Learning Riemannian Stable Dynamical Systems via Diffeomorphisms

Anonymous Author(s)
Affiliation
Address
email

Abstract: Dexterous and autonomous robots should be capable of executing elaborated dynamical motions skillfully. Learning techniques may be leveraged to build models of such dynamic skills. To accomplish this, the learning model needs to encode a stable vector field that resembles the desired motion dynamics. This is challenging as the robot state does not evolve on a Euclidean space, and therefore the stability guarantees and vector field encoding need to account for the geometry arising from, for example, the orientation representation. To tackle this problem, we propose learning Riemannian stable dynamical systems (RSDS) from demonstrations, allowing us to account for different geometric constraints resulting from the dynamical system state representation. Our approach provides Lyapunov-stability guarantees on Riemannian manifolds that are enforced on the desired motion dynamics via diffeomorphisms built on neural manifold ODEs. We show that our Riemannian approach makes it possible to learn stable dynamical systems displaying complicated vector fields on both illustrative examples and real-world manipulation tasks, where Euclidean approximations fail.

Keywords: Dynamical systems, Riemannian manifolds, Motion learning

1 Introduction

The promise of having fully-autonomous robots performing a large variety of tasks implies that robots should be able to execute highly-dynamic motions. The inherent complexity of these movements makes hand coding an infeasible approach, and therefore learning techniques arise as a potential solution. In particular, learning dynamic robot motions from human demonstrations is a promising approach to build models of dynamic skills in an intuitive, sample-efficient and quick manner. However, learning dynamical motions is not trivial as the learning model requires to provide stability guarantees, which is also an intrinsic property in human motion generation [1]. In this context, most research works have focused on learning time-invariant stable dynamical systems for goal-driven motions (a.k.a point-to-point movements) with Lyapunov-stability guarantees [2, 3, 4, 5, 6].

Khansari-Zadeh and Billard [2] proposed one of the first approaches to learn stable dynamical systems from human demonstrations by imposing quadratic Lyapunov non-linear constraints on the model parameters optimization, which limited the range of possible learnable motions. As the class of stable dynamical systems constrained by a predefined Lyapunov function is a subset of all possible stable dynamical systems, this limits the learned model accuracy [4]. To improve accuracy, a more general parametric control Lyapunov function [7] can be learned from demonstrations [3, 8, 9]. In other works, the trade-off between stability and accuracy motivated the use of diffeomorphisms [4, 5, 10, 6], which leverage a more general class of stable dynamical systems. Their main idea is to design or learn a canonical Lyapunov-stable dynamical system on a latent space and use a diffeomorphic mapping to transform the demonstrations to the latent space so that they are consistent with the desired Lyapunov-stable behavior. Thus, the modeling accuracy depends on the expressiveness of the diffeormorphic function, often modeled by a neural network [5, 10, 6].

Submitted to the 6th Conference on Robot Learning (CoRL 2022). Do not distribute.

Most of aforementioned works assume that the training data lie in the Euclidean space [4, 5, 6], with the exception of Urain et al. [10], which severely limits their use in real robotic applications. For instance, there are various types of representation for the robot end-effector's orientation, namely unit quaternions in the 3-Sphere manifold [11], and rotation matrices in the special orthogonal group ($\mathrm{SO}(3)$) manifold [12], which do not lie in the Euclidean space. Accounting for the data geometry has proven critical when learning trajectory-based movement primitives on quaternion space [13, 14, 15, 16], as relying on Euclidean approximations leads to modeling distortions and compromises extrapolation. The importance of geometry-aware methods when learning dynamical systems was recently addressed in [10], where a stable dynamical system was learned via diffeomorphisms over Lie groups. Although Lie theory has been exploited to operate with data of specific geometries [17], a potential limitation is that not all types of manifolds arising in robotics can be easily endowed with a Lie group structure (e.g., the space of symmetric positive-definite matrices (SPD)).

A more general solution based on Riemannian geometry [18] is proposed in this paper. Specifically, we consider that our dynamical system evolves on a Riemannian manifold. This arises two main challenges: *(1)* designing a canonical Lyapunov-stable dynamical system on Riemannian manifolds, and *(2)* learning a diffeomorphism that accounts for the Riemannian geometry. To address these challenges, we leverage the Lyapunov stability analysis on Riemannian manifolds introduced in [19] and exploited for robot control [20]. Moreover, we exploit neural ordinary differential equations (ODEs) on Riemannian manifolds [21, 22] for constructing the diffeomorphism to learn a Riemannian stable dynamical system (RSDS). Unlike previous works using diffeomorphisms [4, 5, 6], our approach extends this concept to systems whose state variable evolves on a Riemannian manifold. Our approach is conceptually similar to the Lie-groups approach introduced in [10], as both explicitly consider the data geometry to design technically-sound learning models via diffeomorphisms. However, our Riemannian formulation substantially differs from [10] in its technical development, and provides a more general approach that may be exploited for a variety of Riemannian manifolds.

In summary, we propose a method to learn Riemannian stable dynamical systems from demonstrations. Our approach provides Lyapunov-stability guarantees on Riemannian manifolds (see § 3.1) that are enforced on the desired motion dynamics via diffeomorphisms built on neural manifold ODEs (see § 3.2 and 3.3). Through a set of evaluations on the 2-sphere manifold $\mathcal{S}^2$, presented in § 4, we show that our Riemannian approach is able to learn complicated dynamical systems, in contrast to Euclidean approximations which fail to encode stable vector fields. Also, we learn realistic motion skills on a 7-DoF robotic manipulator featuring complex full-pose trajectories on $\mathbb{R}^3 \times \mathcal{S}^3$.

2 Background

2.1 Dynamical Systems and Lyapunov Stability

We here give a short review of Lyapunov stability in the Euclidean setting. Let us assume an autonomous dynamical system $\dot{\boldsymbol{x}} = f(\boldsymbol{x})$, with a single equilibrium point $\boldsymbol{x}^*$, where $\boldsymbol{x} \in \mathbb{R}^n$ is the state variable. Consider a potential function $V(\boldsymbol{x}(t))$ describing the energy of such a system. If this system loses energy over time and the energy is never restored, the system must eventually reach some final resting state. This idea is formally described as (see [23] for details):

Theorem 1 (Lyapunov Stability)**.** *A dynamical system $\dot{\boldsymbol{x}} = f(\boldsymbol{x})$ is globally asymptotically stable at $\boldsymbol{x}^*$ if there exists a continuously differentiable Lyapunov function $V(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}$ such that*

$$V(\boldsymbol{x}^*) = 0, \quad \dot{V}(\boldsymbol{x}^*) = 0, \quad V(\boldsymbol{x}) > 0, \; \forall\, \boldsymbol{x} \neq \boldsymbol{x}^*, \quad \dot{V}(\boldsymbol{x}) < 0, \; \forall\, \boldsymbol{x} \neq \boldsymbol{x}^*. \tag{1}$$

From Theorem 1 we know that we can always find a Lyapunov function that fulfills these four conditions in Eq. 1 for a globally asymptotically stable dynamical system.

2.2 Riemannian Manifolds

A smooth manifold $\mathcal{M}$ can be seen as a set of points that locally, but not globally, resemble the Euclidean space $\mathbb{R}^d$ [18, 24]. An abstract definition of a manifold specifies the topological, differential

and geometric structure by using *charts*, which are maps between parts of $\mathcal{M}$ to $\mathbb{R}^d$. The collection of these charts (a.k.a. local parameterizations) is called *atlas*. More formally, a chart on a smooth manifold $\mathcal{M}$ is a diffeomorphic mapping (i.e. a bijective and differentiable function) $\varphi : U \to \tilde{U}$ from an open set $U \subset \mathcal{M}$ to an open set $\tilde{U} \subseteq \mathbb{R}^d$ (see Fig. 6 in App. A.1). Moreover, the transition map between two intersecting sets U_1 and U_2, given by $\varphi_1 \circ \varphi_2^{-1}$ or $\varphi_2 \circ \varphi_1^{-1} : \mathbb{R}^d \to \mathbb{R}^d$ is also a diffeomorphism. The smooth structure of $\mathcal{M}$ makes it possible to take derivatives of curves on the manifold, leading to tangent vectors in $\mathbb{R}^d$. The set of tangent vectors of all curves at $\boldsymbol{x} \in \mathcal{M}$ spans a d-dimensional affine subspace of $\mathbb{R}^d$, which is known as the *tangent space* $\mathcal{T}_{\boldsymbol{x}}\mathcal{M}$ of $\mathcal{M}$ at $\boldsymbol{x}$. The collection of all tangent spaces of $\mathcal{M}$ is the *tangent bundle* $\mathcal{TM} = \bigsqcup_{\boldsymbol{x}\in\mathcal{M}} \mathcal{T}_x\mathcal{M}$. Therefore, a velocity vector $\dot{\boldsymbol{x}}$ at $\boldsymbol{x} \in \mathcal{M}$ lies on $\mathcal{T}_x\mathcal{M}$, and consequently a vector field on $\mathcal{M}$ lies on $\mathcal{TM}$.

The above definitions do not provide the mechanisms to measure how curved $\mathcal{M}$ is, or to compute distances on $\mathcal{M}$. To do so, we can endow $\mathcal{M}$ with a *Riemannian metric*, which is a family of inner products $g_{\boldsymbol{x}} : \mathcal{T}_{\boldsymbol{x}}\mathcal{M} \times \mathcal{T}_{\boldsymbol{x}}\mathcal{M} \to \mathbb{R}$ associated to each point $\boldsymbol{x} \in \mathcal{M}$. As a result, a *Riemannian manifold* $(\mathcal{M}, g)$ is a smooth manifold endowed with a Riemannian metric [24]. To operate with Riemannian manifolds, it is common practice to exploit the Euclidean tangent spaces. To do so, we resort to mappings back and forth between $\mathcal{T}_{\boldsymbol{x}}\mathcal{M}$ and $\mathcal{M}$ using the exponential and logarithmic maps. The exponential map $\mathrm{Exp}_{\boldsymbol{x}}(\boldsymbol{u}) : \mathcal{T}_{\boldsymbol{x}}\mathcal{M} \to \mathcal{M}$ maps a point $\boldsymbol{u}$ in the tangent space of $\boldsymbol{x}$ to a point $\boldsymbol{y}$ on the manifold, so that it lies on the geodesic starting at $\boldsymbol{x}$ in the direction $\boldsymbol{u}$, and such that the geodesic distance $d_{\mathcal{M}}(\boldsymbol{x}, \boldsymbol{y}) = d_{\mathbb{R}}(\boldsymbol{x}, \boldsymbol{u})$. The inverse operation is the logarithmic map $\mathrm{Log}_{\boldsymbol{x}}(\boldsymbol{y}) : \mathcal{M} \to \mathcal{T}_{\boldsymbol{x}}\mathcal{M}$. We provide all the necessary operations in App. A.1.

2.3 Diffeomorphism

A diffeomorphism $\psi : \mathcal{M} \to \mathcal{N}$ is a smooth bijective mapping between two smooth manifolds which preserves the topological properties of $\mathcal{M}$, and whose inverse ψ^{-1} is also smooth. When learning stable dynamical systems, diffeomorphisms can be exploited to impose Lyapunov stability guarantees by transferring a manually-designed stable dynamical system on $\mathcal{N}$ to the desired manifold $\mathcal{M}$. We here focus on constructing learnable diffeomorphisms represented by continuous normalizing flows (CNFs) [25, 26, 27], which are bijective and bidirectionally differentiable mappings, and have been recently exploited on density estimation problems [28, 29, 30]. We exploit them to learn diffeomorphic mappings between Riemannian manifolds.

Most CNFs are constructed from *neural ordinary differential equation* (Neural ODEs) in Euclidean space [25, 26, 27], with the exception of *neural manifold ordinary differential equations* (Neural MODEs) on Riemannian manifolds [21, 22]. Generally, Neural ODEs parametrize the dynamics of a hidden variable using a continuous-time ODE represented by a neural network, as follows,

$$\dot{\boldsymbol{z}}(t) = f_{\boldsymbol{\theta}}(\boldsymbol{z}(t), t), \tag{2}$$

where $\boldsymbol{z} \in \mathbb{R}^d$ is the state variable and $f_{\boldsymbol{\theta}} : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$ is a neural network. According to Mathieu and Nickel [22] (see Theorem 2 in App. A.2), we can extend CNFs to the Riemannian setting, where the state variable $\boldsymbol{z} \in \mathcal{M}$ and the vector field $f_{\boldsymbol{\theta}} : \mathcal{M} \times \mathbb{R} \to \mathcal{TM}$. As a result, we can use (2) as a manifold ODE, whose initial value problem (IVP) solution results in a diffeormorphic mapping $\psi_{\boldsymbol{\theta}} : \mathcal{M} \to \mathcal{N}$, $\boldsymbol{x} = \boldsymbol{z}(t_s) \in \mathcal{M}$ and $\boldsymbol{y} = \boldsymbol{z}(t_e) \in \mathcal{N}$. i.e. $\boldsymbol{y} = \psi_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{x} + \int_{\tau=t_s}^{t_e} f_{\boldsymbol{\theta}}(\boldsymbol{z}(\tau), \tau) d\tau$. To solve the IVP on $\mathcal{M}$, we leverage integrators on Riemannian manifolds based on the local representation via coordinate charts [31]. This method uses a local representation of $\mathcal{M}$ defined by a coordinate map $\varphi : \mathcal{M} \supseteq U \to \tilde{U} \subseteq \mathbb{R}^d$ with coordinates $\boldsymbol{w}(t) = \varphi(\boldsymbol{z}(t))$. Computing integrators on $\mathcal{M}$ can be approximated by solving an equivalent ODE in $\mathbb{R}^d$

$$\dot{\boldsymbol{w}}(t) = D_{\varphi^{-1}(\boldsymbol{w}(t))}\varphi \circ f_{\boldsymbol{\theta}}(\varphi^{-1}(\boldsymbol{w}(t)), t), \tag{3}$$

where $D_{\varphi^{-1}(\boldsymbol{w}(t))}\varphi$ represents the differential of φ at $\varphi^{-1}(\boldsymbol{w}(t))$ (see App. A.2 and A.3 for details). Additionally, we use the adjoint method [25] on Riemannian manifolds [21] to compute gradients, which can also be used for calculating differential. Consider a loss function $\mathcal{L} : \mathcal{M} \to \mathbb{R}$, in order to compute the derivative of $\mathcal{L}$ with respect to an intermediate variable $\boldsymbol{z}(t)$ of the manifold ODE, we can solve $\dot{\boldsymbol{a}}(t)^\mathsf{T} = -\boldsymbol{a}(t)^\mathsf{T} D_{\boldsymbol{z}(t)} f_{\boldsymbol{\theta}}(\boldsymbol{z}(t), t)$, where $\boldsymbol{a}(t)^\mathsf{T} := D_{\boldsymbol{z}(t)}\mathcal{L}$ (as detailed in App. A.4).

3 Learning Riemannian Stable Dynamical Systems

We here introduce our approach for learning stable dynamical systems on Riemannian manifolds from demonstrated point-to-point motions. First, let us consider that the recorded demonstrations follow a dynamical system $\dot{\boldsymbol{x}} = f(\boldsymbol{x})$, where the state $\boldsymbol{x}$ evolves on a Riemannian manifold $\mathcal{M}$ with velocity $\dot{\boldsymbol{x}} \in \mathcal{T}_{\boldsymbol{x}}\mathcal{M}$. This dynamical system is equivalent, under a change of coordinates, to another system defined on a latent Riemannian manifold $\mathcal{N}$. Under the diffeomorphism $\psi_{\boldsymbol{\theta}} : \boldsymbol{x} \mapsto \boldsymbol{y} \in \mathcal{N}$, parameterized by $\boldsymbol{\theta}$, we map the observed states $\boldsymbol{x}$ onto $\mathcal{N}$. Then, we evaluate the canonical stable vector field $g_{\boldsymbol{\gamma}}(\boldsymbol{y})$ to obtain the velocity $\dot{\boldsymbol{y}} \in \mathcal{T}_{\boldsymbol{y}}\mathcal{N}$. Finally, we leverage the *pullback operator* $D_{\boldsymbol{y}}\psi^{\star}_{\boldsymbol{\theta}}$ to project $\dot{\boldsymbol{y}}$ back to the tangent space $\mathcal{T}_{\boldsymbol{x}}\mathcal{M}$. The whole procedure can be expressed as follows,

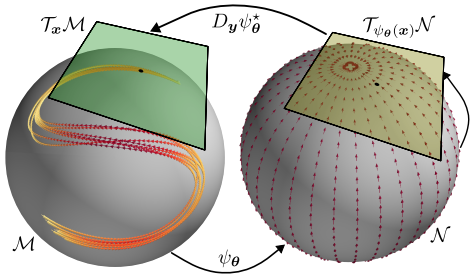

Figure 1: Architecture of a diffeomorphism-based stable vector field on the Riemannian manifold $\mathcal{S}^2$.

$$\dot{\boldsymbol{x}} = (D_{\boldsymbol{y}}\psi^{\star}_{\boldsymbol{\theta}} \circ g_{\boldsymbol{\gamma}} \circ \psi_{\boldsymbol{\theta}})(\boldsymbol{x}) = D_{\boldsymbol{y}}\psi^{\star}_{\boldsymbol{\theta}}(\dot{\boldsymbol{y}}), \tag{4}$$

which is illustrated in Fig. 1 (a proof is given in App. B). In the sequel, we describe how we design a Lyapunov-stable vector field $g_{\boldsymbol{\gamma}}$ on $\mathcal{N}$ to provide stability guarantees on the learned dynamical system. Later, we explain how to compute the diffeomorphism between the target and latent manifolds. Finally, we introduce two different methods to compute the pullback operator $D_{\boldsymbol{y}}\psi^{\star}_{\boldsymbol{\theta}}$.

3.1 Lyapunov-stable Geodesic Vector Fields

To design a stable vector field on the Riemannian manifold $\mathcal{N}$, we enforce the canonical dynamical system to follow geodesic curves that converge to a single equilibrium. Such a vector field can be designed via the logarithmic map. Specifically, given an equilibrium point $\boldsymbol{y}^* \in \mathcal{N}$, the corresponding velocity vector $\dot{\boldsymbol{y}} \in \mathcal{T}_{\boldsymbol{y}}\mathcal{N}$ can be computed as $\dot{\boldsymbol{y}} = g_{\boldsymbol{\gamma}}(\boldsymbol{y}) = k_{\boldsymbol{\gamma}}(\boldsymbol{y})g_n(\boldsymbol{y})$ with the normalized geodesics vector field $g_n(\boldsymbol{y}) := \frac{\mathrm{Log}_{\boldsymbol{y}}(\boldsymbol{y}^*)}{\|\mathrm{Log}_{\boldsymbol{y}}(\boldsymbol{y}^*)\|_2}$. This implies that the direction of tangent vectors is fully specified by $\mathrm{Log}_{\boldsymbol{y}}(\boldsymbol{y}^*)$, while their magnitude depends on the scaling factor $k_{\boldsymbol{\gamma}} : \mathbb{R}^n \supset \mathcal{N} \to \mathbb{R}_{\geq 0}$. We can prove the stability of this geodesic vector field by choosing the Lyapunov function $V(\boldsymbol{y}) := \langle F, F \rangle_{\boldsymbol{y}^*}$ with $F = \mathrm{Log}_{\boldsymbol{y}^*}(\boldsymbol{y})$, and applying Theorem 3 for Lyapunov stability on Riemannian manifolds, as detailed in App. B. Given that our geodesic vector field is Lyapunov stable, we can easily prove that the desired dynamical system is also globally asymptotically stable by defining a new valid Lyapunov function $\tilde{V}(\boldsymbol{x}) := V(\psi_{\boldsymbol{\theta}}(\boldsymbol{x}))$ via the diffeomorphism $\psi_{\boldsymbol{\theta}}$, with a single equilibrium point $\boldsymbol{x}^* = \psi^{-1}_{\boldsymbol{\theta}}(\boldsymbol{y}^*) \in \mathcal{M}$. As $\psi_{\boldsymbol{\theta}}$ preserves the topological properties of $\mathcal{N}$, the equilibrium point $\boldsymbol{x}^*$ is also globally asymptotically stable on $\mathcal{M}$ (see App. B for the proof).

Note that we separate the parameterization for the magnitude and direction of vector fields to improve the expressiveness of our framework. By relocating the scaling factor $k_{\boldsymbol{\gamma}}$ and normalizing the vector fields governed by (4), we can obtain our final learning framework

$$\dot{\boldsymbol{x}} = \hat{k}_{\boldsymbol{\gamma}}(\boldsymbol{x}) \frac{(D_{\boldsymbol{y}}\psi^{\star}_{\boldsymbol{\theta}} \circ g_n \circ \psi_{\boldsymbol{\theta}})(\boldsymbol{x})}{\|(D_{\boldsymbol{y}}\psi^{\star}_{\boldsymbol{\theta}} \circ g_n \circ \psi_{\boldsymbol{\theta}})(\boldsymbol{x})\|_2}, \tag{5}$$

where $\hat{k}_{\boldsymbol{\gamma}}(\boldsymbol{x})$ is the new positive scaling factor that fully determines the magnitude of the learned vector fields. In App. C, we prove that the models (4) and (5) are equivalent.

3.2 Diffeomorphisms on Riemmanian Manifolds

Having designed our canonical Lyapunov-stable dynamical system on $\mathcal{N}$, we can endow the desired vector field with stability properties using a diffeomorphism. To do so, we leverage Neural MODEs [21] to build such a diffeomorphism $\psi_{\boldsymbol{\theta}}$. According to Theorem 2 in App. A.2, the dynamics $f_{\boldsymbol{\theta}}$ of Neural MODEs only has to be a $\mathcal{C}^1$ function. To compute the diffeomorphism with a

parametric Neural MODE, we solve an integration problem based on the local parameterization $\boldsymbol{w}(t) = \varphi(\boldsymbol{z}(t))$ (described in App. A.3). Using this method requires the selection of coordinate charts, which can be created via the exponential map $\varphi_i^{-1} = \mathrm{Exp}_{\boldsymbol{z}_i}$ and logarithmic map $\varphi_i = \mathrm{Log}_{\boldsymbol{z}_i}$, similarly to [21]. Under this choice of coordinate mapping and given a fixed number of charts k, the diffeomorphism $\psi_{\boldsymbol{\theta}} : \boldsymbol{x} = \boldsymbol{z}_0 \mapsto \boldsymbol{z}_k = \boldsymbol{y}$, obtained via integration on the manifold can be then viewed as the composition of blocks of solving Neural ODEs and chart transitions defined as,

$$\begin{gathered}\psi_{\boldsymbol{\theta}} = \mathrm{Exp}_{\boldsymbol{z}_{k-1}} \circ \hat{\psi}_{\boldsymbol{\theta},k-1} \circ \mathrm{Log}_{\boldsymbol{z}_{k-1}} \circ \ldots \circ \mathrm{Exp}_{\boldsymbol{z}_0} \circ \hat{\psi}_{\boldsymbol{\theta},0} \circ \mathrm{Log}_{\boldsymbol{z}_0}, \quad \text{with} \\ \hat{\psi}_{\boldsymbol{\theta},i}(\boldsymbol{w}_i(t_{i,s})) = \boldsymbol{w}_i(t_{i,s}) + \int_{\tau=t_{i,s}}^{t_{i,e}} D_{\varphi_i^{-1}(\boldsymbol{w}_i(\tau))}\varphi_i \circ f_{\boldsymbol{\theta}}(\varphi_i^{-1}(\boldsymbol{w}_i(\tau)), \tau) d\tau,\end{gathered} \tag{6}$$

where i is the chart index, $t_{i,s}$ and $t_{i,e}$ are the starting and end time for i^{th} chart. $\hat{\psi}_{\theta,i}$ defines a diffeomorphism computed by the classical ODE solver on the tangent space (i.e. Euclidean space) and provides the solution of the IVP of the equivalent ODE (3).

3.3 Differential of the Inverse Diffeomorphism

We are now left with the problem of computing the pullback operator $D_{\boldsymbol{y}}\psi_{\boldsymbol{\theta}}^{\star}$ in (4), which maps the latent velocity $\dot{\boldsymbol{y}}$ back to the original tangent space $\mathcal{T}_{\boldsymbol{x}}\mathcal{M}$. This operator can be considered as the inverse mapping of the differential $D_{\boldsymbol{x}}\psi_{\boldsymbol{\theta}} : \mathcal{T}_{\boldsymbol{x}}\mathcal{M} \to \mathcal{T}_{\psi_{\boldsymbol{\theta}}(\boldsymbol{x})}\mathcal{N}$. As we already have the diffeomorphism $\psi_{\boldsymbol{\theta}}$, the straightforward solution is to compute its derivatives and then obtain the required inverse. Nevertheless, under the Riemannian setting, particularly for d-dimensional submanifolds $\mathcal{M}^d$ embedded in $\mathbb{R}^n$, computing the inverse directly becomes problematic due to the geometric constraints arising from $\mathcal{M}$. Next, we provide two methods to deal with this problem.

Pullback operator via constrained optimization: Instead of naively differentiating through the ODE solver of $\psi_{\boldsymbol{\theta}}$, we can use the adjoint method to calculate the differential of a diffeomorphism constructed by a Neural MODE. Assuming that we have the differential $D_{\boldsymbol{x}}\psi_{\boldsymbol{\theta}}$ (as computed in Algorithm 1 in App. A.4), the connection between tangent vectors $\dot{\boldsymbol{x}}$ and $\dot{\boldsymbol{y}}$ can be written as $D_{\boldsymbol{x}}\psi_{\boldsymbol{\theta}}(\boldsymbol{x})\dot{\boldsymbol{x}} = \dot{\boldsymbol{y}}$. In the Euclidean case, we can directly compute $\dot{\boldsymbol{x}} = (D_{\boldsymbol{x}}\psi_{\boldsymbol{\theta}}(\boldsymbol{x}))^{-1}\dot{\boldsymbol{y}}$. However, under the Riemannian setting, computing the inverse $(D_{\boldsymbol{x}}\psi_{\boldsymbol{\theta}}(\boldsymbol{x}))^{-1}$ often leads to a loss of rank in the matrix representation of $D_{\boldsymbol{x}}\psi_{\boldsymbol{\theta}}(\boldsymbol{x})$ for a embedded submanifold $\mathcal{M}^d$ due to the intrinsic geometric constraints of $\boldsymbol{x}$. We address this problem by introducing geometric constraints that allow us to compute $\dot{\boldsymbol{x}}$ on $\mathcal{T}_{\boldsymbol{x}}\mathcal{M}$. For example, for manifold $\mathcal{S}^d$, the tangent vector $\dot{\boldsymbol{x}}$ is orthogonal to $\boldsymbol{x}$, that is $\boldsymbol{x}^{\mathsf{T}}\dot{\boldsymbol{x}} = 0$. Hence, we can find a solution by solving a constrained optimization problem, from which the pullback operator $D_{\boldsymbol{y}}\psi_{\boldsymbol{\theta}}^{\star}$ is obtained as,

$$D_{\boldsymbol{y}}\psi_{\boldsymbol{\theta}}^{\star} = \left[D_{\boldsymbol{x}}\psi_{\boldsymbol{\theta}}(\boldsymbol{x})^{\mathsf{T}} D_{\boldsymbol{x}}\psi_{\boldsymbol{\theta}}(\boldsymbol{x}) + \boldsymbol{x}\boldsymbol{x}^{\mathsf{T}}\right]^{-1} D_{\boldsymbol{x}}\psi_{\boldsymbol{\theta}}(\boldsymbol{x})^{\mathsf{T}}. \tag{7}$$

The full derivation and discussions can be found in App. D.1. Note that $D_{\boldsymbol{y}}\psi_{\boldsymbol{\theta}}^{\star}$ in (7) is specific to hypersphere manifolds due to the choice of constraints. Thus, this constrained optimization approach does not easily scale to compute the pullback operator for arbitrary Riemannian manifolds.

Pullback operator via the adjoint method: To generalize the computation of the pullback operator for arbitrary Riemannian manifolds, we introduce a new approach based on a modified version of the adjoint method. By reversing the integration time interval (i.e. from $[t_s, t_e]$ to $[t_e, t_s]$), we can determine the inverse diffeomorphism $\psi_{\boldsymbol{\theta}}^{-1}$, which is a distinct benefit of Neural MODEs. Thus, the pullback operator $D_{\boldsymbol{y}}\psi_{\boldsymbol{\theta}}^{\star}$ can be viewed as the differential of the inverse diffeomorphism $D_{\boldsymbol{y}}(\psi_{\boldsymbol{\theta}}^{-1})$. Furthermore, we leverage the adjoint method to compute the differential of $\psi_{\boldsymbol{\theta}}^{-1}$ using the adjoint ODE $\dot{\boldsymbol{A}}^*(t) = -\boldsymbol{A}^*(t)D_{\boldsymbol{z}(t)}f_{\boldsymbol{\theta}}(\boldsymbol{z}(t), t)$, with $\boldsymbol{A}^*(t) := D_{\boldsymbol{z}(t)}(\psi_{\boldsymbol{\theta}}^{-1})$. Due to the availability of starting states $\boldsymbol{z}(t_s) = \boldsymbol{x}$ and $\boldsymbol{A}^*(t_s) = \boldsymbol{I}_n$, we can integrate both the Neural MODE (2) and adjoint ODE to get the $D_{\boldsymbol{y}}(\psi_{\boldsymbol{\theta}}^{-1})$. For clarification, we provide Algorithm 2 in App. D.2 for computing $D_{\boldsymbol{y}}(\psi_{\boldsymbol{\theta}}^{-1})$. Although, we can use dynamic charts method to solve the Neural MODE, dealing with the adjoint ODE dynamics is still not straightforward. The main challenge is to compute the differential of the vector fields on the Riemannian manifold $D_{\boldsymbol{z}(t)}f_{\boldsymbol{\theta}}(\boldsymbol{z}(t), t)$, despite it is nothing but

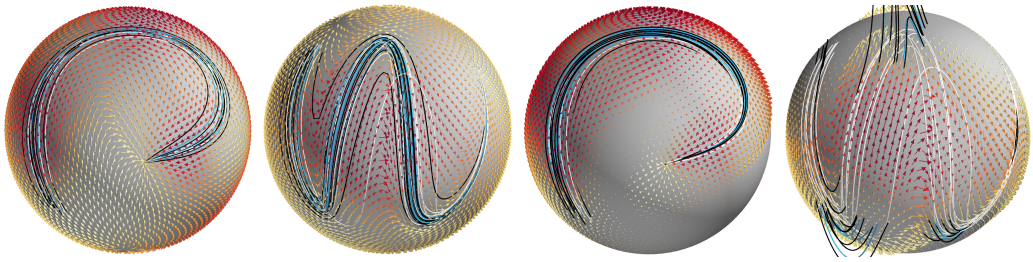

Figure 2: Experiments on LASA dataset on $\mathcal{S}^2$ (P and W letters): Demonstrations (white), learned vector fields, and reproductions (black and blue). Blue trajectories start at the same initial points as the demonstrations, while the black ones depart from randomly-sampled points around the initial points of the demonstrations. The first two plots show the results for our RSDS approach, and the last two display the *EuclideanFlow* results.

partial derivatives in the Euclidean case. To avoid directly computing the differential of vector fields on $\mathcal{M}$, we adopt an approach similar to (6), such that a component $\boldsymbol{z}_i = (\mathrm{Exp}_{\boldsymbol{z}_i} \circ \hat{\psi}^{-1}_{\boldsymbol{\theta},i} \circ \mathrm{Log}_{\boldsymbol{z}_i})\boldsymbol{z}_{i+1}$ for computing the inverse diffeomorphism has its differential as,

$$D_{\boldsymbol{z}_{i+1}} \boldsymbol{z}_i = D_{\boldsymbol{w}_i(t_{i,s})} \mathrm{Exp}_{\boldsymbol{z}_i} \circ D_{\boldsymbol{w}_i(t_{i,e})} \hat{\psi}^{-1}_{\boldsymbol{\theta},i} \circ D_{\boldsymbol{z}_{i+1}} \mathrm{Log}_{\boldsymbol{z}_i}, \tag{8}$$

where $D_{\boldsymbol{w}_i(t_{i,e})} \hat{\psi}^{-1}_{\boldsymbol{\theta},i}$ boils down to partial derivatives (the proof of (8) is provided in App. D.2).

4 Experiments

We evaluate our method in two different settings: reproducing trajectories based on the LASA dataset [24] projected on $\mathcal{S}^2$; and reproducing real dynamic motions learned from demonstrations. To show the importance of incorporating geometry, we provide a comparison with a baseline method similar to Euclideanizing flows [5], which is implemented using CNFs with Neural ODEs in $\mathbb{R}^n$ for illustrative experiments on a modified LASA dataset. For convenience, we refer to this baseline as *EuclideanFlow* and our model as *RSDS*.

4.1 LASA dataset on $\mathcal{S}^2$

Architectures: For *EuclideanFlow*, we use a fully-connected neural network with an input vector on $\mathbb{R}^4$ (i.e., the 3-dimensional state $\boldsymbol{x}$ and time), and 3 hidden layers each, with 32 hidden units for $\mathcal{S}^2$. We use $\tanh$ as activation function to guarantee a $\mathcal{C}^1$-bounded mapping for modeling the Neural ODE. The RSDS architecture has an additional projection operator proju on the head of network to impose the output on the tangent space of manifolds. The scaling factor $\hat{k}_{\boldsymbol{\gamma}}$ is generated using a network composed of a RBF layer and a linear layer without bias. The network architectures are depicted in App. E (Fig. 8). For all the experiments, we use an Euler ODE solver with step size of $1/32$, and 4 coordinate charts for the integration. For each dataset, one trajectory is used for testing and the remaining ones as training set. All models are trained using the ADAM optimizer [32] with learning rate of 10^{-3} (decaying by factor 0.1 after 1000 epochs) over 2000 epochs.

To illustratively show how RSDS and *EuclideanFlow* perform, we use a modified version of the LASA dataset of hand-written letters [33], whose trajectories are projected on $\mathcal{S}^2$. The corresponding vector field can be easily computed from the projected trajectories by using the logarithmic map. As a result, we have a new dataset $\{\{\boldsymbol{x}_{n,t}, \dot{\boldsymbol{x}}_{n,t}\}_{t=1}^{T_n}\}_{m=1}^{M}$ of M demonstrations for each letter, with positions $\boldsymbol{x} \in \mathcal{S}^2$ and velocities $\dot{\boldsymbol{x}} \in \mathcal{T}_{\boldsymbol{x}}\mathcal{S}^2$, from which we learn stable dynamical systems. Figure 2 shows the resulting demonstrations as white curves for datasets P and W, the corresponding learned vector fields and the reproduced trajectories. Note that we provide more results in App. F and comparison to an alternative method to *EuclideanFlow* that projects trajectories onto the manifold after computing the integration in Euclidean space. Concerning the reproductions, the blue and black trajectories are rollouts starting from the same initial position as the demonstrations and from randomly-sampled points around them, respectively. Regarding our RSDS approach (first two plots in Fig. 2), it is evident that all blue and black rollouts closely match the demonstrations and converge to the equilibrium. In contrast, the *EuclideanFlow* reproductions constantly leave the manifold since

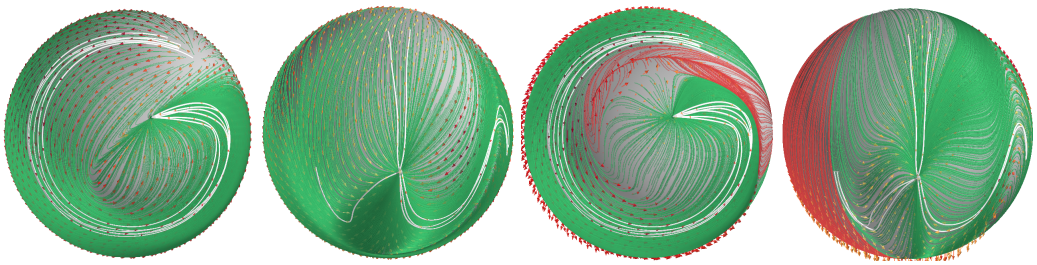

Figure 3: Stability of reproductions on $\mathcal{S}^2$ for G and MultiModels: 1000 trajectories starting from uniformly-sampled points. The successful and failed trajectories are depicted in green and red, respectively. The first two spheres from the left correspond to RSDS reproductions while the other two relate to *EuclideanFlow* results. For *EuclideanFlow*, we first project the vector fields onto $\mathcal{S}^2$ and then compute the integration trajectories.

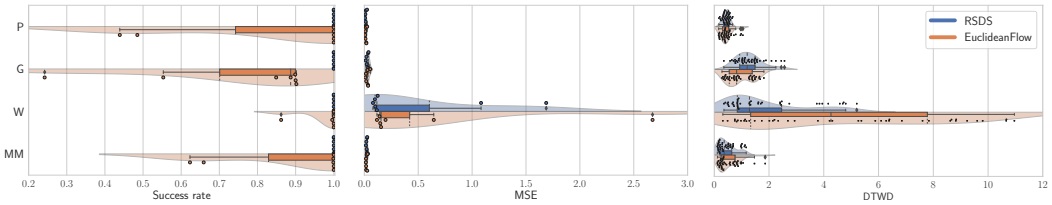

Figure 4: *Left*: Average success rate of RSDS and *EuclideanFlow* over randomly-sampled initial points on $\mathcal{S}^2$ using 7 different trained models indicated as points. *Middle:* Average mean square error (MSE) between observed and predicted velocity over data points in the test trajectories indicated as points. *Right:* Dynamic time warping distance (DTWD) between demonstrations and reproductions.

there are no mechanisms accounting for the inherent geometric constraints of the data (see last two plots in Fig. 2). The bald regions on the manifold where the velocity vectors point inwards the sphere are also evidence of this phenomenon. We also provide quantitative metrics for accuracy comparisons. Figure 4-*right*, and -*middle* show the *dynamic time warping distance* (DTWD) as a measure of reproduced position trajectory accuracy, and the *mean squared error* (MSE) of the velocities reproduction. These metrics show that RSDS outperforms *EuclideanFlow* for the most complex trajectories, e.g. the W dataset. Although Fig. 4 shows that both models seem to perform well on the P, G, and MultiModels datasets, as pointed out before, Fig. 2 displays that the *EuclideanFlow* trajectories do not obey the data geometry.

Secondly, we evaluate the stability of both approaches. For a fair comparison, we first project the vector fields onto $\mathcal{S}^2$ and compute the integration trajectories for *EuclideanFlow*. To quantitatively assess this, we measure the stability of the learned vector fields (i.e. convergence to the equilibrium), by uniformly sampling 1000 initial points on $\mathcal{S}^2$ and counting the number of trajectories that successfully converge. This procedure is repeated for 7 different trained models, with the average success rate computed over the initial points. Using one of the trained models, Fig. 3 shows green and red curves representing successful and failed trajectories, respectively. It is evident that a large number of the *Euclideanflows* trajectories failed to converge despite the projection, however all the *RSDS* trajectories succeeded. This result is supported by the success rate metric displayed in Fig. 4–*left*. These results show that accounting for the data geometry, as in our RSDS approach, is crucial to provide stability guarantees of the learned dynamical system. Additionally, we also compare the learning efficiency between the two methods, where RSDS generally requires fewer training epochs than *EuclideanFlow*. Nevertheless, each training epoch of *RSDS* is more computationally expensive due to manifold operations. The details of these results are provided in App. F (Fig. 9).

4.2 Real robot experiments on $\mathbb{R}^3 \times \mathcal{S}^3$

We evaluated two different manipulation tasks using the 7-DOF Franka-Emika Panda robot: *(1)* a GraspingTask with 90 degrees rotation, and *(2)* a V-shape DrawingTask. For both experiments, we collected 10 kinesthetic demonstrations of the robot end-effector motion as position-velocity trajectories at a frequency of 10 Hz. Here, we show that RSDS can indeed be used in real-world applications. We train our model on the $\mathbb{R}^3 \times \mathcal{S}^3$ manifold accounting for position and orientation

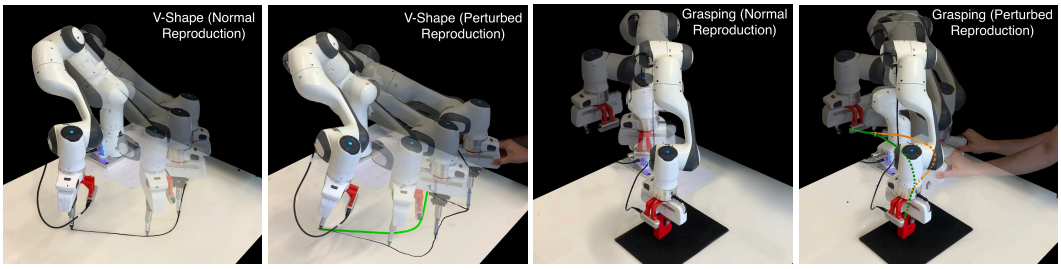

Figure 5: DrawingTask and GraspingTask: Time evolution of the reproductions is depicted by superimposed images from different time frames. The transparent robots depict the trace of the motion trajectory. The second plot for each task displays the task reproduction under perturbation, where the unperturbed reproduction is depicted as a green trajectory for reference.

of robot end-effector with the same network architecture as our illustrative experiments, except that we use 16 hidden units for faster computations. As shown in Fig. 5, while following the V-shape curve in the DrawingTask, the end-effector always faces the moving direction; similarly, when approaching the object in the GraspingTask, the gripper rotates 90 degrees. These motion patterns require synchronized position and orientation trajectories, which is only attainable by training a model with a state variable on a product manifold, i.e., $\boldsymbol{x} \in \mathbb{R}^3 \times \mathcal{S}^3$. To deploy the reproduced motion on the robot, we numerically integrated the desired velocity vector $\hat{\boldsymbol{x}} \in \mathcal{T}_{\hat{\boldsymbol{x}}}\mathcal{M}$ online, and used it as reference for a Cartesian impedance controller.

As observed in Fig. 5, the reproductions governed by the vector fields learned with our RSDS model accurately imitate the demonstrated motion patterns and converge to the goal position. Furthermore, these experiments incorporated some target shifts to test if our model could cope with them without retraining. We further evaluated the stability of the learned vector fields by perturbing the robot during the task reproduction. As Fig. 5 shows, after perturbing the robot, the end-effector still follows an alternative trajectory computed from the learned vector field (see black and orange trajectories for DrawingTask and GraspingTask, respectively).

5 Discussion

We presented a new approach RSDS to accurately learn vector fields on Riemannian manifolds while ensuring global asymptotic stability, which can not be achieved without taking into account the underlying geometry structure of the data. Our model inherits all the advantages of stable dynamical systems, such as high robustness against environmental perturbations. To our knowledge, RSDS is the first to leverage continuous normalizing flows on the Riemannian manifolds to learn an Lyapunov-stable Riemannian dynamical systems. Moreover, RSDS builds on a new methodology to compute the pullback operator leveraging the characteristics of neural MODEs. Our framework is generic and can theoretically be used to learn vector fields on any Riemannian manifolds with defined exponential and logarithmic maps. As future work, we will leverage RSDS to learn vector fields on other Riemannian manifolds such as the manifold of symmetric-positive-definite (SPD) matrices $\mathcal{S}^d_{++}$, which is relevant in manipulability learning [20] and video tracking [34].

Limitations: Due to the complexity of the Riemannian operators and the Neural MODE solvers, our framework runs relatively slowly, making it unsuitable for hard real-time applications. This problem can be alleviated by switching to faster ODE solvers after training, which allows us to accelerate the query time at the expense of precision. To improve the accuracy and stability of solving Neural MODEs, we can take advantage of techniques such as regularization [27] and recording checkpoint for the forward mode [35]. In addition, since we leverage the Lyapunov stability to a single fixed point, the model may still reproduce some trajectories that are inconsistent with the trend of demonstration data due to lack of information for points far from demonstrations. It may be worthwhile exploring other stability criteria, such as contraction analysis [36], to ensure the incremental exponential stability of trajectories with respect to each other on the manifold.

References

[1] E. Burdet, K. P. Tee, I. Mareels, T. E. Milner, C. M. Chew, D. W. Franklin, R. Osu, and M. Kawato. Stability and motor adaptation in human arm movements. *Biological Cybernetics*, 20(32), 2006. doi:10.1007/s00422-005-0025-9. 1

[2] S. M. Khansari-Zadeh and A. Billard. Learning stable nonlinear dynamical systems with Gaussian mixture models. *IEEE Transactions on Robotics*, 27(5):943–957, 2011. doi:10.1109/TRO.2011.2159412. 1

[3] S. M. Khansari-Zadeh and A. Billard. Learning control Lyapunov function to ensure stability of dynamical system-based robot reaching motions. *Robotics and Autonomous Systems (RAS)*, 62(6):752–765, 2014. doi:https://doi.org/10.1016/j.robot.2014.03.001. 1, 16

[4] K. Neumann and J. J. Steil. Learning robot motions with stable dynamical systems under diffeomorphic transformations. *Robotics and Autonomous Systems (RAS)*, 70:1–15, 2015. doi:https://doi.org/10.1016/j.robot.2015.04.006. 1, 2

[5] M. A. Rana, A. Li, D. Fox, B. Boots, F. Ramos, and N. Ratliff. Euclideanizing flows: Diffeomorphic reduction for learning stable dynamical systems. In *Conference on Learning for Dynamics and Control (L4DC)*, pages 630–639, 2020. URL https://proceedings.mlr.press/v120/rana20a.html. 1, 2, 6

[6] Y. Zhang, L. Cheng, H. Li, and R. Cao. Learning accurate and stable point-to-point motions: A dynamic system approach. *IEEE Robotics and Automation Letters (RA-L)*, 7(2):1510–1517, 2022. doi:10.1109/LRA.2022.3140677. 1, 2

[7] E. D. Sontag. A Lyapunov-like characterization of asymptotic controllability. *SIAM Journal on Control and Optimization*, 21(3):462–471, 1983. doi:10.1137/0321028. 1

[8] H. Ravanbakhsh and S. Sankaranarayanan. Learning control Lyapunov functions from counterexamples and demonstrations. *Autonomous Robots*, 45:275–307, 2019. doi:10.1007/s10514-018-9791-9. 1

[9] I. D. J. Rodriguez, A. D. Ames, and Y. Yue. Lyanet: A Lyapunov framework for training neural ODEs. In *International Conference on Machine Learning (ICML)*, pages 1–10, 2022. URL https://arxiv.org/abs/2202.02526. 1, 16

[10] J. Urain, D. Tateo, and J. Peters. Learning stable vector fields on Lie groups, 2021. URL https://arxiv.org/abs/2110.11774. 1, 2

[11] J.-Y. Wen and K. Kreutz-Delgado. The attitude control problem. *IEEE Transactions on Automatic Control*, 36(10):1148–1162, 1991. doi:https://doi.org/10.1109/9.90228. 2

[12] Y. Gu. Analysis of orientation representations by Lie algebra in robotics. In *International Conference on Robotics and Automation (ICRA)*, pages 874–879, 1988. doi:https://doi.org/10.1109/ROBOT.1988.12170. 2

[13] A. Ude, B. Nemec, T. Petrić, and J. Morimoto. Orientation in Cartesian space dynamic movement primitives. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2997–3004, 2014. URL https://doi.org/10.1109/ICRA.2014.6907291. 2

[14] M. Zeestraten. *Programming by Demonstration on Riemannian Manifolds*. PhD thesis, University of Genova, Italy, 2018. URL https://iris.unige.it/handle/11567/930621#.YCgbRuoo85k. 2

[15] L. Koutras and Z. Doulgeri. A correct formulation for the orientation dynamic movement primitives for robot control in the Cartesian space. In *Conference on Robot Learning (CoRL)*, pages 293–302, 2020. URL https://proceedings.mlr.press/v100/koutras20a.html. 2

[16] L. Rozo and V. Dave. Orientation probabilistic movement primitives on Riemannian manifolds. In *Conference on Robot Learning (CoRL)*, 2021. URL https://openreview.net/forum?id=csMg2h_LR37. 2

[17] J. Solà, J. Deray, and D. Atchuthan. A micro Lie theory for state estimation in robotics. *CoRR*, abs/1812.01537, 2018. URL http://arxiv.org/abs/1812.01537. 2

[18] M. P. do Carmo. *Riemannian Geometry*. Birkhäuser, 1st edition, 1992. 2

[19] F. Pait and D. Colón. Some properties of the Riemannian distance function and the position vector X, with applications to the construction of Lyapunov functions. In *IEEE Conference on Decision and Control (CDC)*, pages 6277–6280, 2010. doi:10.1109/CDC.2010.5717398. 2

[20] N. Jaquier, L. Rozo, D. G. Caldwell, and S. Calinon. Geometry-aware manipulability learning, tracking, and transfer. *The International Journal of Robotics Research (IJRR)*, 40(2-3):624–650, 2021. doi:10.1177/0278364920946815. 2, 8, 15

[21] A. Lou, D. Lim, I. Katsman, L. Huang, Q. Jiang, S. N. Lim, and C. M. De Sa. Neural manifold ordinary differential equations. In *Neural Information Processing Systems (NeurIPS)*, pages 17548–17558, 2020. URL https://proceedings.neurips.cc/paper/2020/file/cbf8710b43df3f2c1553e649403426df-Paper.pdf. 2, 3, 4, 5, 12, 13, 14

[22] E. Mathieu and M. Nickel. Riemannian continuous normalizing flows. In *Neural Information Processing Systems (NeurIPS)*, pages 2503–2515, 2020. URL https://proceedings.neurips.cc/paper/2020/file/1aa3d9c6ce672447e1e5d0f1b5207e85-Paper.pdf. 2, 3, 12

[23] J. E. Slotine and W. Li. *Applied Nonlinear Control*. Prentice-Hall, Englewood Cliffs, NJ, 1991. 2

[24] J. Lee. *Introduction to Riemannian Manifolds*. Springer, 2nd edition, 2018. doi:10.1007/978-3-319-91755-9. 2, 3, 6

[25] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud. Neural ordinary differential equations. In *Neural Information Processing Systems (NeurIPS)*, 2018. URL https://proceedings.neurips.cc/paper/2018/file/69386f6bb1dfed68692a24c8686939b9-Paper.pdf. 3, 12, 13

[26] W. Grathwohl, R. T. Q. Chen, J. Bettencourt, and D. Duvenaud. Scalable reversible generative models with free-form continuous dynamics. In *International Conference on Learning Representations (ICLR)*, 2019. URL https://openreview.net/forum?id=rJxgknCcK7. 3

[27] C. Finlay, J.-H. Jacobsen, L. Nurbekyan, and A. M. Oberman. How to train your neural ODE: the world of jacobian and kinetic regularization. In *International Conference on Machine Learning (ICML)*, pages 3154–3164, 2020. doi:10.5555/3524938.3525234. 3, 8

[28] D. J. Rezende and S. Mohamed. Variational inference with normalizing flows. In *International Conference on Machine Learning (ICML)*, pages 1530—1538, 2015. doi:doi/10.5555/3045118.3045281. 3

[29] G. Papamakarios, E. Nalisnick, D. J. Rezende, S. Mohamed, and B. Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. *Journal of Machine Learning Research*, 22(57):1–64, 2021. URL http://jmlr.org/papers/v22/19-1028.html. 3

[30] I. Kobyzev, S. J. Prince, and M. A. Brubaker. Normalizing flows: An introduction and review of current methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(11):3964–3979, 2021. doi:10.1109/TPAMI.2020.2992934. 3

[31] E. Hairer. Solving differential equations on manifolds, 2011. URL http://www.unige.ch/~hairer/polycop.html. 3, 13

[32] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *International Conference for Learning Representations (ICLR)*, 2015. URL http://arxiv.org/abs/1412.6980. 6

[33] A. Lemme, Y. Meirovitch, M. Khansari-Zadeh, T. Flash, A. Billard, and J. J. Steil. Open-source benchmarking for learned reaching motion generation in robotics. *Paladyn, Journal of Behavioral Robotics*, 6(1), 2015. doi:doi:10.1515/pjbr-2015-0002. 6

[34] G. Cheng and B. C. Vemuri. A novel dynamic system in the space of SPD matrices with applications to appearance tracking. *SIAM Journal on Imaging Sciences*, 6(1):592–615, 2013. doi:10.1137/110853376. 8

[35] J. Zhuang, N. Dvornek, X. Li, S. Tatikonda, X. Papademetris, and J. Duncan. Adaptive checkpoint adjoint method for gradient estimation in neural ODE. In *International Conference on Machine Learning (ICML)*, pages 11639–11649, 2020. URL https://proceedings.mlr.press/v119/zhuang20a.html. 8

[36] C. Dawson, S. Gao, and C. Fan. Safe control with learned certificates: A survey of neural Lyapunov, barrier, and contraction methods. *ArXiv*, abs/2202.11762, 2022. URL https://arxiv.org/abs/2202.11762. 8