all Dirac measures $\delta_{(x,V,s)}$ belong to the RKHS dual space W^* used to define the varifold norm. The corresponding dual metric satisfies:

$$\langle \delta_{(x_1,V_1,s_1)}, \delta_{(x_2,V_2,s_2)} \rangle_{W^*} = k_{\text{pos}}(x_1, x_2)\, k_{\text{or}}(V_1, V_2)\, k_{\text{sig}}(s_1, s_2). \tag{1.10}$$

This is the key element for approximating the double integral in Equation (1.6) by a double sum. However, before this, we need to define a quantization process. The basic idea is outlined here,

with detailed explanations provided in Section 6.1 of [Art3], Section 4 of [Art6], Section 6 of [Art11], and [Chap1].

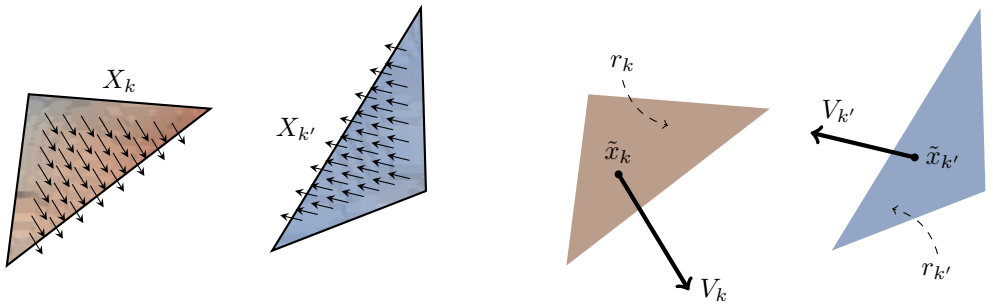
As in Section 1.5.2, a continuous fvarifold (X, f) is known through a discrete shape $(\boldsymbol{x}, \boldsymbol{f}, \boldsymbol{C})$ on which we associate a (still continuous) polyhedral object $(\tilde{X}, \tilde{f})$, which can generally be written as $(\tilde{X}, \tilde{f}) = \bigcup_{k=1}^{T}(X_k, f_k)$, where the cells X_k are distinct from each other (modulo their boundaries). To this polyhedral fshape, we associate the fvarifold $\mu_{(\tilde{X},\tilde{f})} = \sum_{k=1}^{T} \mu_{(X_k,f_k)}$, where each $\mu_{(X_k,f_k)}$ is the fvarifold associated with the flat cell X_k according to formula (1.5). As illustrated by Figure 1.9, we can make the approximation:

$$\mu_{(X_k,f_k)}(\omega) \approx \int_{X_k} \omega(\tilde{x}_k, V_k, \tilde{f}_k) d\mathcal{H}^d(x) = r_k \omega(\tilde{x}_k, V_k, \tilde{f}_k) = r_k \delta_{(\tilde{x}_k, V_k, \tilde{f}_k)}(\omega),$$

where position $\tilde{x}_k$ is the barycenter of the cell X_k, orientation $V_k \in G_d^n$ is the linear subspace spanned by X_k, the signal $\tilde{f}_k$ is the mean value on the cell of f_k, and $r_k \doteq \mathcal{H}^d(X_k)$ is the volume (or length or area) of the cell (resp. segment or triangle). This leads to replacing each $\mu_{(X_k,f_k)}$ by a single weighted Dirac $r_k \delta_{(\tilde{x}_k, V_k, \tilde{f}_k)}$. Let us define

$$\mu_{(\tilde{X},\tilde{f})} \approx \mu_{(\boldsymbol{x},\boldsymbol{f})} \doteq \sum_{k=1}^{T} r_k \delta_{(\tilde{x}_k, V_k \tilde{f}_k)}(\omega),$$

where, the center cell positions $\tilde{x}_k$, cell orientations V_k, and mean cell signals $\tilde{f}_k$ are readily computed from $\boldsymbol{x}$, $\boldsymbol{f}$, and $\boldsymbol{C}$. In the case of varifolds (no signal), Proposition 1 in [Proc2] shows that this finite sum of Dirac approximations provides an acceptable approximation of the polyhedral shape $\tilde{X}$ in terms of varifolds, as long as the size of the cells remains small enough. A general error bound, including the fshape case and the continuous (unobserved) (X, f), is given in Lemma 6.2 in [Art11].



(a) Continuous triangles with signal

(b) Discrete approximation with mean signal

Figure 1.9: An example of quantization of two triangle cells with signal.

We finally obtain a equivalent for discrete fshapes of formula (1.6) that writes:

$$\langle \mu_{(\boldsymbol{x},\boldsymbol{f})}, \mu_{(\boldsymbol{x}',\boldsymbol{f}')} \rangle_{W^*} = \sum_{k=1}^{T} \sum_{k'=1}^{T'} k_{\text{pos}}(\tilde{x}_k, \tilde{x}'_{k'})\, k_{\text{or}}(V_k, V'_{k'})\, k_{\text{sig}}(\tilde{f}_k, \tilde{f}'_{k'}) r_k r'_{k'}, \tag{1.11}$$

for the two discretizations $\tilde{\mu}_{(X,f)} = \sum_{k=1}^{T} r_k \delta_{(\tilde{x}_k, V_k, \tilde{f}_k)}$ and $\tilde{\mu}_{(Y,g)} = \sum_{k'=1}^{T'} r'_k \delta_{(\tilde{x}'_k, V'_k, \tilde{f}'_k)}$. In the initial implementation in FshapesTk [Soft1], the expressions for the fvarifold distances derived from (1.11) and for their gradients were manually coded in CUDA to scale to real data. Formulations for generic radial kernels in dimensions $d = 2, 3$ were provided with the paper [Proc2]. Today, the distance $\|\mu_X - \tilde{\mu}_X\|_{W^*}$ can be implemented for arbitrary kernels in just a few lines using KeOps [Soft3], and the gradients with respect to both the positions of the shape's vertices and the signal

can be computed automatically — without memory overflow — thanks to automatic differentiation, as described in Chapter 3.

1.5.4 Metamorphosis in the discrete setting

Discrete Hamiltonian equations

A metamorphosis is determined by a couple $(v_t, \boldsymbol{h}_t)$ with $v \in L^2([0,1], V)$ and $\boldsymbol{h}_t = (h_{i,t}) \in \mathbb{R}^{N\times 1}$ such that we have the finite-dimensional evolution equations:

$$\begin{cases} \dot{x}_{i,t} = v_t(x_{i,t}) \\ \dot{f}_{i,t} = h_{i,t} \end{cases}$$

The energy (1.3) becomes:

$$E_{\boldsymbol{x}}(v, \boldsymbol{h}) = \frac{\gamma_V}{2} \int_0^1 \|v_t\|_V^2 \, dt + \frac{\gamma_f}{2} \int_0^1 \boldsymbol{h}_t^T D_s(\boldsymbol{x}_t) \boldsymbol{h}_t dt.$$

The Hamiltonian corresponding to the minimization problem with this discrete energy also takes the form:

$$\begin{aligned} H\big(\boldsymbol{x}_t, \boldsymbol{f}_t, \boldsymbol{p}_t, \boldsymbol{p}_t^f, v, \boldsymbol{h}_t\big) &\doteq \big(\boldsymbol{p}_t \mid v_t(\boldsymbol{x})\big) + \big(\boldsymbol{p}_t^f \mid \boldsymbol{h}_t\big) - \frac{\gamma_V}{2}\big\|v_t\big\|_V^2 - \frac{\gamma_f}{2}\boldsymbol{h}_t^T D_s\big(\boldsymbol{x}_t\big)\boldsymbol{h}_t \\ &= \left\langle \sum_{i=1}^N p_{i,t}^T K_V\left(x_{i,t}, \cdot\right), v_t \right\rangle_V + \boldsymbol{h}_t^T \boldsymbol{p}_t^f - \frac{\gamma_V}{2}\big\|v_t\big\|_V^2 - \frac{\gamma_f}{2}\boldsymbol{h}_t^T D_s\big(\boldsymbol{x}_t\big)\boldsymbol{h}_t \end{aligned}$$

where $\boldsymbol{p} \in \mathbb{R}^{N\times n}$ and $\boldsymbol{p}^f \in \mathbb{R}^{N\times 1}$ are the discrete co-state variables. Denoting K_V the vector kernel associated to the RKHS V, the optimality conditions along geodesics $\partial_v H\big(\boldsymbol{x}_t, \boldsymbol{f}_t, \boldsymbol{p}_t, \boldsymbol{p}_t^f, v_t, \boldsymbol{h}_t\big) = 0$ and $\partial_{\boldsymbol{h}} H\big(\boldsymbol{x}_t, \boldsymbol{f}_t, \boldsymbol{p}_t, \boldsymbol{p}_t^f, v_t, \boldsymbol{h}_t\big) = 0$ from the Pontryagin Maximum Principle lead to the following expressions of the optimal controls:

$$\begin{cases} v_t = \frac{1}{\gamma_V} \sum\limits_{i=1}^N K_V(x_{i,t}, \cdot) p_{i,t} \\ \boldsymbol{h}_t = \frac{1}{\gamma_f} D_s^{-1}\left(\boldsymbol{x}_t\right) \boldsymbol{p}_t^f \end{cases}$$

As usual for the LDDMM model, the optimal velocity fields v_t are fully parameterized (via a convolution operation) by finite-dimensional momentum vectors $\boldsymbol{p}_t = (p_{i,t})_{i=1}^N$ associated with each vertex position. This leads to the following discrete reduced Hamiltonian:

$$H_r\big(\boldsymbol{x}_t, \boldsymbol{f}_t, \boldsymbol{p}_t, \boldsymbol{p}_t^f\big) = \frac{1}{2\gamma_V} \boldsymbol{p}_t^T K_{\boldsymbol{x}_t, \boldsymbol{x}_t} \boldsymbol{p}_t + \frac{1}{2\gamma_f}\big(\boldsymbol{p}_t^f\big)^T D_s^{-1}\left(\boldsymbol{x}_t\right) \boldsymbol{p}_t^f \tag{1.12}$$

where $\boldsymbol{p}_t^T K_{x_t, x_t} \boldsymbol{p}_t \doteq \sum_{i,j=1}^N p_{i,t}^T K_V\left(x_{i,t}, x_{j,t}\right) p_{j,t}$. In the tangential model used in [Art3] and recalled Section 1.2.3, the functional norm does not depend on the geometry evolution and second term is replaced by $\frac{1}{2\gamma_f}\big(\boldsymbol{p}_t^f\big)^T D_s^{-1}\left(\boldsymbol{x}_0\right) \boldsymbol{p}_t^f$.

Forward equations

The hamiltonian being conserve along the time, and using formula (1.12), we can derive explicit discrete Hamiltonian evolution equations[5] for the **full metamorphoses model**:

$$\begin{pmatrix} \dot{\boldsymbol{x}}_t \\ \dot{\boldsymbol{f}}_t \\ \dot{\boldsymbol{p}}_t \\ \dot{\boldsymbol{p}}_t^f \end{pmatrix} = \begin{pmatrix} \partial_{\boldsymbol{p}} H_r\big(\boldsymbol{x}, \boldsymbol{f}, \boldsymbol{p}, \boldsymbol{p}^f\big) \\ \partial_{\boldsymbol{p}^f} H_r\big(\boldsymbol{x}, \boldsymbol{f}, \boldsymbol{p}, \boldsymbol{p}^f\big) \\ -\partial_{\boldsymbol{x}} H_r\big(\boldsymbol{x}, \boldsymbol{f}, \boldsymbol{p}, \boldsymbol{p}^f\big) \\ -\partial_{\boldsymbol{f}} H_r\big(\boldsymbol{x}, \boldsymbol{f}, \boldsymbol{p}, \boldsymbol{p}^f\big) \end{pmatrix} = \begin{pmatrix} \frac{1}{\gamma_V} K_{\boldsymbol{x}_t, \boldsymbol{x}_t} \boldsymbol{p}_t \\ \frac{1}{\gamma_f} \boldsymbol{v}_t^f \\ -\frac{1}{2\gamma_V} \boldsymbol{p}_t^T \partial_{\boldsymbol{x}_t} K_{\boldsymbol{x}_t, \boldsymbol{x}_t} \boldsymbol{p}_t + \frac{1}{2\gamma_f} \big(\boldsymbol{v}^f\big)^T \partial_{\boldsymbol{x}_t} D_s\left(\boldsymbol{x}_t\right) \boldsymbol{v}^f \\ 0 \end{pmatrix}. \tag{1.13}$$

Here, $\boldsymbol{v}_t^f = D_s^{-1}(\boldsymbol{x}_t)\boldsymbol{p}^f$ is a term that requires the inversion of an $N \times N$ (sparse) matrix for its computation. We also note that the functional momentum $\boldsymbol{p}_t^f$ remains constant over time (hence the subscript t is dropped). The geometric momentum $\dot{\boldsymbol{p}}_t$ depends on the functional momentum $\boldsymbol{p}^f$, meaning that variations in the signal induce variations in the geometry as discussed Section 1.2.3. For the sake of completeness, we provide below the forward equation for the **tangential metamorphosis model** derived in [Art3]:

$$\begin{pmatrix} \dot{\boldsymbol{x}}_t \\ \dot{\boldsymbol{p}}_t \\ \boldsymbol{f}_t \end{pmatrix} = \begin{pmatrix} \frac{1}{\gamma_V} K_{\boldsymbol{x}_t, \boldsymbol{x}_t} \boldsymbol{p}_t \\ -\frac{1}{2\gamma_V} \boldsymbol{p}_t^T \partial_{\boldsymbol{x}_t} K_{\boldsymbol{x}_t, \boldsymbol{x}_t} \boldsymbol{p}_t \\ f + t\boldsymbol{\zeta} \end{pmatrix}. \tag{1.14}$$

Equations (1.13) and (1.14) provide insight into the behavior illustrated in Figure 1.4.

Geodesic shooting

We have all the ingredients to define the discrete equivalent of fshape registration Equation (1.8) between a source $(\boldsymbol{x}, \boldsymbol{f})$ and a target $(\boldsymbol{x}^{\text{tar}}, \boldsymbol{f}^{\text{tar}})$. It can then be cast as a finite dimensional, non convex, optimization problem on the initial momenta variables $\boldsymbol{p}_0 \doteq \boldsymbol{p}_{t=0} \in \mathbb{R}^{N \times n}$ and $\boldsymbol{p}^f \in \mathbb{R}^{N \times 1}$ that writes:

$$\min_{\boldsymbol{p}_0, \boldsymbol{p}^f} J\big(\boldsymbol{p}_0, \boldsymbol{p}^f\big) \doteq \frac{\gamma_V}{2} \boldsymbol{p}_0^T K_{\boldsymbol{x},\boldsymbol{x}} \boldsymbol{p}_0 + \frac{\gamma_f}{2} \big(\boldsymbol{p}^f\big)^T D_s^{-1}(\boldsymbol{x}_0) \boldsymbol{p}^f + \underbrace{\gamma_W \|\mu_{(\boldsymbol{x}_1, \boldsymbol{f}_1)} - \mu_{(\boldsymbol{x}^{\text{tar}}, \boldsymbol{f}^{\text{tar}})}\|_{W^*}^2}_{\doteq g(\boldsymbol{x}_1, \boldsymbol{f}_1)} \tag{1.15}$$

subject to the dynamics described by Equation (1.13). We use a geodesic shooting scheme for solving the minimization generalizing widely used similar frameworks in diffeomorphic shape matching [3].

In this case, the problem essentially reduces to performing a gradient descent on the initial momenta variables $(\boldsymbol{p}_0, \boldsymbol{p}^f)$. The gradients of the first two terms in Equation (1.15) are straightforward to compute. The only slightly more involved part is the last term, $g(\boldsymbol{x}_1, \boldsymbol{f}_1)$, which depends on the final states — $(\boldsymbol{x}_1, \boldsymbol{f}_1) = (\phi_1, \boldsymbol{\zeta}_1) \cdot (\boldsymbol{x}, \boldsymbol{f})$ — obtained via the forward equations (1.13). Nowadays, this can be computed directly using KeOps and automatic differentiation as explained in Chapter 3. Alternatively, it can be approached by integrating backward the so-called adjoint linearized system of equations[6]:

$$\begin{pmatrix} \dot{X}_t \\ \dot{F}_t \\ \dot{P}_t \\ \dot{P}_t^f \end{pmatrix} = \Big(-dF\big(\boldsymbol{x}_t, \boldsymbol{f}_t, \boldsymbol{p}_t, \boldsymbol{p}_t^f\big)\Big)^T \begin{pmatrix} X_t \\ F_t \\ P_t \\ P_t^f \end{pmatrix} \tag{1.16}$$

[5] The time evolution of $\big(\boldsymbol{x}, \boldsymbol{f}, \boldsymbol{p}, \boldsymbol{p}^f\big)$ is perpendicular to the gradient of H_r.

[6] The terminology is similar to that of reverse-mode automatic differentiation, though, at the time, the communities using them were distinct.

with the adjoint variables $X_t \in \mathbb{R}^{N\times n}, F_t \in \mathbb{R}^{N\times 1}, P_t \in \mathbb{R}^{N\times n}, P_t^f \in \mathbb{R}^{N\times 1}$ and the endpoint conditions $X_1 = \partial_{\boldsymbol{x}} g(\boldsymbol{x}_1, \boldsymbol{f}_1), F_1 = \partial_{\boldsymbol{f}} g(\boldsymbol{x}_1, \boldsymbol{f}_1), P_1 = \partial_{\boldsymbol{p}} g(\boldsymbol{x}_1, \boldsymbol{f}_1) = 0$ and $P_1^f = \partial_{\boldsymbol{p}} g(\boldsymbol{x}_1, \boldsymbol{f}_1) = 0$. In practice, the system of equations 1.16 is tedious to implement and we use instead the finite difference trick presented in [6] (Section 4.1 just before Proposition 9). To integrate the adjoint system (1.16), rather than explicitly compute each term in the matrix dF^T, we only need to compute a single directional derivative at each time step with a finite difference method. This has several advantages: it is rather general, it greatly simplifies the implementation and in the end amounts in about twice the computational cost of the forward system of equations (1.13).

In summary, the gradient of the objective functional with respect to $\boldsymbol{p}_0$ and $\boldsymbol{p}^f$ is obtained by the following forward-backward scheme:

1. Compute $\left(\boldsymbol{x}_t, \boldsymbol{f}_t, \boldsymbol{p}_t, \boldsymbol{p}_t^f\right)$ by integrating Equation (1.13) forward with initial conditions $\left(\boldsymbol{x}_0, \boldsymbol{f}_0, \boldsymbol{p}_0, \boldsymbol{p}_0^f\right)$.
2. Compute the gradients of $g(\boldsymbol{x}_1, \boldsymbol{f}_1)$ with respect to $\boldsymbol{f}$ and $\boldsymbol{x}$.
3. Transport the gradients to $t = 0$ by integrating backward equation 1.16 with final conditions $X_1 = \partial_{\boldsymbol{x}} g(\boldsymbol{x}_1, \boldsymbol{f}_1), F_1 = \partial_{\boldsymbol{f}} g(\boldsymbol{x}_1, \boldsymbol{f}_1), P_1 = 0, P_1^f = 0$.
4. Set $\nabla_{\boldsymbol{p}_0} J = \frac{1}{\gamma_V} K_{\boldsymbol{x},\boldsymbol{x}} \boldsymbol{p}_0 + P_0$ and $\nabla_{\boldsymbol{p}^f} J = D_0\left(\boldsymbol{x}_0\right)\left(\frac{1}{\gamma_f} D_s^{-1}\left(\boldsymbol{x}_0\right) \boldsymbol{p}^f + \gamma_W P_0^f\right)$

We point out that the gradient with respect to the functional momentum $\boldsymbol{p}^f$ at last step is computed with respect to the L^2 metric on X_0 instead of the Euclidean metric, which adds the extra weight matrix $D_0\left(\boldsymbol{x}_0\right)$. This can be crucial for example when the mesh X_0 is not regular but contains triangles of very different areas. The updates on $\boldsymbol{p}^f$ obtained from the gradient computed with respect to this metric ensures that the signal variations $\dot{\boldsymbol{f}} = D_s^{-1}(\boldsymbol{x})\boldsymbol{p}^f$ will not be too much affected by the quality of the initial mesh.

CHAPTER 2

Longitudinal Datasets and Shape Evolution

> Pour qu'une chose soit intéressante, il suffit de la regarder longtemps.
>
> G. Flaubert in [25]

Longitudinal data The work presented in this chapter was carried out at ICM from 2016 to 2019, within the Aramis team, which focuses on the design of computational, mathematical, and statistical approaches for the analysis of multimodal patient data, with a particular emphasis on neuroimaging. The team also develops various clinical applications of its research, especially in the context of neurodegenerative disorders such as Alzheimer's disease.

This chapter presents my contributions to various models used in the analysis of longitudinal data. Longitudinal refers to observations of a patient's state at multiple time points, allowing us to capture the rate of change over time. A common approach is to model the evolution of clinical markers as trajectories — sampled at discrete time points — in an abstract space, using tools from Riemannian geometry. These contributions are primarily related to the theoretical paper [Art7], the software [Soft2] and the applied studies [Proc5, Art8, Art10]. This series of works corresponds to part of the PhD thesis of M. Louis, which I co-supervised together with S. Durrleman.

Deformetrica The ICM provided a dynamic environment for developing research projects, spanning theoretical advances and practical applications in medical research. During my research stay, I had the opportunity to collaborate with skilled engineers such as A. Routier, M. Bacci, and B. Martin, from whom I learned many of the software development skills that later proved invaluable.

At Aramis, a strong emphasis was placed on implementation, and I was actively involved in the development of the Deformetrica software. Initially written in C++/CUDA [Proc1], it was later ported to Python in 2018 by PhD students A. Bône and M. Louis, who — frustrated by the intricacies of the language — chose a cleaner and more accessible approach. For example, computing gradients in registration problems originally required integrating the adjoint Hamiltonian system backward in time. Early versions of the software encoded full, explicit expressions, including tedious second-order derivative terms (as in FshapesTk). This was later simplified through a single directional derivative approximated by finite difference, as explained in Section 1.5.4. But the transition to

Python marked a major breakthrough, leveraging the PyTorch library and KeOps for automatic differentiation. This made the implementation of new methods significantly more accessible and flexible. The notable difference is that the initial method implemented a discretized version of the gradient derived from the continuous framework to optimize the discrete model, whereas automatic differentiation provides the exact gradient of the discretized problem.

Outline We begin this chapter with a study of the numerical properties of the fanning scheme algorithm for computing parallel transport on manifolds and motivate its use on some applications. Several experiments are presented on shape space to evaluate its performance. Geodesic regression and trajectory estimation in shape space using parallel transport were implemented in the software [Soft2]. Finally, we present the application paper [Art10], which analyzes the variability in Alzheimer's disease progression using multimodal data, including brain images, clinico-neuropsychological assessments, and biomarkers.

2.1 A fanning scheme to compute the parallel transport on Riemannian manifolds

2.1.1 Introduction

Many data modalities with complex structures cannot be accurately analyzed within standard Euclidean spaces. Such data often exhibit invariance properties and are better represented as points in quotient spaces — such as shape spaces, spaces of orthogonal frames, or linear subspaces — which fundamentally belong to curved, non-Euclidean spaces. Moving beyond classical statistics in Euclidean settings, and instead adopting statistical learning frameworks grounded in Riemannian geometry, has become a common and effective strategy. Computational anatomy is a prominent example of this approach.

The need for efficient algorithms to compute intrinsic quantities on high-dimensional, abstract manifolds — while operating on real, concrete data — has become increasingly pressing, especially with the growing availability of data and computational resources. As a result, software packages incorporating these rigorous mathematical developments have been released. Notable examples include Deformetrica[1] and the Geomstats library [57].

In [Art7], we focus on parallel transport and study a numerical scheme designed to scale efficiently to high-dimensional data. Parallel transport is an isometry that enables the comparison of quantities defined in the tangent spaces at different points on a manifold — such as probability density functions, coordinates, or vectors associated with distinct locations.

There are typically no tractable formulas for computing parallel transport on a given manifold. Either no closed-form solutions are available, or the computational complexity makes it impractical in real-world scenarios. Formally, parallel transport can be defined as the solution to an ordinary differential equation involving Christoffel symbols (which are computed from the manifold's metric). However, the number of these symbols explodes with the dimension of the manifold, making the approach computationally unrealistic.

Alternative methods involve approximation techniques, which can be categorized into two main families. The first family consists of ladder methods, which iterate through steps that include the construction of small geodesic parallelograms. In principle, these methods require

[1] March 2025, the project seems to be discontinued... website down.

the computationally expensive step of calculating the Riemannian logarithm at each iteration. Recently, [37] addressed this limitation by providing a convergence study for these methods. We will revisit this topic in Section 2.1.4. The second family consists of the Fanning scheme (FS) which use a well-chosen Jacobi fields to approximate parallel transport along geodesics, up to the second order error. Overall, it relies solely on the computations of Riemannian exponentials much cheaper than the logarithm.

2.1.2 Fanning scheme

Notations and assumptions

We assume that γ is a geodesic defined for all time $t > 0$ on a smooth manifold $\mathcal{M}$ of finite dimension $d \in \mathbb{N}$ provided with a smooth Riemannian metric g. We denote the Riemannian exponential Exp and ∇ the covariant derivative. For $p \in \mathcal{M}$, $T_p\mathcal{M}$ denotes the tangent space of $\mathcal{M}$ at p. For all $s, t \geq 0$ and for all $w \in T_{\gamma(s)}\mathcal{M}$, we denote $\mathrm{P}_{s,t}(w) \in T_{\gamma(t)}\mathcal{M}$ the parallel transport of w from $\gamma(s)$ to $\gamma(t)$. It is the unique solution at time t of the differential equation $\nabla_{\dot\gamma(u)} \mathrm{P}_{s,u}(w) = 0$ for $\mathrm{P}_{s,s}(w) = w$. We also denote $\mathrm{J}^w_{\gamma(t)}(h)$ the Jacobi field emerging from $\gamma(t)$ in the direction $w \in T_{\gamma(t)}\mathcal{M}$, that is

$$\mathrm{J}^w_{\gamma(t)}(h) = \frac{\partial}{\partial\varepsilon}_{\restriction_{\varepsilon=0}} \mathrm{Exp}_{\gamma(t)}(h(\dot\gamma(t) + \varepsilon w)) \in T_{\gamma(t+h)}\mathcal{M}$$

for $h \in \mathbb{R}$ small enough. It verifies the Jacobi equation

$$\nabla^2_{\dot\gamma} J^w_{\gamma(t)}(h) + R\big(J^w_{\gamma(t)}(h), \dot\gamma(h)\big)\dot\gamma(h) = 0,$$

where R is the curvature tensor. We denote $\|\cdot\|_g$ the Riemannian norm on the tangent spaces defined from the metric g, and $g_p : T_p\mathcal{M} \times T_p\mathcal{M} \to \mathbb{R}$ the metric at any $p \in \mathcal{M}$.

We describe here a way to compute an approximation of $P_{0,1}(w)$.

Rationale

The fanning scheme is based on the following identity from [81] with further credits to [78] For all $t > 0$, and $w \in T_{\gamma(0)}\mathcal{M}$ we have

$$\mathrm{P}_{0,t}(w) = \frac{\mathrm{J}^w_{\gamma(0)}(t)}{t} + \mathrm{O}\big(t^2\big). \tag{2.1}$$

It is illustrated by Figure 2.1. This control on the approximation of the transport by a Jacobi field suggests dividing $[0, 1]$ into n intervals $[\frac{k}{n}, \frac{k+1}{n}]$ of length $h = \frac{1}{n}$ for $k = 0, \ldots, n-1$ and to approximate the parallel transport of a vector $w \in T_{\gamma(0)}$ from $\gamma(0)$ to $\gamma(1)$ by a sequence of vectors $w_k \in T_{\gamma\left(\frac{k}{n}\right)}\mathcal{M}$ defined as

$$\begin{cases} w_0 = w \\ w_{k+1} = n\, \mathrm{J}^{w_k}_{\gamma\left(\frac{k}{n}\right)}\left(\frac{1}{n}\right). \end{cases} \tag{2.2}$$

Equation (2.1) tells us that we can expect an error of order $\mathrm{O}\big(\frac{1}{n^2}\big)$ at each step and hence a speed of convergence in $\mathrm{O}\big(\frac{1}{n}\big)$ overall. There are manifolds for which the approximation of the parallel transport by a Jacobi field is exact e.g. Euclidean space, but in the general case, one cannot expect to get a better convergence rate.

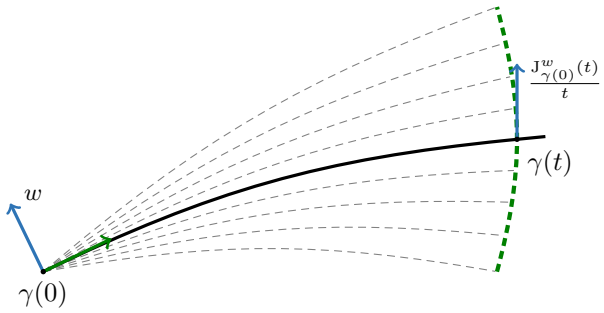


Figure 2.1: The solid line depicts the geodesic. The green dotted line is formed by the perturbed geodesics at time t. The blue arrows are the initial vector and its approximated parallel transport at time t.

Algorithm

In general, there are no closed form expressions for the geodesics and the Jacobi fields. Hence, in most practical cases, these quantities also need to be computed using numerical methods.

Computing geodesics In order to avoid the computation of the Christoffel symbols, we propose to integrate the first-order Hamiltonian equations to compute geodesics. Let $x(t) = (x_1(t), \dots, x_d(t))^T$ be the coordinates of $\gamma(t)$ in a given local chart, and $\alpha(t) = (\alpha_1(t), \dots, \alpha_d(t))^T$ be the coordinates of the momentum $g_{\gamma(t)}(\dot{\gamma}(t), \cdot) \in T^*_{\gamma(t)}\mathcal{M}$ in the same local chart. We have then (see [82])

$$\begin{cases} \dot{x}(t) = K(x(t))\alpha(t) \\ \dot{\alpha}(t) = -\frac{1}{2}\nabla_x \left(\alpha(t)^T K(x(t))\alpha(t)\right) \end{cases}, \tag{2.3}$$

where $K(x(t))$, a d-by-d matrix, is the inverse of the metric g expressed in the local chart. Note that using (2.3) to integrate the geodesic equation will require us to convert initial tangent vectors into initial momenta, as seen in the algorithm description below. This point is crucial because, in one of our targeted usecase, d can be large, see 2.2.2.

Computing $\mathrm{J}^w_{\gamma(t)}(h)$ The Jacobi field may be approximated with a numerical differentiation from the computation of a perturbed geodesic with initial position $\gamma(t)$ and initial velocity $\dot{\gamma}(t) + \varepsilon w$ where ε is a small parameter

$$\mathrm{J}^w_{\gamma(t)}(h) \simeq \frac{\mathrm{Exp}_{\gamma(t)}\big(h(\dot{\gamma}(t) + \varepsilon w)\big) - \mathrm{Exp}_{\gamma(t)}\big(h\dot{\gamma}(t)\big)}{\varepsilon}, \tag{2.4}$$

where the Riemannian exponential may be computed by integration of the Hamiltonian equations (2.3) over the time interval $[t, t+h]$ starting at point $\gamma(t)$, as shown on Figure 2.2. We will also see that a choice for ε ensuring a $\mathrm{O}\big(\frac{1}{n}\big)$ order of convergence is $\varepsilon = \frac{1}{n}$.

The algorithm Let $n \in \mathbb{N}$. We divide $[0,1]$ into n intervals $[t_k, t_{k+1}]$ with $t_k = \frac{k}{n}$ and denote $h = \frac{1}{n}$ the size of the integration step. We initialize $\gamma_0 = \gamma(0)$, $\dot{\gamma}_0 = \dot{\gamma}(0)$, $\tilde{w}_0 = w$ and solve $\tilde{\beta}_0 = K^{-1}(\gamma_0)\tilde{w}_0$ and $\tilde{\alpha}_0 = K^{-1}(\gamma_0)\dot{\gamma}_0$. We propose to compute, at step k:

1. The new point $\tilde{\gamma}_{k+1}$ and momentum $\tilde{\alpha}_{k+1}$ of the main geodesic, by performing one step of length h of a second-order Runge-Kutta method on equation (2.3).

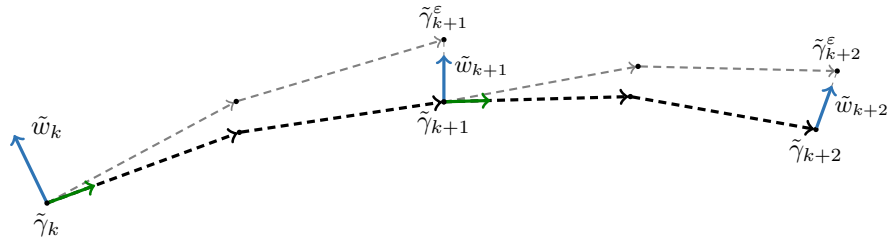


Figure 2.2: Two steps of the numerical scheme are illustrated. The dotted arrows represent the Runge-Kutta integration steps for the main geodesic γ (in black) and the perturbed geodesic γ^{ε} (in gray). The blue arrows indicate the approximated parallel transport of the vector w.

2. The perturbed geodesic starting at $\tilde{\gamma}_k$ with initial momentum $\tilde{\alpha}_k + \varepsilon\tilde{\beta}_k$ at time h, that we denote $\tilde{\gamma}^{\varepsilon}_{k+1}$, by performing one step of length h of a second-order Runge-Kutta method on equation (2.3).

3. Compute
$$\hat{w}_{k+1} = \frac{\tilde{\gamma}^{\varepsilon}_{k+1} - \tilde{\gamma}_{k+1}}{h\varepsilon}. \tag{2.5}$$
The estimated parallel transport can be set to $\tilde{w}_{k+1} = \hat{w}_{k+1}$, or we can apply an extra normalization step as described below.

4. The corresponding momentum $\hat{\beta}_{k+1}$, by solving: $K(\tilde{\gamma}_{k+1})\hat{\beta}_{k+1} = \hat{w}_{k+1}$.

At the end of the scheme, $\tilde{w}_N$ is the proposed approximation of $P_{0,1}(w)$. Figure 2.2 illustrates the principle. It is remarkable that we can substitute the computation of the Jacobi field with only four calls to the Hamiltonian equations (2.3) at each step, including the calls necessary to compute the main geodesic. Note however that the step (4) of the algorithm requires to solve a linear system of size d. Solving the linear system can be done with a complexity less than cubic in the dimension (in $\mathrm{O}(d^{2.374})$ using the Coppersmith–Winograd algorithm).

Possible variations There are a few possible variations of the presented algorithm.

1. The first variation is to use higher-order Runge-Kutta methods to integrate the geodesic equations at step (1) and (2). We prove that a second-order integration of the geodesic equation is enough to guarantee convergence and notice experimentally the absence of convergence with a first order integration of the geodesic equation. Experiments indicate a linear convergence with an improved constant using this variation. Depending on the situation, the extra computations required at each step may be counterbalanced by this increased precision.

2. The second variation uses a higher-order finite difference scheme by replacing step (2) and step (3) the following way. At the k-th iteration, compute two perturbed geodesics starting at $\tilde{\gamma}_k$ and with initial momentum $\tilde{\alpha}_k + \varepsilon\tilde{\beta}_k$ (resp. $\tilde{\alpha}_k - \varepsilon\tilde{\beta}_k$) at time h, that we denote $\tilde{\gamma}^{+\varepsilon}_{k+1}$ (resp. $\tilde{\gamma}^{-\varepsilon}_{k+1}$), by performing one step of length h of a second-order Runge-Kutta method on equation (2.3). Then proceed to a second-order differentiation to approximate the Jacobi field, and set:
$$\hat{w}_{k+1} = \frac{\tilde{\gamma}^{+\varepsilon}_{k+1} - \tilde{\gamma}^{-\varepsilon}_{k+1}}{2h\varepsilon}. \tag{2.6}$$

Empirically, this variation does not seem to bring any substantial improvement to the scheme.

3. The final variation of the scheme consists in adding an extra renormalization step at the end of each iteration:

   (5) Renormalize the momentum and the corresponding vector using

$$\tilde{\beta}_{k+1} = a_k \hat{\beta}_{k+1} + b_k \tilde{\alpha}_{k+1}$$
$$\tilde{w}_{k+1} = K(\tilde{\gamma}_{k+1})\tilde{\beta}_{k+1}$$

   where a_k and b_k are factors ensuring $\tilde{\beta}_{k+1}^\top K(\tilde{\gamma}_{k+1})\tilde{\beta}_{k+1} = \beta_0^\top K(\gamma_0)\beta_0$ and $\tilde{\beta}_{k+1}^\top K(\tilde{\gamma}_{k+1})\tilde{\alpha}_{k+1} = \beta_0^\top K(\gamma_0)\alpha_0$. Indeed, the quantities $\beta(t)^\top K(\gamma(t))\beta(t)$ and $\beta(t)^\top K(\gamma(t))\alpha(t)$ are preserved along the parallel transport. This extra step is cheap even when the dimension is large. Empirically, it leads to the same rate of convergence with a smaller constant.

The proposed algorithm and the variations 1 and 2 ensure convergence of the final estimate. Convergence of the variation 3 has not been demonstrated. But this additional step can be expected to improve the quality of the approximation at each step, at least when the discretization is sufficiently thin, by enforcing the conversation of quantities which should be conserved. Note that the best accuracy for a given computational cost is not necessarily obtained with the method in Section 2.1.2, but might be attained with one of the proposed variations, as a bit more computations at each step may be counter-balanced by a smaller constant in the convergence rate.

2.1.3 Convergence result

We obtain the following convergence result, guaranteeing a linear decrease of the error with the size of the step h.

Theorem 1 *We suppose here the hypotheses stated in Section 2.1.2. Let $n \in \mathbb{N}$ be the number of integration steps. Let $w \in T_{\gamma(0)}\mathcal{M}$ be the vector to be transported. We denote the error*

$$\delta_k = \|P_{0,t_k}(w) - \tilde{w}_k\|_2$$

where $\tilde{w}_k$ is the approximate value of the parallel transport of w along γ at time t_k and where the 2-norm is taken in the coordinates of the chart Φ on Ω. We denote ε the parameter used in the step (2) and $h = \frac{1}{n}$ the size of the step used for the Runge-Kutta approximate solution of the geodesic equation.
If we take $\varepsilon = h$, then we have

$$\delta_N = O\Big(\frac{1}{n}\Big).$$

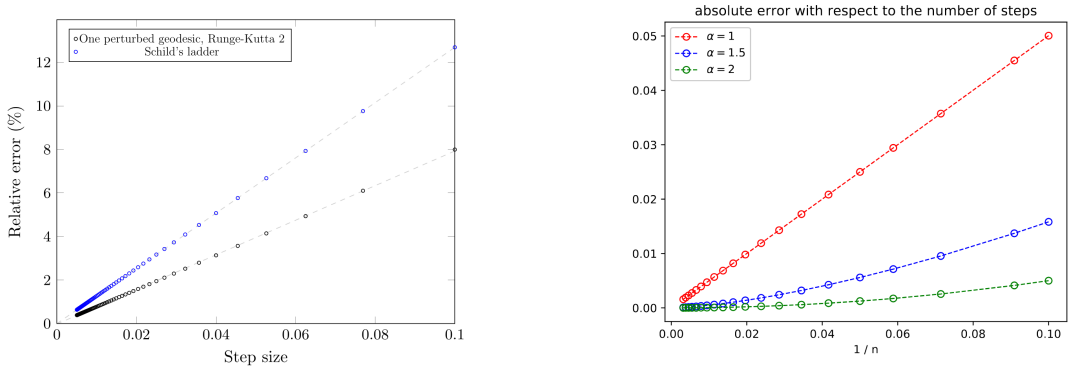
It is shown in [Art7] that taking $\varepsilon = h$ is a recommended choice for the size of the step in the differentiation of the perturbed geodesics. Further decreasing ε has no noticeable impact on estimation accuracy, while increasing ε degrades the quality of the approximation.

2.1.4 Ladder Methods Strike Back

The ladder construction process was first introduced in [58] and later formalized as the Schild Ladder (SL) in [43]. A variant known as the Pole Ladder (PL) was subsequently proposed in [48]. However, a rigorous theoretical convergence analysis for these methods was lacking until the recent work of [37], which also provided this valuable historical clarification.

A key result is that ladder methods are, in fact, second-order accurate — that is, their error decreases quadratically with step size — when implemented with a fourth-order Runge-Kutta (RK4) time integrator and an appropriate scaling/rescaling step (see Formula 6 in [37]), involving the parameter α, which serves to properly "flatten" the geodesic parallelogram.

In contrast, the Fanning Scheme (FS) exhibits first-order convergence (i.e., linear decrease in error with step size, as shown in Theorem 1) and requires only a second-order Runge-Kutta (RK2) scheme for time integration. In [Art7], SL was found to converge only linearly, which can potentially be explained[2] by the absence of the scaling/rescaling step and the use of an RK2 scheme in the implementation. See the Figure 5 below.



(a) Figure 5 of [Art7]

(b) Figure 5 of [37]

Figure 5[3] – Error rates of various parallel transport approximation schemes on the unit sphere $\mathbb{S}^2$. (a) illustrates a suboptimal, linear convergence rate for the Schild Ladder method in a naive implementation (blue curve). (b) shows that a more refined implementation of SL achieves a quadratic convergence rate (green curve).

One of the main limitations of ladder schemes is that they essentially require the computation of Riemannian logarithms at each step. Recall that computing the exponential map can be done by integrating Hamiltonian equations (also required in the FS), as described in Section 2.1.2. In contrast, computing the logarithm is often significantly more costly, as it typically involves solving an inverse problem of the form $\mathrm{Exp} \circ \mathrm{Log}(x) = x$ using an optimization procedure.

To address this, [37] noted in their numerical experiments on low-dimensional examples that only a few gradient steps are often sufficient to compute the logarithm[4]. They also studied a variant of the Pole Ladder that uses only an approximation of the logarithm and showed that this does not harm the convergence rate of the scheme. In conclusion, ladder schemes may remain competitive despite the need for logarithm computations [36], thanks to their quadratic convergence properties.

[2] These are, admittedly, assumptions. I do not have access to the implementation used in [Art7], and the codebase from [37] relies on outdated Python libraries and no longer runs out of the box. A good reminder of the importance of reproducible science.

[3] How might it be named differently?

[4] In the case of shape spaces, computing the logarithm corresponds to solving a registration problem. Since the steps are small, this observation may hold in this context as well.

Method	Error	Step size	Integrator	Operations
FS	$1/n$	$1/n$	rk2	Exp
SL/PL	$1/n^2$	$1/\sqrt{n}$	rk4	Exp and Log
infinitesimal PL	$1/n^2$	$1/\sqrt{n}$	rk4	Exp

Table 2.1: Recap of the various approximation methods for parallel transport. FS stands for Fanning Scheme, SL for Schild Ladder, and PL for Pole Ladder.

2.2 Study of longitudinal datasets

2.2.1 Context and Application Setting

The primary pathological developments of neurodegenerative diseases such as Alzheimer's are believed to begin long before the first symptoms of cognitive decline appear. The ability to detect, monitor, or predict changes in radiological and clinical signs is therefore crucial for estimating an individual's stage of disease progression, selecting appropriate patients, and defining endpoints in clinical trials. The methodological development of parallel transport computation presented in Section 2.1 was specifically motivated by the need to analyze multimodal longitudinal datasets. A representative example is the Alzheimer's Disease Neuroimaging Initiative (ADNI) [39] database, which provides multimodal follow-up data from a large cohort of patients.

When longitudinal data is limited, a useful strategy is to transfer knowledge from another subject with a longer observation period. This requires a spatio-temporal matching method to align shape trajectories across individuals [54, 80]. Such ideas have already been explored in the computational anatomy community, where parallel transport in diffeomorphism groups has been used to infer future deformations from baseline alignments [47, 72]. Building on this, the quantitative study [Proc5] and the application paper [Art10] model the evolution of an object of interest as a trajectory in a Riemannian manifold, derived from the deformation path of a mean template object — within shape space and in a multimodal setting, respectively. The proposed approach relies on transporting geodesic paths from a reference point to the observed data, using a technique known as **Exp-parallelization** [69, 70]. This method allows for estimating the object's state at arbitrary time points through interpolation or extrapolation.

Analyzing non-Euclean data in low-dimensional model spaces, such as Euclidean spheres [51] or matrix subgroups [63] already involve advanced specific tools. Instantiating computational statistical methods to high-dimensional shape spaces based on real observations is an even more delicate endeavor. The first challenge is to scale these methods to high-dimensional data, in order to manage the computational complexity of the associated regression models. The second challenge lies in dealing with real-world data, which often suffer from significant acquisition noise — typically introduced by both the imaging process and the complex preprocessing pipeline (segmentation, meshing, etc.) — as well as from small sample sizes or missing data, as Figure 2.4 illustrates.

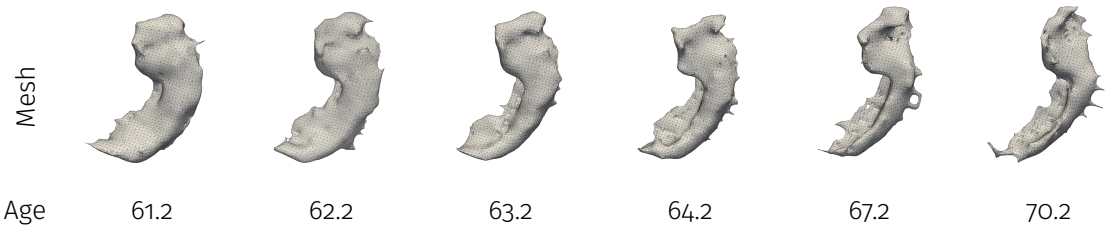


Figure 2.4: Evolution along the time of the (left) hippocampus of a patient extracted (after segmentation and meshing) from ADNI MRI data bank.

A third challenge lies in the limitations of current models to fully capture the complexity of observed dynamics. In particular, the geodesic nature of mean template trajectories imposes a constant-speed evolution, which may not align with the biological variability seen across subjects. As a result, the statistical significance of simple regression models applied to individuals with differing progression patterns may be limited. In [Proc5], introducing an additional time reparametrization to adapt the speed of evolution for each subject have improved model performance. However, the constraint of geodesic evolution may still lack biological plausibility, especially in capturing non-linear phenomena such as cycles of remission and aggravation, much like how linear models fail to represent periodic behavior. To overcome these limitations, more flexible dynamical models have been proposed, including piecewise-geodesic evolutions, as explored in [15].

In addition, the choice of metric is critical for model performance. In shape analysis, standard kernel metrics — such as those described in Chapter 1 — are insensitive to the mechanical properties of the deformed material and may fail to reproduce even simple observed deformation patterns. Designing constrained [5] or structured deformations [Proc9] remains an active area of research [PreP1].

2.2.2 Geodesic regression in shape analysis

The aim of paper [Proc5] is to predict subject-specific longitudinal progression of brain structures derived from baseline MRI scans. The estimation was carried out for $P = 74$ patients, each with a sequence of typically 9 observations. The proposed methodology is based on three key components: (1) extrapolating from a subject's own past progression; (2) transferring the progression pattern from a reference subject with a longer observation window to new subjects; and (3) refining this transfer using information on relative disease dynamics, as inferred from cognitive assessments.

As discussed in Chapter 1, shape spaces in computational anatomy are often modeled via the action of diffeomorphism groups. In this setting, geodesic regression estimates a subject-specific flow of diffeomorphisms that best fits repeated shape observations over time [46, 65], effectively generalizing linear regression to Riemannian manifolds. However, empirical studies indicate that accurate prediction of future shape evolution typically requires a large number of time points. In contrast, performance tends to be limited in scenarios with only a few observations — a more frequent and clinically realistic situation.

One of the key components of the method is the parallel transport of the deformation field along the geodesic. To perform this computation, we employ the fanning scheme described in Section 2.1, which is well-suited for high-dimensional settings. The main computational bottleneck lies in the inversion step (4). In practice, the method remains tractable for deformations involving

up to a few thousand of control points in a 3D ambient space.

Deformation model

Let $(\boldsymbol{y}_j)_{j=1,\dots,n_\ell}$ denote a time series of segmented surface meshes for a given subject $\ell \in \{1,\dots,P\}$, acquired at the corresponding ages $(t_j)_{j=1,\dots,n_\ell}$. We consider a finite-dimensional family of diffeomorphisms of the ambient space acting on the segmented meshes, following the procedure described in [21], which is conceptually similar to the approach presented in Section 1.5.4, but based on a framework involving **control points**. Specifically, flows of diffeomorphisms in $\mathbb{R}^3$ are generated by integrating time-varying vector fields of the form $v_t(x) = \sum_{i=1}^{N_{cp}} K(x, c_{i,t})\beta_{i,t}$ where K is a Gaussian kernel, $\boldsymbol{c}_t = \big(c_{i,t}\big)_{i=1,\dots,N_{cp}}$ and $\boldsymbol{\beta}_t = \big(\beta_{i,t}\big)_{i=1,\dots,N_{cp}}$ denote, respectively, the control points and the associated momenta of the deformation at time t.

We endow the space of diffeomorphisms with a metric that quantifies the cost associated with a deformation. In what follows, we focus exclusively on geodesic flows of diffeomorphisms — i.e., flows that minimize the deformation energy between the identity and a target diffeomorphism. Such flows are uniquely determined by their initial control points and momenta, denoted $\boldsymbol{c}_0, \boldsymbol{\beta}_0 \in \mathbb{R}^{N_{cp}\times 3}$. Under the action of this flow, an initial template shape X_* is continuously deformed, tracing a trajectory in shape space denoted by $t \mapsto \phi_t^{(\boldsymbol{c}_0,\boldsymbol{\beta}_0)}(X_*)$. In parallel, following the approach in Section 1.5.3, we equip the surface meshes with a varifold norm $\|\cdot\|_{W^*}$, which enables the definition of a data attachment term between shapes without requiring pointwise correspondence.

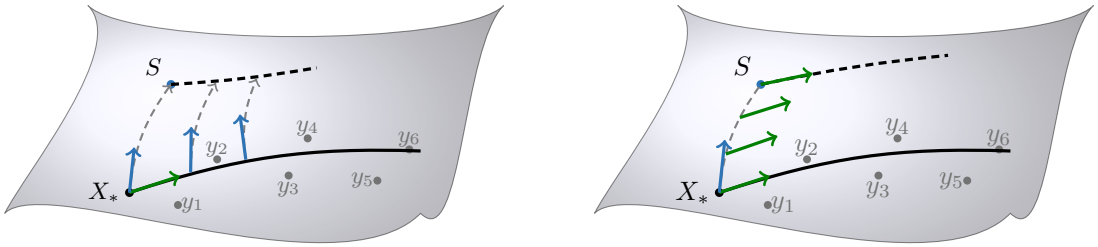
Geodesic regression and Exp-parallelization

Analogous to linear regression, one can perform **geodesic regression** in shape space by estimating an "intercept" X_* and a "slope" $(\boldsymbol{c}_0, \boldsymbol{\beta}_0)$ such that the trajectory $t \mapsto \phi_t^{(\boldsymbol{c}_0,\boldsymbol{\beta}_0)}(X_*)$ minimizes the following objective functional:

$$\inf_{\boldsymbol{c},\boldsymbol{\beta},X} \sum_{j=1}^{n_\ell} \left\| \phi_{t_j}^{(\boldsymbol{c},\boldsymbol{\beta})}(X) - y_j \right\|_{W^*}^2 + \boldsymbol{\beta}^T K_{\boldsymbol{c},\boldsymbol{c}} \boldsymbol{\beta}. \tag{2.7}$$

We estimate a solution to Equation (2.7) using a Nesterov gradient descent algorithm, as implemented in the Deformetrica software. The gradient with respect to the control points, the momenta, and the template is computed via backward integration of the data attachment term along the geodesic — an optimization procedure similar to that described in Section 1.5. Once an optimum is reached, this yields a representation of the progression of brain structures, expressed in the tangent space at the identity of the diffeomorphism group.

Given a reference geodesic, we use Riemannian parallel transport to generate a new trajectory starting from a subject-specific baseline point S. This begins with baseline matching between the template X_* and S, represented as a vector in the tangent space of the diffeomorphism group — illustrated by a blue arrow in Figure 2.5. Several methods have been proposed to generate the corresponding evolution from the subject's baseline S. The simplest approach consists of parallel transporting the initial velocity of the reference geodesic and then computing a new geodesic from S using this transported velocity (a method known as **geodesic parallel**, shown in Figure 2.5b). Another method, known as **Exp-parallelization** transports the matching vector along the reference geodesic and then reconstructs a trajectory from each point on the reference geodesic using the transported vector, as illustrated in Figure 2.5a.



(a) Exp-parallelization. Green arrow: geodesic regression. Blue arrows: transported baseline matching. Black dotted line: exp-parallelization of the reference geodesic for the given subject (not a geodesic).

(b) Geodesic parallelization. Blue arrow: baseline matching. Green arrows: transported regression. Black dotted line: exponentiation of the transported regression (a geodesic).

Figure 2.5: Two methods for transporting the geodesic regression path (black curve) fitted to the data points (gray dots).

The two paradigms for transporting parallel trajectories were shown to perform similarly well in this prediction task. However, the exp-parallelization method offers a methodological advantage: the generated trajectories do not depend on the choice of a specific point along the reference geodesic, unlike those obtained via geodesic parallelization.

Reparametrization of the geodesic

The protocol described in the previous section presents two main limitations. First, the choice of the matching time along the reference trajectory is arbitrary: the baseline is selected for convenience, but ideally, matching should occur at comparable stages of disease progression. Second, the method does not account for individual variation in the pace of progression. To address these issues, [70] introduces a statistical model that learns, in an unsupervised fashion, subject-specific dynamical parameters from ADAS-Cog scores — a standardized cognitive test used to monitor disease progression. Specifically, the model assumes that each patient follows a trajectory parallel to the mean progression, but with a subject-specific time reparametrization:

$$\psi(t) = \alpha\,(t - t_0 - \tau) + t_0 \tag{2.8}$$

which maps individual subject time to a normalized time frame using two scalar parameters, $\alpha > 0$ and τ. A high (respectively low) value of α corresponds to a faster (respectively slower) progression of the scores, while a negative (respectively positive) value of τ indicates an earlier (respectively later) onset of cognitive decline. In the ADNI dataset used in this work, the estimated acceleration factors $(\alpha_\ell)_\ell$ range from 0.15 to 6.01, and the time shifts $(\tau_\ell)_\ell$ span from -20.6 to 22.8. This wide variability highlights the heterogeneity in individual disease dynamics and underlines the importance of accounting for these effects. Using these dynamic parameters, the shape evolution can be individualized by reparametrizing the parallel trajectory via the same formula (2.8).

Discussion

Obtaining a reliable estimate of the geodesic regression requires multiple observations per subject; in this study, only subjects with 7 to 12 observations were included. Despite this, the quantitative study on geodesic regression extrapolation in [Proc5] revealed limited predictive accuracy. To

address this, we proposed a method to transport spatiotemporal trajectories between subjects by incorporating cognitive decline and derived time reparametrization, which demonstrated slightly improved predictive performance. These results highlight the crucial role of dynamic modeling in disease progression and the value of leveraging cross-modality data to enhance learning algorithms. The methods developed were further applied in the more ambitious applications [Art10].

2.2.3 Multimodal longitudinal data analysis

Alzheimer's disease (AD[5]) is characterized by progressive alterations observable in brain imaging, which are associated with the emergence of various symptoms. Nevertheless, the variability in the temporal dynamics of both neuroimaging markers and cognitive decline remains poorly understood. The study [Art10] introduces AD Course Map, a spatiotemporal atlas of Alzheimer's disease progression. It captures and summarizes inter-individual variability in the trajectories of neuropsychological assessments, the propagation of hypometabolism and cortical thinning across brain regions, and the morphological deformation of the hippocampus. The methods employed represent a refinement and generalization of the framework described in Section 2.2.2, extended to encompass multiple biomarkers. These methods were scaled to handle a large patient cohort, requiring significant implementation effort.

The analysis of these variations revealed strong genetic determinants of disease progression, as well as evidence of potential compensatory mechanisms active during the course of the disease. AD Course Map also outperformed competing approaches in predicting cognitive decline in the TADPOLE open challenge [52]. Its predictive performance has since been further validated on an independent dataset [50]. Results can be visualized in `https://project.inria.fr/digitalbrain/`.

[5]Not to be confused with Automatic Differentiation, as used in another section of this thesis.

CHAPTER 3

Kernel Methods in Action: General Formulas and Real-World Scale Datasets

KeOps by Uderzo and Goscini in [30]

This chapter outlines the principles and main functionalities of the Kernel Operations (KeOps) library, as described in [Art9, Proc6], [Proc8] and the official documentation at www.kernel-operations.io. The project is developed by a core team composed of B. Charlier, J. Feydy, and J. Glaunès. KeOps is now a mature library and serves as a dependency for various projects across diverse fields such as machine learning, statistics, optimal transport, chemistry, and physics.

KeOps builders Everything started in the late 2000s with a set of CUDA routines developed by J. Glaunès. The goal was to accelerate Matlab scripts involving Gaussian convolutions and their derivatives, as used in the LDDMM algorithms and their refinements. These independent CUDA files were compiled with Matlab binders (C/mex files) into various static libraries. Starting in 2012, I began adapting these files to match the formulas required for the Matlab FshapesTk package [Soft1], which I was starting developing during my postdoc at CMLA. In 2016, I integrated these CUDA codes into the C++ project Deformetrica [Soft2], repeatedly performing similar adaptations as I had done for FshapeTk. At some point, the idea of creating a truly versatile library imposed

itself naturally. We began discussing it with J. Glaunès, and soon after, J. Feydy — who had just started his PhD — joined the effort. The results have been far more successful than we initially planned.

KeOps mythology Version 1 (released in 2019) of the KeOps code generation engine was an original idea by J. Glaunès: it used C++ variadic templates as an abstract syntax tree to generate CUDA code. While this approach worked remarkably well, it came with significant drawbacks. First, compilation times grew substantially for complex formulas. Second, the system relied on resolving quantities at compile time — something only made possible by modern C++ features that were not fully compatible with CUDA at the time. Finally, the compilation stack was hard to set up across the variety of user configurations. Despite this, we managed to provide a functional and versatile API: a C++ interface (thanks to F.-D. Collin), along with Matlab, Python, and R bindings (with the R interface developed by G. Durif).

Completely refactored and rewritten for its version 2 release in 2022, the KeOps engine is now implemented purely in Python. The actively supported backends now focus on Python and R where most usage are. Thanks to significant improvements, formula compilations are now fast enough to be performed Just-In-Time (JIT), making the system highly flexible and responsive. In addition, extensive documentation has been developed to support users.

KeOps continues to be actively maintained and improved — driven by our ongoing need for efficient kernel operations in developing new methods, particularly in shape analysis (e.g., deformation models). Recently, KeOps was awarded with a *Prix de la science ouverte* in the Documentation category, recognizing its quality and accessibility[1].

3.1 Kernel Operation as Tensor reduction

3.1.1 Tensor reduction

In the following, $b, B, M, N, d, D \in \mathbb{N}$ denote dimensions. Bold symbols represent multi-indices; for example, $\boldsymbol{d} = (d_1, \dots, d_D)$ and $|\boldsymbol{d}| = d_1 + \cdots + d_D$. We consider real-valued tensors of the form

$$\eta_{\boldsymbol{\ell},i,j,\boldsymbol{k}} = \eta_{\ell_1,\dots,\ell_b,i,j,k_1,\dots,k_d} \in \mathbb{R}^{B_1 \times \cdots \times B_b} \times \mathbb{R}^{M \times N} \times \mathbb{R}^{D_1 \times \cdots \times D_d}$$

where the first indices $\boldsymbol{\ell} = (\ell_1, \dots, \ell_b)$ correspond to the so-called **batch** dimensions, i and j are the **outer** dimensions, and $\boldsymbol{k} = (k_1, \dots, k_d)$ represent the **inner** dimensions. The quantity $D \doteq |\boldsymbol{D}| = D_1 + \cdots + D_d$ denotes the inner space dimension. We define a **reduction** as the application of a commutative binary operation $* : E \times E \to E$ to one of the outer dimensions:

$$\gamma_{\boldsymbol{\ell},j,\boldsymbol{k}} = \bigstar_{i=1}^{M} \eta_{\boldsymbol{\ell},i,j,\boldsymbol{k}} \quad \text{or} \quad \gamma_{\boldsymbol{\ell},i,\boldsymbol{k}} = \bigstar_{j=1}^{N} \eta_{\boldsymbol{\ell},i,j,\boldsymbol{k}} \tag{3.1}$$

We emphasize here that the dimensions indexed by i and j should be regarded as the large ones and thus require special treatment. This formulation is still too generic, so we focus on problems where η can, in fact, be computed from tensors containing at most one of the indices i or j. Assume there exists an interaction map represented by a closed-form formula F.

$$\eta_{\boldsymbol{\ell},i,j,\boldsymbol{k}} = F(\underbrace{p^1_{\boldsymbol{\ell},\boldsymbol{k}}, \dots, p^P_{\boldsymbol{\ell},\boldsymbol{k}}}_{\text{parameters}}, \underbrace{x^1_{\boldsymbol{\ell},i,\boldsymbol{k}}, \dots, x^X_{\boldsymbol{\ell},i,\boldsymbol{k}}}_{i\text{-variables}}, \underbrace{y^1_{\boldsymbol{\ell},j,\boldsymbol{k}}, \dots, y^Y_{\boldsymbol{\ell},j,\boldsymbol{k}}}_{j\text{-variables}}) \tag{3.2}$$

[1] See https://www.ouvrirlascience.fr/remise-des-prix-science-ouverte-du-logiciel-libre-de-la-recherche-2023/

where P, X, and $Y \in \mathbb{N}$ denote the number of each type of data tensor required to compute the formula. This contraction formula pattern encompasses many common operations. In particular, it includes the case where two sets of particles (data) in a D-dimensional space (features) are considered — one point cloud indexed by i, the other by j — and their **interactions** are described by a kernel function given in closed form by the formula F. In the next section, we recall three examples that served as motivation for this work.

3.1.2 Examples

As a common notation, let $x_1, \dots, x_M$ and $y_1, \dots, y_N$ denote two sets of points in a common vector space $\mathbb{R}^D$.

Convolution

Let $b_1, \dots, b_N \in \mathbb{R}^D$. We compute, for every $i = 1, \dots, M$ and $k = 1, \dots, D$,

$$\gamma_{i,k} = \sum_{j=1}^{N} k(x_i, y_j) b_{j,k}.$$

Typical use cases involve M and N ranging from hundreds to billions, with $D = 2$ or 3, and $k(x, y)$ being a radial Gaussian kernel or one of its first or second derivatives, with bandwidth $\sigma > 0$. This operation is commonly interpreted as a matrix-vector product involving the Gaussian kernel matrix $(\boldsymbol{K}_{i,j}) = (k(x_i, y_j))_{i,j}$, or equivalently, as a convolution of the kernel function k sampled over the point clouds $(x_i)_{i=1,\dots,M}$ and $(y_j)_{j=1,\dots,N}$.

Nearest neighbor search

Assume that $\mathbb{R}^D$ is endowed with a distance d whose values can be computed as a function of the point coordinates (e.g., distances derived from norms, cosine similarity, hyperbolic geometry, etc.). We consider

$$\nu_i = \min_{j=1,\dots,N} d(x_i, y_j)$$

as the closest-point distance between x_i and all the y_j's. In real-world applications, both N and M may reach into the billions. The dimension D can vary widely depending on the domain: from 2 or 3 in graphics applications to over 1000 in machine learning, where the ambient space represents a set of features.

Sinkhorn algorithm

Define $C_{i,j} = \frac{1}{2}\|x_i - y_j\|^2$ as the squared Euclidean norm in $\mathbb{R}^D$, and let $(\alpha_i)_{i=1,\dots,M}$ and $(\beta_j)_{j=1,\dots,N}$ be non-negative weights summing to one. The Sinkhorn algorithm is an iterative method, and its variants are applied in many fields. This algorithm is nearly a hundred years old [19], and it has more recently been successfully used to compute approximate solutions to the Optimal Transport problem with entropic regularization [16]. The goal is to compute two potentials, $f_i \in \mathbb{R}^M$ and $g_j \in \mathbb{R}^N$, of interest, as follows. Begin by initializing $f_i = 0$ and $g_j = 0$ for all $i = 1, \dots, M$ and

$j = 1, \dots, N$. Given $\varepsilon > 0$, perform the following two updates:

$$f_i \leftarrow -\varepsilon \log \sum_{j=1}^{M} \beta_j \exp\Big(\frac{1}{\varepsilon}(g_j - C_{i,j})\Big)$$
$$g_j \leftarrow -\varepsilon \log \sum_{i=1}^{N} \alpha_i \exp\Big(\frac{1}{\varepsilon}(f_i - C_{i,j})\Big)$$

until convergence. The reduction used here is the so-called **LogSumExp** operation. To be useful, stable numerical schemes are employed in practice, including the symmetrization of Sinkhorn steps and online normalization of the LogSumExp reduction.

Clustering, Gaussian Processes, and more...

Examples above, and many others, can be found with a great level of detail in the KeOps documentation[2] and Appendix of [Proc8].

3.2 Reducing the memory footprint of kernel Operations

Performing these operations on large dimensions can be challenging: optimizing memory consumption and computational speed are key factors in scaling methods to real-world data. Instantiating the full tensor to be reduced is not an option due to its quadratic $O(MN)$ memory footprint. By maintaining the brute-force computation scheme and taking advantage of modern GPU architecture, the approach computes the quantity on the fly while optimizing the use of cache and shared memory to accelerate the process. The KeOps library, presented below, does this seamlessly for the end user.

3.2.1 Scientific computing with GPU programming

GPU Hardware

In a nutshell, modern Graphics Processing Units (GPUs) are hardware components initially developed to accelerate graphical rendering and later adapted for scientific computing. They provide an affordable (relatively — see discussion below) massively parallel computing architecture. The leading company, Nvidia, was the first in the 2000s to recognize the potential of this emerging market, expanding its sales beyond gamers and graphics professionals to computational scientists. To encourage this, Nvidia — followed by other industry players such as AMD, Intel — developed a dedicated line of products with higher-quality components and larger memory capacities than gaming-oriented cards. These scientific computing GPUs were also designed to be compatible with data center racks, energy supplies, and cooling systems. However, the prices of these specialized GPUs rapidly increased and are now roughly 5 to 10 times higher (tens of thousands of euros) than gaming GPUs (around a thousand euros). Despite this aggressive marketing strategy, depending on the application, a €1000 gaming GPU may still provide excellent acceleration for scientific workloads.

As of 2025, more than 3/4 of the Top500 supercomputers[3] now feature both CPUs and GPU-accelerating devices, mainly from Intel, AMD or Nvidia. Thence, a large part of the peak performance

[2] See https://www.kernel-operations.io/keops/_auto_tutorials/index.html
[3] See https://top500.org/statistics/treemaps/

of new supercomputers is due to these GPU accelerators. As a result, modern computational methods are urged to adapt to this specific hardware to fully leverage its computational power. The recent explosion of machine learning (ML) has propelled GPU programming as a key technology. Although ML relies on processing vast amounts of data, computational methods in this field have adapted to GPUs' constrained memory capacity. The typical memory available on a GPU is now between 20 and 100 GB, whereas CPU RAM is an order of magnitude larger. To overcome this limitation, deep learning networks exploit, among other techniques, the parallel nature of their architecture, batch optimization schemes, and the ability to use low-precision data formats. In other fields — especially those involving complex, large-scale, and sequential codes, such as physics-based simulations — this transition is more challenging and remains a work in progress.

Supercomputers often have their own specific hardware setups, sometimes even with specific and proprietary compiler stacks. However, in academic labs or local computation clusters, the situation is simpler, as available hardware is surprisingly homogeneous — typically Intel CPUs paired with Nvidia GPUs and running a standard Linux distribution as Operating System. The emergence of alternatives to Nvidia GPUs has been tedious and slow. The KeOps software, whose target is the development of research code, unfortunately still relies only on the proprietary Nvidia/CUDA framework. While not ideal in a Free-Open Source Software and reproducible science perspective, this is of little practical limitation due to the aforementioned reason.

Software stack

Inspired by the computer graphics development stack, in 2007, Nvidia strategically initiated the development of its fully documented **CUDA** language ecosystem. It enables programmers to implement scientific code specifically on Nvidia hardware while remaining fully compatible with the widespread `C/C++` ecosystem. Another similar but cross-hardware solution, called **OpenCL**, was released a few years later thanks to a consortium that included various industry players such as AMD, Intel, Apple, and Nvidia. However, this effort never reached the same level of documentation quality and performance as the almighty Nvidia/CUDA combination. As a result, Apple has recently discontinued support for OpenCL in favor of its proprietary Metal framework[4]. A universal standard for writing hardware-independent code on GPUs remains a distant goal.

To overcome this, many software tools have been created to free the end user from manually translating code into specific languages. We will not review these solutions in detail here, as the past two decades have been rich in such projects, ranging from low-level (e.g. LLVM) to higher-level abstractions (e.g. KeOps). Instead, we will mention two well-known examples that could have been adapted to achieve what KeOps does. The first is Triton, developed by OpenAI to accelerate code fragments commonly found in ML methods. It first converts code into its Intermediate Representation (IR) language, potentially optimizing it through an *autotune* method. Finally, it compiles the code to run on hardware. The highest level of compatibility is currently announced with Nvidia, but since it relies on the LLVM-IR language, it could theoretically support other platforms as well.[5] This process is extremely fast, as it only considers small code segments and performs compilation on-the-fly, a technique known as Just-In-Time (JIT) compilation. Triton has now been integrated into PyTorch, enabling seamless and significant speedups in Deep Neural Network applications. The second example is the Taichi language [38], which follows a similar approach but is not limited to ML applications. It defines a high-level language on top of Python. Again, this language is transformed into an IR, which is then automatically compiled into various

[4] See `https://developer.apple.com/opencl/`

[5] As of 2025, the documentation remains somewhat unclear on this point...

other languages, including C++, CUDA, OpenCL, and Metal. KeOps, in a sense, has a similar goal but is specifically oriented toward kernel methods, with support limited to C++ and CUDA.

When parallelizing code, the routine is divided into smaller sub-tasks, each executed by an individual thread. In the **Single Instruction, Multiple Thread** (SIMT) paradigm, the program specifies the operations to be performed by each thread running concurrently. When a thread depends on the result of another, a synchronization barrier is introduced to ensure all threads reach the same point before proceeding. If not properly managed, such barriers can introduce unnecessary latency or even lead to bugs. In GPU-based SIMT programming models like CUDA and OpenCL, execution is split into two main components: (1) the **host** part, which runs on the CPU, coordinates the overall computation and interfaces with external programs; and (2) the **device** part, which runs on the GPU and performs the actual computation. This latter portion is typically referred to as a GPU **kernel** — a concept similar to a **shader** in computer graphics.

Threads, blocks and memories

Writing high-performance code requires a solid understanding of computer hardware fundamentals. A processor consists of physical cores, each capable of executing multiple threads simultaneously. In contrast, a GPU contains thousands of floating-point cores — organized into units called Streaming Multiprocessors (SMs)[6] — with the exact number depending on the floating-point (FP) precision used: FP64 (double), FP32 (single), FP16 (half), or even 8-bit operations. The number of SMs, cores, and other architectural features varies across GPU generations. As with CPUs, the raw count of physical cores is abstracted away in practice, as developers primarily work with virtual threads. The first step in writing a GPU kernel is to define a computation plan — or grid — that distributes the workload across the SMs. The objective is to select an arrangement that maximizes GPU occupancy and delivers results as efficiently as possible. Inherited from the graphics rendering domain, the grid structure can be one-, two-, or three-dimensional. The first dimension can contain up to $2^{31} \approx 10^9$ threads, which is sufficient for most applications; however, the second and third dimensions are restricted to a maximum of $2^{16} = 65536$, which may impose constraints in certain scenarios. Furthermore, each grid dimension is subdivided into **blocks**, with each block supporting up to 1024 threads in total. Threads within the same block can share data at minimal cost, a feature that is essential for achieving high performance. For every kernel launch, two parameters must be provided: the number of blocks in each grid dimension and the number of threads per block in each dimension.

GPUs feature multiple types of memory, each with different sizes, access scopes, and speeds. As a general rule, the broader the scope and capacity of a memory type, the slower its access. The largest is global memory (or GPU RAM), typically comprising tens of gigabytes and shared across all cores. Every thread can read from and write to global memory, but access times are relatively high. To improve efficiency within blocks, GPUs provide **shared memory** — a small, low-latency cache (on the order of tens of kilobytes) accessible only to threads within the same block. Because it resides close to the cores, access to shared memory typically costs just a few clock cycles. At the finest level, each thread has access to a limited number of **registers** (sometimes called local memory), which are extremely fast but very limited in size.[7]

It is important to note that GPUs are very efficient with single-precision floating-point numbers.

[5] This use of the term "kernel" is unrelated to the mathematical concept of a kernel operator.

[6] For example, the 2025 Nvidia Blackwell GPU features 192 SMs, each with 128 FP32 cores, totaling 24,576 cores

[7] The 2025 Nvidia RTX 6000 comes with 96 GB of RAM, 228 KB of shared memory per SM and/or 227 KB of shared memory per block, 64K 32-bit registers per SM, and only 255 registers per thread.

[8] From https://www3.nd.edu/~zxu2/acms60212-40212-S12/Lec-12-02.pdf

Variable declaration	Memory	Scope	Lifetime	Access	Size
`__device__ __local__`	local	thread	thread	very fast	kB
`__device__ __shared__`	shared	block	block	very fast	kB
`__device__`	global	grid	application	very slow	GB
`__device__ __constant__`	constant	grid	application	fast	MB

Table 3.1: CUDA variable Types[8]

Recent advances in hardware have made double-precision computations faster than in the early days when operations with doubles were ten times slower. There also exist specialized cores capable of performing very specific operations at blazing speed. For instance, Nvidia Tensor Cores compute a 4×4 matrix multiplication added to another 4×4 matrix — known as fused multiply-add — ten times faster than a standard implementation. This is obviously targeted at the ML community, as this operation is at the core of DNN models, but any code using this pattern can benefit from it.

3.2.2 Tiled reduction scheme to avoid memory transfers

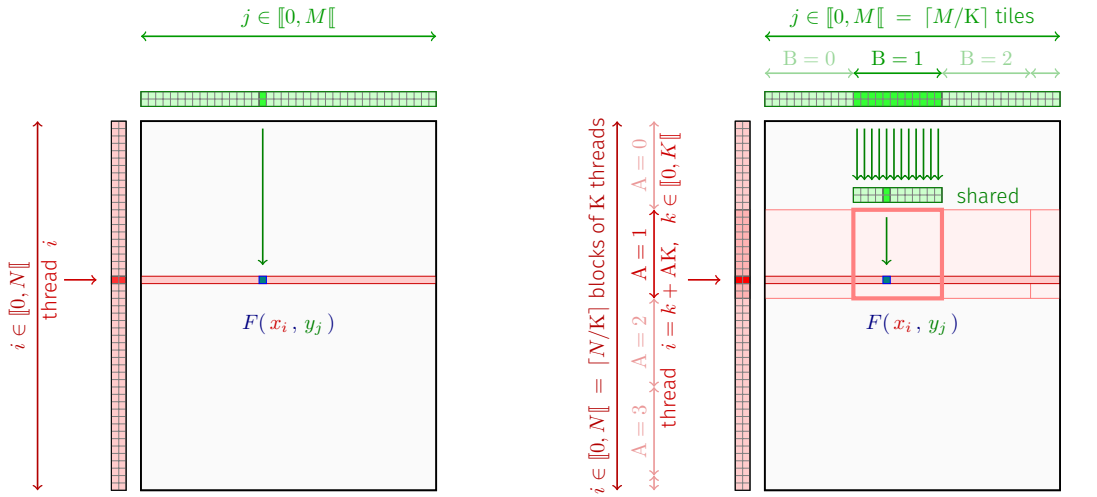
The general reduction used in KeOps follows a standard implementation of a fast parallel scheme. We rely on the well-known N-body simulation example — described in the eponymous chapter of [60] — to compute reductions of symbolic matrices, as shown in Equation (3.2). The reduction operation follows a one-dimensional computation plan along the dimension i, as illustrated in Figure 3.1a. To clarify the concept, consider a simple sum reduction: $\gamma_i = \sum_j F(x_i, y_j)$.

A first straightforward but naïve approach is to assign a thread to each value of i, allowing it to compute γ_i by iterating over the reduction index j and calculating $F(x_i, y_j)$ on the fly. Unfortunately, this approach is not very efficient. Even though the computation is parallelized across thousands of cores, the vast majority of time is spent not on calculations but on memory transfers involving the y_js. This inefficiency arises because we do not take advantage of the block structure mentioned in Section 3.2.1.

A well-known note[9] from NVIDIA developers illustrates how crucial a good understanding of hardware structure is in SIMT programming. By making clever use of low-latency memory, sequential addressing, loop unrolling, and other optimizations purely based on hardware-specific features, it is possible to achieve a $30\times$ speedup on a simple sum reduction.

The key to efficient programming in our case is to split the reduction into **tiles**, leveraging the low latency of the shared memory buffer and block-wise memory accesses. This approach is implemented in KeOps and illustrated in Figure 3.1b. The computation plan, still applied to the variable i, is divided into blocks of size $k \ll N$. Each thread within a block accesses the low-latency shared memory with both read and write permissions. The main loop over j is then split into tiles that are processed sequentially. For each tile, shared memory is first populated with the required values to compute the $F(x_i, y_j)$ belonging to it. The tile set to K, as the threads fetch a single y_j simultaneously. A synchronization barrier is then invoked, ensuring that shared memory is fully loaded before proceeding. Next, the reduction is performed on this tile, and the result is accumulated into the variable containing the partial sum computed so far.

[9] See https://developer.download.nvidia.com/assets/cuda/files/reduction.pdf



(a) Naïve reduction scheme

(b) Tiled reduction scheme using shared memory

Figure 3.1: Comparison of reduction schemes.

3.2.3 Advanced computational plan

Large internal dimension

The first development of KeOps was done with shape analysis applications in mind. In this field, the sizes N and M are large, typically ranging from 1000 to 10^7, while D is small — around 10, for instance. In this setup, the shared memory size described in Section 3.2.1 is not a real limitation. However, feedback from users in the machine learning community, where models can involve dimensions D ranging from 100 to 1000, has reported performance drops.

The main limitation of KeOps' reduction scheme stems from the overflow of shared memory registers in the reduction of Figure 3.1b. This results in decreased performance on large feature vectors with $D > 100$. Indeed, with a default block size $K = 192$ and $D = 100$, we need to store around 80kB, which is the typical size of the shared memory. The compiler then allocates extra space in the slow global memory, leading to large transfer times when the threads access the data. This is known as register spilling.

A common optimization strategy, when the formula F is separable along the coordinate corresponding to the internal dimension, is to split the computation into **chunks**[10]. This approach allows sequential loading of only a subset of the D coordinates required to compute F, reducing memory overhead. For instance, given a chunk size $C \in \mathbb{N}$, and if a (sub)formula corresponds to the squared Euclidean norm $F(x_i, y_j) = \sum_{k=1}^{D} (x_{i,k} - y_{j,k})^2$, the computation can be restructured so that each chunk represents a partial sum over C consecutive terms. Currently, this technique is implemented only for sum and scalar product operations, which cover the case of radial kernels. Selecting optimal chunk, block, and tile sizes depends on the hardware specifications.

[10] There are blocks, tiles, and now chunks — different names as they apply at different stages, but following a common pattern.

Block partial reductions

In parallel computing, the ideal scenario is when all threads perform similar tasks. It is often more efficient to compute extra, unnecessary terms rather than introduce conditionals that desynchronize threads, leading to waiting times at synchronization barriers. However, partial reductions can sometimes be beneficial, particularly when the kernel has small compact support or when most of the interactions are negligible. KeOps supports the specification of block-wise sparsity masks as optional attributes. Block reduction tiles are encoded using a collection of so-called ranges, represented as tuples of indices $(i_{\text{start}}, i_{\text{end}}, j_{\text{start}}, j_{\text{end}})$. These ranges help focus the computational effort on a specific subset of the full set of interaction pairs $[\![1, N]\!] \to [\![1, M]\!]$. This approach enables pruning negligible terms from symbolic reductions while preserving the contiguous memory accesses that are crucial for CUDA kernel performance.

However, users should be mindful of the number of ranges utilized, as they may become significant in size. Since M and N can be large, these ranges are stored as long integer (int64), potentially leading to non-negligible memory overhead.

3.3 Providing High Level Computational tools

Few researchers developing applied mathematical methods possess the expertise or have the time to delve deeply into the SIMT subtleties outlined above. There is a need for high-level, yet performance-competitive, tools to develop, test, and deploy original methods in laboratories. KeOps aims to automate the generation of efficient code for large tensor reductions.

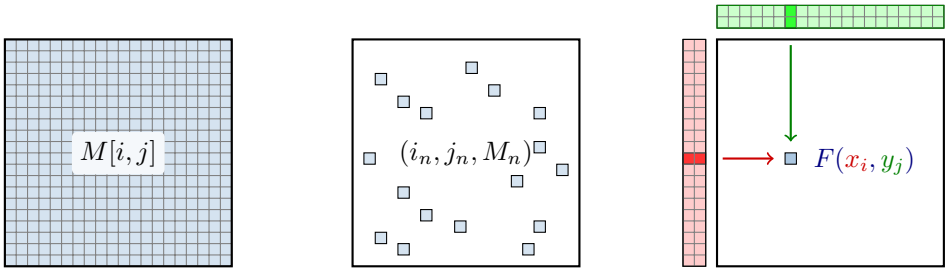
As mentioned earlier, other high-end metaprogramming tools could be used to perform efficient computations related to kernel methods. However, development efforts in these libraries have mostly focused on supporting atomic or specialized (in machine learning or physics) algebraic operations and cross-hardware compatibility. The academic niche of KeOps is its ability to align with the mathematical concept of kernel operators in all their variety. Despite its limited range of supported backends (CUDA and C++), it has found its audience in various communities, as kernel operators are ubiquitous tools in many practical fields.

In shape analysis, it has been widely used to implement increasingly refined deformation models [Proc9] and efficient distance measures between shapes [Proc2]. In machine learning, it has been applied with transformers [68] and the Sinkhorn algorithm. The Python Optimal Transport library [24] has benefited from these advancements, as have Gaussian processes, particularly through the GPyTorch library [28]. It has also been successfully applied in biological research, notably for studying protein structures [61, 1], and in large-scale data analysis [53].

3.3.1 The Symbolic Tensor abstraction

Tensor data

Modern scientific libraries now fully implements both dense and sparse tensor-like data structure type, as illustrated Figure 3.2. Dense data structures are the most commonly used and store every coefficient explicitly in memory. They are supported by well-established and portable libraries, ranging from low-level operations (e.g., BLAS, MKL, cuBLAS) to advanced linear algebra routines (e.g., LAPACK, MKL). Dense array manipulation is versatile, efficient, and user-friendly — even across



(a) Dense tensor (b) Sparse tensor (c) Symbolic tensor

Figure 3.2: Scientific computing libraries represent most objects as tensors

programming languages[11]. For example, the Python scientific computing community has made significant efforts to define a standard API for tensor-like objects[12], aiming to ensure compatibility across various implementations such as NumPy arrays, PyTorch tensors, JAX tensors, and others.

In many graph-related problems common in modern applications, the majority of tensor entries are zero. In such cases, sparse data structures offer a significant advantage by storing only the indices and corresponding values of the non-zero entries. Various formats exist — such as Dictionary of Keys or List of Lists or Compressed Sparse Row/Column — and standard linear algebra libraries have been adapted accordingly, making sparse support widely available. While sparse structures significantly reduce memory usage, they are not always well suited for massively parallel computing architectures. Although libraries like cuSPARSE provide some support, efficiently utilizing the full capacity of computational devices remains a challenge. This is particularly true when threads do not follow similar data access patterns, which limits occupancy and performance.

Symbolic Tensors

Introduced in version 2 of KeOps, **symbolic tensors** (known as `LazyTensor` in the current version of KeOps) provide an abstraction for seamlessly manipulating operations described by Equation (3.2). When working with kernels, the full tensor $\eta_{\boldsymbol{\ell},i,j,\boldsymbol{k}}$ is generally not sparse. However, Assumption (3.1) ensures that the elements required to fully define it remain memory-efficient.

A symbolic tensor is a user-friendly class with syntactic sugar that encapsulates all the information required to evaluate a formula efficiently and transparently. Its two main attributes are: (1) a string representing the generic mathematical formula, and (2) metadata about the variables referenced in that formula. Several pieces of information are essential for unambiguously evaluating a symbolic tensor: the index assigned to each variable in the formula, indicating its position among the function's arguments; the internal dimension of each variable; and the dimensionality of the formula's output. Additionally, it is necessary to track whether the underlying data is stored contiguously and whether it resides on the CPU or GPU, in order to avoid unnecessary memory transfers.

Internal dimension

A wide range of common mathematical operations can be applied to a symbolic tensor. These operations act only on the internal dimensions (indexed by $\boldsymbol{k}$). Standard mathematical operators

[11] Two conventions coexist: column-major storage (used in FORTRAN, R, Matlab) and row-major storage (used in C, Python)

[12] See the initiative at `https://data-apis.org/`

```python
from pykeops.torch import LazyTensor

def gaussian_kernel(x, y, sigma2):
    x_i = LazyTensor(x[:, None, :])      # ([M, 1], D)
    y_j = LazyTensor(y[None, :, :])      # ([1, N], D)
    D_ij = ((x_i - y_j) ** 2).sum(-1)    # ([M, N], 1) SymbolicTensor (squared distances)
    return (-D_ij / sigma2).exp()        # ([M, N], 1) SymbolicTensor (Gaussian kernel)

x = torch.randn(M, D)                # (M, D) torch tensor
y = torch.randn(N, D)                # (N, D) torch tensor
p = torch.randn(N, D)                # (N, D) torch tensor

K_ij =  gaussian_kernel(x, y)        # ([M, N]) SymbolicTensor (Gaussian kernel)
v = K_ij @ p                         # MatMult ([M, N], 1) @ (N, D) = (N, D) torch tensor
```

Listing 1: The adaptation of a Gaussian convolution implementation from PyTorch to KeOps required only minimal modifications, limited to the orange-highlighted lines.

such as `+`, `-`, `*`, `/`, `**`, etc., are fully supported, along with a variety of mathematical functions including `exp`, `cos`, `tensordot`, and many others[13].

Given the following dummy dataset,

```python
import numpy as np
M, N, D = 1e6, 2e6, 3
x = np.random.randn(M, D)                    # (M, D) numpy array
y = np.random.randn(N, D)                    # (N, D) numpy array
```

we can create symbolic tensors representing the squared Euclidean distance between these data points. Operations applied to a symbolic tensor are recorded and assembled into a formula, which is internally parsed as an abstract syntax tree (AST). The variables in the expression correspond to the leaves of this tree. They can be explicitly declared as follows:

```python
from pykeops.numpy import Vi, Vj
x_i = Vi(x)                                  # [(M, 1), D] SymbolicTensor
y_j = Vj(y)                                  # [(1, N), D] SymbolicTensor
eta = ((x_i - y_j) ** 2).sum()               # [(M, N),] SymbolicTensor
```

Here, the `Vi` (resp. `Vj`) constructor outputs a `LazyTensor` indexed by i (resp. j) and representing a variable. Another constructor uses a modified view of the tensor, allowing KeOps to infer the variable type (i or j) directly from the `shape` attribute. The two lines below are equivalent to the declarations shown above:

```python
from pykeops.numpy import LazyTensor
x_i = LazyTensor(x[:, np.newaxis, :])        # [(M, 1), D] SymbolicTensor
y_j = LazyTensor(y[np.newaxis, :, :])        # [(1, N), D] SymbolicTensor
```

The memory footprint of `eta` is negligible as it only stores the following information

[13] See the full list at https://github.com/getkeops/keops/tree/main/keopscore/keopscore/formulas/maths

```
print(eta)
# KeOps LazyTensor
#     formula: Sum(Square((Var(0, D, 0) - Var(1, D, 1))))
#     shape: (M, N)
```

The formula involves two variables, where the triplet `Var(pos, dim, type)` specifies the position in the function argument list, the internal dimension, and the variable type (i or j).

The current implementation of `LazyTensor` is not yet complete, as the inner dimension is necessarily flattened. For instance, when a variable $x \in \mathbb{R}^{M \times D_0 \times D_1}$ is represented, it is implemented as a `LazyTensor` with shape `[M, D0 * D1]`.

```
mat = np.random.randn(M, D, D)
vec = np.random.randn(N, D)

mat_i = Vi(mat.reshape(M, D * D))          # flatten the inner dimension
vec_j = Vj(vec)                            # already flattened
```

In practice, to emulate multi-index operations on `LazyTensor`, the user must explicitly provide the sizes of the inner dimensions. High-level slicing and broadcasting operations are not yet supported and must be implemented manually. For instance, consider the following tensor field contraction: $\eta_{i,j,k} = \sum_{k_1} x_{i,k_0,k_1} y_{j,k_1}$ which corresponds to a matrix-vector multiplication. It could be written

```
eta = mat_i.matvecmult(vec_j)              # unflatten implicitly
```

As KeOps ensures that the internal dimension of `vec_j` divides the dimension of `mat_i` in order to perform the contraction. However, under the hood, it calls the generic `keops_tensordot()` operation, which can be somewhat verbose

```
eta = mat_i.keops_tensordot(vec_j,
                            (D, D), (D,),     # unflatten explicitly
                            (1,), (0,))       # contraction indices
```

Efforts are ongoing to fully abstract away this additional boilerplate for the user. The next version of `LazyTensor` allowing this will be called `SymbolicTensors`.

Reductions

When considering a kernel operator, we are primarily interested in the result of applying this operator to an input vector. This is the case when computing the scalar product in a Reproducing Kernel Hilbert Space (RKHS), where the key element is a function that evaluates the kernel along the data, rather than the kernel matrix itself. Similarly, this is true when searching for nearest neighbors, where the most relevant information is often the shortest distance, rather than the full distance matrix.

This is why we do not need to access the actual values of the tensor η, which could correspond to the dense version of `eta` in our example. As a result, any mathematical operation on symbolic tensors can be processed lazily, meaning that the actual computations are only performed when a reduction on the external variable is invoked.

Many reduction operations have been implemented in KeOps[14]. The ones used in the examples of Section 3.1.2 include `sum_reduction`, `argkmin_reduction`, and `logsumexp_reduction`. To distinguish these from operations acting on internal dimensions, the keyword `_reduction` has been appended to their names. The user should specify the index i or j to be reduced via the `axis` or `dim` parameter. Invoking these methods triggers the compilation of optimized code and generates an executable, which is then used within the program.

Coming back to our example with the squared Euclidean norm, if we are interested in a nearest neighbor query, we simply need to apply the following reduction:

```
indices = eta.argKmin_reduction(K=1, axis=1)  # (M, 1) numpy array
```

This returns the full list of indices corresponding to the points in `y` that are nearest neighbors to the points in `x`. The parameter `K=1` specifies the number of argmins to be returned, and the reduction is applied along the j index (i.e., `axis=1`).

Sometimes, it is useful to apply the same formula to different data. A `LazyTensor` can be declared with symbolic variables, which serve as placeholders and do not explicitly link to an existing data array. In this case, the CUDA kernel is compiled, but no computations are triggered. Instead, a function capable of applying this kernel is returned. The `LazyTensor` with symbolic variables can then be called with actual data as arguments when the user needs to perform the reduction[15]:

```
eta =  ((Vi(0, D) - Vj(1, D)) ** 2).sum()     # symbolicTensor with placeholders
eta_fun = eta.argKmin_reduction(K=1, axis=1)  # a function with two args
indices = eta_fun(x, y)                       # (M, 1) numpy array
```

These constructors take the position of the variable in the argument list and the internal dimension D of each variable. For example, `Vi(0, D)` essentially means that the first variable (at position 0) has an internal dimension of D. Knowing this information for each variable is required at compile time, allowing KeOps to generate optimized code, such as templates or loop unrolling. Therefore, each time a formula involves a different internal dimension, a new compilation occurs, as described in Section 3.4.2 below. However, it is important to note that the external dimensions are only known at runtime, as reductions can be applied to different observed data.

3.3.2 Automatic differentiation

Automatic differentiation (AD) has existed for decades, dating back to the 1960s [79]. However, in the past ten years, with the rise of machine learning libraries such as PyTorch, JAX, and TensorFlow, AD has become compatible with high-level scientific languages. AD is also useful when dealing with kernel methods, and KeOps provides an internal AD engine for symbolic tensors. Every differentiable KeOps operation comes with its corresponding differential operations, required for both forward and reverse-mode AD. Additionally, we have made the `LazyTensor` abstraction fully compatible with the PyTorch AD engine, enabling the use of KeOps operations in any Torch script.

The formula $F : \mathbb{R}^n \to \mathbb{R}^d$ can be written as a composition $F = F_p \circ \cdots \circ F_1$ of p functions $F_i : E_{i-1} \to E_i$, where $E_i = \mathbb{R}^{d_i}$. With these notations $d_0 = n$ and $d_p = d$. Evaluating the gradient of F with the **backpropagation algorithm** requires:

[14] A full list is available at https://www.kernel-operations.io/keops/api/math-operations.html#reductions
[15] See https://www.kernel-operations.io/keops/_auto_tutorials/a_LazyTensors/plot_lazytensors_c.html

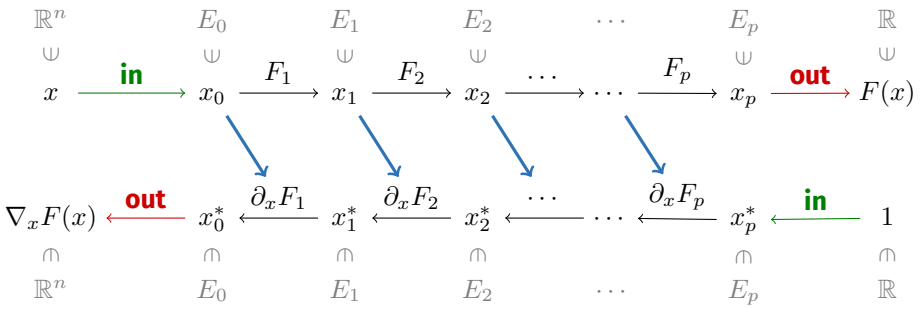


Figure 3.3: Reverse automatic differentiation principle. The first row depicts the forward pass, while the second row illustrates the backward pass.

1. A **Forward pass** to evaluate the functions

$$\begin{array}{rccc} F_i & : & E_{i-1} & \to & E_i \\ & & x & \mapsto & F_i(x) \end{array}$$

   and thus compute the value $F(x)$.

2. A **Backward pass** to evaluate the (adjoints of the) differentials

$$\begin{array}{rccc} \partial F_i & : & E_{i-1} \times E_i & \to & E_{i-1} \\ & & (x_{i-1}, x_i^*) & \mapsto & [\mathrm{d}F_i^*(x_{i-1})](x_i^*) = x_{i-1}^* \end{array}$$

   and compute the gradient of F at location x, applied to an arbitrary e in the space of outputs. When F is real-valued (i.e., $d = 1$), choosing $e = 1$ yields $\nabla F(x) = \partial_x F(x) \cdot 1$. This step is sometimes called the vector-Jacobian product (`VJP`). It is often the memory bottleneck of many algorithms [33], as it requires storing the intermediate values of the forward pass, as illustrate by the blue arrows in Figure 3.3.

At a low level, KeOps enables us to compute gradients efficiently using the `Grad` instruction. Given a formula F, the symbolic expression `Grad(F, V, E)` represents the formula for the gradient $[\partial_V F(x)](E)$ with respect to the variable V, evaluated on the input variable E. If `V` is a variable placeholder that appears in the expression for `F`, and if `E` has the same dimension and category as `F`, then `Grad(F, V, E)` can be fed into KeOps just like any other symbolic expression. The resulting output will have the same dimension and category as the variable `V`, making it compatible for tasks such as gradient descent or higher-order differentiation. KeOps fully supports nested gradient calculations, such as `Grad(Grad(...,...,...),...,...)`.

The formula derived by the `Grad` operator may lead to long and complex expressions, especially for higher-order derivatives. To manage these potentially suboptimal formulas, KeOps includes a recursive formula simplifier accessible through the `auto_factorize` option[16]. It automatically factorizes expressions to reduce their complexity. This can sometimes help avoid large intermediate results within the internal dimensions, which risk overwhelming the shared memory.

Additionally, KeOps provides higher-level operators, such as `Divergence` and `Laplacian`, which have been implemented and can be used directly in KeOps formulas. These operators are optimized and maintains computational efficiency compared to naïve implementations[17].

[16] See https://www.kernel-operations.io/keops/python/api/common/GenericLazyTensor.html#pykeops.common.lazy_tensor.GenericLazyTensor.auto_factorize

[17] See Symbolic gradients and linear operators section of https://www.kernel-operations.io/keops/api/math-operations.html#math-operators

3.3.3 Advanced linear algebra operations with kernels

LinearOperator compatibility

As emphasized in Section 3.3.1, many methods do not require knowledge of the individual entries of a linear operator matrix to be effective. The lazy evaluation of KeOps reductions fits into this framework, which is common in high-level scientific libraries. We have also discussed the Jacobian-vector product (JVP)[18] (and VJP for the backward pass) in forward (and backward) AD engines. This abstraction is similarly used in iterative methods for sparse linear algebra in SciPy through the `scipy.sparse.linalg.LinearOperator`[19] class. For instance, these include: Linear problem solvers, such as `cg()` (conjugate gradient) or `gmres()` (Generalized Minimal Residual); and matrix factorizations like `eigsh()` (for largest eigenvalues) or `svds()` (partial Singular Value Decomposition), among others.

To convert a `LazyTensor` into a `LinearOperator`, the only methods that need to be provided are the `shape` attribute, which returns the dimensions M and N, and the callable methods `matvec()` or `rmatvec()`. These correspond exactly to the functions returned by symbolic tensors with placeholders, as described in Section 3.3.1, with reductions applied to the j — or i — variable, respectively.

Let us provide two examples that illustrate the usefulness of this feature. First, given a regularization parameter $\alpha > 0$ and vectors $x \in \mathbb{R}^{M\times D}$, $b \in \mathbb{R}^M$, suppose we need to compute $a = (\alpha \operatorname{Id} + K_{xx})^{-1} b$, where K is a Gaussian kernel. We can then call the conjugate gradient method with the following code — the `gaussian_kernel()` is declared in Listing 1:

```
from scipy.sparse.linalg import aslinearoperator, cg
from scipy.sparse import diags
x = np.random.randn(M, D)
G_ij = gaussian_kernel(x, x)                    # ([M, M]) SymbolicTensor
K_ij = aslinearoperator(G_ij)                   # ([M, M]) linearOperator
alphaId_ij = aslinearoperator(
                diags(alpha * np.ones(M)))      # ([M, M]) LinearOperator

b = np.random.randn((M, 1))                     # (M, 1) numpy array
a = cg(K_ij + alphaId_ij, b)                    # (M, 1) numpy array
```

Second, we consider the problem of computing the first five eigenvalues of the kernel K. This can be done simply by writing:

```
from scipy.sparse.linalg import eigs
eigenvalues, eigenvectors = eigsh(K_ij, k=5)  # (5, ) and (M, 5) numpy arrays
```

It returns the eigenspaces computed using the Implicitly Restarted Lanczos Method, as implemented in the ARPACK software package. Optimized routines from SciPy can now be applied to symbolic tensors of size in the millions. For further examples, refer to the KeOps documentation.

[18] https://docs.jax.dev/en/latest/_autosummary/jax.jvp.html

[19] https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.linalg.LinearOperator.html#scipy.sparse.linalg.LinearOperator

Solve reduction

The main limitation of casting `LazyTensor` as SciPy `LinearOperator` lies in the memory transfers required at each iteration of the solver. Since SciPy relies on NumPy and C++ or FORTRAN routines, arrays are stored in CPU memory, and KeOps must transfer the data back and forth between the CPU and GPU at each call of the `LinearOperator` methods `matvec()` or `rmatvec()` routines. Users should also be aware that KeOps avoids silent type downcasting from FP64 (the default in NumPy) to FP32 (the default on the GPU), which helps preserve precision but can result in significant performance degradation on older GPUs.

Note that a similar effort is currently under development through the PyTorch Tensor API via the LinearOperator[20] project. However, this project is still in its early stages. To address this, we provide a conjugate gradient implementation that allows users to solve linear systems involving kernel operators. Thanks to the PyTorch tensor framework, data remains on the GPU throughout the iterations, avoiding costly memory transfers.

The evaluation of the conjugate gradient method is viewed as a reduction along the j variable. It can be easily called with the `.solve()` method. Similar to the `scipy.sparse.linalg.cg` example above, the syntax is as follows, using the declared `gaussian_kernel()` from Listing 1:

```
b = torch.randn(M, 1)                    # (M, 1) torch tensor
K_xx = gaussian_kernel(x, x)             # ([M, M]) SymbolicTensor
a = K_xx.solve(b, alpha=.5)              # (M, 1) torch tensor
```

A few warnings for users: computing in FP32 may lead to numerical instability. Additionally, adding a preconditioner is crucial for achieving good performance[21].

3.4 Implementation details and structure of the library

The KeOps project is divided into two independent components. The first is a low-level module, which can be thought of as an Application Programming Interface (API). It contains the internal `keopscore` metaprogramming engine that generates and compiles the code corresponding to a given formula. Built on top of this core, the bindings — implemented as `PykeOps` for Python and `RKeOps` for R — act as interfaces between user commands and the engine, exposing high-level functionality. This second module is responsible for triggering computations when results are needed and transferring data to the appropriate device if necessary.

An overview of the data flow in KeOps using the Python binding with PyTorch is shown in Figure 3.4. This is the most commonly used setup, as it demonstrates how PyKeOps integrates seamlessly with PyTorch's automatic differentiation engine. Other bindings follow a similar pipeline. The main steps are: (1) the creation of a generic formula from symbolic tensor expressions; (2) parallel code generation and binary compilation; (3) execution of the computation on the data, with the result returned to the user.

3.4.1 KeOps formulas

The internal engine of KeOps is designed to generate an optimized and parallelized routine from a given formula — an expression involving mathematical functions and a set of variables — which can

[20] https://github.com/cornellius-gp/linear_operator

[21] https://www.kernel-operations.io/keops/_auto_benchmarks/plot_benchmark_invkernel.html#sphx-glr-auto-benchmarks-plot-benchmark-invkernel-py

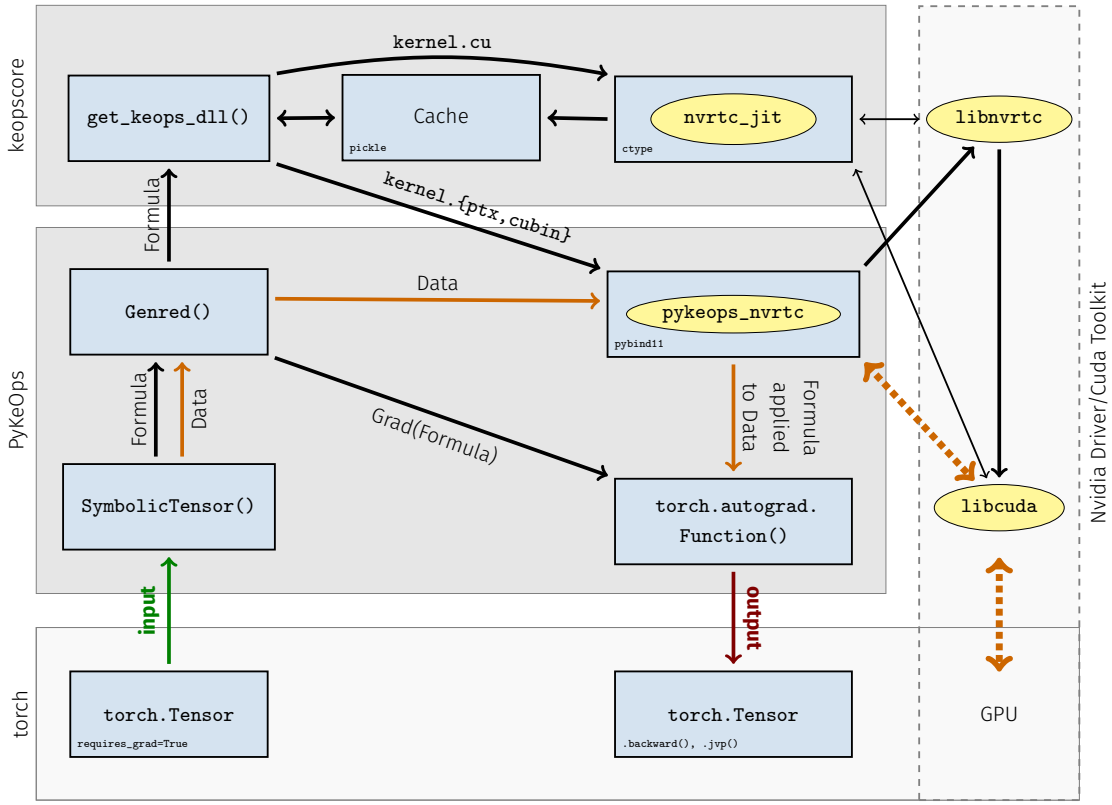


Figure 3.4: Workflow of the KeOps stack with PyTorch bindings. Blue boxes represent Python entities, while yellow ellipses denote dynamic libraries written in C/C++/CUDA. Dotted orange arrows indicate potential points where a data copy may occur.

then be applied to user data either immediately or at a later stage. On the Python side, the formula is handled by the `pykeops.Genred()` module (where `Genred` stands for Generic Reduction), which subsequently calls the `keopscore.get_keops_dll()` entry point in the `keopscore` package to retrieve the corresponding computational routine.

The formula and variables are provided as a string, where each token must correspond to a valid class from the `keopscore.formulas` module. This module defines the full KeOps grammar, including standard mathematical operations on inner dimensions (`exp`, `cos`, `+`, ...), reduction operators on outer dimensions (`sum`, `min`, `prod`, ...), and differential operators acting on complete formulas (`Grad`, `Div`, `Laplacian`, ...). Each class implements two methods: one for generating the code corresponding to the direct evaluation of the operation (called `Op` in mathematical operations and `FinalizeOutput()` in reductions), and another for generating the corresponding derivative formula (called `DiffT()` for the adjoint of the differential). An example for the log operation is provided in Listing 2 and can be compared to with the C++ implementation of KeOps version 1 in the Listing page 48 of [23].

The `keopscore` module then uses the Python parser to recursively evaluate the formula expression, calling the `Op` method for each term to inject the corresponding code for direct evaluation into a formatted file containing the raw source code. Generating the gradient formula (i.e., the backward pass in reverse-mode automatic differentiation) is achieved by calling the `DiffT` methods recursively, which can then, in turn, be used to inject the corresponding evaluation code into a source file.

Currently, the KeOps engine can generate code in C++ (as `.cpp` files), leveraging the OpenMP

```
    class Log(Operation):
        """the logarithm operation"""
        def __init__(self, arg):
            # arg is another instance of Operation
            self.children = arg
            self.string_id = "Log"

        def Op(self, out_var, arg_var) -> str:
            # c++/cuda code to evaluate the log
            loop = f"for(int k = 0; k < {out_var.dim}; k++)"
            code = f"   {out_var.id}[k] = log({arg_var.id}[k]);"
            return "\n".join(loop, log_code)

        def DiffT(self, v, gradin):
            # return an Operation that could be use in the backward pass of the AD engine
            f = self.children
            df = 1 / f
            return f.DiffT(v, gradin) * df
```

Listing 2: The (pseudocode) implementation of log operation in KeOps version 2.

library for multicore CPU parallelization, or in C++/CUDA (as `.cu` files) for execution on Nvidia GPUs via the CUDA toolkit and Nvidia driver stack. The compilation process differs slightly between these two targets and is detailed in the following sections.

3.4.2 Just-In-Time Compilation

KeOps Cache System

Generating the source code corresponding to a given formula involves simple string manipulations and is fast, typically taking only a few milliseconds. Compiling this code into a binary takes longer — ranging from a few tenths of a second to several seconds depending on the compiler — but this process is performed only once. When a formula is passed to `get_keops_dll()`, the system checks an internal cache for an existing binary corresponding to the requested operation and compatible with the user's target device. If such a binary exists, it is returned immediately. Otherwise, if the formula is encountered for the first time, `get_keops_dll()` proceeds with the full compilation process.

The cache system is implemented using Python's `pickle` serialization module. Since the compiled binary depends on the hardware architecture, it may need to be recompiled when the system configuration changes. The cache can be safely cleared at any time by deleting its directory — by default, KeOps stores cached binaries in `$HOME/.cache/keopsX.X.X`.

CPU and OpenMP backend

The routines generated with the C++/OpenMP backend run exclusively on multicore CPUs. This backend is primarily intended for development purposes, enabling users to test KeOps on systems without an Nvidia GPU. Reduction operations are implemented using straightforward `for` loops (with optional range-based variants) and parallelized via OpenMP pragmas. These computational

schemes, available in `keopscore.mapreduces.cpu`[22], have a low memory footprint since no shared data buffers are used. Their performance depends entirely on compiler optimizations and may vary significantly with both the formula and the underlying hardware.

The compilation procedure is as follows. Each formula and hardware architecture are referenced through a unique hash. Two `.cpp` files are created with this identifier. The first contains the code parallelized for the reduction, while the second, acting as a main file, includes the `pybind11` [40] bridge between C++ objects and the Python module. Compilation is carried out using either `g++` or `clang` via a CMake[23] recipe and produces a single dynamically linked shared object that can be imported seamlessly into Python as a standard module. This module includes methods to perform the user-requested reduction and can be serialized with `pickle` into a cache file, as described in Section 3.4.2.

GPU with CUDA and runtime compiler

Version 1 of the KeOps compilation stack with the C++/CUDA backend used exactly the same compilation procedure as in CPU mode, described in Section 3.4.2, except that it relied on the Nvidia compiler `nvcc` to generate the binary corresponding to a given formula.

To reduce compilation time, KeOps version 2 now uses the Nvidia Runtime Compiler `nvrtc`[24] to produce binaries that are launched directly by the GPU driver through `libnvrtc` and `libcuda`. This process is illustrated in Figure 3.4. When PyKeOps is launched for the first time, it generates two shared libraries that will be used later for all the formulas. This compilation stack requires both the CUDA toolkit (which provides `libnvrtc`) and the Nvidia driver (which provides `libcuda`) to be installed on the system. Issues related to detecting these two libraries are a primary topic of concerns raised by users.

The first shared library, called `nvrtc_jit`, is part of the `keopscore` Python module. It generates a binary kernel from the raw CUDA source code `.cu` file corresponding to a formula reduction with its unique hash identifier. The different computation schemes described Sections 3.2.2[25] and 3.2.3 are available in the `keopscore.mapreduce.gpu` module. This binary kernel is produced either as a `.cubin` or `.ptx` file, which represent the binary and text versions of the IR language[26] of CUDA. These objects can be serialized in a cache file using `pickle`, ensuring that this compilation step is only performed the first time a formula is used. This significantly reduces overhead when the same formula is applied multiple times to different data.

The second shared library, named `pykeops_nvrtc`, serves as a gateway for computation and is part of the `pykeops` binder. Like the C++/OpenMP backend, it is linked to Python through the `pybind11` library. It accepts any binary kernel provided by `keopscore` along with their associated data tensors and launches the computation of the reduction on the GPU. It then exposes the result to the user on the Python side. The `pykeops_nvrtc` shared library handles the necessary data transfers, potentially copying data between the CPU memory (host) and the GPU memory (device) as needed.

[22] See https://github.com/getkeops/keops/tree/main/keopscore/keopscore/mapreduce/cpu
[23] See https://cmake.org/
[24] See https://docs.nvidia.com/cuda/nvrtc/
[25] See https://github.com/getkeops/keops/blob/main/keopscore/keopscore/mapreduce/gpu/GpuReduc1D.py
[26] A kind of high-level assembly language.

3.4.3 High level Binders

The binders provide a high-level interface for computing reductions from interpreted scientific programming languages such as Python or R. Leveraging symbolic tensor abstractions, they offer syntax similar to that of popular tensor-based libraries like NumPy, PyTorch, or R matrices. Version 1 of KeOps also included support for Matlab (via KeOpsLab) and a C++ API (KeOps++). However, these two interfaces were discontinued in Version 2 to concentrate development efforts on the platforms most widely used by the community.

Since each language has its own specificities, a binder must include both an implementation of symbolic tensors to construct formulas and an interface that bridges the language either to C/C++/CUDA — for direct interaction with the Nvidia driver — or to Python, using PyKeOps and NumPy as an intermediate layer (see the discussion in Section 3.4.3).

PyKeOps binder for Python

The PyKeOps binder is compatible with both `numpy.ndarray` and `torch.Tensor` objects. While NumPy is the only strict dependency of KeOps, PyTorch is a recommended module due to its richer GPU integration. In particular, the PyTorch backend offers the most comprehensive support: it handles `torch.cuda.Tensor` types in FP64, FP32, and FP16 precisions, avoids unnecessary data transfers between host and device, and integrates seamlessly with PyTorch's AD engine. That said, PyTorch — being primarily designed for machine learning — lacks some advanced linear algebra capabilities provided by the SciPy library, which is based on NumPy.

The central element of PyKeOps is the `Genred()` class. Given a formula — either constructed using symbolic tensors or provided directly by the user — it queries the `keopscore` module to trigger JIT compilation. The resulting binary kernel is then passed to the `pykeops_nvrtc` gateway that bridges PyKeOps with the Nvidia driver, which launches the computation directly on the GPU. Meanwhile, PyKeOps inspects the location of the input tensors (CPU or GPU memory), performs memory transfers if necessary, and allocates memory for the output before executing the reduction.

RKeOps binder for R

The RKeOps binder was completely rewritten with the release of KeOps version 2. In version 1, it relied on the Rcpp library [22] to directly trigger the compilation of binaries for given formulas. While Rcpp is the standard approach to link R code with C++/CUDA, it imposes significant maintenance overhead: any change in the core KeOps API required synchronized updates across all binders (including Python and MATLAB).

With the full refactoring and rewrite of the metaprogramming engine in `keopscore`, RKeOps was redesigned to delegate the compilation and GPU interfacing to PyKeOps, using the excellent Reticulate library [76]. Although this introduces some installation overhead — such as creating a Python virtual environment — this is offset by significantly reducing the amount of R-specific implementation, thereby simplifying development. For example, the memory layout mismatch between R (column-major, inherited from FORTRAN) and Python/C (row-major) is handled directly by Reticulate. RKeOps uses the NumPy backend of PyKeOps exclusively, as R matrices reside in CPU memory.

Version 2 of RKeOps implements its own version of symbolic tensors, called `Lazytensor`[27]. The syntax has been designed to closely mirror that of PyKeOps, as shown in Listing 3. This alignment

[27] See https://github.com/getkeops/keops/blob/main/rkeops/R/lazytensor_preprocess.R

```r
library("rkeops")
Gaussian_Kernel <- function(x, y, sigma2) {
  # Turn our Tensors into KeOps symbolic variables:
  x_i <- LazyTensor(x, "i")        # (M, D) symbolic object
  y_j <- LazyTensor(y, "j")        # (N, D) symbolic object

  D_ij <- sum((x_i - y_j)^2)       # (M, N) symbolic matrix of pairwise squared distances
  K_ij <- exp(- D_ij / sigma2)     # (M, N) symbolic matrix

  return(K_ij)
}

X <- matrix(runif(nx*D), nrow=M)   # matrix M x D
Y <- matrix(runif(ny*D), nrow=N)   # matrix N x D
P <- matrix(runif(ny*D), nrow=N)   # matrix N x D

V <- sum(Gaussian_Kernel(X, Y, s) * LazyTensor(B, "j"), index="j")
```

Listing 3: An example of RKeOps code, similar to Listing 1

is convenient in the short term, enabling users to switch easily between languages. However, it poses long-term maintenance risks[28].

[28] We warmly recommend the article https://www.construct.net/en/blogs/ashleys-blog-2/reality-long-term-software-1892 for insights on software maintenance challenges.

Conclusion and Perspectives

Position/features data analysis in biological data

The work presented in this thesis has consistently been driven by the need to develop original methods for the practical analysis of specific measurement (feature) with a geometric component (positions): the OCT dataset for glaucoma detection using functional shapes (fshapes), and the ADNI MRI dataset for understanding the progression of Alzheimer's disease through longitudinal data analysis. The models introduced here, which can be seen as extension of the LDDMM framework, are mathematically grounded, with results ensuring the existence and smoothness of solutions to the posed problems. The implementations have been optimized to accommodate a generic formulation and remain scalable to large datasets. All software developed is freely available and adheres to open science principles. Nevertheless, routinely analyzing high-dimensional shapes using this family of methods on small- to medium-sized datasets[29] — as is often the case in clinical settings — or applying such models outside the immediate research community presents ongoing difficulties. In this section, I outline several perspectives for future work in two main directions aimed at addressing this limitation.

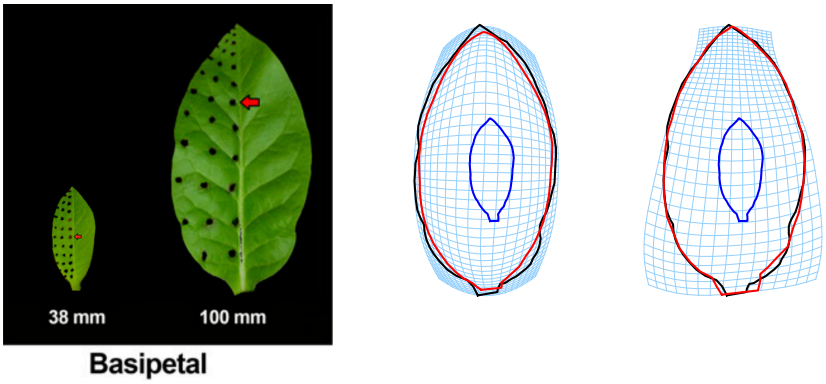
The first direction concerns the biological relevance of deformation models. In practice, shape analysis methods often rely on arbitrary default choices — such as the use of radial Gaussian RKHS — which are rarely adapted to the specific data at hand. This is due to several factors: computational demands, the limited size of available datasets, or simply inherited conventions from earlier work. A current line of research for me involves incorporating physical or biological constraints into these models, with the aim of generating more realistic deformations and narrowing the gap with mechanical models.

Secondly, I will present some perspectives on the analysis of emerging data modalities. Recent advances, particularly in omics sequencing technologies, have opened up new opportunities. Beyond single-cell data, it is now possible to measure gene expression within tissue samples while preserving spatial resolution. This makes it theoretically possible to compare biological tissues across scales, from molecular to phenotypic. Developing deformation models to preprocess and analyze such data is both a challenging and exciting task.

Modular deformations

Using standard non-rigid deformation models to study the evolution of biological shapes may lead to unrealistic results. Incorporating biomechanical constraints can help produce more coherent and biologically plausible deformations. The estimation of deformation in a leaf groth model is a good example as illustrated in Figure 3.5.

[29] With very few notable exceptions, which are often limited to highly specific data and models; see, e.g., https://mricloud.org/.



(a) Raw data [17] (b) LDDMM (c) IMODAL

Figure 3.5: Curve registration. The source curve is shown in blue, the target in green, and the deformed source in red. The unstructured deformation model (b) fails to capture basipetal growth, in contrast to the constrained deformation model (c).

The deformation modules introduced in the PhD thesis [34] of B. Gris (CNRS) provide a framework for generating non-rigid deformations under arbitrary constraints [35]. The core idea is a modular approach that allows users to construct deformation models by combining multiple elementary components that evolve jointly. This enables the generation of a wide range of deformations while keeping the overall model both simple and interpretable [Proc9]. In particular, the evolution between two shapes can be modeled using a non-rigid deformation that satisfies specific **elastic constraints**. These are known as implicit deformation modules, and their theoretical development, using the space of featured landmark, is detailed in [PreP1]. As in Chapter 1, a kinetic energy is naturally associated with a given collection of modules, and modeling shape evolution then becomes an optimal control problem.

One of the long-term motivations behind this work is the modeling of cortical growth to better understand the variability of folding patterns, which is a key phenotype in human neuroimaging. A common assumption in biomechanical folding models is that these patterns are partly determined by mechanical constraints [75]. The idea of applying modular deformation models to species such as the ferret is currently being discussed in collaboration with B. Gris and R. Torro (Institut Pasteur). As a first step, we aim to adapt the framework of implicit deformation modules to more accurately capture cortical gyrification. This will involve designing modules specifically tailored to this process and investigating the influence of their parameters.

To study surfaces derived from real datasets, it will also be necessary to improve the computational efficiency and robustness of the current algorithms. The main implementation has been carried out in the IMODAL software [Soft4], which is still under active development. Particularly we plan to focus on the following areas:

Scalability: Improving computational efficiency is essential for processing large datasets. The main bottleneck lies in the matrix inversion step required for computing modular geodesic shooting. Currently, we use a conjugate gradient method to solve the linear system, but a key question is how to design effective preconditioners that are compatible with the KeOps framework. Theoretical results on convergence rates [31] suggest that good preconditioners should be full-rank while having a small number of distinct eigenvalues.

Statistical robustness: The estimation of optimal trajectories and growth parameters suffers from identifiability issues. To address this, we plan to employ appropriate regularization

techniques — such as L^1 regularization for sparsity — to ensure that the resulting trajectories are both biologically meaningful and interpretable [21]. Another promising approach is to adopt a multiscale framework, in which deformation modules are computed at different resolutions. Once the full registration is obtained, selecting modules at a specific scale allows us to filter out the influence of other scales and better capture complex deformation patterns, as illustrated in Section 3.2 of [Proc9].

Practical usability: An open challenge is to move beyond the proof-of-concept implementations provided in the IMODAL documentation and make these models more accessible to practitioners. While generating and positioning modules in 2D datasets is feasible using Python scripts, the process becomes tedious in 3D due to the lack of effective navigation tools. A potential solution is to develop a user interface — possibly interactive — that enables users to explore the dataset intuitively and select the most relevant elements from a dictionary of modules.

Analysis of spatial transcriptomic data

Spatial transcriptomics is a recent technology that enables precise measurement of both gene expression (features) and its spatial localization within a tissue sample. As is often the case with emerging technologies, multiple standards currently coexist — some with short lifespans — making it challenging to design a versatile yet scalable model formulation. The goal here is to develop methods capable of comparing these various subcellular-scale datasets by establishing a common reference coordinate system, and enabling analysis across scales. For example, to study the spatial organization of gene expression in the mouse brain, Figure 3.6 illustrates the registration of brain atlas regions (Allen CCFv3) onto a spatial transcriptomics data (MERFISH). It gives a more accurate interpretation of gene expression patterns within the context of brain structure.

The mathematical treatment of such data is particularly challenging, as it combines discrete information (e.g., gene expression levels, atlas ontology) with continuous spatial localization. I became interested in this topic in 2023 while collaborating with K. Stouffer, a visiting PhD student from CIS, on the analysis of spatial transcriptomics data from the mouse brain across multiple modalities (MERFISH and BARSeq). This research area is especially appealing to me, as it aligns closely with the focus of the MIAT laboratory (INRAE), which I recently joined. It provides a natural bridge between omics data — widely studied at MIAT — and spatial information, which lies at the heart of shape analysis. I plan to further develop this line of research in the coming years, and briefly describe below the direct continuation of our work presented in the preprint [Art15].

In [74], a deformation model framework was proposed based on a Varifold representation of the acquired data. This approach is particularly promising, as kernel-based distances can effectively handle heterogeneous data—such as gene expression and subcortical anatomical regions from the atlas—by capturing shared position/feature patterns. The framework introduces a norm $\|\cdot\|_{W^*}$ to measure the similarity between varifolds, defined analogously to Section 1.5.3.

In a spatial transcriptomics acquisition, each mRNA read, indexed by i, is represented as a single weighted Dirac mass in the varifold space: $w_i\delta_{x_i} \otimes p_i$, where $w_i\delta_{x_i}$ encodes the spatial location with gene expression intensity $w_i > 0$, and p_i is a conditional probability distribution over a D-dimensional feature space, capturing gene expression characteristics a $x_i \in \mathbb{R}^3$. A full acquisition comprising N measurements is then represented by a weighted sum of Dirac masses:

$$\mu = \sum_{i=1}^{N} w_i\delta_{x_i} \otimes p_i.$$

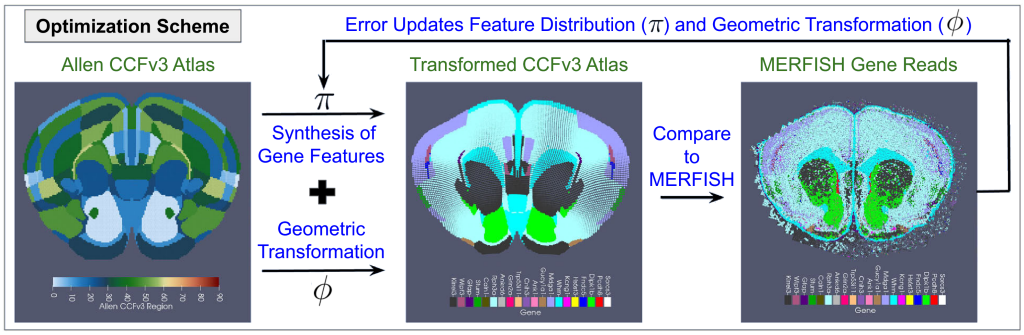


Figure 3.6: Registration of a brain atlas (Allen CCFv3) onto spatial transcriptomics data (MERFISH) from [74]. The left image shows the original atlas (source), and the right image displays the MERFISH gene expression data (target). The middle image presents the geometrically transformed atlas, now equipped with gene features π, allowing direct comparison with the MERFISH data.

The registration problem between two varifold μ and μ' optimizes over the diffeomorphism:

$$\begin{cases} \inf_{\phi \in Diff} \text{penalty}(\phi) + \|\phi \cdot \mu - \mu'\|^2_{W^*} \\ \text{with } \phi \cdot \mu = \sum_{i=1}^{N} |d_{x_i}\phi|\, w_i \delta_{\phi(x_i)} \otimes p_i \end{cases} \tag{3.3}$$

where action of diffeomorphisms on varifolds involves the Jacobian determinant $|d_{x_i}\phi|\, w_i$ of the transformation ϕ, ensuring that total gene expression remains coherent across the deformation. In practice, an additional spatial mask is applied to account for the partial acquisition typically encountered in spatial transcriptomics data [Art15].

When registration is performed between two different modalities, the formulation of problem (3.3) must be adapted to account for the differences in feature spaces. In the CCFv3-to-MERFISH example shown in Figure 3.6, it is assumed that for each atlas region ℓ, there exists a distribution π_ℓ over the target feature space (i.e., the set of genes detected by MERFISH) that characterizes the typical gene expression profile for that region. The underlying idea is that different tissue types are distinguished by different gene expression patterns. These region-specific distributions π_ℓ are estimated during the registration process, along with the spatial deformation. To regularize this estimation, an additional penalty term is introduced, taking the form of a Kullback–Leibler divergence.[30]

A full acquisition can reach up to $N = 6 \cdot 10^9$ mRNA reads over a feature space of dimension $D = 1000$. Standard kernel methods struggle to scale to such high-dimensional and high-volume data, as commonly found in transcriptomics. To address this, practical solutions have been implemented in [Art15], where the data is approximated by a lower-dimensional varifold representation using the KeOps library. Nonetheless, even with this approximation, the number of points N can still reach tens of millions, with feature dimensions D around 100 — pushing the upper bounds of KeOps' efficient performance.

Future work in the direct lineof [Art15] will focus on two main directions. First, we aim to reduce the computational complexity of kernel operations while preserving the current syntax, by implementing approximate kernel methods using the ranges framework, inspired by fast multipole methods. This approach must include theoretical bounds on the approximation error to ensure reliable performance.

Second, we plan to enhance computational efficiency for high-dimensional feature spaces, particularly when the number of dimensions exceeds $D > 100$. Current parallel reduction strategies

[30] Note: There is a notation switch between p and π in Equation (3) in [74] and Equation (12) in [Art15].

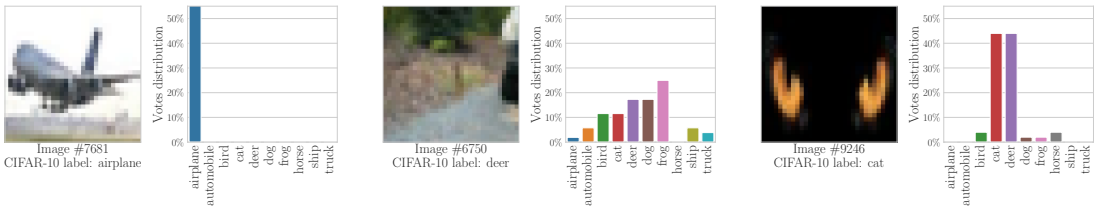
in KeOps are not well-suited for such settings (see Section 3.2.3). To address this, we propose developing a more sophisticated reduction scheme based on a 2D computational grid that leverages the larger cache memory available in modern GPUs. This design is expected to significantly accelerate computations, potentially scaling up to dimensions around 1000. However, going beyond this threshold will likely saturate memory resources and remains a challenging optimization problem.

Crowd sourced data in machine learning

This section presents a topic rather independant from the previous ones. This is about machine learning classification of images with crowd-sourced data.

Context

I have been working on the treatment of crowd-sourced data for machine learning with J. Salmon since 2020. In supervised classification, one assumes a sample of n pairs (x_i, y_i), where $x_i \in \mathcal{X}$ (typically images) and $y_i \in \mathcal{Y}$ is the corresponding label. In crowdsourcing, however, each x_i is labeled by multiple, often non-expert, annotators. The true label y_i is unknown; instead, we observe a distribution $\hat{y}_i$ over $\mathcal{Y}$ (see Figure 3.7). When there is strong consensus, $\hat{y}_i$ becomes a Dirac mass, recovering the standard supervised setting. Such data have become increasingly common with the rise of participatory projects.



(a) True label: `airplane` (b) True label: `deer` (c) True label: `cat`

Figure 3.7: Three examples from the CIFAR-10H dataset [64], a dataset obtained by re-labeling CIFAR-10 using Amazon Mechanical Turks service with a variable number of voters for each image. (a) `airplane` is easy to classify. (b) Poor image quality makes the task ambiguous. (c) A black cat (`cat`) is confused with the antlers of a deer (`deer`).

With A. Joly (INRIA) and J. Salmon, we co-supervised T. Lefort during his Master's and PhD [45]. His research focused on detecting label ambiguity in crowd-sourced data to identify ambiguous tasks assigned to workers [Art13]. He also developed PeerAnot, a framework for handling crowdsourced image classification datasets and benchmarking aggregation strategies (i.e., how to infer labels y_i from worker votes $\hat{y}_i$) [Art12]. This work was part of a larger project involving Pl@ntNet[31], a citizen science platform for plant identification [Art14].

The following section outlines the future of this project (funding application planned for 2025) and the main research directions we intend to pursue. The project aims to improve data quality collected via crowdsourcing and enhance machine learning performance in this setting.

[31]Available at `https://plantnet.org/`

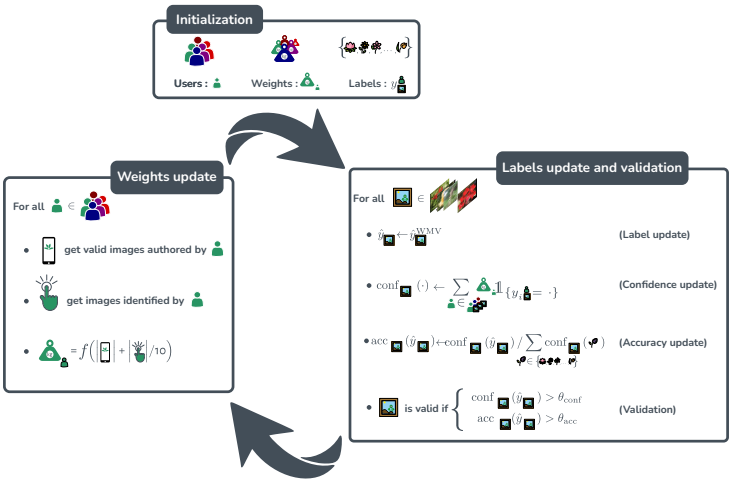


Figure 3.8: Pl@ntNet label validation scheme

The Pl@ntNet project

Advances in machine learning and citizen science are introducing new tools to applied fields like ecology [26]. Platforms such as Pl@ntNet, iNaturalist, and eBird combine amateur and expert contributions to train deep neural networks (DNNs) and offer flexible tools for species identification, benefiting both the general public and experts. However, challenges emerge due to the nature of the collected data and the need to quantify uncertainty in predictions.

A common challenge for these approaches is providing the general public with efficient and reliable tools, despite the heterogeneity of collected labels. A major weakness is the lack of statistical guarantees for DNN predictions, compounded by the uncertainty of user-collected labels, as user expertise varies.

A key example is Pl@ntNet, developed by members of this consortium over 15 years, with a community of 25 million users. Primarily a mobile application, users photograph plants, and the app suggests possible species. The underlying machine learning relies on DNNs trained on extensive image datasets to deliver predictions in milliseconds.

The Pl@ntNet platform enables contributors to validate plant observations through a voting system that incorporates a label aggregation strategy. Users can vote on the identification of plant observations submitted by others. To improve data reliability, Pl@ntNet estimates each user's expertise through a confidence score, which is recursively calculated on the basis of the correctness of their identifications (see Figure 3.8). Observations receive a consensus-based label when enough trusted users agree. This ensures that inputs of botanical experts are properly weighted while preventing noisy or unreliable data from being used in AI training.

Although the application is already successful, the quality of the collected data and the predictions made by the DNN are not always reliable. Errors in DNN predictions can mislead users during labeling, degrading the quality of the collected data. The goal of this project is to provide a statistical framework and software to improve, assess, and robustly validate DNN predictions in this context.

User's confusion matrix estimation at scale

In applications like Pl@ntNet though (that can handle more than 75,000 species), many classes have very few observations, as illustrated in Figure 3.9. One potential use of confusion matrices

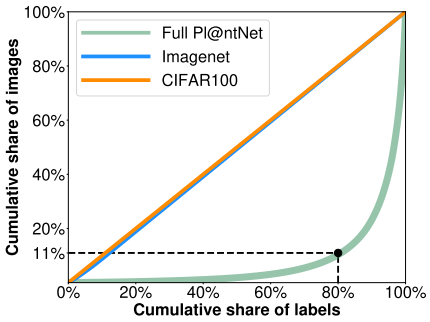
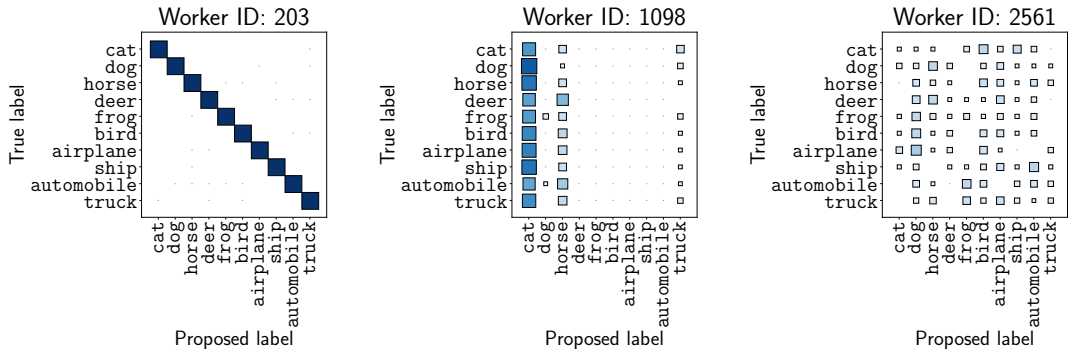


Figure 3.9: Pl@ntNet has a long-tail label distribution (unlike CIFAR-10 and Imagenet): for instance 80% of species are obtained for 11% of the images $\iff$ 20% of species represents 89% of the total images.



(a) Good worker: very few confusions

(b) Bad worker: cat, cat, cat, ..

(c) Bad worker: random, random, ...

Figure 3.10: Examples of confusion matrices, estimated with the Dawid and Skene algorithm [18], on CIFAR-10H. The larger the square, the larger are the coefficients of the confusion matrices.

(see Figure 3.10) is to group species with similar confusion patterns, such as similar number of confusions or entropy. The optimal decision threshold is likely similar for these species. Additionally, model uncertainty related to the number of images must be considered. Visually similar species are challenging to distinguish, but those with more training data are likely to be predicted more often due to better representation. Using a confusion matrix could enable re-weighting overpredictions for common species, thereby favoring rare species crucial for assessing habitat health.

A crucial modeling tool in this context is the confusion matrix, which assesses the likelihood of one label being mistaken for another. These matrices are essential for understanding label quality and enhancing image recommendations for labeling by users. While applicable at the project scale, they often need to be tailored to individual users to incorporate their expertise, potentially with a time-dependent component. Given the scale of the problem, managing over 25 million confusion matrices (one per user) poses significant statistical and computational challenges, especially with a large number of labels. A scalable solution is required, relying on efficient algorithms and compact structures like sparse matrices to process these matrices and recommend images for labeling.

The propose research direction is user's confusion matrix estimation at scale. To improve label accuracy, a scalable method for estimating user-specific confusion matrices will be developed.

These matrices assess the likelihood of mislabeling and must be efficiently computed for a large-scale platform like Pl@ntNet, where over 25 million users contribute to more than 75K labels. The task will involve designing efficient algorithms and compact data structures (such as sparse matrices) to model individual user expertise while possibly integrating time-dependent factors to track improvements over time.

APPENDIX A

Scientific Production

A.1 List of Publications

In this section, I present an exhaustive list of my work and links to the full texts. The abstract is reproduced when the work is not yet accessible online.

Peer-Reviewed Journals

[Art1] J. Bigot et B. Charlier. **On the consistency of Fréchet means in deformable models for curve and image analysis**. *Electronic Journal of Statistics*, Vol 5: 1054–1089, 2011. http://projecteuclid.org/euclid.ejs/1316092868

[Art2] B. Charlier. **Necessary and sufficient condition for the existence of a Fréchet mean on the circle**. *ESAIM: Probability and Statistics*, Vol 17: 635–649, 2013. http://dx.doi.org/10.1051/ps/2012015

[Art3] B. Charlier, N. Charon, A. Trouvé. **The fshape framework for the variability analysis of functional shapes**. *Foundations of Computational Mathematics*, Vol 17: 287–357, 2017. http://link.springer.com/article/10.1007/s10208-015-9288-2

[Art4] S. Lee, N. Charon, B. Charlier, K. Popuri, E. Lebed, M. Sarunic, A. Trouvé, F. Beg. **Atlas-based Shape Analysis and Classification of Retinal Optical Coherence Tomography Images using the Functional Shape (fshape)**. *Medical Image Analysis*, Vol 35: 570–581, 2016. http://www.sciencedirect.com/science/article/pii/S1361841516301608

[Art5] S. Lee, M.L. Heisler, K. Popuri, N. Charon, B. Charlier, A. Trouvé, P. J. Mackenzie, M. V. Sarunic, M. F. Beg. **Age and glaucoma-related changes in retinal nerve fiber layer and choroid: point-wise analysis and visualization using functional shapes registration**. *Frontiers in Neuroscience*, Vol 11, 2017. http://journal.frontiersin.org/article/10.3389/fnins.2017.00381

[Art6] N. Charon, B. Charlier, A. Trouvé. **Metamorphoses of functional shapes in Sobolev spaces**. *Foundations of Computational Mathematics*, Vol 18: 1535–1596, 2018. https://link.springer.com/article/10.1007/s10208-018-9374-3

[Art7] M. Louis, B. Charlier, P. Jusselin, S. Pal, S. Durrleman. **A Fanning Scheme for the Parallel Transport Along Geodesics on Riemannian Manifolds**. *SIAM Journal on Numerical Analysis*, Vol 56(4): 2563–2584, 2018. https://epubs.siam.org/doi/abs/10.1137/17M1130617

[Art8] P. Piras, V. Varano, M. Louis, A. Profico, S. Durrleman, B. Charlier, F. Milicchio, L. Teresi. **Transporting Deformations of Face Emotions in the Shape Spaces: A Comparison of Different Approaches**. *J Math Imaging Vis*, Vol 63: 875–893, 2021. https://doi.org/10.1007/s10851-021-01030-6

[Art9] B. Charlier, J. Feydy, J. Glaunès, F.D. Collin, G. Durif. **Kernel Operations on the GPU, with Autodiff, without Memory Overflows**. *Journal of Machine Learning Research*, Vol 22(74): 1–6, 2021. https://jmlr.org/papers/v22/20-275.html

[Art10] I. Koval, A. Bône, M. Louis, T. Lartigue, S. Bottani, A. Marcoux, J. Samper-González, N. Burgos, B. Charlier, A. Bertrand, S. Epelbaum, O. Colliot, S. Allassonnière, S. Durrleman. **AD Course Map charts Alzheimer's disease progression**, *Nature Scientific Reports*, Vol 11(8020), 2021. https://doi.org/10.1038/s41598-021-87434-1

[Art11] G. Nardi, B. Charlier, A. Trouvé. **The matching problem between functional shapes via a BV-penalty term: a Γ-convergence result**, *Interfaces and Free Boundaries*, Vol 26(3): 381–414, 2024. https://arxiv.org/abs/1503.07685

[Art12] T. Lefort, B. Charlier, A. Joly, J. Salmon. **Peerannot: classification for crowdsourced image datasets with Python**, *Computo*, 2024. https://openreview.net/pdf?id=lhhmJFa2bU

[Art13] T. Lefort, B. Charlier, A. Joly, J. Salmon. **Identify ambiguous tasks combining crowdsourced labels by weighting Areas Under the Margin**, *TMLR*, 2024. https://arxiv.org/pdf/2209.15380.pdf

[Art14] T. Lefort, A. Affouard, P. Bonnet, B. Charlier, A. Joly, J. Salmon. **Cooperative learning of Pl@ntNet's Artificial Intelligence algorithm**, *Methods in Ecology and Evolution*, 2025. https://doi.org/10.1111/2041-210X.14486

[Art15] K. M. Stouffer, X. Chen, H. Zeng, B. Charlier, L. Younes, A. Trouvé, M. I. Miller. **xIV-LDDMM Toolkit: A Suite of Image-Varifold Based Technologies for Representing and Mapping 3D Imaging and Spatial-omics Data Simultaneously Across Scales**, Nature Communication in Biology , 2025 (To appear). https://www.biorxiv.org/content/10.1101/2024.11.04.621983v1

Conference proceedings

[Proc1] A. Routier, M. Prastawa, B. Charlier, C. Doucet, J. Glaunès, S. Durrleman. **Deformetrica: a software for statistical analysis of anatomical shapes**. *OHBM 2015*. https://hal.archives-ouvertes.fr/hal-01187469

[Proc2] I. Kaltenmark, B. Charlier, N. Charon. **A general framework for curve and surface comparison and registration with oriented varifolds**. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2017. http://openaccess.thecvf.com/content_cvpr_2017/papers/Kaltenmark_A_General_Framework_CVPR_2017_paper.pdf

[Proc3] J. Feydy, B. Charlier, F.-X. Vialard, G. Peyré. **Optimal Transport for Diffeomorphic Registration**. *MICCAI 2017*. https://hal.archives-ouvertes.fr/hal-01540455

[Proc4] K. Kumar, P. Gori, B. Charlier, S. Durrleman, O. Colliot, C Desrosiers. **White Matter Fiber Segmentation Using Functional Varifolds**. *6th MICCAI Workshop on Mathematical Foundations of Computational Anatomy*, 2017. https://hal.inria.fr/hal-01589649

[Proc5] A. Bône, M. Louis, A. Routier, J. Samper, M. Bacci, B. Charlier, O. Colliot, S. Durrleman. **Prediction of the progression of subcortical brain structures in Alzheimer's disease from baseline**. *6th MICCAI Workshop on Mathematical Foundations of Computational Anatomy*, 2017. https://hal.inria.fr/hal-01563587

[Proc6] B. Charlier, J. Feydy, J. Glaunès, A. Trouvé. **An efficient kernel product for automatic differentiation libraries, with applications to measure transport**, GFW03, 2017. http://www.math.ens.fr/~feydy/Talks/GFSW03_2017/GFSW_Cambridge_2017_workingversion.pdf

[Proc7] M. Louis, R. Couronné, I. Koval, B. Charlier, S. Durrleman. **Riemannian geometry learning for disease progression modelling**. *26th international conference on Information Processing in Medical Imaging (IPMI)*, 2019. https://hal.archives-ouvertes.fr/hal-02079820/document

[Proc8] J. Feydy, B. Charlier, J. Glaunès, M. Bronstein. **Fast geometric learning with symbolic matrices**, *Advances in Neural Information Processing Systems 33 (NeurIPS 2020)*, 2020. https://proceedings.neurips.cc/paper/2020/file/a6292668b36ef412fa3c4102d1311a62-Paper.pdf

[Proc9] L. Lacroix, B. Charlier, A. Trouvé, B. Gris. **IMODAL: creating learnable user-defined deformation models**, Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2021. https://openaccess.thecvf.com/content/CVPR2021/papers/Lacroix_IMODAL_Creating_Learnable_User-Defined_Deformation_Models_CVPR_2021_paper.pdf

[Proc10] T. Moreau, M. Massias, A. Gramfort, Alexandre, P. Ablin, P.A. Bannier, B. Charlier, M. Dagréou, T. Dupré la Tour, G. Durif, C. F. Dantas, Q. Klopfenstein, J. Larsson, E. Lai, T. Lefort, B. Malézieux, B. Moufad, B. T. Nguyen, A. Rakotomamonjy, Z. Ramzi, J. Salmon, S. Vaiter. **Benchopt: Reproducible, efficient and collaborative optimization benchmarks**, *Advances in Neural Information Processing Systems (NeurIPS 2022)*, 2022. https://openreview.net/forum?id=1uSzacpyWLH

[Proc11] K. Stouffer, X. Chen, M. Rue, A. Trouvé, B. Charlier, M. Miller. **3D Cross-Modality Mapping of Tissue Scale Brain Atlas to Cellular Scale Spatial Transcriptomics**, *Cosyne*, 2024.

Book chapter

[Chap1] N. Charon, B. Charlier, J. Glaunès, P. Gori, P. Roussillon. **Fidelity metrics between curves and surfaces: currents, varifolds, and normal cycles**. *Riemannian Geometric Statistics in Medical Image Analysis, Academic Press*, 2020. https://www.sciencedirect.com/science/article/pii/B9780128147252000212

Preprint

[PreP1] B. Charlier, B. Gris, A. Redjimi, A. Trouvé. **Structured Deformation Modeling with Implicit Deformation Modules**.

Abstract: This paper presents a comprehensive theoretical exploration of implicit deformation modules and their application in shape space and registration. The implicit deformation module framework

enables the generation of advanced structured deformations in a user-friendly and efficient manner. To establish the well-posedness of deformations generated by these modules and to show the existence of solutions to the resulting registration problem, we introduce the featured landmark shape space. This concept extends the classical notion of shape by incorporating a semi-direct group action on the space of locations and features. These contributions bridge the gap left by the paper [Proc9] introducing IMODAL software [Soft4], which provides an implementation of some deformation modules presented here (explicit and implicit of order one) but lacks advanced theoretical guarantees. Finally, we demonstrate how complex structured deformation models can be estimated from data, supported by practical examples.

A.2 List of Softwares

Here is a list of the most important projects I have participated in as the lead developer (unless explicitly stated otherwise).

[Soft1] **FshapesTk**

FshapesTk est une boîte à outils qui permet l'estimation d'atlas de formes fonctionnelles (*fshapes*). À partir d'un échantillon de formes fonctionnelles observées, un atlas consiste à calculer une forme fonctionnelle moyenne ainsi que les déformations de cette moyenne vers les observations. Bien qu'écrite principalement en Matlab, fshapesTk est rapide car les parties coûteuses du code sont codées en Cuda et peuvent être exécutées sur un GPU.

Ce code contient une implémentation des métamorphoses de formes fonctionnelles tangentielles et riemanniennes, distances entre formes (courants et varifolds fonctionnels, distance du transport optimal avec régularisation entropique), *etc.* C'est avec ce logiciel qu'ont été générées les simulations numériques des publications associées.

Auteurs: B. Charlier
Publications associées: [Art3, Art4, Art5, Art6, Art11], [Proc2, Proc3, Proc4]
Langage: Matlab, C/Cuda
Taille: $16\,000$ lignes de codes
Maturité: Stable. Le code a été développé entre 2012 et 2017
Codes: `https://plmlab.math.cnrs.fr/benjamin.charlier/fshapesTk/`

[Soft2] **Deformetrica**

Deformetrica est aussi un code pour l'analyse de formes (nuages de points, courbes, surfaces) qui possède des fonctionnalités complémentaires à fshapesTk. Il permet de manipuler des données géométriques longitudinales. Il est conçu principalement pour:

1. Le recalage (*matching* ou *registration*): calcule la déformation la moins coûteuse pour apparier un objet sur un autre.
2. La construction d'atlas: calcule une forme moyenne à partir d'un échantillon d'objets géométriques, ainsi que les déformations entre cette moyenne et chacune des données.
3. La régression géodésique: à partir des observations d'un objet à des temps différents, il calcule la déformation de cet objet au plus près possible des différentes observations.

Auteurs: Projet initié par s. Durrleman. Une liste complète des contributeurs peut être trouvée à la page `http://www.deformetrica.org/#contributors`. Les quatre développeurs principaux sont A. Bône, B. Charlier, M. Louis et B. Martin
Publications associées: [Art7, Art8, Proc1, Proc5]
Langage: Python, C++, C/Cuda
Taille: $35\,000$ lignes de codes
Maturité: Stable. Portage en `Python` en 2017. N'est plus activement maintenu.
Utilisateurs connus: quelques milliers de téléchargements
Codes: `https://gitlab.com/icm-institute/aramislab/deformetrica`
Web: `http://www.deformetrica.org/` (hors ligne en 2025)

[Soft3] **KeOps**

KeOps (*Kernel Operations*) pour calculer des opérations dans les espaces à noyau à l'aide d'un GPU. Le but de cette bibliothèque est de fournir des fonctions simples et optimisées dans les principaux langages scriptés pour faire ce type de calculs.

La motivation de départ est le calcul rapide de convolutions discrètes (et de leurs dérivées) apparaissant dans les méthodes utilisant des espaces de Hilbert à noyau auto-reproduisant. Les applications sont multiples: elles touchent tous les domaines utilisant des méthodes à noyaux (apprentissage, estimation de densité, estimation d'atlas, etc.).

Dans les versions `v1.x`, le moteur de génération de code de KeOps était développé directement en C++/Cuda avec des *bindings* en Python, Matlab et R. Depuis 2022 et les versions `v2.x`, le moteur de méta-programmation a été réécrit en Python. Cela a permis des gains très substantiels de performance pour les temps de compilation et une meilleure compatibilité sur les serveurs de calcul ne disposant pas d'une pile de compilation aisément accessible.

Le code Cuda généré par KeOps à partir des formules données par l'utilisateur est particulièrement efficace car il utilise une implémentation dite 'par tuile' permettant de conserver une empreinte mémoire en $O(N)$ (contrairement aux bibliothèques tensorielles standards qui stockent de l'ordre de $O(N^2)$ éléments). La bibliothèque KeOps dispose aussi d'un module de différentiation automatique permettant de générer automatiquement du code Cuda optimisé à partir d'une formule symbolique fournie par l'utilisateur au moment de la compilation.

Le développement de KeOps utilise les outils modernes: système de version git, documentation générée automatiquement à partir des *docstrings* avec Sphinx, intégration continue, JIT via *Nvidia Runtime Compiler* (`nvrtc`), etc.

Auteurs: B. Charlier, J. Feydy, J. Glaunès
Langage: C++/Cuda, Python, R
Taille: $60\,000$ lignes de codes
Maturité: Stable. Développement actif: des fonctionnalités sont ajoutées au fil de l'eau
Publications associées: [Proc6, Proc8, Proc9], [Chap1], [Art9]
Code: `https://github.com/getkeops/keops/`
Web: `https://www.kernel-operations.io/`
Utilisateurs connus: plus de $750\,000$ téléchargements à ce jour

[Soft4] **IMODAL**

IMODAL est un module Python pour l'appariement des formes (courbes, maillages ou images)

avec de grandes déformations structurées. Les structures sont incorporées *via* des modules de déformation qui génèrent des champs de vecteurs choisis de types particuliers. Ils peuvent être définis explicitement (en générant des échelles ou des rotations locales par exemple) ou implicitement à partir de contraintes. En outre, il est possible de les combiner de manière à ce qu'une structure complexe puisse être facilement définie comme la superposition de structures simples. Les trajectoires de ces champs vectoriels modulaires peuvent ensuite être intégrées pour construire de grandes déformations modulaires. Leurs paramètres peuvent être optimisés pour enregistrer les formes observées et analysées.

Auteurs: Benjamin Charlier, Barbara Gris, Leander Lacroix, Alain Trouvé
Langage: Python (`Pytorch`)
Taille: 50 000 lignes de codes
Maturité: En développement
Publication associée: [Proc9], [PreP1]
Code: `https://github.com/imodal/imodal`
Web: `https://www.kernel-operations.io/im/`

[Soft5] **BenchOpt**

Benchopt est une solution logiciel pour comparer des algorithmes d'optimisation de manière simple, transparente et reproductible. Étant donné un programme d'optimisation (utilisant des données simulées ou réelles), les performances des solveurs installés sur la machine sont comparés et affichés de manière synthétique. Il est écrit en Python mais est disponible dans de nombreux langages de programmation: il a été testé avec Python, R, Julia et des binaires C/C++. Il peut être appelé en ligne de commande depuis un terminal.

Contrairement aux projets décrits ci-dessus, je suis contributeur de `BenchOpt` et non développeur principal.

Auteurs: T. Moreau, M. Massias, A. Gramfort, *et al.*
Langage: Python
Taille: 20 000 lignes de codes
Maturité: En développement
Publication associée: [Proc10]
Code: `https://github.com/benchopt/benchopt`
Web: `https://benchopt.github.io/`
Utilisateurs connus: plus de 75 000 téléchargements à ce jour

Bibliography

[1] G. Ahdritz, N. Bouatta, C. Floristean, S. Kadyan, Q. Xia, W. Gerecke, T. J. O'Donnell, D. Berenberg, I. Fisk, N. Zanichelli, B. Zhang, A. Nowaczynski, B. Wang, M. M. Stepniewska-Dziubinska, S. Zhang, A. Ojewole, M. E. Guney, S. Biderman, A. M. Watkins, S. Ra, P. Ribalta Lorenzo, L. Nivon, B. Weitzner, Y.-E. A. Ban, P. K. Sorger, E. Mostaque, Z. Zhang, R. Bonneau, and M. AlQuraishi. OpenFold: Retraining AlphaFold2 yields new insights into its learning mechanisms and capacity for generalization. https://lupoglaz.github.io/OpenFold2/, 2022.

[2] W. K. Allard. On the first variation of a varifold. *Annals of Mathematics*, 95(3):417–491, May 1972.

[3] S. Allassonnière, A. Trouvé, and L. Younes. Geodesic shooting and diffeomorphic matching via textured meshes. In Anand Rangarajan, Baba Vemuri, and Alan L. Yuille, editors, *Energy Minimization Methods in Computer Vision and Pattern Recognition*, pages 365–381, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[4] Jr. Almgren and J. Frederick. *Plateau's Problem: An Invitation to Varifold Geometry*. Mathematics Monograph Series. W. A. Benjamin, New York, 1966.

[5] S. Arguillère, M. I. Miller, and L. Younes. Diffeomorphic surface registration with atrophy constraints. *SIAM Journal on Imaging Sciences*, 9(3):975–1003, 2016.

[6] S. Arguillère, E. Trélat, A. Trouvé, and L. Younes. Shape deformation analysis from the optimal control viewpoint. *Journal de Mathématiques Pures et Appliquées*, 104(1):139–178, 2015.

[7] V. I. Arnold. Sur la géométrie différentielle des groupes de Lie de dimension infinie et ses application à l'hydrodynamique des fluides parfaits. *Ann. Inst. Fourier (Grenoble)*, 16:319–361, 1966.

[8] N. Aronszajn. Theory of reproducing kernels. *Trans. Amer. Math. Soc.*, 68:337–404, 1950.

[9] M. F. Beg, M. I. Miller, A. Trouvé, and L. Younes. Computing Large Deformation Metric Mappings via Geodesic Flows of Diffeomorphisms. *International Journal of Computer Vision*, 61(2):139–157, Feb 2005.

[10] A. Berlinet and C. Thomas-Agnan. *Reproducing Kernel Hilbert Spaces in Probability and Statistics*. Springer US, 2003.

[11] B. Buet, G. P. Leonardi, and S. Masnou. Weak and approximate curvatures of a measure: A varifold perspective. *Nonlinear Analysis*, 222:112983, 2022.

[12] B. Buet, G.P. Leonardi, and S. Masnou. Discretization and approximation of surfaces using varifolds. *Geometric Flows*, 3(1):28–56, 2018.

[13] N. Charon and A. Trouvé. Functional currents : a new mathematical tool to model and analyse functional shapes. *Journal of Mathematical Imaging and Vision*, 48:413–431, 2014.

[14] N. Charon and A. Trouvé. The Varifold Representation of Nonoriented Shapes for Diffeomorphic Registration. *SIAM Journal on Imaging Sciences*, 7(1):1–39, Feb 2014.

[15] J. Chevallier, V. Debavelaere, and S. Allassonnière. A coherent framework for learning spatiotemporal piecewise-geodesic trajectories from longitudinal manifold-valued data. *SIAM Journal on Imaging Sciences*, 14(1):349–388, 2021.

[16] M. Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 2292–2300, 2013.

[17] M. Das Gupta and U. Nath. Divergence in patterns of leaf growth polarity is associated with the expression divergence of mir396. *Plant Cell*, 27(10):2785–2799, Oct 2015.

[18] A.P. Dawid and A.M. Skene. Maximum likelihood estimation of observer error-rates using the EM algorithm. *JRSSC*, 28(1):20–28, 1979.

[19] W. E. Deming and F. F. Stephan. On a least squares adjustment of a sampled frequency table when the expected marginal totals are known. *The Annals of Mathematical Statistics*, 11(4):427–444, 1940.

[20] S. Durrleman, X. Pennec, A. Trouvé, and N. Ayache. A forward model to build unbiased atlases from curves and surfaces. In *Proc. of the International Workshop on the Mathematical Foundations of Computational Anatomy*, 2008.

[21] S. Durrleman, M. Prastawa, N. Charon, J. R. Korenberg, S. Joshi, G. Gerig, and A. Trouvé. Morphometry of anatomical shape complexes with dense deformations and sparse parameters. *NeuroImage*, 101:35–49, 2014.

[22] D. Eddelbuettel and R. Romain François. Rcpp: Seamless r and c++ integration. *Journal of Statistical Software*, 40(8):1–18, 2011.

[23] J. Feydy. *Analyse de données géométriques, au delà des convolutions*. PhD thesis, Université Paris-Saclay, 2020. Thèse de doctorat dirigée par Trouvé, Alain Mathématiques appliquées université Paris-Saclay 2020.

[24] R. Flamary, N. Courty, A. Gramfort, M. Z. Alaya, A. Boisbunon, S. Chambon, L. Chapel, A. Corenflos, K. Fatras, N. Fournier, L. Gautheron, N. T.H. Gayraud, H. Janati, A. Rakotomamonjy, I. Redko, A. Rolet, A. Schutz, V. Seguy, D. J. Sutherland, R. Tavenard, A. Tong, and T. Vayer. Pot: Python optimal transport. *Journal of Machine Learning Research*, 22(78):1–8, 2021.

[25] G. Flaubert. *Correspondance*, volume Volume 1. Louis Conard, 1926.

[26] Dilek Fraisl, Gerid Hager, Baptiste Bedessem, Margaret Gold, Pen-Yuan Hsing, Finn Danielsen, Colleen B Hitchcock, Joseph M Hulbert, Jaume Piera, Helen Spiers, et al. Citizen science in environmental and ecological sciences. *Nature reviews methods primers*, 2(1):64, 2022.

[27] A. Franquin. *Gaffe à Lagaffe*, volume 15 of *Gaston Lagaffe*. Marsu Productions, 1996.

[28] J. R. Gardner, G. Pleiss, D. Bindel, K. Q. Weinberger, and A. G. Wilson. Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration. In *Advances in Neural Information Processing Systems*, 2018.

[29] P. Gori, O. Colliot, Y. Worbe, L. Marrakchi-Kacem, S. Lecomte, C. Poupon, A. Hartmann, N. Ayache, and S. Durrleman. Bayesian atlas estimation for the variability analysis of shape complexes. In *MICCAI*, pages 267–274, 2013.

[30] R. Goscinny and A. Uderzo. *Astérix et Cléopâtre*, volume 6 of *Les Aventures d'Astérix le Gaulois*. Dargaud, 1965.

[31] A. Greenbaum. *Iterative methods for solving linear systems*, volume 17 of *Frontiers in applied mathematics*. Society for Industrial and Applied Mathematics, 1987.

[32] U. Grenander. *General Pattern Theory: A Mathematical Study of Regular Structures*. Clarendon Press, Oxford, 1993.

[33] A. Griewank and A. Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Society for Industrial and Applied Mathematics (SIAM), 2nd edition, 2008.

[34] B. Gris. *Approche modulaire sur les espaces de formes, géométrie sous-riemannienne et anatomie computationnelle*. Phd thesis, Université Paris-Saclay, 2016.

[35] B. Gris. Incorporation of a deformation prior in image reconstruction. *Journal of Mathematical Imaging and Vision*, 61:691–709, 2019.

[36] N. Guigui and X. Pennec. Chapter 8 - parallel transport, a central tool in geometric statistics for computational anatomy: Application to cardiac motion modeling. In F. Nielsen, S.R. Arni, and C. R. Rao, editors, *Geometry and Statistics*, volume 46 of *Handbook of Statistics*, pages 285–326. Elsevier, 2022.

[37] N. Guigui and X. Pennec. Numerical accuracy of ladder schemes for parallel transport on manifolds. *Foundations of Computational Mathematics*, 22:757–790, 2022.

[38] Y. Hu, T.-M. Li, L. Anderson, J. Ragan-Kelley, and F. Durand. Taichi: a language for high-performance computation on spatially sparse data structures. *ACM Trans. Graph.*, 38(6), November 2019.

[39] C. R. Jack, M. A. Bernstein, N. C. Fox, P. Thompson, Alexander G., D. Harvey, B. Borowski, P. J. Britson, J. L. Whitwell, C. Ward, A. M. Dale, J. P. Felmlee, J. L. Gunter, D. L.G. Hill, R. Killiany, N. Schuff, S. Fox-Bosetti, C. Lin, C. Studholme, C. S. DeCarli, G. Krueger, H. A. Ward, G. J. Metzger, K. T. Scott, R. Mallozzi, D. Blezek, J. Levy, J. P. Debbins, A. S. Fleisher, M. Albert, R. Green, G. Bartzokis, G. Glover, J. Mugler, and M. W. Weiner. The alzheimer's disease neuroimaging initiative (adni): Mri methods. *Journal of Magnetic Resonance Imaging*, 27(4):685–691, April 2008.

[40] W. Jakob, J. Rhinelander, and D. Moldovan. pybind11 – seamless operability between c++11 and python, 2017. https://github.com/pybind/pybind11.

[41] D. G. Kendall. Shape manifolds, procrustean metrics, and complex projective spaces. *Bull. London Math. Soc.*, 16(2):81–121, 1984.

[42] D. G. Kendall, D. Barden, T. K. Carne, and H. Le. *Shape and Shape Theory*. Wiley Series in Probability and Statistics. Wiley, 2009.

[43] A. Kheyfets, W. A. Miller, and G. A. Newton. Schild's Ladder Parallel Transport Procedure for an Arbitrary Connection. *International Journal of Theoretical Physics*, 39(12):2891–2898, 2000.

[44] S. Lee, S. X. Han, M. Young, M. F. Beg, M. V. Sarunic, and P. J. Mackenzie. Optic nerve head and peripapillary morphometrics in myopic glaucoma. *Investigative Ophthalmology & Visual Science*, 55(7):4378–4393, 07 2014.

[45] T. Lefort. *Label ambiguity in crowdsourcing and expert feedback*. Phd thesis, Univ. Montpellier, 2024.

[46] M. Lorenzi, N. Ayache, G. Frisoni, and X. Pennec. 4d registration of serial brain's mr images: a robust measure of changes applied to alzheimer's disease. *Spatio Temporal Image Analysis Workshop (STIA), MICCAI*, 2010.

[47] M. Lorenzi and X. Pennec. Geodesics, parallel transport & one-parameter subgroups for diffeomorphic image registration. *International journal of computer vision*, 105(2):111–127, 2013.

[48] M. Lorenzi and X. Pennec. Efficient parallel transport of deformations in time series of images: From schild to pole ladder. *Journal of Mathematical Imaging and Vision*, 50(1):5–17, 2014.

[49] J. Ma, M. I. Miller, and L. Younes. A bayesian generative model for surface template estimation. *International Journal of Biomedical Imaging*, 2010(1):974957, 2010.

[50] E. Maheux, I. Koval, J. Ortholand, C. Birkenbihl, D. Archetti, V. Bouteloup, S. Epelbaum, C. Dufouil, Hofmann-Apitius M., and Durrleman S. Forecasting individual progression trajectories in alzheimer's disease. *Nature Communications*, 14(1):761, 2023.

[51] K.V. Mardia and P.E. Jupp. *Directional Statistics*. Wiley Series in Probability and Statistics. Wiley, 2009.

[52] R. V. Marinescu, N. P. Oxtoby, A. L. Young, E. E. Bron, A. W. Toga, M. W. Weiner, F. Barkhof, N. C. Fox, A. Eshaghi, T. Toni, M. Salaterski, V. Lunina, M. Ansart, S. Durrleman, P. Lu, S. Iddi, D. Li, W. K. Thompson, M. C. Donohue, A. Nahon, Y. Levy, D. Halbersberg, M. Cohen, H. Liao, T. Li, K. Yu, H. Zhu, J. G. Tamez-Peña, A. Ismail, T. Wood, H. C. Bravo, N. Nguyen, M.and Sun, J. Feng, B.T. T. Yeo, G. Chen, K. Qi, S. Chen, D. Qiu, I. Buciuman, A. Kelner, R. Pop, D. Rimocea, M. M. Ghazi, M. Nielsen, S. Ourselin, L. Sørensen, V. Venkatraghavan, K. Liu, C. Rabe, P. Manser, S. M. Hill, J. Howlett, Z. Huang, S. Kiddle, S. Mukherjee, A. Rouanet, B. Taschler, B. D. M. Tom, S. R. White, N. Faux, S. Sedai, J. de Velasco Oriol, E. E. V. Clemente, K. Estrada, L. Aksman, A. Altmann, C. M. Stonnington, Y. Wang, J. Wu, V. Devadas, C. Fourrier, L. Lau Raket, A. Sotiras, G. Erus, J. Doshi, C. Davatzikos, J. Vogel, A. Doyle, A. Tam, A. Diaz-Papkovich, E. Jammeh, I. Koval, P. Moore, T. J. Lyons, J. Gallacher, J. Tohka, R. Ciszek, B. Jedynak, K. Pandya, M. Bilgel, W. Engels, J. Cole, P. Golland, S. Klein, D. C. Alexander, The EuroPOND Consortium , and The Alzheimer's Disease Neuroimaging Initiative . The alzheimer's disease prediction of longitudinal evolution (tadpole) challenge: Results after 1 year follow-up. *Machine Learning for Biomedical Imaging*, 1:1–60, 2021.

[53] G. Meanti, L. Carratino, L. Rosasco, and A. Rudi. Kernel Methods Through the Roof: Handling Billions of Points Efficiently. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[54] C.T. Metz, S. Klein, M. Schaap, T. van Walsum, and W.J. Niessen. Nonrigid registration of dynamic medical imaging data using nd + t b-splines and a groupwise optimization approach. *Medical Image Analysis*, 15(2):238 – 249, 2011.

[55] P. Michor and D. Mumford. Riemannian geometries on spaces of plane curves. *J. Eur. Math. Soc. (JEMS)*, 8(1):1–48, 2006.

[56] M. I. Miller, A. Trouvé, and L. Younes. Space-feature measures on meshes for mapping spatial transcriptomics. *Medical Image Analysis*, 93:103068, 2024.

[57] N. Miolane, N. Guigui, A. Le Brigant, J. Mathe, B. Hou, Y. Thanwerdas, S. Heyder, O. Peltre, N. Koep, H. Zaatiti, H. Hajri, Y. Cabanes, T. Gerald, P. Chauchat, C. Shewmake, D. Brooks, B. Kainz, C. Donnat, S. Holmes, and X. Pennec. Geomstats: A python package for riemannian geometry in machine learning. *Journal of Machine Learning Research*, 21(223):1–9, 2020.

[58] C. W. Misner, K. S. Thorne, and J. A. Wheeler. *Gravitation*. Princeton University Press, 1973.

[59] D. Mumford. Pattern theory: The mathematics of perception. In *Proceedings of ICM 2002, Beijing, Vol. I*, pages 401–422, Beijing, 2002. Higher Education Press.

[60] H. Nguyen. *Gpu gems 3*. Addison-Wesley Professional, first edition, 2007.

[61] O. Orasch, N. Weber, M. Müller, A. Amanzadi, C. Gasbarri, and C. Trummer. Protein-Protein Interaction Prediction for Targeted Protein Degradation. *International Journal of Molecular Sciences*, 23(13):7033, 2022.

[62] X. Pennec. Statistical computing on manifolds: from Riemannian geometry to computational anatomy. In Frank Nielsen, editor, *Emerging Trends in Visual Computing*, volume 5416 of *LNCS*, pages 347–386. Springer, 2008.

[63] X. Pennec. 3 - manifold-valued image processing with spd matrices. In X. Pennec, S. Sommer, and T. Fletcher, editors, *Riemannian Geometric Statistics in Medical Image Analysis*, pages 75–134. Academic Press, 2020.

[64] J. C. Peterson, R. M. Battleday, T. L. Griffiths, and O. Russakovsky. Human uncertainty makes classification more robust. In *ICCV*, pages 9617–9626, 2019.

[65] J.M. Peyrat, H. Delingette, M. Sermesant, X. Pennec, C. Xu, and N. Ayache. Registration of 4d time-series of cardiac images with multichannel diffeomorphic demons. *Med Image Comput Comput Assist Interv*, 2008.

[66] G. Peyré and M. Cuturi. *Computational Optimal Transport: With Applications to Data Science*. Foundations and trends in machine learning. Now Publishers, 2019.

[67] Quino. *Mafalda*, volume 5. Edit. Jorge Alvarez, 1 edition, 1969.

[68] M. E. Sander, P. Ablin, M. Blondel, and G. Peyré. Sinkformers: Transformers with doubly stochastic attention. In *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*, volume 151 of *Proceedings of Machine Learning Research*, pages 3515–3530. PMLR, 28–30 Mar 2022.

[69] J.-B. Schiratti, S. Allassonnière, O. Colliot, and S. Durrleman. Learning spatiotemporal trajectories from manifold-valued longitudinal data. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *NIPS 28*, pages 2404–2412. Curran Associates, Inc., 2015.

[70] Jean-Baptiste Schiratti, Stéphanie Allassonniere, Olivier Colliot, and Stanley Durrleman. A Bayesian mixed-effects model to learn trajectories of changes from repeated manifold-valued observations. *Journal of Machine Learning Research*, 18:1–33, December 2017.

[71] B. Schölkopf, K. Tsuda, and J.-P. Vert. *Kernel Methods in Computational Biology*. The MIT Press, 07 2004.

[72] N. Singh, J. Hinkle, S. Joshi, and P. T. Fletcher. Hierarchical geodesic models in diffeomorphisms. *International Journal of Computer Vision*, 117(1):70–92, 2016.

[73] B. K. Sriperumbudur, K. Fukumizu, and G. Lanckriet. On the relation between universality, characteristic kernels and RKHS embedding of measures. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS-10)*, volume 9, pages 773–780, 2010.

[74] K. M. Stouffer, A. Trouvé, L. Younes, M. Kunst, L Ng, H. Zeng, M. Anant, J. Fan, Y. Kim, X. Chen, M Rue, and M. I. Miller. Cross-modality mapping using image varifolds to align tissue-scale atlases to molecular-scale measures with application to 2d brain sections. *Nat Commun*, 15(3530), 2024.

[75] T. Tallinen, J. Y. Chung, J. S. Biggins, and L. Mahadevan. Gyrification from constrained cortical expansion. *Proceedings of the National Academy of Sciences*, 111(35):12667–12672, 2014.

[76] K. Ushey, JJ Allaire, and Y. Tang. *reticulate: Interface to 'Python'*, 2025. R package version 1.42.0, https://github.com/rstudio/reticulate.

[77] M. Vaillant and J. Glaunès. Surface matching via currents. In *Information Processing in Medical Imaging*, volume 3565 of *Lecture Notes in Computer Science*. Springer, 2005.

[78] K. Vogtmann, A. Weinstein, and V.I. Arnold. *Mathematical Methods of Classical Mechanics*. Graduate Texts in Mathematics. Springer New York, 1997.

[79] R. E. Wengert. A simple automatic derivative evaluation program. *Communications of the ACM*, 7(8):463–464, Aug 1964.

[80] G. Wu, Q. Wang, J. Lian, and D. Shen. Estimating the 4d respiratory lung motion by spatiotemporal registration and building super-resolution image. In *MICCAI*, pages 532–539, 2011.

[81] L. Younes. Jacobi fields in groups of diffeomorphisms and applications. *Quarterly of applied mathematics*, pages 113–134, 2007.

[82] L. Younes. *Shapes and Diffeomorphisms*, volume 171 of *Applied Mathematical Sciences*. Springer Berlin Heidelberg, 2010.