# Fast and accurate optimization on the orthogonal manifold without retraction

**Pierre Ablin** **Gabriel Peyré**
CNRS, Département de mathématiques et applications
ENS, PSL University

## Abstract

We consider the problem of minimizing a function over the manifold of orthogonal matrices. The majority of algorithms for this problem compute a direction in the tangent space, and then use a retraction to move in that direction while staying on the manifold. Unfortunately, the numerical computation of retractions on the orthogonal manifold always involves some expensive linear algebra operation, such as matrix inversion, exponential or square-root. These operations quickly become expensive as the dimension of the matrices grows. To bypass this limitation, we propose the landing algorithm which does not use retractions. The algorithm is not constrained to stay on the manifold but its evolution is driven by a potential energy which progressively attracts it towards the manifold. One iteration of the landing algorithm only involves matrix multiplications, which makes it cheap compared to its retraction counterparts. We provide an analysis of the convergence of the algorithm, and demonstrate its promises on large-scale and deep learning problems, where it is faster and less prone to numerical errors than retraction-based methods.

## 1 INTRODUCTION

We consider a differentiable function $f$ from $\mathbb{R}^{p\times p}$ to $\mathbb{R}$, and want to solve the problem

$$\min_{X\in\mathcal{O}_p} f(X) \ , \tag{1}$$

Proceedings of the 25th International Conference on Artificial Intelligence and Statistics (AISTATS) 2022, Valencia, Spain. PMLR: Volume 151. 

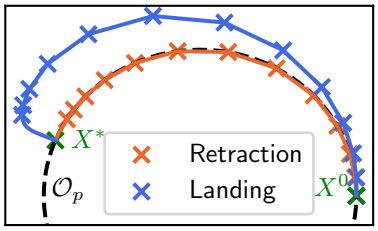


Figure 1: Trajectories of the landing algorithm and of a retraction gradient descent with $p = 2$. The iterates are $2 \times 2$ matrices, the $x$-axis corresponds to coefficient $(1, 1)$ of the matrices, and the $y$-axis to coefficient $(1, 2)$. The retraction method stays on $\mathcal{O}_p$ (black dotted line) while the landing algorithm can deviate. Both methods start from $X_0$ and converge to the correct solution $X_*$. In higher dimension, the landing algorithm is much cheaper than the retraction method.

where $\mathcal{O}_p$ is the *Orthogonal manifold*, that is the set of matrices $X \in \mathbb{R}^{p\times p}$ such that $XX^\top = I_p$. Problem (1) appears in many practical applications, like principal component analysis, independent component analysis (Comon, 1994; Nishimori, 1999; Ablin et al., 2018), procrustes problem (Schönemann, 1966), and more recently in deep learning, where the weights of a layer are parametrized by an orthogonal matrix (Arjovsky et al., 2016; Bansal et al., 2018). This is a particular instance of minimization over a matrix Riemannian manifold, $\mathcal{O}_p$ (Edelman et al., 1998). Many standard Euclidean algorithms for function minimization have been adapted on Riemannian manifolds. We can cite for instance gradient descent (Absil et al., 2009; Zhang and Sra, 2016), second order quasi-Newton methods (Absil et al., 2007; Qi et al., 2010), and stochastic methods (Bonnabel, 2013) which are the workhorse for training deep neural networks. More recently, several works propose to adapt accelerated methods in the Riemannian setting (Zhang and Sra, 2018; Tripuraneni et al., 2018).

All these methods are *feasible*, i.e. generate a sequence of iterates $X_k$ where each iterate is in $\mathcal{O}_p$. Unlike what

we assume in the first sentence of the present article, they do not need the function $f$ to be defined outside $\mathcal{O}_p$. This comes with a computational drawback: in order to compute $X_{k+1}$ from $X_k$, one needs a way to move and stay on the manifold, called *retraction* (Absil and Malick, 2012). Unfortunately, retractions on $\mathcal{O}_p$ are computationally expensive: they usually require a matrix inversion, square root, or exponential. These operations are also generally slow on modern computing hardware such as GPU's. Therefore, when the dimension $p$ is large, computing a retraction can become the computational bottleneck in the processing pipeline.

In this work, we propose the **landing algorithm**. It is an *infeasible* method, which produces iterates $X_k$ that are not necessarily orthogonal, but which converge to a local minimum of (1) as $k \to +\infty$. The iterates get closer and closer to the manifold, and at the limit, *land* on $\mathcal{O}_p$. The algorithm is illustrated in Figure 1 on a low dimensional problem. The main advantage of the method is that the update rule is much simpler than a retraction since it involves only a few matrix multiplications. As a result, our method can be much faster than standard feasible methods when $p$ is large.

Furthermore, retraction methods often suffer from an accumulation of numerical errors, which means that the iterates can get far from $\mathcal{O}_p$ after many steps of the algorithm. This effect is worsened by the low precision of floating point number that is customary in modern deep learning frameworks. On the other hand, our method can only converge to matrices such that $\|XX^\top - I_p\| = 0$ to numerical precision. Even though the proposed method is infeasible, it returns a solution that is closer to the manifold than most feasible methods in practice. Infeasible methods on $\mathcal{O}_p$ have recently gained interest (Xiao et al., 2020b,a, 2021). Closest to this work is (Gao et al., 2019), which proposes a Lagragian based update. It is not robust to the choice of hyper-parameter, which makes it hard to use in practice (see Appendix B).

The article is organized as follows: in Section 2, we recall some usual results about the geometry of $\mathcal{O}_p$ and Riemannian optimization algorithms. In Section 3, we introduce the landing algorithm and study global and local convergence. Some extensions are discussed. Finally, experiments in Section 4 show the benefit the landing algorithm over retraction methods in terms of computational efficiency and final distance to $\mathcal{O}_p$.

**Notation**: $\mathrm{Skew}_p$ is the set of skew-symmetric matrices, $\mathrm{Sym}_p$ is the set of symmetric $p \times p$ matrices. The Skew of a matrix $M \in \mathbb{R}^{p\times p}$ is $\mathrm{Skew}(M) = \frac{1}{2}(M - M^\top)$, and the Sym is $\mathrm{Sym}(M) = \frac{1}{2}(M + M^\top)$. The Euclidean gradient of $f$ is $\nabla f$, the Riemannian gradient is $\mathrm{Grad}\, f$. The norm is the Frobenius $\ell^2$ norm. The squared "distance" to the manifold is $\mathcal{N}(X) = \frac{1}{4}\|XX^\top - I_p\|^2$.

We give sketches of proofs in the main text. Detailed proofs are in appendix.

## 2 PRELIMINARIES

We recall concepts about optimization on manifolds that will be useful in the rest of this article.

### 2.1 Geometry of the orthogonal manifold

The orthogonal manifold is $\mathcal{O}_p \triangleq \{X \in \mathbb{R}^{p\times p} |\ XX^\top = I_p\}$. If $X(t)$ for $t \in [0, 1]$ is a differentiable curve on the manifold, differentiating the equation $X(t)X(t)^\top = I_p$ gives $\dot{X}(t)X(t)^\top + X(t)\dot{X}(t)^\top = 0$, hence $\dot{X}(t) \in \mathcal{T}_{X(t)}$ where $\mathcal{T}_X$ is the *tangent space* at $X$, given by $\mathcal{T}_X = \{\xi \in \mathbb{R}^{p\times p} |\ \xi X^\top + X\xi^\top = 0\}$. We see that a matrix $\xi$ is in $\mathcal{T}_X$ if and only if for $A \in \mathrm{Skew}_p$ we have $\xi = AX$. It then easily seen that the tangent space is a linear space of dimension $\frac{p(p-1)}{2}$. The projection on the manifold $\mathcal{P}(X) \triangleq \arg\min_{Y\in\mathcal{O}_p} \|X - Y\|$ is $\mathcal{P}(X) = (XX^\top)^{-\frac{1}{2}}X$. We now turn our attention to optimization on $\mathcal{O}_p$.

### 2.2 Relative optimization on $\mathcal{O}_p$ and extension to $\mathbb{R}^{p\times p}$

Vectors in the tangent space at $X$ are of the form $AX$ with $A \in \mathrm{Skew}_p$. The effect of small perturbations of $X$ in the direction $AX$ on $f$ leads to so-called *relative* derivatives (Cardoso and Laheld, 1996):

**Definition 1.** *For $X \in \mathbb{R}^{p\times p}$, the relative gradient $\psi(X) \in \mathrm{Skew}_p$ is defined with the Taylor expansion, for $A \in \mathrm{Skew}_p$: $f(X + AX) = f(X) + \langle A, \psi(X)\rangle + o(\|A\|)$. The relative Hessian $\mathcal{H}_X$ is the linear operator $\mathrm{Skew}_p \to \mathrm{Skew}_p$ such that $\psi(X + AX) = \psi(X) + \mathcal{H}_X(A) + o(\|A\|)$.*

These quantities are not defined only on $\mathcal{O}_p$, but on the whole $\mathbb{R}^{p\times p}$, and can be computed easily from the Euclidean derivatives of $f$.

**Proposition 1** (Relative from Euclidean)**.** *Let $\nabla f(X) \in \mathbb{R}^{p\times p}$ the Euclidean gradient and $H_X : \mathbb{R}^{p\times p} \to \mathbb{R}^{p\times p}$ the Euclidean Hessian of $f$ at $X$. We have $\psi(X) = \mathrm{Skew}(\nabla f(X)X^\top)$ and $\mathcal{H}_X(A) = \mathrm{Skew}(H_X(AX)X^\top - \nabla f(X)X^\top A)$*

We can recover the *Riemannian* gradient and Hessian of $f$ from the Relative derivatives:

**Proposition 2** (Riemannian from relative)**.** *For $X \in \mathcal{O}_p$, we have $\mathrm{Grad}\, f(X) = \psi(X)X$, and for $A \in \mathrm{Skew}_p$, we have $\mathrm{Hess}\, f(X)(AX) = \mathcal{H}_X(A)X + \mathrm{Skew}(\psi(X)A)X$.*

Therefore, the critical points of $f$ on $\mathcal{O}_p$, i.e. the points such that $\mathrm{Grad}\, f(X) = 0$, are exactly the points

such that $\psi(X) = 0$, and at those points, we have $\operatorname{Hess} f(X)(AX) = \mathcal{H}_X(A)X$: the Hessians are the same up to a remapping.

### 2.3 Optimization on the orthogonal manifold with retractions

A simple method to solve Problem (1) is the Riemannian gradient flow, which is the Ordinary Differential Equation (ODE) starting from $X_0 \in \mathcal{O}_p$

$$X(0) = X_0, \quad \dot{X}(t) = -\operatorname{Grad} f(X(t)) \ . \tag{2}$$

It is easily seen that the trajectory of the ODE stays in $\mathcal{O}_p$, and that $f(X(t))$ decreases with $t$. Further assumptions on $f$, like Polyak-Lojasiewicz inequalities (Karimi et al., 2016; Balashov et al., 2020) or geodesic strong-convexity allow to prove the convergence of $X(t)$ to a minimizer as $t \to +\infty$. If $f$ is Lipschitz then we have global convergence to a stationary point: $\liminf \|\operatorname{Grad} f(X(t))\| = 0$. In order to obtain a practical optimization algorithm, one should discretize the gradient flow. Sadly, a naive Euler discretization, iterating $X_{k+1} = X_k - \eta \operatorname{Grad} f(X_k)$ with $\eta > 0$ yields iterates which do not belong to the manifold, because the curvature is not considered.

This motivates the use of retractions. A retraction $\mathcal{R}$ maps $(X, \xi)$ where $X \in \mathcal{O}_p$ and $\xi \in \mathcal{T}_X$ to a point $\mathcal{R}(X, \xi) \in \mathcal{O}_p$, and is such that $\mathcal{R}(X, \xi) = X + \xi + o(\|\xi\|)$. Since the tangent space has such a simple structure, it is easier to describe a retraction with the mapping $\tilde{\mathcal{R}}(X, A)$, where $A \in \mathrm{Skew}_p$, such that $\tilde{\mathcal{R}}(X, A) = \mathcal{R}(X, AX)$. Table 1 lists four popular

| Name | Formula for $\tilde{\mathcal{R}}(X, A)$ |
|---|---|
| Exponential | $\exp(A)X$ |
| Projection | $\mathcal{P}(X + AX)$ |
| Cayley | $(I_p - \frac{A}{2})^{-1}(I_p + \frac{A}{2})X$ |
| QR | $\mathrm{QR}(X + AX)$ |

Table 1: Popular retractions

retractions. They all involve linear algebra operations on matrices like inversion, square root, or exponential. There is no "simpler" retraction:

**Proposition 3** (No polynomial retraction)**.** *Fix $X \in \mathcal{O}_p$. There is no polynomial $P(A)$ such that $\tilde{\mathcal{R}}(X, A) = P(A)$ is a retraction at $X$.*

*Proof.* By contradiction, such polynomial must satisfy $P(A)P(A)^\top = I_p$. Thus, $PP^\top$ is of degree 0, hence $P$ is of degree 0, and $P$ is constant. Therefore, we cannot have $P(A) = X + AX + o(\|AX\|)$. □

Of course, in practice, most retractions are implemented using polynomial approximations (see e.g. (Moler and Van Loan, 2003; Li et al., 2020)). The previous proposition simply shows that polynomials can only be approximations, and as a consequence, *any* retraction must involve some linear algebra more complicated than matrix multiplication.

Riemannian gradient descent uses a retraction to stay on the manifold. It iterates

$$X_{k+1} = \mathcal{R}(X_k, -\eta \operatorname{Grad} f(X_k)) \ , \tag{3}$$

where $\eta > 0$ is a step-size. Riemannian gradient descent is conveniently written with the relative gradient $\psi$ as $X_{k+1} = \tilde{\mathcal{R}}(X_k, -\eta\psi(X_k))$. We now present the landing algorithm, which *does not require retractions.*

## 3 THE LANDING ALGORITHM

In the following, we use the function $\mathcal{N}(X) \triangleq \frac{1}{4}\|XX^\top - I_p\|^2$. This function is minimized if and only if $X \in \mathcal{O}_p$. A simple way to build an algorithm that converges to $\mathcal{O}_p$ consists in following $-\nabla\mathcal{N}(X)$, which leads in the continuous setting to Oja's flow $\dot{X} = -\nabla\mathcal{N}(X)$ (Oja, 1982; Yan et al., 1994) and in the discrete setting to Potter's algorithm $X_{k+1} = X_k - \eta\nabla\mathcal{N}(X_k)$ (Cardoso and Laheld, 1996). Note that the Euclidean gradient has the simple formula $\nabla\mathcal{N}(X) = (XX^\top - I_p)X$, and that it is always orthogonal to the Riemannian gradient of $f$, since $\nabla\mathcal{N}(X)$ is written as $SX$ with $S \in \mathrm{Sym}_p$, while $\operatorname{Grad} f(X)$ is written as $AX$ with $A \in \mathrm{Skew}_p$.

The landing algorithm combines the previous orthogonalizing method with the minimization of $f$. We define

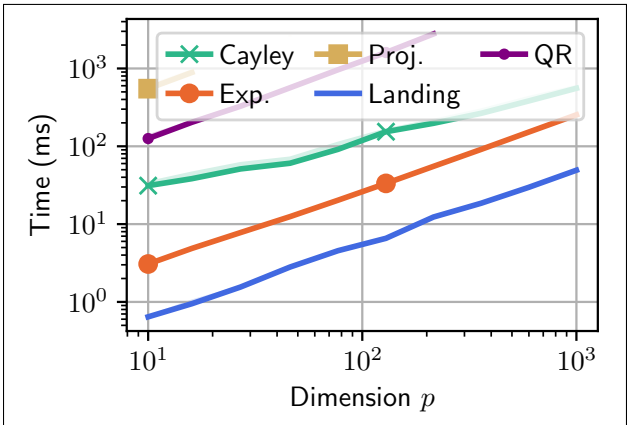


Figure 2: Time required to compute 500 retractions $\tilde{\mathcal{R}}(X, A)$ when $A$ and $X$ are of size $p \times p$, on a GPU.

the **landing field** as the mapping $\mathbb{R}^{p\times p} \to \mathbb{R}^{p\times p}$:

$$\boxed{\Lambda(X) \triangleq \psi(X)X + \lambda\nabla\mathcal{N}(X)} \ , \tag{4}$$

where $\lambda > 0$ is a fixed parameter. This allows us to define the **landing algorithm**, which iterates:

$$X_{k+1} = X_k - \eta^k \Lambda(X_k) \ , \tag{5}$$

with $\eta^k > 0$ a sequence of step-sizes. Its continuous counterpart is the **landing flow**:

$$\dot{X}(t) = -\Lambda(X(t)). \tag{6}$$

We stress that the field $\Lambda$ is not the Riemannian gradient nor the Euclidean gradient of a function (its Jacobian is not symmetric). In particular, the landing flow does *not* have the same trajectory as the Euclidean gradient flow associated to the function $f(X)+\lambda\mathcal{N}(X)$.

Before we move on to the analysis of the algorithm, we can already see that one iteration of the landing algorithm only involves some matrix multiplications instead of expensive linear algebra. In Figure 3, we show the cost of computing on a GPU one iteration of the Riemannian gradient descent using the standard retractions, and the cost of computing one iteration of the landing algorithm as $p$ grows. The proposed method is about 8 times faster than retraction methods.

**Comparison to penalty methods** An idea to get an approximation of problem (1) is to minimize, without constraint, the penalized function $g(X) = f(X) + \lambda\mathcal{N}(X)$. This is conceptually simpler than Riemannian optimization, and can be implemented very easily. However, the main drawback of this method is that the solution will not in general be feasible. Furthermore, to implement this method, we need to compute the gradient of $g$ given by $\nabla f(X) + \lambda(XX^\top - I_p)X$. On top of computing the gradient, we see that it involves 2 matrix multiplications. By comparison, computing the landing field $\Lambda$ requires 3 matrix multiplications. Hence, computing the landing field is only 50% more costly than the gradient of $g$, and as we will see, it provides us with a feasible solution.

**Computational cost of Riemannian gradient descent** Riemannian gradient descent on $\mathcal{O}_p$ first computes the descent direction, using the Euclidean gradient of $f$, and then computes the next iterate using a retraction. Depending on the problem, the main computational bottleneck may come from either of the two steps. For instance, when training a Recurrent Neural Network (RNN) with orthogonal weights (Arjovsky et al., 2016; Helfrich et al., 2018; Lezcano Casado, 2019), there is usually only one orthogonal matrix used in a large computational graph. Here, the cost of computing the Euclidean gradient with backpropagation is much higher than the cost of computing a retraction. Consequently, using the landing algorithm in this setting will only slightly reduce the cost of computations. On the other hand, it is common to impose an orthogonality constraint on multilayer perceptron or convolutional neural networks (Rodríguez et al., 2016; Bansal et al., 2018). It has been reported that this constraint allows faster training and better generalization. In this case, there are many orthogonal matrices, and the main bottleneck in training can be computing the retraction. Therefore, as we will see in the experiments, it is interesting to use the landing algorithm in this setting.

We now turn to a theoretical analysis of the method, and begin by a study of the critical points.

**Proposition 4** (Critical points of $\Lambda$)**.** *Let $X \in \mathbb{R}^{p\times p}$ invertible. We have $\Lambda(X) = 0$ if and only if $X \in \mathcal{O}_p$ and $\psi(X) = 0$.*

*Proof.* If $\Lambda(X) = 0$, we have $\psi(X)+\lambda(XX^\top - I_p) = 0$. Looking at the symmetric part, we obtain $XX^\top = I_p$, i.e. $X \in \mathcal{O}_p$. Looking at the skew-symmetric part, we obtain $\psi(X) = 0$. Conversely, if $\psi(X) = 0$ and $X \in \mathcal{O}_p$, we have $\Lambda(X) = 0$. □

This result shows that the stationary points of the landing algorithm are the stationary points of the original problem (1). It holds regardless of the value of the hyper-parameters $\lambda$. We also stress that the invertibility condition on $X$ is not a problem in practice: as we will see in the next section, the iterates stay close enough to $\mathcal{O}_p$ so that they are bounded away from the singular matrices set: the stationary points for the landing algorithm/flow are the stationary points of $f$.

### 3.1 Orthogonalization property

We start by showing that the landing flow (6) is well defined and *orthogonalizing*: the flow converges to the orthogonal manifold regardless of initialization.

**Proposition 5** (Convergence of the flow to $\mathcal{O}_p$)**.** *There is a solution $X(t)$ of the landing flow (6) defined for all $t \geq 0$. Then, $\mathcal{N}(X(t))$ decreases, and denoting $N_0 \triangleq \mathcal{N}(X_0)$, we have $\mathcal{N}(X(t)) \leq e^{-\lambda t}\left(\frac{N_0}{(\sqrt{N_0}-1)^2}\right)$.*

*Proof.* Let $n(t) = \mathcal{N}(X(t))$. We find $n'(t) = \langle \dot{X}(t), \nabla\mathcal{N}(X(t))\rangle$. Then, we have for all $X$, $\langle \psi(X)X, \nabla\mathcal{N}(X)\rangle = \langle \psi(X), (XX^\top - I_p)XX^\top\rangle$. The matrix on the left is skew-symmetric, the matrix on the right is symmetric, hence this scalar product cancels. Therefore, we get $n'(t) = -\lambda\|\nabla\mathcal{N}(X(t))\|^2$. This shows that $n(t)$ decreases. This proves the existence of a solution for all times, by a standard Lyapunov argument. Next, we use the inequality $\|(XX^\top - I_p)X\|^2 \geq \mathcal{N}(X) - \mathcal{N}(X)^{\frac{3}{2}}$ which gives us $n'(t) \leq -\lambda(n(t) - n(t)^{\frac{3}{2}})$. This inequality is then integrated to obtain the result. □

This shows that the landing flow produces a trajectory that *lands* on the manifold: the distance to the manifold decreases at a linear rate to 0. If the landing flow starts

**Algorithm 1** Landing algorithm with safe step-size

**Input :** Initial point $X_0 \in \mathcal{O}_p$, step-size sequence $\eta_k$, number of iterations $N$.
**for** $k = 1$ **to** $N$ **do**
  Compute $\eta^*$ (Proposition 6)
  Set $\eta_k = \min(\eta^*, \eta_k)$
  Update $X_{k+1} = X_k - \eta_k \Lambda(X_k)$
**end for**
**Return :** $X_N$.

on the manifold ($X_0 \in \mathcal{O}_p$), then $\mathcal{N}(X(t)) = 0$ for all $t \geq 0$, i.e. the flow stays on the manifold, and is equal to the Riemannian gradient flow (2).

**Safe rule for the discrete algorithm** The convergence of the landing *algorithm* towards $\mathcal{O}_p$ is more complicated to study. For instance if $X_0 \in \mathcal{O}_p$, then $X_1 = X_0 - \eta\psi(X_0)X_0$ is not orthogonal unless $\psi = 0$. Therefore, there is no hope that $\mathcal{N}(X_k)$ is a decreasing sequence. Instead, we set $\varepsilon > 0$, and get a criterion on the step-size which ensures $\mathcal{N}(X_k) \leq \varepsilon$ for all $k$.

**Proposition 6** (Safe step-size interval)**.** *Assume that $X_k$ is such that $d \triangleq \mathcal{N}(X_k) \leq \varepsilon$. Let $a \triangleq \|\psi(X_k)\|$ and $\eta^*(a,d) \triangleq \frac{\sqrt{\alpha^2 + 4\beta(\varepsilon - d)} + \alpha}{2\beta}$, where $\alpha \triangleq 2\lambda d - 2ad - 2\lambda d^2$ and $\beta \triangleq a^2 + \lambda^2 d^3 + 2\lambda a d^2 + a^2 d$. Then if $\eta \in [0, \eta^*(a,d)]$, we have $\mathcal{N}(X_{k+1}) \leq \varepsilon$.*

As a consequence, if the algorithm starts from $X_0 \in \mathcal{O}_p$ and $\eta$ is in the safe interval for each step, the iterates all verify $\mathcal{N}(X_k) \leq \varepsilon$. It is worth mentioning that while the above formula is complicated, it is only a matter of computing a scalar function given $\|\psi(X_k)\|$ and $\mathcal{N}(X_k)$, so computing $\eta^*(a,d)$ is negligible in front of the other computations. In practice, we provide a sequence of target step-size $\eta_k$ to the algorithm, and at each iteration, we compute $\eta^*$. We then use $\min(\eta_k, \eta^*)$ as the step-size. Importantly, we see that when $a = 0$, the safe step-size is of the order $\eta^* \simeq \frac{4}{\lambda d^2} \leq \frac{4}{\lambda \varepsilon^2}$, which is large when $\varepsilon$ is small.When $d = 0$, we have $\eta^* = \frac{\sqrt{\varepsilon}}{a}$, which is reminiscent of the baseline step-size (inverse of Lipschitz constant of the problem). In practice, we take $\varepsilon = \frac{1}{2}$, which ensures that the safe-step size is not small in the two previous settings. This safe rule therefore does not restrict much the choice of step-size, which is also observed in practice. This gives us a safe landing algorithm, described in Algorithm 1. We stress that the condition $\mathcal{N}(X) \leq \varepsilon$, which is imposed using this safe-step technique, guarantees that $X$ is invertible as soon as $\varepsilon < 1$, since for any singular $X$ we have $\mathcal{N}(X) \geq 1$.

Convergence of $N(X_k)$ to 0 depends on the convergence of $\psi(X_k)$ to 0, which requires global convergence results, presented later in Section 3.3.

**Stochastic method and distance to $\mathcal{O}_p$ in the small gradient regime** When $f$ has a sum structure, $f(X) = \sum_{i=1}^n f_i(X)$, it is possible to use stochastic gradient descent, which takes a step in the opposite direction of the gradient of one of the $f_i$ instead of $f$. Such method is easily adapted to the Riemannian setting, by taking Riemannian stochastic gradients and using retractions or the landing algorithm. Defining $\psi_i$ as the relative gradient of the function $f_i$, the stochastic landing algorithm samples $i_k$ at random between 1 and $n$, and then does a step $X_{k+1} = X_k - \eta_k \left(\psi_{i_k}(X_k) + \lambda(X_k X_k^\top - I_p)\right) X_k$.

We now detail an informal computation to control the distance of the iterates to $\mathcal{O}_p$ when the gradients $\psi_i(X_k)$ are small. Denoting $\Delta_k \triangleq X_k X_k^\top - I_p$, and neglecting high order terms in $\psi$ and $\Delta_k$, one has the approximate relationship $\Delta_{k+1} \simeq (1 - 2\eta_k\lambda)\Delta_k - \eta_k^2 \left(\psi_i(X_k)\right)^2$. Assuming that the gradients $\psi_i(X_k)$ are independent from $\Delta_k$ and have an average norm $a$, we find $\mathbb{E}[\|\Delta_{k+1}\|^2] = (1 - 2\eta_k\lambda)^2 \mathbb{E}[\|\Delta_k\|^2] + \eta_k^4 a^4$. If the step-sizes $\eta_k$ are fixed to $\eta > 0$, the above equation indicates that $\mathbb{E}[\|\Delta_k\|^2]$ converges to a limit value given by $\left(\mathbb{E}[\|\Delta_k\|^2]\right)^{1/2} \to \delta_* \triangleq \frac{\eta a^2}{2\lambda}$. The above reasoning is informal and there are many approximations. However, we find that $\delta^*$ is close to the distance to the manifold $\mathcal{O}_p$ observed in practice.

### 3.2 Local convergence

In this section, we assume that the iterates are close to a local minimum of (1), and study its stability. We let $X_* \in \mathcal{O}_p$ such that $\psi(X_*) = 0$ and $\mathcal{H}_{X^*}$ is positive, and study its stability. We let $\mu_{\min} > 0$ the smallest eigenvalue of $\mathcal{H}_{X^*}$.

**Proposition 7** (Local convergence, landing flow)**.** *For any $\delta > 0$, there exists $\epsilon > 0$ such that if $\|X_0 - X_*\| \leq \epsilon$, the landing flow starting from $X_0$ verifies $\|X(t) - X_*\| = O\left(\exp(-(\min(\mu_{\min}, \lambda) + \delta)t)\right)$.*

Therefore, if $\lambda \geq \mu_{\min}$, we get the same local convergence speed for the landing flow and the Riemannian gradient flow. We obtain a similar result in the discrete case, using a Lipschitz assumption.

**Proposition 8** (Local convergence, landing algorithm)**.** *Assume that $\Lambda$ is $L-$Lipschitz. Then for any $\delta > 0$ there exists $\epsilon > 0$ such that if $\|X_0 - X_*\| \leq \epsilon$, the landing algorithm starting from $X_0$ with constant step $\eta \leq \frac{1}{L}$ verifies $\|X_k - X_*\| = O\left((1 - \frac{\min(\mu_{\min}, \lambda)}{L} + \delta)^k\right)$.*

These two results follow from the expression of the Jacobian of the field $\Lambda$ at $X_*$. Once again, when $\lambda \geq \mu_{\min}$, we get the same rate as Riemannian gradient descent (Zhang and Sra, 2016).

**Hyper-parameter trade-off** The hyper-parameter $\lambda$ plays a key role in the convergence results. Proposition 8 suggests that $\lambda$ should be chosen to maximize $\frac{\min(\mu_{\min},\lambda)}{L}$. Since $L$ is the Lipschitz constant of $\Lambda$, we have an upper bound of the form $L \leq l_1 + \lambda l_2$ with $l_1, l_2$ the respective Lipschitz constants of $\psi(X)X$ and $\nabla\mathcal{N}(X)$. Then, $\frac{\min(\mu_{\min},\lambda)}{L}$ is maximized for $\lambda = \mu_{\min}$: this is in theory the best value of $\lambda$ to get fast local convergence. However, this constant is usually intractable. In the experiments, we take $\lambda = 1$, which in practice gives satisfying results.

### 3.3 Global convergence

We now give a global convergence result for the landing flow:

**Proposition 9** (Global convergence, continuous case)**.** *Let $T \geq 0$. We assume that for all $t \leq T$, $\|\operatorname{Sym}(\nabla f(X(t))X(t)^\top)\| \leq K$, and we let $f^* = \min f$. We have*

$$\inf_{t\leq T} \|\psi(X_t)\| \leq \frac{1}{\sqrt{T}}\left(f(X_0) - f^* + 2K\frac{\sqrt{N_0}}{\sqrt{N_0}-1}\right)^{\frac{1}{2}}.$$

This shows global convergence of the flow at the usual rate $1/\sqrt{T}$. This result is analogous to the one one would get following the Riemannian gradient flow on the manifold (e.g. (Boumal, 2020, Prop. 4.6)).

In the discrete case, we show that the landing algorithm with constant step-size $\eta$ produces iterates that get at a distance of the order $\eta$ to the stationary points.

**Proposition 10** (Global convergence, discrete + fixed step-size case)**.** *Let $X_k$ the sequence of iterates of the landing algorithm with step-size $\eta$, starting from $X_0 \in \mathcal{O}_p$. There exists constants $\delta, C_1, C_2 > 0$ (given in Appendix) such that when $\eta \leq \delta$, it holds $\mathcal{N}(X_k) \leq \eta \cdot C_1$ and $\inf_{k\geq 0}\|\psi(X_k)\|^2 \leq \sqrt{\eta}\cdot C_2$.*

We have not been able to show stronger convergence results in the fixed step-size regime. Based on empirical evidence, we conjecture that for $\eta$ small enough we have $\lim_{k\to+\infty}\|\psi(X_k)\|^2 = 0$ and $\lim_{k\to+\infty}\mathcal{N}(X_k) = 0$. The following proposition shows convergence of the algorithm with decreasing step-size:

**Proposition 11** (Global convergence, discrete + decreasing step-size case)**.** *Let $X_k$ the sequence of iterates of the landing algorithm with step-size $\eta_k = k^{-\alpha}$ with $\alpha \in (\frac{1}{2}, 1)$, starting from $X_0 \in \mathcal{O}_p$. Then, $\mathcal{N}(X_k) = \mathcal{O}(k^{-\alpha})$ and $\inf_{k\geq 0}\|\psi(X_k)\|^2 = O(k^{-\min(\frac{\alpha}{2}, 1-\alpha)})$.*

This proposition shows convergence of the landing algorithms: the iterates land on the manifold since $\mathcal{N}(X_k)$ goes to 0, and they go towards stationary points of $f$ since $\inf_{k\geq 0}\|\psi(X_k)\|^2$ goes to 0. The best rate of convergence is obtained for $\alpha = \frac{2}{3}$ and we find $\inf_{k\geq 0}\|\psi(X_k)\|^2 = O(k^{-\frac{1}{3}})$. In contrast, Riemannian gradient descent achieves a rate of $O(k^{-1})$.

### 3.4 Acceleration with momentum

It is straightforward to derive a momentum version of the landing algorithm by accumulating the relative gradients. Starting from the initial speed $A_0 = 0$, for a momentum term $\gamma \in [0, 1]$, the landing algorithm with momentum iterates

$$\begin{cases} A_{k+1} = (1-\gamma)A_k + \gamma\psi(X_k) \\ X_{k+1} = X_k - \eta_k(A_k X_k + \lambda\nabla\mathcal{N}(X_k)). \end{cases} \tag{7}$$

In Eq. (7), the relative gradient can be replaced by a stochastic estimate. This leads to significant acceleration in the deep learning experiments. The corresponding second order ODE is

$$\begin{cases} \dot{A}(t) = -A(t) + \psi(X(t)) \\ \dot{X}(t) = -\left(A(t) + \lambda(X(t)X(t)^\top - I_p)\right)X(t) \end{cases} \tag{8}$$

It is readily seen that $A(t)$ is skew-symmetric for all $t$, and therefore that we get the same convergence result as Prop. 5. Classical arguments with the Lyapunov function $f(X(t)) + \frac{1}{2}\|A(t)\|^2$ also provide global convergence : $\liminf \|\psi(X(t))\| = 0$ (See Appendix): the analysis in the continuous case is almost as straightforward as with no momentum.

### 3.5 A landing field for other manifolds ?

The landing field can in principle be extended to (sub-)manifolds $\mathcal{M}$ of $\mathbb{R}^d$ that are *orientable*(see e.g. (Boumal, 2020, Chapter 3) and (Lee, 2013, Prop 15.23)). Indeed, one can derive a field $G(x)$ and a potential $\mathcal{N}(x)$ such that when $x \in \mathcal{M}$, $G(x) = \operatorname{Grad} f(x)$, such that $\nabla\mathcal{N}(x)$ is 0 if and only if $x \in \mathcal{M}$, and such that $G(x)$ and $\nabla\mathcal{N}(x)$ are always orthogonal. These properties are sufficient to obtain Proposition 4. However, these maps might not be tractable, while on $\mathcal{O}_p$ their expressions are simple and cheap to compute.

**Stiefel manifold** The Stiefel manifold $\mathcal{S}_{n,p}$ is the set of *rectangular* matrices $X \in \mathbb{R}^{n\times p}$ with $n > p$ such that $X^\top X = I_p$. The Riemannian gradient of $f$ is once again given by the formula $\operatorname{Grad} f(X) = \psi(X)X$, with $\psi(X) = \operatorname{Skew}(\nabla f(X)X^\top)$. Here, $\psi(X)$ is a large $n \times n$ matrix,but only $\psi(X)X$ appears in the formula which can be computed at a $O(n \times p^2)$ cost. The distance function becomes $\mathcal{N}(X) = \|X^\top X - I_p\|^2$, and the landing field can then be defined as $\Lambda(X) = \psi(X)X + \lambda\nabla\mathcal{N}(X)$. We obtain the equivalent of Proposition 4 and Proposition 5: the points such that $\Lambda(X) = 0$ are exactly those for which $X \in \mathcal{S}_{n,p}$ and $\operatorname{Grad} f(X) = 0$,

and we get a similar orthogonalization property of the flow. We finish by stressing that some "fast" retractions are available for $\mathcal{S}_{n,p}$ when $p$ is much smaller than $n$: Cayley retraction can be computed by inverting a small $2p \times 2p$ matrix. In this setting, the landing flow might not be much faster than this retraction.

### 3.6 Numerical errors

An advantage of our method is that it is robust to numerical errors. Indeed, at convergence, the landing flow goes to a point $X$ such that $\|\Lambda(X)\|^2 \leq \delta_{\text{num}}$, where $\delta_{\text{num}}$ is a small constant that depends on the floating point precision. Therefore, $\|XX^\top - I_p\|^2 \leq \delta_{\text{num}}$: at the limit, the orthogonalization error is of the order of the floating point precision. This is observed in practice.

On the contrary, consider for instance the exponential retraction. Starting from $X_0 \in \mathcal{O}_p$, it iterates $X_k = \exp(A_k)X_{k-1}$, where $A_k$ is a skew-symmetric matrix. For simplicity, assume that $X_0 = I_p$. The iterate $X_k$ can be compactly rewritten as $X_k = \prod_{i=1}^k \exp(A_i)$. Hence, if the $\exp(A_i)$ are not perfectly orthogonal because of numerical errors, $X_k$ can get further and further from orthogonality as $k$ increases. Therefore, the landing algorithm, while it is a non-feasible method, gives a solution that is more orthogonal than methods using the exponential or the Cayley retraction.

## 4 EXPERIMENTS

We conclude by showcasing the usefulness of the landing algorithm on an array of optimization problems. The deep learning experiments are run on a single Tesla V100 GPU with Pytorch (Paszke et al., 2019), while the other experiments are run on a small laptop CPU and Numpy (Harris et al., 2020). The code for the landing flow as a Pytorch `Optimizer` and to reproduce the experiments is available at `https://github.com/pierreablin/landing`. In all experiments, we use the safe rule for the step-size described in Proposition 6, with $\varepsilon = 0.5$, and set $\lambda = 1$.

**Orthogonal procrustes** We let $A, B$ two $p \times p$ matrices, and define the Procrustes cost function as $f(X) = \|XA - B\|^2$ where $X \in \mathcal{O}_p$. We set $p = 40$. We generate $A$ and $B$ two random matrices with i.i.d. normal entries. We apply the different algorithms with a fixed step-size $\eta = 0.1$. We record the distance to the solution $X_*$, and the orthogonalization error $\|XX^\top - I_p\|$. Figure 3 displays the results. Looking at the distance to the optimum, all methods are similar in term of iterations. Since one iteration of the landing algorithm is cheaper, we get an overall faster method. Looking at the distance to the manifold, the landing algorithm starts by moving away from the manifold, but in the end lands on the manifold. The exponential and Cayley retractions suffer from numerical errors, and end up being further from the manifold than the landing algorithm.

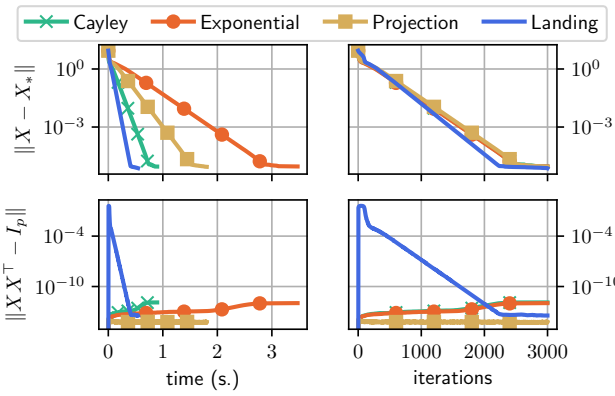


Figure 3: Gradient descent on the orthogonal procrustes problem using different retraction methods and the landing algorithm. Top: distance to the optimum. Bottom: distance to the orthogonal manifold. Left: w.r.t. time. Right: w.r.t. iterations.

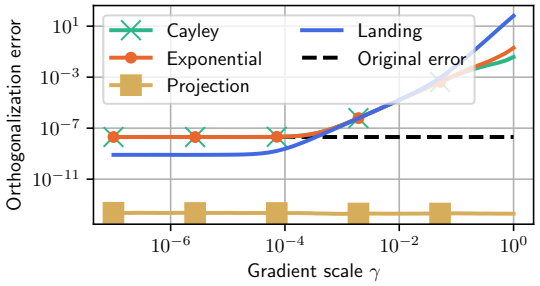


Figure 4: Orthogonality error after one step of each algorithm, starting from a matrix that is close to, but not in $\mathcal{O}_p$.

**Orthogonalization property** We illustrate the orthogonalization property of the landing algorithm. We take $p = 100$, $X_0 = I_p$, and add a small error $X = X_0 + E$, where $E$ has i.i.d. entries of law $\mathcal{N}(0, \sigma^2)$, with $\sigma = 10^{-4}$. Therefore, $X$ is close to, but not perfectly in $\mathcal{O}_p$. Then, we generate a random 'gradient', that is a random matrix $A \in \text{Skew}_p$ of i.i.d. entries where $A_{ij} \simeq \mathcal{N}(0, \gamma^2)$ for $i > j$, where $\gamma$ controls the scale of the 'gradient' $A$. Setting $\gamma$ small emulates an optimization problem closed from being solved, and $\gamma$ high an optimization problem far from being solved. We then take a step in the direction of $A$, with step-size $\eta = .3$. For the landing algorithm, we set $\lambda = 1$. In other words, we take the output as $X_{out} = \tilde{\mathcal{R}}(X, \eta A)$ for the retraction algorithms, and $X_{out} = X - \eta\left(A + XX^\top - I_p\right)X$ for the landing algorithm. We then record the orthogonality error of

the output, $\|X_{out}X_{out}^\top - I_p\|$. For each $\gamma$, we repeat the experiment 50 times with different random seeds. Figure 4 shows the average orthogonality error of the different algorithms as a function of the gradient scale $\gamma$. The projection retraction yields a very small orthogonalization error. The exponential and Cayley retraction do not have this correcting effect: the orthogonalization error stays the same if $\gamma$ is small. When $\gamma$ gets large enough, they increase the orthogonalization error. Finally, the landing algorithm has a hybrid behavior. When the $\gamma$ is small, it acts mainly as a cheap projection algorithm, which means that the orthogonalization error decreases. Here, one iteration reduces the error by a factor $\simeq 10$, so after a few iterations the algorithm would reach numerical precision. This illustrates a "self-correcting" behavior of the landing algorithm: unlike the exponential and Cayley retractions, it can decrease the orthogonalization error.

**Deep learning** We now turn to applications in deep learning, where the function $f$ involves a neural network. In this part, we discard the projection retraction, which is orders of magnitude more costly to compute than other retractions on a GPU (see Figure 3).

**Distillation** We begin by considering a fully connected neural network of depth $D$ that maps the input $x_0 \in \mathbb{R}^p$ to the output $x_D \in \mathbb{R}^p$ following the recursion $x_{n+1} = \tanh(W_n x_n + b_n)$, where $W_n \in \mathcal{O}_p$ are the weight matrices and $b_n \in \mathbb{R}^p$ are the biases. We denote $\Phi_\theta(x)$ the output of the network with input $x \in \mathbb{R}^p$ and parameters $\theta = (W_1, b_1, \dots, W_D, b_D)$. In this experiment, we consider a *distillation* problem (Hinton et al., 2015): we generate a random set of parameters $\theta^*$ that gives the target *teacher* network $\Phi_{\theta^*}$. Then, starting from a random parameter initialization, we try to learn the mapping $\Phi_{\theta^*}$ with a *student* network with parameters $\theta$, by minimizing the loss $\mathcal{L}(\theta) = \sum_{q=1}^{Q} \|\Phi_\theta(x^q) - \Phi_{\theta^*}(x^q)\|^2$, where the $x^q$ are the training examples, drawn i.i.d. from a normal distribution. We consider the optimization of the orthogonal weights $W_1, \dots, W_L$ with different methods. We use stochastic Riemannian gradient descent using the exponential or Cayley retraction or the landing flow, with or without momentum for the latter. We also consider trivializations (Lezcano-Casado and Martınez-Rubio, 2019; Lezcano Casado, 2019), where each matrix is parametrized as $W_i = \exp(A_i)$ with $A_i \in \text{Skew}_p$, and the optimization is carried over $A_i$ (more details in Appendix C). We use trivializations with SGD, SGD + momentum, Adam, and RMSProp. We use matrices of size $p = 100$, with a depth $D = 10$. The learning rate is $\eta = 0.5$, and the batch size is 256. Orthogonalization and test error are displayed in Figure 5. Orthogonalization error is not displayed for trivialization methods, since they are exact. The landing flow with momentum is the fastest method. It also leads to smaller orthogonalization error than the other retraction methods, because it does not suffer from accumulation of numerical errors (see subsection 3.6).

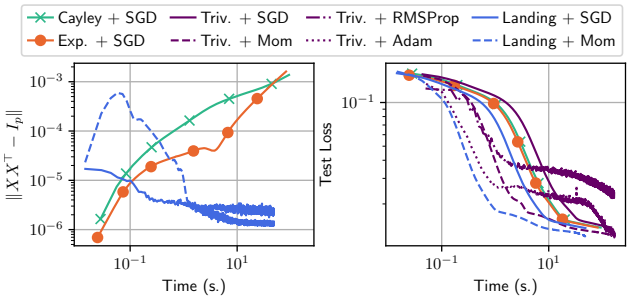


Figure 5: Training a fully connected neural network with orthogonal weights. Left: orthogonalization error of all weights. Right: test loss.

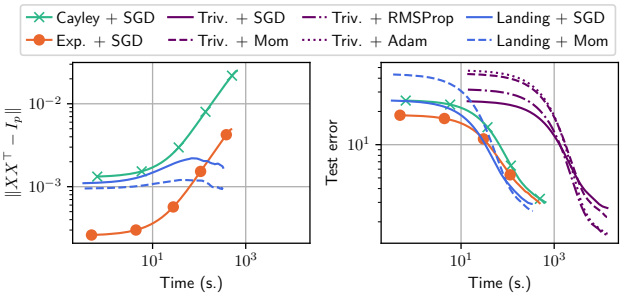


Figure 6: Training a LeNet5 on MNIST. Left: orthogonalization error. Right: Test Error.

**LeNet on MNIST** We train a LeNet5 (LeCun et al., 1998) for classification on the MNIST dataset. The network has 3 convolutional layers, and we impose an orthogonal constraint on the square kernel matrices. We take a batch size of 4, and for each algorithm, we take the learning rate that gives the fastest convergence in $4^i, i = -5 \dots 0$. We compare the same algorithms as before. Figure 6 displays the results of our experiment. Here, the landing algorithm is about 50% faster than retraction methods, and reaches a smaller orthogonalization error. While trivialization methods with advanced optimizers like RMSprop or Adam allow to reach the smallest test error, these methods are an order of magnitude slower than traditional methods in this case. The main reason is that they backprop through a matrix exponential at each iteration, which is costly (more details in Appendix).

**ResNet on CIFAR** In Fig. 7 we train a ResNet18 (He et al., 2016) on the CIFAR-10 dataset (Krizhevsky et al., 2009). Once again, we impose an orthogonality constraint on each convolution kernel. We take a batch size of 128, and use SGD with momentum to train each algorithm. The landing algorithm is here once again

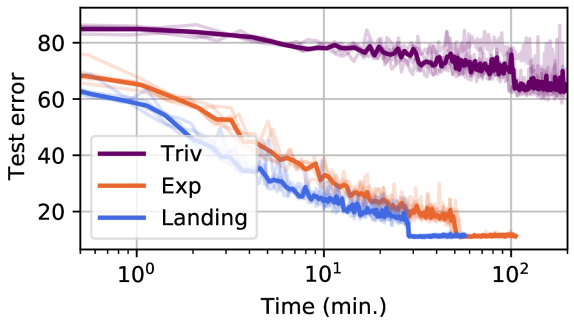


Figure 7: Training a ResNet 18 on CIFAR 10.

50% faster than retraction-based methods. Also, we notice in this case that trivialization methods fail to reach a high accuracy.

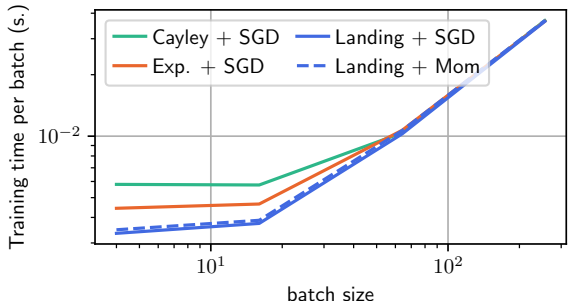


Figure 8: On MNIST with a LeNet5, median training time per batch, as a function of batch size.

**Diminishing returns** On the MNIST problem, we vary the batch size, and give the training time per batch in Fig 8. The landing method yields the greatest computational benefit when the batch size is small, because in this case, the computational bottleneck is computing the retraction. As the batch size increases, the computational cost is dominated by backpropagation, and we see a smaller gain with the landing algorithm. However, the other advantage of the landing algorithm – that it leads to small orthogonal error – remains.

**Discussion** We have presented a novel method to replace retraction-based algorithms on the orthogonal manifold. Our method is faster than retractions, it is therefore interesting to use in settings where computing a retraction is the computational bottleneck. It is also useful when one needs a solution that is accurately orthogonal, since it suffers less from numerical errors than widely used retractions. Future research directions include the development of second-order methods in this framework, and a thorough extension to the Stiefel manifold.

## Acknowledgement

This work was supported by the French government under management of Agence Nationale de la Recherche as part of the "Investissements d'avenir" program, reference ANR19-P3IA-0001 (PRAIRIE 3IA Institute). This work was supported in part by the European Research Council (ERC project NORIA).

## References

Ablin, P., Cardoso, J.-F., and Gramfort, A. (2018). Faster ICA under orthogonal constraint. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4464–4468. IEEE.

Absil, P.-A., Baker, C. G., and Gallivan, K. A. (2007). Trust-region methods on Riemannian manifolds. *Foundations of Computational Mathematics*, 7(3):303–330.

Absil, P.-A., Mahony, R., and Sepulchre, R. (2009). *Optimization algorithms on matrix manifolds*. Princeton University Press.

Absil, P.-A. and Malick, J. (2012). Projection-like retractions on matrix manifolds. *SIAM Journal on Optimization*, 22(1):135–158.

Arjovsky, M., Shah, A., and Bengio, Y. (2016). Unitary evolution recurrent neural networks. In *International Conference on Machine Learning*, pages 1120–1128.

Balashov, M., Polyak, B., and Tremba, A. (2020). Gradient projection and conditional gradient methods for constrained nonconvex minimization. *Numerical Functional Analysis and Optimization*, 41(7):822–849.

Bansal, N., Chen, X., and Wang, Z. (2018). Can we gain more from orthogonality regularizations in training deep networks? 31:4261–4271.

Bonnabel, S. (2013). Stochastic gradient descent on Riemannian manifolds. *IEEE Transactions on Automatic Control*, 58(9):2217–2229.

Boumal, N. (2020). An introduction to optimization on smooth manifolds. *Available online, May*.

Boyce, W. E., DiPrima, R. C., and Meade, D. B. (2017). *Elementary differential equations*. John Wiley & Sons.

Cardoso, J.-F. and Laheld, B. H. (1996). Equivariant adaptive source separation. *IEEE Transactions on signal processing*, 44(12):3017–3030.

Comon, P. (1994). Independent component analysis, a new concept? *Signal processing*, 36(3):287–314.

Edelman, A., Arias, T. A., and Smith, S. T. (1998). The geometry of algorithms with orthogonality constraints. *SIAM journal on Matrix Analysis and Applications*, 20(2):303–353.

Gao, B., Liu, X., and Yuan, Y.-x. (2019). Parallelizable algorithms for optimization problems with orthogonality constraints. *SIAM Journal on Scientific Computing*, 41(3):A1949–A1983.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., et al. (2020). Array programming with numpy. *Nature*, 585(7825):357–362.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Helfrich, K., Willmott, D., and Ye, Q. (2018). Orthogonal recurrent neural networks with scaled cayley transform. In *International Conference on Machine Learning*, pages 1969–1978. PMLR.

Hinton, G., Vinyals, O., and Dean, J. (2015). Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Karimi, H., Nutini, J., and Schmidt, M. (2016). Linear convergence of gradient and proximal-gradient methods under the polyak-łojasiewicz condition. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 795–811. Springer.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Krizhevsky, A., Hinton, G., et al. (2009). Learning multiple layers of features from tiny images.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Lee, J. M. (2013). Smooth manifolds. In *Introduction to Smooth Manifolds*, pages 1–31. Springer.

Lezcano Casado, M. (2019). Trivializations for gradient-based optimization on manifolds. *Advances in Neural Information Processing Systems*, 32:9157–9168.

Lezcano-Casado, M. and Martınez-Rubio, D. (2019). Cheap orthogonal constraints in neural networks: A simple parametrization of the orthogonal and unitary group. In *International Conference on Machine Learning*, pages 3794–3803. PMLR.

Li, J., Li, F., and Todorovic, S. (2020). Efficient riemannian optimization on the stiefel manifold via the cayley transform. In *International Conference on Learning Representations*.

Moler, C. and Van Loan, C. (2003). Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM review*, 45(1):3–49.

Nishimori, Y. (1999). Learning algorithm for independent component analysis by geodesic flows on orthogonal group. In *IJCNN'99. International Joint Conference on Neural Networks. Proceedings (Cat. No. 99CH36339)*, volume 2, pages 933–938. IEEE.

Oja, E. (1982). Simplified neuron model as a principal component analyzer. *Journal of mathematical biology*, 15(3):267–273.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. (2019). Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*.

Qi, C., Gallivan, K. A., and Absil, P.-A. (2010). Riemannian BFGS algorithm with applications. In *Recent advances in optimization and its applications in engineering*, pages 183–192. Springer.

Rodríguez, P., Gonzalez, J., Cucurull, G., Gonfaus, J. M., and Roca, X. (2016). Regularizing cnns with locally constrained decorrelations. *arXiv preprint arXiv:1611.01967*.

Schönemann, P. H. (1966). A generalized solution of the orthogonal procrustes problem. *Psychometrika*, 31(1):1–10.

Tripuraneni, N., Flammarion, N., Bach, F., and Jordan, M. I. (2018). Averaging stochastic gradient descent on Riemannian manifolds. *arXiv preprint arXiv:1802.09128*.

Xiao, N., Liu, X., and Yuan, Y. (2020a). Exact penalty function for l2, 1 norm minimization over the stiefel manifold. *SIAM J. Optim.*

Xiao, N., Liu, X., and Yuan, Y.-x. (2020b). A class of smooth exact penalty function methods for optimization problems with orthogonality constraints. *Optimization Methods and Software*, pages 1–37.

Xiao, N., Liu, X., and Yuan, Y.-x. (2021). A penalty-free infeasible approach for a class of nonsmooth opimtization problems over the stiefel manifold. *arXiv preprint arXiv:2103.03514*.

Yan, W.-Y., Helmke, U., and Moore, J. B. (1994). Global analysis of Oja's flow for neural networks. *IEEE Transactions on Neural Networks*, 5(5):674–683.

Zhang, H. and Sra, S. (2016). First-order methods for geodesically convex optimization. In *Conference on Learning Theory*, pages 1617–1638.

Zhang, H. and Sra, S. (2018). Towards Riemannian accelerated gradient methods. *arXiv preprint arXiv:1806.02812*.

# A Proofs

## A.1 Proof of Proposition 1

We recall the Euclidean Taylor expansion of $f$, where $\nabla f(X)$ is the gradient of $f$ at $X$ and $H_X$ the Hessian of $f$ at $X$:

$$f(X+E) = f(X) + \langle \nabla f(X), E\rangle + o(\|E\|), \tag{9}$$
$$\nabla f(X+E) = \nabla f(X) + H_X(E) + o(\|E\|). \tag{10}$$

The relative gradient $\psi(X)$ is such that

$$f(X+AX) = f(X) + \langle \psi(X), A\rangle + o(A).$$

Letting $E = AX$ in (9) gives, on the other hand

$$f(X+AX) = f(X) + \langle \nabla f(X), AX\rangle + o(A).$$

Identification of the first order term shows that for all $A \in \mathrm{Skew}_p$, it holds

$$\langle \psi(X), A\rangle = \langle \nabla f(X), AX\rangle \ ,$$

or by transposition:

$$\langle \psi(X) - \nabla f(X)X^\top, A\rangle = 0.$$

This scalar product cancels for all $A \in \mathrm{Skew}_p$, so the matrix on the left has to be in the orthogonal of $\mathrm{Skew}_p$, i.e. it is a symmetric matrix. In other words, its skew-symmetric part cancels. We therefore have

$$\mathrm{Skew}(\psi(X) - \nabla f(X)X^\top) = 0,$$

and since $\psi(X)$ is skew-symmetric, we find

$$\psi(X) = \mathrm{Skew}(\nabla f(X)X^\top).$$

For the relative Hessian, we have

$$\begin{aligned}
\psi(X+AX) &= \mathrm{Skew}\left(\nabla f(X+AX)(X+AX)^\top\right) \\
&= \mathrm{Skew}\left((\nabla f(X) + H_X(AX))(X+AX)^\top\right) + o(A) \\
&= \mathrm{Skew}\left(\nabla f(X)X^\top\right) + \mathrm{Skew}\left(H_X(AX)X^\top\right) + \mathrm{Skew}\left(\nabla f(X)X^\top A^\top\right) + o(A)
\end{aligned}$$

By identification of the first order term, we find

$$\mathcal{H}_X(A) = \mathrm{Skew}\left(H_X(AX)X^\top - \nabla f(X)X^\top A\right).$$

## A.2 Proof of Proposition 2

The Riemannian gradient, $\mathrm{Grad}\, f(X)$, and Hessian, $\mathrm{Hess}\, f(X)$, are such that for $\xi \in \mathcal{T}_x$, it holds

$$f(\mathcal{R}(X,\xi)) = f(X) + \langle \mathrm{Grad}\, f(X), \xi\rangle + \frac{1}{2}\langle \xi, \mathrm{Hess}\, f(X)[\xi]\rangle + o(\|\xi\|^2),$$

where $\mathcal{R}(X,\xi)$ is the exponential retraction: $\mathcal{R}(X,\xi) = \exp(\xi X^\top)X$. We find using the same method as above:

$$\mathrm{Grad}\, f(X) = \mathrm{Skew}(\nabla f(X)X^\top)X$$
$$\mathrm{Hess}\, f(X)[\xi] = \mathrm{Skew}\left(H_X(\xi)X^\top - \mathrm{Sym}(\nabla f(X)X^\top A)\right)X.$$

This gives the expected identities.

### A.3 Proof of Proposition 3

By contradiction, such polynomial must satisfy:

- Orthogonality: for all $X \in \mathcal{O}_p$ and $A \in \mathrm{Skew}_p$, $P(X, A)P(X, A)^\top = I_p$.
- Retraction: for all $X \in \mathcal{O}_p$ and $A \in \mathrm{Skew}_p$, $P(X, A) = X + AX + o(A)$.

The first equality shows that the polynomial $PP^\top$ is a constant polynomial, it is therefore a polynomial of degree 0. Since the degree of $PP^\top$ is greater than the degree of $P$, $P$ must also be a constant polynomial. This contradicts the second equality.

### A.4 Proof of Proposition 4

If $\Lambda(X) = 0$, since $X$ is invertible, we have $\psi(X) + \lambda(XX^\top - I_p) = 0$. This is the sum of two matrices, one skew-symmetric, the other symmetric, which is zero. Therefore, both matrices are 0, and we deduce $\psi(X) = 0$ and $XX^\top = I_p$.

Conversely, if $X \in \mathcal{O}_p$ and $\psi(X) = 0$, we have $\Lambda(X) = 0$.

### A.5 Proof of Proposition 5

By differentiation, denoting $n(t) = \mathcal{N}(X(t))$, we find

$$\dot{n}(t) = -\langle \Lambda(X(t)), \nabla\mathcal{N}(X(t))\rangle \tag{11}$$
$$= -\langle \psi(X), (XX^\top - I)XX^\top\rangle - \lambda\|(XX^\top - I_p)X\|^2. \tag{12}$$

The first term cancels, since $\psi$ is skew-symmetric and $(XX^\top - I)XX^\top$ is symmetric. Therefore, we obtain

$$\dot{n}(t) = -\lambda\|(X(t)X(t)^\top - I_p)X(t)\|^2 \tag{13}$$

Now, we would like to upper-bound this by a quantity involving only $n(t) = \|X(t)X(t)^\top - I_p\|^2$.

Dropping the time $(t)$ for now, and letting $\Delta = XX^\top - I_p$, we find

$$\|(XX^\top - I_p)X\|^2 = \mathrm{Tr}(\Delta XX^\top\Delta) \tag{14}$$
$$= \mathrm{Tr}(\Delta(I_p + \Delta)\Delta) \tag{15}$$
$$= \|\Delta\|^2 + \mathrm{Tr}(\Delta^3) \tag{16}$$

Next, we need to control $\mathrm{Tr}(\Delta^3)$. Denoting $\lambda_1, \dots, \lambda_p$ the eigenvalues of $\Delta$, we have $\mathrm{Tr}(\Delta^3) = \sum_{i=1}^p \lambda_i^3$. This is lower bounded by $-\sum_{i=1}^p |\lambda_i|^3$, and then using the non-increasing property of $\ell_p$ norms, we have

$$\left(\sum_{i=1}^p |\lambda_i|^3\right)^{\frac{1}{3}} \leq \left(\sum_{i=1}^p |\lambda_i|^2\right)^{\frac{1}{2}},$$

and gathering all inequalities together, we have:

$$\mathrm{Tr}(\Delta^3) = \sum_{i=1}^p \lambda_i^3 \geq -\sum_{i=1}^p |\lambda_i|^3 \geq -\left(\sum_{i=1}^p |\lambda_i|^2\right)^{\frac{3}{2}} = -\|\Delta\|^3.$$

Finally, using $\mathcal{N}(X) = \|\Delta\|^2$, we find that Equation 16 gives the bound

$$\|(XX^\top - I_p)X\|^2 \geq \mathcal{N}(X) - \mathcal{N}(X)^{\frac{3}{2}}$$

We therefore obtain the differential inequation in Equation 13:

$$\dot{n}(t) \leq \lambda(n(t) - n(t)^{\frac{3}{2}}).$$

Dividing by the right hand side, we get

$$\frac{\dot{n}(t)}{n(t) - n(t)^{\frac{3}{2}}} \leq -\lambda.$$

And by integration, using the fact that $\gamma : x \mapsto \log(x) - 2\log(1 - \sqrt{x})$ is a primitive of $x \mapsto \frac{1}{x - x^{\frac{3}{2}}}$, it holds:

$$\gamma(n(t)) - \gamma(N_0) \leq -\lambda t$$

which overall gives the bound

$$n(t) \leq \gamma^{-1}\left(-\lambda t + \gamma(N_0)\right)$$

where the inverse of $\gamma$ is $\gamma^{-1}(x) = \frac{1}{(\exp(-\frac{x}{2})+1)^2}$. We get:

$$n(t) \leq \frac{1}{(\exp(\frac{\lambda t - \gamma(N_0)}{2}) + 1)^2} \tag{17}$$

$$\leq \exp(-\lambda t)\frac{1}{(\exp(-\frac{\lambda t}{2}) + \exp(-\frac{\gamma(N_0)}{2}))} \tag{18}$$

The fraction on the right is then upper-bounded by $\exp(\frac{\gamma(N_0)}{2}) = \frac{N_0}{\sqrt{1-N_0^2}}$, which gives the advertised result.

### A.6 Proof of Proposition 6

Let $X \in \mathbb{R}^{p\times p}$, and define $\Delta = XX^\top - I_p$ and $A = \psi(X)$. The landing algorithm maps $X$ to $\tilde{X} = (I_p - \eta(A + \lambda\Delta))X$. Defining $\tilde{\Delta} = \tilde{X}\tilde{X}^\top - I_p$, we find

$$\tilde{\Delta} = (I_p - \eta(A + \lambda\Delta))(\Delta + I_p)(I_p - \eta(-A + \lambda\Delta)) - I_p \tag{19}$$

$$= (1 - 2\eta\lambda)\Delta + (\eta - \eta^2\lambda)[A, \Delta] - (2\eta\lambda - \eta^2\lambda^2)\Delta^2 - \eta^2 A^2 + \eta^2\lambda^2\Delta^3 + \lambda\eta^2[A, \Delta^2] - \eta^2 A\Delta A, \tag{20}$$

where $[A, \Delta] = A\Delta - \Delta A$ is the Lie bracket.

Using the sub-multiplicativity of the norm, and the triangular inequality, denoting $a = \|A\|$ and $d = \|\Delta\|$, we get

$$\tilde{d} \leq (1 - 2\eta\lambda)d + 2(\eta - \eta^2\lambda)ad + (2\eta\lambda - \eta^2\lambda^2)d^2 + \eta^2 a^2 + \eta^2\lambda^2 d^3 + 2\lambda\eta^2 ad^2 + \eta^2 a^2 d \tag{21}$$

$$\leq (1 - 2\eta\lambda)d + 2\eta ad + 2\eta\lambda d^2 + \eta^2 a^2 + \eta^2\lambda^2 d^3 + 2\lambda\eta^2 ad^2 + \eta^2 a^2 d \tag{22}$$

Reordering terms in ascending powers of $\eta$, we find $\tilde{d} \leq d - \alpha\eta + \beta\eta^2$ with

$$\alpha = 2\lambda d - 2ad - 2\lambda d^2$$

$$\beta = a^2 + \lambda^2 d^3 + 2\lambda ad^2 + a^2 d > 0$$

Therefore, we find that when $\eta \leq \eta^*(\alpha, \beta) = \frac{\alpha + \sqrt{\alpha^2 + 4\beta(\varepsilon - d)}}{2\beta}$, we have $\tilde{d} \leq \varepsilon$.

### A.7 First order expansion of the landing field

In order to study the local convergence of the algorithm, we develop the landing field to the first order.

**Proposition 12** (First order expansion of $\Lambda$)**.** *At the first order in $(A, S)$, we have $\Lambda((I_p + A + S)X_*) = \mathcal{J}(A, S)X_*$, with $\mathcal{J}(A, S) \triangleq \mathcal{H}_{X_*}(A) + \mathcal{H}^{\mathrm{Sym}}_{X_*}(S) + \lambda S$ , where $\mathcal{H}_{X_*}$ is the Relative Hessian and $\mathcal{H}^{\mathrm{Sym}}_{X_*}$ is a linear operator from $\mathrm{Sym}_p$ to $\mathrm{Skew}_p$.*

*Proof.* We recall that

$$\Lambda(X) = \left(\psi(X) + \lambda(XX^\top - I_p)\right) X$$

Letting $X = (I_p + S + A)X_*$, we have at the first order

$$\psi(X) = \mathcal{H}_{X_*}(A) + \mathcal{H}_*^{Sym}(S)$$

and

$$XX^\top - I_p = S$$

which overall gives

$$\Lambda(X) = \mathcal{J}(A, S)X_*$$

with $\mathcal{J}(A, S) = \mathcal{H}_{X_*}(A) + \mathcal{H}_*^{Sym}(S) + \lambda S$.

□

The linear operator $\mathcal{J}$ can be conveniently written in the basis $(\mathrm{Skew}_p, \mathrm{Sym}_p)$ where it is block-diagonal since for all $A \in \mathrm{Skew}_p$, we have $\mathrm{Sym}\left(\mathcal{J}(A, 0)\right) = 0$. The operator is written in this basis

$$\mathcal{J} = \begin{bmatrix} \mathcal{H}_{X_*} & \mathcal{H}_{X_*}^{\mathrm{Sym}} \\ 0 & \lambda Id \end{bmatrix} .$$

As a consequence, the eigenvalues of $\mathcal{J}$ are the eigenvalues of $\mathcal{H}_{X_*}$ and $\lambda$: $\mathrm{Sp}(\mathcal{J}) = \mathrm{Sp}(\mathcal{H}_{X_*}) \cup \{\lambda\}$.

### A.8 Proof of Proposition 7

First, it is easily seen that since $\mathcal{J}$ is invertible, then the system $\dot{X} = -\Lambda(X)$ is an "almost linear" system (Boyce et al., 2017, Ch.9.3), and the eigenvalues of $\mathcal{J}$ are all non-negative and real, which shows that $X_*$ is asymptotically stable: therefore, there exists $\delta > 0$ such that the flow, initialized from any $X$ such that $\|X - X_*\| \leq \delta$, converges to $X_*$. Classical manipulations then give us the advertised convergence speed. [1]

### A.9 Proof of Proposition 8

The landing flow with step $\eta = \frac{1}{L}$ iterates $X_{k+1} = \Phi(X_k)$ with $\Phi(X) = X - \frac{1}{L}\Lambda(X)$. The Jacobian of this map is $Id - \frac{1}{L}\mathcal{J}(\Lambda)(X)$, where $\mathcal{J}(\Lambda)$ is the Jacobian of $\Lambda$. The eigenvalues of this map are the $1 - \frac{\mu}{L}$, where $\mu$ spans the eigenvalues of $\mathcal{J}(\Lambda)(X)$.

Since the eigenvalue of $\mathcal{J}(\Lambda)(X)$ are all real positive at $X^*$, there is a neighborhood of $X^*$ such that in that neighborhood, the eigenvalues of $\mathcal{J}(\Lambda)(X)$ are close to real positive: for $\delta > 0$, there is a neighborhood of $X^*$ such that for $\mu$ an eigenvalue of $\mathcal{J}(\Lambda)(X)$, we have $Re(\mu) > \min(\mu_{min}, \lambda)$ and $|Im(\mu)| \leq \delta$. Further, thanks to the Lipschitz assumption, we have $|\mu| \leq L$.

As a consequence, the eigenvalues of $\Phi$, in this neighborhood, are of modulus squared:

$$|1 - \frac{\mu}{L}|^2 = (1 - \frac{Re(\mu)}{L})^2 + \eta^2 Im(\mu)^2 \tag{23}$$

$$\leq (1 - \frac{\min(\mu_{min}, \lambda)}{L})^2 + \eta^2\delta^2 \tag{24}$$

Hence, the iterative scheme $X_{k+1} = \Phi(X_k)$ converges at the speed $O((1 - \frac{\min(\mu_{min},\lambda)}{L})^k)$.

[1] See for instance corollary 4.23 of "Chicone, Carmen. Ordinary differential equations with applications. Vol. 34. Springer Science & Business Media, 2006."

### A.10 Proof of Proposition 9

We have

$$[f(X(t))]' = \langle \Lambda(X), \nabla f(X) \rangle \tag{25}$$
$$= -\langle \mathrm{Skew}(\nabla f(X) X^\top) X, \nabla f(X) \rangle - \lambda \langle (XX^\top - I_p) X, \nabla f(X) \rangle \tag{26}$$
$$= -\|\psi(X)\|^2 - \lambda \langle XX^\top - I_p, \mathrm{Sym}(\nabla f(X) X^\top) \rangle \tag{27}$$

Therefore, using the majorization of $\mathrm{Sym}(\nabla f(X) X^\top)$, the upper bound on $\|XX^\top - I_p\|$ and Cauchy-Schwarz, we find

$$\|\psi(X(t))\|^2 \leq -[f(X(t))]' + \lambda \exp(-\frac{\lambda}{2} t) \frac{\sqrt{N_0}}{\sqrt{N_0}+1} K$$

Then, by integration between $t = 0$ and $T$, it holds

$$\int_0^T \|\psi(X(t))\|^2 dt \leq f(X_0) - f(X(T)) + 2\frac{\sqrt{N_0}}{\sqrt{N_0}+1} K \leq f(X_0) - f^* + 2\frac{\sqrt{N_0}}{\sqrt{N_0}+1} K$$

Finally, we use

$$\inf_{t \leq T} \|\psi(X(t))\| \leq \left( \frac{1}{T} \int_0^T \|\psi(X(t))\|^2 dt \right)^{\frac{1}{2}}$$

to obtain the advertised result.

### A.11 Proof of Proposition 10

We assume that we follow the safe rule, so that we are close to the manifold. We let $\alpha > 0$ such that for all iterates, $\|X_k(X_k^\top X_k - I_p)\|^2 \leq \alpha \mathcal{N}(X_k)$ and such that the Hessian of $\mathcal{N}$ is bounded by $\alpha$. We also let $\beta$ such that $\|X_k(X_k^\top X_k - I_p)\|^2 \geq \beta \mathcal{N}(X_k)$.

We will use the following result extensively:

$$\|\Lambda(X_k)\|^2 = \|\psi(X_k)\|^2 + \lambda \|X_k(X_k^\top X_k - I_p)\|^2 \leq \|\psi(X_k)\|^2 + \alpha\lambda \mathcal{N}(X_k)$$

Then, we look at the decrease towards the manifold:

$$\mathcal{N}(X_{k+1}) \leq \mathcal{N}(X_k) - \eta \langle \Lambda(X_k), \nabla \mathcal{N}(X_k) \rangle + \alpha \eta^2 \|\Lambda(X_k)\|^2 \tag{28}$$
$$\leq (1 - \eta\beta) \mathcal{N}(X_k) + \alpha \eta^2 (\|\psi(X_k)\|^2 + \alpha\lambda \mathcal{N}(X_k)) \tag{29}$$
$$= (1 - \eta\beta + \eta^2\alpha^2) \mathcal{N}(X_k) + \alpha\eta^2 \|\psi(X_k)\|^2 \tag{30}$$

Next, we turn to the study of the decrease. Letting $L > 0$ the Lipschitz constant of $f$, we have

$$f(X_{k+1}) \leq f(X_k) - \eta \langle \Lambda(X_k), \nabla f(X_k) \rangle + \frac{1}{2}\eta^2 L \|\Lambda(X_k)\|^2 \tag{31}$$
$$\leq f(X_k) - \eta(\|\psi(X_k)\|^2 + \lambda \langle X_k X_k^\top - I_p, \nabla f(X_k) X_k^\top \rangle) + \frac{1}{2}\eta^2 L (\|\psi(X_k)\|^2 + \alpha\lambda \mathcal{N}(X_k)) \tag{32}$$

Isolating the terms in $\psi(X_k)$, for $\eta \leq \frac{1}{L}$, we find

$$\|\psi(X_k)\|^2 \leq \frac{2}{\eta} \left( f(X_k) - f(X_{k+1}) - \eta\lambda \langle X_k X_k^\top - I_p, \nabla f(X_k) X_k^\top \rangle + \frac{1}{2}\eta^2 \lambda \alpha L \mathcal{N}(X_k) \right)$$

We let $F$ an upper bound of $\frac{f(X_k) - f(X_{k+1})}{\eta}$, and $G$ and upper bound of $\|\nabla f(X_k) X_k^\top\|$ (these quantities exist by compacity since $X_k$ belong to a compact set).

Then, using Cauchy-Schwarz

$$\|\psi(X_k)\|^2 \leq 2F + 2\lambda\sqrt{\mathcal{N}(X_k)}G + \eta\lambda\alpha L\mathcal{N}(X_k) \tag{33}$$

Pluging this in Eq. (30), we get the inequality

$$\mathcal{N}(X_{k+1}) \leq (1 - \eta\beta + \alpha\eta^2 + \alpha^2\eta^3\lambda L)\mathcal{N}(X_k) + 2\lambda G\alpha\eta^2\sqrt{\mathcal{N}(X_k)} + 2\alpha F\eta^2$$

To conclude, we majorize

$$\sqrt{\mathcal{N}(X_k)} \leq \frac{1}{2}\mathcal{N}(X_k) + \frac{1}{2}$$

to obtain

$$\mathcal{N}(X_{k+1}) \leq (1 - \eta\beta + \alpha\eta^2 + \lambda G\alpha\eta^2 + \alpha^2\eta^3\lambda L)\mathcal{N}(X_k) + (2\alpha F + \lambda\alpha G)\eta^2$$

For $\eta$ small enough, we get

$$\mathcal{N}(X_{k+1}) \leq (1 - \frac{1}{2}\eta\beta)\mathcal{N}(X_k) + (2\alpha F + \lambda\alpha G)\eta^2 \tag{34}$$

which gives, starting from $\mathcal{N}(X_0) = 0$,

$$\mathcal{N}(X_k) \leq \eta\frac{2(2\alpha F + \lambda\alpha G)}{\beta}$$

This shows that $\mathcal{N}(X_k)$ is at most $\propto \eta$. In the following, for short, we let $\gamma = \frac{2(2\alpha F + \lambda\alpha G)}{\beta}$.

We now use once again (32):

$$\|\psi(X_k)\|^2 \leq \frac{2}{\eta}\left(f(X_k) - f(X_{k+1}) + \eta^{\frac{3}{2}}\lambda G\sqrt{\gamma} + \frac{1}{2}\eta^3\lambda\alpha L\gamma\right)$$

By averaging up to an integer $K$, we find

$$\frac{1}{K}\sum_{k=1}^{K}\|\psi(X_k)\|^2 \leq \frac{2}{\eta}\left(\frac{f(X_0) - f^*}{K} + \eta^{\frac{3}{2}}\lambda G\sqrt{\gamma} + \frac{1}{2}\eta^3\lambda\alpha L\gamma\right)$$

which gives as advertised

$$\inf_{k\geq 0}\|\psi(X_k)\|^2 \leq 2\sqrt{\eta}\lambda G\sqrt{\gamma} + \eta^2\lambda\alpha L\gamma \ .$$

### A.12 Proof of Proposition 11

We now assume that the step-size $\eta$ depends on the iterate $k$ with $\eta_k \propto \frac{1}{k^\alpha}$.

The iterates verify the same inequality (34):

$$\mathcal{N}(X_{k+1}) \leq (1 - \frac{1}{2}\eta_k\beta)\mathcal{N}(X_k) + (2\alpha F + \lambda\alpha G)\eta_k^2$$

When $\eta_k \propto \frac{1}{k^\alpha}$ with $\alpha < 1$, unrolling this inequality gives

$$\mathcal{N}(X_k) = \mathcal{O}(\frac{1}{k^\alpha}).$$

This shows that the iterates converge towards the manifold.

Next, we use once again(32):

$$\eta_k\|\psi(X_k)\|^2 \leq 2\left(f(X_k) - f(X_{k+1}) + \eta_k^{\frac{3}{2}}\lambda G\sqrt{\gamma} + \frac{1}{2}\eta_k^3\lambda\alpha L\gamma\right)$$

Since $\eta_k$ goes to 0, we have $\eta_k^3 = o(\eta_k^{\frac{3}{2}})$. Therefore, for $k$ large enough, we have

$$\eta_k \|\psi(X_k)\|^2 \leq 2\left(f(X_k) - f(X_{k+1}) + 2\eta_k^{\frac{3}{2}} \lambda G \sqrt{\gamma}\right)$$

Summing these inequalities up to an integer $K$ gives

$$\sum_{k=1}^{K} \eta_k \|\psi(X_k)\|^2 \leq 2\left(f(X_0) - f(X_K) + 2\lambda G\sqrt{\gamma} \sum_{k=1}^{K} \eta_k^{\frac{3}{2}}\right)$$

We now have two cases.

**First case:** When $\alpha > \frac{2}{3}$, then $\sum_{k=1}^{K} \eta_k^{\frac{3}{2}}$ is bounded as $K$ increases, hence $\sum_{k=1}^{K} \eta_k \|\psi(X_k)\|^2$ is bounded. We then have

$$\inf_{k \leq K} \|\psi(X_k)\|^2 \leq \frac{\sum_{k=1}^{K} \eta_k \|\psi(X_k)\|^2}{\sum_{k=1}^{K} \eta_k} = \mathcal{O}(\frac{1}{\sum_{k=1}^{K} \eta_k}) = \mathcal{O}(\frac{1}{K^{1-\alpha}}).$$

**Second case:** When $\alpha \leq \frac{2}{3}$, then $\sum_{k=1}^{K} \eta_k^{\frac{3}{2}}$ is of the order of $K^{1-\frac{3}{2}\alpha}$, and we find:

$$\inf_{k \leq K} \|\psi(X_k)\|^2 \leq \frac{\sum_{k=1}^{K} \eta_k \|\psi(X_k)\|^2}{\sum_{k=1}^{K} \eta_k} = \mathcal{O}(\frac{K^{1-\frac{3}{2}\alpha}}{K^{1-\alpha}}) = \mathcal{O}(\frac{1}{K^{\frac{\alpha}{2}}})$$

These two results can be compactly rewritten as

$$\inf_{k \leq K} \|\psi(X_k)\|^2 = \mathcal{O}(K^{-\min(\frac{\alpha}{2}, 1-\frac{\alpha}{2})})$$

### A.13 Global convergence of the momentum method in the continuous case

We consider the momentum extension of the landing flow

$$\dot{A}(t) = -A(t) + \psi(X(t)) \tag{35}$$
$$\dot{X}(t) = -(A(t) + \lambda(X(t)X(t)^\top - I_p))X(t) \tag{36}$$

We consider the energy

$$E(t) = f(X(t)) + \frac{1}{2}\|A(t)\|^2$$

and find

$$E'(t) = \langle \dot{X}(t), \nabla f(X(t)) \rangle + \langle \dot{A}(t), A(t) \rangle \tag{37}$$
$$= -\lambda \langle X(t)X(t)^\top - I_p, \nabla f(X(t))X(t)^\top \rangle - \|A(t)\|^2 \tag{38}$$

As a consequence, it holds

$$\|A(t)\|^2 \leq E'(t) + \sqrt{\mathcal{N}(X(t))}K$$

where $K$ bounds $\|\nabla f(X(t))X(t)^\top\|$, and $\|A(t)\|^2$ is integrable.

Next, we let $F(t) = f(X(t))$, and $\Delta(t) = -\lambda\langle X(t)X(t)^\top - I_p, \nabla f(X(t))X(t)^\top\rangle$. We have

$$F'(t) = -\langle A(t), \psi(X(t)) \rangle + \Delta(t)$$

$$F''(t) = -\|\psi(X(t))\|^2 + \langle A(t), \psi(X(t)) \rangle - \langle A(t), \mathcal{H}_{X(t)}(A(t)) \rangle + \Delta'(t)$$

so that

$$F''(t) + F'(t) = -\|\psi(X(t))\|^2 - \langle A(t), \mathcal{H}_{X(t)}(A(t)) \rangle + \Delta(t) + \Delta'(t)$$

and we have

$$\|\psi(X(t))\|^2 \leq -F''(t) - F'(t) + \Delta(t) + \Delta'(t),$$

so that $\|\psi(X(t))\|^2$ is integrable. This implies $\liminf \|\psi(X(t))\|^2 = 0$.

## B Comparison with the proximal linearized augmented Lagrangian algorithm method

Gao et al. (2019) propose the proximal linearized augmented Lagrangian algorithm (PLAM) method. Like the landing algorithm, it is an infeasible method, which in spirit is very close to our method.

Instead of the landing field

$$\Lambda_{landing}(X) = \mathrm{Skew}(\nabla f(X)X^\top)X + \lambda(XX^\top - I_p)X$$

the authors consider

$$\Lambda_{plam}(X) = \nabla f(X) - \mathrm{Sym}(\nabla f(X)X^\top)X + \lambda(XX^\top - I_p)X$$

When $X$ is orthogonal, both methods are similar and we have

$$\text{For } X \in \mathcal{O}_p,\ \Lambda_{landing}(X) = \Lambda_{plam}(X) = \mathrm{Grad}\, f(X)$$

However, these fields differ when $X \notin \mathcal{O}_p$. In theory, (Gao et al., 2019) provides a global and local convergence proof. However, the proof requires that the gradient of the function is not too large. On the other hand, we prove global convergence of the landing algorithm under a very mild Lipschitz assumption on $f$.

In the following, we provide some theoretical and practical arguments to argue that $\Lambda_{landing}(X)$ is far more robust to the choice of the hyper-parameter $\lambda$, and that in some settings, using $\Lambda_{plam}(X)$ can lead to highly instable behavior, while the landing algorithm behaves nicely.

The first argument is that we cannot have a proposition similar to Prop. 4 for the PLAM field. Indeed, we have

**Proposition 13.** *Let $X \in \mathbb{R}^{p\times p}$ such that $\nabla f(X) = SX$ with $S$ a symetric matrix. Let $\Delta = XX^\top - I_p$. If $S\Delta + \Delta S = 2\lambda S$, then $\Lambda_{plam}(X) = 0$*

*Proof.* In this case, we have

$$\begin{aligned}
\Lambda_{plam}(X) &= SX - \frac{1}{2}(SXX^\top + XX^\top S)X + \lambda(XX^\top - I_p)X && (39)\\
&= (S - \frac{1}{2}(S\Delta + \Delta S + 2S) + \lambda\Delta)X && (40)\\
&= (-\frac{1}{2}(S\Delta + \Delta S) + \lambda\Delta)X && (41)\\
&= 0. && (42)
\end{aligned}$$

□

Therefore, if there is a matrix $X$ such that $\nabla f(X) = \lambda X$, then we automatically have $\Lambda_{plam}(X) = 0$: this means that there might be some spurious critical points of $\Lambda_{plam}$. On the other hand, as demonstrated in Prop. 4, the landing field does not have such problem. In practice, to circumvent this problem, PLAM must assume that the function does not have a too large gradient norm (Gao et al., 2019, Lemma 2.5).

Similarly, we cannot have an orthogonalization property like Prop. 5. It is in fact easy to exhibit problems where the continuous ODE following the PLAM field

$$\dot{X} = -\Lambda_{plam}(X)$$

explodes in finite time:

**Proposition 14.** *We fix $p = 2$. Let $\alpha > 0$ and consider $f(X) = \|\alpha X - B\|^2$ with $B = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$. Assume that the flow $\dot{X} = -\Lambda_{plam}(X)$ starts from $X_0 = \begin{bmatrix} 1 & 0 \\ 0 & 1+\delta \end{bmatrix}$. Then, $X(t)$ is of the form $\begin{bmatrix} 1 & 0 \\ 0 & 1+\delta(t) \end{bmatrix}$ with $\delta(0) = \delta$. If $\alpha^2 = \beta$, then $\delta(t) = \delta$ for all $t$. If $\alpha^2 < \beta$, then $\delta(t)$ goes to infinity in a finite time if $\delta(0) > 0$. If $\delta(0) < 0$, $\delta(t)$ goes to $-1$. If $\alpha^2 > \beta$, then $\delta(t)$ goes to $0$.*

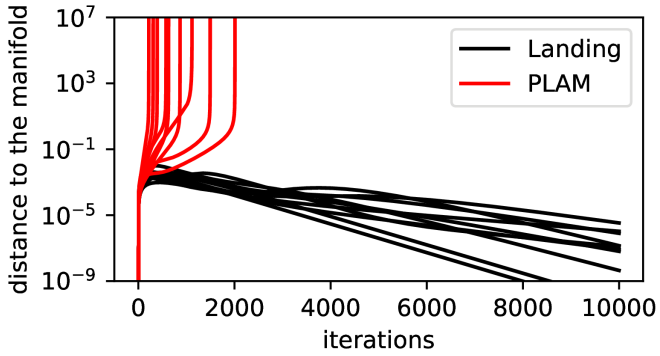


Figure 9: PLAM can explode easily even on simple problems, while the landing algorithm is robust

*Proof.* It is easy to see that for this problem, when $X = \begin{bmatrix} 1 & 0 \\ 0 & 1+\delta \end{bmatrix}$, it holds

$$\Lambda_{plam}(X) = \begin{bmatrix} 0 & 0 \\ 0 & (\alpha^2 - \beta)(\delta - 1 - (\delta-1)^3) \end{bmatrix}$$

This shows that $X(t)$ is of the form $\begin{bmatrix} 1 & 0 \\ 0 & 1+\delta(t) \end{bmatrix}$ for all $t$, with

$$\delta'(t) = -(\alpha^2 - \beta)(\delta - 1 - (\delta-1)^3)$$

The behavior of $\delta(t)$ is then concluded from an elementary study of the previous ODE. □

In particular, in the setting where $\alpha^2 > \beta$, the algorithm can start arbitrarily close to the manifold, and still explode in finite time. In conclusion, global convergence of the continuous flow associated with $\Lambda_{plam}$ does not hold for the previous function. Thanks to Prop.5, such bad behavior cannot happen for the landing flow.

Note that the discrete algorithm also explodes with the previous problem. This is not just a theoretical concern: the following experiment shows that such bad behavior happens in practice.

We conduct the following experiment. For $p = 2$, we generate $A, B \in \mathbb{R}^{p\times p}$ with normal i.i.d. entries, and apply both the landing algorithm and PLAM to the minimization of $\|AX - B\|^2$ starting from $X_0 = I_p$. We use $\lambda = 1$ for both algorithms, and a very small step size $\eta = 10^{-3}$. We generate 10 different such problems. In Fig. 9 we display the distance to the manifold $\|XX^\top - I_p\|$ for the 10 trajectories: PLAM always explodes, while the landing algorithm always succeeds.

As a consequence, the landing algorithm seems more robust than PLAM.

## C Notes on trivializations

In order to solve the optimization problem $\min_{\mathcal{O}_p} f(X)$, trivializations use a reparametrization of $\mathcal{O}_p$ with a linear space. For instance, we can recast the previous problem as

$$\min_{A\in \mathrm{Skew}_p} f(\exp(A))$$

This formulation has several advantages (Lezcano-Casado and Martınez-Rubio, 2019):

- The optimization problem is now on an Euclidean space, hence is it easy to use Euclidean algorithms like quasi-Newton methods (for instance, L-BFGS), or in the context of deep learning, momentum methods, Adam (Kingma and Ba, 2014), or RMSProp.
- It is also easy to implement: in a deep learning framework, the layer is parametrized with a simple skew-symmetric matrix, and there is no need to be careful with the optimizer.

It also has several drawbacks

- It may severely change the optimization landscape. For instance, if we consider an orthogonal procrustes problem where $f(X) = \langle M, X\rangle$, the new cost function becomes $\langle M, \exp(A)\rangle$, which has a more complicated structure.To alleviate this problem, it has been proposed to periodically change the foot of the trivialization space during optimization (Lezcano Casado, 2019).
- It adds a computational cost which can be significant. Indeed, in a deep learning setting, one needs to differentiate through the exp function. This is done by computing the exp of a matrix of size $2p \times 2p$. When $p$ is large, this cost may very well be prohibitive, since it is about 8 times as costly as computing the exp of a $p \times p$ matrix. For instance, using the Pytorch implementation of the matrix exponential, on a single laptop CPU, it takes 100ms to compute the exp of a $1000 \times 1000$ matrix, while it takes 800ms to compute the exponential of a $2000 \times 2000$ matrix.

# D Experiments details

## D.1 Distillation experiment

The network is a fully-connected multilayer perceptron of depth $L$ that maps the input $x_0$ to the output $x_L$ by the iteration $x_{n+1} = \sigma(W_{n+1}x_n + b_{n+1})$, where $W_n \in \mathcal{O}_p$ and $b_n \in \mathbb{R}^p$, and $\sigma$ is the tanh function. We denote $\Phi_\theta(x)$ the output of the network with input $x \in \mathbb{R}^p$ and parameters $\theta = (W_1, b_1, \dots, W_L, b_L)$. We set $p = 100$, and $L = 10$. We choose a random set of parameters $\theta^* = (W_1^*, b_1^*, \dots, W_L^*, b_L^*)$ as the teacher network, and starting from a new random initialization, we try to approximate this network.

We let $\theta$ the new parameters, and minimize the loss

$$\mathbb{E}_{x\sim d}[\|\Phi_\theta(x) - \Phi_{\theta^*}(x)\|^2]$$

where the density $d$ is $\mathcal{N}(0, 1)$.

This is done with stochastic gradient descent, using a retraction for orthogonal parameters. For each method, the learning rate the one that yields the fastest convergence in $\{1, 0.1, 0.01, 0.001\}$, and we perform 10000 iterations with a batch size of 256.

## D.2 MNIST experiment

We consider a standard LeNet network which consists of three convolutional layers: the first one maps one channel to 6 with a kernel of size $5 \times 5$, the second maps the 6 channels to 16 with a kernel size of $5 \times 5$, and the last one maps the 16 channels to 120 channels with a kernel size of $4 \times 4$. A last linear layer is used to obtain an output in dimension 10. The kernel are assumed to be orthogonal. We use a batch size of 4. For each method, the learning rate the one that yields the fastest convergence in $\{1, 0.1, 0.01, 0.001\}$.

## D.3 CIFAR-10 experiment

We use a ResNet18 architecture as described in (He et al., 2016). Since there are only 10 classes in CIFAR-10, we replace the last fully-connected layer so that the output is a vector of size 10. The rest of the architecture is left unchanged.

We impose orthogonality on all kernels of the network, using the landing method, the exponential retraction, and trivializations. We only compare to the exponential retraction, since in Pytorch it is the fastest retraction.

For all non-orthogonal parameters (biases, and fully connected last layer), we use SGD + momentum and weight decay, with a learning rate of 0.1, a momentum of 0.9 and weight decay of $5 \times 10^{-4}$.

For the orthogonal parameters, we use SGD + momentum with a learning rate in the grid $\{2, 1, \frac{1}{2}\}$. Larger learning rates lead to instabilities, and smaller learning rates lead to slow convergence. The momentum is once again taken as 0.9.

After 100 epochs, the learning rate for all parameters is divided by 10.

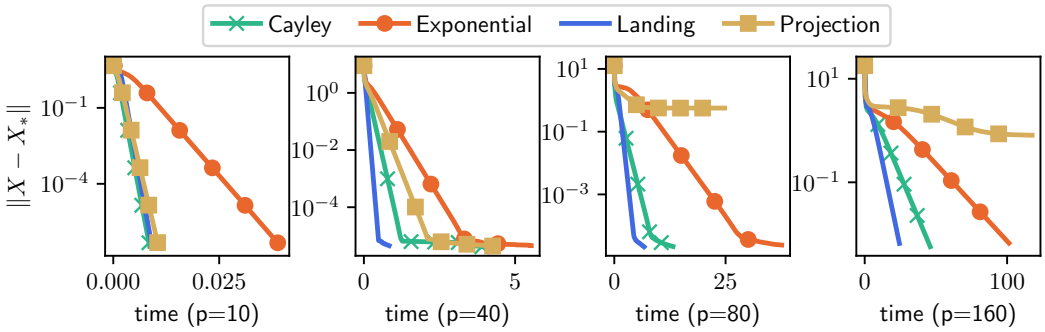


Figure 10: Varying the problem size $p$ on the Procrustes problem.

We a batch-size of 128.

We repeat the traning with 5 random seeds for each learning rate in the grid. In the figure in the main text, we only display the curves for the learning rate that lead to the fastest convergence. Bold curves correspond to the median of the runs, and individual runs are overlayed with a transparency.

Despite our best effort, we could not make trivializations converge to a satisfying accuracy, even when using a very small or very large learning rate. In this case, trivializations are both much more expensive than the proposed method, and cannot properly train the network.

### D.4 Additional experiment: effect of the problem dimension

On the procrustes problem, we vary the size of the problem $p$, and display the convergence curves in Fig 10. We observe that the landing algorithm is more useful for large values of $p$, where the cost of computing retractions becomes very high.