Optimization on Manifolds in Julia
`Manopt.jl` and `Manifolds.jl`

Ronny Bergmann[a], `ronnybergmann_`

[a]Technische Universität Chemnitz, Chemnitz, Germany

Mini-Workshop: Computational Optimization on Manifolds,
Mathematisches Forschungsinstitut Oberwolfach,
Twitter
Oberwolfach, March, 2020.

Contents

1. Introduction

Introduction

There are currently 3 packages available to do optimization on manifolds

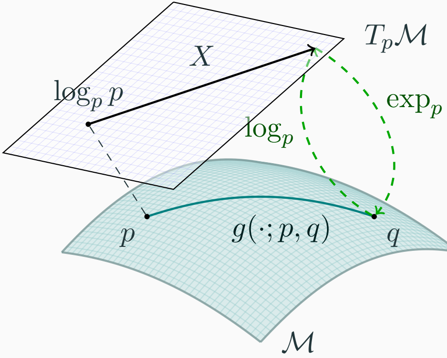
- `Manopt` – in Matlab, since 2013 [N. Boumal]
- `pymanopt` - in Python, since 2015 [J. Townsend, N. Koep, S. Weichwald]
- `MVIRT` - Matlab, since 2015 [RB]

Goal Today

A nicely typed, flexible optimization toolbox in Julia

`https://julialang.org`

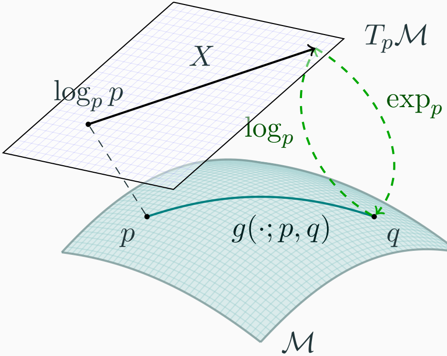
A d-dimensional Riemannian Manifold $\mathcal{M}$



A d-dimensional Riemannian manifold can be informally defined as a set $\mathcal{M}$ covered with a 'suitable' collection of charts, that identify subsets of $\mathcal{M}$ with open subsets of $\mathbb{R}^d$ and a continuously varying inner product on the tangential spaces.

[Absil, Mahony, Sepulchre, 2008]

A d-dimensional Riemannian Manifold $\mathcal{M}$



Geodesic $g(\cdot; p, q)$ shortest path (on $\mathcal{M}$) between $p, q \in \mathcal{M}$

Tangent space $\mathrm{T}_p\mathcal{M}$ at p, with inner product $(\cdot, \cdot)_p$

Logarithmic map $\log_p q = \dot{g}(0; p, q)$ "speed towards q"

Exponential map $\exp_p X = g(1)$, where $g(0) = p$, $\dot{g}(0) = X$

Optimization on Manifolds

Let $\mathcal{M}$ and $\mathcal{N}$ be Riemannian Manifolds and $\mathcal{E}\colon \mathcal{N} \to \mathbb{R}$.

We want to consider the optimization problem

$$\operatorname*{arg\,min}_{p \in \mathcal{N}} \mathcal{E}(p)$$

where $\mathcal{E}$ is

- (maybe) non-smooth,
- (locally) convex,
- high-dimensional, e. g.
  - a manifold valued signal, $\mathcal{N} = \mathcal{M}^d$
  - a manifold-valued image, $\mathcal{N} = \mathcal{M}^{d_1 \times d_2}$
- can be decomposed $\mathcal{E} = \sum_{i=1}^{K} f_i$ in two (or even more) summands

Installation & Documentation

The Julia package `Manopt.jl` can be installed using

```
] add Manopt
```

where `]` switches to `Pkg` mode.

The documentation is available at

`https://www.manoptjl.org/stable/`.

2. A Nonsmooth Optimization Task

An example problem: The Riemannian median

The mean of $x_1,\ldots,n_N \in \mathbb{R}$, i.e. $\displaystyle\frac{1}{N}\sum_{i=1}^{N} x_i$

can be written as the unique minimizer of the optimization problem

$$\operatorname*{arg\,min}_{x\in\mathbb{R}} \sum_{i=1}^{N} \lVert x - x_i\rVert^2.$$

Similarly the median can be optained by the non-smooth optimization problem

$$\operatorname*{arg\,min}_{x\in\mathbb{R}} \sum_{i=1}^{N} \lVert x - x_i\rVert.$$

➔ A nonsmooth optimization problem on an Euclidean space

An example problem: The Riemannian median

The mean of $x_1, \ldots, n_N \in \mathcal{M}$

is defined as the unique minimizer of the optimization problem

$$\operatorname*{arg\,min}_{x \in \mathcal{M}} \sum_{i=1}^{N} d_{\mathcal{M}}(x, x_i)^2.$$

Similarly the median can be optained by the non-smooth optimization problem

$$\operatorname*{arg\,min}_{x \in \mathcal{M}} \sum_{i=1}^{N} d_{\mathcal{M}}(x, x_i).$$

➔ A nonsmooth optimization problem on a Riemannian manifold

Proximal Map

For $\varphi\colon \mathcal{M} \to (-\infty, +\infty]$, $\lambda > 0$ we define the Proximal Map

[Moreau, 1965; Rockafellar, 1976; Ferreira, Oliveira, 2002]

$$\operatorname{prox}_{\lambda\varphi}(p) \coloneqq \operatorname*{arg\,min}_{u\in\mathcal{M}^n} \frac{1}{2}\sum_{i=1}^{n} d(u_i, p_i)^2 + \lambda\varphi(u).$$

! For a Minimizer u^* of φ we have $\operatorname{prox}_{\lambda\varphi}(u^*) = u^*$.

- For $\varphi\colon \mathbb{R}^n \to \mathbb{R}$ proper, convex, lower semicontinuous:
    - the $\operatorname{proximal}$ map is unique.
    - PPA $x_k = \operatorname{prox}_{\lambda\varphi}(x_{k-1})$ converges to $\operatorname{arg\,min}\varphi$
- Without splitting, i.e. with $\varphi = \mathcal{E}$, not that useful

The Cyclic Proximal Point Algorithm

If we split $\mathcal{E} = \sum_{l=1}^{c} \varphi_l$, we can apply the Cyclic Proximal Point-Algorithmus (CPPA): [Bertsekas, 2011; Bačák, 2014]

$$p^{(k+\frac{l+1}{c})} = \operatorname{prox}_{\lambda_k \varphi_l}(p^{(k+\frac{l}{c})}), \quad l = 0,\ldots,c-1,\ k = 0,1,\ldots$$

On a Hadamard manifold $\mathcal{M}$:
convergence to a minimizer of φ if

- all φ_l proper, convex, lower semicontinuous
- $\{\lambda_k\}_{k\in\mathbb{N}} \in \ell_2(\mathbb{N})\setminus\ell_1(\mathbb{N})$.

3. An Example for `Manopt.jl`

The solver

A `Solver` in `Manopt.jl` works on two data structures

- a `Problem p` contains $\mathcal{M}$, $\mathcal{E}$, and further static data
- `Options o` contain the current state

➔ problem and options together determine a solver

☺ Starting a solver: just call `solve!(p,o)`

For our example:

- `ProximalProblem` storing $\mathcal{M}$, $\mathcal{E}$, and the $\operatorname{prox}_{\lambda\varphi_i}$
- `CyclicProximalPointOptions` storing λ, the current iterate and a `StoppingCriterion`

☺ easier: use `cyclicProximalPoint(M, cost, proxes, x0)`

Preparing the Input

```julia
using Manopt, Random
Random.seed!(42)
n=100
# our manifold - the hyperbolic space
M = Hyperbolic(2)
# a point
p = HnPoint([0., 0., 1.]) #for Manopt 0.1.0
# noisy data around p
data = [ addNoise(M,p) for i=1:100]
# Our cost functions: distances to y
cost = y -> sum( 1/(2*n) * distance.(Ref(M),Ref(y),data))
# The proximal maps - an array of anonymous functions
proxes = Function[
    (λ, y) -> proxDistance(M, λ/n, di, y, 1) for di in data
]
```

Stopping criteria

A `StoppingCriterion` is a functor: a `struct` that is also a function `(p,o,i) -> Boolean`, for example

- `stopAfterIteration(n)`, `stopAfter(time)`
- `stopWhenChangeLess(eps)`
- or more specific `stopWhenTrustRegionIsExceeded`
- and for multiple criteria: `stopWhenAny`, `stopWhenAll`

Example. Stop after 100 iterations

```
m = cyclicProximalPoint(M, cost, proxes, data[1];
    stoppingCriterion=stopAfterIteration(1000)
)
```

Debug & Record

In order to get more details: Decorators for options:

Both `DebugOptions(o,A)`, `RecordOptions(o,A)` act as if they where just the `Options o`.

They certain `DebugActions`, `RecordActions` that are evaluated every iteration to print or store data.

Examples.

- `RecordEntry(Float64,:x)` to record current iterate
- `DebugEntry(:λ)` to print current value of λ.
- `DebugCost()` evaluate $\mathcal{E}$ and print its value

Example with Debug and Record

```
o = cyclicProximalPoint(M, cost, proxes, data[1];
  debug = [
    :Iteration," | ", :x," | ", :Change," | ", :Cost,"\n",
    50, :Stop
  ],
  record = [:Iteration, :Change, :Cost],
  returnOptions = true # return options to access record
)
```

- use keywords (`:Iteration` or fields (`:x`) of `Options o`
- `debug=` can be interleaved with strings
- a number, `50`, reduced output to every 50th iteration
- `:Stop` prints the reason the solver stopped

Interface: Implement your own solver

Given a `Problem p` and some `Options o`,

a solver consists of the functions (all modifying the `Options o`)

- `initializeSolver!(p,o)` that initializes the Options
- `doSolverStep(p,o,i)` to perform the `i`th iteration
- `stopSolver!(p,o,i)` that uses a `StoppingCriterion` to determine whether to stop after the `i`th iteration

😊 You have to implement the first two.
and eventually additional stopping criteria

Available Solvers

- Cyclic Proximal Point
- Douglas-Rachford
- Gradient Descent
- Nelder Mead
- Subgradient Method
- Riemannian Trust Regions

😊 high level interface, a default stopping criterion, debug, record

➔ Just get started and try them!

4. `Manifolds.jl` & `ManifoldsBase.jl`

Manifolds in Julia

`ManifoldsBase.jl`

- lightweight interface for/to work on manifolds
- unified base for further projects

`Manifolds.jl`

- specific manifolds based on the interface.
- provide properties in a transparent way (decorators)
- rich documentation of formulae and sources

Seth Axen
UCSF, San Francisco, USA.

Mateusz Baran
AGH UST, Kraków, Poland.

Ronny Bergmann
TU Chemnitz, Germany.

A Riemannian manifold

A manifold in general is a **`type`** that inherits from `Manifold`. `ManifoldsBase` provides the interface of functions like

- `exp(M, p, X)`, `log(M, p, q)`
- `retract(M, p, X, m)`, where `m` is a retraction method
- `vector_transport_to(M, p, X, q, t)` where `t` is a transport method

😊 mutating version `exp!(M,q,p,X)` works in place in `q`
➜ generic algorithms for any `Manifold`
⚠ suitable error messages if a function is not implemented

Default implementations for `norm(M,p,X)`, `geodesic(M,p,X)` and `shortest_geodesic(M,p,q)`

A manifold decorator

Properties are often implicitly given, like the metric.

The interface provides a decorator manifold that acts semi-transparently, i.e. transparent for all functions not affected by an explicit different implementation.

Example.

`ArrayManifold(M)` performs (when applicable)

- `is_manifold_point(M,p)`
- `is_tangent_vector(M,p,X)`

before and after every basic function from the interface.

`MetricManifold{Manifold,Metric}`

Goal. Implement different metrics for a manifold.

⇒ transparent e.g. for `manifold_dimension(M)`

- existing implementation: default metric (transparent)
- other functions: implementation using parametric type

Example.

- `M = SymmetricPositiveDefinite(3)` has
- `MetricManifold(M,LinearAffineMetric)` as synonym
- `MetricManifold(M,LogEuclidean)` is a second metric
- `MetricManifold(M,LogCholesky)` is a metric providing an `exp`

😊 `exp` defaults to a method numerically solving the ODE.

EmbeddedManifolds

Goal. Model embedded manifold(s) of a manifold

😊 reuse functions (like `inner`) from embedding.

- different types via `AbstractEmbeddingType`
- provide `embed` and `project` & `get_embedding`

Examples.

- `Sphere{N} <: AbstractEmbeddedManifold`
  `{DefaultIsometricEmbeddingType}`
  into `Euclidean(N+1)`, ➔ its `inner` is used
- `SymmetricMatrices{N,𝔽} <: AbstractEmbeddedManifold`
  `{TransparentIsometricEmbedding}`
  into `Euclidean(N,N; field=𝔽)`, ➔ use its `exp` & `log`
- or use directly `EmbeddedManifold(Manifold, Embedding)`

GroupManifolds

Goal. Model Lie groups

- a manifold with a smooth binary operator $\circ$,
  e.g. translation, multiplication, composition
- an `identity` element
- together with `MetricManifold`:
  left-, right- & bi-invariant metric

Examples.

- `TranslationGroup(n)` is $\mathbb{R}^n$ with translation action
- `SpecialEuclidean(n)` is a `SemidirectProductGroup`
- `SpecialOrthogonal{n} <: GroupManifold{Rotations{n},MultiplicationOperation}`
- or directly `GroupManifold(Manifold, Operation)`

Currenlty Available Manifolds

- `Euclidean(n1,n2; field=ℝ)`
  short: `ℝ^(n1,n2)`, `ℂ^(n1,n2)`
- `Cholesky(n)`
- `FixedRank(n,m,k; field=ℝ)`
- `Grassmann(m,n; field=ℝ)`
  `GeneralizedGrassmann(m,n,B)`
- `Hyperbolic(n)`...embedded in:
- `Lorentzian(n)`
- `Stiefel(m,n; field=ℝ)`
  `GeneralizedStiefel(m,n,B)`
- `SymmetricMatrices(n; field=ℝ)`
- `SkewSymmetricMatrices(n)`
- `SymmetricPositiveDefinite(n)`
- `Circle(ℝ)`
- `Sphere(n)`
- `Rotations(n)`
- `Oblique(n,m)=Sphere(n)^m`
- `Torus(n)=Circle()^n`

Combine these with

- `ProductManifold(M1,M2, ... )`
  short: `M1`×`M2`
- `PowerManifold(M, n1, n2, ... )`
  short: `M^(n1,n2, ... )`
- `GraphManifold(M,G)`
- `Vector-` & `TangentBundle`

5. Summary & Outlook

Planned Further Features of `Manopt.jl`

- More solvers, e.g.
  - BFGS, including its limited memory implementation
  - Conjugate Gradient (nearly finished)
- Cache for function, gradient and Hessian evaluations (maybe based on `Memoize.jl`)
- Switch to `Manifolds.jl` (work in progess)

Get started:

- `manoptjl.org/stable`
- `juliamanifolds.github.io/Manifolds.jl/stable`
- GitHub Gist of the median example `gist.github.com/kellertuer/61d83d4855eb159081c24425725d08fd`

Selected References

Absil, P.-A.; Mahony, R.; Sepulchre, R. (2008). *Optimization Algorithms on Matrix Manifolds*. Princeton University Press. DOI: 10.1515/9781400830244.

Bačák, M. (2014). *Convex analysis and optimization in Hadamard spaces*. Vol. 22. De Gruyter Series in Nonlinear Analysis and Applications. De Gruyter, Berlin. DOI: 10.1515/9783110361629.

Bertsekas, D. P. (2011). "Incremental proximal methods for large scale convex optimization". *Mathematical Programming, Series B* 129.2, pp. 163–195. DOI: 10.1007/s10107-011-0472-0.

Ferreira, O. P.; Oliveira, P. R. (2002). "Proximal point algorithm on Riemannian manifolds". *Optimization. A Journal of Mathematical Programming and Operations Research* 51.2, pp. 257–270. DOI: 10.1080/02331930290019413.

Moreau, J.-J. (1965). "Proximité et dualité dans un espace hilbertien". *Bulletin de la Société Mathématique de France* 93, pp. 273–299.

Rockafellar, R. T. (1976). "Monotone operators and the proximal point algorithm". *SIAM Journal on Control and Optimization* 14.5, pp. 877–898. DOI: 10.1137/0314056.

ronnybergmann.net/talks/2020-Twitter-Manopt.pdf