Manifolds.jl: An Extensible Julia Framework for Data Analysis on Manifolds

SETH D. AXEN, University of Tübingen, Germany

MATEUSZ BARAN, AGH University of Krakow, Poland and Cracow University of Technology, Poland

RONNY BERGMANN, Norwegian University of Science and Technology, Trondheim, Norway

KRZYSZTOF RZECKI, AGH University of Krakow, Poland and Cracow University of Technology, Poland

We present the Julia package `Manifolds.jl`, providing a fast and easy-to-use library of Riemannian manifolds and Lie groups. This package enables working with data defined on a Riemannian manifold, such as the circle, the sphere, symmetric positive definite matrices, or one of the models for hyperbolic spaces. We introduce a common interface, available in `ManifoldsBase.jl`, with which new manifolds, applications, and algorithms can be implemented. We demonstrate the utility of `Manifolds.jl` using Bézier splines, an optimization task on manifolds, and principal component analysis on nonlinear data. In a benchmark, `Manifolds.jl` outperforms all comparable packages for low-dimensional manifolds in speed; over Python and Matlab packages, the improvement is often several orders of magnitude, while over C/C++ packages, the improvement is twofold. For high-dimensional manifolds, it outperforms all packages except for Tensorflow-Riemopt, which is specifically tailored for high-dimensional manifolds.

CCS Concepts: • **Software and its engineering** → **Software libraries and repositories**; • **Mathematics of computing** → **Mathematical software performance**;

Additional Key Words and Phrases: Riemannian manifold, Lie group, nonlinear spaces, exponential map, logarithmic map, Julia, scientific computing, optimization on manifolds

ACM Reference format:
Seth D. Axen, Mateusz Baran, Ronny Bergmann, and Krzysztof Rzecki. 2023. Manifolds.jl: An Extensible Julia Framework for Data Analysis on Manifolds. *ACM Trans. Math. Softw.* 49, 4, Article 33 (December 2023), 23 pages.
https://doi.org/10.1145/3618296

1 INTRODUCTION

In many applications, measured data appears in non-linear spaces. In such spaces, common data analysis operations like the computation of a mean are not directly possible, since for example there is no addition operation on such data. However, many of these spaces have the advantage of being Riemannian manifolds, that is, they are smooth manifolds equipped with a Riemannian metric, i.e., with a locally varying inner product [2, 13, 18]. Such manifolds appear in many applications. For example, in Interferometric Synthetic Aperture Radar (InSAR) [15], the measured data are angles, and in diffusion tensor magnetic resonance imaging (DT-MRI) [19, 21, 22], diffusion at each voxel is described by a symmetric positive definite matrix. The special orthogonal group of rotations is used in analysis of data from orientation sensors [33]. The sphere is used in various applications of directional statistics such as hydrology [16]. The discipline of computational anatomy [17, 24] extensively uses geometric methods, including Large Deformation Diffeomorphic Metric Mapping [35] with applications to medical data analysis [37, 42]. In image analysis, Grassmann and Stiefel manifolds are used for pose estimation by Turaga et al. [43]. Various manifolds with Fisher-Rao-like metrics are also used for shape analysis by, for example, Srivastava et al. [40], Baran [3], and Rzecki and Baran [38].

Other commonly encountered spaces include spheres or the Stiefel manifold of matrices with orthonormal columns.

Using tools of differential geometry, many operations can be generalized to manifolds by using only generic functions like the inner product, its induced norm, geodesics, the exponential, or the logarithmic map. Closed-form expressions for the generic functions do not always exist, but efficient implementations to approximate them are still often possible. Recent advances continue to resolve numerical challenges with efficiently implementing these operations. Our library offers a wide range of such implementations, which makes performant data analysis for manifold-valued data more accessible.

One example of an operation that can be realized with these generic functions is the mean, which can be rephrased as the point in the space that (globally) has the minimum sum of squared distances from the given data points. This definition can even be generalized to metric spaces where this minimizer is called the Fréchet mean. Local minimizers may also exist on a Riemannian manifold; these are the Riemannian centers of mass [25, 28], sometimes also called Karcher means.

A main necessity for a wider adoption of geometric methods is a common, extensible, and efficient collection of operations on a large variety of manifolds. One of the first toolboxes for optimization on manifolds was the Matlab package Manopt [14]. The manifolds within this package are objects, where a fixed metric is assumed. Implementing a second metric for the same manifold requires a new Matlab object, or replacing appropriate functions in the object. A manifold can have an exponential map. Most have at least one retraction implemented. Setting a retraction for the solver requires then to set that retraction as the `retr` function in the object. In that sense, this function can be considered the default retraction that is used in solvers. Several libraries followed, such as the C++ library ROPTLIB [27], which does provide a short user manual with introductory examples and an interface overview but lacks comprehensive documentation. The seven major libraries for differential geometry in Python are Pymanopt [41], Geomstats [36], Geoopt [29], TheanoGeometry [30], Jax Geometry [30], Tensorflow RiemOpt [39], and McTorch [34], each with their own implementations of a sometimes limited set of Riemannian manifolds. Geomstats implements a metric interface, such that different metrics on one manifold can be easily distinguished. Some libraries are more focused on specific tasks. For example, MVIRT [6] provides a variational model formulation for manifold-valued image processing. This toolbox introduces implementations of a cyclic proximal point algorithm [4] applied to first- and second-order total variation [5], as well as

total generalized variation [8]. MVIRT also provides the parallel Douglas–Rachford algorithm [10] for large-scale, nonsmooth optimization.

The above packages all follow the same basic approach: they generically implement their statistical or optimization algorithms for an arbitrary Riemannian manifold and provide a certain set of Riemannian manifolds that implement the necessary properties for these generic implementations to work. The speed of an algorithm then depends on efficiency of both the abstract implementation of the algorithm as well as of employed operations on the specific manifold considered in an application.

Here we present the Julia package `Manifolds.jl`, together with its accompanying interface package `ManifoldsBase.jl`. They provide an easily extensible interface to implement methods on a generic Riemannian manifold. Moreover, they implement a comprehensive, well-documented library of efficient operations on common manifolds.

The remainder of this article is organized as follows: Section 2 provides basic definitions of concepts from differential geometry and illustrates them using the unit sphere. Next, Section 3 describes the general design of `Manifolds.jl`. Section 4 compares features of this library to what other libraries offer. A few examples of usage of `Manifolds.jl` are given in Section 5. Section 6 presents performance benchmarks of `Manifolds.jl` and other libraries on a set of manifolds for selected operations. Finally, conclusions are contained in Section 7.

2 MANIFOLDS

In this section, we introduce the necessary terms to work on Riemannian manifolds. We refer the reader to the textbooks [2, 13, 18, 31]. After introducing manifolds and related terms and definitions, the concepts are illustrated for the unit sphere in $\mathbb{R}^3$ and the manifold of symmetric positive definite matrices.

2.1 General Definitions

We denote by $\mathcal{M}$ a d-dimensional manifold. A manifold $\mathcal{M}$ is defined using an atlas $\mathcal{A}$ consisting of charts $\varphi \colon U \to \mathbb{R}^d$ that introduce bijections from subsets $U \subset \mathcal{M}$ of $\mathcal{M}$ to open subsets of $\mathbb{R}^d$ [2]. One way to work on manifolds is to work in charts. Often, however, it is more computationally convenient or efficient to represent points on a manifold using coordinates of an embedding, instead of using a chart and working with *intrinsic* methods. Intrinsic here refers to the idea that the methods and tools used are "well defined regardless of the embedding space" [13, p. 3].

At every point $p \in \mathcal{M}$ we denote the tangent space by $T_p\mathcal{M}$ and the tangent vectors by $X_p, Y_p \in T_p\mathcal{M}$, where we leave out the index when the point p is clear from context. We denote the Riemannian metric by $\langle \cdot, \cdot \rangle_p \colon T_p\mathcal{M} \times T_p\mathcal{M} \to \mathbb{R}$. The metric also introduces a norm for tangent vectors $X \in T_p\mathcal{M}$, which we denote by $\|X\|_p$. Note that a smooth manifold can be equipped with different metrics.

For $p, q \in \mathcal{M}$ and $X \in T_p\mathcal{M}$ we denote by $\gamma_{p,X} \colon I \to \mathcal{M}$ the geodesic starting in $\gamma_{p,X}(0) = p$ with $\dot{\gamma}_{p,X}(0) = X$, where I is the maximal interval over which the geodesic can be defined [32, Cor. 4.28] and $O_p = \{X \in T_p\mathcal{M} : \gamma_{p,X} \text{ is defined on an interval containing } [0,1]\}$. A shortest geodesic denoted by $\gamma_{p;q}(t)$ starts at p at time $t = 0$, reaches $\gamma_{p;q}(1) = q$ at $t = 1$, and is length-minimizing. Note that the shortest geodesic might not be unique.

We further denote by $\exp_p \colon T_p\mathcal{M} \to \mathcal{M}$ the exponential map. We define the injectivity radius $\mathrm{inj}(p)$ as the supremum over all radii $r > 0$ such that $\exp_p$ is a diffeomorphism on the open ball $B(p,r) = \{X \in T_p\mathcal{M} : \|X\|_p < r\}$. We denote by $U_p = B(p, \mathrm{inj}(p))$ and introduce $\mathcal{U}_p = \exp_p(U) \subset \mathcal{M}$. By definition $\exp_p |_{U_p} \colon U_p \to \mathcal{U}_p$ is a diffeomorphism and hence the inverse of $\exp_p |_{U_p}$ exists,

called the logarithmic map. Formally the logarithmic map is defined as [13, Def. 10.20]

$$\log_p q = \underset{X \in O_p}{\arg\min} \|X\| \quad \text{subject to } \exp_p X = q.$$

In some cases, the exponential and logarithmic maps can be derived in closed form, but otherwise they are expensive to evaluate. Therefore, one often uses a retraction $\mathrm{retr}_p \colon T_p \mathcal{M} \to \mathcal{M}$ that approximates the exponential map to (at least) first order [13, Def. 3.47]. When the retraction can be inverted, i.e., from p and q we obtain X, this inverse retraction $\mathrm{retr}_p^{-1} \colon q = \mathrm{retr}_p X \mapsto X$ can be used to approximate the logarithmic map. While the exponential map, and hence also the logarithmic map, depend on the metric, a retraction is defined by properties of the retraction along a curve $c(t) = \mathrm{retr}_p(tX)$ for $X \in T_p \mathcal{M}$ and its derivative $c'(t)$ and hence is independent of the metric.

A tangent space can be seen as "the directions to walk in" from a point p. Let q be a second point on $\mathcal{M}$ such that the shortest geodesic is unique. For a tangent vector $X \in T_p \mathcal{M}$ at p we denote the parallel transport to q along the shortest geodesic (assuming it is unique) by $\mathrm{PT}_{q \leftarrow p} X$. Again, since this map might not be given in closed form or might be expensive to compute, it is often approximated with methods called vector transports.

Computationally, points $p \in \mathcal{M}$ and tangent vectors $X \in T_p \mathcal{M}$ are usually represented by arrays, i.e., vectors or matrices. This is realized usually by embedding a manifold into some larger space. A smooth function $i \colon \mathcal{M} \to \mathbb{R}^m$ whose differential $Di(p)$ has rank d for every $p \in \mathcal{M}$ is called an immersion. The space $\mathbb{R}^m$ is then called the embedding space of $\mathcal{M}$, and $\mathcal{M}$ is an embedded manifold.

A manifold $\mathcal{M}$ is called a Lie group when we have a group operation $\circ \colon \mathcal{M} \times \mathcal{M} \to \mathcal{M}$ that is smooth and whose inverse map $p \mapsto p^{-1}$ is also smooth.

2.2 Example: The Two-dimensional Unit Sphere

The unit sphere in the three-dimensional Euclidean space, denoted $\mathbb{S}^2$, is an intuitive example of a Riemannian manifold. Points $p \in \mathbb{S}^2$ on the unit sphere have unit norm $\|p\|_2 = 1$. Adding (or subtracting) two of these points yields a point with a non-unit norm, which therefore does not belong to $\mathbb{S}^2$. However, there is an intuitive way of reinterpreting addition and subtraction that makes an analogous operation on the sphere possible.

We can interpret the difference $y - x$ between two points $x, y \in \mathbb{R}^3$ as a vector starting from x and pointing to y. Multiplying such a vector by a scalar and adding it to x defines a set of points on a straight line passing through x and y, i.e., $t \mapsto x + t(y - x)$. This line segment joining x and y is the shortest connecting curve on $\mathbb{R}^3$ called a shortest geodesic $\gamma_{x;y}(t)$ on $\mathbb{R}^3$. On $\mathbb{S}^2$, the shortest geodesic is no longer a line segment but is instead a shortest great arc passing through points $p, q \in \mathbb{S}^2$. For any two non-opposing points, the intersection of their common plane through the origin with the sphere yields two geodesics, one shorter than the other. We call this shorter great arc the shortest geodesic. For two opposing points $p, -p \in \mathbb{S}^2$, all great arcs that join the two points are half-circles; they are therefore joined by infinitely many shortest geodesics $\gamma_{p;-p}$. We can measure the distance $d_{\mathbb{S}^2}(p, q)$ between the points p and q as the length of a shortest geodesic, depicted as the blue curve in Figure 1.

To conceptualize tangent vectors, consider the set of vectors orthogonal to a point $p \in \mathbb{S}^2$, that is

$$T_p \mathbb{S}^2 := \{X \in \mathbb{R}^3 | \langle X, p \rangle = 0\}.$$

We obtain an actual plane tangent to the sphere if we consider all points $p + X$, $X \in T_p \mathbb{S}^2$, which is tangent to the sphere and "touches" the sphere at p (at $X = 0$). For every $X \in T_p \mathbb{S}^2$ there exists a great arc $c(t)$, which is a geodesic (acceleration free curve when "measuring" in the tangent spaces it passes through) that runs through $c(0) = p$ with derivative $\dot{c}(0) = X$. When parametrizing the great arc with constant speed $\|\dot{c}(t)\|_2 = \|X\|_2$ we reach some point $c(1) = q \in \mathbb{S}^2$ after time $t = 1$.
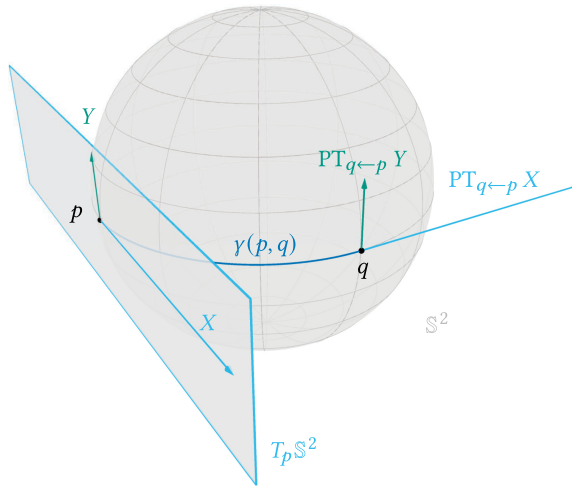
Fig. 1. Unit sphere in $\mathbb{R}^3$ with two points p, q. Tangent space at p is labeled $T_p\mathbb{S}^2$, geodesic between p and q is the blue arc $\gamma(p,q)$, and the vector X is the tangent vector from $T_p\mathbb{S}^2$ that points towards q. Y is another vector from $T_p\mathbb{S}^2$ orthogonal to X. All tangent vectors are blue. Parallel transports of X and Y from p to q are shown as vector $\mathrm{PT}_{q\leftarrow p} X$ and $\mathrm{PT}_{q\leftarrow p} Y$, respectively.

This point is $q = \exp_p X$. As long as the norm of X is less than π, the inverse function mapping q to $X = \log_p q$ is unique as well. With a start point p and an initial velocity X, the great arc $c(t)$ can also be determined if both end points $c(0) = p$ and $c(1) = q$, $p \neq -q$ are given. The plane containing the origin, p, and q is uniquely determined, and its intersection with the sphere $\mathbb{S}^2$ forms a circle. The shorter arc segment connecting p and q on this circle is the image of the curve $c(t)$. Finally, there exists a unique constant speed parametrization $\|\dot{c}(t)\|_2 = \mathrm{const}$ that reaches $q = c(1)$ after time 1. Then $X = \dot{c}(0) = \log_p q$. If $q = -p$, the intersection plane is not unique, and any tangent vector $X \in T_p\mathbb{S}^2$ with $\|X\|_2 = \pi$ is a solution of the above construction.

Vectors tangent to the sphere at different points cannot be added or subtracted. First, they need to be transported to the same point. This is performed using parallel transport. Transport of a vector X tangent at p to a point q is denoted $\mathrm{PT}_{q\leftarrow p} X$. This operation results in a new vector tangent to the sphere at q. Note that transporting a vector X tangent at p between a series of points – for example from p to q, from q to r, and from r back to p – may not result in the same vector X.

These operations can be extended to all manifolds as described in Section 2.1.

2.3 Example: Two Metrics on the Symmetric Positive-definite Matrices

A second, more elaborate example is the set of $n \times n$ symmetric positive definite (SPD) matrices $\mathcal{P}(n)$. The set consists of all matrices $p \in \mathbb{R}^{n \times n}$ that are symmetric ($p = p^{\mathrm{T}}$) and positive definite, that is, for any $0 \neq x \in \mathbb{R}^n$, it holds that $x^{\mathrm{T}} p x > 0$, which equivalently means that all eigenvalues of p are strictly positive.

The "directions to walk in" , or to be precise, the tangent space $T_p\mathcal{P}(n)$ is the set of symmetric matrices $X = X^{\mathrm{T}}$. Note that for an arbitrary symmetric matrix $q = p + tX$ where $t > 0$, q might only be SPD for very small values of t. Similar to the sphere, addition might mean leaving the manifold.

There are several metrics we can introduce on the tangent spaces $T_p\mathcal{P}(n)$. In the following, we compare two and their effect on the exponential map, or informally said "how to move into a

direction X". One possible metric is the affine invariant metric, given by [37]

$$\langle X, Y \rangle_{p,\text{AI}} = \text{tr}(p^{-1}Xp^{-1}Y).$$

With this metric, the exponential map is

$$\exp_p X = p^{\frac{1}{2}} \text{Exp}(p^{-\frac{1}{2}}Xp^{-\frac{1}{2}})p^{\frac{1}{2}},$$

where Exp is the matrix exponential. The simplest case of $n = 1$ yields the manifold of positive numbers, whose inner product simplifies to $\langle X, Y \rangle_{p,\text{AI}} = \frac{XY}{p^2}$ and whose exponential is $\exp_p X = pe^{X/p}$. Hence, non-positive numbers can never be reached. The main computational effort for this metric is the computation of square roots and exponentials.

Another possibility is to use the Bures–Wasserstein metric, which is given by [12]

$$\langle X, Y \rangle_{p,\text{BW}} = \frac{1}{2}\text{tr}(L_p(X)Y),$$

where $q = L_p(X)$ denotes the Lyapunov operator, which solves $pq + qp = X$. The exponential map then reads

$$\exp_p X = p + X + L_p(X)pL_p(X),$$

where for both operations the computational effort is mainly to compute the Lyapunov operator.

3 STRUCTURE OF THE LIBRARY

`Manifolds.jl`[1] is a registered Julia package. Its most recent version, currently 0.8.61, can be easily installed typing **using** `Pkg; Pkg.add("Manifolds")` in the Julia REPL and activated by typing **using** `Manifolds` afterwards.

The library of manifolds `Manifolds.jl` is based on the interface `ManifoldsBase.jl`,[2] whose current version is 0.14.5. This interface defines a common base for Riemannian manifolds in Julia. It hence serves two purposes. On one hand, it can be used to implement additional manifolds like the ones in `Manifolds.jl`. On the other hand, it can be used to define arbitrary functionality on arbitrary manifolds without depending on or loading the whole library of manifolds.

Both the library and the interface use GitHub Actions to automatically generate their documentation as well as run tests, following the philosophy of continuous integration. These tests are run on Linux and Mac OS with the latest (Julia 1.9), long-term-support (Julia 1.6), and nightly versions of Julia. For pull requests (PRs) and for commits on the master branch a format test is run to ensure that all source code follows the formatting rules using `JuliaFormatter.jl`[3]

Using the `JuliaRegistrator` GitHub application and `TagBot` GitHub action, new releases are automatically registered to the registry and new GitHub release entries as well as documentation page generation is triggered.

Currently, the code coverage, i.e., percentage of code lines executed during testing, is 99.86 % for `ManifoldsBase.jl` and 98.93 % for `Manifolds.jl`, respectively.

3.1 Manifolds and Functions Thereon

A Riemannian manifold in `Manifolds.jl` is defined by introducing a subtype of `AbstractManifold{F}`, where $\mathbb{F}$ is the field the manifold is based on: the real numbers $\mathbb{R}$, complex numbers $\mathbb{C}$, or quaternions $\mathbb{H}$. For example,

---

[1]The documentation for `Manifolds.jl` is available at https://juliamanifolds.github.io/Manifolds.jl/stable/

[2]The documentation for `ManifoldsBase.jl` is available at https://juliamanifolds.github.io/ManifoldsBase.jl/stable/

[3]See https://juliamanifolds.github.io/Manifolds.jl/stable/misc/contributing.html for details.

```
M1 = Sphere(2)
M2 = Euclidean(2, 3, ℂ)
M3 = SymmetricPositiveDefinite(3)
```

creates the sphere $\mathbb{S}^2 \subset \mathbb{R}^3$ from 2.2, the space of complex-valued 2×3 matrices $\mathbb{C}^{2 \times 3}$, and the set of 3×3 symmetric positive definite matrices $\mathcal{P}(3)$ from 2.3, respectively.

The interface provides a unified way to work with data defined on arbitrary manifolds, including those listed above. As a first example, `manifold_dimension(M)` returns the dimension of the manifold. The two functions `is_point(M, p)` and `is_vector(M, p, X)` can be used to check whether points or tangent vectors are valid. The last two have an optional positional parameter to activate throwing an error for invalid points or tangent vectors, which is `false` by default. Hence calling `is_point(M, p, true)` returns `true` if `p` is a point on `M` and otherwise throws an error describing why `p` is not a point on `M`.

For functions that return points or tangent vectors, interface functions have both in-place and allocating versions. By default, the allocating version allocates an output and then calls the in-place version. For example, for the exponential map, the function `exp!(M, q, p, X)` stores $\exp_p X$ in the variable `q`. When the in-place variant is implemented for a new manifold, the allocating version `exp(M, p, X)` is automatically available. Calling `exp(M, p, X)` first allocates memory and then calls the in-place function mentioned above. The same holds for `geodesic(M, p, X)`, which returns the function $\gamma_{p,X}$. To evaluate the geodesic at a certain t one can call `geodesic(M, p, X, t)`. In both cases, in-place variants are available as well. The same principle of in-place and allocating variants is also realized for `log(M, p, q)` and `log!(M, X, p, q)`, which should return a deterministic result if the logarithmic map is not unique as for opposite points on the sphere. With the latter, the former is available with a default implementation as well as `shortest_geodesic(M, p, q)` (and its in-place variant) again with `t` as an optional further parameter, similar to `geodesic`. While all these default implementations are available, they can be overwritten if for example the shortest geodesic $\gamma_{p;q}$ can be computed more efficiently than by calling $\exp_p(t \log_p q)$. Finally, the parallel transport $\mathrm{PT}_{q \leftarrow p} X$ is implemented as `parallel_transport_to(M, p, X, q)` and `parallel_transport_to!(M, Y, p, X, q)` in place of `Y`, respectively.

When using `SymmetricPositiveDefinite(n)`, the affine-invariant metric is used by default.

3.2 Representation of Points and Vectors

Usually points $p \in \mathcal{M}$ and tangent vectors $X \in T_p\mathcal{M}$ are not explicitly typed, and only the manifolds type is specified for dispatch. Most manifolds implicitly assume that both points and tangent vectors are instances of `AbstractArray` or at least have several functions like addition defined. This enables a great deal of flexibility in types to use for these parameters in a function, such as the statically sized arrays from `StaticArrays.jl`, which for small dimensions are stack-allocated. In two situations the points and tangent vectors are typed. If the representation requires more than one array, as is the case for the fixed-rank matrices, then a type is used to enforce this. Since the interface functions are not typed, features like the allocating variant of a function being available as soon as the in-place one is implemented still work. Another reason for using types is to distinguish different representations of points and tangent vectors on one manifold. For example, for the hyperbolic space $\mathcal{H}^n$, three representations are implemented: the hyperboloid model, the Poincaré ball model, and the Poincaré half-space model. These are distinguished by using different types for points and tangent vectors, such as `PoincareBallPoint` and `PoincareBallTVector` for points and tangent vectors in the Poincaré ball model. Using (plain) arrays on $\mathcal{H}^n$ is equivalent to using `HyperboloidPoint` and `HyperboloidTVector`.

3.3 Retractions and Vector Transports

The exponential map $\exp_p$, the logarithmic map $\log_p$, and parallel transport $\text{PT}_{q\leftarrow p}$ are computationally expensive or not given in closed form for some manifolds. For these cases one can specify subtypes of `AbstractRetractionMethod`, `AbstractInverseRetractionMethod`, or `AbstractVectorTransportMethod` and create objects of these subtypes, respectively denoted `r`, `s`, and `v` in the following example. The corresponding functions are called by `retract(M, p, X, r)`, `inverse_retract(M, p, q, s)`, and `vector_transport_to(M, p, X, q, v)`. These methods also have in-place variants. On the sphere $\mathbb{S}^2$, for example, `r = ProjectionRetraction()` computes the retraction given by projecting $p + X$ onto the sphere.

These three methods can be used to implement generic algorithms on manifolds. Since the exponential map might not be given in closed form, the interface provides the function `default_retraction_method(M)` to obtain the retraction that is most often used on a certain manifold. If the exponential map is available (and can be computed efficiently), this method returns `ExponentialRetraction()`, which lets `retract` fall back to using `exp`. This way algorithms can easily be modified to use different available retractions. The same scheme applies for inverse retractions and vector transports.

3.4 Features on Manifolds using a Decorator Pattern

As mentioned above, the manifold `M3 = SymmetricPositiveDefinite(3)` implicitly uses the affine invariant metric. To change that, the interface `ManifoldsBase.jl` provides a trait-based decorator pattern. To decorate the manifold with the Bures-Wasserstein metric, we can declare

```
M3b = MetricManifold(M3, BuresWassersteinMetric())
```

This acts like a decorator for the manifold in the following sense: all methods that depend on the metric, like `exp` and `log` mentioned above or `inner(M3b, p, X, Y)` and `norm(M3b, p, X)`, are implemented differently for this manifold. Other methods like `manifold_dimension` or `is_point` that do not depend on the metric are "passed down" to `M3` automatically.

Traits can be seen as a list of properties "attached" to a manifold. For example, `M2 = Euclidean(2, 3, ℂ)` has the property `IsDefaultMetric(EuclideanMetric())`, that is, using the manifold itself is completely equivalent to explicitly equipping the manifold with the same metric `M2b = MetricManifold(M2, EuclideanMetric())`. This formulation allows a manifold to be implemented with an implicit default metric without needing to directly implement a metric type or use `MetricManifold`. But one can still implement a second metric and decorate the manifold with it.

Similarly, a default embedding can be implicitly assumed, for example by implementing `get_embedding(M)`, `embed(M, p)` and `project(M, p)`. An embedding of `M` into a different manifold `N` can be represented by `EmbeddedManifold(M, N)` similarly to the metric above. For example, for the sphere `M1 = Sphere(2)` we have that `get_embedding(M1)` returns `Euclidean(3)`, and `embed(M1, p)` is the identity function since points are represented by unit vectors. The projection `project(M1, p)` normalizes `p` to norm 1. Also, for a Lie group, this implicit approach can be realized by implementing `compose(M, p, q)`. The explicit decorator is the `GroupManifold(M, op)`, where `op` is the Lie group operation.

As a final example of a decorator, we mention the `ValidationManifold`. For all default functions mentioned above, this manifold adds checks (`is_point` and `is_vector`) to their input and output values. While this somewhat reduces performance, using `M1b = ValidationManifold(M1)` provides an easy way to verify that none of the operations receive invalid inputs or produce invalid outputs. Here the `IsExplicitDecorator` property is set; that is, if no new method is implemented for the validation manifold `M1b`, the method for `M1` is automatically called.

When implementing a new manifold, the properties or traits are set using the function `active_traits`. For example, for the sphere this is defined as

```
function active_traits(f, ::AbstractSphere, args...)
    return merge_traits(IsIsometricEmbeddedManifold(), IsDefaultMetric(
    EuclideanMetric()))
end
```

which finally illustrates that setting `IsIsometricEmbeddedManifold()` specifies that the metric from the embedding is used and does not need to be implemented. Here the metric $\langle X, Y \rangle_p$ is computed when calling `inner(M1, p, X, Y)` as `inner(get_embedding(M1), embed(M1, p), embed(M1, p, X), embed(M1, p, Y))`. It does not need its own implementation. Since both embedding points and embedding tangent vectors are just the identity function for the sphere, this does not even introduce any overhead. Further, since all the information required for these dispatches is given by the type of the manifold, this dispatch-logic can be determined at compile time. It therefore does not yield any runtime overhead.

4   COMPARISON TO OTHER LIBRARIES

Table 1 compares availability of manifolds in `Manifolds.jl` 0.8.61 to other existing libraries: Geomstats 2.5.0 [36], Geoopt 0.5.0 [29], Manopt 7.1 [14], McTorch 1.1 [34], Pymanopt 2.1.1 [41], ROPTLIB 0.8 [27], TheanoGeometry [30], Jax Geometry [30], and TensorFlow RiemOpt 0.1.2 [39]. Availability of different operations is compared in Table 2.

Note that the extent of support varies significantly across libraries. For example `Manifolds.jl` currently implements five different metrics (affine-invariant, Bures-Wasserstein, Log-Euclidean, Log-Cholesky, and generalized Bures-Wasserstein) on the manifold of symmetric positive definite matrices, Geomstats only the first three ones and a Power-Euclidean one, Manopt only the first two, and Pymanopt only the first.

On the other hand, `Manifolds.jl` focuses solely on implementing Riemannian manifolds, metrics, Lie groups, and functions thereon. Geomstats further provides several statistical tools, and Manopt and Pymanopt further focus on optimization algorithms, such as a gradient descent scheme. For optimization on manifolds in Julia, we refer to `Manopt.jl` [7], which, like Manopt and Pymanopt, focuses also on nonsmooth optimization algorithms on manifolds. For that reason, Table 1 compares features provided by the libraries that focus on implementations of manifolds but not statistics and/or optimization.

All packages but ROPTLIB provide support for several Automatic Differentiation (AD) backends to compute Euclidean gradients and Hessians of functions defined in the embedding of a manifold and to convert them to Riemannian variants afterward.

5   EXAMPLES

This section demonstrates a few examples of composing generic operations from `Manifolds.jl` with the Julia ecosystem [11]. The first example defines a Bézier spline on a generic manifold to demonstrate how to easily define new functions on manifolds using `ManifoldsBase.jl`. Then, `Manopt.jl`[4] [7] is used as an example of a complex optimization library built on top of `Manifolds.jl`. Further, it is demonstrated how `Manifolds.jl` can be easily combined with independently developed libraries to implement fast, geometry-aware algorithms.

Note that all examples can be split into two components. First, a manifold and data on the manifold are constructed. Second, the algorithm is executed on this manifold. Since all methods are implemented in a generic way, switching to data from another manifold just requires to change the first part of introducing the manifold and loading/generating corresponding data.

---

[4]Current version 0.3.39, available at https://manoptjl.org

Table 1. Comparison of Manifolds Available in Related Libraries

Manifold	Geomstats	Geoopt	Manifolds.jl	Manopt	McTorch	Pymanopt	ROPTLIB	TheanoGeometry	Jax Geometry	TF RiemOpt
Centered matrices	X	X	C	R	X	X	X	R	X	X
Discretized curves	R	X	X	X	X	X	X	X	X	X
Ellipsoid	X	X	X	X	X	X	X	R	R	X
Elliptope	X	X	R	R	X	X	X	X	X	X
Embedded cylinder	X	X	X	X	X	X	X	X	R	X
Essential manifold	X	X	R	R	X	X	X	X	X	X
Euclidean	R	R	H[a]	C	R	R	R	R	X	R
Fixed rank matrices	X	X	C	R	X	X	C	X	X	X
Flag	X	X	R	X	X	X	X	X	X	X
General linear group	R	X	C	X	X	X	X	R	X	X
General multin. doubly stoc. matrices	X	X	X	R	X	X	X	X	X	X
Generalized Grassmann	X	X	C	R	X	X	X	X	X	X
Generalized Stiefel	X	X	C	R	X	X	X	X	X	X
Grassmann	R	X	C	C	X	R	R	X	X	R
Heisenberg group	R	X	R	X	X	X	X	X	R	X
Hilbert sphere	X	X	X	X	X	X	R	X	X	X
Hyperbolic space	R	R	R	R	X	X	X	X	R[b]	R
Kendall's preshape space	R	R	R	X	X	X	X	R	R	X
Kendall's shape space	R	R	R	X	X	X	X	R	R	X
Lorentzian manifold	R	R	R	X	X	X	X	X	X	X
Multin. doubly stoc. matrices	X	R	R	R	X	X	X	X	X	X
Multinomial matrices	X	X	R	R	X	X	X	X	X	X
Multinomial symmetric matrices	X	X	R	R	X	X	X	X	X	X
Oblique manifold	X	X	R	R	X	R	R	X	X	X
Phases of real DFT	X	X	X	C	X	X	X	X	X	X
Poincaré polydisk	R	X	X	X	X	X	X	X	X	X
Positive numbers	X	X	R	R	X	R	R	X	X	X
Probability simplex	R	X	R	R	X	X	X	X	X	X
Product Space	R	R	H	C	X	C	C	X	X	R
Projective Space	R	X	H[a]	C	X	X	R	X	X	R
Rotations	R	X	R	R	X	R	R	R	X	R
Skew-symmetric matrices	R	X	C	R	X	X	X	X	X	X
Special Euclidean group	R	X	R	R	X	X	X	X	X	X
Spectrahedron	X	X	R	R	X	X	X	X	X	X
Sphere / Circle	R	R[a]	C[a]	C[a]	R	C[a]	C	R	R[b]	R
Stiefel	R	R	C	C	R	R	C	X	X	R
Sym. matrices	R	X	C	R	X	X	X	X	X	X
Sym. pos. def. matrices	R	X	R	R	R	R	R	R	X	R
Sym. pos. def. simplex	X	X	X	C	X	X	X	X	X	X
Sym. pos. semidef. fixed rank matrices	R	X	C	C	X	C	C	X	X	X
Symplectic	X	X	R	X	X	X	X	X	X	X
Symplectic Stiefel	X	X	R	X	X	X	X	X	X	X
Tangent bundle with Sasaki metric	R	X	R	X	X	X	X	X	X	X
Tensors of fixed tensor train rank	X	X	X	R	X	X	X	X	X	X
Tucker manifold	X	X	C	R	X	X	X	X	X	X
Torus	R	R	R	R	X	X	X	X	R	R
Unit-norm sym. matrices	X	X	C	R	X	X	X	X	X	X

A manifold is marked as available if at least a retraction is implemented using a generic interface. The fields R, C, H are meant in increasing order, X indicating that a manifold is not available.
[a] this manifold is available for arrays with any number of indices.   [b] only the two-dimensional manifold is available.

Each of the examples illustrates one feature original to Manifolds.jl. While all of them are implemented agnostic of a specific manifold at hand, using a recursion by dispatch is unique to Julia. An optimization problem can be set up in three lines, defining the cost and the gradient as inline functions. The last example illustrates the generic implementation of bases of tangent spaces. These

Table 2. Comparison of Features Available in Related Libraries

Feature	Geomstats	Geoopt	Manifolds.jl	Manopt	McTorch	Pymanopt	ROPTLIB	TheanoGeometry	Jax Geometry	TF RiemOpt
Atlases and charts	✗	✗	✓	✗	✗	✗	✗	✗	✓	✗
Automatic differentiation	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓
Compatibility with extended precision floating point types	✗	✗	✓	✓	✗	✗	✗	✗	✗	✗
Bases of tangent spaces	✗	✗	✓	✓	✗	✗	✗	✗	✗	✗
Connections	✓	✗	✓	✗	✗	✗	✗	✗	✗	✗
Exponential maps (retractions)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Inner product	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Lie groups and actions	✓	✗	✓	✗	✗	✗	✗	✓	✗	✗
Logarithmic maps (inverse retractions)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Riemannian gradients	✗	✓	✓	✓	✓	✓	✓	✓	✗	✓
Riemannian Hessians	✗	✓	✓	✓	✓	✓	✓	✗	✗	✗
Vector transport	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

can even be abstract in the sense that no tangent space basis vectors are stored and they are only allocated if needed.

5.1 Implementing Functions on Manifolds

Functions on manifolds dispatch on the manifold itself, which is included as the first argument. As an example, we implement a Bézier curve on an arbitrary manifold. Bézier curves are used to model curves and surfaces in computer graphics. Their shape is determined by so called control points. The curve starts in the first control point and ends in the last. For more details on their use on manifolds, see for example [1, 9, 23].

A Bézier curve can be defined using the De Casteljau algorithm. Let us first consider the classical Euclidean case: given a degree n and $n + 1$ (control) points $x_0, \ldots, x_n \in \mathbb{R}^n$, the Bézier curve $b_n(t; x_0, \ldots, x_n)$ is defined recursively as

$$b_n(t; x_0, \ldots, x_n) = b_1(t; b_{n-1}(t; x_0, \ldots, x_{n-1}), b_{n-1}(t; x_1, \ldots, x_n))$$
$$b_1(t; x_0, x_1) = x_0 + t(x_1 - x_0)$$

A Bézier curve $b_1(t; x_0, x_1)$ of order 1 is just a line. To evaluate a Bézier curve $b_2(t; x_0, x_1, x_2)$ at a point t, we would need three line evaluations: $y_0 = b_1(t; x_0, x_1)$, $y_1 = b_1(t; x_1, x_2)$ and their connecting line $b_2(t; x_0, x_1, x_2) = b_1(t; y_0, y_1)$. The result is a quadratic curve in $\mathbb{R}^n$.

On a Riemannian manifold $\mathcal{M}$ we obtain a Bézier curve by evaluating shortest geodesics $\gamma(t; p, q)$ connecting $p, q \in \mathcal{M}$ instead of line segments.

The implementation for an arbitrary Riemannian manifold is given by the following code and returns the Bézier curve evaluated at t.

```julia
function bezier(M::AbstractManifold, t, pts::NTuple)
    p = bezier(M, t, pts[1:(end - 1)])
    q = bezier(M, t, pts[2:end])
    return shortest_geodesic(M, p, q, t)
end
function bezier(M::AbstractManifold, t, pts::NTuple{2})
    return shortest_geodesic(M, pts[1], pts[2], t)
end
```

For four points on the sphere given by

$$p_0 = \begin{pmatrix} 0 & -1 & 0 \end{pmatrix}^{\mathrm{T}}, \quad p_1 = \begin{pmatrix} -\frac{1}{2} & -\frac{1}{\sqrt{2}} & -\frac{1}{2} \end{pmatrix}^{\mathrm{T}}, \quad p_2 = \begin{pmatrix} -\frac{1}{\sqrt{2}} & -\frac{1}{2} & \frac{1}{2} \end{pmatrix}^{\mathrm{T}}, \quad p_3 = \begin{pmatrix} -1 & 0 & 0 \end{pmatrix}^{\mathrm{T}},$$
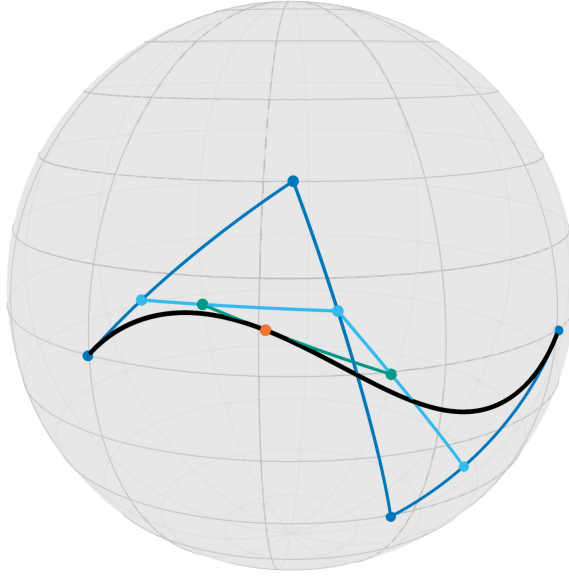
Fig. 2. Illustration of the De Casteljau algorithm to evaluate a Bézier curve on the Sphere $\mathbb{S}^2$. The initial points and their shortest geodesics (blue) yield three points (cyan) and two points (green) during the recursion, and their connecting geodesic the point on the Bézier curve (orange).

and an evaluation of b_3 at $t = \frac{2}{3}$ all shortest geodesics involved in the evaluation are shown in Figure 2. This figure has been adapted from [9, Figure 9].

5.2 Optimization using Manopt.jl

Now we consider the generalization of the mean to Riemannian manifolds. The mean of some (Euclidean) data $x_1, \ldots, x_N \in \mathbb{R}^n$ can be computed as

$$\hat{x} = \frac{1}{N} \sum_{k=1}^{N} x_k.$$

One can interpret this formula as the solution of the first-order optimality condition of the optimization problem

$$\hat{x} = \operatorname*{arg\,min}_{y \in \mathbb{R}^n} \sum_{k=1}^{N} \|x_k - y\|^2.$$

That is, $\hat{x}$ is the point y that minimizes the sum of squared distances, or equivalently, the variance. This interpretation can be generalized to manifolds, but usually no closed form for the minimizer $\hat{x}$ exists. Still, replacing the squared distance with the squared Riemannian distance, the gradient can easily be computed to perform a gradient descent algorithm with a suitable line search strategy, cf. for example [2, Chapter 4]. For given points $p_1, \ldots, p_N \in \mathcal{M}$ the cost function and the gradient read

$$F(q) = \frac{1}{N} \sum_{k=1}^{N} d_{\mathcal{M}}^2(q, p_k) \quad \text{and} \quad \operatorname{grad} F(q) = -\frac{1}{2N} \sum_{k=1}^{N} \log_q p_k.$$

This algorithm `gradient_descent` is implemented in `Manopt.jl`. Furthermore, a library of cost functions, gradients, differentials, adjoint differentials, as well as proximal maps is available in the

package as well. They are implemented on arbitrary manifolds based only on `ManifoldsBase.jl` and can hence be used to design an optimization problem even independent of the manifold. One example that we also require here is the already mentioned gradient of the (exponentiated) distance function `distance(M,p,q)^s` with respect to `q`, where `s=2` is the default. This function is called `grad_distance(M, p, q, s)` in `Manopt.jl`.

Note that since `Manopt.jl` is based on the lightweight interface `ManifoldsBase.jl`, we need to load `Manifolds.jl` only to have our manifold of interest available.

```julia
using Manopt, Manifolds, LinearAlgebra, Random
Random.seed!(42)
M = Sphere(2)
N = 100
σ = π / 8
# generate random points on M centered at a random point p
p = rand(M)
pts = [exp(M, p, rand(M; vector_at=p, σ=σ)) for i in 1:N]
# define the loss function and its gradient
F(M, q) = sum(p -> distance(M, p, q)^2 / 2N, pts)
gradF(M, q) = sum(p -> grad_distance(M, p, q) / N, pts)
# compute the mean
p_mean = gradient_descent(M, F, gradF, pts[1])
```

`Manifolds.jl` itself includes two approaches for computing the weighted mean: gradient descent and on-line repeated geodesic interpolation [26]. Using these methods, implementations of weighted variance, skewness, kurtosis, and higher moments are also provided.

5.3 Tangent Space PCA

Thanks to the high composability of the Julia ecosystem and the design of the `Manifolds.jl` library, it is very easy to compose independent libraries to achieve new functionality. For example, tangent space principal component analysis (PCA) described by [20] can be split into (1) computing the Riemannian centers of mass and coordinates of tangent vectors using `Manifolds.jl` and (2) calculation of PCA vectors using `MultivariateStats.jl`. Using a d-dimensional manifold, tangent vectors might be stored in a format different from the d-dimensional vector. The interface `ManifoldsBase.jl` provides `c = get_coordinates(M, p, X, B)` to obtain a d-dimensional representation and `X = get_vector(M, p, x, B)` to reconstruct the tangent space format. For example, tangent vectors from $T_p\mathbb{S}^2$ are stored as vectors X orthogonal to p (in $\mathbb{R}^3$) and they have a representation in 2 coordinates. Both these functions require a basis `B`, which can be a full set of vectors in a `CachedBasis` or a memory-efficient type to represent a deterministic function of a `DefaultOrthonormalBasis` to not store the basis explicitly. The whole procedure is demonstrated by the following code.

```
using Manifolds, MultivariateStats, LinearAlgebra, Random
M = Sphere(2)
N = 100
σ = π / 8
# generate random points on M
p = rand(M)
pts = [exp(M, p, rand(M; vector_at=p, σ=σ)) for i in 1:N]
# compute their mean
m = mean(M, pts)
# compute logarithmic maps of points at their mean
vectors = map(p -> log(M, m, p), pts)
# choose a deterministic, default orthonormal basis of the tangent space at m
basis = DefaultOrthonormalBasis()
# compute coordinates of tangent vectors in the selected basis
coords = map(X -> get_coordinates(M, m, X, basis), vectors)
# flatten the array of coordinates to a matrix
coords_red = reduce(hcat, coords)
z = zeros(manifold_dimension(M))
# compute the first principal vector assuming zero mean
model = fit(PCA, coords_red; maxoutdim=1, mean=z)
# get the tangent vector corresponding to the first principal vector
X = get_vector(M, m, reconstruct(model, [1.0]), basis)
```

6 BENCHMARKS

This section presents and discusses benchmarks that compare performance of `Manifolds.jl` to other libraries. These are Geomstats [36] with its backends AutoGrad, NumPy, PyTorch, and TensorFlow; Geoopt [29], which is based on PyTorch; Pymanopt [41]; TensorFlow RiemOpt [39]; Manopt [14] in Matlab; and ROPTLIB [27] as a C++ library. We build three groups of manifolds: The Euclidean space $\mathbb{R}^n$, the hyperbolic space $\mathcal{H}^n$, and the sphere $\mathbb{S}^n$ for the dimensions $n = 2, 3, 4, 8, 16, 32, 2^{10}, 2^{15}, 2^{20}$. For these, ROPTLIB only provides an exponential map; the hyperbolic manifold is missing for ROPTLIB and Pymanopt.

The second set of manifolds are the special orthogonal group SO(n), the symmetric positive definite matrices $\mathcal{P}(n)$ with the affine invariant metric, and for a high dimensional test the power manifold $(\mathcal{P}(n))^{128\times128}$, where in all three cases we used $n = 2, 3, 4, 8, 16, 32$. Here again, ROPTLIB only provides an exponential map for the last two manifolds. The power manifold as well as a distance on SO(n) is not available in Geomstats and GeoOpt.

6.1 General Setup

A computer running Linux Mint 20 Ulyana with Intel(R) Core(TM) i7-9700KF CPU @ 3.60 GHz CPU and 32 GB of RAM was used to benchmark all libraries. In particular, no limitations were imposed on the number of threads.

The benchmarking was performed using points generated randomly by `Manifolds.jl`. Each benchmarked operation was performed using 100 pairs of these points 10^N times in a loop written in the language native to the library. The number N was chosen to be large enough to ensure that the entire benchmarking loop takes at least one tenth of a second to reduce the influence of system clock inaccuracy. Results of operations were accumulated in an array, which incurs a small performance penalty but prevents the compiler from optimizing out an otherwise unused computation. Details of benchmarks specific to particular libraries and the settings used are given in Appendix B.

6.2 Discussion

The results for the first experiment are shown in Figure 3. On all three manifolds, `Manifolds.jl` performs the fastest for the smaller dimensions $n = 2, 3, 4, 8, 16$, and for example in the sphere for $n = 3$ ROPTLIB performs the exponential map (bottom center bar plot) only a factor of 2 slower. This is likely due to the more sophisticated memory management and use of specialized linear algebra packages in `Manifolds.jl`.

For larger dimensions, e.g., for the manifold $\mathcal{M} = \mathbb{R}^{2^{10}}$, Pymanopt computes the logarithmic map about 1/6 faster. Similarly for $n = 32$, ROPTLIB computes the exponential map on the sphere $\mathbb{S}^n$ about 33% faster than `Manifolds.jl` and on $\mathbb{R}^n$ about four times faster.

For very high dimensions, RiemOpt with its TensorFlow backend is faster by a factor of up to 20 on most functions, and just for the hyperbolic manifold, our distance function is a factor of 4 faster. For such large dimensions, RiemOpt benefits from advanced automatic transformations available in the Tensorflow backend that cannot yet be easily replicated in Julia.

For the second experiment, the results are shown in Figure 4. Compared to ROPTLIB, we see the same effect: often the Julia package is a factor of at least 2 faster. Only for the exponential map on $\mathcal{P}(n)$ ROPTLIB is a factor of 2 faster, probably again due to different linear algebra library or memory management.

Concerning RiemOpt, again, only for the very high-dimensional power manifold, this Tensor-flow based package is usually a factor of approximately 10 faster. This starts already for the case $n = 3$, where `Manifolds.jl` is a factor of 2 slower for exponential and logarithmic maps but a factor of 5 faster for distance; all other packages are at least a factor of 4–5 slower, while Manopt seems to be the slowest by several orders of magnitude, probably due to the use of cell arrays in Matlab.

For the highest dimension $n = 32$ and the power manifold, both Pymanopt and Manopt reach the same order of magnitude as `Manifolds.jl` and Riemopt, where the latter is still a factor of 5 faster.

`Manifolds.jl` is the fastest package overall – on par with ROPTLIB wherever it provides an implementation – and only significantly surpassed by Riemopt for some large-scale manifolds of dimensions of more than 2^{16}.

We further compared the accuracy of the methods run in the benchmark. We first computed reference results using `Manifolds.jl` with `BigFloat` as number type and both `GenericLinearAlgebra.jl` and `GenericSchur.jl` for the more precise linear algebra routines. We then performed the accuracy test against these exact results for all above benchmarked functions and packages. The full results are given in Appendix A.

Overall, `Manifolds.jl` lies among the best performing ones here as well. Only for the symmetric positive definite matrices $\mathcal{P}(n)$ – more precisely the logarithmic and exponential map thereon – our library is slightly less accurate than others. The PyTorch backend seems to only reach a significantly lower accuracy, apparently due to an internal conversion to 32-bit floating point representation.

7 CONCLUSIONS

`Manifolds.jl` provides a comprehensive API and set of features for working with points on manifolds. It also offers one of the largest collections of implementations of manifolds. Using this API, `Manopt.jl` provides a library of algorithms for optimization on manifolds, and `ManifoldDiffEq.jl` offers a variety of solvers for ordinary differential equations. As a result, all manifolds defined with `ManifoldsBase.jl` are compatible with both libraries and can directly be used therein.

One prominent advantage of `Manifolds.jl` over similar packages is that, by using Julia, the "two-language-barrier" is overcome while keeping the code concise and expressive. Implementations in `Manifolds.jl` were demonstrated to be superior in performance to those in libraries
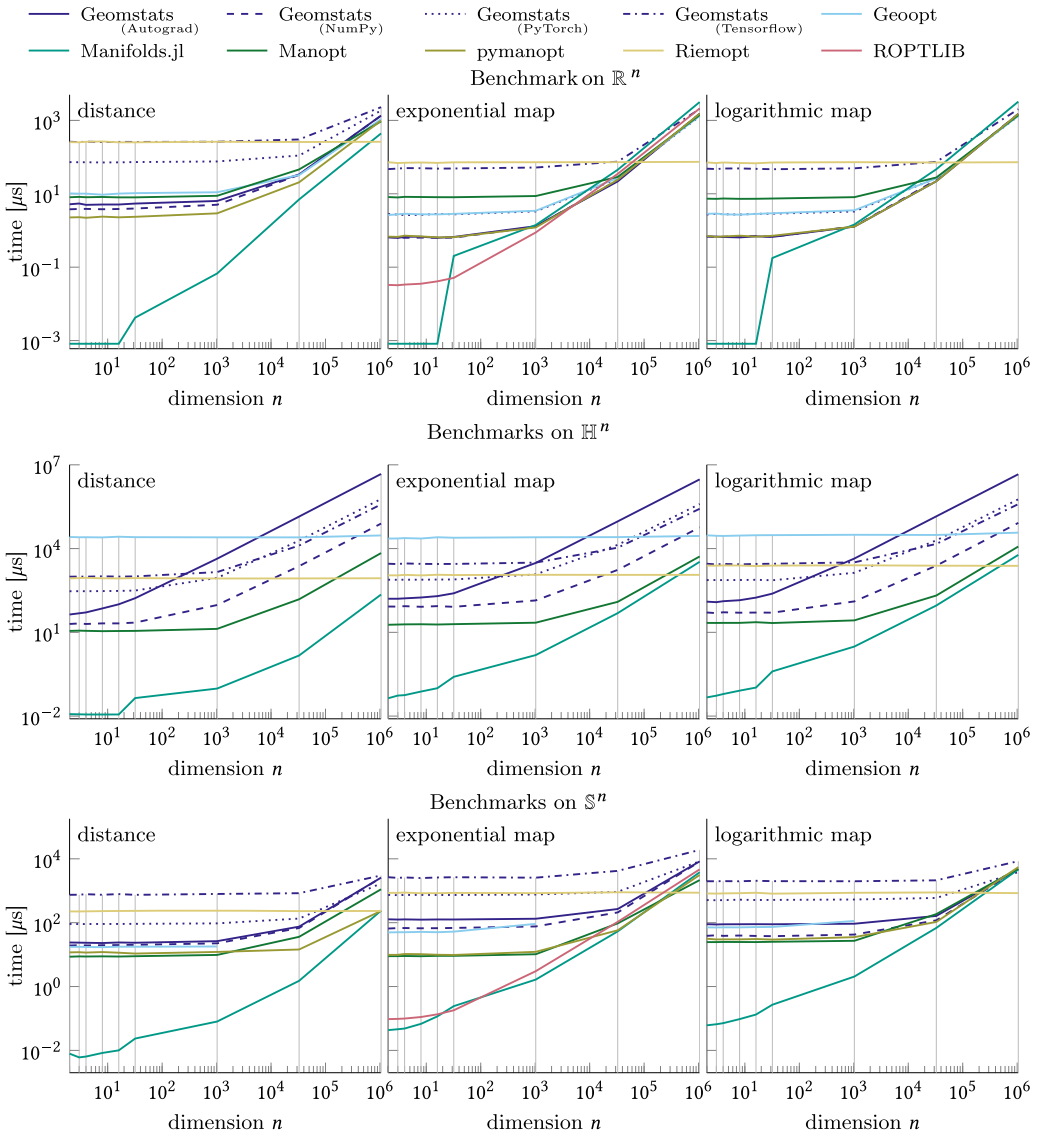
Fig. 3. Benchmark of distance (left), exponential maps (middle), and logarithmic maps (right) on constant curvature manifolds $\mathbb{R}^n$ (top) $\mathcal{H}^n$ (middle), and $\mathbb{S}^n$ for different dimensions n. The different colors correspond to the different software packages, where for Geomstats additionally the different line styles refer to different backends.

with interfaces in Python and Matlab, where only for large scale manifolds Riemopt outperforms `Manifolds.jl` slightly. The library is approximately twice as fast as ROPTLIB, though that is probably due to the linear algebra library.

However, this high performance did not require sacrificing clarity of the implementations themselves, the ease of use of a dynamic programming language, or computational accuracy.

Future work on `Manifolds.jl` will improve integration of Julia's automatic differentiation packages, add support for GPU acceleration, and expand the set of available manifolds and operations. Other plans include support for more statistical and machine learning methods.
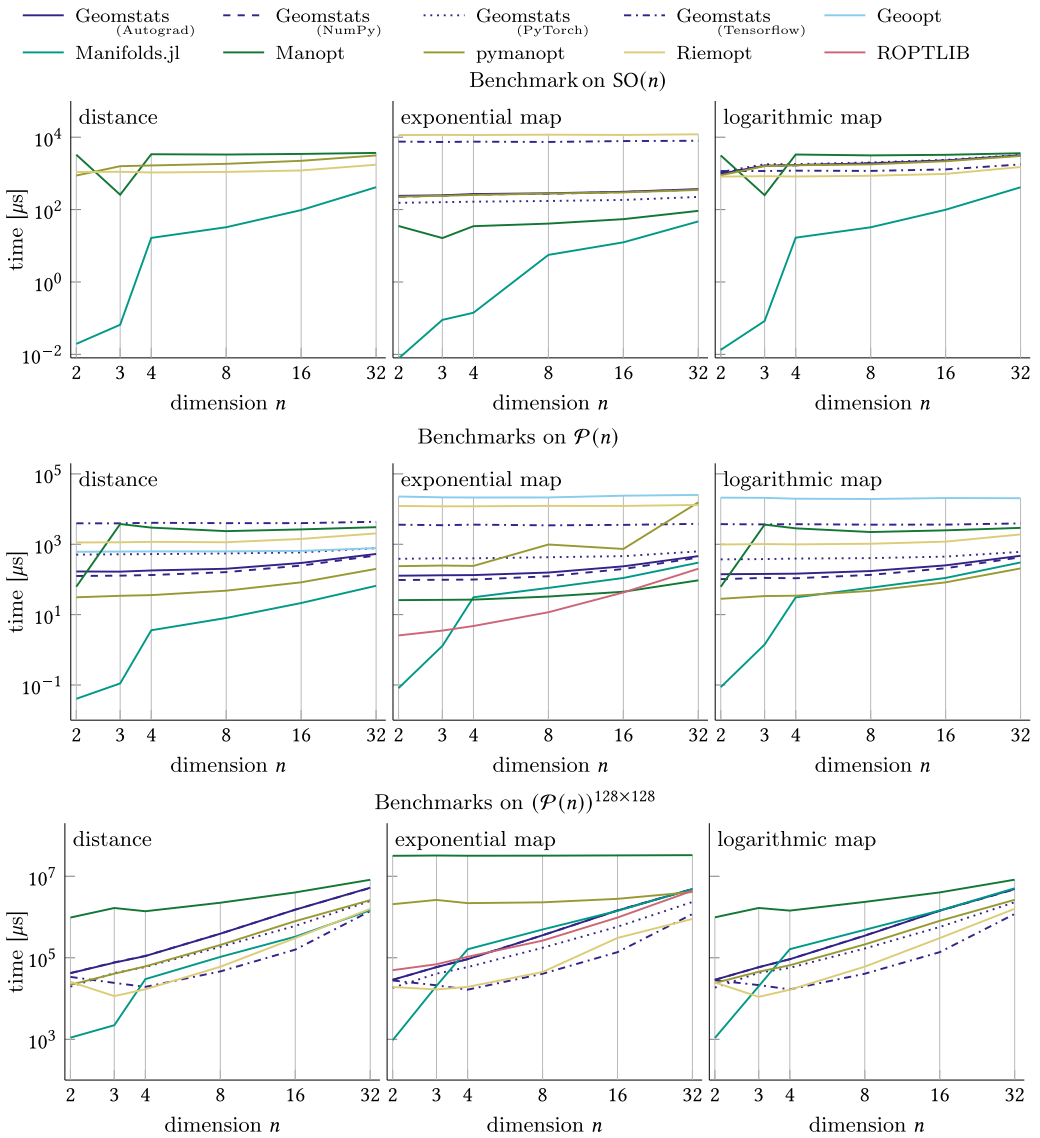
Fig. 4. Benchmark of distance (left), exponential maps (middle), and logarithmic maps (right) on the manifolds of rotations (top), manifold of symmetric positive definite matrices (middle), and its power manifold (bottom). The different colors correspond to the different software packages, where for Geomstats additionally the different line styles refer to different backends.

APPENDICES

A   ACCURACY COMPARISONS

Accuracy of computations performed by different libraries is compared in Figures 5 and 6. In all cases 10 points and tangent vectors were randomly sampled from those used for benchmarking. Reference results were obtained in `Manifolds.jl` using computations on arbitrary precision numbers (`BigFloat`) instead of `Float64`. Packages `GenericLinearAlgebra.jl` and `GenericSchur.jl`
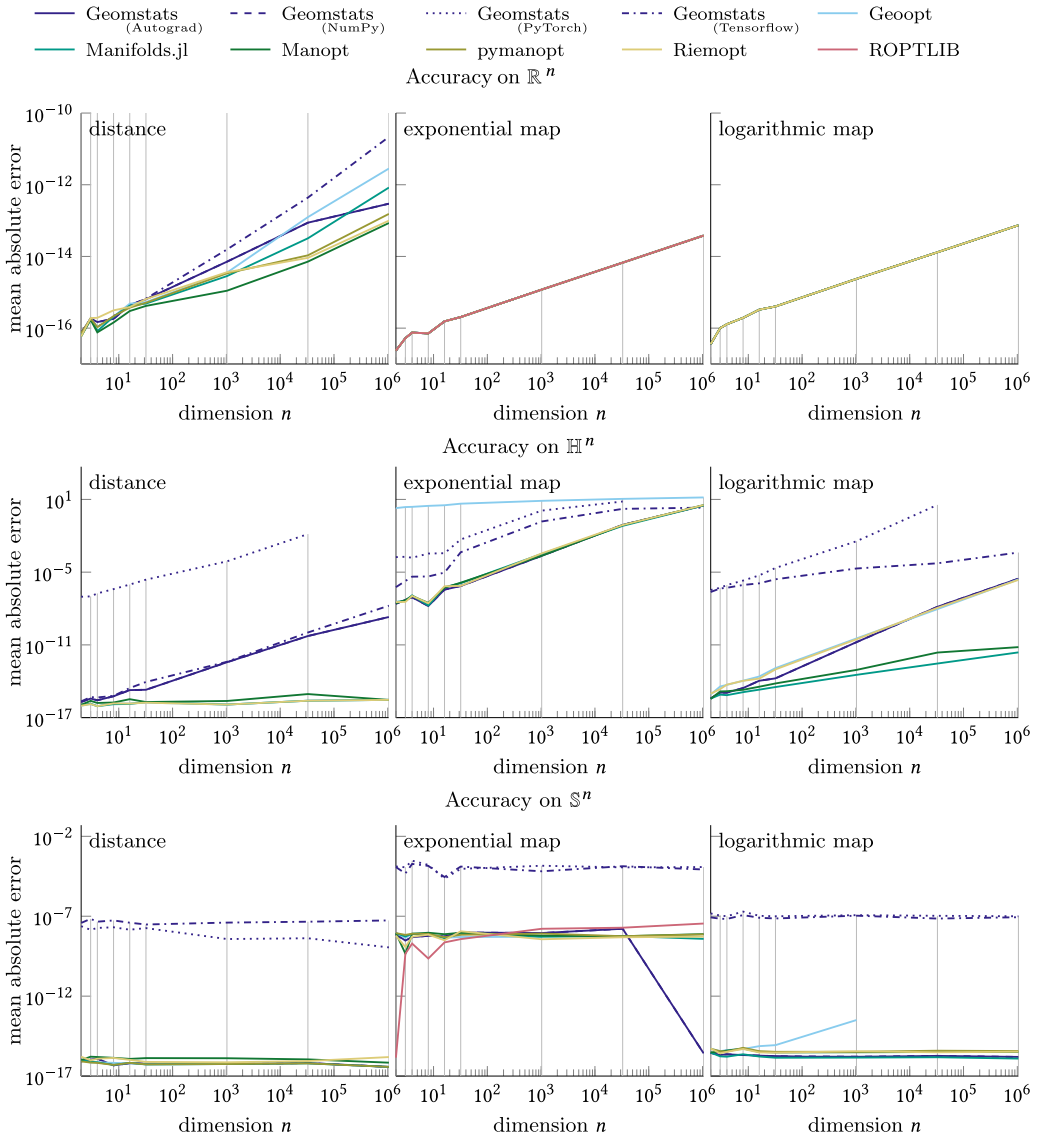
Fig. 5. Mean absolute error of distance (left), exponential maps (middle), and logarithmic maps (right) on constant curvature manifolds $\mathbb{R}^n$ (top) $\mathcal{H}^n$ (middle), and $\mathbb{S}^n$ for different dimensions n. The different colors correspond to the different software packages, where for Geomstats additionally the different line styles refer to different backends.

were used to supply linear algebra routines compatible with `BigFloat`. Additionally, generic matrix exponential and logarithm were implemented using generic eigen decomposition.

On $\mathbb{R}^n$ all libraries exhibit similar accuracy, except for distance calculation on high-dimensional manifolds. There is, however, little correlation between accuracy and performance.

On $\mathcal{H}^n$ geomstats is the least accurate and the slowest. Geoopt appears to return wrong results of exponential map. Manopt and `Manifolds.jl` implement a more accurate logarithmic map than other libraries, which is still fast to compute.
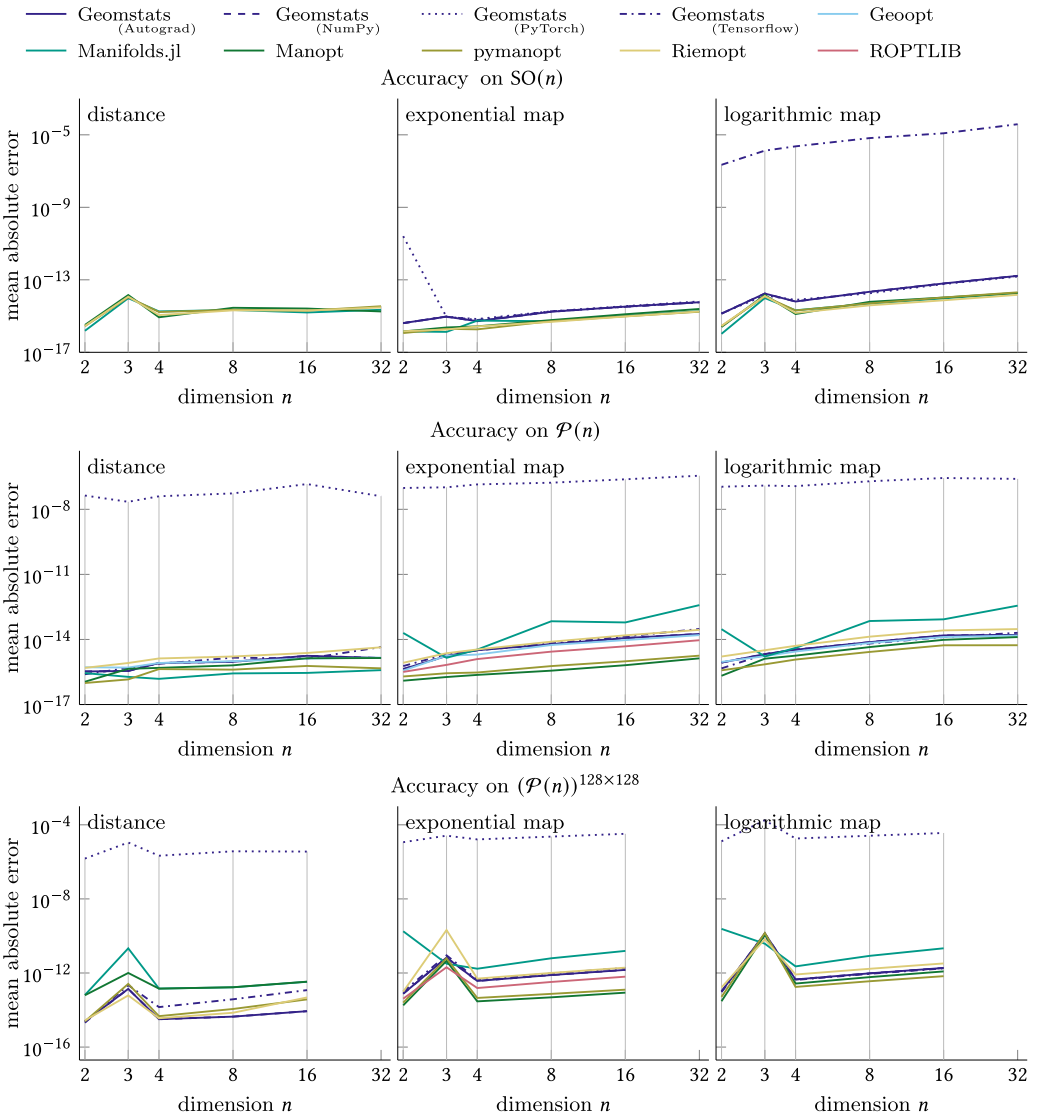
Fig. 6. Mean absolute error of distance (left), exponential maps (middle), and logarithmic maps (right) on the manifolds of rotations (top), manifold of symmetric positive definite matrices (middle), and its power manifold (bottom). The different colors correspond to the different software packages, where for Geomstats additionally the different line styles refer to different backends.

On $\mathbb{S}^n$ geomstats appears to convert intermediate results to a 32-bit floating point type with PyTorch and Tensorflow backends. Apparent high accuracy of exponential map with autograd backend in the high-dimensional case is likely an artifact caused by estimating accuracy based on 10 points. Logarithmic map in geoopt was less accurate than in other libraries.

Accuracy of SO(n) was roughly similar across all libraries except for exponential map on SO(2) in geomstats with Numpy and logarithmic map with Tensorflow.

On $\mathcal{P}(n)$ geomstats with Pytorch is the least accurate. `Manifolds.jl` has in some cases less accurate exponential and logarithmic maps than other libraries. These results roughly translate to

the power manifold $\mathcal{P}(n)^{128 \times 128}$. Accuracy on $\mathcal{P}(32)^{128 \times 128}$ is not included because calculating reference results would require large computational resources, while these results would be unlikely to provide any new insight.

B   LIBRARY-SPECIFIC DETAILS FOR THE SPEED AND ACCURACY BENCHMARK

For `Manifolds.jl`, version 0.8.57 of the library and `ManifoldsBase.jl` version 0.14.5 was used.

— $\mathbb{S}^n$: `Sphere(n)`
— $\mathbb{R}^n$: `Euclidean(n)`
— $\mathcal{H}^n$: `Hyperbolic(n)`
— $\mathcal{P}(n)$: `SymmetricPositiveDefinite(n)`
— $\mathcal{P}(n)^{128 \times 128}$:
   `PowerManifold(SymmetricPositiveDefinite(n), ArrayPowerRepresentation(),`
   `128, 128)`
— SO(n): `Rotations(n)`

Statically sized arrays from the `StaticArrays.jl` library were used to represent points and vectors for $\mathbb{S}^n$, $\mathbb{R}^n$ and $\mathcal{H}^n$ when $n \leq 16$, for $\mathcal{P}(n)$ when $n \in \{2, 3\}$ and for SO(n) when $n \leq 4$. For $\mathcal{P}(n)^{128 \times 128}$, the array type from `HybridArrays.jl`[5] version 0.4.15 was used with the first two axes statically sized. All other cases used the standard `Array` type.

All Python libraries were tested with Python version 3.9.16.final.0 (64 bit).

Geomstats version 2.5.0 was used with Autograd 1.5.0, NumPy 1.24.3, PyTorch 2.0.1, and Tensorflow 2.12.1. The following commands were used to construct manifolds:

— $\mathbb{S}^n$: `hypersphere.HypersphereMetric(n)`
— $\mathbb{R}^n$: `euclidean.EuclideanMetric(n)`
— $\mathcal{H}^n$: `hyperbolic.Hyperbolic(n).metric`
— $\mathcal{P}(n)$: `spd_matrices.SPDMetricAffine(n, 1)`
— $\mathcal{P}(n)^{128 \times 128}$:
   `M = SPDMatrices(3)`
   `M.metric.shape = (3, 3) # fixes an issue in geomstats`
   `MPm = NFoldMetric(M.metric, 128*128)`
— SO(n): `special_orthogonal.SpecialOrthogonal(n)`

All manifolds were taken from the `geomstats.geometry` path (omitted in the list above).

Geoopt version 0.5.0 was benchmarked. The following commands were used to construct manifolds:

— $\mathbb{S}^n$: `Sphere(torch.ones([n+1,n+1]))`
— $\mathbb{R}^n$: `Euclidean(1)`
— $\mathcal{H}^n$: `lorentz.math`

Manifolds listed above were taken from the `geoopt.manifolds` path (omitted in the list above). Other tested manifolds are not available in Geoopt.

Riemopt version 0.1.2 was benchmarked. The following commands were used to construct manifolds:

— $\mathbb{S}^n$: `Sphere()`
— $\mathbb{R}^n$: `Euclidean()`
— $\mathcal{H}^n$: `Hyperboloid()`

---

[5]see https://github.com/JuliaArrays/HybridArrays.jl

- $\mathcal{P}(n)$: `SPDAffineInvariant()`
- $\mathcal{P}(n)^{128\times128}$ `SPDAffineInvariant()`
- $\mathrm{SO}(n)$: `SpecialOrthogonal()`

All manifolds were taken from the `tensorflow_riemopt.manifolds` path (omitted in the list above).

Manopt version 7.1.0 (released on September 30, 2022) was used with Matlab version R2023a. The following commands were used to construct manifolds:

- $\mathbb{S}^n$: `spherefactory(n+1)`
- $\mathbb{R}^n$: `euclideanfactory(n)`
- $\mathcal{H}^n$: `hyperbolicfactory(n+1)`
- $\mathcal{P}(n)$: `sympositivedefinitefactory(n)`
- $\mathcal{P}(n)^{128\times128}$: `powermanifold(sympositivedefinitefactory(n), 128*128)`
- $\mathrm{SO}(n)$: `rotationsfactory(n)`

Pymanopt version 2.1.1 was used. The following commands were used to construct manifolds:

- $\mathbb{S}^n$: `Sphere(n+1)`
- $\mathbb{R}^n$: `euclidean.Euclidean(n)`
- $\mathcal{P}(n)$: `PositiveDefinite(n)`
- $\mathcal{P}(n)^{128\times128}$: `PositiveDefinite(n, 128 * 128), 128*128)`
- $\mathrm{SO}(n)$: `rotations.Rotations(n)`

Manifolds listed above were taken from the `Pymanopt.manifolds` path (omitted above).

ROPTLIB version 0.8 (released on August 11, 2020) was used, compiled using GCC version 9.3.0 using the provided makefile. The following commands were used to construct manifolds:

- $\mathbb{S}^n$: `Sphere(n+1)`
- $\mathbb{R}^n$: `Euclidean(n, "double")`
- $\mathcal{P}(n)$: `SPDManifold(n)`
- $\mathcal{P}(n)^{128\times128}$: `ProductManifold prodMani(1, &spdMani, 128 * 128);`, where the variable `spdMani` was defined as `SPDManifold spdMani(n);`
- $\mathrm{SO}(n)$ is not directly implemented (the manual suggests using a special case of Stiefel instead) and was thus not tested

Hyperbolic manifold was not available in ROPTLIB. The tested version of ROPTLIB does not provide logarithmic maps, and distance calculation can only be performed as a norm of a tangent vector and were excluded as well.

REFERENCES

[1] P.-A. Absil, P.-Y. Gousenbourger, P. Striewski, and B. Wirth. 2016. Differentiable piecewise-Bézier surfaces on Riemannian manifolds. *SIAM Journal on Imaging Sciences* 9, 4 (2016), 1788–1828. DOI: https://doi.org/10.1137/16M1057978

[2] P.-A. Absil, R. Mahony, and R. Sepulchre. 2008. *Optimization Algorithms on Matrix Manifolds*. Princeton University Press. DOI: https://doi.org/10.1515/9781400830244

[3] Mateusz Baran. 2018. Closest paths in graph drawings under an elastic metric. *International Journal of Applied Mathematics and Computer Science* 28, 2 (2018), 387–397. DOI: https://doi.org/10.2478/amcs-2018-0029

[4] M. Bačák. 2014. Computing medians and means in hadamard spaces. *SIAM Journal on Optimization* 24, 3 (2014), 1542–1566. DOI: https://doi.org/10.1137/140953393

[5] Miroslav Bačák, Ronny Bergmann, Gabriele Steidl, and Andreas Weinmann. 2016. A second order non-smooth variational model for restoring manifold-valued images. *SIAM Journal on Scientific Computing* 38, 1 (2016), A567–A597. DOI:https://doi.org/10.1137/15M101988X

[6] Ronny Bergmann. 2017. MVIRT, A toolbox for manifold-valued image restoration. In *Proceedings of the IEEE International Conference on Image Processing, IEEE ICIP 2017, Beijing, China, September 17–20, 2017.* DOI:https://doi.org/10.1109/ICIP.2017.8296271

[7] R. Bergmann. 2022. Manopt.jl: Optimization on manifolds in julia. *Journal of Open Source Software* 7, 70 (2022), 3866. DOI:https://doi.org/10.21105/joss.03866

[8] Ronny Bergmann, Jan Henrik Fitschen, Johannes Persch, and Gabriele Steidl. 2018. Priors with coupled first and second order differences for manifold-valued image processing. *Journal of Mathematical Imaging and Vision* 60, 9 (2018), 1459–1481. DOI:https://doi.org/10.1007/s10851-018-0840-y

[9] R. Bergmann and P.-Y. Gousenbourger. 2018. A variational model for data fitting on manifolds by minimizing the acceleration of a Bézier curve. *Frontiers in Applied Mathematics and Statistics* 4 (2018). DOI:https://doi.org/10.3389/fams.2018.00059 arXiv:1807.1009

[10] Ronny Bergmann, Johannes Persch, and Gabriele Steidl. 2016. A parallel douglas rachford algorithm for minimizing ROF-like functionals on images with values in symmetric hadamard manifolds. *SIAM Journal on Imaging Sciences* 9, 4 (2016), 901–937. DOI:https://doi.org/10.1137/15M1052858

[11] J. Bezanson, A. Edelman, S. Karpinski, and V. Shah. 2017. Julia: A fresh approach to numerical computing. *SIAM Rev.* 59, 1 (2017), 65–98. DOI:https://doi.org/10.1137/141000671

[12] Rajendra Bhatia, Tanvi Jain, and Yongdo Lim. 2019. On the bures–wasserstein distance between positive definite matrices. *Expositiones Mathematicae* 37, 2 (2019), 165–191. DOI:https://doi.org/10.1016/j.exmath.2018.01.002

[13] Nicolas Boumal. 2023. An introduction to optimization on smooth manifolds cambridge university press, 400 pages. DOI:https://doi.org/10.1017/9781009166164

[14] N. Boumal, B. Mishra, P.-A. Absil, and R. Sepulchre. 2014. Manopt, a matlab toolbox for optimization on manifolds. *Journal of Machine Learning Research* 15, 42 (2014), 1455–1459. Retrieved from https://www.manopt.org

[15] Roland Bürgmann, Paul A. Rosen, and Eric J. Fielding. 2000. Synthetic aperture radar interferometry to measure earth's surface topography and its deformation. *Annual Reviews Earth and Planetary Science* 28, 1 (2000), 169–209. DOI:https://doi.org/10.1146/annurev.earth.28.1.169

[16] Lu Chen, Vijay P. Singh, Shenglian Guo, Bin Fang, and Pan Liu. 2013. A new method for identification of flood seasons using directional statistics. *Hydrological Sciences Journal* 58, 1 (2013), 28–40. DOI:https://doi.org/10.1080/02626667.2012.743661

[17] L. M. Cruz-Orive, H. Hoppeler, O. Mathieu, and E. R. Weibel. 1985. Stereological analysis of anisotropic structures using directional statistics. *Journal of the Royal Statistical Society: Series C (Applied Statistics)* 34, 1 (1985), 14–32. DOI:https://doi.org/10.2307/2347881

[18] Manfredo Perdigão do Carmo. 1992. *Riemannian Geometry.* Birkhäuser Boston, Inc., Boston, MA.

[19] Ian L. Dryden, Alexey Koloydenko, and Diwei Zhou. 2009. Non-euclidean statistics for covariance matrices, with applications to diffusion tensor imaging. *Annals of Applied Statistics* 3, 3 (2009), 1102–1123. DOI:https://doi.org/10.1214/09-AOAS249 Publisher: Institute of Mathematical Statistics.

[20] P.T. Fletcher, C. Lu, S.M. Pizer, and S. Joshi. 2004. Principal geodesic analysis for the study of nonlinear statistics of shape. *IEEE Transactions on Medical Imaging* 23, 8 (2004), 995–1005. DOI:https://doi.org/10.1109/TMI.2004.831793

[21] P. Thomas Fletcher and Sarang Joshi. 2007-02. Riemannian geometry for the statistical analysis of diffusion tensor data. *Signal Processing* 87, 2 (2007-02), 250–262. DOI:https://doi.org/10.1016/j.sigpro.2005.12.018

[22] Aurobrata Ghosh, Maxime Descoteaux, and Rachid Deriche. 2008. Riemannian framework for estimating symmetric positive definite 4th order diffusion tensors. *Medical Image Computing and Computer-assisted Intervention: MICCAI ... International Conference on Medical Image Computing and Computer-Assisted Intervention* 11, Pt 1 (2008), 858–865. DOI:https://doi.org/10.1007/978-3-540-85988-8_102

[23] Pierre-Yves Gousenbourger, Estelle Massart, and P.-A. Absil. 2018. Data fitting on manifolds with composite Bézier-like curves and blended cubic splines. *Journal of Mathematical Imaging and Vision* 61, 5 (2018), 645–671. DOI:https://doi.org/10.1007/s10851-018-0865-2

[24] Ulf Grenander and Michael I. Miller. 1998. Computational anatomy: An emerging discipline. *Quarterly of Applied Mathematics* 56, 4 (1998), 617–694. DOI:https://doi.org/10.1090/qam/1668732

[25] Karsten Grove and Hermann Karcher. 1973. How to conjugate C^1-close group actions. *Mathematische Zeitschrift* 132, 1 (1973), 11–20. DOI:https://doi.org/10.1007/BF01214029

[26] Jeffrey Ho, Guang Cheng, Hesamoddin Salehian, and Baba Vemuri. 2013. Recursive karcher expectation estimators and geometric law of large numbers. In *Proceedings of the 16th International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research, Vol. 31).* Carlos M. Carvalho and Pradeep Ravikumar (Eds.), PMLR, Scottsdale, Arizona, 325–332. Retrieved from http://proceedings.mlr.press/v31/ho13a.html

[27] Wen Huang, P.-A. Absil, Kyle A. Gallivan, and Paul Hand. 2018. ROPTLIB: An object-oriented C++ library for optimization on riemannian manifolds. *ACM Transactions on Mathematical Software* 44, 4 (2018), 43:1–43:21. DOI: https://doi.org/10.1145/3218822

[28] H. Karcher. 1977. Riemannian center of mass and mollifier smoothing. *Communications on Pure and Applied Mathematics* 30, 5 (1977), 509–541. DOI: https://doi.org/10.1002/cpa.3160300502

[29] Max Kochurov, Rasul Karimov, and Serge Kozlukov. 2020. Geoopt: Riemannian optimization in pytorch. arXiv:2005.02819. Retrieved from https://arxiv.org/abs/2005.02819

[30] Line Kühnel and Stefan Sommer. 2017. Computational anatomy in theano. In *Proceedings of the Graphs in Biomedical Image Analysis, Computational Anatomy and Imaging Genetics.* M. Jorge Cardoso, Tal Arbel, Enzo Ferrante, Xavier Pennec, Adrian V. Dalca, Sarah Parisot, Sarang Joshi, Nematollah K. Batmanghelich, Aristeidis Sotiras, Mads Nielsen, Mert R. Sabuncu, Tom Fletcher, Li Shen, Stanley Durrleman, and Stefan Sommer (Eds.), Lecture Notes in Computer Science, Springer International Publishing, Cham, 164–176. DOI: https://doi.org/10.1007/978-3-319-67675-3_15

[31] John M. Lee. 2012. *Introduction to Smooth Manifolds* (2nd. ed.). Springer New York. DOI: https://doi.org/10.1007/978-1-4419-9982-5

[32] John M. Lee. 2018. *Introduction to Riemannian Manifolds.* Springer International Publishing. DOI: https://doi.org/10.1007/978-3-319-91755-9

[33] J.L. Marins, Xiaoping Yun, E.R. Bachmann, R.B. McGhee, and M.J. Zyda. 2001. An extended kalman filter for quaternion-based orientation estimation using MARG sensors. In *Proceedings of the 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No.01CH37180).* 2003–2011. DOI: https://doi.org/10.1109/IROS.2001.976367

[34] Mayank Meghwanshi, Pratik Jawanpuria, Anoop Kunchukuttan, Hiroyuki Kasai, and Bamdev Mishra. 2018. McTorch, a manifold optimization library for deep learning. arXiv:1810.01811. Retrieved from https://arxiv.org/abs/1810.01811

[35] Michael I. Miller. 2004. Computational anatomy: Shape, growth, and atrophy comparison via diffeomorphisms. *NeuroImage* 23 Suppl 1 (2004), S19–33. DOI: https://doi.org/10.1016/j.neuroimage.2004.07.021

[36] Nina Miolane, Nicolas Guigui, Alice Le Brigant, Johan Mathe, Benjamin Hou, Yann Thanwerdas, Stefan Heyder, Olivier Peltre, Niklas Koep, Hadi Zaatiti, Hatem Hajri, Yann Cabanes, Thomas Gerald, Paul Chauchat, Christian Shewmake, Daniel Brooks, Bernhard Kainz, Claire Donnat, Susan Holmes, and Xavier Pennec. 2020. Geomstats: A python package for riemannian geometry in machine learning. *Journal of Machine Learning Research* 21, 223 (2020), 1–9. Retrieved from http://jmlr.org/papers/v21/19-027.html

[37] Xavier Pennec, Stefan Sommer, and Tom Fletcher (Eds.). 2019. *Riemannian Geometric Statistics in Medical Image Analysis* (1st. ed.). Academic Press, San Diego.

[38] Krzysztof Rzecki and Mateusz Baran. 2021. Application of elastic shape analysis to user authentication and identification. *IEEE Transactions on Emerging Topics in Computing* 10, 2 (2021), 1157–1165. DOI: https://doi.org/10.1109/TETC.2021.3074242

[39] Oleg Smirnov. 2021. TensorFlow RiemOpt: a library for optimization on Riemannian manifolds. arXiv:2105.13921. Retrieved from https://arxiv.org/abs/2105.13921

[40] A. Srivastava, E. Klassen, S. H. Joshi, and I. H. Jermyn. 2011. Shape analysis of elastic curves in euclidean spaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33, 7 (2011), 1415–1428. DOI: https://doi.org/10.1109/TPAMI.2010.184

[41] James Townsend, Niklas Koep, and Sebastian Weichwald. 2016. Pymanopt: A python toolbox for optimization on manifolds using automatic differentiation. *Journal of Machine Learning Research* 17, 137 (2016), 1–5. Retrieved from http://jmlr.org/papers/v17/16-177.html

[42] Monika Tulik, Damian Kabat, Mateusz Baran, Radosaw Antoni Kycia, and Zbisaw Tabor. 2019. Use of statistical approaches to improve the quality control of the dose delivery in radiotherapy. *Physics in Medicine and Biology* 64, 14 (2019), 145018. DOI: https://doi.org/10.1088/1361-6560/ab25ab

[43] Pavan Turaga, Ashok Veeraraghavan, Anuj Srivastava, and Rama Chellappa. 2011. Statistical computations on grassmann and stiefel manifolds for image and video-based recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33, 11 (2011), 2273–2286. DOI: https://doi.org/10.1109/TPAMI.2011.52