The Manopt Toolbox, Optimization on manifolds in 3 minutes

www.manopt.org

Nicolas Boumal
Bamdev Mishra
Pierre-Antoine Absil
Yurii Nesterov
Rodolphe Sepulchre

A Matlab toolbox to solve optimization problems

$$\min_{X \in \mathcal{M}} f(X)$$

$\mathcal{M}$ is a Riemannian manifold

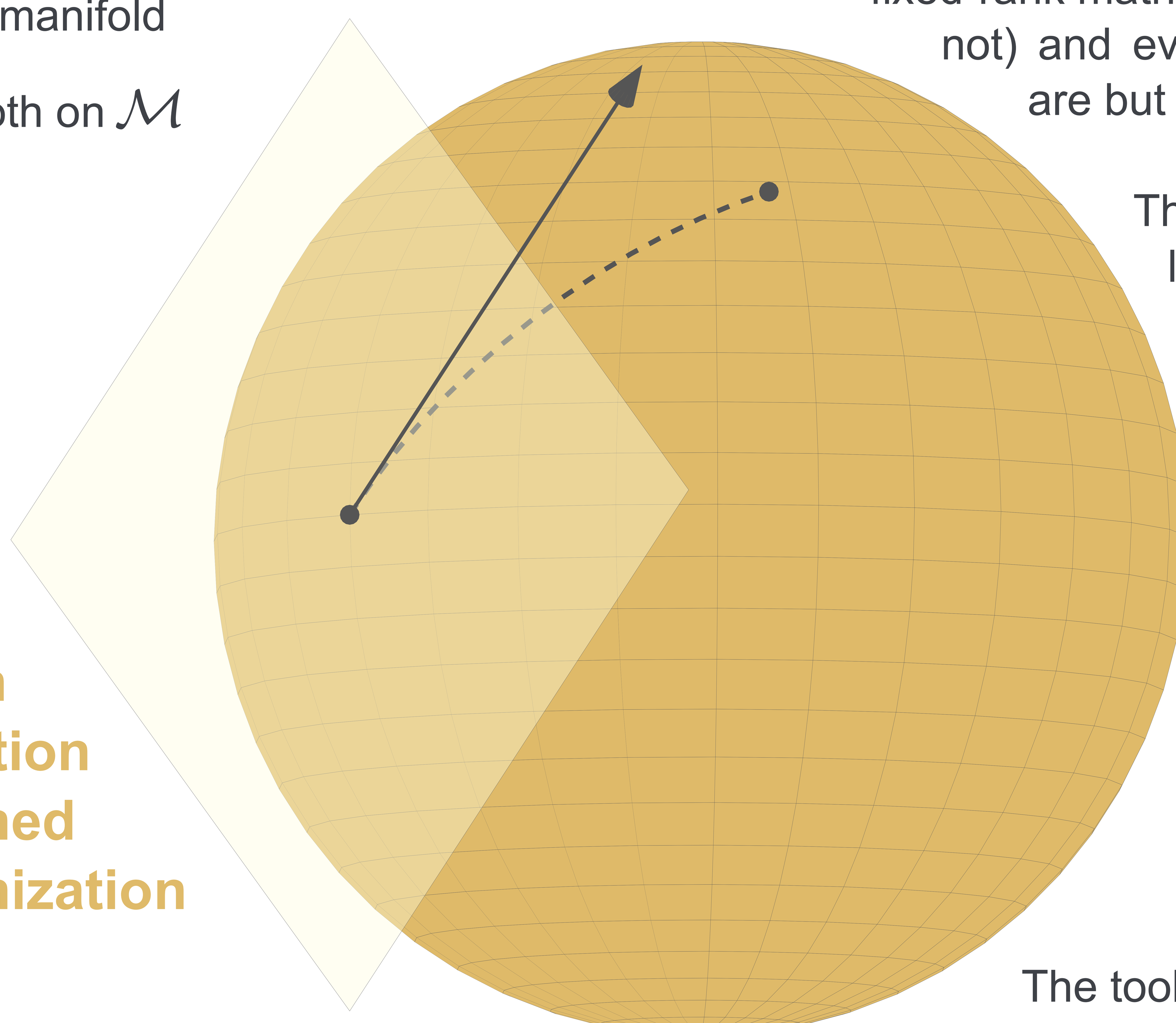
f is sufficiently smooth on $\mathcal{M}$

under a wide class of constraints

Many natural constraints in applications exhibit a Riemannian geometry.

Optimization of rotations, orthonormal matrices or fixed-rank matrices (symmetric or not, positive or not) and even optimization of linear subspaces are but a few examples.

The framework is well suited for large-scale computational engineering.



based on a generalization of unconstrained nonlinear optimization

Most well-known algorithms work just as well on important manifolds:

Steepest descent, conjugate gradients, BFGS, Newton, trust-regions, and many more.

And they come with essentially all the standard convergence guarantees.

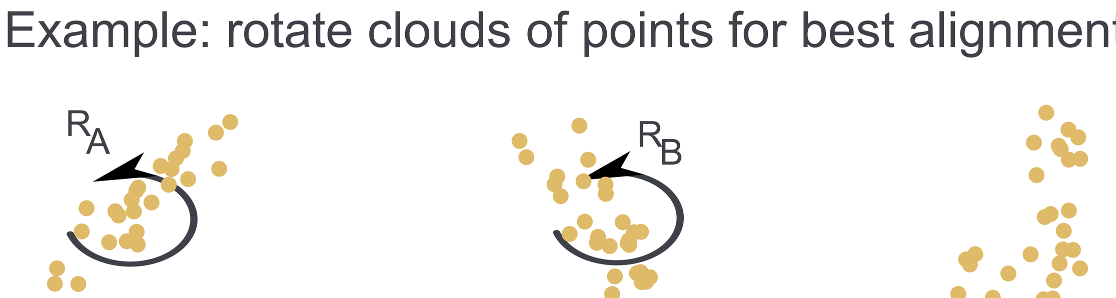
with mature theory and a friendly design

The toolbox keeps it simple:

Automatic conversion of Euclidean derivatives to their Riemannian counterpart, automatic Hessian approximation when needed, default values for all parameters, derivative checks, ...

Manopt is open source and documented.

Example: rotate clouds of points for best alignment



Rotation group: $\mathrm{SO}(3) = \{R \in \mathbb{R}^{3\times 3} : R^TR = I \text{ and } \det(R) = 1\}$

Search space: $\mathcal{M} = \mathrm{SO}(3) \times \mathrm{SO}(3)$

Cost function: $\|AR_A - BR_B\|^2 + \|AR_A - C\|^2 + \|BR_B - C\|^2$

$$f(R_A, R_B) = -2\mathrm{Trace}\left[R_A^TA^TBR_B + R_A^TA^TC + R_B^TB^TC\right]$$

$$\nabla f(R_A, R_B) = -2\begin{pmatrix} A^T(BR_B + C) \\ B^T(AR_A + C)\end{pmatrix}$$

```
import manopt.solvers.trustregions.*;
import manopt.manifolds.rotations.*;

% Create the problem structure.
manifold = rotationsfactory(n, 2);
problem.M = manifold;

% Define the problem cost function and its gradient.
problem.cost = @cost;
function f = cost(R)
    RA = R(:, :, 1); RB = R(:, :, 2);
    f = norm(A*RA-B*RB, 'fro')^2 + norm(A*RA-C, 'fro')^2 ...
      + norm(B*RB-C, 'fro')^2;
end

problem.grad = @(R) manifold.egrad2rgrad(R, grad(R));
function G = grad(R)
    RA = R(:, :, 1); RB = R(:, :, 2);
    G = zeros(n, n, 2);
    G(:, :, 1) = -2*A'*(B*RB+C);
    G(:, :, 2) = -2*B'*(A*RA+C);
end

% Solve.
[R Rcost info] = trustregions(problem);

% Display some statistics.
semilogy([info.iter], [info.gradnorm], '.-');
```