




# Manopt.jl: Optimization on Manifolds in Julia

Ronny Bergmann<sup>1</sup>

<sup>1</sup> Norwegian University of Science and Technology, Department of Mathematical Sciences,  
Trondheim, Norway

DOI: [10.21105/joss.03530](https://doi.org/10.21105/joss.03530)

## Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

Editor: [Pending Editor](#) 

Submitted: 22 July 2021

Published: 23 July 2021

## License

Authors of papers retain  
copyright and release the work  
under a Creative Commons  
Attribution 4.0 International  
License ([CC BY 4.0](#)).

## Summary

[Manopt.jl](#) provides a set of optimization algorithms for optimization problems given on a Riemannian manifold  $\mathcal{M}$ . Based on a generic optimization framework together with the interface `ManifoldsBase.jl` for Riemannian manifolds, classical and recently developed methods are provided in an efficient implementation. Algorithms include the least requiring Particle Swarm and Nelder Mead algorithms as well as a classical gradient or stochastic gradient descent. Furthermore, quasi Newton methods like a Riemannian L-BFGS ([Huang et al., 2015](#)) and nonsmooth optimization algorithms like a Cyclic Proximal Point Algorithm ([Bačák, 2014](#)), Douglas-Rachford ([Bergmann et al., 2016](#)) and Chambolle-Pock algorithm ([Bergmann et al., 2021](#)) are provided together with several basic cost functions, gradients and proximal maps as well as debug and record capabilities.

## Statement of Need

In many applications and optimization tasks, nonlinear data appears naturally. For example when data on the sphere is measured, diffusion data can be captured as a signal or multivariate data of symmetric positive definite matrices or orientations like they appear for electron backscattered diffraction (EBSD) data. Another example are fixed rank matrices, appearing in dictionary learning. Working on these data, for example doing data interpolation, data approximation, denoising, inpainting, or performing matrix completion, can be phrased as an optimization problem

$$\text{Minimize } f(x) \quad \text{where } x \in \mathcal{M},$$

where the optimization problem is phrased on a Riemannian manifold  $\mathcal{M}$ .

A main challenge of these algorithms is that, compared to the (classical) Euclidean case, there is no addition available. For example on the unit sphere  $\mathbb{S}^2$  of unit vectors in  $\mathbb{R}^3$ , adding two vectors of unit lengths yields a point that is not of unit norm. The resolution is to generalize the notion of a shortest path from the straight line to what is called a (shortest) geodesic, or acceleration free curves. In the same sense, other features and properties have to be rephrased and generalized, when performing optimization on a Riemannian manifold. Algorithms to perform the optimisation can still often be stated in the generic form, i.e. on an arbitrary Riemannian manifold  $\mathcal{M}$ .

Further examples and a thorough introduction can be found in ([Absil et al., 2008](#)), ([Boumal, 2020](#)).

For a user facing an optimization problem on a manifold, there are two obstacles to the actual numerical optimisation: on the one hand, a suitable implementation of the manifold at hand is

required, for example how to evaluate the above mentioned geodesics. On the other hand, an implementation of the optimisation algorithm that employs said methods from the manifold, such that the algorithm can be applied to the cost function  $f$  a user already has.

Using the interface for manifolds, `ManifoldsBase.jl`, the algorithms are implemented in the optimization framework can therefore be used with any manifold from `Manifolds.jl` (Axen et al., 2021), a library of efficiently implemented Riemannian manifolds. `Manopt.jl` provides a low level entry to optimization on manifolds while also providing efficient implementations, that can easily be extended to cover own manifolds.

## Functionality

`Manopt.jl` provides a comprehensive framework for optimization on Riemannian manifolds, including a generic way to specify a step size and a stopping criterion as well as enhance the algorithm with debug and recording capabilities. Based on this interface a variety

An optimization problem in `Manopt.jl` consists of a `Problem p` and `Options o`. The `Problem` consists of all static information like the cost function and a potential gradient of the optimization task. The `Options` specify the type of algorithm and the details. For example by default most options specify that the exponential map, which generalizes the notion of addition to the manifold should be used and the algorithm steps are performed following an acceleration free curve on the manifold. This might not be known in closed form for some manifolds and hence also arbitrary retractions can be specified for this instead. This yields approximate algorithms that are numerically more efficient. Similarly, tangent vectors at different points are identified by vector transport, which by default is the parallel transport. By providing always a default, the start for a user can start right away and modify these settings to improve speed or specify the retraction to their needs.

The main methods to implement for an own solver are the `initialize_solver!(p,o)` which should fill the data in the options with initial state. The second method to implement is the `step_solver!(p,o,i)` performing the  $i$ th iteration.

Using a decorator pattern, the `Options` can be encapsulated in `DebugOptions` or `RecordOptions` which either print or record arbitrary data stored within the `Options`. This enables to investigate how the optimization is performed in detail and use the algorithms from within this package also for numerical analysis.

In the current version `Manopt.jl` version 0.3.11 the following algorithms are available

- Alternating Gradient Descent
- Chambolle-Pock (Bergmann et al., 2021)
- Conjugate Gradient Descent, including eight update rules
- Cyclic Proximal Point (Bačák, 2014)
- (parallel) Douglas–Rachford (Bergmann et al., 2016)
- Gradient Descent, including direction update rules including Momentum, Average, and a Nestorv-type one
- Nelder-Mead
- Particle Swarm Optimization (Borckmans et al., 2010)
- Quasi-Newton, with the BFGS, DFP, Broyden and a symmetric rank 1 update, their inverse updates as well as a limited memory variant of (inverse) BFGS (Huang et al., 2015)
- Stochastic Gradient Descent
- Subgradient Method
- Trust Regions, with inner Steihaug-Toint TCG solver (Absil et al., 2006)

## Example

Manopt.jl is registered in the general Julia registry and can hence be installed typing `]add Manopt` in Julia REPL. Given the Sphere from Manifolds.jl and a set of unit vectors  $p_1, \dots, p_N \in \mathbb{R}^3$ , where  $N$  is the number of data points. we can compute the generalization of the mean, called the Riemannian Center of Mass, which is defined as the minimizer of the squared distances to the given data

$$\text{Minimize}_{x \in \mathcal{M}} \sum_{k=1}^N d_{\mathcal{M}}(x, p_k)^2,$$

where  $d_{\mathcal{M}}$  denotes the Riemannian distance. For the sphere this distance is given by the length of the shorter great arc connecting the two points.

```
using Manopt, Manifolds, LinearAlgebra
M = Sphere(2)
n = 100
pts = [ normalize(rand(3)) for _ in 1:n ]

F(M, y) = sum(1/(2*n) * distance.(Ref(M), pts, Ref(y)).^2)
gradF(M, y) = sum(1/n * grad_distance.(Ref(M), pts, Ref(y)))

xMean = gradient_descent(M, F, gradF, pts[1])
```

In order to print the iteration, the current iterate, change and cost every 50th iteration as well as the stopping reason and record iteration number, change and cost, these can be specified as optional parameters. These can then be easily accessed using the `get_record` function.

```
o = gradient_descent(M, F, gradF, pts[1],
    debug=[:Iteration, " | ", :x, " | ", :Change, " | ", :Cost, "\n", :Stop],
    record=[:Iteration, :Change, :Cost],
    return_options=true
)
xMean3 = get_solver_result(o)
values = get_record(o)
```

## Related research and software

There are two projects that are most similar to Manopt.jl are [Manopt](#) (Boumal et al., 2014) in Matlab and [pymanopt](#) (Townsend et al., 2016) in Python. Similarly [ROPTLIB](#) (Huang et al., 2018) is a package for optimization on Manifolds in C++. While all three packages cover some algorithms, most are less flexible for example in stating the stopping criterion, which is fixed to mainly maximal number of iterations or a small gradient. Most prominently, Manopt.jl is the first package that also covers methods for high-performance and high-dimensional nonsmooth optimization on manifolds.

The algorithm presented in (Bergmann et al., 2021) was developed using Manopt.jl. Based on this theory and algorithm, a higher order algorithm was introduced in (Diepeveen & Lellmann, 2021). Optimized examples from (Bergmann & Gousenbourger, 2018) performing data interpolation and approximation with manifold-valued Bézier curves, are also included in Manopt.jl.

## References

- Absil, P.-A., Baker, C. G., & Gallivan, K. A. (2006). Trust-region methods on riemannian manifolds. *Foundations of Computational Mathematics*, 7(3), 303–330. <https://doi.org/10.1007/s10208-005-0179-9>
- Absil, P.-A., Mahony, R., & Sepulchre, R. (2008). *Optimization algorithms on matrix manifolds*. Princeton University Press. <https://doi.org/10.1515/9781400830244>
- Axen, S. D., Baran, M., Bergmann, R., & Rzecki, K. (2021). *Manifolds.jl: An extensible Julia framework for data analysis on manifolds*. <http://arxiv.org/abs/2106.08777>
- Bačák, M. (2014). Computing medians and means in hadamard spaces. *SIAM Journal on Optimization*, 24(3), 1542–1566. <https://doi.org/10.1137/140953393>
- Bačák, M. (2014). Computing medians and means in hadamard spaces. *SIAM Journal on Optimization*, 24(3), 1542–1566. <https://doi.org/10.1137/140953393>
- Bergmann, R., & Gousenbourger, P.-Y. (2018). A variational model for data fitting on manifolds by minimizing the acceleration of a bézier curve. *Frontiers in Applied Mathematics and Statistics*, 4. <https://doi.org/10.3389/fams.2018.00059>
- Bergmann, R., Herzog, R., Silva Louzeiro, M., Tenbrinck, D., & Vidal-Núñez, J. (2021). Fenchel duality theory and a primal-dual algorithm on riemannian manifolds. *Foundations of Computational Mathematics*. <https://doi.org/10.1007/s10208-020-09486-5>
- Bergmann, R., Persch, J., & Steidl, G. (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), 901–937. <https://doi.org/10.1137/15M1052858>
- Borckmans, P. B., Ishteva, M., & Absil, P.-A. (2010). A modified particle swarm optimization algorithm for the best low multilinear rank approximation of higher-order tensors. In *Lecture notes in computer science* (pp. 13–23). Springer Berlin Heidelberg. [https://doi.org/10.1007/978-3-642-15461-4\\_2](https://doi.org/10.1007/978-3-642-15461-4_2)
- Boumal, N. (2020, August). *An introduction to optimization on smooth manifolds*. <http://www.nicolasboumal.net/book>
- Boumal, N., Mishra, B., Absil, P.-A., & Sepulchre, R. (2014). Manopt, a Matlab toolbox for optimization on manifolds. *Journal of Machine Learning Research*, 15(42), 1455–1459. <https://www.manopt.org>
- Diepeveen, W., & Lellmann, J. (2021). *Duality-based higher-order non-smooth optimization on manifolds*. <http://arxiv.org/abs/2102.10309>
- Huang, W., Absil, P.-A., Gallivan, K. A., & Hand, P. (2018). ROPTLIB: An object-oriented c++ library for optimization on riemannian manifolds. *Association for Computing Machinery. Transactions on Mathematical Software*, 44(4), Art. 43, 21. <https://doi.org/10.1145/3218822>
- Huang, W., Gallivan, K. A., & Absil, P.-A. (2015). A broyden class of quasi-newton methods for riemannian optimization. *SIAM Journal on Optimization*, 25(3), 1660–1685. <https://doi.org/10.1137/140955483>
- Huang, W., Gallivan, K. A., & Absil, P.-A. (2015). A broyden class of quasi-newton methods for riemannian optimization. *SIAM Journal on Optimization*, 25(3), 1660–1685. <https://doi.org/10.1137/140955483>
- Townsend, T., Koep, N., & Weichwald, S. (2016). Pymanopt: A python toolbox for optimization on manifolds using automatic differentiation. *Journal of Machine Learning Research*, 17(137), 1–5. <http://jmlr.org/papers/v17/16-177.html>