juliacon

ManifoldsBase.jl – an interface for manifolds in Julia

Mateusz Baran[2] and Ronny Bergmann[1, 2]

[1]Department of Mathematical Sciences, Norwegian University of Science and Technology, Trondheim, Norway

[2]AGH University of Science and Technology, Kraków, Poland

ABSTRACT

We present an overview about the interface of the Julia package `ManifoldsBase.jl` to define and work with manifolds in Julia. The package is a central component of the `JuliaManifolds` ecosystem and is used by nearly all other packages in the ecosystem and several packages in the Julia community.
We discuss the main design ideas of the interface and present an overview of the features and functions provided in `ManifoldsBase.jl` 2.0.

Keywords

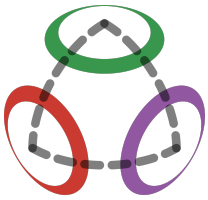
Julia, Riemannian manifolds, Lie groups, differential geometry, numerical analysis, scientific computing

1. Introduction

In many scenarios one encounters data that does not lie in a Euclidean space. Informally phrased, for certain data, we are not allowed do just add or scale them. Examples are rotations, data on spheres, in hyperbolic spaces, like when working with general relativity, symmetric positive definite matrices, like when working with covariance matrices, or when the data is either bases of subspaces or subspaces themselves, i.e. data on the Stiefel or Grassmann manifold. The data of interest for this paper still bear enough "structure", i.e. they live on smooth manifolds.
In a prior paper about `Manifolds.jl` [2], a partial presentation of the interface of `ManifoldsBase.jl` was given, when the interface was still in an early stage. A few fundamental design decisions have changed since then.
This paper presents the current state of the interface of `ManifoldsBase.jl` 2.0, since that state can be considered as having reached a very stable state. Its logo is shown in the Figure on the right.
The paper is organized as follows: in Section **??** we introduce the mathematical background and notation. In Section 4 we present the main design principles and structure of the interface. Section 3 presents the interface components in detail. In Section 6 we present an example of defining a new manifold, and Section 6 presents an overview of packages using `ManifoldsBase.jl`.

2. Mathematical Background

This section introduces the mathematical background and notation used throughout this paper, for more details see, e.g. [5, 1, 4].



Logo of `ManifoldsBase.jl`.

2.0.1 A topological manifold. A *manifold* $\mathcal{M}$ is a topological space[1] that locally resembles Euclidean space $\mathbb{R}^n$, $n \in \mathbb{N}$. To make this more precise, for every point $p \in \mathcal{M}$, there exists a neighborhood $\mathcal{U} \subseteq \mathcal{M}$ and a homeomorphism $\varphi\colon \mathcal{U} \to \mathbb{R}^n$. The dimension of the manifold is defined as the dimension of the Euclidean space $\mathbb{R}^n$. This can be interpreted as the number of degrees of freedom. The homeomorphism φ is called a *chart*, $\varphi(p)$ is called the *coordinates* of the point p in the chart φ. A collection of charts $\varphi_i\colon \mathcal{U}_i \to \mathbb{R}^n$, $i \in \mathcal{I}$ for some index set $\mathcal{I}$, that cover the manifold $\mathcal{M} = \bigcup_i \mathcal{U}_i$ is called an *atlas*. If we can "smoothly change charts", i.e. the transition maps $\varphi_j \circ \varphi_i^{-1}$ are smooth[2], we say that the atlas $\mathcal{A}$ is *smooth*. A manifold $\mathcal{M}$ together with a smooth atlas is called a *smooth manifold*[3]
Informally, a smooth manifold allows us to compute derivatives on the manifold by "looking at it through a chart". The smooth atlas then allows to argue, that this derivative is independent of the chosen chart.
As a guiding example, consider the n-dimensional sphere

$$\mathbb{S}^n = \{p \in \mathbb{R}^{n+1} \mid \|p\|_2 = 1\},$$

which is a smooth manifold of dimension n. A set of charts is for example the so-called *stereographic projections* onto the planes tangent to the sphere. Indeed, it is enough to consider the projections from the north and south pole to cover the sphere.

2.0.2 Tangent spaces. Consider any $p \in \mathcal{M}$ on a smooth manifold. In order to take (directional) derivatives at the point p, consider the set of all curves $c\colon (-\epsilon, \epsilon) \to \mathcal{M}$ with $c(0) = p$ that "run through p". We can now consider the derivative of the curve $\dot{c}$ around p in any chart φ such that $p \in \mathcal{U}$.

[1]we assume that the spaces we consider here are second countable and Hausdorff

[2]while one could consider different kinds of smoothness, C^1, C^2,..., for simplicity one usually just assumes C^∞

[3]more precisely, one usually considers a maximal smooth atlas collecting all compatible charts.

Collecting all $\dot{c}(0)$ yields "all possible walking directions", which is called the *tangent space* $T_p\mathcal{M}$ at $p \in \mathcal{M}$. Formally, we have to consider equivalence classes of curves, that have the same derivative in a chart as elements of $T_p\mathcal{M}$. The tangent space $T_p\mathcal{M}$ is a vector space of dimension n. The disjoint union of all tangent spaces $T\mathcal{M} = \bigsqcup_{p\in\mathcal{M}} T_p\mathcal{M}$ is called the *tangent bundle*.
For the sphere $\mathbb{S}^n$, the tangent space at a point $p \in \mathbb{S}^n$ is given by

$$T_p\mathbb{S}^n = \left\{X \in \mathbb{R}^{n+1} \;\middle|\; X^{\mathrm{T}}p = 0\right\},$$

i.e. the set of all vectors in $\mathbb{R}^{n+1}$ that are orthogonal to p. Visually, these are all vectors that are tangent to the sphere at the point p.
Since the tangent space is a vector space, we can introduce an inner product $\langle\cdot,\cdot\rangle_p \colon T_p\mathcal{M} \times T_p\mathcal{M} \to \mathbb{R}$ on each tangent space $T_p\mathcal{M}$. With two tangent vector fields $\mathcal{X}, \mathcal{Y} \colon \mathcal{M} \to T\mathcal{M}$, $\mathcal{X}(p), \mathcal{Y}(p) \in T_p\mathcal{M}$, we can consider the function $p \mapsto \langle\mathcal{X}(p), \mathcal{Y}(p)\rangle_p$. We say family of inner products $\langle\cdot,\cdot\rangle_p$ is *smoothly varying* (in p) if this map is a smooth map.
A *Riemannian manifold* is a smooth manifold $\mathcal{M}$ together with a smoothly varying inner product on the tangent spaces $T_p\mathcal{M}$, called the *Riemannian metric*.
All further terminology will be introduced throughout the introduction of the interface.

3. The interface working with manifolds

3.1 Main types

The central type to represent a manifold is `AbstractManifold{𝔽}` to represent a Riemannian manifold. The type parameter $\mathbb{F}$ indicates, whether the representation of points and tangent vectors are arrays or data types over the reals $\mathbb{R}$, complex numbers $\mathbb{C}$ or quaternions $\mathbb{H}$.
To build a possibly hierarchical structure of points or tangent vectors, the abstract types `AbstractManifoldPoint` and `AbstractTangentVector` exist[4]. These are, however not necessary to define or use the interface. For example the sphere can be used with plain `Vector{Float64}` as well. We recommend to use these as supertypes when defining new manifold point or tangent vector types, especially when these are mainly used to avoid type piracy.

3.2 General scheme of functions

Functions within `ManifoldsBase.jl` have a consistent order of arguments. The first argument should always be the manifold, followed by points and tangent vectors, and finally possibly positional arguments. when ever a function computes or returns a point, tangent vector, or array, both an allocating and an in-place version are available. The in-place version has an additional argument at the second position, i. e. directly following the manifold. For example generating a zero vector in the tangent space at a point p on a manifold $\mathcal{M}$ is either `zero_vector(M, p)` or `zero_vector!(M, X, p)` to return the result in the pre-allocated vector `X`. By default, the allocating version should allocate a result and call the in-place version. This can be realised by using `allocate_result(M, f, args...)` for a function `f(M, args...)`. This allows to for example allocate the correct type of a tangent vector, when the arguments only contain the manifold and points. This is the case for example in the function `zero_vector(M, p)`.

[4] the tangent vector type is more precisely a `AbstractFibreVector{TangentSpaceType}`, since other vector spaces can be modelled as well

Retraction	comment/code
`ApproximateRetraction`	(internal)
`EmbeddedRetraction{R}`	use retraction `R` in the embedding
`ExponentialRetraction`	falls back to `exp`
`PadeRetraction{m}`	of order m
`PolarRetraction`	
`ProductRetraction{R,S,...}`	
`ProjectionRetraction`	falls back to `project`
`QRRetraction`	
`SasakiRetraction`	
`SoftmaxRetraction`	
`StabilizedRetraction`	

Table 1. : Retraction types available in `ManifoldsBase.jl` 2.0.

Table 2. : Inverse retraction types available in `ManifoldsBase.jl` 2.0.

Inverse retraction	comment/code
`ApproximateInverseRetraction`	
`EmbeddedInverseRetraction{R}`	use `R` in the embedding
`InverseProductRetraction{R,S,...}`	
`LogarithmicInverseRetraction`	falls back to `log`
`PadeInverseRetraction`	
`PolarInverseRetraction`	
`ProjectionInverseRetraction`	falls back to `project`
`QRInverseRetraction`	
`SoftmaxInverseRetraction`	

3.3 Topological functions

—`manifold_dimension(M, p)`
—`is_point(M, p)` (mention error and kwargs...)
—`is_vector(M, X, p)`
—`isapprox(M, p, q)`
—`copy(M, p)` & `copy(M, p, X)`
—`copy_to!(M, q, p)` and `copy_to!(M, Y, X, p)`
—`rand(M)` & `rand(M; vector_at=p)`
—technically: `allocate(M, p, T)` & `allocate_result(M, f, args...)`
—`base_manifold(M)`
—`embed(M, p)` & `embed(M, p, X)`
—`project(M, p)` & `project(M, p, X)` (mention `embed_project`.)
—`get_embeddding(M)`
—`get_coordinates(M, p, X, b)` & `get_vector(M, p, c, b)`
—`zero_vector(M, p)`
—maybe a bit about bases?

3.4 Functions on smooth manifolds

—`inner(M, p, X, Y)`
—`angle(M, p, X, Y)`
—`norm(M, p, X)`

Table 3. : Vector transport types available in `ManifoldsBase.jl` 2.0.

Vector transport	comment/code
`EmbeddedVectorTransport{V}`	use `V` in the embedding
`ProductVectorTransport{V,W,...}`	
`ScaledVectorTransport`	

3.5 Functions on Riemannian manifolds

—`distance(M, p, q)`
—`exp(M, p, X)`
—`exp_fused(M, p, X, t)` (in footnote?)
—`retract(M, p, X, m)`
—`retract_fused(M, p, X, t, m)` (in footnote?)
—`geodesic(M, p, X, t)`
—`shortest_geodesic(M, p, q, t)`
—`injection_radius(M, p)`
—`log(M, p, q)`
—`inverse_retract(M, p, q, m)`
—`is_flat(M)`
—`parallel_transport_to(M, p, X, p)`
—`parallel_transport_direction(M, p, X, d)`
—`riemannian_tensor(M, p, X, Y, Z)`
—`sectional_curvature(M, p, X, Y)` (mention max/min)
—`ricci_curvature(M, p, X)`
—`Weingarten(M, p, X, V)`
—`vector_transport_to(M, p, X, q, vtm)`
—`vector_transport_direction(M, p, X, d, vtm)`

4. The Interface to define new manifolds.

This section presents the main data structures and design principles of `ManifoldsBase.jl`.

4.1 Avoiding ambiguities with 3 layers of multiple dispatch.

Multiple dispatch is used in three separate layers to avoid ambiguities. The parameters of a function are groups into three parts: the manifold, the points and tangent vectors, and finally positional arguments. A concrete example would be the retraction `retract(M, p, X, m)`, that besides a manifold `M`, a point `p` and a tangent vector `X` at $p \in \mathcal{M}$, has a positional `m` as the type of retraction, that has a reasonable default.

Layer I.. On the first layer, only the manifold type should be used to dispatch the function. This allows to define functions for specific manifolds or generic manifold types. This layer should pass on to layer two. A user working on manifolds should usually only interact with functions defined on this layer.

Layer II.. On the second layer, dispatching on certain positional arguments should be done. This allows to define functions that depend on specific positional arguments for all manifolds. Functions in this layer should be prefixed with an underscore `_` and should pass on to layer three. For our retraction example, this would be `_retract(M, p, X, ::ExponentialRetraction)` which would pass to the exponential map for any manifold with any point/tangent vector representation.

Layer III.. On the third layer, dispatching on the point and tangent vector types should be done. This is usually also on concrete manifold types. Functions on this layer should usually have a postfix indicating the kind of positional arguments or be prefixed with two underscores. This layer would allocate a result and call the in-place version of the same function. A user implementing a new manifold would usually only have to implement the in-place functions on this layer. For a retraction example, consider the `ProjectionRetraction`, for which layer two would pass to a function `retract_project(M, p, X)`. This function would allocate the result `Y` and call `retract_project!(M, Y, p, X)`. This function defaults to calling `project!(M, Y, p, X)`, but also allows a user to implement a more efficient version, either for a specific manifold or for specific point or tangent vector types.
If there are no positional arguments, layer two and hence functions with the double underscore prefix or a suffix can be skipped.

4.2 A trait system to avoid code duplication.

Certain properties of a manifold can be accessed via traits[5]. This aims to avoid code duplication. A manifold using this trait system has to use the `AbstractDecoratorManifold{𝔽}` as supertype.
Then, on layer one, function arguments are prefixed with an argument of type `AbstractForwardingType`, which is used to "forward" the function call, e. g. to the embedding.
They can also be "marked" as `StopForwardingType` to indicate, that something is not passed, but needs to actually be implemented.
For the example of embedded manifolds, the `EmbeddedForwardingType` is used to distinguish between different types of embeddings,, e. g. isometric or submanifold type embeddings and whether the representation does change or not.
As a concrete example consider again the sphere $\mathbb{S}^n$ as embedded submanifold of $\mathbb{R}^{n+1}$. Here, the function `is_point(M, p)` is implemented by forwarding to first check that `p` is a point on the embedding manifold $\mathbb{R}^{n+1}$ and then check that $\|p\|_2 = 1$. We avoid implemeting the test in the embedding. Similarly since the sphere is an `IsometricallyEmbeddedManifold`, also `inner(M, p, X, Y)` is forwarded to the embedding using the `SimpleForwardingType`, i.e. without any changes to the point or tangent vector representation.

5. An example

5.1 Defining an own manifold

5.2 Adding a second metric to an existing manifold

5.3 Implementing a generic algorithm on a manifold

6. Where the interface is used

—Manifolds.jl
—Manopt.jl [3]
—ManifoldDiff.jl
—ManifoldDiffEq.jl
—GeometricKalman.jl
—ExponentialFamilyManifolds.jl (Mykola) - and a second one he built on that?
—ROMe.jl (?)

[5] This is sometimes referred to as Tim Holy's Traits Trick (THTT) from the issue github.com/JuliaLang/julia#2345

7. References

[1] Pierre-Antoine Absil, Robert Mahony, and Rodolphe Sepulchre. *Optimization Algorithms on Matrix Manifolds*. Princeton University Press. doi:10.1515/9781400830244.

[2] Seth D. Axen, Mateusz Baran, Ronny Bergmann, and Krzysztof Rzecki. Manifolds.jl: An extensible julia framework for data analysis on manifolds. *ACM Transactions on Mathematical Software*, 49(4), dec 2023. doi:10.1145/3618296.

[3] Ronny Bergmann. Manopt.jl: Optimization on manifolds in Julia. *Journal of Open Source Software*, 7(70):3866, 2022. doi:10.21105/joss.03866.

[4] Manfredo Perdigão do Carmo. *Riemannian Geometry*. Mathematics: Theory & Applications. Birkhäuser Boston, Inc., 1992.

[5] John M. Lee. *Introduction to Smooth Manifolds*, volume 218 of *Graduate Texts in Mathematics*. Springer, New York. doi:10.1007/978-0-387-21752-9.