![juliacon]

Groups and smooth geometry using LieGroups.jl

Ronny Bergmann[1]

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

ABSTRACT

`LieGroups.jl` is a Julia package that provides an interface to define Lie groups as well as the corresponding Lie algebra and Lie group actions. The package also offers a well-documented, performant, and well-tested library of existing Lie groups, their algebra and corresponding group actions.
This paper presents the main features of the interfaces and how that is integrated within the `JuliaManifolds` ecosystem. We further present an overview on existing Lie groups implemented in `LieGroups.jl` as well as how to get started to use the package in practice.

Keywords

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

1. Introduction

In many situations, one encounters data that does not reside in a vector space. We can hence not use standard linear algebra tools to work with such data. For example in robotics, the configuration space of a rigid body in three-dimensional space is given by the special Euclidean group $SE(3)$, consisting of all translations and rotations. A subset of these is the space of rotations, given by the special orthogonal group $SO(3)$, or more generally $SO(n)$ in n-dimensional space.

These spaces are examples of Lie groups, formally defined as a smooth manifold equipped with a group structure. They have applications in physics, robotics, stochastic processes, information geometry, and many other areas see [5, 6], but are also interesting from their mathematical viewpoint [8] and their numerical aspects, for example when solving differential equations on Lie groups [9]. The package `LieGroups.jl`[1] provides an easy access to both defining and using Lie groups within the Julia programming language [4] by defining an interface of Lie groups, as well as implementing a library of Lie groups, that can directly be used.

This paper provides an overview of the main features of `LieGroups.jl` as of version 0.1.6. After introducing the necessary mathematical background and notation in Section 2, we present the interface in Section 3, split into the Lie group, the Lie algebra, and group actions. In Section 4, we present an overview of currently implemented Lie groups. Finally, in Section 5, we demonstrate how to get started and use `LieGroups.jl`.

---

[1]Available at juliamanifolds.github.io/LieGroups.jl/stable/, see also the zenodo archive [1].
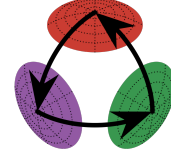
Logo of `LieGroups.jl`.

2. Mathematical Background

The following notation and definitions follow the text book [8], for more details on Riemannian manifolds, see also [7].

2.1 Lie groups

We denote a Lie group by $\mathcal{G} = (\mathcal{M}, \cdot)$ where $\mathcal{M}$ is a smooth manifold and $\cdot$ is the group operation. A smooth manifold $\mathcal{M}$ is a topological space that is locally isomorphic to an Euclidean space $\mathbb{R}^n$ for some $n \in \mathbb{N}$, but globally may have a different topology. We call n the dimension of the manifold $\mathcal{M}$, denoted by $\dim(\mathcal{M}) = n$. As an example, take the 2-dimensional sphere

$$\mathbb{S}^2 = \left\{ p \in \mathbb{R}^3 \mid \|p\| = 1 \right\},$$

which locally looks like $\mathbb{R}^2$, think of charts in an atlas, but globally it is not. Finally we denote the tangent space at a point $p \in \mathcal{M}$ by $T_p\mathcal{M}$. This can be thought of as all "velocities" (direction and speed) in which a curve can "pass through" a point. Formally it is set the equivalence classes of derivatives of smooth curves. Each tangent space $T_p\mathcal{M}$ is a n-dimensional vector space and we call the disjoint union of all tangent spaces

$$T\mathcal{M} = \dot{\bigcup}_{p \in \mathcal{M}} T_p\mathcal{M}$$

the *tangent bundle* of $\mathcal{M}$.

As a group operation $\cdot \colon \mathcal{G} \times \mathcal{G} \to \mathcal{G}$ has to satisfy the group axioms: associativity, existence of an identity element $e \in \mathcal{G}$, and existence of inverses $g^{-1} \in \mathcal{G}$ for all $g \in \mathcal{G}$. Furthermore the group operation $\cdot$ (on $\mathcal{G} \times \mathcal{G}$) and the inversion map $\iota \colon \mathcal{G} \to \mathcal{G}, g \mapsto g^{-1}$ have to be smooth maps. As an example, consider the special orthogonal group $SO(n)$, consisting of all $n \times n$ orthogonal matrices, with determinant 1, that is for $p \in SO(n)$, we have $p^{\mathrm{T}}p = I$ and $\det(p) = 1$ together with the group operation $\cdot$ given by matrix multiplication. For $n = 2$ these are rotations in the plane, hence each operation can be identified with an angle $\alpha \in [-\pi, \pi)^2$, or, continuously, with the circle. The identity element is given by the identity matrix I (or the angle $\alpha = 0$) and the inverse of a rotation matrix is given by its transpose (or an angle $-\alpha$).

---

[2]The identification is not continuous.

2.2 Tangent spaces and the Lie algebra

The tangent space at the identity element $e \in \mathcal{G}$, denoted by $\mathfrak{g} = T_e\mathcal{G}$, plays a special role and is called the *Lie algebra* of the Lie group $\mathcal{G}$. The reason is that to represent arbitrary tangent vectors $X \in T_g\mathcal{G}$ at a point $g \in \mathcal{G}$ we can use the group operation: we denote by $\lambda_g(h) = g \cdot h$ the left multiplication with $g, h \in \mathcal{G}$. Then, using the differential (or push forward) $D\lambda_g(h): T_h\mathcal{G} \to T_{g \cdot h}\mathcal{G}$[3], we can generate a so-called *left-invariant vector field* $\mathcal{X}(g) := D\lambda_g(e)[X]$ which is uniquely determined by the choice of $X \in \mathfrak{g}$.[4] Hence we can identify tangent vectors $\mathcal{X}(g) \in T_g\mathcal{G}$ at arbitrary points $g \in \mathcal{G}$ with X from the Lie algebra $\mathfrak{g}$.

As a first example, consider $\mathcal{G} = (\mathbb{R}^n, +)$, where the tangent space at any point g is again $\mathbb{R}^n$, especially at the identity $e = 0$. We further have $\lambda_g(h) = g + h$ and hence $D\lambda_g(h)[X] = X$ for all $g, h, X \in \mathbb{R}^n$. Here, a tangent vector X induces the constraint vector field $\mathcal{X}(g) = X$ for all $g \in \mathbb{R}^n$.

As a second example, consider again the special orthogonal group $\mathrm{SO}(n)$. The tangent space at the identity element $e = I$ is given by the Lie algebra $\mathfrak{so}(n) = T_e\mathrm{SO}(n) = \{X \in \mathbb{R}^{n \times n} \mid X = -X^\mathrm{T}\}$ consists of all skew-symmetric $n \times n$ matrices. For $g, h \in \mathrm{SO}(n)$ and $X \in \mathfrak{so}(n)$ we have $\lambda_g(h) = g \cdot h$ and hence $D\lambda_g(h)[X] = g \cdot X$. Here, the tangent vector $X \in \mathfrak{so}(n)$ induces the left-invariant vector field $\mathcal{X}(g) = g \cdot X \in T_g\mathrm{SO}(n)$, $g \in \mathrm{SO}(n)$. In other words, this formulation allows to represent tangent vectors $Y \in T_g\mathrm{SO}(n)$ also using $X = g^{-1}Y \in \mathfrak{g}$.

An important tool to "move around" on the Lie group is the exponential map. The (Lie group) exponential function $\exp: \mathfrak{g} \to \mathcal{G}$ maps elements from the Lie algebra to the Lie group, and is formally defined [8, Def. 9.2.2] by evaluating the unique curve $\gamma: \mathbb{R} \to \mathcal{G}$ that fulfils the differential equation

$$\gamma'(t) = D\lambda_{\gamma(t)}(e)[X] \quad \text{with} \quad \gamma(0) = e \text{ and } \gamma'(0) = X$$

at time $t = 1$, that is $\exp(X) = \gamma(1)$. This can be interpreted as starting at the identity element $e \in \mathcal{G}$ and following the curve whose velocity at each point is given by the left-invariant vector field induced by $X \in \mathfrak{g}$ for one time unit.

For $(\mathbb{R}^n, +)$ the exponential map is given by $\exp(X) = X$, $X \in \mathbb{R}^n$ and for $\mathrm{SO}(n)$ it is given by the matrix exponential $\exp(X) = \mathrm{e}^X$ for all $X \in \mathfrak{so}(n)$. Additionally for the unit circle in the complex plane, using the group operation of multiplication, the exponential map is given by the complex exponential $\exp(X) = \mathrm{e}^{\mathrm{i}X}$ for all $X \in \mathbb{R}$.

Concerning a metric on the tangent spaces, manifolds are turned into Riemannian manifolds when they are equipped with a Riemannian metric $\langle \cdot, \cdot \rangle_p: T_p\mathcal{M} \times T_p\mathcal{M} \to \mathbb{R}$ for each point $p \in \mathcal{M}$ that smoothly varies with p. For Lie groups, we can use a single inner product $\langle \cdot, \cdot \rangle$ on the Lie algebra $\mathfrak{g}$ and use the change in representation as mentioned above to define

$$\langle X, Y \rangle_g = \langle D\lambda_{g^{-1}}(g)[X], D\lambda_{g^{-1}}(g)[Y] \rangle_e$$

for all $X, Y \in T_g\mathcal{G}$ and $g \in \mathcal{G}$. Representing tangent vectors at arbitrary points $g \in \mathcal{G}$ using the Lie algebra $\mathfrak{g}$ yields here, that we can use the single inner product directly to evaluate this Riemannian metric. Hence representing tangent vectors using the Lie algebra is the default in the following.

For the two examples above we obtain the Euclidean inner product

$\langle X, Y \rangle = X^\mathrm{T}Y$ for $X, Y \in \mathbb{R}^n$ for $(\mathbb{R}^n, +)$ and the Frobenius inner product $\langle X, Y \rangle = \mathrm{trace}(X^\mathrm{T}Y)$, $X, Y \in \mathfrak{so}(n)$ can be used for $\mathrm{SO}(n)$.

Note that a Riemannian metric can also be used to define an exponential map using the Levi-Civita affine connection. This exponential map often differs from the Lie group exponential, in particular many Lie groups, such as the special Euclidean group in two or more dimensions, can not have a metric compatible with the Lie group exponential.

2.3 Group Actions

A *group action* of a Lie group $\mathcal{G}$ on a smooth manifold $\mathcal{M}$ is a smooth map $\sigma: \mathcal{G} \times \mathcal{M} \to \mathcal{M}$ such that for all $g, h \in \mathcal{G}$ and $p \in \mathcal{M}$ it holds that $\sigma(e, p) = p$ and $\sigma(g, \sigma(h, p)) = \sigma(g \cdot h, p)$.[5] Informally a group action describes how elements of the Lie group $\mathcal{G}$ "act on" points on the manifold $\mathcal{M}$. As an example, think of the special orthogonal group $\mathrm{SO}(3)$ acting on points on Euclidean space $\mathbb{R}^3$ "moving" them somewhere by rotating them around the origin. We obtain the group action $\sigma: \mathrm{SO}(3) \times \mathbb{R}^3 \to \mathbb{R}^3$, $\sigma(R, x) := Rx$.

The same action can also be applied to points from the sphere $\mathbb{S}^2$, resulting in a similar group action $\sigma: \mathbb{SO}(3) \times \mathbb{S}^2 \to \mathbb{S}^2$, $\sigma(R, p) := Rp$.

3. The interface

Since a Lie group $\mathcal{G}$ consists of two main components, the smooth manifold $\mathcal{M}$ and the group operation $\cdot$, we can reuse existing functionality from the existing interface for manifolds provided by `ManifoldsBase.jl`, and later concrete manifolds provided by `Manifolds.jl` [2]. This is done in a transparent way, i.e. the `AbstractLieGroup` itself is a subtype of `AbstractManifold` from `ManifoldsBase.jl` and can hence also be used in all existing places, as for example optimization on manifolds provided by `Manopt.jl` [3]. In notation, we use typewriter font to denote functions in the interface, but we keep the same letters of notation as before, i.e. $\mathcal{G}$ for a Lie group is `G` in code, a point $g \in \mathcal{G}$ is `g` in code, and so on, just that for the Lie algebra we use $\mathfrak{g}$ in both text and code.

The interface follows the philosophy of `ManifoldsBase.jl`, that the Lie group or algebra is the first argument and even a mutated argument comes second. Similarly, if a function computes something like a new point on the Lie group or a tangent vector, there also exists a variant, that computes this in-place. For example `identity_element(G)` returns the identity element e of a Lie group G in a default representation, `identity_element!(G, e)` writes the result into the pre-allocated variable `e`, which can also be used with other representation types.

3.1 Lie groups

The main type for Lie Groups is the `LieGroup{F, O, M} <: AbstractLieGroup`, which contains a manifold `M <: AbstractManifold{F}` as well as the group operation `O <: AbstractGroupOperation`, where $\mathbb{F}$ is the field of scalars used in the representation of the manifold, usually $\mathbb{F} = \mathbb{R}$ and in some cases $\mathbb{F} = \mathbb{C}$ or $\mathbb{F} = \mathbb{H}$.

Topological functions. From a topological viewpoint, one has to first distinguish between points on the Lie group and tan-

---

[3] MB: TODO: does the reader know that a differential is?

[4] Analogously, one can use the right multiplication $\rho_g(h) = h \cdot g$ and its differential $D\rho_g(h)$ to define right-invariant vector fields. `LieGroups.jl` uses left-invariant vector fields as default.

[5] This is the convention for left actions. Alternatively, right actions fulfil $\sigma(g, \sigma(h, p)) = \sigma(h \cdot g, p)$.

Table 1. : Group operations available in `LieGroups.jl`.

Group operation ·	comment/code
`AdditionGroupOperation`	falls back to +
`AbelianMultiplicationGroupOperation`	falls back to *
`LeftMatrixMultiplicationGroupOperation`	falls back to *
`PowerGroupOperation{Op}`	`PowerLieGroup`
`ProductGroupOperation{Ops}`	product Lie groups
`SemidirectProductGroupOperation`	semidirect products

gent vectors on the Lie algebra. `LieGroups.jl` introduces a
`AbstractLieGroupPoint <: AbstractManifoldPoint`, that
is used to represent points on the Lie group. This abstract super-
type is not necessary, points can be represented by arrays or other
types as well, but the type allows to introduce special types when
one Lie groups has *different representations* of points, that have to
be distinguished. When there are different representations, it is rec-
ommended to introduce a point type for each representation and
make the default one fall back to (just) using arrays.
Alternatively to points, also different manifolds can be used. This
is for example the case for the circle group $\mathbb{S}^1$, which can be rep-
resented as angles, points on the unit circle in $\mathbb{R}^2$ or as complex
numbers with unit norm, all three of which are different manifolds
to be used internally. To access the underlying manifold, the func-
tion `base_manifold(G)` is provided.
Beyond that, the following functions are available and pass directly
to the underlying manifold interface from `ManifoldsBase.jl`:
`is_approx(G, g, h)` to check for (approximate) equality of
two points, `is_point(G, g; error=:none)` to check if a point
is a valid point on the manifold, where the keyword can be
used to throw an `:error`, a `:warn` or an `:info`. Similarly
`manifold_dimension(G)` to get the dimension of the manifold,
`project(G, q)`[6] to project a point onto the manifold, as well as
`rand(G)` and `rand(rng, G)`[7] to sample a random point from the
manifold using a random number generator `rng` also pass directly
on to the manifold. To access the underlying manifold of the Lie
group one can use `base_manifold(G)`.

Group operation related functions. For the group operation · of a
Lie group $\mathcal{G}$, the abstract supertype `AbstractGroupOperation` is
mandatory. There are two main group operation types provided in
`LieGroups.jl`. On the one hand operations that fall back to using
· = + or · = *, where the latter has two variants, the one where it
is Abelian (like for numbers) and where it is not (like for matrices).
On the other hand, specific meta groups like the (direct) product of
two or more groups, the case where the product is taken for just one
group, i.e. the power group, or semidirect products. The available
group operation types are summarized in Table 1. More on these
meta groups is explained in Subsection 4.1.
For all these group operations, the following functions have default
implementations. They might not be the most performant ones for
every case, but provide working implementations out-of-the-box. It
is always possible to override these by defining a new group opera-
tion type and implementing the following functions for a `LieGroup`
with that new group operation type.
To avoid unnecessary allocations, the identity element can be
used without allocations, calling `e = IdentityElement(G)` or
`IdentityElement(op)` where `op` is the group operation of `G`. If
the actual value is needed, one can call

`identity_element(G)` or `identity_element(G, typeof(g))`,
where `g` is an element of the group. The first variant will always
generate a point in the default representation of the group, while
the latter will generate a point in the same representation as `g`.[8]
The two central functions are `compose(G, g, h)`[9] to compute the
group operation $g \cdot h$ for two points $g, h \in \mathcal{G}$ and the inverse
`inv(G, g)`[10] which computes g^{-1} for a point $g \in \mathcal{G}$.
Another function is $c_g(h) = g \cdot h \cdot g^{-1}$ called *conjugation*, which
is available as `conjugate(G, g, h)`[11].
For these three functions, also the differentials are available: adding
a `diff_` prefix to the function name and a final argument for the Lie
algebra tangent vector, for compose additionally the argument with
which to differentiate to, i.e. `diff_left_compose(G, g, h, X)`
and `diff_right_compose(G, g, h, X)`, respectively.[12]
Finally, there is a specific function for the differential of the conju-
gate at the identity $h = e$ called `adjoint(G, g, X)`[13] as well as
the combinations `inv_left_compose(G, g, h)`[14] for comput-
ing $g^{-1} \cdot h$ and `inv_right_compose(G, g, h)`[15] for computing
$g \cdot h^{-1}$. All three fall back to the previously defined functions, but
provide an interface to possibly implement more efficient variants
in case such exist.

Exponential and logarithm.. In `LieGroups.jl`, the exponential
function is given by `exp(G,X)`[16]. If we want to "start walking"
from another point, we can "move" (or interpret X) as being from
the tangent space at some point g and obtain by the chain rule
the *exponential function* $\exp_g\colon \mathfrak{g} \to \mathcal{G}$ defined by $\exp_g(X) \coloneqq
g \cdot \exp(X)$. In `LieGroups.jl`, the exponential map is imple-
mented as `exp(G, g, X)`[17]. As a word of carefulness, note that
on the underlying manifold, there is a further exponential map, the
Riemannian exponential map. The Riemannian exponential map
`exp(M, p, X)`[18] can be distinguished in that the first argument is a
manifold, and the tangent vector `X` has to be from the tangent space
at the point `p` on the manifold. Here, again, to access the Rieman-
nian exponential, one can use the base manifold of the Lie group,
i.e. `exp(base_manifold(G), g, X)` and have to make sure that
`X` is from the tangent space at `g`.
Locally around the identity element $e \in \mathcal{G}$, the exponential map is
a diffeomorphism, i.e. there exists an inverse map on some neigh-
bourhood of e. The logarithmic function $\log\colon \mathcal{G} \to \mathfrak{g}$ and logarith-
mic map $\log_g\colon \mathcal{G} \to \mathfrak{g}$ have the function signatures `log(G, g)`[19]
and `log(G, h, g)`[20], respectively, with the same caveat to the
Riemannian logarithmic map as for the exponential.
When the exponential and logarithmic map are not known in
closed form, it might be beneficial to use retractions and inverse
retractions instead, respectively. These are first or second order ap-
proximations of the exponential and logarithmic map, respectively

---

[6]in-place variant `project!(G, p, q)`.
[7]in-place variant `rand!(G, p)` and `rand!(rng, G, p)`, respectively.

---

[8]The in-place variant `identity_element!(G, e)` infers the representa-
tion from the type of `e`.
[9]in-place variant `compose!(G, k, g, h)`.
[10]in-place variant `inv!(G, h, g)`.
[11]in-place variant `conjugate!(G, k, g, h)`.
[12]in-place variants have a second argument `Y` to compute the result in.
[13]in-place variant `adjoint!(G, Y, g, X)`.
[14]in-place variant `inv_left_compose!(G, k, g, h)`
[15]in-place variant `inv_right_compose!(G, k, g, h)`
[16]in-place variant `exp!(G, g, X)`.
[17]in-place variant `exp!(G, h, g, X)`.
[18]with the same in-place signature as before `exp!(M, q, p, X)`.
[19]in-place variant `log!(G, X, g)`.
[20]in-place variant `log!(G, X, h, g)`.

and their interface is already provided in `ManifoldsBase.jl`[21]. In `LieGroups.jl`, one can either implement new variants based on a subtype of `AbstractRetractionMethod` and `AbstractInverseRetractionMethod`, resp., or use the wrappers `BaseManifoldRetraction` and `BaseManifoldInverseRetraction`, resp., to directly use the retraction and inverse retraction from the underlying manifold.

3.2 Lie algebras

Similar to points on the Lie group, when representing elements $X \in \mathfrak{g}$ from a Lie algebra, we do not type the general functions of the interface. This allows to use either plain arrays or own structures to represent these in code. The optional abstract supertype `AbstractLieAlgebraTangentVector`, which is a `AbstractTangentVector`, is provided to also distinguish different representations here. Keep in mind, that the Lie algebra is a vector space, so that addition and scalar multiplication are assumed to be defined in case you use an individual data type.

A major difference to the usual representation of tangent vectors on the underlying manifold is, that here the usual representation is done in the Lie algebra.

Topological basics. Given a Lie group G, we obtain the Lie algebra by calling `g = LieAlgebra(G)`. To access the Lie group again, use `base_lie_group(g)`. Similarly `base_manifold(g)` returns the underlying manifold of the Lie group.

As a technical detail, the Lie algebra is modelled as a tangent space[22] using the already mentioned `Identity(G)` as base point. A zero vector is generated via `zero_vector(g)` for the default representation and `zero_vector(g, T)` for a specific representation type `T`.[23]

The main topological function is to test the validity of a tangent vector `is_vector(g, X; error=:none)` using the same `error=` keyword as `is_point` on the Lie group.

Vector space related functions. For the following functions related to vector space features, using the Lie algebra $\mathfrak{g}$ as first argument is equivalent to specifying the Lie Group G and an arbitrary point $g \in \mathcal{G}$. This yields that the Lie group complies with the general interface for manifolds.

The inner product and norm on the Lie algebra are available as `inner(g, X, Y)` and `norm(g, X)`, respectively. Furthermore, there are two functions to convert between a coordinate-free representation of X as a tangent vector and its representation in coordinates of a basis.

Given a vector $c \in \mathbb{R}^n$, we obtain the tangent vector by calling `get_vector(g, c, b)` where `b` is a basis of the tangent space, i.e. a subtype of `AbstractBasis` and defaults within `LieGroups.jl` to `DefaultLieAlgebraOrthogonalBasis()`. When a Lie group has different representations of points and tangent vectors, these are distinguished by calling `get_vector(g, c, b, T)` specifying the tangent vector type.[24] Given a tangent vector $X \in \mathfrak{g}$, we obtain its coordinates by calling `get_coordinates(g, X, B)`[25], where the Basis is again optional.

Table 2. : Group actions types available in `LieGroups.jl`.

Group action type	comment/code
`AdditionGroupAction`	falls back to +
`ColumnwiseGroupAction{A}`	of a group action type `A`
`(Inverse)LeftGroupOperationAction`	requires $\mathcal{M} = \mathcal{G}$
`LeftMultiplicationAction`	falls back to *
`(Inverse)RightGroupOperationAction`	requires $\mathcal{M} = \mathcal{G}$
`RotationAroundAxisAction`	
`RowwiseGroupAction{A}`	of a group action type `A`

For the case of the default, the `DefaultLieAlgebraOrthogonalBasis()` the more commonly used names `hat(g, c)`[26] (again with an optional vector type `T`) and `vee(g, X)`[27] are implemented

Push forward and pull back of tangent vectors. To the representation of a tangent vector $X \in \mathfrak{g}$ to be represented in the tangent space at $g \in \mathcal{G}$ we have to use the push forward of the left multiplication with g, i.e. $D\lambda_g(e)[X]$. This is implemented as `push_forward_tangent(G, g, X)`[28] Conversely, to represent a tangent vector $Y \in T_g\mathcal{G}$ back in the Lie algebra $\mathfrak{g}$, we have to use the pull back of the left multiplication with g^{-1}, i.e. $D\lambda_{g^{-1}}(g)[Y]$. This is implemented as `pull_back_tangent(G, g, Y)`[29]

Jacobians. On Euclidean space, the terms differential and Jacobian are often used interchangeably. Within differential geometry, the differential (or push forward) of a smooth map $f\colon \mathcal{M} \to \mathcal{N}$ between two manifolds is a map between the corresponding tangent spaces. We denote it by $Df(p)\colon T_p\mathcal{M} \to T_{f(p)}\mathcal{N}$, where $p \in \mathcal{M}$. The differential is a linear map between the tangent spaces. However, in the coordinate-free representation of tangent vectors, there is no matrix representation of this linear map. But as soon as we choose bases for the tangent spaces and we represent tangent vectors in coordinates (cf. `get_coordinates` previously), we can represent the differential as a matrix, called the Jacobian matrix.

Currently for both `exp` as well as `conjugate`, the Jacobian is implemented. Note that for `exp`, the domain is the Lie algebra. Since that is a vector space, its tangent space(s) can be identified with the Lie algebra again. Hence both Jacobians are maps from the Lie algebra to itself. The signatures are `jacobian_exp(G, p, X, b)`[30] and `jacobian_conjugate(G, g, X, b)`[31], where `b` is again the basis to represent the tangent vectors in coordinates and defaults to `DefaultLieAlgebraOrthogonalBasis()`.

3.3 Group actions

Group actions as defined in Subsection **??** are implemented as a struct `GroupAction{G, M, A} <: AbstractGroupAction` containing the Lie group `G <: AbstractLieGroup`, the manifold `M`, and a group action tpe `A` in order to distinguish possible different ways a group could act on a manifold. Table 2 summarizes the currently available group action types in `LieGroups.jl`. the main functions then are `apply(a::GroupAction, g, p)`[32] to compute $\sigma(g, p)$ for a group action `a`, where

---

[21]see juliamanifolds.github.io/ManifoldsBase.jl/stable/retractions/.

[22]see the documentation of ManifoldsBase.TangentSpace

[23]the in-place variant `zero_vector!(g, X::T)` contains the type automatically.

[24]in-place variant `get_vector!(g, X::T, c, b)` contains the type information directly.

[25]in-place variant `get_coordinates!(g, c, X, b)`

[26]in-place variant `hat!(g, X::T, c)`

[27]in-place variant `vee!(g, c, X)`

[28]in-place variant `push_forward_tangent!(G, Y, g, X)`.

[29]in-place variant `pull_back_tangent!(G, X, g, Y)`.

[30]in-place variant `jacobian_exp!(g, J, p, X, b)`

[31]in-place variant `jacobian_conjugate!(g, J, g, X, b)`

[32]in-place variant `apply!(a::GroupAction, q, g, p)`.

Table 3. : Implemented Lie groups in `LieGroups.jl` version 0.1.6.

Lie Group $\mathcal{G}$	Symbol	comment/code
`CircleGroup()`	$\mathbb{S}^1$	3 representations
`GeneralLinearGroup(n, F)`	$\mathrm{GL}(n,\mathbb{F})$	$\mathbb{F} \in \{\mathbb{R},\mathbb{C}\}$
`HeisenbergGroup(n)`	$\mathrm{H}(n)$	
`OrthogonalGroup(n)`	$\mathrm{O}(n)$	
`PowerLieGroup(G, n)`	$\mathcal{G}^n$	`G^n`
`ProductLieGroup(G1, G2,...)`	$\mathcal{G}_1 \times \mathcal{G}_2 \times \dots$	`G1 × G2 × …`
Semidirect product group	$\mathcal{G}_1 \ltimes \mathcal{G}_2$	`G1 ⋉ G2`
	$\mathcal{G}_1 \rtimes \mathcal{G}_2$	`G1 ⋊ G2`
`SpecialEuclideanGroup(n)`	$\mathrm{SE}(n)$	
`SpecialGalileanGroup(n)`	$\mathrm{SGal}(n)$	
`SpecialLinearGroup(n, F)`	$\mathrm{SL}(n,\mathbb{F})$	$\mathbb{F} \in \{\mathbb{R},\mathbb{C}\}$
`SpecialOrthogonalGroup(n)`	$\mathrm{SO}(n)$	
`SpecialUnitaryGroup(n)`	$\mathrm{SU}(n)$	
`SymplecticGroup(n)`	$\mathrm{Sp}(2n)$	
`TranslationGroup(n; field=F)`	$\mathbb{F}^n$	$\mathbb{F} \in \{\mathbb{R},\mathbb{C},\mathbb{H}\}$
`UnitaryGroup(n)`	$\mathrm{U}(n)$	
`ValidationLieGroup(G)`		wraps `G` for numerical verification

$g \in \mathcal{G}$ and $p \in \mathcal{M}$, as well as its differential `diff_apply(a::GroupAction, g, p, X)`[33] to compute $D\sigma(g,p)[X]$ for a tangent vector $X \in T_p\mathcal{M}$ with respect to the manifold argument as well as `diff_group_apply(a::GroupAction, g, p, Y)`[34] to compute $D\sigma(g,p)[Y]$ for a Lie algebra element $Y \in \mathfrak{g}$.

Due to the property that $\sigma(g^{-1}, \sigma(g,p)) = p$ for all $g \in \mathcal{G}$ and $p \in \mathcal{M}$, there is also an inverse group action available $\sigma^{-1}(g,p) = \sigma(g^{-1}, p)$, which can be obtained by calling `inv(a)`.

TODO: Discuss left and right group actions here?

4. Implemented Lie groups

Table 3 summarizes the currently implemented Lie groups in `LieGroups.jl` 0.1.6.

4.1 Meta Lie groups

There are three Lie groups that are built upon other Lie groups. We mention them here briefly and point out specific functions and features that are additionally available for these.

4.1.1 Product Lie group. Given two Lie groups $\mathcal{G} = (\mathcal{M}, *)$ and $\mathcal{H} = (\mathcal{N}, \diamond)$, their (direct) product $\mathcal{G} \times \mathcal{H}$ is again a Lie group $(\mathcal{M} \times \mathcal{N}, \cdot)$ with group operation defined component-wise, i.e.

$$(g_1, h_1) \cdot (g_2, h_2) = (g_1 * g_2, h_1 \diamond h_2)$$

for $g_1, g_2 \in \mathcal{G}$ and $h_1, h_2 \in \mathcal{H}$. Since $\times$ is a binary operator in Julia, given two Lie Groups `G,H`, their product Lie group can be constructed via $G \times H$. The same applies for more than two groups, i.e. you can construct arbitrary long product Lie groups $G1 \times G2 \times G3 \times \dots$.

As for the representation of points and tangent vectors, the default representation requires to load `RecursiveArrayTools.jl`[35].

If you want to construct nested Products of Lie groups, e.g. $(\mathcal{G}_1 \times \mathcal{G}_2) \times \mathcal{G}_3$, the binary operator would not

---

[33]in-place variant `diff_apply!(a::GroupAction, Y, g, p, X)`.

[34]in-place variant `diff_group_apply!(a::GroupAction, Z, g, p, Y)`.

[35]see docs.sciml.ai/RecursiveArrayTools/stable/

---

work, since it automatically flattens the input; use `ProductLieGroup(ProductLieGroup(G1, G2), G3)` can be used instead.

4.1.2 Power Lie groups. Though mathematically the case of a product Lie Group, where $G = H$ is also (just) a product, computationally we can benefit from the special structure. Therefore the `Powermanifold(G, n1, n2, ...)` is provided to construct the $n_1 \times n_2 \times \dots$-fold Cartesian product of the Lie group G with itself. As a shortcut, you can use `G^n` to construct the n-fold Cartesian product of the Lie group G with itself.

Note that inheriting from the power manifold, there are two different data types to represent data, per default a single large array, or a

4.1.3 Semidirect product Lie groups.

4.2 A Lie group for Numerical verification

The `ValidationLieGroup` is a special Lie group that is intended for numerical verification and debugging of code using Lie groups. The `ValidationLieGroup` is implemented as a wrapper around any existing Lie group, `G2 = ValidationLieGroup(G)`. It provides additional functionality to check the correctness of computations involving the Lie group. Most prominently, all input and output of a function is checked for validity, e.g. by calling `is_point` and `is_vector` on all points and tangent vectors, respectively. Similarly the inner manifold is wrapped into the similar concept from `ManifoldsBase.jl`, the `ValidationManifold`. While by default these checks do result in errors, this can be changed setting the `error=` keyword to either `:warn` or `:info`.

4.3 Concrete Lie groups

For the `CircleGroup()`, three different representations of points are available: as angles in $[0, 2\pi)$, as complex numbers with unit norm, and as points on the unit circle in $\mathbb{R}^2$. These are constructed by specifying the representation of points, i.e. `CircleGroup(`$\mathbb{R}$`)`, `CircleGroup(`$\mathbb{C}$`)`, and `CircleGroup(`$\mathbb{R}^2$`)`, respectively. Note that each of these representations uses a different underlying manifold, namely `Circle(`$\mathbb{R}$`)`, `Circle(`$\mathbb{C}$`)`, and `Circle(`$\mathbb{R}^2$`)`, respectively. This is a case, where it is not only the representation of points and tangent vectors that differ, but indeed even the underlying manifold.

5. An example how to use `LieGroups.jl`

6. References

[1] Seth Axen, Mateusz Baran, Ronny Bergmann, Yueh-Hua Tu, and Olivier Verdier. LieGroups.jl, 2025. doi:10.5281/zenodo.15343362.

[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] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. Julia: A fresh approach to numerical computing. *SIAM review*, 59(1):65–98, 2017. doi:10.1137/141000671.

[5] Gregory S. Chirikjian. *Stochastic Models, Information Theory, and Lie Groups, Volume 1: Classical Results and Geometric Methods*. 2009. doi:10.1007/978-0-8176-4803-9.

[6] Gregory S. Chirikjian. *Stochastic Models, Information Theory, and Lie Groups, Volume 2: Analytic Methods and Modern Applications*. 2012. doi:10.1007/978-0-8176-4944-9.

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

[8] Joachim Hilgert and Karl-Hermann Neeb. *Structure and Geometry of Lie Groups*. Springer Monographs in Mathematics, 2012. doi:10.1007/978-0-387-84794-8.

[9] Hans Munthe-Kaas. Runge-kutta methods on lie groups. *BIT Numerical Mathematics*, 38(1):92–111, 1998. doi:10.1007/bf02510919.