

RIEMANNIAN FUZZY K-MEANS ON PRODUCT MANIFOLDS

000
001
002
003
004
005
006
007
008
009
010
011
012
013
014
015
016
017
018
019
020
021
022
023
024
025
026
027
028
029
030
031
032
033
034
035
036
037
038
039
040
041
042
043
044
045
046
047
048
049
050
051
052
053
Anonymous authors
Paper under double-blind review

ABSTRACT

In this paper, we address an open problem: how to perform fast clustering on product manifolds. With the increasing interest in non-Euclidean data representations, clustering such data has become an important problem. However, a naive extension of the classic K-Means algorithm to product manifolds requires $\mathcal{O}(\nu\omega)$ time, where ω is the number of alternating iterations and ν is the time complexity of each Riemannian optimization. Due to the need for numerous Riemannian optimizations, the naive Riemannian K-Means (NRK) is not suitable for large-scale data. To this end, we propose the Riemannian Fuzzy K-Means (RFK) algorithm for product manifolds, which reduces the time complexity to $\mathcal{O}(\nu)$. Importantly, RFK is not a straightforward extension of K-Means or Fuzzy K-Means to manifolds, it avoids the computation of the Fréchet mean and and achieve a true single-loop optimization. Furthermore, we introduce Rada to accelerate the optimization of RFK. We conduct extensive experiments. RFK and Rada outperform across nearly all metrics in almost every dataset, reaching an impressive level of performance. **RFK and Rada have been integrated into several non-Euclidean machine learning libraries, such as here. (See Appendix G)**

1 INTRODUCTION

Non-Euclidean data representations have received widespread attention. Examples include text embeddings in hyperbolic space (Dhingra et al., 2018), tree embeddings in the Poincaré disk (Nickel & Kiela, 2017), and representations of cell-cycle data on the sphere (Bjerregaard et al., 2025). This is because many real-world datasets exhibit non-Euclidean structure, and embedding such data in appropriate non-Euclidean spaces can better preserve those structures (Sinha et al., 2024; Khan et al., 2025). For example, hyperbolic space captures the hierarchical structure (Mandica et al., 2024a; Chlenski et al., 2024), while spheres retain the periodic information (Bonev et al., 2025). Many datasets not only have a single structure, so to preserve as much information as possible (Gu et al., 2018), it is necessary to represent them on product manifolds.

A product manifold is formed by the Cartesian product of multiple manifolds (Wang et al., 2021), i.e., $\mathcal{M} = \mathcal{M}_1 \times \dots \times \mathcal{M}_Q = \bigotimes_{p=1}^Q \mathcal{M}_p$, where \mathcal{M}_p denotes the p -th component manifold of the product manifold, with $p \in \{1, \dots, Q\}$. The product manifold inherits the characteristics of each component manifold and possesses greater expressive power (Chlenski et al., 2025a). As a result, product manifolds have been widely used for representing data from diverse domains (Sun et al., 2022; McNeela et al., 2023; Xu et al., 2022; Chen et al., 2025), and clustering data represented on such manifolds or their components has become an important problem (Sun et al., 2023a).

A natural approach is to naively extend the K-Means to product manifolds, referred to as Naive Riemannian K-Means (NRK) (Miolane et al., 2020). However, we point out that NRK incurs a time complexity of $\mathcal{O}(\nu\omega)$, where ω is the number of alternating iterations and ν is the time complexity of each Riemannian optimization (Yuan et al., 2025b). This results in a double-loop structure to solve the clustering problem, which is unacceptable for large-scale data. Therefore, how to perform clustering efficiently remains an open problem that requires a solution (Tepper et al., 2018).

To address this problem, we propose Riemannian Fuzzy K-Means, abbreviated as RFK. Specifically, we consider the equivalent relaxed version of K-Means, namely Fuzzy K-Means (Dehuriya et al., 2010). We identify a special structure of Fuzzy K-Means and leverage a particular technique to

transform the required double loop into a single loop and reduces the time from complexity $\mathcal{O}(\nu\omega)$ to $\mathcal{O}(\nu)$. In other words, the previously required ω times Riemannian optimizations are reduced to only 1, significantly lowering the computational cost, where $\omega \gg 1$.

To further accelerate RFK, we adapt the well-known Nesterov adaptive optimization algorithm (Adan) (Xie et al., 2024) to product manifolds, resulting in a Riemannian Nesterov acceleration method, termed Radan. We also establish the regret bound (Mukkamala & Hein, 2017) and convergence properties of Radan under certain conditions.

We validate our algorithm on a wide range of datasets. Specifically, we perform clustering using various methods on data represented in hyperbolic space, spherical manifolds, Euclidean space, and their product manifolds. We compare the speed of RFK with that of NRK, the speed of Radan with that of Riemannian Adam (Becigneul & Ganea, 2019), and the clustering performance of RFK with several state-of-the-art clustering algorithms. We conducted extensive experiments, which yielded remarkable results: RFK significantly outperforms NRK in speed, Radan converges faster than Radam, and RFK achieved the best clustering performance on nearly all datasets. In summary, our contributions are following.

- We address the open problem of fast clustering on product manifolds by proposing the RFK algorithm, which reduces the time complexity from $\mathcal{O}(\nu\omega)$ of the naive Riemannian K-Means to $\mathcal{O}(\nu)$.
- We modify the Adan optimizer to make it compatible with product manifolds, resulting in Radan, and provide theoretical guarantees including a regret bound and convergence proof.
- We conduct extensive numerical experiments to demonstrate the effectiveness of our algorithm. RFK is significantly faster than NRK, Radan provides acceleration over the Riemannian Adam (Radam), and RFK substantially outperforms existing algorithms in clustering metrics on manifold-represented data.

In addition, we propose a new insight, pointing out that the reason NRK cannot be accelerated lies in its hard assignment. We recommend RFK instead of NRK for clustering on manifolds.

2 PRELIMINARIES

2.1 NOTATIONS

Let the dataset be $X = \{x_1, \dots, x_N\}$, and let c_j denote the j -th cluster center. C is the number of clusters. For a product manifold denoted by \mathcal{M} , each of its component manifolds is written as \mathcal{M}_p , such that $\mathcal{M} = \otimes_{p=1}^Q \mathcal{M}_p$. For any $\mathbf{x} \in \mathcal{M}$, \mathbf{x} can be represented as (x^1, x^2, \dots, x^Q) , where $x^p \in \mathcal{M}_p$. For any points x^p and y^p on the component manifold \mathcal{M}_p , $d_p(x^p, y^p)$ denotes the geodesic distance between x^p and y^p on \mathcal{M}_p , the geodesic distance on \mathcal{M} is denoted by $d(x, y)$.

Let $\mathbb{H}^{h_i, K}$ denote a Lorentz hyperbolic space of dimension h_i with curvature K , $\mathbb{S}^{s_i, K}$ denote a spherical manifold of dimension s_i with curvature K , \mathbb{R}^{r_i} denote a Euclidean space of dimension r_i , and \mathbb{D} denote a two-dimensional Poincaré disk. Especially, when the curvatures of $\mathbb{S}^{s_i, K}$ and $\mathbb{H}^{h_i, K}$ are $(1, -1)$, we denote them simply as \mathbb{S}^{s_i} and \mathbb{H}^{h_i} , respectively.

$T_{x^p} \mathcal{M}_p$ denotes the tangent space of the component manifold \mathcal{M}_p at point x^p , and $\|\cdot\|$ denotes the norm in Euclidean space. The parallel transport on \mathcal{M}_p from point x^p to y^p is denoted by $\varphi_{x^p \rightarrow y^p}^p(u^p)$, where $u^p \in T_{x^p} \mathcal{M}_p$. When there is no ambiguity, it is abbreviated as $\varphi^p(u^p)$. The parallel transport on the product manifold \mathcal{M} is denoted by $\varphi_{x \rightarrow y}(u)$. The exponential map on \mathcal{M}_p is denoted by $\text{Exp}_{c^p}^p(u^p)$, and the exponential map on \mathcal{M} is denoted by $\text{Exp}_c(u)$. $\text{Log}_{c^p}^p(x^p)$ denotes the logarithmic map on \mathcal{M}_p . $\text{Log}_c(x)$ denotes the logarithmic map on the product manifold \mathcal{M} ; $\log(\cdot)$ refers to the natural logarithm. All the notations are summarized in Table 4.

2.2 CONSTANT-CURVATURE SPACES AND PRODUCT MANIFOLDS

Constant-curvature spaces (Jos et al., 1967) refer to one of the following: spherical spaces (positive curvature), hyperbolic spaces (negative curvature) or Euclidean spaces (Alekseevskij et al., 1993).

For an s -dimensional sphere $\mathbb{S}^{s,K}$, it can be represented as $\mathbb{S}^{s,K} = \{x \in \mathbb{R}^{s+1} \mid \|x\| = \frac{1}{K}, K > 0\}$. $\forall x, y \in \mathbb{S}^{s,K}$, the geodesic distance between x and y is $d(x, y) = \frac{\cos^{-1}(K^2 \langle x, y \rangle)}{K}$, where $\langle x, y \rangle$ denotes the normal inner product in \mathbb{R}^{s+1} (Whittlesey, 2019).

For an h -dimensional hyperbolic space $\mathbb{H}^{h,K}$, it can be represented as: $\mathbb{H}^{h,K} = \{x \in \mathbb{R}^{h+1} \mid \|x\|_h = \langle x, x \rangle_h = -\frac{1}{K^2}, K < 0, x^0 \geq 0\}$, where any $x \in \mathbb{H}^{h,K}$ is written as $x = (x^0, \dots, x^h)$, with $x^i \in \mathbb{R}^1$ (Iversen, 1992), and the Lorentzian inner product (Tsamparlis, 2024) is defined as $\langle x, y \rangle_h = -x^0 y^0 + \sum_{i=1}^h x^i y^i$. For any $x, y \in \mathbb{H}^{h,K}$, the geodesic distance (He et al., 2025) between x and y is given by $d(x, y) = -\frac{\cosh^{-1}(K^2 \langle x, y \rangle_h)}{K}$. where \mathbb{H} is also known as the well-known Lorentz (hyperboloid) model of hyperbolic space.

A product manifold can be represented as $\mathcal{M} = \otimes_{p=1}^Q \mathcal{M}_p$. For any $x, y \in \mathcal{M}$, the geodesic distance is generally given by Equation (1), where $x^p \in \mathcal{M}_p$ (Fumero et al., 2021).

$$d(x, y) = \sqrt{\sum_{p=1}^Q d_p^2(x^p, y^p)}, \quad x, y \in \mathcal{M} = \otimes_{p=1}^Q \mathcal{M}_p, \quad x^p, y^p \in \mathcal{M}_p \quad (1)$$

When we focus on product manifolds composed of constant-curvature spaces, the structure becomes $\mathcal{M} = \otimes_{i=1}^n \mathbb{S}^{s_i, K} \times \otimes_{j=1}^m \mathbb{H}^{h_j, K} \times \mathbb{R}^r$. The dimension is $\sum_{i=1}^n s_i + \sum_{j=1}^m h_j + r$ (Lui, 2012).

2.3 K-MEANS AND FUZZY K-MEANS

The K-Means algorithm is a well-known clustering method (Likas et al., 2003; Na et al., 2010), and its optimization problem can be formulated as following (Sinaga & Yang, 2020):

$$\begin{cases} \min_{c_j, u_{ij}} J_{KM} = \sum_{i=1}^N \sum_{j=1}^C u_{ij} \|x_i - c_j\|^2 \\ \text{s.t.} \quad \sum_{j=1}^C u_{ij} = 1, \quad u_{ij} \in \{0, 1\}, \quad \forall i = 1, \dots, N, \forall j = 1, \dots, C \end{cases} \quad (2)$$

Here, u_{ij} is an indicator variable, where $u_{ij} = 1$ indicates that the i -th sample belongs to the j -th cluster. This problem is typically solved by alternating updates of $\{u_{ij}\}$ and $\{c_j\}$.

Fuzzy K-Means is a relaxed version of K-Means (Xu et al., 2016; Krasnov et al., 2023), in which the constraint $u_{ij} \in \{0, 1\}$ is relaxed to $0 \leq u_{ij} \leq 1$, with the additional requirement that $\sum_{j=1}^C u_{ij} = 1$, where C is the number of clusters. Moreover, the loss term of K-Means $u_{ij} \|x_i - c_j\|^2$ is replaced by $u_{ij}^m \|x_i - c_j\|^2$ when using fuzzy K-Means, where m is the fuzziness parameter (Li & Wang, 2023; Suganya & Shanthi, 2012; Bezdek et al., 1984). Other related work can be found in Appendix C.

3 OUR PROPOSED METHOD

3.1 NAIIVE EXTENSION OF K-MEANS

The K-Means is clearly unsuitable for data represented on a manifold \mathcal{M} , for two following reasons and shown in Figure 1.

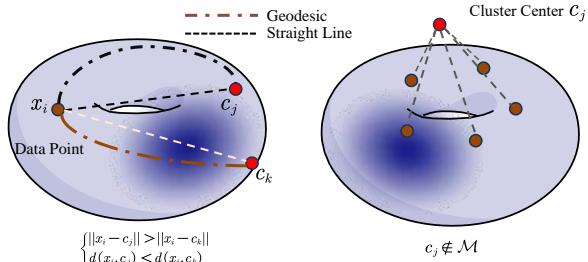


Figure 1: Visualization of the two reasons

- **Incorrect distance comparisons:** When data lie on a manifold, the Euclidean distance may have $\|x_i - c_j\| < \|x_i - c_k\|$, while the actual geodesic distances satisfy $d(x_i, c_j) > d(x_i, c_k)$. This mismatch can lead to incorrect cluster assignments.
- **Invalid cluster centers:** Without appropriate constraints, the computed cluster centers c_j may lie outside the manifold, i.e., $c_j \notin \mathcal{M}$, rendering the cluster centers meaningless in the context of the manifold. Meanwhile, the geodesic distance $d(x_i, c_j)$ is not well-defined.

162 Therefore, a naive approach is to replace the Euclidean distance with geodesic distance and impose
 163 the constraint that the cluster centers lie on the manifold. This leads to the following Equation (3).
 164

$$\begin{cases}
 \min_{c_j, u_{ij}} J_{KM}(u_{ij}, c_j) = \sum_{i=1}^N \sum_{j=1}^C u_{ij} d^2(x_i, c_j) = \sum_{i=1}^N \sum_{j=1}^C \sum_{p=1}^Q u_{ij} d_p^2(x_i^p, c_j^p) \\
 \text{s.t. } \sum_{j=1}^C u_{ij} = 1, u_{ij} \in \{0, 1\}, \quad \forall i = 1, \dots, N, \forall j = 1, \dots, C \\
 \text{s.t. } c_j \in \mathcal{M}, \quad \mathcal{M} = \otimes_{p=1}^Q \mathcal{M}_p, \quad \forall j = 1, \dots, C
 \end{cases} \quad (3)$$

172 Similar to K-Means in Euclidean space, this problem can be solved by alternating updates of $\{u_{ij}\}$
 173 and $\{c_j\}$. The update of $\{u_{ij}\}$ is identical to that in the Euclidean case: for each x_i , one simply
 174 identifies the cluster center c_j that minimizes $\sum_{p=1}^Q d_p^2(x_i^p, c_j^p)$, and sets the corresponding $u_{ij} = 1$.
 175 However, the update of $\{c_j\}$ differs significantly from the Euclidean case.
 176

177 When updating $\{c_j\}$, the constraint $c_j \in \mathcal{M}$, $\mathcal{M} = \otimes_{p=1}^Q \mathcal{M}_p$ leads to the Riemannian optimization
 178 problem (4), which can typically be addressed using methods such as Riemannian gradient descent.
 179

$$\begin{cases}
 \min_{c_j} J_{KM}(c_j) = \sum_{i=1}^N \sum_{j=1}^C u_{ij} d^2(x_i, c_j) = \sum_{i=1}^N \sum_{j=1}^C \sum_{p=1}^Q u_{ij} d_p^2(x_i^p, c_j^p) \\
 \text{s.t. } c_j \in \mathcal{M}, \quad \mathcal{M} = \otimes_{p=1}^Q \mathcal{M}_p, \quad \forall j = 1, \dots, C
 \end{cases} \quad (4)$$

184 This is the well-known problem of finding Fréchet means (Iao et al., 2025; Wu & Pan, 2025a) on a
 185 manifold. In general, closed-form solutions do not exist (Capitaine et al., 2024), it's the fundamental
 186 difference from the flat Euclidean spaces. It is also means that the naive extension of Fuzzy K-Means
 187 to manifolds also requires computing the Fréchet centers, which entails the same time complexity.
 188 This **highlights** that our proposed RFK algorithm is not a naive extension of Fuzzy K-Means.
 189

190 This approach to performing K-Means clustering on product manifolds is referred to as Naive
 191 Riemannian K-Means (NRK). Analyzing this algorithm, it is not difficult to see that if computing
 192 the Fréchet mean in each iteration requires Riemannian optimization with time complexity $\mathcal{O}(\nu)$
 193 (Lou et al., 2020), and the clustering process involves $\mathcal{O}(\omega)$ alternating updates of $\{u_{ij}\}$ and $\{c_j\}$,
 194 then the total time complexity is $\mathcal{O}(\nu\omega)$. Since both ν and ω are typically large, clustering becomes
 195 unacceptable for large-scale data. Therefore, reducing the time complexity is of critical importance.
 196

3.2 RIEMANNIAN FUZZY K-MEANS

197 From the above analysis, it is clear that due to the constraint $c_j \in \mathcal{M}$, $\mathcal{M} = \otimes_{p=1}^Q \mathcal{M}_p$, Riemannian
 198 optimization is unavoidable. Therefore, if we aim to reduce the overall complexity, the only viable
 199 approach is to reconsider the treatment of $\{u_{ij}\}$.
 200

201 If a smooth mapping $u_{ij} = f(c_j)$ can be found, such that J_{KM} becomes a differentiable function of
 202 c_j , then alternating optimization can be avoided entirely. However, for standard K-Means, this is not
 203 possible. The update rule for u_{ij} is inherently non-smooth and discrete:
 204

$$u_{ij} = \begin{cases} 1, & j = \operatorname{argmin}_{j \in \{1, \dots, C\}} \sum_{p=1}^Q d_p^2(x_i^p, c_j^p), \\ 0, & \text{otherwise.} \end{cases} \quad (5)$$

207 To address this issue, we adopt the relaxed version of K-Means, Fuzzy K-Means, whose optimization
 208 objective is given by:
 209

$$\begin{cases}
 \min_{c_j, u_{ij}} J_{FK}(u_{ij}, c_j) = \sum_{i=1}^N \sum_{j=1}^C u_{ij}^m d^2(x_i, c_j) = \sum_{i=1}^N \sum_{j=1}^C \sum_{p=1}^Q u_{ij}^m d_p^2(x_i^p, c_j^p) \\
 \text{s.t. } \sum_{j=1}^C u_{ij} = 1, u_{ij} \geq 0, \quad \forall i = 1, \dots, N, \forall j = 1, \dots, C \\
 \text{s.t. } c_j \in \mathcal{M}, \quad \mathcal{M} = \otimes_{p=1}^Q \mathcal{M}_p, \quad \forall j = 1, \dots, C
 \end{cases} \quad (6)$$

216 For fixed $\{c_j\}$, the optimal memberships u_{ij} are given in closed form by:
 217

$$218 \quad 219 \quad u_{ij}(c_j) = \underset{u_{ij} \geq 0, \sum_{j=1}^C u_{ij} = 1}{\operatorname{argmin}} \left(\sum_{i=1}^N \sum_{j=1}^C \sum_{p=1}^Q u_{ij}^m d_p^2(x_i^p, c_j^p) \right) = \left(\sum_{k=1}^C \left(\frac{\sum_{p=1}^Q d_p^2(x_i^p, c_j^p)}{\sum_{p=1}^Q d_p^2(x_i^p, c_k^p)} \right)^{\frac{1}{m-1}} \right)^{-1}, \quad (7)$$

221 By substituting $u_{ij}(c_j)$ into J_{FK} , the objective function J_{FK} can be expressed as an optimization
 222 problem depending solely on $\{c_j\}$, specifically:
 223

$$224 \quad 225 \quad J_{FK}(u_{ij}(c_j), c_j) = \underbrace{\sum_{i=1}^N \sum_{j=1}^C \left[\sum_{k=1}^C \left(\frac{\sum_{p=1}^Q d_p^2(x_i^p, c_j^p)}{\sum_{p=1}^Q d_p^2(x_i^p, c_k^p)} \right)^{\frac{1}{m-1}} \right]^{-m} \sum_{p=1}^Q d_p^2(x_i^p, c_j^p)}_{A_1} \\ 226 \\ 227 \\ 228 \quad 229 \quad = \sum_{i=1}^N \sum_{j=1}^C \left(\left(\sum_{p=1}^Q d_p^2(x_i^p, c_j^p) \right)^{\frac{1}{m-1}} S_i \right)^{-m} \sum_{p=1}^Q d_p^2(x_i^p, c_j^p) = \sum_{i=1}^N \sum_{j=1}^C \left(\sum_{p=1}^Q d_p^2(x_i^p, c_j^p) \right)^{-\frac{1}{m-1}} S_i^{-m} \quad (8) \\ 230 \\ 231 \quad 232 \quad = \sum_{i=1}^N S_i^{-m} \sum_{j=1}^C \left(\sum_{p=1}^Q d_p^2(x_i^p, c_j^p) \right)^{-\frac{1}{m-1}} = \sum_{i=1}^N S_i^{1-m} = \underbrace{\sum_{i=1}^N \left(\sum_{j=1}^C \left(\sum_{p=1}^Q d_p^2(x_i^p, c_j^p) \right)^{-\frac{1}{m-1}} \right)^{1-m}}_{A_2}.$$

235 Let $S_i = \sum_{j=1}^C \left(\sum_{p=1}^Q d_p^2(x_i^p, c_j^p) \right)^{-\frac{1}{m-1}}$ be an intermediate variable introduced during simplification.
 236 By simplifying to form A_2 , the objective function J_{FK} is expressed solely in terms of $\{c_j\}$.
 237 This enables Riemannian optimization to be performed directly on $\{c_j\}$, without alternating between
 238 $\{u_{ij}\}$ and $\{c_j\}$.

240 It is important to note that the simplification from A_1 to A_2 is necessary, because computing the
 241 gradient of Equation (4) requires evaluating a triple sum, while differentiating A_1 involves a quadruple
 242 sum. Only by converting to the A_2 form, also involving a triple sum, can we ensure that this step
 243 does not introduce additional computational cost.

244 Analyze the time complexity of optimizing J_{FK} : since the time complexity of taking the derivative
 245 of Equation (4) and that of A_2 are the same (both have closed-form solutions), and operations such as
 246 computing the Riemannian gradient during the optimization process also have identical complexity,
 247 while Equation (4) requires ω times alternating updates between $\{u_{ij}\}$ and $\{c_j\}$, A_2 only requires
 248 one optimization. Therefore, we have successfully reduced the time complexity from $\mathcal{O}(\nu\omega)$ to $\mathcal{O}(\nu)$.

249 Specifically, when the distance on the product manifold is replaced by the distance on the manifold
 250 \mathcal{M} , A_2 can be further simplified as Equation (9). **For convenience, we will also use the notation**
 251 **in Equation (9) in the following sections.**

$$252 \quad 253 \quad J_{FK}(u_{ij}(c_j), c_j) = \sum_{i=1}^N \left(\sum_{j=1}^C d(x_i, c_j)^{-\frac{2}{m-1}} \right)^{1-m}, d(x_i, c_j) = \sqrt{\sum_{p=1}^Q d_p^2(x_i^p, c_j^p)}, c_j \in \mathcal{M} = \otimes_{p=1}^Q \mathcal{M}_p \quad (9)$$

258 3.3 RADAN ON PRODUCT MANIFOLDS

260 To further accelerate the RFK algorithm, we modify the Adan optimizer (Xie et al., 2024) and adapt
 261 it to product manifolds. Adan is an algorithm that incorporates Nesterov acceleration (Zhou et al.,
 262 2024) into adaptive optimization (Yue et al., 2021). We expect that this type of Nesterov method can
 263 also be effective for optimization on product manifolds.

264 For Adan, we adopt a standard modification strategy (Boumal, 2023). Our adaptation of Adan consists
 265 of three main components: updating momentum via parallel transport, maintaining the second-order
 266 moment as a scalar, and performing updates using the exponential map. Specifically, let Riemannian
 267 Adan at the t -th iteration involve parameters $\{g_t^p, m_t^p, v_t^p, z_t^p, n_t^p, u_t^p, \alpha_t^p\}, p \in \{1, \dots, Q\}$, where
 268 g denotes the Riemannian gradient, m the momentum, v an estimate of the Riemannian gradient
 269 difference, z and n the estimations of the second-order moment of the gradient, u the update direction,
 and α the learning rate, with p indicating the component on the p -th manifold \mathcal{M}_p . During the update

270 of m_{t+1}^p , we apply parallel transport, i.e., $m_t^p = \beta_{1t}^p \varphi^p(m_{t-1}^p) + (1 - \beta_{1t}^p)g_t^p$, similar updates are
 271 applied to all other vector-based quantities involving subtraction. For the scalar maintenance of n_t^p ,
 272 we use the update $n_t^p = \beta_{3t}^p n_{t-1}^p + (1 - \beta_{3t}^p) \|z_t^p\|_{y_t^p}^2$. Finally, the parameter update is conducted via
 273 the exponential map: $y_{t+1}^p = \text{Exp}^p(-\alpha_t^p u_t^p)$.
 274

$$\begin{array}{lll}
 \left\{ \begin{array}{l} m_t = \beta_{1t} m_{t-1} + (1 - \beta_{1t}) g_t \\ v_t = \beta_{2t} v_{t-1} + (1 - \beta_{2t}) (g_t - g_{t-1}) \\ z_t = g_t + \beta_{2t} (g_t - g_{t-1}) \\ n_t = \beta_{3t} n_{t-1} + (1 - \beta_{3t}) (z_t \odot z_t) \\ u_t = m_t + \beta_{2t} v_t \\ \alpha_t = \frac{\eta_t}{\sqrt{n_t}} + \epsilon_t \\ y_{t+1} = y_t - \alpha_t u_t \end{array} \right. & \left\{ \begin{array}{l} m_t = \beta_{1t} \varphi(m_{t-1}) + (1 - \beta_{1t}) g_t \\ v_t = \beta_{2t} \varphi(v_{t-1}) + (1 - \beta_{2t}) (g_t - \varphi(g_{t-1})) \\ z_t = g_t + \beta_{2t} (g_t - \varphi(g_{t-1})) \\ n_t = \beta_{3t} n_{t-1} + (1 - \beta_{3t}) \|z_t\|_{y_t}^2 \\ u_t = m_t + \beta_{2t} v_t \\ \alpha_t = \frac{\eta_t}{\sqrt{n_t}} + \epsilon_t \\ y_{t+1} = \text{Exp}_{y_t}(-\alpha_t u_t) \end{array} \right. & \left\{ \begin{array}{l} m_t = \beta_{1t} \varphi(m_{t-1}) + (1 - \beta_{1t}) g_t \\ v_t = \beta_{2t} \varphi(v_{t-1}) + (1 - \beta_{2t}) \|g_t\|_{y_t}^2 \\ u_t = m_t \\ \alpha_t = \frac{\eta_t}{\sqrt{v_t}} + \epsilon_t \\ y_{t+1} = \text{Exp}_{y_t}(-\alpha_t u_t) \end{array} \right. \\
 \text{(a) Adan optimizer} & \text{(b) Radan optimizer} & \text{(c) Radam optimizer}
 \end{array}$$

Figure 2: Update Process Illustration of Adan, Radan, and Radam Optimizers.

284 Figure 2 presents the update details of Adan, Radan, and Radam on the product manifold, using the
 285 simplified notation from Equation (9). Here, $m_t = (m_t^1, \dots, m_t^Q)$, $\beta_{1t} = (\beta_{1t}^1, \dots, \beta_{1t}^Q)$, and other
 286 variables are similarly updated on each component manifold.
 287

288 To characterize its local convergence rate, We adopt a standard approach by analyzing the algorithm
 289 in a region where geodesic convexity holds, as the vicinity of a local minimum is guaranteed to
 290 be geodesically convex under standard second-order optimality conditions (Boumal, 2023). In this
 291 setting, we assume the product manifold \mathcal{M} is bounded by a diameter D_∞ and has a curvature
 292 function $\zeta(\kappa, c)$. This is also a common assumption in the literature (Becigneul & Ganea, 2019).

293 **Theorem 3.1.** *Let y_t be the sequence generated by the Radan algorithm. Under the standard
 294 assumptions, the regret bound R_T satisfies the following. The proof is in Appendix A.1.*

$$\begin{aligned}
 R_T &\leq \frac{\zeta(\kappa, c) \cdot (3 - 2\beta_1) \eta G^2 \sqrt{T} \sqrt{1 + \log T}}{2(1 - \beta_1)^3} + \frac{2\eta\beta_1 G}{(1 - \beta_1)^3} \sqrt{T} \\
 &\quad + \frac{GD_\infty^2 (1 + 2\beta_2) \sqrt{T}}{2(1 - \beta_1) \cdot \eta} + \sum_{t=1}^T \frac{4D_\infty^2 G^2 \beta_{2t}}{1 - \beta_1} + \sum_{t=1}^T \frac{\sqrt{t}(1 + 2\beta_2) GD_\infty^2 \beta_{1t}}{\eta(1 - \beta_1)}
 \end{aligned} \tag{10}$$

301 **Theorem 3.2.** *In the bound Equation (10), any non-summation term $K(T)$ satisfies $\mathbf{o}\left(\frac{K(T)}{T}\right) = 0$. For the summation terms, as long as the parameter decay conditions $\mathbf{o}\left(\frac{\sum_{t=1}^T \beta_{1t} \sqrt{t}}{T}\right) = 0$,
 302 $\mathbf{o}\left(\frac{\sum_{t=1}^T \beta_{2t}}{T}\right) = 0$ and $\beta_{3t} = 1 - \frac{1}{t}$ are met, Radan converges to the optimum. Here, $\mathbf{o}(\cdot)$ represent
 303 asymptotically vanishing terms. The proof is in Appendix A.2.*

304 While our convergence proof requires decaying β , our experiments adopt the standard practice of
 305 using fixed values for their proven empirical effectiveness and simplicity (Becigneul & Ganea, 2019;
 306 Kochurov et al., 2020). By optimizing Equation (9), we obtain the final cluster centers $\{c_j\}$ upon
 307 completion. Then, by applying Equation (7), we compute the assignment results $\{u_{ij}\}$, completing
 308 the clustering process.

3.4 CALCULATE RIEMANNIAN GRADIENT

315 During the Riemannian optimization process, it is also necessary to compute the Riemannian gradient.
 316 Below, we provide the expressions for the Riemannian gradient on three constant curvature manifolds:
 317 Euclidean space, hyperspherical manifold, and hyperbolic space.
 318

319 **Theorem 3.3.** *On a single constant-curvature manifold \mathbb{R}^r , $\mathbb{S}^{s,K}$, or $\mathbb{H}^{h,K}$, the Riemannian gradient
 320 of the Riemannian Fuzzy K-Means objective function J_{FK} with respect to the cluster center c_k is
 321 uniformly expressed as:*

$$\text{grad}_{c_k} J_{FK} = -2 \sum_{i=1}^N S_i^{-m} d(x_i, c_k)^{-\frac{2m}{m-1}} \text{Log}_{c_k}(x_i), \tag{11}$$

324 where $\text{Log}_{c_k}(x_i)$ denotes the logarithmic map of point x_i at c_k . The $\text{Log}_{c_k}(x_i)$ on three types of
 325 constant-curvature manifolds are given as follows. The proof is in Appendix A.3.

$$327 \quad \text{Log}_c(x) = \begin{cases} x - c, & \text{if } x, c \in \mathbb{R}^r, \\ 328 \quad \frac{\theta}{\sin(\theta)} (x - \cos(\theta) c), & \theta = \cos^{-1}(K^2 \langle c, x \rangle), \text{ if } x, c \in \mathbb{S}^{s, K}, \\ 329 \quad \frac{\theta}{\sinh(\theta)} (x + K^2 \langle c, x \rangle_h c), & \theta = \cosh^{-1}(K^2 \langle c, x \rangle_h), \text{ if } x, c \in \mathbb{H}^{h, K}. \\ 330 \end{cases} \quad (12)$$

331 After computing according to Equation (11), the expression of the Riemannian gradient can be
 332 obtained. By combining the Riemannian gradient with the corresponding logarithmic map, exponen-
 333 tial map, and other operations on different manifolds, all steps of the Riemannian optimization to
 334 solve RFK can be completed. Thereafter, we conduct extensive experiments on the RFK and Radan
 335 algorithms to validate their speed and superior performance.

337 4 EXPERIMENTS

340 In this section, we conducted extensive experiments aiming to answer the following three questions:

341 • Q1: How much faster is the RFK algorithm compared to the NRF algorithm when run on
 342 product manifolds? Does it achieve a lower loss value?

343 • Q2: When running Radan on product manifolds, does it accelerate the RFK algorithm
 344 compared to Radam with standard hyperparameters?

345 • Q3: Compared to the current state-of-the-art clustering algorithms, can RFK demonstrate
 346 better advantages for data represented on product manifolds?

348 We also provide several sensitivity analyses, including those on the fuzziness index m , the number of
 349 cluster centers, and other key hyperparameters in the Appendix F.2.

351 4.1 DATASETS

353 The datasets on product manifolds include four parts: synthetic data, graph embedding data and
 354 mixed-curvature VAE latent space data. More details are in Table 5.

355 **Synthetic Data:** We use the '*gaussian mixture*' function from Manify (Chlenski et al., 2025b) to
 356 generate data with 3 clusters on different product manifolds, and generate a set of labels for clustering.

358 **Graph Embedding Data:** For the graph embedding data, it is divided into two parts. One part
 359 selects the optimal embedding from $\{(\mathbb{H}^2)^2, \mathbb{H}^2\mathbb{E}^2, \mathbb{H}^2\mathbb{S}^2, \mathbb{S}^2\mathbb{E}^2, (\mathbb{S}^2)^2, \mathbb{H}^4, \mathbb{E}^4, \mathbb{S}^4\}$ by means
 360 of curvature estimation (Gu et al., 2018). The other part embeds the data into the 2D Poincaré disk \mathbb{D} .

362 **Mixed-curvature VAE Latent Space:** We use data from the latent space of a mixed-curvature
 363 variational autoencoder (Skopek et al., 2020) as the datasets, including the MNIST with over 600,000
 364 samples. These product manifold representations are derived from the Manify (Chlenski et al.,
 365 2025b).

366 We emphasize that the data already lying on the manifolds are the actual data we use, without
 367 requiring any additional preprocessing.

368 4.2 EXPERIMENTS SETUP

370 4.2.1 EXPERIMENT SETUP FOR Q1

372 To verify that our RFK algorithm is faster than NRK, we ran both algorithms on the aforementioned
 373 datasets and recorded their execution times. To ensure fair timing comparisons, we replaced non-
 374 vectorized operations with matrix-based implementations for NRK (see Equation (7)). For the
 375 optimization part, we used the proposed Radan optimizer for both methods, with parameters set as
 376 $\{\text{Radan: } \beta_1^p = 0.7, \beta_2^p = 0.99, \beta_3^p = 0.99\}$, and a common learning rate of 0.5 for testing. For RFK, the
 377 stopping criterion for Radan was that the change in loss between iteration t and $t + 1$ was less than
 378 $1e - 4$. For NRK, there are two convergence criteria: the condition for updating the Fréchet mean is

378 **Table 1: RFK & NRK Time (s) and Cost on Datasets, OT means out-of-time**

Method	Gauss \mathbb{R}^4		Gauss \mathbb{H}^4		Gauss $\mathbb{S}^2\mathbb{H}^2$		Gauss $\mathbb{R}^2\mathbb{S}^2\mathbb{H}^2$		Gauss $\mathbb{S}^2(\mathbb{H}^2)^2$		Gauss $\mathbb{R}^4\mathbb{S}^4\mathbb{H}^4$		Gauss $\mathbb{R}^{16}\mathbb{S}^{16}\mathbb{H}^{16}$		CiteSeer	
	Time	Loss	Time	Loss	Time	Loss	Time	Loss	Time	Loss	Time	Loss	Time	Loss	Time	Loss
RFK	0.07	1499.84	0.21	1832.57	0.19	791.87	0.23	1569.92	0.45	1518.82	0.28	3549.86	0.25	35869.52	1.02	17.77
NRK	0.60	1451.24	36.27	1845.32	2.37	791.87	6.12	1569.92	63.90	1518.82	2.78	3549.86	0.82	35869.54	52.23	17.89
Method	Cora	PolBlogs	Olsson	Paul	PoolBooks	CIFAR-100	Lymphoma	MNIST								
RFK	0.29	16.00	0.13	39.54	0.17	65.88	0.25	88.06	0.07	127.00	67.28	46450.93	3.61	878.25	82.76	668268.50
NRK	68.46	16.01	0.44	39.54	4.82	65.88	606.23	88.07	2.40	127.01	OT	OT	OT	OT	OT	OT

385
386 the same as in RFK, while the global convergence condition is that the distance between the Fréchet
387 centers of two consecutive iterations is less than $1e-4$.

389 4.2.2 EXPERIMENT SETUP FOR Q2

390
391 To evaluate the optimization capabilities of Radan and Radam on the RFK loss function, we designed
392 Experiment 2, where both optimizers adopt their standard parameter settings: {Radan: $\beta_1^p = 0.7$,
393 $\beta_2^p = 0.99$, $\beta_3^p = 0.99$ }, {Radam: $\beta_1^p = 0.99$, $\beta_2^p = 0.999$ }. We trained using a range of learning
394 rates {0.1, 0.3, 0.5, 0.7, 1}, comparing the minimum and last values of the mean RFK loss under
395 different learning rates. Each optimizer was run for 300 iterations. Notably, we use standard
396 hyperparameters since adaptive optimizers are considered insensitive to them (Gkouti et al., 2024),
397 and we aim to spare users from tuning when applying RFK.

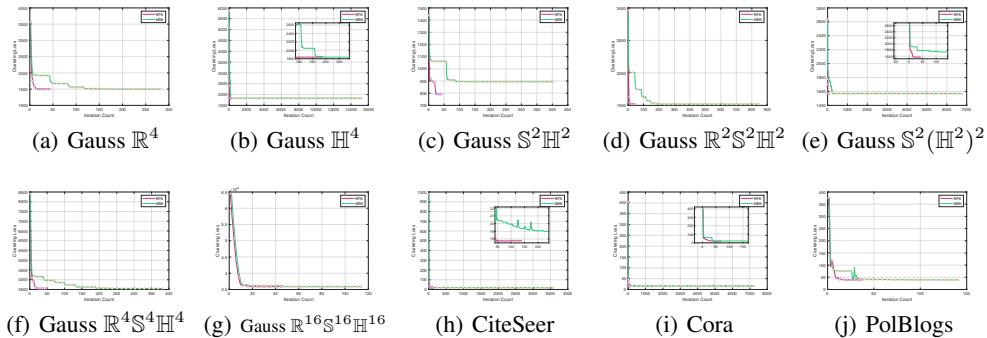
398 4.2.3 EXPERIMENT SETUP FOR Q3

399
400 To compare the clustering performance of the RFK algorithm, we evaluated it on the above datasets
401 against 10 competitive algorithms (Hu et al., 2023; Abdullah et al., 2024; Nie et al., 2024; Zhong
402 & Pun, 2021; Chen et al., 2017; Huang et al., 2019; Nie et al., 2023; Liu et al., 2012; Elhamifar &
403 Vidal, 2013), using five metrics: ACC (Yuan et al., 2025a; Wang et al., 2025), NMI (Xie et al., 2025),
404 ARI (Yuan et al., 2024), F1 (Du et al., 2024), and Purity (Huang et al., 2024). RFK was optimized by
405 Radan. Detailed experimental settings are in Appendix E.2.

406 4.3 EXPERIMENTS RESULT

407 4.3.1 EXPERIMENT RESULT FOR Q1

408
409 Table 1 presents the runtime and final loss of the RFK and NRK algorithms. As shown, RFK achieves
410 speedups of over 100x compared to NRK on some datasets. On certain large-scale datasets, NRK
411 runs out of time. Although RFK and NRK optimize the same objective, RFK generally attains a
412 lower final loss. Figure 3 shows the loss curves of RFK and NRK. The NRK curves exhibit step-like
413 drops due to alternating updates of the assignment and the Fréchet center, whereas the RFK curves
414 decrease more smoothly, require significantly fewer iterations, and converge to a lower final value.



415
416 Figure 3: Clustering loss curves for RFK and NRK
417

418 4.3.2 EXPERIMENT RESULT FOR Q2

419
420 Table 2 presents the average loss reduction results using the Radan and Radam optimizers. It can be
421 seen that Radan generally achieves lower loss values than Radam. Figure 4 shows the loss curves,

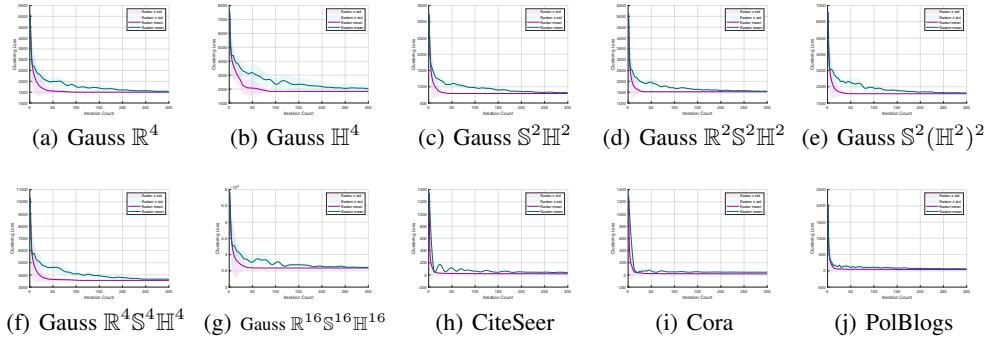
432 Table 2: Radan & Radam Min and Last Loss on Various Datasets, OT means out-of-time

Method	Gauss \mathbb{R}^4		Gauss \mathbb{H}^4		Gauss $\mathbb{S}^2\mathbb{H}^2$		Gauss $\mathbb{R}^2\mathbb{S}^2\mathbb{H}^2$		Gauss $\mathbb{S}^2(\mathbb{H}^2)^2$		Gauss $\mathbb{R}^4\mathbb{S}^4\mathbb{H}^4$		Gauss $\mathbb{R}^{16}\mathbb{S}^{16}\mathbb{H}^{16}$		CiteSeer Min Last	
	Min	Last	Min	Last	Min	Last	Min	Last	Min	Last	Min	Last	Min	Last		
Radan	1499.84	1499.84	1832.57	1832.58	791.87	792.61	1518.82	1518.89	1569.92	1569.98	3459.86	3459.86	35869.50	35869.50	18.61	18.62
Radam	1533.27	1533.27	2016.92	2016.92	814.98	814.98	1544.57	1546.15	1619.51	1621.24	3627.76	3627.76	36047.06	36090.44	39.35	41.45
Method	Cora		PolBlogs		Olsson		Paul		PoolBooks		CIFAR-100		Lymphoma		MNIST	
Radan	17.23	17.23	39.60	39.74	66.92	66.93	83.95	84.11	126.65	126.71	48850.73	49127.77	878.26	878.27	662611.31	667403.93
Radam	36.60	44.79	61.43	64.19	66.08	66.08	80.70	80.70	126.86	126.86	70860.14	71248.29	3381.86	3716.99	746378.43	727619.73

439 Table 3: ACC for all benchmarks. OT means out-of-time

	Dataset	Signature	RFK	NRK	K-Means	Ncut	FCM	UFCM	LRR	SSC	SBMC	USPEC	Fast-CD
Synthetic	Gaussian	\mathbb{R}^4	96.00	95.40	96.02	99.00	96.00	96.10	88.80	37.30	78.05	73.60	97.20
		\mathbb{H}^4	99.80	99.00	40.42	99.10	55.40	39.30	99.80	47.80	60.30	90.41	66.50
		$\mathbb{S}^2\mathbb{H}^2$	95.20	94.80	83.91	88.00	87.70	87.30	97.70	41.20	68.46	74.00	87.70
		$\mathbb{R}^2\mathbb{S}^2\mathbb{H}^2$	96.20	95.80	61.20	99.40	75.80	60.36	93.70	45.10	82.07	46.22	86
		$\mathbb{S}^2(\mathbb{H}^2)^2$	97.80	97.80	44.23	77.05	47.70	44.67	96.60	43.60	73.21	60.37	61.20
		$\mathbb{R}^4\mathbb{S}^4\mathbb{H}^4$	99.10	98.90	39.55	68.40	64.70	41.50	97.20	38.70	87.17	62.21	95.90
		$\mathbb{R}^{16}\mathbb{S}^{16}\mathbb{H}^{16}$	98.00	77.10	37.98	76.30	40.50	37.71	38.60	37.40	55.38	63.93	53.50
Graph	CiteSeer	$(\mathbb{H}^2)^2$	25.36	20.09	20.80	24.91	20.05	21.60	19.86	25.35	19.81	23.92	20.62
		\mathbb{H}^4	29.22	18.19	18.06	20.57	18.27	18.83	18.15	29.10	17.08	20.00	18.19
		$(\mathbb{S}^2)^2$	94.36	93.62	93.90	54.66	93.54	94.01	68.66	51.96	54.65	59.16	93.70
		D	67.72	67.45	61.57	60.24	60.37	60.71	44.16	60.73	51.31	57.25	60.21
		Paul	D	52.73	48.15	47.05	45.47	46.57	46.86	22.94	13.88	26.01	44.03
		PolBooks	D	81.90	68.57	39.68	34.27	36.34	42.44	OT	OT	8.81	44.12
		CIFAR-100	$(\mathbb{H}^2)^4$	71.19	OT	5.75	OT	6.00	5.53	OT	OT	5.21	OT
		Lymphoma	$(\mathbb{S}^2)^2$	100.00	OT	78.28	OT	78.28	OT	OT	OT	78.28	OT
		MNIST	$\mathbb{S}^2\mathbb{E}^2\mathbb{H}^2$	96.09	OT	12.09	OT	15.40	13.01	OT	OT	11.42	OT

454 with the red curve representing the mean loss of Radan and the blue representing Radam. The shaded
455 areas indicate variance. Radan consistently converges faster than Radam, typically within 50–100
456 iterations, whereas Radam requires around 300 iterations. Additionally, Radan generally achieves
457 lower final loss values.



470 Figure 4: Clustering loss curves for Radan and Radam

473 4.3.3 EXPERIMENT RESULT FOR Q3

474 Table 3 presents the ACC metric of different clustering algorithms across various datasets. The
475 Dataset column lists all the datasets used, and Signature indicates the geometric structure of each
476 dataset. RFK is our proposed algorithm. As shown in the table, our method achieves the best
477 performance on nearly every dataset. In particular, for the MNIST dataset with 600,000 data points,
478 most clustering algorithms fail to produce results; K-Means achieves only about **12%** accuracy,
479 whereas RFK reaches **96.09%** accuracy, which is a remarkable outcome. This result is reasonable
480 because MNIST is well represented in non-Euclidean space, where existing algorithms cannot respect
481 the intrinsic geometric structure, while RFK effectively operates in non-Euclidean space, yielding this
482 impressive performance. Other results can be found in Appendix F.1. **Here, OT denotes out-of-time.**
483 **All experiments were run on an Intel(R) Core(TM) i5-10200H CPU @ 2.40 GHz, with a predefined**
484 **time limit of 3600 seconds (1 hour). Any algorithm that fails to converge within this time window is**
485 **marked as out-of-time (OT).**

486

5 LIMITATIONS

488 We acknowledge that this work still has several limitations that warrant further investigation. First, our
 489 theoretical assumptions rely on geodesic convexity, meaning that the convergence analysis of Radan
 490 focuses on its behavior in a neighborhood around a local optimum. In future work, we aim to establish
 491 convergence guarantees under more general conditions. Second, our analysis of Radan’s convergence
 492 relies on a decaying learning rate, whereas our experiments use a fixed learning rate. Although this is
 493 a common practice in Riemannian adaptive optimization, we plan to explore how to bridge this gap.
 494 Finally, Riemannian Fuzzy K-Means requires access to closed-form geodesic distance formulas for
 495 the manifolds on which it operates. While these formulas are known for commonly used manifolds
 496 such as spheres, hyperbolic spaces, and their product manifolds, future work will investigate how to
 497 extend our method to manifolds whose geodesic distances lack closed-form expressions.
 498

499

6 CONCLUSION

500 In this paper, we address an open problem and propose the RFK algorithm, which reduces the time
 501 complexity from $\mathcal{O}(\nu\omega)$ to $\mathcal{O}(\nu)$. Furthermore, we introduce Radan as an optimizer for product
 502 manifolds. Extensive experiments demonstrate that our algorithm achieves remarkable performance:
 503 on some certain datasets, it runs over 100 times faster than NRK while achieving better clustering
 504 results and lower loss values. Additionally, Radan converges faster than Radam under the RFK loss
 505 with standard hyperparameters. Across almost all datasets, RFK significantly outperforms other
 506 state-of-the-art clustering algorithms in all clustering metrics.
 507

508

7 STATEMENT

510 For the reproducibility of this paper, we have submitted the complete anonymized code, data, and
 511 experiment files with fixed random seeds, as detailed in Appendix G. In addition, large language
 512 models (LLMs) were only used for language polishing.
 513

514

REFERENCES

516 Abdulhady Abas Abdullah, Aram Mahmood Ahmed, Tarik Rashid, Hadi Veisi, Yassin Hussein
 517 Rassul, Bryar Hassan, Polla Fattah, Sabat Abdulhameed Ali, and Ahmed S Shamsaldin. Advanced
 518 clustering techniques for speech signal enhancement: A review and metanalysis of fuzzy c-means,
 519 k-means, and kernel fuzzy c-means methods. *arXiv preprint arXiv:2409.19448*, 2024.

520 Dmitrij V Alekseevskij, Ernest B Vinberg, and Aleksandr S Solodovnikov. Geometry of spaces of
 521 constant curvature. In *Geometry II: Spaces of Constant Curvature*, pp. 1–138. Springer, 1993.

523 Foivos Alimisis and Bart Vandereycken. Geodesic convexity of the symmetric eigenvalue problem
 524 and convergence of steepest descent. *Journal of Optimization Theory and Applications*, 203(1):
 525 920–959, 2024.

526 Horst Alzer and Man Kam Kwong. On young’s inequality. *Journal of Mathematical Analysis and*
 527 *Applications*, 469(2):480–492, 2019.

528 Marc Arnaudon and Frank Nielsen. On approximating the riemannian 1-center. *Computational*
 529 *Geometry*, 46(1):93–104, 2013.

531 Mina Ashizawa, Hiroaki Sasaki, Tomoya Sakai, and Masashi Sugiyama. Least-Squares Log-Density
 532 Gradient Clustering for Riemannian Manifolds. In Aarti Singh and Jerry Zhu (eds.), *Proceed-
 533 ings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54
 534 of *Proceedings of Machine Learning Research*, pp. 537–546. PMLR, 20–22 Apr 2017. URL
 535 <https://proceedings.mlr.press/v54/ashizawa17a.html>.

536 Gregor Bachmann, Gary Becigneul, and Octavian Ganea. Constant curvature graph convolutional
 537 networks. In Hal Daumé III and Aarti Singh (eds.), *Proceedings of the 37th International*
 538 *Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*,
 539 pp. 486–496. PMLR, 13–18 Jul 2020. URL <https://proceedings.mlr.press/v119/bachmann20a.html>.

540 Gary Becigneul and Octavian-Eugen Ganea. Riemannian adaptive optimization methods. In *Inter-
541 national Conference on Learning Representations*, 2019. URL [https://openreview.net/
542 forum?id=r1eiqi09K7](https://openreview.net/forum?id=r1eiqi09K7).

543

544 Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new
545 perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828,
546 2013.

547 James C Bezdek, Robert Ehrlich, and William Full. Fcm: The fuzzy c-means clustering algorithm.
548 *Computers & geosciences*, 10(2-3):191–203, 1984.

549

550 Andreas Bjerregaard, Søren Hauberg, and Anders Krogh. Riemannian generative decoder. In *ICML
551 2025 Generative AI and Biology (GenBio) Workshop*, 2025. URL [https://openreview.net/
552 forum?id=5i4ABK5QQp](https://openreview.net/forum?id=5i4ABK5QQp).

553 Boris Bonev, Max Rietmann, Andrea Paris, Alberto Carpentieri, and Thorsten Kurth. Attention on
554 the sphere. *arXiv preprint arXiv:2505.11157*, 2025.

555 Nicolas Boumal. *An introduction to optimization on smooth manifolds*. Cambridge University Press,
556 2023.

557

558 Louis Capitaine, Jérémie Bigot, Rodolphe Thiébaut, and Robin Genuer. Fréchet random forests
559 for metric space valued regression with non euclidean predictors. *Journal of Machine Learning
560 Research*, 25(355):1–41, 2024.

561 Alex Chen, Philippe Chlenski, Kenneth Munyuza, Antonio Khalil Moretti, Christian A. Naesseth,
562 and Itsik Pe'er. Variational combinatorial sequential monte carlo for bayesian phylogenetics in
563 hyperbolic space. In Yingzhen Li, Stephan Mandt, Shipra Agrawal, and Emtilay Khan (eds.),
564 *Proceedings of The 28th International Conference on Artificial Intelligence and Statistics*, volume
565 258 of *Proceedings of Machine Learning Research*, pp. 2962–2970. PMLR, 03–05 May 2025.
566 URL <https://proceedings.mlr.press/v258/chen25f.html>.

567

568 Xiaojun Chen, Joshua Zhexue Haung, Feiping Nie, Renjie Chen, and Qingyao Wu. A self-balanced
569 min-cut algorithm for image clustering. In *Proceedings of the IEEE International Conference on
570 Computer Vision*, pp. 2061–2069, 2017.

571

572 Philippe Chlenski, Ethan Turok, Antonio Khalil Moretti, and Itsik Pe'er. Fast hyperboloid decision
573 tree algorithms. In *The Twelfth International Conference on Learning Representations*, 2024. URL
<https://openreview.net/forum?id=TTonmgTT9X>.

574

575 Philippe Chlenski, Quentin Chu, Raiyan R. Khan, Kaizhu Du, Antonio Khalil Moretti, and Itsik Pe'er.
576 Mixed-curvature decision trees and random forests. In *Forty-second International Conference on
577 Machine Learning*, 2025a. URL <https://openreview.net/forum?id=wpt1UkP48t>.

578

579 Philippe Chlenski, Kaizhu Du, Dylan Satow, Raiyan R Khan, and Itsik Pe'er. Manify: A python
580 library for learning non-euclidean representations. *arXiv preprint arXiv:2503.09576*, 2025b.

581

582 Jose A Costa and Alfred O Hero. Manifold learning using euclidean k-nearest neighbor graphs
[image processing examples]. In *2004 IEEE International Conference on Acoustics, Speech, and
583 Signal Processing*, volume 3, pp. iii–988. IEEE, 2004.

584

585 {Tim R.} Davidson, Luca Falorsi, Nicola {De Cao}, Thomas Kipf, and {Jakub M.} Tomczak.
586 Hyperspherical variational auto-encoders. In Ricardo Silva, Amir Globerson, and Amir Globerson
587 (eds.), *34th Conference on Uncertainty in Artificial Intelligence 2018, UAI 2018*, 34th Conference
588 on Uncertainty in Artificial Intelligence 2018, UAI 2018, pp. 856–865. Association For Uncertainty
589 in Artificial Intelligence (AUAI), January 2018. 34th Conference on Uncertainty in Artificial
590 Intelligence 2018, UAI 2018 ; Conference date: 06-08-2018 Through 10-08-2018.

591

592 Vinod Kumar Dehariya, Shailendra Kumar Shrivastava, and RC Jain. Clustering of image data set
593 using k-means and fuzzy k-means algorithms. In *2010 International conference on computational
intelligence and communication networks*, pp. 386–391. IEEE, 2010.

594

595 Bhuwan Dhingra, Christopher J Shallue, Mohammad Norouzi, Andrew M Dai, and George E Dahl.
596 Embedding text in hyperbolic spaces. *arXiv preprint arXiv:1806.04313*, 2018.

594 Jiarui Ding and Aviv Regev. Deep generative model embedding of single-cell rna-seq profiles on
 595 hyperspheres and hyperbolic spaces. *Nature communications*, 12(1):2554, 2021.
 596

597 Liang Du, Yunhui Liang, Mian Ilyas Ahmad, and Peng Zhou. K-means clustering based on chebyshev
 598 polynomial graph filtering. In *ICASSP 2024-2024 IEEE International Conference on Acoustics,
 599 Speech and Signal Processing (ICASSP)*, pp. 7175–7179. IEEE, 2024.

600 Ehsan Elhamifar and René Vidal. Sparse subspace clustering: Algorithm, theory, and applications.
 601 *IEEE transactions on pattern analysis and machine intelligence*, 35(11):2765–2781, 2013.
 602

603 Tadashi Fujioka. Noncritical maps on geodesically complete spaces with curvature bounded above.
 604 *Annals of Global Analysis and Geometry*, 62(3):661–677, 2022.

605 Marco Fumero, Luca Cosmo, Simone Melzi, and Emanuele Rodolà. Learning disentangled repre-
 606 sentations via product manifold projection. In *International conference on machine learning*, pp.
 607 3530–3540. PMLR, 2021.

608 Sagar Ghosh and Swagatam Das. Consistent spectral clustering in hyperbolic spaces. *arXiv preprint
 609 arXiv:2409.09304*, 2024.

610 Nefeli Gkouti, Prodromos Malakasiotis, Stavros Toumpis, and Ion Androutsopoulos. Should i try
 611 multiple optimizers when fine-tuning pre-trained transformers for nlp tasks? should i tune their
 612 hyperparameters?, 2024. URL <https://arxiv.org/abs/2402.06948>.

613 Albert Gu, Frederic Sala, Beliz Gunel, and Christopher Ré. Learning mixed-curvature representations
 614 in product spaces. In *International conference on learning representations*, 2018.

615 Neil He, Menglin Yang, and Rex Ying. Lorentzian residual neural networks. In *Proceedings of the
 616 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 1*, pp. 436–447,
 617 2025.

618 Haize Hu, Jianxun Liu, Xiangping Zhang, and Mengge Fang. An effective and adaptable k-means
 619 algorithm for big data cluster analysis. *Pattern Recognition*, 139:109404, 2023.

620 Dong Huang, Chang-Dong Wang, Jian-Sheng Wu, Jian-Huang Lai, and Chee-Keong Kwoh. Ultra-
 621 scalable spectral clustering and ensemble clustering. *IEEE Transactions on Knowledge and Data
 622 Engineering*, 32(6):1212–1226, 2019.

623 Huajie Huang, Bo Liu, Xiaoyu Xue, Jiuxin Cao, and Xinyi Chen. Imbalanced credit card fraud
 624 detection data: A solution based on hybrid neural network and clustering-based undersampling
 625 technique. *Applied Soft Computing*, 154:111368, 2024.

626 Su I Iao, Yidong Zhou, and Hans-Georg Müller. Deep fréchet regression. *Journal of the American
 627 Statistical Association*, (just-accepted):1–30, 2025.

628 Birger Iversen. *Hyperbolic geometry*. Number 25. Cambridge University Press, 1992.

629 Alan Julian Izenman. Introduction to manifold learning. *Wiley Interdisciplinary Reviews: Computa-
 630 tional Statistics*, 4(5):439–446, 2012.

631 Vladimir Jaćimović and Aladin Crnkić. Clustering in hyperbolic balls. *arXiv preprint
 632 arXiv:2501.19247*, 2025.

633 Francisco Jos, AuthorNameForHeading-FJ Herranz, and A Ballesteros. Spaces of constant curvature.
 634 In *none*, 1967.

635 Raiyan R. Khan, Philippe Chlenski, and Itsik Pe’er. Hyperbolic genome embeddings. In
 636 *The Thirteenth International Conference on Learning Representations*, 2025. URL <https://openreview.net/forum?id=NkGDNM8LB0>.

637 Max Kochurov, Rasul Karimov, and Serge Kozlukov. Goopt: Riemannian optimization in pytorch,
 638 2020. URL <https://arxiv.org/abs/2005.02819>.

648 Daniel Krasnov, Dresya Davis, Keiran Malott, Yiting Chen, Xiaoping Shi, and Augustine Wong.
 649 Fuzzy c-means clustering: A review of applications in breast cancer detection. *Entropy*, 25(7):
 650 1021, 2023.

651

652 Hongzong Li and Jun Wang. From soft clustering to hard clustering: A collaborative annealing fuzzy
 653 c-means algorithm. *IEEE Transactions on Fuzzy Systems*, 32(3):1181–1194, 2023.

654

655 Jun Li, Jinpeng Wang, Chaolei Tan, Niu Lian, Long Chen, Yaowei Wang, Min Zhang, Shu-Tao Xia,
 656 and Bin Chen. Enhancing partially relevant video retrieval with hyperbolic learning. In *Proceedings*
 657 of the IEEE/CVF International Conference on Computer Vision (ICCV), pp. 23074–23084, October
 658 2025.

659 Aristidis Likas, Nikos Vlassis, and Jakob J Verbeek. The global k-means clustering algorithm. *Pattern
 660 recognition*, 36(2):451–461, 2003.

661

662 Fang-Yu Lin, Bing Bai, Kun Bai, Yazhou Ren, Peng Zhao, and Zenglin Xu. Contrastive multi-view
 663 hyperbolic hierarchical clustering. In *International Joint Conference on Artificial Intelligence*,
 664 2022. URL <https://api.semanticscholar.org/CorpusID:248525138>.

665

666 Guangcan Liu, Zhouchen Lin, Shuicheng Yan, Ju Sun, Yong Yu, and Yi Ma. Robust recovery
 667 of subspace structures by low-rank representation. *IEEE transactions on pattern analysis and
 machine intelligence*, 35(1):171–184, 2012.

668

669 Aaron Lou, Isay Katsman, Qingxuan Jiang, Serge Belongie, Ser-Nam Lim, and Christopher De Sa.
 670 Differentiating through the fréchet mean. In *International conference on machine learning*, pp.
 671 6393–6403. PMLR, 2020.

672

673 Yui Man Lui. Human gesture recognition on product manifolds. *The Journal of Machine Learning
 Research*, 13(1):3297–3321, 2012.

674

675 Paolo Mandica, Luca Franco, Konstantinos Kallidromitis, Suzanne Petryk, and Fabio Galasso.
 676 Hyperbolic learning with multimodal large language models. In *European Conference on Computer
 Vision*, pp. 382–398. Springer, 2024a.

677

678 Paolo Mandica, Luca Franco, Konstantinos Kallidromitis, Suzanne Petryk, and Fabio Galasso.
 679 Hyperbolic learning with multimodal large language models. In *European Conference on Computer
 Vision*, pp. 382–398. Springer, 2024b.

680

681 Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and
 682 projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

683

684 Daniel McNeela, Frederic Sala, and Anthony Gitter. Mixed-curvature representation learning for
 685 biological pathway graphs. In *2023 ICML Workshop on Computational Biology, Honolulu, Hawaii,
 USA*, 2023.

686

687 Nina Miolane, Nicolas Guigui, Alice Le Brigant, Johan Mathe, Benjamin Hou, Yann Thanwerdas,
 688 Stefan Heyder, Olivier Peltre, Niklas Koep, Hadi Zaatiti, et al. Geomstats: A python package for
 689 riemannian geometry in machine learning. *Journal of Machine Learning Research*, 21(223):1–9,
 690 2020.

691

692 Gal Mishne, Zhengchao Wan, Yusu Wang, and Sheng Yang. The numerical stability of hyperbolic
 693 representation learning. In *International Conference on Machine Learning*, pp. 24925–24949.
 694 PMLR, 2023.

695

696 Mahesh Chandra Mukkamala and Matthias Hein. Variants of rmsprop and adagrad with logarithmic
 697 regret bounds. In *International conference on machine learning*, pp. 2545–2553. PMLR, 2017.

698

699 Shi Na, Liu Xumin, and Guan Yong. Research on k-means clustering algorithm: An improved
 700 k-means clustering algorithm. In *2010 Third International Symposium on intelligent information
 technology and security informatics*, pp. 63–67. Ieee, 2010.

701

Maximillian Nickel and Douwe Kiela. Poincaré embeddings for learning hierarchical representations.
Advances in neural information processing systems, 30, 2017.

702 Feiping Nie, Jitao Lu, Danyang Wu, Rong Wang, and Xuelong Li. A novel normalized-cut solver with
 703 nearest neighbor hierarchical initialization. *IEEE Transactions on Pattern Analysis and Machine*
 704 *Intelligence*, 46(1):659–666, 2023.

705 Feiping Nie, Runxin Zhang, Yu Duan, and Rong Wang. Unconstrained fuzzy c-means based
 706 on entropy regularization: An equivalent model. *IEEE Transactions on Knowledge and Data*
 707 *Engineering*, 2024.

708 Huan Ren, Wenfei Yang, Xiang Liu, Shifeng Zhang, and Tianzhu Zhang. Learning shape-
 709 independent transformation via spherical representations for category-level object pose estima-
 710 tion. In *The Thirteenth International Conference on Learning Representations*, 2025. URL
 711 <https://openreview.net/forum?id=D4xztKoz0Y>.

712 Frederic Sala, Chris De Sa, Albert Gu, and Christopher Ré. Representation tradeoffs for hyperbolic
 713 embeddings. In *International conference on machine learning*, pp. 4460–4469. PMLR, 2018.

714 Kristina P Sinaga and Miin-Shen Yang. Unsupervised k-means clustering algorithm. *IEEE access*, 8:
 715 80716–80727, 2020.

716 Aditya Sinha, Siqi Zeng, Makoto Yamada, and Han Zhao. Learning structured representations with
 717 hyperbolic embeddings. *Advances in Neural Information Processing Systems*, 37:91220–91259,
 718 2024.

719 Ondrej Skopek, Octavian-Eugen Ganea, and Gary Bécigneul. Mixed-curvature variational au-
 720 toencoders. In *8th international conference on learning representations (ICLR 2020)(virtual)*.
 721 International Conference on Learning Representations, 2020.

722 Raghad Subbarao and Peter Meer. Nonlinear mean shift over riemannian manifolds. *International*
 723 *journal of computer vision*, 84(1):1–20, 2009.

724 R Suganya and R Shanthi. Fuzzy c-means algorithm-a review. *International Journal of Scientific and*
 725 *Research Publications*, 2(11):1, 2012.

726 Li Sun, Zhongbao Zhang, Junda Ye, Hao Peng, Jiawei Zhang, Sen Su, and Philip S Yu. A self-
 727 supervised mixed-curvature graph neural network. In *Proceedings of the AAAI Conference on*
 728 *Artificial Intelligence*, volume 36, pp. 4146–4155, 2022.

729 Li Sun, Feiyang Wang, Junda Ye, Hao Peng, and S Yu Philip. Congregate: Contrastive graph
 730 clustering in curvature spaces. In *IJCAI*, pp. 2296–2305, 2023a.

731 Li Sun, Feiyang Wang, Junda Ye, Hao Peng, and Philip S Yu. Contrastive graph clustering in
 732 curvature spaces. *arXiv preprint arXiv:2305.03555*, 2023b.

733 Puoya Tabaghi, Chao Pan, Eli Chien, Jianhao Peng, and Olgica Milenkovic. Linear classifiers in
 734 product space forms. *arXiv preprint arXiv:2102.10204*, 2021.

735 Mariano Tepper, Anirvan M Sengupta, and Dmitri Chklovskii. Clustering is semidefinitely not that
 736 hard: Nonnegative sdp for manifold disentangling. *Journal of Machine Learning Research*, 19(82):
 737 1–30, 2018.

738 Michael Tsamparlis. Lorentz inner product and lorentz tensors. In *Solved Problems and Systematic*
 739 *Introduction to Special Relativity*, pp. 69–96. Springer, 2024.

740 Shen Wang, Xiaokai Wei, Cicero Nogueira Nogueira dos Santos, Zhiguo Wang, Ramesh Nallapati,
 741 Andrew Arnold, Bing Xiang, Philip S Yu, and Isabel F Cruz. Mixed-curvature multi-relational
 742 graph neural network for knowledge graph completion. In *Proceedings of the web conference*
 743 2021, pp. 1761–1771, 2021.

744 Shuo Wang, Shunyang Huang, Jinghui Yuan, Zhixiang Shen, and zhao kang. Cooperation of experts:
 745 Fusing heterogeneous information with large margin. In *Forty-second International Conference on*
 746 *Machine Learning*, 2025. URL <https://openreview.net/forum?id=1Z18hxItYI>.

747 Marshall Whittlesey. *Spherical geometry and its applications*. Chapman and Hall/CRC, 2019.

756 Chengmao Wu and Sifan Pan. Fuzzy c-poincaré fréchet means clustering in hyperbolic space. *Expert*
 757 *Systems with Applications*, pp. 128245, 2025a.
 758

759 Chengmao Wu and Sifan Pan. Fuzzy c-poincaré fréchet means clustering in hyperbolic space. *Expert*
 760 *Systems with Applications*, 288:128245, 2025b. ISSN 0957-4174. doi: <https://doi.org/10.1016/j.eswa.2025.128245>. URL <https://www.sciencedirect.com/science/article/pii/S0957417425018640>.
 761
 762

763 Fangyuan Xie, Jinghui Yuan, Feiping Nie, and Xuelong Li. Dual-bounded nonlinear optimal transport
 764 for size constrained min cut clustering. *arXiv preprint arXiv:2501.18143*, 2025.
 765

766 Xingyu Xie, Pan Zhou, Huan Li, Zhouchen Lin, and Shuicheng Yan. Adan: Adaptive nesterov
 767 momentum algorithm for faster optimizing deep models. *IEEE Transactions on Pattern Analysis*
 768 *and Machine Intelligence*, 46(12):9508–9520, 2024.
 769

770 Jinglin Xu, Junwei Han, Kai Xiong, and Feiping Nie. Robust and sparse fuzzy k-means clustering.
 771 In *IJCAI*, pp. 2224–2230, 2016.
 772

773 Zhirong Xu, Shiyang Wen, Junshan Wang, Guojun Liu, Liang Wang, Zhi Yang, Lei Ding, Yan Zhang,
 774 Di Zhang, Jian Xu, et al. Amcad: adaptive mixed-curvature representation based advertisement
 775 retrieval system. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, pp.
 776 3439–3452. IEEE, 2022.
 777

778 Menglin Yang, Min Zhou, Rex Ying, Yankai Chen, and Irwin King. Hyperbolic representation
 779 learning: Revisiting and advancing. In *International Conference on Machine Learning*, pp.
 780 39639–39659. PMLR, 2023.
 781

782 Kisung You. Gradient of squared distance on a Riemannian manifold. URL https://kisungyou.com/Blog/blog_004_GradientSquaredDistance.html.
 783

784 Jinghui Yuan, Chusheng Zeng, Fangyuan Xie, Zhe Cao, Mulin Chen, Rong Wang, Feiping Nie,
 785 and Yuan Yuan. Doubly stochastic adaptive neighbors clustering via the marcus mapping. *arXiv*
 786 *preprint arXiv:2408.02932*, 2024.
 787

788 Jinghui Yuan, Hao Chen, Renwei Luo, and Feiping Nie. A margin-maximizing fine-grained ensemble
 789 method. In *ICASSP 2025-2025 IEEE International Conference on Acoustics, Speech and Signal*
 790 *Processing (ICASSP)*, pp. 1–5. IEEE, 2025a.
 791

792 Jinghui Yuan, Fangyuan Xie, Feiping Nie, and Xuelong Li. Riemannian optimization on relaxed
 793 indicator matrix manifold. *arXiv preprint arXiv:2503.20505*, 2025b.
 794

795 Dongdong Yue, Simone Baldi, Jinde Cao, and Bart De Schutter. Distributed adaptive optimization
 796 with weight-balancing. *IEEE Transactions on Automatic Control*, 67(4):2068–2075, 2021.
 797

798 Kun Zhao, Azadeh Alavi, Arnold Wiliem, and Brian C Lovell. Efficient clustering on riemannian
 799 manifolds: A kernelised random projection approach. *Pattern Recognition*, 51:333–345, 2016.
 800

801 Guo Zhong and Chi-Man Pun. Improved normalized cut for multi-view clustering. *IEEE Transactions*
 802 *on Pattern Analysis and Machine Intelligence*, 44(12):10244–10251, 2021.
 803

804 Pan Zhou, Xingyu Xie, Zhouchen Lin, Kim-Chuan Toh, and Shuicheng Yan. Win: Weight-decay-
 805 integrated nesterov acceleration for faster network training. *Journal of Machine Learning Research*,
 806 25(83):1–74, 2024.
 807

808 Yidong Zhou, Su I Iao, and Hans-Georg Müller. Fréchet geodesic boosting. In *Advances in Neural*
 809 *Information Processing Systems*, 2025. in press.
 810
 811
 812
 813
 814
 815
 816
 817
 818
 819
 820
 821
 822
 823
 824
 825
 826
 827
 828
 829
 830
 831
 832
 833
 834
 835
 836
 837
 838
 839
 840
 841
 842
 843
 844
 845
 846
 847
 848
 849
 850
 851
 852
 853
 854
 855
 856
 857
 858
 859
 860
 861
 862
 863
 864
 865
 866
 867
 868
 869
 870
 871
 872
 873
 874
 875
 876
 877
 878
 879
 880
 881
 882
 883
 884
 885
 886
 887
 888
 889
 890
 891
 892
 893
 894
 895
 896
 897
 898
 899
 900
 901
 902
 903
 904
 905
 906
 907
 908
 909
 910

810	CONTENTS	
811		
812		
813	1 Introduction	1
814		
815	2 Preliminaries	2
816	2.1 Notations	2
817	2.2 Constant-curvature Spaces and Product Manifolds	2
818	2.3 K-Means and Fuzzy K-Means	3
819		
820		
821	3 Our proposed method	3
822	3.1 Naive Extension of K-Means	3
823	3.2 Riemannian Fuzzy K-Means	4
824	3.3 Radan on Product Manifolds	5
825	3.4 Calculate Riemannian Gradient	6
826		
827		
828		
829	4 Experiments	7
830	4.1 Datasets	7
831	4.2 Experiments Setup	7
832	4.2.1 Experiment Setup for Q1	7
833	4.2.2 Experiment Setup for Q2	8
834	4.2.3 Experiment Setup for Q3	8
835	4.3 Experiments Result	8
836	4.3.1 Experiment Result for Q1	8
837	4.3.2 Experiment Result for Q2	8
838	4.3.3 Experiment Result for Q3	9
839		
840		
841		
842		
843		
844	5 Limitations	10
845		
846	6 Conclusion	10
847		
848		
849	7 Statement	10
850		
851	Appendices	18
852		
853	A Proofs of Theorems	18
854		
855	A.1 Proof of Theorem 3.1	18
856	A.1.1 Assumptions	18
857	A.1.2 Proof Details	19
858	A.1.3 Proof of Lemma	24
859	A.2 Proof of Theorem 3.2	25
860	A.3 Proof of Theorem 3.3	26
861	A.3.1 Proof Details	26
862		
863		

864	A.3.2 Proof of Lemma	27
865		
866		
867	B Notations	28
868		
869	C Related Work about Clustering on Manifold	28
870		
871	D Background	29
872		
873	D.1 What kind of data do we learn?	29
874	D.2 Difference from manifold learning	29
875	D.3 Basic principles of Riemannian machine learning	29
876	D.4 Miscellaneous questions	29
877		
878		
879	E Details of the Experimental Setup	30
880		
881	E.1 Datasets Description	30
882	E.2 Experiment 3 Setup	31
883	E.2.1 Benchmark Clustering Algorithms	31
884	E.2.2 Clustering Accuracy (ACC)	32
885	E.2.3 Normalized Mutual Information (NMI)	32
886	E.2.4 Adjusted Rand Index (ARI)	33
887	E.2.5 F1 Score	33
888	E.2.6 Purity	33
889		
890		
891		
892	F Additional Experimental Results	34
893		
894	F.1 Experimental 3 Results	34
895	F.2 Sensitivity Analysis	34
896	F.2.1 Sensitivity Analysis of m	34
897	F.2.2 Sensitivity analysis of random initialization	35
898	F.2.3 Sensitivity Analysis of the Number of Cluster Centers	36
899		
900		
901		
902	G Run and Reference Code	37
903		
904	G.1 Run the Code	37
905	G.2 Replication Statement	37
906	G.3 Code of Riemannian Fuzzy K-Means	38
907	G.4 Code of Riemannian Adan	42
908		
909		
910		
911		
912		
913		
914		
915		
916		
917		

918 A PROOFS OF THEOREMS
919920 A.1 PROOF OF THEOREM 3.1
921922 In this section, we will prove the Theorem 3.1, which provides the regret bound for the Radan
923 algorithm. Before proceeding, we make the following assumption. **These assumptions are all**
924 **standard assumptions in Riemannian optimization.**925 A.1.1 ASSUMPTIONS
926928 Assumption 1. In the optimization problem solved by the Radan algorithm, the feasible domain is
929 geodesically bounded (Fujioka, 2022). That is, for any geodesic $\gamma(t)$ within the feasible domain D ,
930 its length satisfies that:

931
$$\int_{t_0}^{\infty} \|\gamma(t)\|_{y_t} dt \leq D_{\infty} \quad (13)$$

932

933 Furthermore, let Log denote the logarithmic map. Then we have the inequality
934

935
$$\|\text{Log}_{y_t}(y)\|_{y_t} \leq D_{\infty} \quad (14)$$

936

937 which we state as Lemma 1, and will prove later in the paper.

938 Assumption 2. We assume that in the Riemannian optimization problem solved by the Radan
939 algorithm, the curvature ζ of the Riemannian manifold on which the constraints are defined is
940 bounded. Specifically, in the Riemannian cosine law (Arnaudon & Nielsen, 2013):
941

942
$$d^2(y_{t+1}, y^*) \leq d^2(y_{t+1}, y_t) \cdot \zeta(\kappa, c) + d^2(y_t, y^*) - 2d(y_{t+1}, y_t)d(y_t, y^*) \cos A \quad (15)$$

943

944 For general spherical, hyperbolic, and their product manifolds, the curvature function $\zeta(\kappa, c)$ also
945 admits a unified formulation:

946
$$\zeta(\kappa, c) = \begin{cases} \frac{\sqrt{|\kappa|} c}{\tanh(\sqrt{|\kappa|} c)}, & \kappa < 0, \\ \frac{\tan(\sqrt{\kappa} c)}{\sqrt{\kappa} c}, & \kappa > 0, \end{cases} \quad (16)$$

947
948
949
950

951 which is a function of curvature and distance, and is commonly used in the convergence analysis of
952 Riemannian optimization algorithms. Here, c denotes the distance function, and it satisfies $c \leq D_{\infty}$.
953 The function $\zeta(\kappa, c)$ is assumed to be bounded (Becigneul & Ganea, 2019).955 Assumption 3. We assume that the gradient is bounded, i.e., the norm of the gradient at y_t
956 satisfies $\|g_t\|_{y_t} \leq G$, which is a standard assumption commonly used in the proof of the theorem.958 Assumption 4. Let the parallel transport of the vector m_{t-1} from y_{t-1} to y_t be denoted by
959 $\varphi_{y_{t-1} \rightarrow y_t}(m_{t-1})$, which we abbreviate as $\varphi(m_{t-1})$ when there is no ambiguity. We assume that the
960 parallel transport preserves the inner product of the vector, i.e.,

961
$$\langle m_{t-1}, v_{t-1} \rangle_{y_{t-1}} = \langle \varphi(m_{t-1}), \varphi(v_{t-1}) \rangle_{y_t}. \quad (17)$$

962

963 Assumption 5. We assume that in the Riemannian optimization problem solved by the Radan
964 algorithm, the objective function is geodesically convex (Alimisis & Vandereycken, 2024). That is,
965 for any $p, q \in \mathcal{M}$ and $t \in [0, 1]$, the following holds:

967
$$f(\gamma(t)) \leq (1-t)f(p) + tf(q), \quad (18)$$

968

969 where γ is the geodesic connecting p and q . Furthermore, it can be shown that

970
$$f(y_t) - f(y^*) \leq \langle -g_t, \text{Log}_{y_t}(y^*) \rangle_{y_t}, \quad (19)$$

971

and we will provide a proof of this in Lemma 2.

972 A.1.2 PROOF DETAILS
973

974 We now present Theorem 3.1 along with its proof.

975 **Theorem A.1.** *Let y_t be the sequence generated by the Radan algorithm. Under the standard
976 assumptions, the regret bound R_T satisfies the following.*

977
$$978 R_T \leq \frac{\eta\sqrt{T}\sqrt{1+\log T}G}{(1-\beta_1)^2} \left[\frac{\zeta(\kappa, c)(3-2\beta_1)G}{2(1-\beta_1)} + \frac{\beta_1}{\sqrt{1-\beta_{3t}}(1-\delta)} \right] \\ 979 + \frac{GD_\infty^2(1+2\beta_2)\sqrt{T}}{2(1-\beta_1)\cdot\eta} + \sum_{t=1}^T \frac{4D_\infty^2G^2\beta_{2t}}{1-\beta_1} + \sum_{t=1}^T \frac{\sqrt{t}(1+2\beta_2)GD_\infty^2\beta_{1t}}{\eta(1-\beta_1)} \quad (20)$$

980
981
982

983 *Proof.* According to the Radan algorithm, the update from step t to step $t+1$ is as follows:

984
$$985 \left\{ \begin{array}{l} m_t = \beta_{1t}\varphi(m_{t-1}) + (1-\beta_{1t})g_t \\ 986 v_t = \beta_{2t}\varphi(v_{t-1}) + (1-\beta_{2t})(g_t - \varphi(g_{t-1})) \\ 987 z_t = g_t + \beta_{2t}(g_t - \varphi(g_{t-1})) \\ 988 n_t = \beta_{3t}n_{t-1} + (1-\beta_{3t})\|z_t\|_{y_t}^2 \\ 989 u_t = m_t + \beta_{2t}v_t \\ 990 \alpha_t = \frac{\eta_t}{\sqrt{n_t}} + \epsilon_t \\ 991 y_{t+1} = \text{Exp}_{y_t}(-\alpha_t u_t) \\ 992 \end{array} \right. \quad (21)$$

993

994 Here, φ is a parallel translation, which is assumed to preserve the inner product. $\text{Exp}(\cdot)$ is the
995 exponential map, and $\text{Log}(\cdot)$ is the logarithmic map. Also, $\beta_{1t} = \beta_1 \cdot f_1(t)$, where $f_1(t)$ is a decay
996 function, and $\beta_{2t} = \beta_2 \cdot f_2(t)$, where $f_2(t)$ is a decay function.997 Given $y_{t+1} = \text{Exp}(-\alpha_t u_t)$, according to the cosine theorem on the manifold, we have:

998
$$999 a^2 \leq b^2\zeta(\kappa, c) + c^2 - 2bc \cos A \quad (22)$$

1000 Based on the assumption of bounded curvature, we have that $\zeta(\kappa, c)$ is bounded, let:

1001
$$1002 a = d(y_{t+1}, y^*), \quad b = d(y_{t+1}, y_t), \quad c = d(y_t, y^*). \quad (23)$$

1003 Then, that is:

1004
$$1005 d^2(y_{t+1}, y^*) \leq d^2(y_{t+1}, y_t) \cdot \zeta(\kappa, c) + d^2(y_t, y^*) - 2d(y_{t+1}, y_t)d(y_t, y^*) \cos A \quad (24)$$

1006 According to the definition of $\cos A$, we have:

1007
$$1008 d(y_{t+1}, y_t)d(y_t, y^*) \cos A = \langle \text{Log}_{y_t}(y_{t+1}), \text{Log}_{y_t}(y^*) \rangle_{y_t} = -\alpha_t \langle u_t, \text{Log}_{y_t}(y^*) \rangle_{y_t} \quad (25)$$

1009 Substituting this into the above Equation (24), we get:

1010
$$1011 2d(y_{t+1}, y_t)d(y_t, y^*) \cos A = -2\alpha_t \langle u_t, \text{Log}_{y_t}(y^*) \rangle_{y_t} \\ 1012 \leq d^2(y_t, y^*) - d^2(y_{t+1}, y^*) + \zeta(\kappa, c) \cdot d^2(y_{t+1}, y_t) \\ 1013 = d^2(y_t, y^*) - d^2(y_{t+1}, y^*) + \zeta(\kappa, c)\alpha_t^2\|u_t\|_{y_t}^2 \quad (26)$$

1014 By rearranging the terms and dividing both sides by α_t , we obtain the following expression:

1015
$$1016 \langle -u_t, \text{Log}_{y_t}(y^*) \rangle_{y_t} \leq \frac{1}{2\alpha_t} [d^2(y_t, y^*) - d^2(y_{t+1}, y^*)] + \frac{\zeta(\kappa, c)\alpha_t\|u_t\|_{y_t}^2}{2} \quad (27)$$

1017

1018 Since $u_t = m_t + \beta_{2t}v_t$, then:

1019
$$1020 \langle -m_t, \text{Log}_{y_t}(y^*) \rangle_{y_t} \leq \frac{1}{2\alpha_t} [d^2(y_t, y^*) - d^2(y_{t+1}, y^*)] + \frac{\zeta(\kappa, c)\alpha_t\|u_t\|_{y_t}^2}{2} + \beta_{2t} \langle v_t, \text{Log}_{y_t}(y^*) \rangle_{y_t} \quad (28)$$

1021

1022 Also, because $m_t = \beta_{1t}\varphi(m_{t-1}) + (1-\beta_{1t})g_t$, finally we have:

1023
$$1024 \langle -(1-\beta_{1t})g_t, \text{Log}_{y_t}(y^*) \rangle_{y_t} \leq \frac{1}{2\alpha_t} [d^2(y_t, y^*) - d^2(y_{t+1}, y^*)] + \frac{\zeta(\kappa, c)\alpha_t\|u_t\|_{y_t}^2}{2} \\ 1025 + \beta_{2t} \langle v_t, \text{Log}_{y_t}(y^*) \rangle_{y_t} + \beta_{1t} \langle \varphi(m_{t-1}), \text{Log}_{y_t}(y^*) \rangle_{y_t} \quad (29)$$

Dividing both sides of the equation by $(1 - \beta_{1t})$, we get the following expression:

$$\begin{aligned} \langle -g_t, \text{Log}_{y_t}(y^*) \rangle_{y_t} &\leq \frac{1}{2\alpha_t(1 - \beta_{1t})} [d^2(y_t, y^*) - d^2(y_{t+1}, y^*)] + \frac{\zeta(\kappa, c) \cdot \alpha_t}{2(1 - \beta_{1t})} \|u_t\|_{y_t}^2 \\ &\quad + \frac{\beta_{2t}}{(1 - \beta_{1t})} \langle v_t, \text{Log}_{y_t}(y^*) \rangle_{y_t} + \frac{\beta_{1t}}{1 - \beta_{1t}} \langle \varphi(m_{t-1}), \text{Log}_{y_t}(y^*) \rangle_{y_t} \end{aligned} \quad (30)$$

Since $f(x)$ is geodesically convex, according to Lemma 2, we have the following:

$$f(y_t) - f(y^*) \leq \langle -g_t, \text{Log}_{y_t}(y^*) \rangle_{y_t} \quad (31)$$

The regret bound is:

$$\begin{aligned} R_T &= \sum_{t=1}^T f(y_t) - f(y^*) \leq \sum_{t=1}^T \langle -g_t, \text{Log}_{y_t}(y^*) \rangle_{y_t} \\ &\leq \underbrace{\sum_{t=1}^T \frac{1}{2\alpha_t(1 - \beta_{1t})} [d^2(y_t, y^*) - d^2(y_{t+1}, y^*)]}_{B_1} + \underbrace{\sum_{t=1}^T \frac{\zeta(\kappa, c) \alpha_t}{2(1 - \beta_{1t})} \|u_t\|_{y_t}^2}_{B_2} \\ &\quad + \underbrace{\sum_{t=1}^T \frac{\beta_{2t}}{(1 - \beta_{1t})} \langle v_t, \text{Log}_{y_t}(y^*) \rangle_{y_t}}_{B_3} + \underbrace{\sum_{t=1}^T \frac{\beta_{1t}}{1 - \beta_{1t}} \langle \varphi(m_{t-1}), \text{Log}_{y_t}(y^*) \rangle_{y_t}}_{B_4} \end{aligned} \quad (32)$$

For B_1 : First, we estimate the B_1 term and identify its upper bound.

$$\begin{aligned} B_1 &\leq \frac{1}{2(1 - \beta_1)} \left[\sum_{t=1}^T \left(\frac{1}{\alpha_t} d^2(y_t, y^*) - \frac{1}{\alpha_t} d^2(y_{t+1}, y^*) \right) \right] \\ &= \frac{1}{2(1 - \beta_1)} \left[\sum_{t=2}^T \left(\frac{1}{\alpha_t} - \frac{1}{\alpha_{t-1}} \right) d^2(y_t, y^*) + \frac{1}{\alpha_1} d^2(y_1, y^*) - \frac{1}{\alpha_T} d^2(y_{T+1}, y^*) \right] \\ &\leq \frac{1}{2(1 - \beta_1)} \cdot \sum_{t=2}^T \left(\frac{1}{\alpha_t} - \frac{1}{\alpha_{t-1}} \right) D_\infty^2 + \frac{1}{\alpha_1} D_\infty^2 \\ &= \frac{1}{2(1 - \beta_1)} \cdot \frac{1}{\alpha_T} D_\infty^2 - \frac{1}{\alpha_1} D_\infty^2 + \frac{1}{\alpha_1} D_\infty^2 \\ &= \frac{1}{2(1 - \beta_1) \alpha_T} D_\infty^2 \\ &= \frac{D_\infty^2}{2(1 - \beta_1) \cdot \eta_T} \sqrt{n_T} \quad (\text{where } \eta_T = \frac{\eta}{\sqrt{T}}) \end{aligned} \quad (33)$$

The first inequality follows from the fact that $\beta_{1t} = \beta_1 f_1(t)$, which decays term by term. Therefore, $\frac{1}{1 - \beta_{1t}} \leq \frac{1}{1 - \beta_1}$. The second inequality follows from the assumption that the feasible domain is bounded, i.e.,

$$d(y_t, y^*) \leq \sup_{x \in D} d(x, y^*) \leq D_\infty. \quad (34)$$

The mathematical logic behind the second inequality also includes $\frac{1}{\alpha_t} \geq \frac{1}{\alpha_{t-1}}$, which is derived from $\beta_{3t} = 1 - \frac{1}{t}$.

For B_4 : Next, we provide the upper bound for the fourth term B_4 . By Young's inequality (Alzer & Kwong, 2019), after making simple transformations, we can obtain the following expression:

$$\langle \varphi(m_{t-1}), \text{Log}_{y_t}(y^*) \rangle_{y_t} \leq \underbrace{\frac{\eta_t}{\sqrt{n_t}} \|\varphi(m_{t-1})\|_{y_t}^2}_{B_{41}} + \underbrace{\frac{\sqrt{n_t}}{\eta_t} \|\text{Log}_{y_t}(y^*)\|_{y_t}^2}_{B_{42}} \quad (35)$$

Considering B_{41} : Since φ preserves the inner product, we can obtain the following equality:

$$\|\varphi(m_{t-1})\|_{y_t}^2 = \langle \varphi(m_{t-1}), \varphi(m_{t-1}) \rangle_{y_t} = \langle m_{t-1}, m_{t-1} \rangle_{y_{t-1}} = \|m_{t-1}\|_{y_{t-1}}^2. \quad (36)$$

1080 Furthermore, we can perform an equivalence transformation on B_{41} .
 1081

$$1082 \sum_{t=1}^T \frac{\eta_t}{\sqrt{n_t}} \|\varphi(m_{t-1})\|_{y_t}^2 = \sum_{t=1}^T \frac{\eta_t}{\sqrt{n_t}} \|m_{t-1}\|_{y_{t-1}}^2 \quad (37)$$

1085 Since $\|g_{t+1}\|_{y_{t+1}}$ is bounded, it is evident that $\|m_{t+1}\|_{y_{t+1}}$ is also bounded. Therefore, to prove that
 1086 B_{41} is bounded, it suffices to show that $\sum_{t=1}^T \frac{\eta_t}{\sqrt{n_t}} \|m_t\|_{y_t}^2$ is bounded.
 1087

1088 Since $m_t = \beta_{1t}\varphi(m_{t-1}) + (1 - \beta_{1t})g_t$, by using the recurrence relation and mathematical induction,
 1089 it can be proven that:

$$1090 \begin{cases} m_1 = \beta_{11}\varphi(m_0) + (1 - \beta_{11})g_1 = (1 - \beta_{11})g_1 \\ m_2 = \beta_{12}\varphi(m_1) + (1 - \beta_{12})g_2 = \beta_{12}(1 - \beta_{11})\varphi(g_1) + (1 - \beta_{12})g_2 \\ m_3 = \beta_{13}\varphi(m_2) + (1 - \beta_{13})g_3 = \beta_{12}\beta_{13}(1 - \beta_{11})\varphi(g_1) + (1 - \beta_{12})\beta_{13}\varphi(g_2) + (1 - \beta_{13})g_3 \\ \vdots \\ m_t = \sum_{j=1}^t (1 - \beta_{1j}) \left(\prod_{k=1}^{t-j} \beta_{1,(t-k+1)} \right) \varphi(g_j) \end{cases} \quad (38)$$

1099 According to Lemma 3, using the inequality
 1100

$$1101 \left\| \sum_{i=1}^n a_i p_i \right\|^2 \leq \left(\sum_{i=1}^n a_i \right) \left(\sum_{i=1}^n a_i \|p_i\|^2 \right), \quad (39)$$

1104 we can derive the following.
 1105

$$1106 \begin{aligned} \|m_t\|_{y_t}^2 &= \left\| \sum_{j=1}^t (1 - \beta_{1j}) \left(\prod_{k=1}^{t-j} \beta_{1,(t-k+1)} \right) \varphi(g_j) \right\|_{y_t}^2 \\ 1107 &\leq \left(\sum_{j=1}^t (1 - \beta_{1j}) \prod_{k=1}^{t-j} \beta_{1,(t-k+1)} \right) \left(\sum_{j=1}^t (1 - \beta_{1j}) \prod_{k=1}^{t-j} \beta_{1,(t-k+1)} \cdot \|g_j\|_{y_j}^2 \right) \\ 1108 &\leq \left(\sum_{j=1}^t (1 - \beta_{1j}) \beta_1^{t-j} \right) \left(\sum_{j=1}^t (1 - \beta_{1j}) \beta_1^{t-j} \|g_j\|_{y_j}^2 \right) \end{aligned} \quad (40)$$

1116 Using the formula for the sum of a geometric series combined with the fact that $\beta_1 < 1$:

$$1117 1 - \beta_{1j} < 1, \quad \sum_{j=1}^t \beta_1^{t-j} = \beta_1^t \sum_{j=1}^t \beta_1^{-j} = \beta_1^t \cdot \frac{\beta_1^{-1}(1 - \beta_1^{-t})}{1 - \beta_1^{-1}} \leq \frac{1}{1 - \beta_1} \quad (41)$$

1120 we can proceed to derive the desired result.
 1121

$$1122 \|m_t\|_{y_t}^2 \leq \left(\sum_{j=1}^t (1 - \beta_{1j}) \beta_1^{t-j} \right) \left(\sum_{j=1}^t (1 - \beta_{1j}) \beta_1^{t-j} \|g_j\|_{y_j}^2 \right) \leq \frac{1}{1 - \beta_1} \sum_{j=1}^t \beta_1^{t-j} \|g_j\|_{y_j}^2 \quad (42)$$

1126 For n_t , because $n_t = \beta_{3t}n_{t-1} + (1 - \beta_{3t})\|z_t\|_{y_t}^2$, a similar discussion still applies:

$$1127 \begin{cases} n_1 = \beta_{31} \cdot n_0 + (1 - \beta_{31})\|z_1\|_{y_1}^2 = \beta_{31} \cdot 0 + (1 - \beta_{31})\|z_1\|_{y_1}^2 \\ n_2 = \beta_{32}(1 - \beta_{32})\|z_1\|_{y_1}^2 + (1 - \beta_{32})\|z_2\|_{y_2}^2 \\ \vdots \\ n_t = \frac{1}{t} \sum_{j=1}^t \|z_j\|_{y_j}^2 \end{cases} \quad (43)$$

1134 For z_j , because of following:
 1135

$$1136 \quad z_j = g_j + \beta_{2t}(g_j - \varphi(g_{j-1})) = (1 + \beta_{2t})g_j - \beta_{2t}\varphi(g_{j-1}) \quad (44)$$

1137 we have that:
 1138

$$1139 \quad \|z_j\|_{y_j} \geq (1 + \beta_{2t})\|g_j\|_{y_j} - \beta_{2t}\|\varphi(g_{j-1})\|_{y_j} = \|g_j\|_{y_j} + \beta_{2t}(\|g_j\|_{y_j} - \|\varphi(g_{j-1})\|_{y_j}) \quad (45)$$

1140 g_j is the gradient at y_j , g_{j-1} is the gradient at y_{j-1} . For the step - size formula, $\eta_t = \frac{\eta}{\sqrt{t}}$, when t is
 1141 large, $y_j \approx y_{j-1}$, assume $\|g_j\|_{y_j} \approx \|\varphi(g_{j-1})\|_{y_j}$, similar to the Lipschitz continuity of the gradient.
 1142 Therefore, we have $\|z_j\|_{y_j}^2 \geq \|g_j\|_{y_j}^2$. From another perspective, if $\|z_j\|_{y_j}^2 \leq \|g_j\|_{y_j}^2$, one can always
 1143 restrict $\beta_{2t} = 0$, in which case $\|z_j\|_{y_j}^2 \geq \|g_j\|_{y_j}^2$. Then for n_t :
 1144

$$1145 \quad n_t = \frac{1}{t} \sum_{j=1}^t \|z_j\|_{y_j}^2 \geq \frac{1}{t} \sum_{j=1}^t \|g_j\|_{y_j}^2 \quad (46)$$

1146 Next, consider the sum:
 1147

$$1148 \quad \sum_{t=1}^T \frac{\eta_t}{\sqrt{n_t}} \|m_t\|_{y_t}^2 \leq \sum_{t=1}^T \frac{\eta_t}{(1 - \beta_1)} \cdot \frac{\sum_{j=1}^t (\beta_1^{t-j} \|g_j\|_{y_j}^2)}{\sqrt{\frac{1}{t} \sum_{j=1}^t \|g_j\|_{y_j}^2}} \\ 1149 \quad = \sum_{t=1}^T \frac{\eta}{(1 - \beta_1)} \cdot \frac{\sum_{j=1}^t (\beta_1^{t-j} \|g_j\|_{y_j}^2)}{\sqrt{\sum_{j=1}^t \|g_j\|_{y_j}^2}} \\ 1150 \quad \leq \frac{2\eta}{(1 - \beta_1)^2} \cdot \sqrt{\sum_{j=1}^T \|g_j\|_{y_j}^2} \quad (47)$$

1151 Since we assume the gradient is bounded, i.e., $\|g_j\|_{y_j} \leq G$, we can proceed accordingly in the
 1152 analysis.
 1153

$$1154 \quad \sum_{t=1}^T \frac{\eta_t}{\sqrt{n_t}} \|m_t\|_{y_t}^2 \leq \frac{2\eta}{(1 - \beta_1)^2} \cdot \sqrt{\sum_{j=1}^T \|g_j\|_{y_j}^2} \leq \frac{2\eta}{(1 - \beta_1)^2} \cdot \sqrt{TG^2} = \frac{2\eta G}{(1 - \beta_1)^2} \cdot \sqrt{T} \quad (48)$$

1155 In summary, we have:
 1156

$$1157 \quad \sum_{t=1}^T \frac{\beta_{1t}}{1 - \beta_{1t}} \cdot \frac{\eta_t}{\sqrt{n_t}} \|m_t\|_{y_t}^2 \leq \sum_{t=1}^T \frac{\beta_1}{1 - \beta_1} \cdot \frac{\eta_t}{\sqrt{n_t}} \|m_t\|_{y_t}^2 \leq \frac{2\eta\beta_1 G}{(1 - \beta_1)^3} \cdot \sqrt{T} \quad (49)$$

1158 Considering B_{42} , we can directly use the boundedness of the feasible domain to obtain the following
 1159 expression:
 1160

$$1161 \quad \sum_{t=1}^T \frac{\sqrt{n_t}}{\eta_t} \|Log_{y_t}(y^*)\|_{y_t}^2 \cdot \frac{\beta_{1t}}{1 - \beta_{1t}} \leq \frac{1}{1 - \beta_1} \sum_{t=1}^T \frac{\sqrt{t} \cdot \sqrt{n_t}}{\eta} D_\infty^2 \cdot \beta_{1t} \quad (50)$$

1162 Since $n_t = \frac{1}{t} \sum_{j=1}^t \|z_j\|_{y_j}^2$, we have the following:
 1163

$$1164 \quad n_t = \frac{1}{t} \sum_{j=1}^t \|z_j\|_{y_j}^2 \\ 1165 \quad \leq \frac{1}{t} \sum_{j=1}^t (\|g_j\|_{y_j} + \beta_{2t}\|g_j\|_{y_j} + \beta_{2t}\|\varphi(g_{j-1})\|_{y_j})^2 \\ 1166 \quad \leq \frac{1}{t} G^2 \sum_{j=1}^t (1 + 2\beta_{2t})^2 \\ 1167 \quad \leq \frac{1}{t} G^2 t (1 + 2\beta_2)^2 \leq (1 + 2\beta_2)^2 G^2 \quad (51)$$

1188
1189 The above expression still uses the bounded gradient assumption. Substituting the earlier result, we
1190 obtain:
1191
1192

$$\sum_{t=1}^T \frac{\sqrt{n_t}}{\eta_t} \|Log_{y_t}(y^*)\|_{y_t}^2 \cdot \frac{\beta_{1t}}{1 - \beta_{1t}} \leq \frac{(1 + 2\beta_2)GD_\infty^2}{1 - \beta_1} \sum_{t=1}^T \frac{\sqrt{t}\beta_{1t}}{\eta} \quad (52)$$

1193
1194 For B_2 , we aim to provide an upper bound for $\sum_{t=1}^T \frac{\zeta(\kappa, c)\alpha_t}{2(1 - \beta_{1t})} \|u_t\|_{y_t}^2$. According to the update rules:
1195

$$\begin{cases} u_t = m_t + \beta_{2t}v_t \\ v_t = \beta_{2t}\varphi(v_{t-1}) + (1 - \beta_{2t})(g_t - \varphi(g_{t-1})) \end{cases} \quad (53)$$

1196
1197 According to the update rule for v_t and using the triangle inequality, we have
1198

$$\|v_t\|_{y_t} \leq ((1 - \beta_{2t}) \cdot \|g_t - \varphi(g_{t-1})\|_{y_t} + \beta_{2t}\|\varphi(v_{t-1})\|_{y_t}) \leq ((1 - \beta_{2t}) \cdot (\|g_t\|_{y_t} + \|\varphi(g_{t-1})\|_{y_t}) + \beta_{2t}\|\varphi(v_{t-1})\|_{y_t}) \quad (54)$$

1200
1201 Since $(1 - \beta_{2t}) \cdot (\|g_t\|_{y_t} + \|\varphi(g_{t-1})\|_{y_t}) + \beta_{2t}\|\varphi(v_{t-1})\|_{y_t}$ can be viewed as a convex combination
1202 of $(\|g_t\|_{y_t} + \|\varphi(g_{t-1})\|_{y_t})$ and $\|\varphi(v_{t-1})\|_{y_t}$, we have:
1203

$$(1 - \beta_{2t}) \cdot (\|g_t\|_{y_t} + \|\varphi(g_{t-1})\|_{y_t}) + \beta_{2t}\|\varphi(v_{t-1})\|_{y_t} \leq \sup_{y_t} (\|g_t\|_{y_t} + \|\varphi(g_{t-1})\|_{y_t}) \leq 2G. \quad (55)$$

1204
1205 Therefore, based on the update rule for u_t , together with the above result and the triangle inequality,
1206 we obtain the following inequality:
1207

$$\begin{aligned} \|u_t\|_{y_t}^2 &= \|m_t + \beta_{2t}v_t\|_{y_t}^2 \\ &\leq (\|m_t\|_{y_t} + \beta_{2t} \cdot \|v_t\|_{y_t})^2 \\ &\leq (\sup_{y_t} \|m_t\|_{y_t} + \sup_{y_t} \beta_{2t} \cdot \|v_t\|_{y_t})^2 \\ &\leq \left(\frac{G}{1 - \beta_1} + 2\beta_2 G\right)^2 \\ &\leq \left(\frac{3 - 2\beta_1}{1 - \beta_1}\right)^2 G^2 \end{aligned} \quad (56)$$

1208
1209 Therefore, we can obtain the upper bound for B_2 as follows:
1210

$$\begin{aligned} B_2 &= \sum_{t=1}^T \frac{\zeta(\kappa, c)\alpha_t}{2(1 - \beta_{1t})} \|u_t\|_{y_t}^2 \leq \sum_{t=1}^T \frac{\zeta(\kappa, c) \cdot \eta}{2(1 - \beta_1)\sqrt{t}} \left(\frac{3 - 2\beta_1}{1 - \beta_1}\right)^2 G^2 \\ &\leq \frac{\zeta(\kappa, c) \cdot \eta(3 - 2\beta_1)}{2(1 - \beta_1)^3} \sqrt{\sum_{t=1}^T \frac{1}{t}} \sqrt{\sum_{t=1}^T G^4} \leq \frac{\zeta(\kappa, c) \cdot (3 - 2\beta_1)\eta G^2 \sqrt{T} \sqrt{1 + \log T}}{2(1 - \beta_1)^3} \end{aligned} \quad (57)$$

1211
1212 For B_3 , we can directly apply the Cauchy-Schwarz inequality to estimate it.
1213

$$\begin{aligned} \sum_{t=1}^T \frac{\beta_{2t}}{1 - \beta_{1t}} \langle v_t, Log_{y_t}(y^*) \rangle_{y_t} &\leq \sum_{t=1}^T \frac{\beta_{2t}}{1 - \beta_1} \|v_t\|_{y_t}^2 \|Log_{y_t}(y^*)\|_{y_t}^2 \\ &\leq \sum_{t=1}^T \frac{\beta_{2t}}{1 - \beta_1} \cdot (2G)^2 \cdot D_\infty^2 \\ &\leq \sum_{t=1}^T \frac{4\beta_{2t}}{1 - \beta_1} D_\infty^2 G^2 \end{aligned} \quad (58)$$

1214
1215 We also need to slightly rearrange and simplify the previously obtained expression for B_1 .
1216

$$B_1 = \frac{D_\infty^2}{2(1 - \beta_1) \cdot \eta_T} \sqrt{n_T} \leq \frac{D_\infty^2 \sqrt{T}}{2(1 - \beta_1) \cdot \eta} \sqrt{(1 + 2\beta_2)^2 G^2} = \frac{GD_\infty^2(1 + 2\beta_2)\sqrt{T}}{2(1 - \beta_1) \cdot \eta} \quad (59)$$

1242 By organizing all the terms, we obtain the regret bound:
 1243

$$\begin{aligned}
 1244 \quad R_T &= \sum_{t=1}^T (f(y_t) - f(y^*)) \leq \sum_{t=1}^T \langle -g_t, \text{Log}_{y_t}(y^*) \rangle_{y_t} \\
 1245 &\leq \frac{GD_\infty^2(1+2\beta_2)\sqrt{T}}{2(1-\beta_1)\cdot\eta} + \frac{\zeta(\kappa, c) \cdot (3-2\beta_1)\eta G^2 \sqrt{T} \sqrt{1+\log T}}{2(1-\beta_1)^3} + \sum_{t=1}^T \frac{4\beta_{2t}}{1-\beta_1} D_\infty^2 G^2 \quad (60) \\
 1246 \\
 1247 &\quad + \frac{2\eta\beta_1 G}{(1-\beta_1)^3} \cdot \sqrt{T} + \frac{(1+2\beta_2)GD_\infty^2}{1-\beta_1} \sum_{t=1}^T \frac{\sqrt{t}\beta_{1t}}{\eta} \\
 1248 \\
 1249
 \end{aligned}$$

1250 Simplification yields the final expression for the regret bound:
 1251

$$\begin{aligned}
 1252 \quad R_T &\leq \frac{\zeta(\kappa, c) \cdot (3-2\beta_1)\eta G^2 \sqrt{T} \sqrt{1+\log T}}{2(1-\beta_1)^3} + \frac{2\eta\beta_1 G}{(1-\beta_1)^3} \sqrt{T} \\
 1253 &\quad + \frac{GD_\infty^2(1+2\beta_2)\sqrt{T}}{2(1-\beta_1)\cdot\eta} + \sum_{t=1}^T \frac{4D_\infty^2 G^2 \beta_{2t}}{1-\beta_1} + \sum_{t=1}^T \frac{\sqrt{t}(1+2\beta_2)GD_\infty^2 \beta_{1t}}{\eta(1-\beta_1)} \quad (61) \\
 1254 \\
 1255 \\
 1256 \\
 1257 \\
 1258 \\
 1259
 \end{aligned}$$

1260 A.1.3 PROOF OF LEMMA

1261 In this section, we will provide proofs for the three lemmas used in Theorem 3.1.
 1262

1263 Lemma 1. If the feasible domain $D \subset \mathcal{M}$ is geodesically bounded (i.e., there exists a constant D_∞
 1264 such that $d(x, y) \leq D_\infty$ for all $x, y \in D$), then for any $x \in D$,

$$1265 \quad \|\text{Log}_{y_t}(x)\|_{y_t} \leq D_\infty, \quad (62)$$

1266 where x is any point, and $\text{Log}_{y_t}(\cdot)$ is the logarithmic map on \mathcal{M} .
 1267

1268 *Proof.* By definition, the logarithmic map $\text{Log}_x(y)$ maps a point $y \in \mathcal{M}$ to a tangent vector in $T_x \mathcal{M}$
 1269 whose norm equals the geodesic distance $d(x, y)$:
 1270

$$1271 \quad \|\text{Log}_x(y)\|_x = d(x, y). \quad (63)$$

1272 Since D is geodesically bounded, for any $y_t \in D$ and $x \in D$, $d(y_t, x) \leq D_\infty$. Combining the above
 1273 two results,
 1274

$$1275 \quad \|\text{Log}_{y_t}(x)\|_{y_t} = d(y_t, x) \leq D_\infty. \quad (64)$$

1276 This completes the proof.
 1277

1278 Lemma 2. If $f : \mathcal{M} \rightarrow \mathbb{R}$ is a geodesically convex function, then for any $y_t \in \mathcal{M}$,

$$1279 \quad f(y_t) - f(y^*) \leq \langle -\text{grad}f(y_t), \text{Log}_{y_t}(y^*) \rangle_{y_t}, \quad (65)$$

1280 where $\text{grad}f(y_t)$ is the Riemannian gradient of f at y_t .
 1281

1282 *Proof.* A function f is geodesically convex if, for any geodesic $\gamma : [0, 1] \rightarrow \mathcal{M}$,

$$1283 \quad f(\gamma(t)) \leq (1-t)f(\gamma(0)) + tf(\gamma(1)), \quad \forall t \in [0, 1]. \quad (66)$$

1284 Let $\gamma(0) = y_t$ and $\gamma(1) = y^*$. Then,
 1285

$$1286 \quad f(\gamma(t)) \leq (1-t)f(y_t) + tf(y^*). \quad (67)$$

1287 Expand $f(\gamma(t))$ around $t = 0$ using the exponential map $\gamma(t) = \text{Exp}_{y_t}(t \cdot \text{Log}_{y_t}(y^*))$:
 1288

$$1289 \quad f(\gamma(t)) = f(y_t) + t \langle \text{grad}f(y_t), \text{Log}_{y_t}(y^*) \rangle_{y_t} + o(t). \quad (68)$$

1290 Substituting into the geodesic convexity inequality:
 1291

$$1292 \quad f(y_t) + t \langle \text{grad}f(y_t), \text{Log}_{y_t}(y^*) \rangle_{y_t} + o(t) \leq (1-t)f(y_t) + tf(y^*). \quad (69)$$

1296 Rearranging terms and dividing by $t > 0$:

$$1298 \quad \langle \text{grad}f(y_t), \text{Log}_{y_t}(y^*) \rangle_{y_t} + \frac{o(t)}{t} \leq f(y^*) - f(y_t). \quad (70)$$

1300 Taking $t \rightarrow 0$, the higher-order term $\frac{o(t)}{t} \rightarrow 0$, yielding:

$$1302 \quad f(y_t) - f(y^*) \leq -\langle \text{grad}f(y_t), \text{Log}_{y_t}(y^*) \rangle_{y_t}. \quad (71)$$

1304 This completes the proof.

1306 Lemma 3. Let $p_1, \dots, p_k \in \mathbb{R}^d$ and weights $a_1, \dots, a_k \geq 0$. Then

$$1308 \quad \left\| \sum_{i=1}^k a_i p_i \right\|^2 \leq \left(\sum_{i=1}^k a_i \right) \left(\sum_{i=1}^k a_i \|p_i\|^2 \right). \quad (72)$$

1312 *Proof.* Define $w_i = \sqrt{a_i}$, $v_i := \sqrt{a_i} p_i$. Then

$$1314 \quad \left\| \sum_{i=1}^k a_i p_i \right\|^2 = \left\| \sum_{i=1}^k w_i v_i \right\|^2 = \left\langle \sum_{i=1}^k w_i v_i, \sum_{j=1}^k w_j v_j \right\rangle = \sum_{i=1}^k \sum_{j=1}^k w_i w_j \langle v_i, v_j \rangle \\ 1315 \quad \leq \underbrace{\left(\sum_{i=1}^k w_i^2 \right)}_{\text{(by Cauchy Schwarz)}} \underbrace{\left(\sum_{j=1}^k \|v_j\|^2 \right)}_{\text{}} = \left(\sum_{i=1}^k a_i \right) \left(\sum_{j=1}^k a_j \|p_j\|^2 \right), \quad (73)$$

1322 which proves the claim.

1324 A.2 PROOF OF THEOREM 3.2

1325 **Theorem A.2.** In the bound Equation (10), any non-summation term $K(T)$ satisfies $\mathbf{o}\left(\frac{K(T)}{T}\right) = 0$. For the summation terms, as long as the parameter decay conditions $\mathbf{o}\left(\frac{\sum_{t=1}^T \beta_{1t} \sqrt{t}}{T}\right) = 0$, $\mathbf{o}\left(\frac{\sum_{t=1}^T \beta_{2t}}{T}\right) = 0$ and $\beta_{3t} = 1 - \frac{1}{t}$ are met, Radan converges to the optimum.

1331 *Proof.* Recall from Theorem 3.2 that the total regret R_T obeys:

$$1333 \quad R_T \leq K(T) + \sum_{t=1}^T A_t = K(T) + \sum_{t=1}^T \underbrace{\frac{4D_\infty^2 G^2}{1-\beta_1} \beta_{2t}}_{=:a_{2t}} + \sum_{t=1}^T \underbrace{\frac{\sqrt{t}(1+2\beta_2)GD_\infty^2 \beta_{1t}}{\eta(1-\beta_1)}}_{=:a_{1t}}, \quad (74)$$

1337 where we have that:

$$1339 \quad K(T) = \frac{\zeta(\kappa, c) \cdot (3 - 2\beta_1) \eta G^2 \sqrt{T} \sqrt{1 + \log T}}{2(1 - \beta_1)^3} + \frac{2\eta\beta_1 G}{(1 - \beta_1)^3} \sqrt{T} + \frac{GD_\infty^2 (1 + 2\beta_2) \sqrt{T}}{2(1 - \beta_1) \cdot \eta} \quad (75)$$

1341 Dividing both sides by T gives:

$$1343 \quad \frac{R_T}{T} \leq \frac{K(T)}{T} + \frac{1}{T} \sum_{t=1}^T a_{1t} + \frac{1}{T} \sum_{t=1}^T a_{2t}. \quad (76)$$

1346 Set the constants $c_1 = \frac{(1+2\beta_2)GD_\infty^2}{\eta(1-\beta_1)}$, $c_2 = \frac{4D_\infty^2 G^2}{1-\beta_1}$, so that:

$$1348 \quad \frac{1}{T} \sum_{t=1}^T a_{1t} = \frac{c_1}{T} \sum_{t=1}^T \beta_{1t} \sqrt{t}, \quad \frac{1}{T} \sum_{t=1}^T a_{2t} = \frac{c_2}{T} \sum_{t=1}^T \beta_{2t}. \quad (77)$$

1350 Each summand in $K(T)$ scales like $T^{-1/2}$ (up to logarithmic factors), hence $\frac{K(T)}{T} = \mathbf{o}(1) \implies$
 1351 $\mathbf{o}\left(\frac{K(T)}{T}\right) = 0$. By hypothesis, $\mathbf{o}\left(\frac{1}{T} \sum_{t=1}^T \beta_{1t} \sqrt{t}\right) = 0$, $\mathbf{o}\left(\frac{1}{T} \sum_{t=1}^T \beta_{2t}\right) = 0$. Multiplying by
 1352 the constants c_1, c_2 preserves the vanishing rate, so $\frac{1}{T} \sum_{t=1}^T a_{1t} = \mathbf{o}(1)$ and $\frac{1}{T} \sum_{t=1}^T a_{2t} = \mathbf{o}(1)$.
 1353 Combining these,
 1354

$$\frac{R_T}{T} \leq \underbrace{\mathbf{o}(1)}_{K(T)/T} + \underbrace{\mathbf{o}(1)}_{(1/T) \sum a_{1t}} + \underbrace{\mathbf{o}(1)}_{(1/T) \sum a_{2t}} = \mathbf{o}(1). \quad (78)$$

1355 Hence $\lim_{T \rightarrow \infty} R_T/T = 0$, i.e. Radan attains vanishing average regret and converges to the global
 1356 optimum.
 1357

1358 A.3 PROOF OF THEOREM 3.3

1359 A.3.1 PROOF DETAILS

1360 **Theorem A.3.** *On a single constant-curvature manifold \mathbb{R}^r , $\mathbb{S}^{s,K}$, or $\mathbb{H}^{h,K}$, the Riemannian gradient
 1361 of the Riemannian Fuzzy K-Means objective function J_{FK} with respect to the cluster center c_k is
 1362 uniformly expressed as:*

$$1363 \quad \text{grad}_{c_k} J_{FK} = -2 \sum_{i=1}^N S_i^{-m} d(y_i, c_k)^{-\frac{2m}{m-1}} \text{Log}_{c_k}(y_i), \quad (79)$$

1364 where $\text{Log}_{c_k}(x_i)$ denotes the logarithmic map of point x_i at c_k . The $\text{Log}_{c_k}(x_i)$ on three types of
 1365 constant-curvature manifolds are given as follows.
 1366

$$1367 \quad \text{Log}_c(x) = \begin{cases} x - c, & \text{if } x, c \in \mathbb{R}^r, \\ \frac{\theta}{\sin(\theta)} (x - \cos(\theta) c), & \theta = \cos^{-1}(K^2 \langle c, x \rangle), \text{ if } x, c \in \mathbb{S}^{s,K}, \\ \frac{\theta}{\sinh(\theta)} (x + K^2 \langle c, x \rangle_h c), & \theta = \cosh^{-1}(K^2 \langle c, x \rangle_h), \text{ if } x, c \in \mathbb{H}^{h,K}. \end{cases} \quad (80)$$

1368 *Proof.* To transform it into the above form, we simplify J_{FK} using the expression of S_i .
 1369

$$1370 \quad \begin{cases} J_{FK}(u_{ij}(c_j), c_j) = \sum_{i=1}^N \left(\sum_{j=1}^C d(x_i, c_j)^{-\frac{2}{m-1}} \right)^{1-m}, \\ S_i = \sum_{j=1}^C \left(\sum_{p=1}^Q d_p^2(x_i^p, c_j^p) \right)^{-\frac{1}{m-1}} = \sum_{j=1}^C d(x_i, c_j)^{-\frac{2}{m-1}} \end{cases} \quad (81)$$

1371 Due to Equation 81, we can simply express J_{FK} as Equation 82.
 1372

$$1373 \quad J_{FK}(u_{ij}(c_j), c_j) = \sum_{i=1}^N \left(\sum_{j=1}^C d(x_i, c_j)^{-\frac{2}{m-1}} \right)^{1-m} = \sum_{i=1}^N S_i^{1-m} \quad (82)$$

1374 Consider taking the Riemannian gradient with respect to the k -th center c_k . Obviously, when
 1375 differentiating S_i with respect to c_k , only the term with $j = k$ is nonzero. Therefore, according to the
 1376 chain rule of Riemannian gradients, we obtain Equation 83.
 1377

$$1378 \quad \text{grad}_{c_k} J_{FK} = (1-m) \sum_{i=1}^N S_i^{-m} \text{grad}_{c_k} \left(d(x_i, c_k)^{-\frac{2}{m-1}} \right) \quad (83)$$

1379 According to the lemma $\text{grad}_c d(x, c) = -\frac{\text{Log}_c(x)}{d(x, c)}$ (proved later), we further simplify Equation 83
 1380 and obtain Equation 84.
 1381

$$1382 \quad \text{grad}_c d(x, c)^a = ad(x, c)^{a-1} \text{grad}_c d(x, c) = -ad(x, c)^{a-2} \text{Log}_c(x). \quad (84)$$

1404 By setting $a = -\frac{2}{m-1}$, we obtain Equation 85.
 1405

$$1406 \text{grad}_{c_k} (d(x_i, c_k)^{-\frac{2}{m-1}}) = -\frac{2}{m-1} d(x_i, c_k)^{\frac{2}{m-1}-2} \text{Log}_{c_k}(x_i) = -\frac{2}{m-1} d(x_i, c_k)^{\frac{-2m}{m-1}} \text{Log}_{c_k}(x_i) \quad (85)$$

1408 Simply substituting Equation 85 into Equation 83 yields Equation 11.
 1409

$$\begin{aligned} 1410 \text{grad}_{c_k} J_{FK} &= (1-m) \sum_{i=1}^N S_i^{-m} \text{grad}_{c_k} (d(x_i, c_k)^{-\frac{2}{m-1}}) \\ 1411 &= (1-m) \sum_{i=1}^N S_i^{-m} \left(-\frac{2}{m-1} d(x_i, c_k)^{\frac{-2m}{m-1}} \text{Log}_{c_k}(x_i) \right) \\ 1412 &= -2 \sum_{i=1}^N S_i^{-m} d(x_i, c_k)^{-\frac{2m}{m-1}} \text{Log}_{c_k}(x_i) \end{aligned} \quad (86)$$

1419 A.3.2 PROOF OF LEMMA
 1420

1421 We now prove a key lemma.
 1422

1423 **Lemma A.4.** *Let $x, c \in \mathcal{M}$, and let $d(x, c)$ denote the geodesic distance between x and c . Then we
 1424 have $\text{grad}_c d(x, c) = -\frac{\text{Log}_c(x)}{d(x, c)}$.*
 1425

1426 *Proof.* First, consider the function $f(c) = \frac{1}{2}d^2(x, c)$ and its directional derivative along the direction
 1427 w , denoted by $\frac{\partial f}{\partial w}$.
 1428

$$\frac{\partial f}{\partial w} = \lim_{t \rightarrow 0} \frac{1}{2} \frac{d^2(x, \text{Exp}_c(tw)) - d^2(x, c)}{t} \quad (87)$$

1429 Let $\gamma(t) = \text{Exp}_c(tw)$, which is the geodesic starting from c along w . The directional derivative can
 1430 then be written as:
 1431

$$\frac{\partial f}{\partial w} = \lim_{t \rightarrow 0} \frac{1}{2} \frac{d^2(x, \text{Exp}_c(tw)) - d^2(x, c)}{t} = \frac{d}{dt} \Big|_{t=0} \frac{1}{2} d^2(x, \gamma(t)) \quad (88)$$

1435 According to the standard formula in Riemannian geometry (You), we have:
 1436

$$\frac{d}{dt} \Big|_{t=0} \frac{1}{2} d^2(x, \gamma(t)) = \langle -\text{Log}_c(x), w \rangle_c \quad (89)$$

1439 Therefore, we obtain the final equation:
 1440

$$\frac{\partial f}{\partial w} = \frac{d}{dt} \Big|_{t=0} \frac{1}{2} d^2(x, \gamma(t)) = \langle -\text{Log}_c(x), w \rangle_c = \langle \text{grad}_c \left(\frac{1}{2} d^2(x, c) \right), w \rangle_c \quad (90)$$

1443 So that:
 1444

$$\text{grad}_c(d(x, c)) = \frac{\text{grad}_c \left(\frac{1}{2} d^2(x, c) \right)}{d(x, c)} = -\frac{\text{Log}_c(x)}{d(x, c)} \quad (91)$$

1446
 1447
 1448
 1449 With this, we complete all the proofs.
 1450

1451
 1452
 1453
 1454
 1455
 1456
 1457

1458 **B NOTATIONS**
14591460 Table 4 lists all the symbols used and their corresponding meanings.
14611462 Table 4: Notations in this paper.
1463

Notation	Description
$X = \{x_1, \dots, x_N\}$	Dataset notation, consisting of N samples
x_i	The i -th sample
y_t	The coordinate of optimization variable y at step t
c_j	The j -th cluster center
\mathcal{M}_p	The p -th component manifold
$\otimes_{p=1}^Q \mathcal{M}_p$	The product manifold of Q component manifolds
$(x_i^1, x_i^2, \dots, x_i^Q)$	The coordinate representation of x_i under Q product manifolds $\otimes_{p=1}^Q \mathcal{M}_p$
$d_p(x^p, y^p)$	The geodesic distance computed from the coordinates of x and y on the p -th component manifold
$d(x, y)$	The distance between x and y on the product manifold $\otimes_{p=1}^Q \mathcal{M}_p$
$\mathbb{H}^{h_i, K}$	The hyperbolic space of dimension h_i and curvature K , with $K < 0$
\mathbb{H}^{h_i}	The hyperbolic space of dimension h_i and curvature K , with $K = -1$
$\mathbb{S}^{s_i, K}$	The spherical space of dimension s_i and curvature K , with $K > 0$
\mathbb{S}^{s_i}	The spherical space of dimension s_i and curvature K , with $K = 1$
\mathbb{R}^{r_i}	The Euclidean space of dimension r_i
\mathbb{D}	The 2-dimensional Poincaré disk
$T_{x^p} \mathcal{M}_p$	The tangent space at x^p on the p -th component manifold
$T_x \mathcal{M}$	The tangent space at x on the product manifold $\otimes_{p=1}^Q \mathcal{M}_p$
$\ \cdot\ $	Euclidean norm
$\ \cdot\ _{x_t}$	Riemannian norm at x_t on the product manifold
$\varphi_{x^p \rightarrow y^p}^p(u^p)$	On the p -th component manifold, parallel transport u^p from x^p to y^p .
$\varphi_{x \rightarrow y}(u)$	On the product manifold, parallel transport u from x to y .
$\text{Exp}_{c^p}^p(u^p)$	Apply the exponential map to u^p at c^p on the p -th component manifold.
$\text{Exp}_c(u)$	Apply the exponential map to u at c on the product manifold.
$\text{Log}_{c^p}^p(x^p)$	Apply the logarithmic map to x^p at c^p on the p -th component manifold.
$\text{Log}_c(x)$	Apply the logarithmic map to x at c on the product manifold.
$\log(\cdot)$	Logarithmic function
$\{g_t^p, m_t^p, v_t^p, z_t^p, n_t^p, u_t^p, \alpha_t^p\}$	Intermediate quantity of Radan on the p -th component manifold
$\{g_t, m_t, v_t, z_t, n_t, u_t, \alpha_t\}$	Intermediate quantity of Radan on the product manifold
$\zeta(\kappa, c)$	Curvature function
D_∞	Upper bound of the size of the geodesically convex region
$\gamma(t)$	Geodesic
$\langle \cdot, \cdot \rangle_{y_t}$	Riemannian inner product at y_t
$\langle \cdot, \cdot \rangle_h$	hyperbolic inner product
S_i	Intermediate variable of RFK
β_{1t}	First hyperparameter of Radan
β_{2t}	Second hyperparameter of Radan
β_{3t}	Third hyperparameter of Radan
$\mathcal{O}(\cdot)$	Infinitely large of the same order
$\mathbf{o}(\cdot)$	infinitely small of the same order
R_T	Regret bound

1494 **C RELATED WORK ABOUT CLUSTERING ON MANIFOLD**
14951496 In terms of clustering algorithm design for data distributed on manifolds, there has not been extensive
1497 research so far. In (Miolane et al., 2020), an iterative Riemannian K-Means-style algorithm was
1498 implemented by alternately updating the assignments $\{u_{ij}\}$ and the centers $\{c_j\}$, with a time
1499 complexity of $\mathcal{O}(\omega\nu)$. Many application scenarios adopt this alternating update paradigm, such as
1500 (Wu & Pan, 2025b). Some recent methods for clustering data distributed in hyperbolic spaces have
1501 been proposed (Jaćimović & Crnkić, 2025; Ghosh & Das, 2024; Lin et al., 2022). However, these
1502 approaches are not applicable to product manifolds and therefore cannot be compared with RFK.
1503 There also exist deep learning-based clustering methods (Sun et al., 2023b). However, they lack
1504 flexibility, lightweight implementation, and interpretability compared to machine learning-based
1505 algorithms. Moreover, deep clustering frameworks often require a clustering procedure similar to
1506 RFK to generate pseudo-labels for the learned deep representations. Hence, RFK can serve as a
1507 natural and effective replacement for NRK in this context. Moreover, some clustering algorithms
1508 assume data lie on an unknown submanifold; algorithms based on this idea still fail to fully respect
1509 the data's geometry. Such methods, e.g., Zhong & Pun (2021), are included in our comparisons. In
1510 addition, we further compare with several clustering approaches defined on other manifolds (Subbarao
1511 & Meer, 2009; Ashizawa et al., 2017; Zhao et al., 2016).

1512 **D BACKGROUND**
15131514 Since Riemannian machine learning is a relatively novel research direction and not yet widely familiar
1515 to all readers, we provide a detailed introduction to the background of this field in this section.
15161517 **D.1 WHAT KIND OF DATA DO WE LEARN?**
15181519 In fact, Riemannian machine learning focuses on data that have already been represented on manifolds.
1520 With the development of representation learning, researchers commonly use neural networks to
1521 automatically extract features and obtain new representations of data (Bengio et al., 2013). However,
1522 scientists soon realized that many types of data possess non-Euclidean structures, and forcibly
1523 embedding them into Euclidean space causes distortions (Yang et al., 2023; Ren et al., 2025).1524 For example, for periodic data such as cells at different stages of a division cycle, Euclidean
1525 embedding fails to capture periodicity, representing them on spheres, hyperspheres, or tori is more
1526 appropriate (Davidson et al., 2018). Data with hierarchical structures—such as graphs or trees—are
1527 better represented on hyperbolic manifolds (Sala et al., 2018; Mishne et al., 2023). Hyperbolic
1528 representations have been widely used in video retrieval (Li et al., 2025), bioinformatics (Ding &
1529 Regev, 2021), and large language models (Mandica et al., 2024b).1530 Moreover, if data simultaneously exhibit multiple structural properties, they are often embedded into
1531 product manifolds composed of several manifolds (Chlenski et al., 2025a).1532 Riemannian machine learning focuses on such data that are already represented on manifolds, aiming
1533 to perform classification (Bachmann et al., 2020), clustering (Ashizawa et al., 2017), and regression
1534 (Zhou et al., 2025). In the narrow sense, Riemannian machine learning extracts information from
1535 these non-Euclidean data, whereas obtaining these manifold-valued representations is the task of
1536 Riemannian representation learning.
15371538 **D.2 DIFFERENCE FROM MANIFOLD LEARNING**
15391540 A standard assumption in machine learning is that data lie on some unknown manifold (Izenman,
1541 2012). Manifold learning typically exploits local Euclidean approximations—for example, constructing
1542 a KNN graph (Costa & Hero, 2004) and applying spectral clustering methods such as Ncut.
1543 The key distinction from Riemannian machine learning is that manifold learning does not know
1544 the underlying manifold structure. As a result, its algorithms do not leverage manifold geometry
1545 explicitly and often perform poorly when data lie on a known manifold with known structure.
15461547 **D.3 BASIC PRINCIPLES OF RIEMANNIAN MACHINE LEARNING**1548 The fundamental principle of Riemannian machine learning is that problems should be considered
1549 from the perspective of the manifold itself rather than the Euclidean embedding space (Miolane et al.,
1550 2020).1551 A simple example is the construction of an airport at the geometric center of several countries: the
1552 center should be computed using a manifold center (the Fréchet mean) and distances measured on the
1553 manifold (the Earth’s sphere is a manifold). In contrast, using an Euclidean center could lead to a
1554 meaningless point, such as somewhere inside the Earth’s interior.
15551556 **D.4 MISCELLANEOUS QUESTIONS**
15571558 **Q1:** How do we determine which manifold a dataset belongs to?1559 **A1:** Several established methods can identify intrinsic structures in raw data (such as periodicity or
1560 hierarchy) and recommend an appropriate embedding manifold (Tabaghi et al., 2021).1561 **Q2:** How do we embed data onto these manifolds?1563 **A2:** This has also been extensively studied. Methods such as graph neural networks (Wang et al.,
1564 2021), UMAP (McInnes et al., 2018), and coordinate-learning approaches (Gu et al., 2018) can
1565 effectively map data into their corresponding manifolds.

1566
1567
1568

Table 5: Description Table of the benchmark datasets

	Dataset	Signature	Dimension	Class	Objects	
1569 1570 1571 1572 1573 1574 1575	Synthetic	Gaussian	\mathbb{R}^4	4	3	1000
			\mathbb{H}^4	5	3	1000
			$\mathbb{S}^2\mathbb{H}^2$	6	3	1000
			$\mathbb{R}^2\mathbb{S}^2\mathbb{H}^2$	8	3	1000
			$\mathbb{S}^2(\mathbb{H}^2)^2$	9	3	1000
			$\mathbb{R}^4\mathbb{S}^4\mathbb{H}^4$	14	3	1000
			$\mathbb{R}^{16}\mathbb{S}^{16}\mathbb{H}^{16}$	50	3	1000
1577 1578 1579 1580 1581 1582	Graph	CiteSeer	$(\mathbb{H}^2)^2$	6	6	2110
		Cora	\mathbb{H}^4	5	7	2485
		PolBlogs	$(\mathbb{S}^2)^2$	6	2	1222
		Olsson	\mathbb{D}	2	9	382
		Paul	\mathbb{D}	2	20	2730
		PolBooks	\mathbb{D}	2	3	106
1583 1584 1585	VAE	CIFAR-100	$(\mathbb{H}^2)^4$	12	10	500000
		Lymphoma	$(\mathbb{S}^2)^2$	6	10	134100
		MNIST	$\mathbb{S}^2\mathbb{E}^2\mathbb{H}^2$	8	10	600000

1586

E DETAILS OF THE EXPERIMENTAL SETUP

1589 E.1 DATASETS DESCRIPTION

1592 Table 5 presents the basic information of the datasets we used. Here, Signature refers to the type
 1593 of manifold onto which the dataset is embedded, Dimension indicates the dimensionality of the
 1594 embedding space, Class denotes the number of clusters in the data, and Objects specifies the total
 1595 number of samples in the dataset.

1596 Here, we also provide a brief introduction to the background of these datasets, along with the sources
 1597 from which each dataset can be obtained.

1598

- 1599 • All Gaussian datasets are generated using Manify’s ‘gaussian mixture’ function, with the
 1600 specific code as follows:

```
1601 from manify.manifolds import ProductManifold
1602 signature = [
1603     (0.0, 16),    #  $\mathbb{R}^{16}$  (Euclidean space)
1604     (1.0, 16),    #  $\mathbb{S}^{16}$  (Spherical space)
1605     (-1.0, 16),   #  $\mathbb{H}^{16}$  (Hyperbolic space)
1606 ]
1607 P = ProductManifold(signature, device="cpu", stereographic=False)
1608 n_clusters = 3
1609 X, y_true = P.gaussian_mixture(
1610     num_points=1000,
1611     num_classes=n_clusters,
1612     task="classification",
1613     cov_scale_points=.1
1614 )
1615 To ensure reproducibility, we also saved the generated data, which can be found here1.

```

- 1615 • CiteSeer, Cora, and PolBlogs are graph datasets, which can be represented in non-Euclidean
 1616 spaces using the following code:

```
1617 import manify
1618 from manify.utils.dataloaders import load_hf
```

¹<https://anonymous.4open.science/r/Manifold-Clustering-Data-3C53/>

```

1620
1621     features, dists, adj, labels = load_hf("polblogs")
1622
1623     pm = manify.ProductManifold(signature=[(1.0, 4), (-1.0, 4)])
1624
1625     embedder = manify.CoordinateLearning(pm=pm)
1626     X_embedded = embedder.fit_transform(X=None, D=dists,
1627                                         burn_in_iterations=200, training_iterations=800)
1628
1629     In fact, the Manify GitHub repository already provides the pre-trained embeddings of these
1630     datasets, which you can access there2, or alternatively obtain from our anonymous GitHub
1631     repository3.
1632
1633     • Olsson, Paul, and PolBooks are also graph datasets, which are embedded in the Poincaré
1634     disk. You can access the data here4, or alternatively obtain it through our anonymous link.
1635
1636     • The datasets CIFAR-100, Lymphoma, and MNIST are obtained using the VAE method
1637     provided in Manify. The reference code is as follows:
1638
1639     encoder = torch.nn.Sequential(
1640         torch.nn.Linear(784, 128),
1641         torch.nn.ReLU(),
1642         torch.nn.Linear(128, 2 * euclidean_manifold.dim), # The
1643         # INTRINSIC dimension of the manifold
1644     )
1645     decoder = torch.nn.Sequential(
1646         torch.nn.Linear(euclidean_manifold.ambient_dim, 128), # The
1647         # AMBIENT dimension of the manifold
1648         torch.nn.ReLU(),
1649         torch.nn.Linear(128, 784),
1650         torch.nn.Sigmoid(),
1651     )
1652
1653     vae = manify.ProductSpaceVAE(pm=euclidean_manifold, encoder=
1654         encoder, decoder=decoder)
1655
1656     mnist_embeddings = vae.fit_transform(
1657         X=mnist_features.reshape(-1, 784), burn_in_iterations=1,
1658         training_iterations=9, batch_size=128
1659     )
1660
1661     Manify also provides the precomputed embeddings of these datasets, which can be accessed
1662     here5 or through our anonymous link. In particular, MNIST performs poorly under small
1663     learning rates. In the RFK algorithm, its learning rate is set to 3, while in Experiment 2 we
1664     adopt the settings {2.1, 2.3, 2.5, 2.7, 3.0}.

```

E.2 EXPERIMENT 3 SETUP

E.2.1 BENCHMARK CLUSTERING ALGORITHMS

We compare it with 10 benchmark clustering algorithms across 7 toy datasets and 9 real-world datasets. These algorithms include K-Means-based methods, graph-based methods, and subspace-based methods. A detailed introduction to each algorithm is provided below.

- NRK, i.e., Naive Riemannian K-Means, is a K-Means-based algorithm that respects the manifold structure but requires double loops. Our main contribution is to modify it in order to reduce its complexity.
- KM partitions data into predefined clusters by minimizing the sum of squared distances between data points and their corresponding cluster centers. It is simple but sensitive to initial centroids and struggles with non-spherical clusters.

²<https://github.com/pchlenski/manify/tree/Dataset-Generation/data/graphs/embeddings>

³<https://anonymous.4open.science/r/Manifold-Clustering-Data-3C53/>

⁴<https://github.com/drewwiliimitis/hyperbolic-learning/tree/master/data/ucidata-zachary>

⁵<https://github.com/pchlenski/manify/tree/Dataset-Generation/data/mnist/embeddings>

- Ncut improves on Ratio-Cut by normalizing the cut, balancing the partition while considering the total graph weight. It's better suited for non-convex and unevenly distributed clusters.
- FCM Fuzzy C-Means (or Fuzzy K-Means), can be regarded as a relaxation of K-Means. Instead of hard assignments, it computes the similarity between each sample and each cluster center as the assignment criterion. It is also a well-known clustering algorithm.
- UFCM This is an unconstrained Fuzzy C-Means algorithm, which aims to replace the constrained alternating optimization in traditional Fuzzy C-Means with an unconstrained gradient descent approach.
- LRR This is a subspace-based clustering method, which leverages low-rank representations to obtain robust subspace clustering results.
- SSC This is also a subspace clustering method, characterized by sparse representation. Through sparse representation, SSC can often identify the core low-rank structure of the data, achieving excellent clustering performance while simultaneously reducing data dimensionality.
- SBMC is a graph-based balanced clustering method. Being graph-based means it clusters data by constructing a graph adjacency matrix. Balanced clustering indicates that the clustering results tend to have roughly equal numbers of samples in each cluster.
- USPEC is one of the representative ensemble clustering algorithms. Ensemble clustering integrates the information from multiple base clusterers to produce a final result, achieving performance far superior to any single clusterer.
- Fast-CD This is a fast and stable clustering algorithm for solving the Ncut loss function, which often achieves clustering results with lower loss than the Ncut itself, combining efficiency and robustness.

To evaluate the clustering performance comprehensively, three metrics are applied, which are clustering accuracy (ACC), normalized mutual information (NMI) and adjusted rand index (ARI). The calculation of these three metrics are displayed below.

E.2.2 CLUSTERING ACCURACY (ACC)

Clustering Accuracy measures the proportion of correctly clustered data points by aligning predicted cluster labels with ground truth labels. Since clustering algorithms do not inherently assign specific labels, a permutation mapping is applied, often using the Hungarian algorithm, to maximize alignment. The formula for ACC is:

$$\text{ACC} = \frac{\delta(\text{map}(\hat{y}_i), y_i)}{n} \quad (92)$$

where $\delta(a, b)$ is an indicator function defined as:

$$\delta(a, b) = \begin{cases} 1, & \text{if } a = b \\ 0, & \text{otherwise,} \end{cases} \quad (93)$$

Here, \hat{y}_i is the predicted label, y_i is the true label, n is the total number of data points, and $\text{map}(\hat{y}_i)$ is the permutation mapping function that aligns predicted labels with ground truth labels. ACC ranges from 0 to 1, with higher values indicating better clustering performance.

E.2.3 NORMALIZED MUTUAL INFORMATION (NMI)

Normalized Mutual Information quantifies the mutual dependence between clustering results and ground truth labels, normalized to account for differences in label distributions. It evaluates the overlap between clusters and true classes using information theory. Given predicted partitions $\{\hat{C}_i\}_{i=1}^c$ and ground truth partitions $\{C_j\}_{j=1}^c$, NMI is calculated as:

$$\text{NMI} = \frac{\sum_{i=1}^c \sum_{j=1}^c |\hat{C}_i \cap C_j| \log \frac{n|\hat{C}_i \cap C_j|}{|\hat{C}_i||C_j|}}{\sqrt{\left(\sum_{i=1}^c |\hat{C}_i| \log \frac{|\hat{C}_i|}{n}\right) \left(\sum_{j=1}^c |C_j| \log \frac{|C_j|}{n}\right)}} \quad (94)$$

1728 Here, $|\cdot|$ denotes the size of a set, and $\hat{C}_i \cap C_j$ represents the number of data points belonging to both
 1729 the i -th predicted cluster and the j -th ground truth class. NMI ranges from 0 to 1, where 1 indicates
 1730 perfect agreement between clustering results and ground truth. It is particularly effective in scenarios
 1731 with imbalanced class distributions.

1733 E.2.4 ADJUSTED RAND INDEX (ARI)

1735 The Adjusted Rand Index measures the similarity between predicted clustering and ground truth by
 1736 comparing all pairs of samples and evaluating whether they are assigned to the same cluster in both
 1737 results. A contingency table H is first constructed, where each element h_{ij} represents the number of
 1738 samples in both predicted cluster \hat{C}_i and ground truth cluster C_j . The formula for ARI is:

$$1739 \text{ARI}(\bar{C}, C) = \frac{\sum_{ij} \binom{n_{ij}}{2} - \left[\sum_i \binom{n^i}{2} \sum_j \binom{n^j}{2} \right] / \binom{n}{2}}{\frac{1}{2} \left[\sum_i \binom{n^i}{2} + \sum_j \binom{n^j}{2} \right] - \left[\sum_i \binom{n^i}{2} \sum_j \binom{n^j}{2} \right] / \binom{n}{2}} \quad (95)$$

1743 where $\binom{n_{ij}}{2} = \frac{n_{ij}(n_{ij}-1)}{2}$. ARI ranges from -1 to 1, where 1 indicates perfect clustering, 0 represents
 1744 random assignments, and negative values indicate worse-than-random clustering. ARI is robust to
 1745 differences in cluster sizes and does not favor a large number of clusters.

1747 E.2.5 F1 SCORE

1748 The F1 Score evaluates the balance between clustering precision and recall, capturing both the
 1749 completeness and exactness of the clustering results. It is computed based on pairwise precision and
 1750 recall between predicted clusters and ground truth classes. The F1 Score is defined as:

$$1752 \text{F1} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \quad (96)$$

1754 where Precision and Recall are given by:

$$1756 \text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \quad \text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad (97)$$

1758 Here, TP (true positives) is the number of data point pairs correctly assigned to the same cluster,
 1759 FP (false positives) is the number of pairs incorrectly assigned to the same cluster, and FN (false
 1760 negatives) is the number of pairs that belong to the same ground truth cluster but are assigned to
 1761 different clusters. F1 Score ranges from 0 to 1, with higher values indicating better clustering quality.

1763 E.2.6 PURITY

1764 Purity measures the extent to which clusters contain data points from a single ground truth class. For
 1765 each cluster, the class with the maximum frequency is identified, and the sum of these maximum
 1766 frequencies over all clusters is normalized by the total number of data points. Purity is defined as:

$$1768 \text{Purity} = \frac{1}{n} \sum_k \max_j |C_k \cap L_j| \quad (98)$$

1770 where C_k denotes the set of data points in cluster k , L_j denotes the set of data points in ground
 1771 truth class j , and n is the total number of data points. Purity ranges from 0 to 1, with higher values
 1772 indicating that clusters are more homogeneous with respect to the true labels.

1774
 1775
 1776
 1777
 1778
 1779
 1780
 1781

1782 Table 6: NMI for all benchmarks. OT means out-of-time
1783

	Dataset	Signature	RFK	NRK	K-Means	Ncut	FCM	UFCM	LRR	SSC	SBMC	USPEC	Fast-CD
1784 Synthetic	Gaussian	\mathbb{R}^4	88.49	88.37	88.39	95.87	88.36	88.54	69.01	0.51	52.81	62.61	91.18
		\mathbb{H}^4	98.89	98.24	3.82	96.02	20.88	6.37	98.87	7.71	29.22	80.41	38.75
		$\mathbb{S}^2\mathbb{H}^2$	84.16	84.16	69.67	73.00	71.90	71.58	89.96	0.35	42.12	62.28	72.39
		$\mathbb{R}^2\mathbb{S}\mathbb{H}^2$	84.27	83.95	39.94	96.22	45.06	39.84	74.79	0.25	61.67	1.27	60.58
		$\mathbb{S}^2(\mathbb{H}^2)^2$	90.37	89.25	0.53	58.31	8.44	4.50	85.62	3.99	40.49	42.56	29.40
		$\mathbb{R}^4\mathbb{S}^4\mathbb{H}^4$	95.70	95.42	7.13	57.73	57.58	8.97	87.58	5.46	68.57	45.00	86.70
1785 Graph	CiteSeer	$\mathbb{R}^2\mathbb{H}^2$	91.98	73.62	0.53	55.95	1.99	0.52	0.43	1.12	20.02	29.85	23.68
		$(\mathbb{H}^2)^2$	0.28	0.54	0.63	0.57	0.48	0.58	0.60	0.29	0.53	0.59	0.66
		\mathbb{H}^4	0.00	0.74	0.70	0.65	0.71	0.60	0.70	0.24	0.48	0.70	0.74
		$(\mathbb{S}^2)^2$	68.76	65.77	66.94	4.02	65.41	67.26	18.61	0.08	1.41	3.79	66.09
		\mathbb{D}	70.34	70.26	67.35	66.44	66.93	66.77	37.73	58.29	54.92	51.96	65.77
		\mathbb{D}	61.70	59.78	58.28	55.95	58.11	58.24	27.25	0.67	32.06	58.59	56.41
1786 VAE	PolBooks	\mathbb{D}	45.48	41.59	36.83	34.71	34.36	36.17	7.34	7.34	30.13	29.50	39.34
		CIFAR-100	88.24	OT	0.52	OT	0.62	0.24	OT	OT	OT	0.17	OT
		Lymphoma	100.00	OT	0.00	OT	0.00	OT	OT	OT	OT	0.00	OT
		MNIST	$\mathbb{S}^2\mathbb{E}^2\mathbb{H}^2$	93.00	OT	0.56	OT	2.76	0.99	OT	OT	0.20	OT

1795 Table 7: ARI for all benchmarks. OT means out-of-time
1796

	Dataset	Signature	RFK	NRK	K-Means	Ncut	FCM	UFCM	LRR	SSC	SBMC	USPEC	Fast-CD
1797 Synthetic	Gaussian	\mathbb{R}^4	90.12	89.63	90.17	97.42	90.12	90.35	72.99	0.46	52.68	65.73	95.09
		\mathbb{H}^4	99.47	99.07	-0.27	97.35	17.68	-0.55	99.48	2.50	27.41	83.48	62.88
		$\mathbb{S}^2\mathbb{H}^2$	87.70	85.45	70.89	74.88	74.24	73.64	93.58	0.16	41.03	65.74	82.90
		$\mathbb{R}^2\mathbb{S}^2\mathbb{H}^2$	88.04	87.56	33.34	98.15	38.76	32.77	83.47	0.09	60.88	0.02	74.92
		$\mathbb{S}^2(\mathbb{H}^2)^2$	92.48	92.12	-1.55	42.72	-1.06	-1.27	88.69	-1.50	39.29	36.48	60.84
		$\mathbb{R}^4\mathbb{S}^4\mathbb{H}^4$	97.34	96.25	0.99	55.54	53.28	2.07	91.78	0.32	68.88	44.29	92.57
1798 Graph	CiteSeer	$\mathbb{R}^2\mathbb{H}^2$	0.04	0.20	0.33	-0.09	0.18	0.31	0.30	0.07	0.15	0.50	0.19
		\mathbb{H}^4	0.00	0.20	0.15	-0.30	0.20	0.15	0.18	0.00	0.07	-0.29	0.14
		$(\mathbb{S}^2)^2$	78.67	76.08	77.09	1.17	75.79	77.46	13.82	-0.01	1.83	4.84	88.67
		\mathbb{D}	51.10	50.88	49.34	47.10	48.02	48.74	22.86	44.50	33.58	44.06	45.33
		\mathbb{D}	37.36	33.48	35.26	31.71	34.84	35.76	10.34	-0.02	12.19	35.24	30.90
		\mathbb{D}	55.38	44.87	46.47	43.99	44.66	46.01	8.58	53.34	35.74	36.40	51.02
1799 VAE	CIFAR-100	$(\mathbb{H}^2)^4$	78.67	OT	0.05	OT	0.10	0.01	OT	OT	OT	0.01	OT
		$(\mathbb{S}^2)^2$	100.00	OT	0.00	OT	0.00	OT	OT	OT	OT	0.00	OT
		$\mathbb{S}^2\mathbb{E}^2\mathbb{H}^2$	91.52	OT	0.06	OT	1.20	0.42	OT	OT	OT	4.84	OT

1809 F ADDITIONAL EXPERIMENTAL RESULTS
18101811 F.1 EXPERIMENTAL 3 RESULTS
1812

1813 In this section, we present the experimental results of NMI, ARI, F1, and Purity from Experiment 3. Tables 6, 7, 8, and 9 respectively present the NMI, ARI, F1, and Purity metrics of different algorithms across various datasets. It can be observed that, except for the first dataset, RFK consistently and significantly outperforms the other methods on all metrics. This is because the Gauss \mathbb{R}^4 dataset lies in Euclidean space, where RFK degenerates to Fuzzy K-Means, thus yielding results similar to K-Means and other implementations of Fuzzy K-Means. Moreover, it is worth noting that for large-scale datasets, RFK is always able to complete execution while achieving highly competitive results.

1822 In addition, we further compare our method with several clustering approaches defined on other manifolds, such as those presented in (Subbarao & Meer, 2009; Ashizawa et al., 2017), and (Zhao et al., 2016). Result is shown in Table 10.

1826 F.2 SENSITIVITY ANALYSIS
18271828 F.2.1 SENSITIVITY ANALYSIS OF m
1829

1830 In addition, we conducted a sensitivity analysis on the parameter m in RFK. The parameter m represents the fuzziness, reflecting the degree of uncertainty in the assignment. In typical implementations of Fuzzy K-Means, m is usually set to the default value of 2. Similarly, in the RFK algorithm, we consistently use the default $m = 2$. This choice is justified because within a sufficiently wide range, the influence of m on the final results is minimal, as illustrated in Figure 5. Specifically, we set $m = \{1.5, 1.75, 2, 2.25, 2.5\}$ and computed the evaluation metrics. It can be observed that $m = 2$ consistently achieves good performance, and the metrics vary only slightly with changes in m .

Table 8: F1 for all benchmarks. OT means out-of-time

	Dataset	Signature	RFK	NRK	K-Means	Ncut	FCM	UFCM	LRR	SSC	SBMC	USPEC	Fast-CD
Synthetic	Gaussian	\mathbb{R}^4	95.49	95.35	93.53	98.30	93.50	93.65	82.33	41.26	68.74	79.84	95.39
		\mathbb{H}^4	99.77	98.42	48.25	98.26	51.35	46.87	99.66	50.51	52.04	90.09	60.52
	$\mathbb{S}^2\mathbb{H}^2$	94.60	94.55	81.35	83.48	83.06	82.66	93.81	51.11	61.03	79.84	83.17	
		$\mathbb{R}^2\mathbb{S}^2\mathbb{H}^2$	96.27	96.27	62.32	98.85	61.73	62.31	89.57	54.79	74.80	54.41	74.35
		$\mathbb{S}^2(\mathbb{H}^2)^2$	98.17	98.05	52.07	66.03	51.15	52.51	92.97	52.41	60.87	67.71	55.66
		$\mathbb{R}^4\mathbb{S}^4\mathbb{H}^4$	99.09	98.25	48.18	74.07	72.15	48.17	94.52	48.71	79.31	69.17	92.48
	Graph	$\mathbb{R}^{16}\mathbb{S}^{16}\mathbb{H}^{16}$	97.75	64.93	50.26	70.18	48.14	50.46	50.90	50.46	46.16	54.03	46.08
		$(\mathbb{H}^2)^2$	0.07	18.47	19.21	31.32	18.29	20.22	18.23	31.98	18.00	23.74	19.31
		\mathbb{H}^4	0.06	16.89	16.52	20.77	16.17	17.72	16.28	30.12	16.08	20.64	16.15
		$(\mathbb{S}^2)^2$	94.33	93.60	88.56	64.80	87.92	88.74	64.53	66.66	50.99	53.83	88.21
		\mathbb{D}	64.74	64.45	56.54	54.59	54.90	56.59	32.96	53.55	42.44	55.14	52.51
		\mathbb{D}	47.75	46.08	40.14	36.40	39.36	41.22	16.01	14.95	17.98	40.74	35.41
	VAE	\mathbb{D}	72.60	57.23	66.96	67.24	66.20	66.85	43.29	71.12	50.09	63.27	70.79
		CIFAR-100	$(\mathbb{H}^2)^4$	69.08	OT	6.83	OT	6.01	8.71	OT	OT	OT	6.57
		Lymphoma	$(\mathbb{S}^2)^2$	100.00	OT	79.51	OT	79.51	OT	OT	OT	OT	79.51
	MNIST	$\mathbb{S}^2\mathbb{E}^2\mathbb{H}^2$	96.18	OT	18.17	OT	18.13	18.18	OT	OT	OT	18.21	OT

Table 9: Purity for all benchmarks. OT means out-of-time

	Dataset	Signature	RFK	NRK	K-Means	Ncut	FCM	UFCM	LRR	SSC	SBMC	USPEC	Fast-CD
Synthetic	Gaussian	\mathbb{R}^4	96.00	95.84	93.80	98.47	93.77	93.92	81.75	34.58	69.47	66.61	97.20
		\mathbb{H}^4	99.80	99.00	34.25	98.62	43.13	34.14	99.62	35.24	52.68	86.04	66.50
	$\mathbb{S}^2\mathbb{H}^2$	95.20	94.80	79.31	83.73	83.36	82.93	95.02	34.40	61.78	66.63	87.70	
		$\mathbb{R}^2\mathbb{S}^2\mathbb{H}^2$	96.20	95.80	54.55	98.76	62.12	53.89	91.64	37.77	79.25	37.75	86.00
		$\mathbb{S}^2(\mathbb{H}^2)^2$	97.80	97.80	37.22	61.24	37.37	37.32	92.65	37.25	64.56	51.98	68.50
		$\mathbb{R}^4\mathbb{S}^4\mathbb{H}^4$	99.10	98.90	33.81	59.43	59.73	34.22	94.64	33.55	79.18	54.16	95.90
	Graph	$\mathbb{R}^{16}\mathbb{S}^{16}\mathbb{H}^{16}$	98.00	77.10	34.17	66.26	34.23	34.18	34.22	34.10	46.59	52.82	57.20
		$(\mathbb{H}^2)^2$	25.36	20.09	19.29	18.98	19.17	19.26	19.28	19.05	19.15	19.33	25.31
		\mathbb{H}^4	29.22	18.19	17.89	17.55	17.93	17.87	17.91	17.75	17.81	17.56	29.22
		$(\mathbb{S}^2)^2$	94.36	93.62	88.49	50.38	87.85	88.70	54.88	50.04	50.96	52.37	93.70
		\mathbb{D}	76.38	76.38	73.69	68.40	74.45	71.36	50.10	65.71	61.15	58.85	71.20
		\mathbb{D}	58.78	59.51	58.21	57.57	59.82	54.04	35.78	14.32	35.19	51.19	56.30
	VAE	\mathbb{D}	81.90	77.14	78.80	78.10	77.14	77.62	55.24	79.05	76.48	75.05	80.95
		CIFAR-100	$(\mathbb{H}^2)^4$	79.57	OT	5.04	OT	5.08	5.01	OT	OT	OT	5.00
		Lymphoma	$(\mathbb{S}^2)^2$	100.00	OT	65.99	OT	65.99	OT	OT	OT	65.99	OT
	MNIST	$\mathbb{S}^2\mathbb{E}^2\mathbb{H}^2$	96.09	OT	10.06	OT	10.61	10.23	OT	OT	OT	10.03	OT

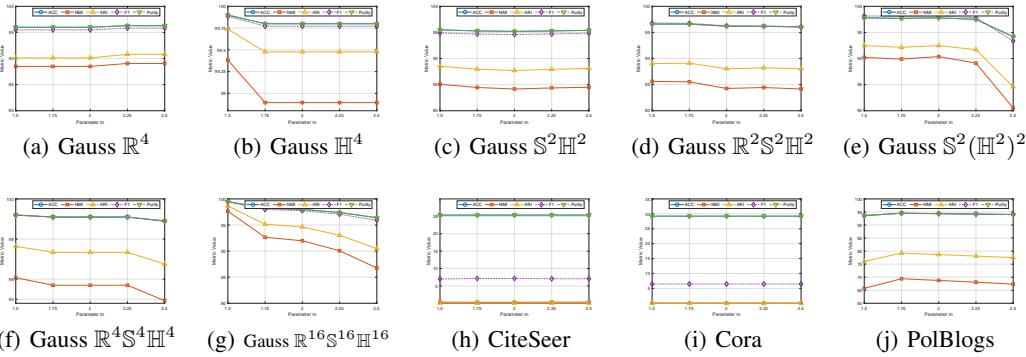


Figure 5: Sensitivity Analysis of m

F.2.2 SENSITIVITY ANALYSIS OF RANDOM INITIALIZATION

We have provided the complete implementation of Riemannian Fuzzy K-Means. It is worth noting that our algorithm adopts random initialization of cluster centers. Therefore, it is necessary to include a sensitivity analysis with respect to random initialization.

We have released the full experimental code from the original paper and fixed all parameters and random seeds. In our experiments, the default seed is set to 1. To assess the robustness of random initialization, we additionally run the algorithm on several datasets with seeds set to 2, 3, and 4, respectively, and obtain the following results:

The column origin corresponds to our default random seed, while each subsequent column reports results obtained using different random seeds. As shown, our algorithm is not sensitive to random initialization of cluster centers. When using different seeds, most of the best results are even better

1890	Method (ACC)	RFK	NMS	LSLDGC	KGRP
1891	Gaussian \mathbb{R}^4	96.00	95.40	94.30	95.40
1892	Gaussian \mathbb{H}^4	99.80	94.40	96.30	98.10
1893	Gaussian $\mathbb{S}^2\mathbb{H}^2$	95.20	94.50	93.10	95.00
1894	Gaussian $\mathbb{R}^2\mathbb{S}^2\mathbb{H}^2$	96.20	95.10	94.70	94.70
1895	Gaussian $\mathbb{S}^2(\mathbb{H}^2)^2$	97.80	90.20	96.40	96.00
1896	Gaussian $\mathbb{R}^4\mathbb{S}^4\mathbb{H}^4$	99.10	99.00	98.80	95.00
1897	Gaussian $\mathbb{R}^{16}\mathbb{S}^{16}\mathbb{H}^{16}$	98.00	88.10	90.50	94.90
1898	CiteSeer	25.36	20.04	21.66	22.23
1899	Cora	29.22	26.27	25.03	20.93
1900	PolBlogs	94.36	94.07	92.94	90.45
1901	CIFAR-100	71.19	OT	OT	OT
1902	Lymphoma	100.00	OT	OT	OT
1903	MNIST	96.09	OT	OT	OT

Table 10: Comparison of RFK with other manifold-based clustering methods.

1905	Seed (ACC)	origin	seed=2	seed=3	seed=4
1906	Gaussian \mathbb{R}^4	96.00	96.00	96.00	96.10
1907	Gaussian \mathbb{H}^4	99.80	99.80	99.80	99.80
1908	Gaussian $\mathbb{S}^2\mathbb{H}^2$	95.20	95.30	95.10	95.50
1909	CiteSeer	25.36	25.45	25.21	24.79
1910	Cora	29.22	29.25	28.49	29.22
1911	PolBlogs	94.36	93.62	93.78	94.68
1912	CIFAR-100	71.19	71.23	71.11	71.16
1913	Lymphoma	100.00	100.00	100.00	100.00
1914	MNIST	96.09	95.91	96.12	95.93

Table 11: Sensitivity analysis of random initialization.

than those originally reported, and in all cases, the clustering performance obtained with different seeds remains close to our reported results.

F.2.3 SENSITIVITY ANALYSIS OF THE NUMBER OF CLUSTER CENTERS

In Riemannian Fuzzy K-Means, an important hyperparameter is the number of cluster centers. Typically, the number of clusters is set to the known number of classes in the dataset. Nevertheless, it is still necessary to analyze the sensitivity of the algorithm to this hyperparameter.

We primarily conduct the sensitivity analysis on the GAUSS datasets, each of which contains three classes regardless of dimensionality. Accordingly, we vary the number of cluster centers to 2, 4, and 5, and evaluate how the clustering ACC changes. In addition, we include real datasets in our analysis, with the true number of classes labeled inside the table.

1929	Data (ACC)	$c = c_{\text{real}}$	$c = 2$	$c = 4$	$c = 5$
1930	Gaussian \mathbb{R}^4	96.00	67.10	91.40	64.40
1931	Gaussian \mathbb{H}^4	99.80	73.80	87.00	64.40
1932	Gaussian $\mathbb{S}^2\mathbb{H}^2$	95.20	72.90	90.30	65.90
1933	Gaussian $\mathbb{R}^2\mathbb{S}^2\mathbb{H}^2$	96.20	58.60	75.20	59.10
1934	Gaussian $\mathbb{S}^2(\mathbb{H}^2)^2$	97.80	74.20	79.80	61.00
1935	Gaussian $\mathbb{R}^4\mathbb{S}^4\mathbb{H}^4$	99.10	69.30	90.50	94.60
1936	Gaussian $\mathbb{R}^{16}\mathbb{S}^{16}\mathbb{H}^{16}$	98.00	70.40	83.50	97.40
1937	CiteSeer ($c_{\text{real}} = 6$)	25.36	25.26(c=5)	23.60(c=7)	24.93(c=8)
1938	Cora ($c_{\text{real}} = 7$)	29.22	29.21(c=6)	23.98(c=8)	28.37(c=9)
1939	PolBlogs ($c_{\text{real}} = 2$)	94.36	70.70(c=3)	59.00(c=4)	42.79(c=5)

Table 12: Sensitivity analysis of the number of cluster centers.

We observe that Riemannian Fuzzy K-Means is relatively sensitive to the number of cluster centers. In fact, this is a well-known property of the entire K-Means family, which explains why the number of centers is the most critical hyperparameter in K-Means-type algorithms.

1944 G RUN AND REFERENCE CODE

1945

1946 G.1 RUN THE CODE

1947

1948 The simplest way to run the code is by using the Anonymous Library. The library submitted in
 1949 the supplementary material is not developed by us, but is part of a publicly available open-source
 1950 community library. To ensure the double-blind review process, we have anonymized the library name
 1951 and included only its minimal implementation unit in the supplementary material.

1952 **First, import the package⁶**

1953

1954 `import AnonymousLibrary`

1955

1956 You can simply perform clustering with the following code.

```
1957 pm=AnonymousLibrary.ProductManifold(signature=[(0, 16),(1, 16),(-1, 16)])
1958 # Use classification labels, which identify clusters by their center
1959 X_clustering, y_clustering = pm.gaussian_mixture(
1960     num_points=1000, num_classes=4, seed=2025, task="classification",
1961     cov_scale_points=0.1
1962 )
1963 # The RFK algorithm is essentially a sklearn-styled clustering algorithm,
1964 # so we call it like this:
1965 rfk = AnonymousLibrary.RiemannianFuzzyKMeans(pm=pm, n_clusters=4,
1966     random_state=2025)
1967 rfk.fit(X_clustering)
1968 y_pred = rfk.predict(X_clustering)
1969
1970 from sklearn.metrics import normalized_mutual_info_score
1971 nmi = normalized_mutual_info_score(y_clustering, y_pred)
1972 print(f"Riemannian Fuzzy K-Means nmi: {nmi:.2f}")
```

1973 **We kindly suggest** that during the review process, reviewers refrain from searching for the source
 1974 code related to Riemannian Fuzzy K-Means and Riemannian Adan, as this may violate the double-
 1975 blind policy. We will provide the fully anonymized versions of RFK and Radan later in the paper.

1976

1977 G.2 REPLICATION STATEMENT

1978

1979 We fully understand the astonishment when seeing the experimental results, especially the clustering
 1980 outcomes in Experiment 3. On some datasets, traditional K-Means achieves only **12%** accuracy,
 1981 while RFK reaches **96%**. Reporting such a striking gap obliges the authors to provide code during
 1982 the review stage. We are not only willing to provide the source code of RFK but also offer a DEMO
 1983 that can reproduce the experimental results with a single command, with parameters and random
 1984 seeds fixed for verification. Our code is available [here⁷](https://anonymous.4open.science/r/Demo-of-RFK-243B/), and our datasets are available [here⁸](https://anonymous.4open.science/r/Manifold-Clustering-Data-3C53/).

1985

1986

1987

1988

1989

1990

1991

1992

1993

1994

1995

⁶<https://anonymous.4open.science/status/AnonymousLibrary-32EB>

⁷<https://anonymous.4open.science/r/Demo-of-RFK-243B/>

⁸<https://anonymous.4open.science/r/Manifold-Clustering-Data-3C53/>

```

1998 G.3 CODE OF RIEMANNIAN FUZZY K-MEANS
1999
2000 from __future__ import annotations
2001
2002 from typing import TYPE_CHECKING
2003
2004 import numpy as np
2005 import torch
2006 from geoopt import ManifoldParameter
2007 from geoopt.optim import RiemannianAdam
2008 from sklearn.base import BaseEstimator, ClusterMixin
2009
2010 if TYPE_CHECKING:
2011     from beartype.typing import Literal
2012     from jaxtyping import Float, Int
2013
2014 from ..manifolds import Manifold, ProductManifold
2015 from ..optimizers.radan import RiemannianAdan
2016
2017 class RiemannianFuzzyKMeans(BaseEstimator, ClusterMixin):
2018     """Riemannian Fuzzy K-Means.
2019
2020     Attributes:
2021         n_clusters: The number of clusters to form.
2022         pm: An initialized manifold object (from manifolds.py) on which
2023             clustering will be performed.
2024         m: Fuzzifier parameter. Controls the softness of the partition.
2025         lr: Learning rate for the optimizer.
2026         max_iter: Maximum number of iterations for the optimization.
2027         tol: Tolerance for convergence. If the change in loss is less
2028             than tol, iteration stops.
2029         optimizer: The optimizer to use for updating cluster centers.
2030         random_state: Seed for random number generation for
2031             reproducibility.
2032         verbose: Whether to print loss information during iterations.
2033         losses_: List of loss values during training.
2034         u_: Final fuzzy partition matrix.
2035         labels_: Cluster labels for each sample.
2036         cluster_centers_: Final cluster centers.
2037
2038     Args:
2039         n_clusters: The number of clusters to form.
2040         manifold: An initialized manifold object (from manifolds.py) on
2041             which clustering will be performed.
2042         m: Fuzzifier parameter. Controls the softness of the partition.
2043         lr: Learning rate for the optimizer.
2044         max_iter: Maximum number of iterations for the optimization.
2045         tol: Tolerance for convergence. If the change in loss is less
2046             than tol, iteration stops.
2047         optimizer: The optimizer to use for updating cluster centers.
2048         random_state: Seed for random number generation for
2049             reproducibility.
2050         verbose: Whether to print loss information during iterations.
2051
2052     """
2053
2054     def __init__(
2055         self,
2056         n_clusters: int,
2057         pm: Manifold | ProductManifold,
2058         m: float = 2.0,
2059         lr: float = 0.1,
2060         max_iter: int = 100,
2061         tol: float = 1e-4,
2062         optimizer: Literal["adan", "adam"] = "adan",
2063     ):
2064         self.n_clusters = n_clusters
2065         self.pm = pm
2066         self.m = m
2067         self.lr = lr
2068         self.max_iter = max_iter
2069         self.tol = tol
2070         self.optimizer = optimizer
2071         self.random_state = random_state
2072         self.verbose = verbose
2073         self.losses_ = []
2074         self.u_ = None
2075         self.labels_ = None
2076         self.cluster_centers_ = None
2077
2078     def fit(self, X: np.ndarray, y: np.ndarray | None = None) -> self:
2079         self._check_input(X, y)
2080         self._init_optimizer()
2081         self._init_partitions()
2082         self._init_centers()
2083         for i in range(self.max_iter):
2084             self._update_centers()
2085             self._update_partitions()
2086             self._compute_loss()
2087             if self._check_convergence():
2088                 break
2089         return self
2090
2091     def _check_input(self, X: np.ndarray, y: np.ndarray | None):
2092         if not isinstance(X, np.ndarray):
2093             raise ValueError("X must be a numpy array")
2094         if X.ndim != 2:
2095             raise ValueError("X must be a 2D array")
2096         if X.shape[0] < self.n_clusters:
2097             raise ValueError("X must have at least as many samples as clusters")
2098         if y is not None and not isinstance(y, np.ndarray):
2099             raise ValueError("y must be a numpy array")
2100         if y is not None and y.shape[0] != X.shape[0]:
2101             raise ValueError("y must have the same number of samples as X")
2102
2103     def _init_optimizer(self):
2104         self.optimizer = RiemannianAdam(self.pm, lr=self.lr, m=self.m)
2105
2106     def _init_partitions(self):
2107         self.u_ = np.ones((self.n_clusters, X.shape[0]), dtype=np.float32)
2108
2109     def _init_centers(self):
2110         self.cluster_centers_ = np.zeros((self.n_clusters, X.shape[1]), dtype=np.float32)
2111
2112     def _update_centers(self):
2113         self.optimizer.zero_grad()
2114         self.optimizer.step(self._compute_loss)
2115
2116     def _update_partitions(self):
2117         self.u_ = self._compute_u()
2118
2119     def _compute_u(self):
2120         u = np.zeros((self.n_clusters, X.shape[0]), dtype=np.float32)
2121         for i in range(self.n_clusters):
2122             u[i] = self._compute_u_i(i)
2123         return u
2124
2125     def _compute_u_i(self, i):
2126         u_i = np.zeros((X.shape[0]), dtype=np.float32)
2127         for j in range(self.n_clusters):
2128             u_i += self._compute_u_ij(i, j)
2129         return u_i
2130
2131     def _compute_u_ij(self, i, j):
2132         u_ij = np.zeros((X.shape[0]), dtype=np.float32)
2133         for k in range(X.shape[1]):
2134             u_ij += self._compute_u_ij_k(i, j, k)
2135         return u_ij
2136
2137     def _compute_u_ij_k(self, i, j, k):
2138         u_ij_k = np.zeros((X.shape[0]), dtype=np.float32)
2139         for l in range(X.shape[0]):
2140             u_ij_k[l] = self._compute_u_ij_k_l(i, j, k, l)
2141         return u_ij_k
2142
2143     def _compute_u_ij_k_l(self, i, j, k, l):
2144         u_ij_k_l = np.zeros((1), dtype=np.float32)
2145         for m in range(self.n_clusters):
2146             u_ij_k_l += self._compute_u_ij_k_l_m(i, j, k, l, m)
2147         return u_ij_k_l
2148
2149     def _compute_u_ij_k_l_m(self, i, j, k, l, m):
2150         u_ij_k_l_m = np.zeros((1), dtype=np.float32)
2151         for n in range(X.shape[1]):
2152             u_ij_k_l_m += self._compute_u_ij_k_l_m_n(i, j, k, l, m, n)
2153         return u_ij_k_l_m
2154
2155     def _compute_u_ij_k_l_m_n(self, i, j, k, l, m, n):
2156         u_ij_k_l_m_n = np.zeros((1), dtype=np.float32)
2157         for o in range(X.shape[0]):
2158             u_ij_k_l_m_n += self._compute_u_ij_k_l_m_n_o(i, j, k, l, m, n, o)
2159         return u_ij_k_l_m_n
2160
2161     def _compute_u_ij_k_l_m_n_o(self, i, j, k, l, m, n, o):
2162         u_ij_k_l_m_n_o = np.zeros((1), dtype=np.float32)
2163         for p in range(X.shape[1]):
2164             u_ij_k_l_m_n_o += self._compute_u_ij_k_l_m_n_o_p(i, j, k, l, m, n, o, p)
2165         return u_ij_k_l_m_n_o
2166
2167     def _compute_u_ij_k_l_m_n_o_p(self, i, j, k, l, m, n, o, p):
2168         u_ij_k_l_m_n_o_p = np.zeros((1), dtype=np.float32)
2169         for q in range(X.shape[0]):
2170             u_ij_k_l_m_n_o_p += self._compute_u_ij_k_l_m_n_o_p_q(i, j, k, l, m, n, o, p, q)
2171         return u_ij_k_l_m_n_o_p
2172
2173     def _compute_u_ij_k_l_m_n_o_p_q(self, i, j, k, l, m, n, o, p, q):
2174         u_ij_k_l_m_n_o_p_q = np.zeros((1), dtype=np.float32)
2175         for r in range(X.shape[1]):
2176             u_ij_k_l_m_n_o_p_q += self._compute_u_ij_k_l_m_n_o_p_q_r(i, j, k, l, m, n, o, p, q, r)
2177         return u_ij_k_l_m_n_o_p_q
2178
2179     def _compute_u_ij_k_l_m_n_o_p_q_r(self, i, j, k, l, m, n, o, p, q, r):
2180         u_ij_k_l_m_n_o_p_q_r = np.zeros((1), dtype=np.float32)
2181         for s in range(X.shape[0]):
2182             u_ij_k_l_m_n_o_p_q_r += self._compute_u_ij_k_l_m_n_o_p_q_r_s(i, j, k, l, m, n, o, p, q, r, s)
2183         return u_ij_k_l_m_n_o_p_q_r
2184
2185     def _compute_u_ij_k_l_m_n_o_p_q_r_s(self, i, j, k, l, m, n, o, p, q, r, s):
2186         u_ij_k_l_m_n_o_p_q_r_s = np.zeros((1), dtype=np.float32)
2187         for t in range(X.shape[1]):
2188             u_ij_k_l_m_n_o_p_q_r_s += self._compute_u_ij_k_l_m_n_o_p_q_r_s_t(i, j, k, l, m, n, o, p, q, r, s, t)
2189         return u_ij_k_l_m_n_o_p_q_r_s
2190
2191     def _compute_u_ij_k_l_m_n_o_p_q_r_s_t(self, i, j, k, l, m, n, o, p, q, r, s, t):
2192         u_ij_k_l_m_n_o_p_q_r_s_t = np.zeros((1), dtype=np.float32)
2193         for u in range(X.shape[0]):
2194             u_ij_k_l_m_n_o_p_q_r_s_t += self._compute_u_ij_k_l_m_n_o_p_q_r_s_t_u(i, j, k, l, m, n, o, p, q, r, s, t, u)
2195         return u_ij_k_l_m_n_o_p_q_r_s_t
2196
2197     def _compute_u_ij_k_l_m_n_o_p_q_r_s_t_u(self, i, j, k, l, m, n, o, p, q, r, s, t, u):
2198         u_ij_k_l_m_n_o_p_q_r_s_t_u = np.zeros((1), dtype=np.float32)
2199         for v in range(X.shape[1]):
2200             u_ij_k_l_m_n_o_p_q_r_s_t_u += self._compute_u_ij_k_l_m_n_o_p_q_r_s_t_u_v(i, j, k, l, m, n, o, p, q, r, s, t, u, v)
2201         return u_ij_k_l_m_n_o_p_q_r_s_t_u
2202
2203     def _compute_u_ij_k_l_m_n_o_p_q_r_s_t_u_v(self, i, j, k, l, m, n, o, p, q, r, s, t, u, v):
2204         u_ij_k_l_m_n_o_p_q_r_s_t_u_v = np.zeros((1), dtype=np.float32)
2205         for w in range(X.shape[0]):
2206             u_ij_k_l_m_n_o_p_q_r_s_t_u_v += self._compute_u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w(i, j, k, l, m, n, o, p, q, r, s, t, u, v, w)
2207         return u_ij_k_l_m_n_o_p_q_r_s_t_u_v
2208
2209     def _compute_u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w(self, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w):
2210         u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w = np.zeros((1), dtype=np.float32)
2211         for x in range(X.shape[1]):
2212             u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w += self._compute_u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x(i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x)
2213         return u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w
2214
2215     def _compute_u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x(self, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x):
2216         u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x = np.zeros((1), dtype=np.float32)
2217         for y in range(X.shape[0]):
2218             u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x += self._compute_u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y(i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y)
2219         return u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x
2220
2221     def _compute_u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y(self, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y):
2222         u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y = np.zeros((1), dtype=np.float32)
2223         for z in range(X.shape[1]):
2224             u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y += self._compute_u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z(i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z)
2225         return u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y
2226
2227     def _compute_u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z(self, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z):
2228         u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z = np.zeros((1), dtype=np.float32)
2229         for a in range(X.shape[0]):
2230             u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z += self._compute_u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a(i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, a)
2231         return u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z
2232
2233     def _compute_u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a(self, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, a):
2234         u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a = np.zeros((1), dtype=np.float32)
2235         for b in range(X.shape[1]):
2236             u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a += self._compute_u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b(i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, a, b)
2237         return u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a
2238
2239     def _compute_u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b(self, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, a, b):
2240         u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b = np.zeros((1), dtype=np.float32)
2241         for c in range(X.shape[0]):
2242             u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b += self._compute_u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c(i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, a, b, c)
2243         return u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b
2244
2245     def _compute_u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c(self, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, a, b, c):
2246         u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c = np.zeros((1), dtype=np.float32)
2247         for d in range(X.shape[1]):
2248             u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c += self._compute_u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d(i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, a, b, c, d)
2249         return u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c
2250
2251     def _compute_u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d(self, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, a, b, c, d):
2252         u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d = np.zeros((1), dtype=np.float32)
2253         for e in range(X.shape[0]):
2254             u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d += self._compute_u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e(i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, a, b, c, d, e)
2255         return u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d
2256
2257     def _compute_u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e(self, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, a, b, c, d, e):
2258         u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e = np.zeros((1), dtype=np.float32)
2259         for f in range(X.shape[1]):
2260             u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e += self._compute_u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f(i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, a, b, c, d, e, f)
2261         return u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e
2262
2263     def _compute_u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f(self, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, a, b, c, d, e, f):
2264         u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f = np.zeros((1), dtype=np.float32)
2265         for g in range(X.shape[0]):
2266             u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f += self._compute_u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g(i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, a, b, c, d, e, f, g)
2267         return u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f
2268
2269     def _compute_u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g(self, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, a, b, c, d, e, f, g):
2270         u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g = np.zeros((1), dtype=np.float32)
2271         for h in range(X.shape[1]):
2272             u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g += self._compute_u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h(i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, a, b, c, d, e, f, g, h)
2273         return u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g
2274
2275     def _compute_u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h(self, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, a, b, c, d, e, f, g, h):
2276         u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h = np.zeros((1), dtype=np.float32)
2277         for i in range(X.shape[0]):
2278             u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h += self._compute_u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i(i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, a, b, c, d, e, f, g, h, i)
2279         return u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h
2280
2281     def _compute_u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i(self, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, a, b, c, d, e, f, g, h, i):
2282         u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i = np.zeros((1), dtype=np.float32)
2283         for j in range(X.shape[1]):
2284             u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i += self._compute_u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j(i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, a, b, c, d, e, f, g, h, i, j)
2285         return u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i
2286
2287     def _compute_u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j(self, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, a, b, c, d, e, f, g, h, i, j):
2288         u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j = np.zeros((1), dtype=np.float32)
2289         for k in range(X.shape[0]):
2290             u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j += self._compute_u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j_k(i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, a, b, c, d, e, f, g, h, i, j, k)
2291         return u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j
2292
2293     def _compute_u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j_k(self, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, a, b, c, d, e, f, g, h, i, j, k):
2294         u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j_k = np.zeros((1), dtype=np.float32)
2295         for l in range(X.shape[1]):
2296             u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j_k += self._compute_u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j_k_l(i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, a, b, c, d, e, f, g, h, i, j, k, l)
2297         return u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j_k
2298
2299     def _compute_u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j_k_l(self, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, a, b, c, d, e, f, g, h, i, j, k, l):
2300         u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j_k_l = np.zeros((1), dtype=np.float32)
2301         for m in range(X.shape[0]):
2302             u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j_k_l += self._compute_u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j_k_l_m(i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, a, b, c, d, e, f, g, h, i, j, k, l, m)
2303         return u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j_k_l
2304
2305     def _compute_u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j_k_l_m(self, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, a, b, c, d, e, f, g, h, i, j, k, l, m):
2306         u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j_k_l_m = np.zeros((1), dtype=np.float32)
2307         for n in range(X.shape[1]):
2308             u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j_k_l_m += self._compute_u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j_k_l_m_n(i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, a, b, c, d, e, f, g, h, i, j, k, l, m, n)
2309         return u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j_k_l_m
2310
2311     def _compute_u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j_k_l_m_n(self, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, a, b, c, d, e, f, g, h, i, j, k, l, m, n):
2312         u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j_k_l_m_n = np.zeros((1), dtype=np.float32)
2313         for o in range(X.shape[0]):
2314             u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j_k_l_m_n += self._compute_u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j_k_l_m_n_o(i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, a, b, c, d, e, f, g, h, i, j, k, l, m, n, o)
2315         return u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j_k_l_m_n
2316
2317     def _compute_u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j_k_l_m_n_o(self, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, a, b, c, d, e, f, g, h, i, j, k, l, m, n, o):
2318         u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j_k_l_m_n_o = np.zeros((1), dtype=np.float32)
2319         for p in range(X.shape[1]):
2320             u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j_k_l_m_n_o += self._compute_u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j_k_l_m_n_o_p(i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p)
2321         return u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j_k_l_m_n_o
2322
2323     def _compute_u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j_k_l_m_n_o_p(self, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p):
2324         u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j_k_l_m_n_o_p = np.zeros((1), dtype=np.float32)
2325         for q in range(X.shape[0]):
2326             u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j_k_l_m_n_o_p += self._compute_u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j_k_l_m_n_o_p_q(i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q)
2327         return u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j_k_l_m_n_o_p
2328
2329     def _compute_u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j_k_l_m_n_o_p_q(self, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q):
2330         u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j_k_l_m_n_o_p_q = np.zeros((1), dtype=np.float32)
2331         for r in range(X.shape[1]):
2332             u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j_k_l_m_n_o_p_q += self._compute_u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j_k_l_m_n_o_p_q_r(i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r)
2333         return u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j_k_l_m_n_o_p_q
2334
2335     def _compute_u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j_k_l_m_n_o_p_q_r(self, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r):
2336         u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j_k_l_m_n_o_p_q_r = np.zeros((1), dtype=np.float32)
2337         for s in range(X.shape[0]):
2338             u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j_k_l_m_n_o_p_q_r += self._compute_u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j_k_l_m_n_o_p_q_r_s(i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s)
2339         return u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j_k_l_m_n_o_p_q_r
2340
2341     def _compute_u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j_k_l_m_n_o_p_q_r_s(self, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s):
2342         u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j_k_l_m_n_o_p_q_r_s = np.zeros((1), dtype=np.float32)
2343         for t in range(X.shape[1]):
2344             u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j_k_l_m_n_o_p_q_r_s += self._compute_u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j_k_l_m_n_o_p_q_r_s_t(i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t)
2345         return u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j_k_l_m_n_o_p_q_r_s
2346
2347     def _compute_u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j_k_l_m_n_o_p_q_r_s_t(self, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t):
2348         u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j_k_l_m_n_o_p_q_r_s_t = np.zeros((1), dtype=np.float32)
2349         for u in range(X.shape[0]):
2350             u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j_k_l_m_n_o_p_q_r_s_t += self._compute_u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j_k_l_m_n_o_p_q_r_s_t_u(i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u)
2351         return u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j_k_l_m_n_o_p_q_r_s_t
2352
2353     def _compute_u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j_k_l_m_n_o_p_q_r_s_t_u(self, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u):
2354         u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j_k_l_m_n_o_p_q_r_s_t_u = np.zeros((1), dtype=np.float32)
2355         for v in range(X.shape[1]):
2356             u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j_k_l_m_n_o_p_q_r_s_t_u += self._compute_u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j_k_l_m_n_o_p_q_r_s_t_u_v(i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v)
2357         return u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j_k_l_m_n_o_p_q_r_s_t_u
2358
2359     def _compute_u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j_k_l_m_n_o_p_q_r_s_t_u_v(self, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v):
2360         u_ij_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_a_b_c_d_e_f_g_h_i_j_k_l_m_n_o_p_q_r_s_t_u
```

```

2052     random_state: int | None = None,
2053     verbose: bool = False,
2054     ):
2055         self.n_clusters = n_clusters
2056         self.pm = pm
2057         self.m = m
2058         self.lr = lr
2059         self.max_iter = max_iter
2060         self.tol = tol
2061         if optimizer not in ("adan", "adam"):
2062             raise ValueError("optimizer must be 'adan' or 'adam'")
2063         self.optimizer = optimizer
2064         self.random_state = random_state
2065         self.verbose = verbose
2066
2067     def __init__(self, X: Float[torch.Tensor, "n_points n_features"]
2068     ) -> None:
2069         if self.random_state is not None:
2070             torch.manual_seed(self.random_state)
2071             np.random.seed(self.random_state)
2072
2073         # Input data X's second dimension should match the pm's ambient
2074         # dimension
2075         if X.shape[1] != self.pm.ambient_dim:
2076             raise ValueError(
2077                 f"Input data X's dimension ({X.shape[1]}) does not match
2078                 "
2079                 f"the manifold's ambient dimension ({self.pm.ambient_dim
2080                 })."
2081             )
2082
2083         # Generate initial centers using the manifold's sample method
2084         # We want n_clusters points, each sampled around the manifold's
2085         # origin (mu0)
2086         # The .sample() method in manifolds.py handles z_mean and sigma/
2087         # sigma_factorized
2088         # defaulting to mu0 and identity covariances if z_mean or sigma
2089         # are not fully specified
2090         # or are set to None in a way that triggers this default.
2091
2092         # For sampling initial centers, we want n_clusters distinct
2093         # points.
2094         # The .sample() method typically takes a z_mean of shape (
2095         #     num_points_to_sample, ambient_dim).
2096         # If we provide self.pm.mu0 repeated n_clusters times,
2097         # it samples n_clusters points, each around mu0.
2098         centers = self.pm.sample(self.n_clusters)
2099
2100         # IMPORTANT: Use self.manifold.manifold for ManifoldParameter,
2101         # as self.manifold is our wrapper and self.manifold.manifold is
2102         # the geoopt object.
2103         self.mu_ = ManifoldParameter(
2104             centers.clone().detach(), # type: ignore
2105             manifold=self.pm.manifold,
2106         ) # Ensure centers are detached
2107         self.mu_.requires_grad_(True)
2108
2109         if self.optimizer == "adan":
2110             self.opt_ = RiemannianAdan([self.mu_], lr=self.lr, betas
2111             =[0.7, 0.999, 0.999])
2112         else:
2113             self.opt_ = RiemannianAdam([self.mu_], lr=self.lr, betas
2114             =[0.99, 0.999])
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2989
2990
2991
2992
2993
2994
2995
2996
2997
2998
2999
2999

```

```

2106     def fit(self, X: Float[torch.Tensor, "n_points n_features"], y: None
2107         = None) -> "RiemannianFuzzyKMeans":
2108         """Fit the Riemannian Fuzzy K-Means model to the data X.
2109
2110         Args:
2111             X: Input data. Features should match the manifold's geometry.
2112             y: Ignored, present for compatibility with scikit-learn's API
2113
2114         Returns:
2115             self: Fitted RiemannianFuzzyKMeans instance.
2116
2117         Raises:
2118             ValueError: If the input data's dimension does not match the
2119                         manifold's ambient dimension.
2120             RuntimeError: If the optimizer is not set correctly or if the
2121                           model has not been initialized properly.
2122
2123         """
2124         if isinstance(X, np.ndarray):
2125             X = torch.from_numpy(X).type(torch.get_default_dtype())
2126         elif not isinstance(X, torch.Tensor):
2127             X = torch.tensor(X, dtype=torch.get_default_dtype())
2128
2129         # Ensure X is on the same device as the manifold
2130         X = X.to(self.pm.device)
2131
2132         if X.shape[1] != self.pm.ambient_dim:
2133             raise ValueError(
2134                 f"Input data X's dimension ({X.shape[1]}) in fit() does
2135                 not match "
2136                 f"the manifold's ambient dimension ({self.pm.ambient_dim
2137                         })."
2138         )
2139
2140         self._init_centers(X)
2141         m, tol = self.m, self.tol
2142         losses = []
2143         for i in range(self.max_iter):
2144             self.opt_.zero_grad()
2145             # self.pm.dist is implemented in manifolds.py and handles
2146             # broadcasting
2147             d = self.pm.dist(X, self.mu_) # X is (N,D), mu_ is (K,D) ->
2148             # d is (N,K)
2149             # Original RFK: d = self.pm.dist(X.unsqueeze(1), self.mu_.
2150             #                                     unsqueeze(0))
2151             # The .dist in manifolds.py uses X[:, None] and Y[None, :],
2152             # so direct call should work if mu_ is (K,D)
2153
2154             S = torch.sum(d.pow(-2 / (m - 1)) + 1e-8, dim=1) # Add
2155             # epsilon for stability
2156             loss = torch.sum(S.pow(1 - m))
2157             loss.backward()
2158             losses.append(loss.item())
2159             self.opt_.step()
2160             if self.verbose:
2161                 print(f"RFK iter {i + 1}, loss={loss.item():.4f}")
2162             if i > 0 and abs(losses[-1] - losses[-2]) < tol:
2163                 break
2164
2165         # save the result
2166         self.losses_ = np.array(losses)
2167         with torch.no_grad(): # Ensure no gradients are computed for
2168             # final calculations
2169             dfin = self.pm.dist(X, self.mu_) # Re-calculate dist to
2170             # final centers

```

```

2160     inv = dfin.pow(-2 / (m - 1)) + 1e-8 # Add epsilon
2161     u_final = inv / (inv.sum(dim=1, keepdim=True) + 1e-8) # Add
2162     epsilon
2163     self.u_ = u_final.detach().cpu().numpy()
2164     self.labels_ = np.argmax(self.u_, axis=1)
2165     self.cluster_centers_ = self.mu_.data.clone().detach().cpu().
2166     numpy()
2167     return self
2168
2169     def predict(self, X: Float[torch.Tensor, "n_points n_features"]) ->
2170     Int[torch.Tensor, "n_points"]:
2171         """Predict the closest cluster each sample in X belongs to.
2172
2173         Args:
2174             X: Input data. Features should match the manifold's geometry.
2175
2176         Returns:
2177             labels: Cluster labels for each sample in X.
2178
2179         Raises:
2180             ValueError: If the input data's dimension does not match the
2181             manifold's ambient dimension.
2182             RuntimeError: If the model has not been fitted yet.
2183
2184         """
2185         if isinstance(X, np.ndarray):
2186             X = torch.from_numpy(X).type(torch.get_default_dtype())
2187         elif not isinstance(X, torch.Tensor):
2188             X = torch.tensor(X, dtype=torch.get_default_dtype())
2189
2190         # Ensure X is on the same device as the manifold
2191         X = X.to(self.pm.device)
2192
2193         if X.shape[1] != self.pm.ambient_dim:
2194             raise ValueError(
2195                 f"Input data X's dimension ({X.shape[1]}) in predict()
2196                 does not match "
2197                 f"the manifold's ambient dimension ({self.pm.ambient_dim
2198                 })."
2199             )
2200
2201         if not hasattr(self, "mu_") or self.mu_ is None:
2202             raise RuntimeError("The RFK model has not been fitted yet.
2203                 Call 'fit' before 'predict'.")
2204
2205         with torch.no_grad():
2206             dmat = self.pm.dist(X, self.mu_) # X is (N,D), mu_ is (K,D)
2207             -> dmat is (N,K)
2208             inv = dmat.pow(-2 / (self.m - 1)) + 1e-8 # Add epsilon
2209             u = inv / (inv.sum(dim=1, keepdim=True) + 1e-8) # Add
2210             epsilon
2211             labels = torch.argmax(u, dim=1).cpu().numpy()
2212             return labels
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2989
2990
2991
2992
2993
2994
2995
2996
2997
2998
2999
2999
3000
3001
3002
3003
3004
3005
3006
3007
3008
3009
3009
3010
3011
3012
3013
3014
3015
3016
3017
3018
3019
3019
3020
3021
3022
3023
3024
3025
3026
3027
3028
3029
3029
3030
3031
3032
3033
3034
3035
3036
3037
3038
3039
3039
3040
3041
3042
3043
3044
3045
3046
3047
3048
3049
3049
3050
3051
3052
3053
3054
3055
3056
3057
3058
3059
3059
3060
3061
3062
3063
3064
3065
3066
3067
3068
3069
3069
3070
3071
3072
3073
3074
3075
3076
3077
3078
3079
3079
3080
3081
3082
3083
3084
3085
3086
3087
3088
3089
3089
3090
3091
3092
3093
3094
3095
3096
3097
3098
3098
3099
3099
3100
3100
3101
3101
3102
3102
3103
3103
3104
3104
3105
3105
3106
3106
3107
3107
3108
3108
3109
3109
3110
3110
3111
3111
3112
3112
3113
3113
3114
3114
3115
3115
3116
3116
3117
3117
3118
3118
3119
3119
3120
3120
3121
3121
3122
3122
3123
3123
3124
3124
3125
3125
3126
3126
3127
3127
3128
3128
3129
3129
3130
3130
3131
3131
3132
3132
3133
3133
3134
3134
3135
3135
3136
3136
3137
3137
3138
3138
3139
3139
3140
3140
3141
3141
3142
3142
3143
3143
3144
3144
3145
3145
3146
3146
3147
3147
3148
3148
3149
3149
3150
3150
3151
3151
3152
3152
3153
3153
3154
3154
3155
3155
3156
3156
3157
3157
3158
3158
3159
3159
3160
3160
3161
3161
3162
3162
3163
3163
3164
3164
3165
3165
3166
3166
3167
3167
3168
3168
3169
3169
3170
3170
3171
3171
3172
3172
3173
3173
3174
3174
3175
3175
3176
3176
3177
3177
3178
3178
3179
3179
3180
3180
3181
3181
3182
3182
3183
3183
3184
3184
3185
3185
3186
3186
3187
3187
3188
3188
3189
3189
3190
3190
3191
3191
3192
3192
3193
3193
3194
3194
3195
3195
3196
3196
3197
3197
3198
3198
3199
3199
3200
3200
3201
3201
3202
3202
3203
3203
3204
3204
3205
3205
3206
3206
3207
3207
3208
3208
3209
3209
3210
3210
3211
3211
3212
3212
3213
3213
3214
3214
3215
3215
3216
3216
3217
3217
3218
3218
3219
3219
3220
3220
3221
3221
3222
3222
3223
3223
3224
3224
3225
3225
3226
3226
3227
3227
3228
3228
3229
3229
3230
3230
3231
3231
3232
3232
3233
3233
3234
3234
3235
3235
3236
3236
3237
3237
3238
3238
3239
3239
3240
3240
3241
3241
3242
3242
3243
3243
3244
3244
3245
3245
3246
3246
3247
3247
3248
3248
3249
3249
3250
3250
3251
3251
3252
3252
3253
3253
3254
3254
3255
3255
3256
3256
3257
3257
3258
3258
3259
3259
3260
3260
3261
3261
3262
3262
3263
3263
3264
3264
3265
3265
3266
3266
3267
3267
3268
3268
3269
3269
3270
3270
3271
3271
3272
3272
3273
3273
3274
3274
3275
3275
3276
3276
3277
3277
3278
3278
3279
3279
3280
3280
3281
3281
3282
3282
3283
3283
3284
3284
3285
3285
3286
3286
3287
3287
3288
3288
3289
3289
3290
3290
3291
3291
3292
3292
3293
3293
3294
3294
3295
3295
3296
3296
3297
3297
3298
3298
3299
3299
3300
3300
3301
3301
3302
3302
3303
3303
3304
3304
3305
3305
3306
3306
3307
3307
3308
3308
3309
3309
3310
3310
3311
3311
3312
3312
3313
3313
3314
3314
3315
3315
3316
3316
3317
3317
3318
3318
3319
3319
3320
3320
3321
3321
3322
3322
3323
3323
3324
3324
3325
3325
3326
3326
3327
3327
3328
3328
3329
3329
3330
3330
3331
3331
3332
3332
3333
3333
3334
3334
3335
3335
3336
3336
3337
3337
3338
3338
3339
3339
3340
3340
3341
3341
3342
3342
3343
3343
3344
3344
3345
3345
3346
3346
3347
3347
3348
3348
3349
3349
3350
3350
3351
3351
3352
3352
3353
3353
3354
3354
3355
3355
3356
3356
3357
3357
3358
3358
3359
3359
3360
3360
3361
3361
3362
3362
3363
3363
3364
3364
3365
3365
3366
3366
3367
3367
3368
3368
3369
3369
3370
3370
3371
3371
3372
3372
3373
3373
3374
3374
3375
3375
3376
3376
3377
3377
3378
3378
3379
3379
3380
3380
3381
3381
3382
3382
3383
3383
3384
3384
3385
3385
3386
3386
3387
3387
3388
3388
3389
3389
3390
3390
3391
3391
3392
3392
3393
3393
3394
3394
3395
3395
3396
3396
3397
3397
3398
3398
3399
3399
3400
3400
3401
3401
3402
3402
3403
3403
3404
3404
3405
3405
3406
3406
3407
3407
3408
3408
3409
3409
3410
3410
3411
3411
3412
3412
3413
3413
3414
3414
3415
3415
3416
3416
3417
3417
3418
3418
3419
3419
3420
3420
3421
3421
3422
3422
3423
3423
3424
3424
3425
3425
3426
3426
3427
3427
3428
3428
3429
3429
3430
3430
3431
3431
3432
3432
3433
3433
3434
3434
3435
3435
3436
3436
3437
3437
3438
3438
3439
3439
3440
3440
3441
3441
3442
3442
3443
3443
3444
3444
3445
3445
3446
3446
3447
3447
3448
3448
3449
3449
3450
3450
3451
3451
3452
3452
3453
3453
3454
3454
3455
3455
3456
3456
3457
3457
3458
3458
3459
3459
3460
3460
3461
3461
3462
3462
3463
3463
3464
3464
3465
3465
3466
3466
3467
3467
3468
3468
3469
3469
3470
3470
3471
3471
3472
3472
3473
3473
3474
3474
3475
3475
3476
3476
3477
3477
3478
3478
3479
3479
3480
3480
3481
3481
3482
3482
3483
3483
3484
3484
3485
3485
3486
3486
3487
3487
3488
3488
3489
3489
3490
3490
3491
3491
3492
3492
3493
3493
3494
3494
3495
3495
3496
3496
3497
3497
3498
3498
3499
3499
3500
3500
3501
3501
3502
3502
3503
3503
3504
3504
3505
3505
3506
3506
3507
3507
3508
3508
3509
3509
3510
3510
3511
3511
3512
3512
3513
3513
3514
3514
3515
3515
3516
3516
3517
3517
3518
3518
3519
3519
3520
3520
3521
3521
3522
3522
3523
3523
3524
3524
3525
3525
3526
3526
3527
3527
3528
3528
3529
3529
3530
3530
3531
3531
3532
3532
3533
3533
3534
3534
3535
3535
3536
3536
3537
3537
3538
3538
3539
3539
3540
3540
3541
3541
3542
3542
3543
3543
3544
3544
3545
3545
3546
3546
3547
3547
3548
3548
3549
3549
3550
3550
3551
3551
3552
3552
3553
3553
3554
3554
3555
3555
3556
3556
3557
3557
3558
3558
3559
3559
3560
3560
3561
3561
3562
3562
3563
3563
3564
3564
3565
3565
3566
3566
3567
3567
3568
3568
3569
3569
3570
3570
3571
3571
3572
3572
3573
3573
3574
3574
3575
3575
3576
3576
3577
3577
3578
3578
3579
3579
3580
3580
3581
3581
3582
3582
3583
3583
3584
3584
3585
3585
3586
3586
3587
3587
3588
3588
3589
3589
3590
3590
3591
3591
3592
3592
3593
3593
3594
3594
3595
3595
3596
3596
3597
3597
3598
3598
3599
3599
3600
3600
3601
3601
3602
3602
3603
3603
3604
3604
3605
3605
3606
3606
3607
3607
3608
3608
3609
3609
3610
3610
3611
3611
3612
3612
3613
3613
3614
3614
3615
3615
3616
3616
3617
3617
3618
3618
3619
3619
3620
3620
3621
3621
3622
3622
3623
3623
3624
3624
3625
3625
3626
3626
3627
3627
3628
3628
3629
3629
3630
3630
3631
3631
3632
3632
3633
3633
3634
3634
3635
3635
3636
3636
3637
3637
3638
3638
3639
3639
3640
3640
3641
3641
3642
3642
3643
3643
3644
3644
3645
3645
3646
3646
3647
3647
3648
3648
3649
3649
3650
3650
3651
3651
3652
3652
3653
3653
3654
3654
3655
3655
3656
3656
3657
3657
3658
3658
3659
3659
3660
3660
3661
3661
3662
3662
3663
3663
3664
3664
3665
3665
3666
3666
3667
3667
3668
3668
3669
3669
3670
3670
3671
3671
3672
3672
3673
3673
3674
3674
3675
3675
3676
3676
3677
3677
3678
3678
3679
3679
3680
3680
3681
3681
3682
3682
3683
3683
3684
3684
3685
3685
3686
3686
3687
3687
3688
3688
3689
3689
3690
3690
3691
3691
3692
3692
3693
3693
3694
3694
3695
3695
3696
3696
3697
3697
3698
3698
3699
3699
3700
3700
3701
3701
3702
3702
3703
3703
3704
3704
3705
3705
3706
3706
3707
3707
3708
3708
3709
3709
3710
3710
3711
3711
3712
3712
3713
3713
3714
3714
3715
3715
3716
3716
3717
3717
3718
3718
3719
3719
3720
3720
3721
3721
3722
3722
3723
3723
3724
3724
3725
3725
3726
3726
3727
3727
3728
3728
3729
3729
3730
3730
3731
3731
3732
3732
3733
3733
3734
3734
3735
3735
3736
3736
3737
3737
3738
3738
3739
3739
3740
3740
3741
3741
3742
3742
3743
3743
3744
3744
3745
3745
3746
3746
3747
3747
3748
3748
3749
3749
3750
3750
3751
3751
3752
3752
3753
3753
3754
3754
3755
3755
3756
3756
3757
3757
3758
3758
3759
3759
3760
3760
3761
3761
3762
3762
3763
3763
3764
3764
3765
3765
3766
3766
3767
3767
3768
3768
3769
3769
3770
3770
3771
3771
3772
3772
3773
3773
3774
3774
3775
3775
3776
3776
3777
3777
3778
3778
3779
3779
3780
3780
3781
3781
3782
3782
3783
3783
3784
3784
3785
3785
3786
3786
3787
3787
3788
3788
3789
3789
3790
3790
3791
3791
3792
3792
3793
3793
3794
3794
3795
3795
3796
3796
3797
3797
3798
3798
3799
3799
3800
3800
3801
3801
3802
3802
3803
3803
3804
3804
3805
3805
3806
3806
3807
3807
3808
3808
3809
3809
3810
3810
3811
3811
3812
3812
3813
3
```

```

2214 G.4 CODE OF RIEMANNIAN ADAN
2215
2216 from __future__ import annotations
2217
2218 from typing import TYPE_CHECKING
2219
2220 import torch
2221 from geoopt import ManifoldParameter, ManifoldTensor
2222 from geoopt.optim.mixin import OptimMixin
2223
2224 if TYPE_CHECKING:
2225     from beartype.typing import Any, Callable
2226     from jaxtyping import Float
2227
2228 from . import _adan
2229
2230
2231 class RiemannianAdan(OptimMixin, _adan.Adan):
2232     """Riemannian Adan with the same API as :class:adan.Adan.
2233
2234     Attributes:
2235         param_groups: iterable of parameter groups, each containing
2236             parameters to optimize and optimization options
2237         _default_manifold: the default manifold used for optimization if
2238             not specified in parameters
2239
2240     Args:
2241         params: iterable of parameters to optimize or dicts defining
2242             parameter groups
2243         lr: learning rate (default: 1e-3)
2244         betas: coefficients used for computing (default: (0.98, 0.92,
2245             0.99))
2246         eps: term added to the denominator to improve numerical stability
2247             (default: 1e-8)
2248         weight_decay: weight decay (L2 penalty) (default: 0)
2249     """
2250
2251     def step(self, closure: Callable | None = None) -> Float[torch.Tensor
2252         , ""] | None:
2253         """Performs a single optimization step.
2254
2255         Args:
2256             closure: A closure that reevaluates the model and returns the
2257                 loss.
2258
2259         Returns:
2260             The loss value if closure is provided, otherwise None.
2261         """
2262         loss = None
2263         if closure is not None:
2264             loss = closure()
2265
2266         with torch.no_grad():
2267             for group in self.param_groups:
2268                 betas = group["betas"]
2269                 weight_decay = group["weight_decay"]
2270                 eps = group["eps"]
2271                 learning_rate = group["lr"]
2272                 stabilize = False
2273                 for point in group["params"]:
2274                     grad = point.grad
2275                     if grad is None:
2276                         continue
2277                     if isinstance(point, ManifoldParameter |
2278                         ManifoldTensor):

```

```

2268         manifold = point.manifold
2269     else:
2270         manifold = self._default_manifold
2271
2272     if grad.is_sparse:
2273         raise RuntimeError("RiemannianAdan does not
2274                         support sparse gradients")
2275
2276     state = self.state[point]
2277
2278     # State initialization
2279     if len(state) == 0:
2280         state["step"] = 0
2281         # Exponential moving average of gradient values
2282         state["exp_avg"] = torch.zeros_like(point)
2283         # Exponential moving average of squared gradient
2284         # values
2285         state["exp_avg_sq"] = torch.zeros_like(point)
2286         # new param
2287         state["exp_avg_diff"] = torch.zeros_like(point)
2288         # last step grad
2289         state["last_grad"] = torch.zeros_like(point)
2290
2291     state["step"] += 1
2292     # make local variables for easy access
2293     exp_avg = state["exp_avg"]
2294     exp_avg_diff = state["exp_avg_diff"]
2295     exp_avg_sq = state["exp_avg_sq"]
2296     last_grad = state["last_grad"]
2297     # actual step
2298
2299     grad.add_(point, alpha=weight_decay)
2300     grad = manifold.egrad2rgrad(point, grad)
2301     # grad_last_diff
2302     grad_last_diff = grad - last_grad
2303     exp_avg.mul_(betas[0]).add_(grad, alpha=1 - betas[0])
2304     # grad_last_diff
2305     exp_avg_diff.mul_(betas[1]).add_(grad_last_diff,
2306                                 alpha=1 - betas[1])
2307     # z_t
2308     zt = grad_last_diff.mul(betas[1]).add_(grad)
2309     # z_t^2
2310     exp_avg_sq.mul_(betas[2]).add_(manifold.
2311                                 component_inner(point, zt), alpha=1 - betas[2])
2312     bias_correction1 = 1 - betas[0] ** state["step"]
2313     bias_correction2 = 1 - betas[1] ** state["step"]
2314     bias_correction3 = 1 - betas[2] ** state["step"]
2315
2316     denom = exp_avg_sq.div(bias_correction3).sqrt_()
2317
2318     # copy the state, we need it for retraction
2319     # get the direction for ascend
2320     direction = (
2321         (exp_avg.div(bias_correction1)).add_(
2322             exp_avg_diff.div(bias_correction2)), alpha=
2323             betas[1])
2324     ) / denom.add_(eps)
2325
2326     # transport the exponential averaging to the new
2327     # point
2328     new_point, exp_avg_new = manifold.retr_transp(point,
2329                                                   -learning_rate * direction, exp_avg)
2330
2331     last_grad.copy_(manifold.transp(point, new_point,
2332                                     grad))

```

```

2322         # transport v_t
2323         exp_avg_diff.copy_(manifold.transp(point, new_point,
2324                         exp_avg_diff))
2325         exp_avg.copy_(exp_avg_new)
2326         point.copy_(new_point)
2327
2328         if group["stabilize"] is not None and state["step"] % 1000 == 0:
2329             stabilize = True
2330
2331         if stabilize:
2332             self.stabilize_group(group)
2333
2334     return loss
2335
2336     @torch.no_grad() # type: ignore
2337     def stabilize_group(self, group: dict[str, Any]) -> None:
2338         """Stabilizes the parameters in the group by projecting them onto
2339         their respective manifolds.
2340
2341         Args:
2342             group: A dictionary containing the parameters and their
2343                 states.
2344
2345         Returns:
2346             None
2347
2348         """
2349         for p in group["params"]:
2350             if not isinstance(p, ManifoldParameter | ManifoldTensor):
2351                 continue
2352             state = self.state[p]
2353             if not state: # due to None grads
2354                 continue
2355             manifold = p.manifold
2356             exp_avg = state["exp_avg"]
2357             exp_avg_diff = state["exp_avg_diff"]
2358             last_grad = state["last_grad"]
2359             p.copy_(manifold.projx(p))
2360             exp_avg.copy_(manifold.proju(p, exp_avg))
2361             exp_avg_diff.copy_(manifold.proju(p, exp_avg_diff))
2362             last_grad.copy_(manifold.proju(p, last))
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375

```