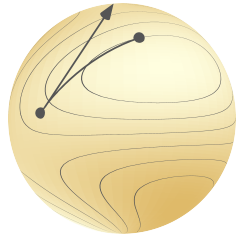

Norges teknisk-naturvitenskapelige universitet



# Manopt.jl

## Numerical Optimization on Manifolds in Julia

Ronny Bergmann

NTNU, Trondheim, Norway.

Geometric Methods in Optimization and Numerical Analysis

Oberwolfach Workshop 2630, July 21, 2026

# Motivation

For a function $f\colon \mathcal{M} \to \mathbb{R}$ defined
on a Riemannian manifold $\mathcal{M}$ compute

$$p^* \in \operatorname*{arg\,min}_{p \in \mathcal{M}} f(p)$$

- easily accessible in math & code
- interchangable & configurable algorithms & manifolds
- efficient & fast

**Manopt family.**

manoptjl.org [RB, 2022]

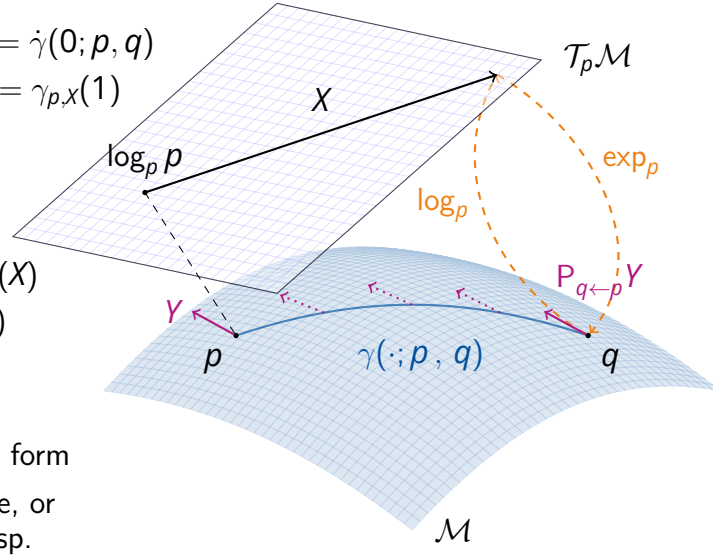
manopt.org [Boumal, Mishra, Absil, Sepulchre, 2014]

pymanopt.org [Townsend, Koep, Weichwald, 2016]

# A Riemannian Manifold $\mathcal{M}$

**Notation.**

- Logarithmic map $\log_p q = \dot{\gamma}(0; p, q)$
- Exponential map $\exp_p X = \gamma_{p,X}(1)$
- Geodesic $\gamma(\cdot; p\,,\, q)$
- Tangent space $\mathcal{T}_p\mathcal{M}$
- inner product $(\cdot\,,\,\cdot)_p$
- parallel transport $\mathrm{PT}_{p\leftarrow q}(X)$
- distance function $d(p, q)$




**Numerics.**

$\exp_p$, $\log_p$, $\mathrm{PT}_{p\leftarrow q}$ maybe not avail, efficiently/in closed form

$\Rightarrow$ use a retraction, its inverse, or a vector transport instead, resp.

# Why Julia?


- high-level language, open source
- multiple dispatch & operator overloading, for example
  `*(::Number, ::AbstractTangentVector)`
  allows to write $\alpha$`*X` for a tangent vector `X`
- just-in-time compilation, solves two-language problem
  $\Rightarrow$ "nice to write" and as fast as C/C++

**I like the community**

- very welcoming Discourse, Slack, Zulip, ...
- good package ecosystem organisation
- packages very modular and relatively easy get them to interact
- packages: environments, versioning and version resolution

# ManifoldsBase.jl – Motivation

**Goal.** Provide a generic interface to manifolds for

- defining own (new) manifolds
- implementing generic algorithms on an arbitrary manifold $\mathcal{M}$

**Manifold.** A Riemannian manifold as subtype of `<:AbstractManifold{𝔽}`

- $\mathbb{F} \in \{\mathbb{R}, \mathbb{C}, \mathbb{H}\}$: field the manifold is build on
- stores all "general" information, e. g. the manifold dimension
- example (from `Manifolds.jl`): `M = Sphere(2)`
- meta manifolds: product $\mathcal{M}_1 \times \mathcal{M}_2$, power $\mathcal{M}^n$, `TangentBundle(M)`

**Points and Tangent vectors.**

- by default not typed, often `<:AbstractArray`
- or use `<:AbstractManifoldPoint` and `<:AbstractTangentVector`

# ManifoldsBase.jl – Functions

**Goal.** Efficient and reusable functions.

**Functions.** We provide functions like

- `exp(M, p, X)`, `log(M, p, q)`, `inner(M, p, X, Y)`, `parallel_transport_to(M, p, X, q)`
- yields defaults for `norm(M, p, X)` or `shortest_geodesic(M, p, q)`
- `retract(M, p, X, method)` to approx. $\exp_p X$, with different `method`s,
- similarly `inverse_retract(M, p, q, m)` and `vector_transport_to(M, p, X, q, m)`

**Efficiency.** `exp(M, p, X)` allocates `q` a point and calls `exp!(M, q, p, X)`.

For a new manifold only implement `exp!(M, q, p, X)` to work in-place of `q`
$\Rightarrow$ avoid memory allocations where possible.

NTNU

Norges teknisk-naturvitenskapelige universitet

# Manopt.jl

# Problem and Solver State in Manopt.jl

## A problem

`<:AbstractManoptProblem`

containing at least

- an `AbstractManifold`
- an `AbstractManifoldObjective`

**Example.**

```
o = ManifoldGradientObjective(f, g)
problem = DefaultManoptProblem(M, o)
```

The objective can be decorated

- to cache function evaluations
- to count function evaluations

## A solver state

`<:AbstractManoptSolverState`

to specify/store

- algorithm parameters, e. g.
  - a `StoppingCriterion`
  - a `Stepsize`
- values like the iterate $p^{(k)}$.

**Example.**

```
state = GradientDescentState(M)
```

The state can be decorated

- to print debug
- to record values

# Two methods for a Random Walk solver

```
struct RandomWalkState{P, R, S}
    <: AbstractManoptSolverState
  p::P # current iterate
  q::P # best point visited
  σ::Float64
  rm::R # A retraction method
  stop::S # A StoppingCriterion
end
```

### Initialization

```
function initialize_solver!(
    mp::AbstractManoptProblem,
    r::RandomWalkState
) # Set best to init point
    M = get_manifold(mp)
    copyto!(M, r.q, r.p)
    return r
end
```

### The $k$th step

```
function step_solver!(
  mp::AbstractManoptProblem,
  r::RandomWalkState, k
) # generate and scale direction
  M = get_manifold(mp)
  X = rand(M; vector_at=r.p)
  X .*= r.σ/norm(M, r.p, X)
  # Walk
  retract!(M, r.p, r.p, X, r.rm)
  # Check/Store best
  if get_cost(mp, r.p) <
        get_cost(mp, r.q)
    copyto!(M, r.q, r.p)
  end
  return r
end
```

# A solver run

Setting up some small example manifold and cost

```
M = Sphere(2) # From Manifolds.jl
data = [[1.0, 0.0, 0.0],[0.0, 1.0, 0.0],[0.0, 0.0, 1.0]]
f(M,p) = sum( distance(M, p, q)^2 for q in data)
```

and a state and problem

```
state = RandomWalkState(
    copy(M, data[1]), # Initial p
    copy(M, data[1]), # Initial q
    0.1, # a value for σ
    ExponentialRetraction(), # use exp
    StopAfterIteration(1000), # when to stop
)
problem = DefaultManoptProblem(M, ManifoldCostObjective(f))
```

Running the solver: `solve!(problem, state);`

But this is not that easy to use nor comfortable

# Ease of use: High-level interfaces

All solvers have a simple function to call for example

```
gradient_descent(M, f, grad_f)      # starts at random point
gradient_descent!(M, f, grad_f, p) # starts at p, overwrites p
```

Parameters are passed as keyword arguments. All accept for example

- `stopping_criterion = StopWhenGradientNormLess(1e-9) | StopAfterIteration(1000)`
  Specify when to stop, can be combined with `|` and `&`
- `debug = [:Iteration, :Cost, "\n", 10, :Stop]`
  Print the $k$ and current cost every 10th iteration and reason to stop
- `record = [:Iterate, :Cost]`
  Record the current iterate $p^{(k)}$ and its cost value $f(p^{(k)})$
- `return_state = false` (default)
  return the whole solver state (instead of just the last iterate)
  then use `get_record(state)` to access vector of recorded values

# The JuliaManifolds ecosystem

We collect several packages in the GitHub organisation JuliaManifolds.

**ManifoldsBase.jl** An interface for Riemannian manifolds

**Manopt.jl** Numerical optimization on manifolds

**ManoptExamples.jl** Reproducible examples of solvers from `Manopt.jl`

**Manifolds.jl** A library of Riemannian manifolds

**LieGroups.jl** An interface for and a library of Lie groups

**ManifoldDiff.jl** differentiations of certain function and AD support for functions on manifolds

**ManifoldDiffEq.jl** use manifolds together `DiffEq.jl` to implement solvers for differential equations on manifolds / Lie groups

**ManifoldMakie.jl** plot manifold-valued data with `Makie.jl`

**ManifoldAsymptote.jl** render manifold-valued data with Asymptote

and a work-in-progress (help wanted!) `ManifoldDistributions.jl`

NTNU

Norges teknisk-naturvitenskapelige universitet

# Examples

# Riemannian Center of Mass

For $q_i \in \mathcal{M}$, $i = 1, \dots, m$, consider the Riemannian Center of Mass

$$f(p) := \frac{1}{2m} \sum_{i=1}^{m} d^2(p, q_i)$$

with Riemannian gradient

$$\operatorname{grad} f(p) = -\frac{1}{m} \sum_{i=1}^{m} \log_p q_i$$

```
using Manifolds, Manopt
m = 20; σ = π/8; M = Sphere(2)
p = [0.0, 0.0, 1.0]
Q = [ exp(M, p, σ * rand(M; vector_at=p))
    for _ in 1:m
]
f(M, p) = 1/(2m) * sum(
    distance(M, p, q)^2 for q in Q
)
grad_f(M, p) = - 1/m * sum(
    log(M, p, q) for q in Q
)

p1 = gradient_descent(M, f, grad_f, p0)
p2 = gradient_sampling(M, f, grad_f, p0)
```

(load `RipQP, QuadraticModels` for the subsolver)

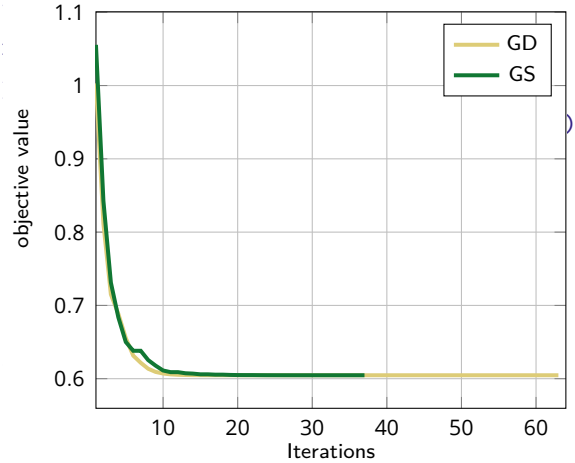
# Riemannian Center of Mass

For $q_i \in \mathcal{M}$, $i = 1, \dots, m$, consider the Riemannian Center of Mass

$$f(p) := \frac{1}{2m} \sum_{i=1}^{m} d^2(p, q_i)$$

with Riemannian gradient

$$\operatorname{grad} f(p) = -\frac{1}{m} \sum_{i=1}^{m} \log_p q_i$$




```
p1 = gradient_descent(M, f, grad_f, p0)
p2 = gradient_sampling(M, f, grad_f, p0)
```

(load `RipQP`, `QuadraticModels` for the subsolver)

# Riemannian Median

For the Riemannian Median

$$f(p) := \sum_{i=1}^{m} d(p, q_i)$$

the problem is nonsmooth.
Use the proximal map

$$\operatorname{prox}_{\lambda f}(p) := \operatorname*{arg\,min}_{q \in \mathcal{M}} d(q,p)^2 + \lambda f(q)$$

for each $f_i(p) = d(p, q_i)$.
Run the
cyclic proximal point algorithm

$$p_{k+\frac{i+1}{m}} = \operatorname{prox}_{\lambda_k f_i}(p_{k+\frac{i}{m}}),$$
$$i = 0, \dots, m-1,$$
$$k = 0, 1, \dots$$

```
using ManifoldDiff, Manifolds, Manopt
using ManifoldDiff: prox_distance
# [...same data Q and p0 as before...]

f(M, p) = 1/m * sum(
    distance(M, p, q) for q in Q
)
p_f = Function[ #An array of functions
    (N,λ,p) -> prox_distance(M,λ/m,q,p,1)
    for q in Q
]

p1 = cyclic_proximal_point(M, f, p_f, p0)
```

Or a derivative-free algorithm: LTMADS

```
p2 = mesh_adaptive_direct_search(M, f, p0)
```

NTNU

Norges teknisk-naturvitenskapelige universitet

# Robust Levenberg-Marquardt

# Nonlinear Least Squares

[Baran, RB, 2026]

**Goal.** Minimize a set $F_i \colon \mathcal{M} \to \mathbb{R}^{n_i}$ of residuals

$$f(p) = \frac{1}{2} \sum_{i=1}^{m} \rho_i \left( \|F_i(p)\|_2^2 \right), \qquad p \in \mathcal{M},$$

where $\rho_i \colon \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$, $\rho_i(0) = 0$, are robustifiers.

- nonlinear least squares $\rho(t) = t$ [Adachi, Okuno, Takeda, 2022]
- robust regression $\rho(t) = \sqrt{t}$
- here: relaxed, $\rho \in C^1$, e.g. Huber $\rho_{\mathrm{Hu}}(t) := \begin{cases} t & \text{if } t \leq 1 \\ 2\sqrt{t} - 1 & \text{if } t > 1. \end{cases}$

**Component wise case.**
For $F = (f_1, \ldots, f_m) \colon \mathcal{M} \to \mathbb{R}^m$, $f_i \colon \mathcal{M} \to \mathbb{R}$ (all $n_i = 1$):
each component is robustified $\Rightarrow$ aims to minimize $\|F(p)\|_1$

# The Jacobian and its adjoint

The Jacobian of $F\colon \mathcal{M} \to \mathbb{R}^n$, $F = (f_1, \dots, f_n)$ is

$$\mathcal{J}_F(p)[X] = \mathrm{D}F(p)[X] = \Big(\langle \operatorname{grad} f_j(p), X\rangle_p\Big)_{j=1}^{n}, \qquad p \in \mathcal{M}, X \in T_p\mathcal{M}.$$

With a basis $Y_1, \dots, Y_d \in T_p\mathcal{M}$: $\mathcal{J}_F(p) = J_F(p) \in \mathbb{R}^{n\times d}$ as a matrix.
But this requires to decompose $X = \sum_i c_i Y_i$ to perform $J_F(p)c$.

The (Riemannian) adjoint Jacobian $\mathcal{J}_F^*\colon \mathbb{R}^n \to T\mathcal{M}$ is given by

$$\mathcal{J}_F^*(p)[x] = \sum_{j=1}^{n} x_j \operatorname{grad} f_j(p), \qquad p \in \mathcal{M}, x \in \mathbb{R}^n.$$

Again: With an orthonormal basis of $T_p\mathcal{M}$: a matrix $J_F^*(p) = J_F(p)^{\mathsf{T}}$.

# Second order Taylor expansion

For a "block" $G(p) := \frac{1}{2}\rho\big(\|F(p)\|_2^2\big)$ the second order Taylor expansion

$$G\big(R_p(X)\big) = G(p) + \langle \operatorname{grad} G(p), X\rangle_p + \frac{1}{2}\langle \operatorname{Hess} G(p)[X], X\rangle_p + o(\|X\|_p^3)$$

reads

$$\begin{aligned} &G(p) + \rho'\left(\|F(p)\|_2^2\right)\langle \mathcal{J}_F^*(p)\big[F(p)\big], X\rangle_p \\ &\quad + \rho'\left(\|F(p)\|_2^2\right)\Big(\langle \mathcal{J}_F^*(p)\big[\mathcal{J}_F(p)[X]\big], X\rangle_p + \frac{1}{2}\sum_{j=1}^{n} f_j(p)\langle \operatorname{Hess} f_j(p)[X], X\rangle_p\Big) \\ &\quad + \rho''\left(\|F(p)\|_2^2\right)\langle F(p), \mathcal{J}_F(p)[X]\rangle\langle \mathcal{J}_F^*(p)\big[F(p)\big], X\rangle_p + o(\|X\|_p^3) \end{aligned}$$

- drop the cubic error term
- drop the Hessian terms of $F$

$\Rightarrow$ We only require $\rho'\left(\|F(p)\|_2^2\right)$, $\rho''\left(\|F(p)\|_2^2\right)$, $\mathcal{J}_F$, and $\mathcal{J}_F^*$

# Main step of Levenberg-Marquardt

At every iteration $p = p^{(k)}$ find minimzer $X^*$ of the damped surrogate

$$s_\lambda(X) := \frac{1}{2}\|\mathcal{L}(X) + y\|_2^2 + \lambda\|X\|_p^2, \qquad X \in T_p\mathcal{M}, \lambda \geq 0$$

- with linear map $\mathcal{L}\colon T_p\mathcal{M} \to \mathbb{R}^n$ and $y \in \mathbb{R}^n$.
- chosen s. t. $s_0$ approximates $G$, i.e. with $r = F(p)$, $t = \|r\|^2$ that
  1. $\operatorname{grad} s_0(0_p) = \operatorname{grad} G(p) = \rho'(t)\mathcal{J}_F(p)^*[r]$
  2. $\operatorname{Hess} s_0(0_p)[X] = \rho'(t)\mathcal{J}_F^*(p)[\mathcal{J}_F(p)[X]] + 2\rho''(t)\langle r, \mathcal{J}_F(p)[X]\rangle\mathcal{J}(p)^*[r]$.
- update $\lambda$ based on "model fit" $m = \dfrac{G(p) - G\big(R_p(X^*)\big)}{\frac{1}{2}\big(s_\lambda(0_p) - s_\lambda(X^*)\big)}$

Overall we have to solve a sub problem in every iteration.

# Manopt.jl Interlude: Sub solvers

To solve the surrogate sub problem, the `LevenbergMarquardState` stores

- `sub_problem` for the objective and here the tangent space as manifold
- `sub_state` specify
  - the solver from `Manopt.jl` to use
  - or that function implements the sub solver (in closed form)
  - default here `ConjugateResidualState` [Lai, Yoshise, 2024]
- `set_parameter!(element, :Name, value)` to update e. g. the base point $p$ or the dampening $\lambda$

$\Rightarrow$ modular design, in coefficients `c` one can "plug in" any Euclidean solver for linear systems

# Example I: Robust Geodesic Regression

[Fletcher, 2013; Loayza-Romero, Sibum, Welker, 2026; Baran, RB, 2026]

**Given** $(t_i, q_i) \in \mathbb{R} \times \mathcal{M}$, $i = 1 \dots, m$

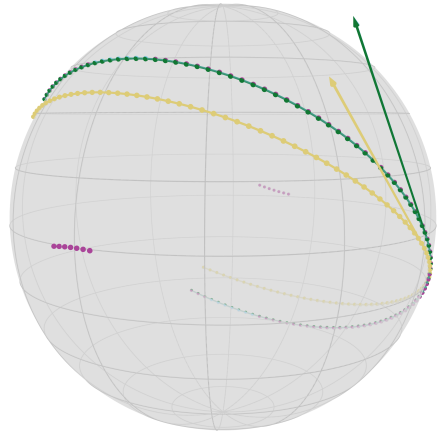
**Find** $\gamma_{p,X}(t) = \exp_p(tX)$, $(p, X) \in T\mathcal{M}$ such that $\gamma_{p,X}(t_i) \approx q_i$

**Objective**

$$f(p, X) = \sum_{i=1}^{m} \rho\big(d^2(\gamma_{p,X}(t_i), q_i)\big)$$

**Example.** $\mathcal{M} = \mathbb{S}^2$

Geodesic with data measurement outliers.



**Remaining Error.**

| | |
|---|---|
| **Least Squares** $\rho(t) = t$ | $1.3904 \times 10^{-2}$ |
| **Robust** $\rho_{\mathrm{Hu},10^{-4}}$ | $2.2737 \times 10^{-6}$ |

# Example II: The Robust Procrustes Problem

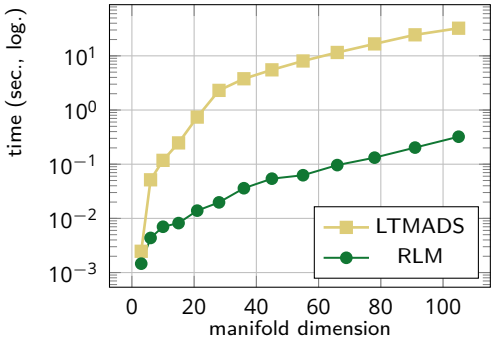
[Jasa, RB, Kümmerle, Athreya, Lubberts, 2025]

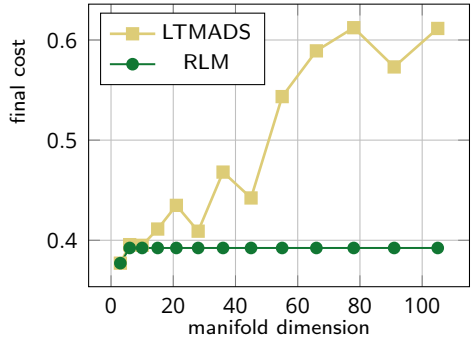
**Given** $A, B \in \mathbb{R}^{d\times n}$, **find** "best" alignment $A \approx pB$, $p \in \mathrm{SO}(d)$.

**Procrustes.** $f(p) = \|A - pB\|_{\mathrm{F}}$ has a closed form minimizer using an SVD.

**Robust.** Use $\|C\|_{\mathrm{R}} := \sum_{i=1}^{n} \|c_i\|_2$, $c_i$, $i = 1, \ldots, n$ the columns of $C \in \mathbb{R}^{d\times n}$

**Example.** Data with $d = 3, \ldots, 15$, $n = \frac{d(d+1)}{2}$.



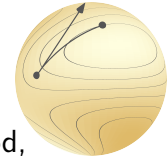



NTNU

Norges teknisk-naturvitenskapelige universitet

# Summary

# List of Algorithms in Manopt.jl



**Derivatve-Free** Nelder-Mead, Particle Swarm, CMA-ES, MADS

**Subgradient-based** Subgradient Method, Convex Bundle Method, Proximal Bundle Method

**Gradient-based** Gradient Descent, Conjugate Gradient, Stochastic, Momentum, Nesterov, Averaged, Gradient Sampling Quasi-Newton with (L-)BFGS(B), DFP, Broyden, SR1, ...

**Hessian-based** Trust Regions, Adaptive Regularized Cubics (ARC)

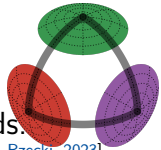
**splitting** Chambolle-Pock, Douglas-Rachford, Cyclic Proximal Point, Proximal Gradient, (robust) Levenberg-Marquardt

**constrained** Augmented Lagrangian, Exact Penalty, Frank-Wolfe, Projected Gradient, Interior Point Newton

**nonconvex** Difference of Convex Algorithm, DCPPA

manoptjl.org/stable/solvers/

# Manifolds.jl



**Goal.**
Efficiently implemented and well-documented Riemannian manifolds.
[Axen, Baran, RB, Rzecki, 2023]

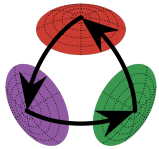
**Meta.** generic implementations for $\mathcal{M}^{n\times m}$, $\mathcal{M}_1 \times \mathcal{M}_2$,
vector- and tangent-bundles, esp. $T_p\mathcal{M}$

**LieGroups.jl** uses the manifolds provided here as a basis

**Library** of implemented manifolds

- Circle, Sphere, Torus, Hyperbolic, Projective Spaces, Hamiltonian
- (generalized, symplectic) Stiefel, Rotations
- (generalized, symplectic) Grassmann, fixed rank matrices
- Symmetric Positive Definite matrices, with fixed determinant
- (several) Multinomial, (skew-)symmetric, and symplectic matrices
- Tucker, Segre & Oblique manifold, Kendall's Shape space
- probability simplex, orthogonal and unitary matrices, ...

# LieGroups.jl



[RB, Baran, 2026]

**Goal.**
Provide an interface to define and use Lie groups and provide efficiently implemented and well-documented Lie groups $\mathcal{G}$.

**Meta.**
Generic implementations for the Lie algebra $\mathfrak{g}$, group operations, group actions, and an identity $e \in \mathfrak{g}$ that only allocates when necessary

$\mathcal{G}^n$, $\mathcal{G} \times \mathcal{H}$, $\mathcal{G} \ltimes \mathcal{H}$, $\mathcal{G} \rtimes \mathcal{H}$

**Library** of implemented Lie groups

- Circle, General linear, Heisenberg, Orthogonal
- Special Euclidean, Special linear, Special Orthogonal, Special Unitary
- Translation, Unitary

# Conclusion

`Manopt.jl`

- General setup of a solver and an example
- works on all Manifolds implemented with `ManifoldsBase.jl`
- ease of use, example runs & tools

Robust Riemannian Levenberg-Marquardt

- solve nonlinear least squares – robustly!
- technical differences of the Jacobian
- two examples

...and a short overview of details in `Manopt.jl`, `Manifolds.jl` & `LieGroups.jl`

# Selected References


Adachi, S.; T. Okuno; A. Takeda (2022). *Riemannian Levenberg-Marquardt Method with Global and Local Convergence Properties*. arXiv: 2210.00253.

Axen, S. D.; M. Baran; RB; K. Rzecki (2023). "Manifolds.jl: An Extensible Julia Framework for Data Analysis on Manifolds". *ACM Transactions on Mathematical Software* 49.4. DOI: 10.1145/3618296.

Baran, M.; RB (2026). *A modified Riemannian Levenberg-Marquardt algorithm for robust and constraint optimization on manifolds*. arXiv: 2606.23560.

RB (2022). "Manopt.jl: Optimization on Manifolds in Julia". *Journal of Open Source Software* 7.70, p. 3866. DOI: 10.21105/joss.03866.

Boumal, N.; B. Mishra; P.-A. Absil; R. Sepulchre (2014). "Manopt, a Matlab toolbox for optimization on manifolds". *Journal of Machine Learning Research* 15, pp. 1455–1459. URL: https://www.jmlr.org/papers/v15/boumal14a.html.

Jasa, H.; RB; C. Kümmerle; A. Athreya; Z. Lubberts (2025). *Procrustes Problems on Random Matrices*. arXiv: 2510.05182.

Townsend, J.; N. Koep; S. Weichwald (2016). *Pymanopt: A Python toolbox for optimization on manifolds using automatic differentiation*. arXiv: 1603.03236.


ronnybergmann.net/talks/2026-Manoptjl-Oberwolfach.pdf



Interested in Numerical Differential Geometry? Join numdiffgeo.zulipchat.com!