A (very) short presentation of Riemannian optimization using Pymanopt

Antoine Collas

Postdoc supervised by Alexandre Gramfort and Rémi Flamary

Optimization on a manifold

Optimization

$f : \mathcal{M} \to \mathbb{R}$, smooth

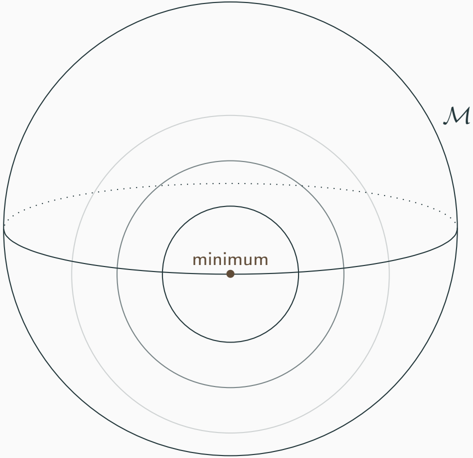
$$\underset{\theta \in \mathcal{M}}{\text{minimize}}\ f(\theta)$$

Examples of $\mathcal{M}$

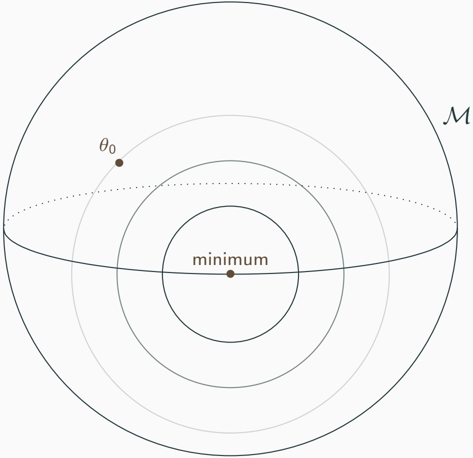
- linear spaces: $\mathbb{R}^{p\times k}$, $\mathcal{S}_p = \{\boldsymbol{X} \in \mathbb{R}^{p\times p} : \boldsymbol{X}^T = \boldsymbol{X}\}$,
- norm constraints: $S^{p^2-1} = \{\boldsymbol{X} \in \mathbb{R}^{p\times p} : \|\boldsymbol{X}\|_F = 1\}$,
- positivity constraints: $\mathcal{S}_p^{++} = \{\boldsymbol{\Sigma} \in \mathcal{S}_p : \forall \boldsymbol{x} \neq \boldsymbol{0} \in \mathbb{R}^p,\ \boldsymbol{x}^T \boldsymbol{\Sigma} \boldsymbol{x} > 0\}$,
- orthogonality constraints: $\mathrm{St}_{p,k} = \{\boldsymbol{U} \in \mathbb{R}^{p\times k} : \boldsymbol{U}^T \boldsymbol{U} = \boldsymbol{I}_k\}$,
- rank constraints: $\mathbb{R}_k^{n\times p} = \{\boldsymbol{X} \in \mathbb{R}^{n\times p} \text{ with } \operatorname{rank}(\boldsymbol{X}) = k\}$,

N. Boumal, “An introduction to optimization on smooth manifolds”

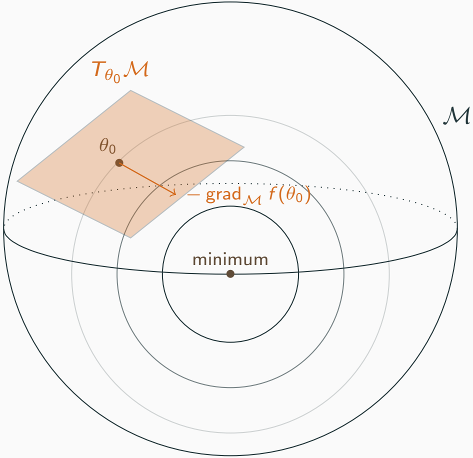
Optimization on a manifold



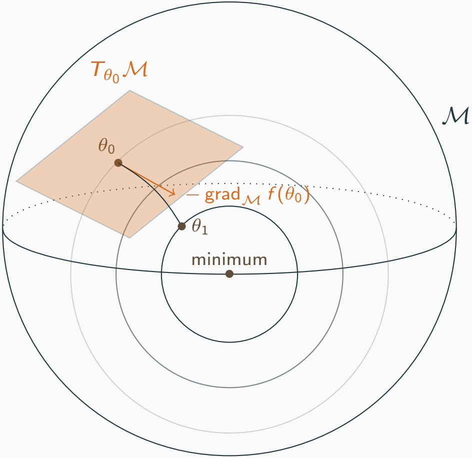
Optimization on a manifold



Optimization on a manifold



Optimization on a manifold

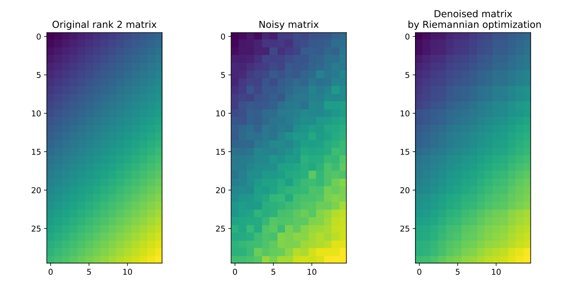


Example: low rank approximation

Given $\boldsymbol{A} \in \mathbb{R}^{n \times p}$,

$$\underset{\boldsymbol{X} \in \mathbb{R}_k^{n \times p}}{\text{minimize}} \; \|\boldsymbol{X} - \boldsymbol{A}\|_F^2$$

where $\mathbb{R}_k^{n \times p}$ is the manifold of $n \times p$ matrices with rank k.



Example: low rank approximation

```
# Generate a rank 2 matrix and its noisy version
n, d = 30, 15
A = np.array([np.arange(i, i + d) for i in range(n)])
A_noisy = torch.from_numpy(A + np.random.randn(n, d))

# Instantiate the manifold and the cost function
manifold = FixedRankEmbedded(n, d, k=2)

@pymanopt.function.pytorch(manifold)
def cost(u, s, vt):
        X = u @ torch.diag(s) @ vt
        return torch.norm(X - A_noisy) ** 2

problem = pymanopt.Problem(manifold, cost)

# Instantiate the optimizer and solve the problem
optimizer = ConjugateGradient()
u, s, vt = optimizer.run(problem).point
solution = u @ np.diag(s) @ vt
```