spheresmooth: An R Package for Penalized Piecewise Geodesic Curve Fitting on a Sphere

by Jae-Hwan Jhong, Seyoung Lee, Ja-Yong Koo, and Kwan-Young Bak

Abstract This paper introduces an R package spheresmooth, which implements a penalized piecewise geodesic curve fitting method on a sphere. Spherical data observed over a continuum arise frequently in various fields including cardiology, computer vision, physiology, and geophysics. We propose an adaptive smoothing method by extending the linear spline approach to spherical data. Penalization based on differences of velocity vectors endows sparsity among control points of the spherical curve, which enables data-adaptive curve fitting. The proposed method is implemented with a Riemannian block coordinate descent algorithm. Illustrations on Triassic and Jurassic polar wander data and tropical cyclone data demonstrate practicality of the proposed method and the associated spheresmooth package.

1 Introduction

This paper aims to introduce an R package spheresmooth, which implements a penalization method for fitting piecewise geodesic curves to spherical data. Spherical data observed over a continuum arise frequently in various fields including cardiology, computer vision, physiology, and geophysics; see, for example, Gould (1969), Jupp and Kent (1987), Kume et al. (2007), Su et al. (2012), Thompson and Clark (1982), and the references therein. We propose an adaptive curve fitting method based on sparsity-inducing penalization, and develop an R package to enable researchers from various fields to easily utilize it for their applications.

Smoothing methods for data observed on a manifold can be broadly categorized into two categories. The methods falling into the first category utilize conventional techniques applied to Euclidean data after mapping the data to the tangent space. Examples include the smoothing spline method based on the rolling and wrapping procedures introduced by Jupp and Kent (1987) and related works employing parallel transport as in Kume et al. (2007) and Kim et al. (2021). The second category involves performing intrinsic optimization on the manifold without explicitly mapping the data to a plane. Approaches based on variational calculus with regularizing constraints fall into this category; for details, refer to Camarinha et al. (1995), Noakes et al. (1989), Samir et al. (2012), Su et al. (2012), Machado et al. (2006). More recently, Bak et al. (2023) introduced an intrinsic curve fitting method based on the generalized Bézier curve and a local penalization scheme on the unit sphere.

The penalized intrinsic Bézier curve fitting method proposed by Bak et al. (2023) has the advantage of adaptively detecting local trends in the data. Furthermore, it takes a form that is easily understandable for researchers in applied fields, akin to an extension of polynomial regression models to the sphere. Spherical spline smoothing methods provide a natural extension of Bézier curve-based approaches. By utilizing spherical splines, one can flexibly model local changes in the data without compromising the overall smoothness. Developing intrinsic spline methods for smoothing spherical data is expected to facilitate the easy understanding and analysis of intricate and dynamically changing patterns of spherical data.

Spherical coordinates and the analysis of spherical data are also highly relevant in astronomy. Many methods and tools for spherical data analysis have been developed and motivated by astronomical observations, such as the Cosmic Microwave Background (CMB) maps produced by the Wilkinson Microwave Anisotropy Probe (WMAP) and Planck satellites. The popular HEALPix (Hierarchical Equal Area isoLatitude Pixelization) scheme

was specifically developed by astronomers for the analysis of CMB data from WMAP (Górski et al., 2005). Such tools have been foundational in handling all-sky observations and have inspired further developments in the field of spherical data analysis. The methods introduced in this paper can be viewed as contributing to this broader context of spherical data applications in fields including astronomy.

In this paper, we consider an intrinsic curve fitting method based on the piecewise geodesic curve and a local penalization scheme on a sphere. The proposed method can be understood as an extension of the linear spline method for spherical data. The performance of the spline-type method is determined by the number and location of the control points. We develop a penalization scheme based on the velocity of the curve to ensure sparsity among the corresponding control points. This allows for the data-adaptive selection of the control points, which guarantees that the curve fits the given spherical data well and identifies the change points on the sphere. The proposed method is implemented with a Riemannian block coordinate descent algorithm.

To the best of our knowledge, no available R packages offer methods for fitting a smooth path to a given set of noisy spherical data observed at known times. We introduce the **spheresmooth** package that implements the proposed piecewise geodesic curve fitting method. This package includes a main function for smoothing spherical data along with several auxiliary functions necessary for data processing. The usages of these functions are explained with examples in detail. Piecewise geodesic curves can be understood as piecewise linear methods on manifolds, making them intuitive and easily applicable for researchers in various fields. Moreover, all processes including the selection of complexity parameters are automatically handled so that users can readily apply the method to data even in exploratory stages. To illustrate the practical utility of the **spheresmooth** package, we provide examples of applying the proposed methodology to Triassic and Jurassic polar wander data for North America (Kent and Irving, 2010) and tropical cyclone data provided by the Regional Specialized Meteorological Center (RSMC) Tokyo Typhoon Center.

Visualization of the generated curve requires loading **sphereplot** (Robotham, 2022), which includes the packages **rgl** (Murdoch and Adler, 2024). These packages are used for generating and manipulating 3D graphics for spherical data. To visualize the fitted curve on a map, we load the packages **rworldmap** (South, 2011) and **ggplot2** (Wickham, 2016). The **sf** package provides a standardized and efficient way to handle spatial data using simple features, making it easy to perform geographic operations and analyses within the R environment.

```
library(spheresmooth)
library(sphereplot)
library(rworldmap)
library(ggplot2)
library(sf)
```

The remainder of this paper is organized as follows. Section 2 collects mathematical preliminaries related to the spherical geometry. Section 3 defines the penalized piecewise curve fitting method and summarizes the implementation scheme based on the Riemannian block coordinate descent algorithm. Section 4 outlines the structure and usage of the **spheresmooth** package, followed by practical application examples in Section 5. Section 6 provides a summary of the results presented in this paper.

2 Preliminaries

This section reviews some basic concepts of Riemannian geometry used in this paper. One may refer to, for example, do Carmo (1976) and do Carmo (1992) for a detailed overview of the relevant mathematical backgrounds.
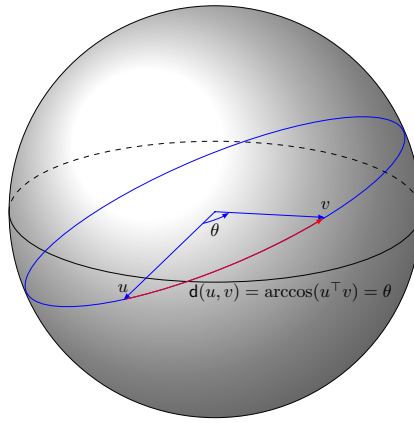
Figure 1: Illustration of geodesic segment between u and v and spherical distance.

Spherical distance and geodesic segment

In this work, the term 'geodesic segment' is equivalent to a 'great circle segment' on a sphere. This distinction is made due to differences in terminology preferences across various academic disciplines.

Let u and v be points on the unit sphere and $\theta, 0 \leq \theta \leq \pi$, be the angle formed by u and v relative to the origin of the sphere. Define the spherical distance between two points u and v as

$$\mathrm{d}(u,v) = \arccos(u^\top v).$$

Observe $\mathrm{d}(u,v) = \theta$ on the unit sphere.

Let S be the two-dimensional unit sphere embedded in $\mathbb{R}^3$. A geodesic segment $\alpha : I = [0,1] \rightarrow \mathsf{S}$ between two points u and v on S is defined as

$$\alpha(t;u,v) = \frac{\sin((1-t)\theta)}{\sin(\theta)}u + \frac{\sin(t\theta)}{\sin(\theta)}v, \quad t \in I$$

where $\theta = \mathrm{d}(u,v)$. Figure 1 illustrates a geodesic segment between u and v on S in the red line. Here, the length of the red line represents the spherical distance between u and v.

Piecewise geodesic curve

The methodology proposed in this paper aims to fit a linear spherical spline or piecewise geodesic curve to spherical data observed over time in a data-adaptive way. To define a piecewise geodesic curve, the time interval needs to be divided into subintervals, and corresponding control points on the sphere should be determined. Given a time interval I, we consider a sequence of knots $\tau_0 < \tau_1 < \tau_2 < \cdots < \tau_J$ that defines subintervals

$$I_0 = [\tau_0, \tau_1), \quad I_1 = [\tau_1, \tau_2), \quad \cdots \quad , \quad I_{J-1} = [\tau_{J-1}, \tau_J).$$

Let $\xi = (\xi_0, \ldots, \xi_J)$ be a set of control points in S corresponding to the knot sequence $\tau_1, \ldots, \tau_J$. The associated piecewise geodesic (linear spherical spline) curve is defined as

$$\gamma(t;\xi_1, \ldots, \xi_J) = \sum_{j=0}^{J-1} \alpha\left(\frac{t-\tau_j}{\tau_{j+1}-\tau_j};\xi_j, \xi_{j+1}\right) \mathrm{ind}(t \in I_j),$$

where $\mathrm{ind}(\cdot)$ denotes the indicator function. The sequence of points on the sphere need not necessarily correspond to different time instances. Rather, they represent a collection of spatial locations that may or may not be observed sequentially in time. This is further discussed in Section 4.
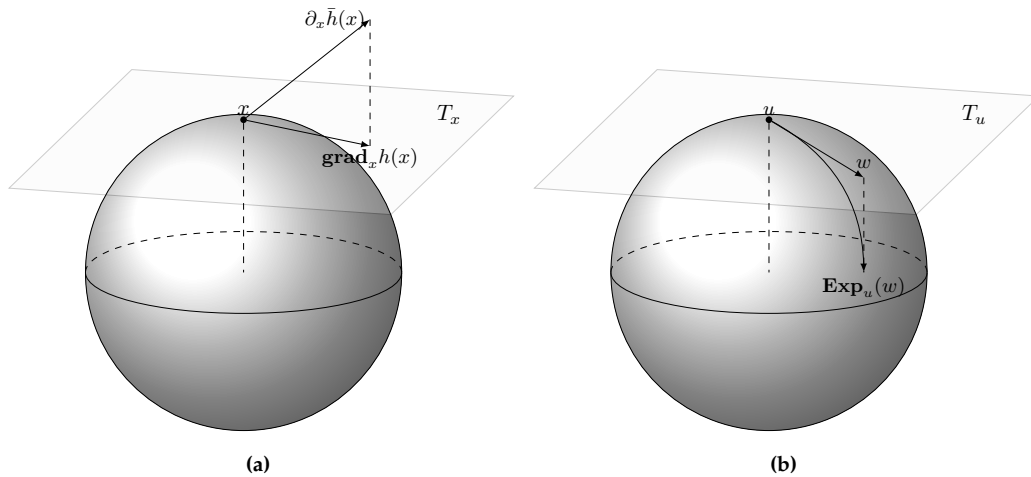
Figure 2: Illustration of (left) Riemannian gradient and (right) exponential map.

Riemannian gradient and exponential map

A blockwise gradient descent algorithm is employed as the algorithm for implementing the spherical smoothing method. The key elements in deriving the algorithm are the Riemannian gradient and the exponential map. The Riemannian gradient is a projection of the Euclidean gradient onto the tangent space. Define the Euclidean gradient as

$$\partial_x \bar{h}(x) = \left[\frac{\partial \bar{h}(x)}{x_j}(x) \right] \quad \text{for} \quad x \in \mathsf{S}, \quad j \leq J$$

where $h : \mathsf{S} \to \mathbb{R}$ and $\bar{h}$ is a differentiable extension of h to $\mathbb{R}^3$. Denote $P_x = I - xx^\top$ as a projection operation onto the tangent space at x. The Riemannian gradient of h with respect to x is defined as

$$\text{grad}_x h(x) = P_x \partial_x \bar{h}(x).$$

The exponential map is a map from a subset of a tangent space $T_u(\mathsf{S})$, $u \in \mathsf{S}$ to S itself. If $w \in T_u(\mathsf{S})$ is a nonzero vector, a point on S corresponding to w is denoted as $\mathsf{Exp}_u(w)$. When $w = 0$, then $\mathsf{Exp}_u(w) = u$. The exponential map is a distance-preserving map in the sense that $|u - w| = d(u, \mathsf{Exp}_u(w))$ for $u, w \in \mathsf{S}$, where $|\cdot|$ is the Euclidean ℓ_2 norm. Figure 2 illustrates the Riemannian gradient and exponential map.

3 Penalized piecewise geodesic curve fitting

Penalized piecewise geodesic curve

Let $\{(t_n, y_n)\}_{n=1}^N$ be a set of data, where $t_n \in I$ are the given time points and y_n are the associated data points on S. Our goal is to find a piecewise geodesic curve that fits the data well. To this end, we adopt the squared distance loss function

$$\ell(\xi) = \sum_{n=1}^N d^2(y_n, \gamma(t_n; \xi)) \quad \text{for} \quad \xi \in \mathsf{S}^{J+1}$$

where S^{J+1} is the $J + 1$ product of S.

The most important aspect of fitting a piecewise curve is selecting the appropriate number of pieces. The number of pieces is determined by the number of knots and the corresponding control points. To avoid the issues of oversmoothing and undersmoothing, we propose a penalty function that allows us to select the number of control points in a data-adaptive way. This enables the fitted spherical curve to capture local trends in the data
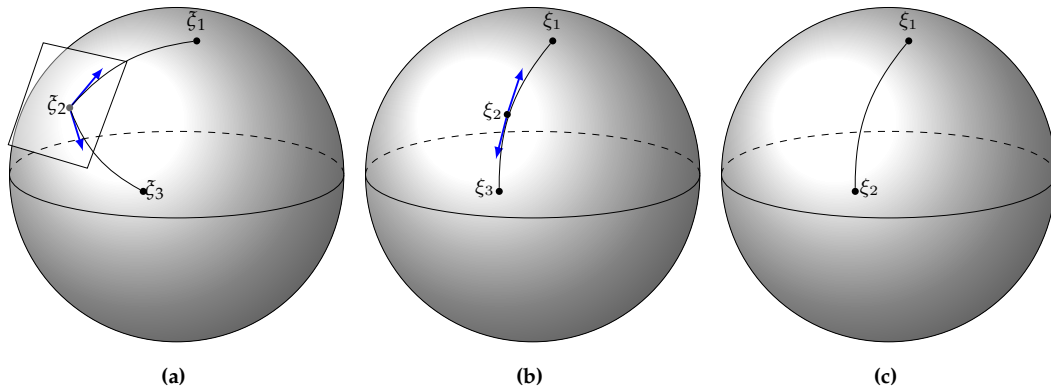
Figure 3: Example of elimination process of the piecewise geodesic curve with three control points. Blue arrow vector indicates tangent vector of ξ_2 in consecutive geodesic segments. The left plot shows the spline curve with small value of complexity parameter. As the complexity parameter λ increases, ξ_2 moves into the geodesic between ξ_1 and ξ_3 and the difference between tangent vectors becomes zero. The middle plot shows that step. When the difference between tangent vectors is zero, control point is removed. The right plot shows the resulting curve after control point is removed in the middle plot.

without compromising the overall smoothness. We introduce a smoothness penalty function

$$\mathsf{p}(\xi) = \sum_{j=1}^{J-1} |D_j(\xi)| \quad \text{for} \quad D_j(\xi) = \frac{1}{\tau_{j+1} - \tau_j} \dot{\alpha}(0; \xi_j, \xi_{j+1}) - \frac{1}{\tau_j - \tau_{j-1}} \dot{\alpha}(1; \xi_{j-1}, \xi_j),$$

which regularizes the number of geodesic segments needed to fit the data. Here, $\cdot$ above a curve denotes its derivative with respect to t and $|\cdot|$ denotes the Euclidean ℓ_2-norm of a vector.

The velocity-based penalty function induces sparsity on the control points. As λ increases, the difference $|D_j(\xi)|$ in the penalty term tends to zero. Suppose that $|D_j(\xi)| = 0$ for some $1 \leq j \leq J - 1$ and λ. This implies that the geodesic segments $\alpha(\cdot; \xi_{j-1}, \xi_j)$ and $\alpha(\cdot; \xi_j, \xi_{j+1})$ are on the same great circle. Then, we eliminate the inactive control point ξ_j and replace the two geodesic segments with $\alpha(\cdot; \xi_{j-1}, \xi_{j+1})$. The elimination process reduces the number of the control points by one. Figure 3 illustrates the elimination process that starts with two geodesic segments.

The objective function to minimize is given by

$$\ell^\lambda(\xi) = \ell(\xi) + \lambda \mathsf{p}(\xi) \quad \text{for} \quad \xi \in \mathsf{S}^{J+1},$$

where $\lambda > 0$ is the complexity parameter. For a fixed λ, the penalized piecewise geodesic (linear spherical spline) curve is defined as

$$\gamma_J(\cdot; \hat{\xi}_\lambda),$$

where

$$\hat{\xi}_\lambda = \underset{\xi \in \mathsf{S}^J}{\operatorname{argmin}} \ell^\lambda(\xi).$$

Block coordinate descent algorithm

To obtain $\hat{\xi}_\lambda$, we adopt a block coordinate descent method, where each block consists of the three-dimensional coordinates of each control point. First, we set the initial values of ξ by $J + 1$ number of data points y_n associated with (j/J)th quantiles of the time points t_n for $j = 0, 1, \ldots, J$. Denote the initial values by $\xi^{(0)} = (\xi_0^{(0)}, \ldots, \xi_J^{(0)})$. Then, we obtain the Riemannian gradient of ℓ^λ with respect to ξ_j and apply the gradient descent algorithm to

update the control points. Since the updated points belong to a tangent space of $\mathbb{S}$, we apply the exponential function to map the points into $\mathbb{S}$.

Let $\xi_j^{(l)}$ denote the current values of the control point ξ_j in the l-th iteration for $j = 0, \ldots, J$. For $l = 1, 2, \cdots$, the update formula has the following form:

$$\xi_0^{(l)} \leftarrow \mathsf{Exp}_{\xi_0^{(l-1)}} \left[-\rho_{l,0} \mathsf{grad}_{\xi_0} \ell^\lambda \left(\xi_0^{(l-1)}, \xi_1^{(l-1)}, \xi_2^{(l-1)}, \ldots, \xi_J^{(l-1)} \right) \right]$$

$$\xi_1^{(l)} \leftarrow \mathsf{Exp}_{\xi_1^{(l-1)}} \left[-\rho_{l,1} \mathsf{grad}_{\xi_1} \ell^\lambda \left(\xi_0^{(l)}, \xi_1^{(l-1)}, \xi_2^{(l-1)}, \ldots, \xi_J^{(l-1)} \right) \right]$$

$$\xi_2^{(l)} \leftarrow \mathsf{Exp}_{\xi_2^{(l-1)}} \left[-\rho_{l,2} \mathsf{grad}_{\xi_2} \ell^\lambda \left(\xi_0^{(l)}, \xi_1^{(l)}, \xi_2^{(l-1)}, \ldots, \xi_J^{(l-1)} \right) \right]$$

$$\cdots$$

$$\xi_J^{(l)} \leftarrow \mathsf{Exp}_{\xi_J^{(l-1)}} \left[-\rho_{l,J} \mathsf{grad}_{\xi_J} \ell^\lambda \left(\xi_0^{(l)}, \xi_1^{(l)}, \xi_2^{(l)}, \ldots, \xi_J^{(l-1)} \right) \right],$$

where $\rho_{l,j}$ denotes a specified step size for the update of ξ_j in the lth cycle. The specific form of the update formula is derived below.

As a line search strategy, we adopt the step-halving method to determine the step size $\rho_{l,j}$ to guarantee the stable convergence of the proposed algorithm. When the objective function does not decrease with the current step size, the step size is halved. The step-halving procedure is repeated until the objective function decreases with the reduced step size. The iteration stops when the difference between the current and the updated values of the objective function ℓ^λ is less than $\varepsilon = 10^{-5}$.

Derivation of the coordinate-wise update formula

The derivation of the update formula is based on the result in Bak et al. (2023). For notational convenience, denote

$$C(z) = \cos(z), \quad S(z) = \sin(z) \quad \text{and} \quad R_s(z) = \frac{S(sz)}{S(z)}$$

for $s \in I$ and $z \in \mathbb{R}$. With this notation, the parametrized geodesic segment between control points u and v is expressed as

$$\alpha(t; u, v) = R_{1-t}(\theta)u + R_t(\theta)v, \quad t \in I,$$

with $\theta = \mathrm{d}(u, v)$. We first obtain the Euclidean derivative of the geodesic segment with respect to each endpoint.

Theorem 3.1 in Bak et al. (2023) implies that the derivatives of α with respect to the control points are given by

$$\left(\frac{d}{du} \gamma(t; u, v) \right)^\top = R_{1-t}(\theta)I + Q_{1-t}(\theta)vu^\top + Q_t(\theta)vv^\top,$$

and

$$\left(\frac{d}{dv} \gamma(t; u, v) \right)^\top = R_t(\theta)I + Q_{1-t}(\theta)uu^\top + Q_t(\theta)uv^\top,$$

where

$$Q_s(z) \triangleq \frac{S_s(z)C(z) - sC_s(z)S(z)}{S^3(z)} \quad \text{for} \quad s \in I \quad \text{and} \quad z \in \mathbb{R}.$$

Consider now a piecewise geodesic curve $\gamma(\cdot, \xi)$ defined by a set of control points

$\xi = (\xi_1, \ldots, \xi_J)$. Since the spline curve consists of connected geodesic segments, we have

$$\frac{d}{d\xi_j}\gamma(t;\xi) = \begin{cases} \frac{d}{d\xi_0}\alpha\left(\frac{t-\tau_0}{\tau_1-\tau_0};\xi_0,\xi_1\right) & \text{for} \quad j=0 \\ \frac{d}{d\xi_J}\alpha\left(\frac{t-\tau_{J-1}}{\tau_J-\tau_{J-1}};\xi_{J-1},\xi_J\right) & \text{for} \quad j=J \\ \frac{d}{d\xi_j}\alpha\left(\frac{t-\tau_{j-1}}{\tau_j-\tau_{j-1}};\xi_{j-1},\xi_j\right) + \frac{d}{d\xi_j}\alpha\left(\frac{t-\tau_{j-1}}{\tau_j-\tau_{j-1}};\xi_j,\xi_{j+1}\right) & \text{o.w.} \end{cases}$$

Combining the results, we are able to obtain the derivatives of $\gamma(\cdot,\xi)$ with respect to each control point. This enables us to compute the Riemannian gradient of the squared distance loss:

$$\text{grad}_{\xi_k}\ell(\xi) = -\sum_{n=1}^{N} \frac{\phi_n}{S(\phi_n)} P_{\xi_k}\left(\frac{d\gamma(t_n;\xi)}{d\xi_k}\right)^{\top} y_n,$$

where $\phi_n \triangleq \mathsf{d}(y_n, \gamma(t_n;\xi))$.

In addition to the result above, the Riemannian gradient of the velocity difference penalty is needed to derive a coordinate-wise update formula. Following Theorem 3.4 in Bak et al. (2023), for a fixed $1 \leq j \leq J-1$, we have

$$\text{grad}_{\xi_{j-1}}|D_j(\xi)| = P_{\xi_{j-1}} \frac{1}{\tau_j - \tau_{j-1}} \left[\frac{C(\theta_{j-1})\theta_{j-1} - S(\theta_{j-1})}{S^3(\theta_{j-1})}\xi_j\xi_{j-1}^{\top} + \frac{\theta_{j-1}}{S(\theta_{j-1})}I\right]\frac{D_j(\xi)}{|D_j(\xi)|},$$

$$\text{grad}_{\xi_{j+1}}|D_j(\xi)| = \frac{1}{\tau_{j+1} - \tau_j} P_{\xi_{j+1}} \left[\frac{C(\theta_j)\theta_j - S(\theta_j)}{S^3(\theta_j)}\xi_j\xi_{j+1}^{\top} + \frac{\theta_j}{S(\theta_j)}I\right]\frac{D_j(\xi)}{|D_j(\xi)|},$$

and

$$\text{grad}_{\xi_j}|D_j(\xi)| = \frac{1}{\tau_{j+1} - \tau_j} P_{\xi_j} \left[\frac{C(\theta_j)\theta_j - S(\theta_j)}{S^3(\theta_j)}\xi_{j+1}\xi_{j+1}^{\top} - \frac{\theta_j C(\theta_j)}{S(\theta_j)}I\right]\frac{D_j(\xi)}{|D_j(\xi)|}$$
$$+ \frac{1}{\tau_j - \tau_{j-1}} P_{\xi_j} \left[\frac{C(\theta_{j-1})\theta_{j-1} - S(\theta_{j-1})}{S^3(\theta_{j-1})}\xi_{j-1}\xi_{j-1}^{\top} - \frac{\theta_{j-1}C(\theta_{j-1})}{S(\theta_{j-1})}I\right]\frac{D_j(\xi)}{|D_j(\xi)|}.$$

Combining the derivatives of the loss function and the penalty function yields the required Riemannian gradient of the objective function with respect to the control points.

Selection of optimal tuning parameter

The procedure for finding the optimal complexity parameter is as follows. The piecewise geodesic curves that minimize the objective function are computed for an increasing sequence of the complexity parameters $\lambda_1 < \cdots < \lambda_K$, where λ_K is sufficiently large. As the complexity parameter increases, we eliminate the control points that are no longer active.

In general, the choice of the control points plays a crucial role in the spline-type curve fitting problem. We begin by setting λ_K to a sufficiently large value and λ_1 to a value close to zero, ensuring the flexibility of the model. As the complexity parameter increases, the elimination process leads to data-adaptive control point selection. Then, we determine the optimal complexity parameter and the corresponding optimal curve whose control points are adaptively determined by the given set of data.

To choose an optimal complexity parameter, we use a modified version of the Bayes information criterion (BIC); see Schwarz (1978). The BIC for a sequence of the complexity parameters is defined as

$$\text{BIC}_k = N\log\ell(\hat{\xi}_{\lambda_k}) + 3J_k\log N \quad \text{for} \quad k = 1, \ldots, K$$

where J_k denotes the number of control points for λ_k. The optimal value for λ is chosen as $\lambda_{\hat{k}}$, where

$$\hat{k} = \underset{1 \leq k \leq K}{\text{argmin}}\text{BIC}_k.$$

4   The `spheresmooth` package: structure and functionality

Overall structure and available functions

The **spheresmooth** package is a versatile toolkit designed for performing mathematical and geometric operations on the sphere, with a primary focus on smoothing spherical data observed at known time points. It finds applications across various fields such as geography, geometry, physics, and computer graphics. Table **??** contains a compact summary of the available functions. When the name of a function is in uppercase, it means it computes the output for multiple points simultaneously except for `exp_map()` function. The package includes a corresponding internal function for a single point calculation. For example, the `geodesic()` function computes the coordinates of a geodesic curve at multiple time points, which is the result of multiple internal executions of the `geodesic_lower()` function for a single time point. These functions take arguments in the form of numeric matrices.

Spherical data may be given in the form of spherical coordinates or Cartesian coordinates depending on the case. We may need to switch between the two coordinate systems as necessary. The `cartesian_to_spherical()` function converts Cartesian coordinates to spherical coordinates, which is crucial for representing points on a spherical surface accurately. The `spherical_to_cartesian()` function performs the opposite operation. The input data for the main function must be in the form of Cartesian coordinates. Therefore, if the data is given in spherical coordinates, one can use the `spherical_to_cartesian()` function to perform coordinate transformation before applying the proposed method.

The proposed method is based on the calculation of piecewise geodesic paths. To this end, we need functions that compute the geodesic segments and partition the given time interval. The `geodesic()` function computes the value of the geodesic curve connecting two points p and q on the sphere at specified time points. On the other hand, the `knots_quantile()` function generates a sequence of knots for a given set of time points based on the quantile values, thereby offering support for interpolating and approximating time-series data using spherical spline curves. The `piecewise_geodesic()` function is one of the core functions in the package. It computes the coordinates of the piecewise geodesic curve at the input time point for the given knots in the time interval and the corresponding control points on the sphere.

To evaluate the goodness-of-fit of the fitted curve, a distance-based loss function is needed. The `spherical_dist()` function calculates the geodesic distance between two input points. Based on this function, the `calculate_loss()` calculates the loss function based on the squared spherical distances between observed values and predicted values on the curve. Finally, the main function `penalized_linear_spherical_spline()` fits the penalized piecewise geodesic curve to the given data. In summary, the **spheresmooth** package provides core functions for smoothing spherical data, along with several useful functions for handling spherical data.

Basic functions for handling spherical data

Before delving into the main functions of the package, we introduce several useful functions for handling spherical data within the package. These functions are essential for performing core functionalities of the package but can also be applied to other spherical data analysis tasks. Only a subset of the functions listed in Table **??** is illustrated based on their frequency of use.

The `spherical_dist()` function is used to compute the geodesic distance between two points on the unit sphere. For example, the distance along the geodesic segment between $(1, 0, 0)$ and $(0, 1, 0)$ is one-fourth of the circumference of a great circle containing two points, which is 2π, and it is calculated as follows in code:

```
x <- c(1, 0, 0)
y <- c(0, 1, 0)
```

Table 1: Summary of the functions in the spheresmooth package.

Function	Description
calculate_loss	Calculates the loss function based on the squared spherical distances between observed values and predicted values on the curve.
cartesian_to_spherical	Converts Cartesian coordinates to spherical coordinates.
cross	Computes the cross product of two input vectors
dot	Computes the dot product of two input vectors u and v.
edp	Computes the equal-distance projection of a point p onto the xy plane.
exp_map	Computes the exponential map on the unit sphere given a base point x and a vector v.
geodesic	Computes the value of the geodesic curve connecting two points p and q on the unit sphere at specified time points.
knots_quantile	Generates a sequence of knots for a given set of time points based on the quantiles.
norm2	Computes the L2 norm (Euclidean norm) of the input vector.
normalize	Normalizes the rows of the input matrix x by dividing each row by its L2 norm (Euclidean norm).
penalized_linear_spherical_spline	Fits a penalized linear spherical spline (piecewise geodesic) curve to the given data.
piecewise_geodesic	Computes a piecewise geodesic path between control points.
spherical_dist	Calculates the spherical distance between two vectors.
spherical_to_cartesian	converts spherical coordinates (θ, ϕ) to Cartesian coordinates.

```
spherical_dist(x, y)
```

```
#> [1] 1.570796
```

Consider another example involving the point $(1/\sqrt{3}, 1/\sqrt{3}, 1/\sqrt{3})$. The antipodal point to this given point is $(-1/\sqrt{3}, -1/\sqrt{3}, -1/\sqrt{3})$, which traverses half the circumference of the associated great circle, resulting in a length of π. This can be checked with the following code.

```
x <- c(1/sqrt(3), 1/sqrt(3), 1/sqrt(3))
y <- c(-1/sqrt(3), -1/sqrt(3), -1/sqrt(3))
spherical_dist(x, y)
```

```
#> [1] 3.141593
```

Another useful function is the `cross` function with `normalize = TRUE` argument used to obtain a unit vector orthogonal to a pair of given vectors. For example, we can use this function to obtain a unit vector $(0, 0, 1)$ orthogonal to the plane spanned by $(1, 0, 0)$ and $(0, 1, 0)$.

```
x <- c(1, 0, 0)
y <- c(0, 1, 0)
cross(x, y, normalize = TRUE)
```

```
#> [1] 0 0 1
```

Besides standard utility functions, there are two crucial auxiliary functions in spherical data analysis: `cartesian_to_spherical()` and `spherical_to_cartesian()`. These transformations have already been discussed earlier in this section. For basic spherical operations, it is utilized as described previously. When data is given in Cartesian coordinates, the `cartesian_to_spherical()` function computes spherical coordinates, enabling visualization and other analyses. On the other hand, when the data is given in spherical coordinates, the main function of the **spheresmooth** package, `penalized_linear_spherical_spline()`, cannot be directly applied. In such cases, the `spherical_to_cartesian()` function allows the transformation of data into the required format. The following code demonstrates examples implementing these transformations.

```
# example: cartesian_to_spherical
cartesian_points <- matrix(c(1, 0, 0, 0, 1, 0, 0, 0, 1), ncol = 3, byrow = TRUE)
cartesian_to_spherical(cartesian_points)
```

```
#>          theta      phi
#> [1,] 1.570796 0.000000
#> [2,] 1.570796 1.570796
#> [3,] 0.000000 0.000000
```

```
# example: spherical_to_cartesian
theta_phi <- matrix(c(pi/4, pi/3, pi/6, pi/4), ncol = 2, byrow = TRUE)
spherical_to_cartesian(theta_phi)
```

```
#>           [,1]      [,2]      [,3]
#> [1,] 0.3535534 0.6123724 0.7071068
#> [2,] 0.3535534 0.3535534 0.8660254
```

The following code demonstrates that these two functions perform inverse operations.

```
theta_phi <- matrix(c(pi/4, pi/3, pi/6, pi/4), ncol = 2, byrow = TRUE)
theta_phi

#>           [,1]      [,2]
#> [1,] 0.7853982 1.0471976
#> [2,] 0.5235988 0.7853982


cartesian_to_spherical(spherical_to_cartesian(theta_phi))

#>          theta       phi
#> [1,] 0.7853982 1.0471976
#> [2,] 0.5235988 0.7853982
```

Piecewise geodesic curve

As illustrated in Section 2, we need to provide a sequence of knots and associated control points to define a piecewise geodesic curve. Although we introduce our methodology targeting data observed over time, it is important to note that the argument of a parametrized curve, denoted as t, does not necessarily imply physical time. This methodology can be applied to the smoothing problem of any directional data evolving over a continuous domain. The only requirements for computing a piecewise geodesic curve are the knots and their corresponding control points. The function that returns the coordinates of the curve at a given t point is the `piecewise_geodesic()` function. Table 2 summarizes the arguments of the function.

Table 2: Summary of the arguments of the `piecewise_geodesic()` function.

Argument	Description
t	A numeric vector representing the time or location.
control_points	A matrix of control points where each row represents a control point.
knots	A numeric vector of knot values.

Here, the `control_points` is a matrix where each row represents a point in Cartesian coordinates that determines the geodesic segments. Since a pair of points defines a geodesic segment, this function can be used to compute the geodesic curve between two points if the `knots` is given as two endpoints. The `knots` is a vector of points separating the curve in the time domain. The `piecewise_geodesic()` function performs the following steps. First, it initializes an empty matrix to store the generated points on the curve. Second, it iterates over the separated geodesic segments to find the segment to which the `t` values belong. Then, it internally calls the `geodesic()` function for the corresponding segment to compute the coordinates of those points on the curve and appends the generated points to the initialized matrix. Finally, it returns the matrix containing all the points corresponding to the given `t` values.

In the Figure 4, the left plot depicts a piecewise geodesic curve determined by the control points $(1, 0, 0)$, $(1/\sqrt{2}, 1/\sqrt{2}, 0)$, $(-1/\sqrt{3}, 1/\sqrt{3}, 1/\sqrt{3})$, and $(0, 0, 1)$. The function `piecewise_geodesic()` is used to compute the coordinates of the curve at given time points. The `piecewise_geodesic` function presented here is roughly equivalent to the `rgl.sphline()` function from the **sphereplot** package, which also generates great circle line segments. However, `piecewise_geodesic` provides additional flexibility for handling complex geometries and segmentations, making it a suitable choice for a broader range of applications. In the following code, we create a variable called `control_points`, which is a matrix containing four control points. The control points are used to determine the curve on the sphere while the `knots` indicate the points where transitions occur in the time domain. To obtain a smooth curve for visualization, we generate `t_example` by slicing the interval
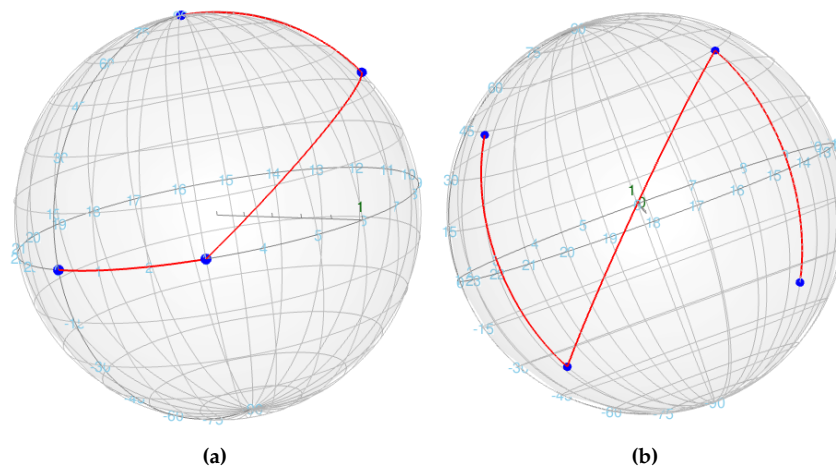
(a)   (b)

Figure 4: Plot of (left) piecewise geodesic curve with the control points at $(1, 0, 0)$, $(1/\sqrt{2}, 1/\sqrt{2}, 0)$, $(-1/\sqrt{3}, 1/\sqrt{3}, 1/\sqrt{3})$, and $(0, 0, 1)$ and (right) piecewise geodesic curve with the control points at $(1/\sqrt{3}, 1/\sqrt{3}, 1/\sqrt{3})$, $(1/\sqrt{3}, 1/\sqrt{3}, -1/\sqrt{3})$, $(-1/\sqrt{3}, 1/\sqrt{3}, 1/\sqrt{3})$, and $(-1/\sqrt{3}, 1/\sqrt{3}, -1/\sqrt{3})$.

from 0 to 4 into increments of 0.01, and evaluate the coordinates of the curve at those points using the `piecewise_geodesic()` function. The following code yields the left plot of Figure 4.

```
control_points <- matrix(c(1, 0, 0,                       # Control point 1
                     1/sqrt(2), 1/sqrt(2), 0,             # Control point 2
                     -1/sqrt(3), 1/sqrt(3), 1/sqrt(3),    # Control point 3
                     0, 0, 1),                            # Control point 4
                     nrow = 4, byrow = TRUE)
knots <- c(1, 2, 3, 3.5)  # Knots indicating transitions
# Example of generating piecewise geodesic curve
t_example <- seq(0, 4, by = 0.01)
gamma_example <- piecewise_geodesic(t_example, control_points, knots)
# Plotting the piecewise geodesic curve
rgl.sphgrid(deggap = 15, col.long = "skyblue", col.lat = "skyblue")
spheres3d(x = 0, y = 0, z = 0, radius = 1, col = "grey", alpha = 0.05)
pch3d(control_points, col = "blue", cex = 0.2, pch = 19)
lines3d(gamma_example, col = "red", lty = 1, lwd = 2)
```

The right plot of Figure 4 is an example of the curve with control points located at $(1/\sqrt{3}, 1/\sqrt{3}, 1/\sqrt{3})$, $(1/\sqrt{3}, 1/\sqrt{3}, -1/\sqrt{3})$, $(-1/\sqrt{3}, 1/\sqrt{3}, 1/\sqrt{3})$, and $(-1/\sqrt{3}, 1/\sqrt{3}, -1/\sqrt{3})$. Below is the code that computes the curve and draws the corresponding plot.

```
control_points <- matrix(c(1/sqrt(3), 1/sqrt(3), 1/sqrt(3),     # Control point 1
                     1/sqrt(3), 1/sqrt(3), -1/sqrt(3),     # Control point 2
                     -1/sqrt(3), 1/sqrt(3), 1/sqrt(3),     # Control point 3
                     -1/sqrt(3), 1/sqrt(3), -1/sqrt(3)),   # Control point 4
                     nrow = 4, byrow = TRUE)
knots <- c(1, 2, 3, 3.5)  # Knots indicating transitions
# Example of generating piecewise geodesic curve
t_example <- seq(0, 4, by = 0.01)
gamma_example <- piecewise_geodesic(t_example, control_points, knots)
# Plotting the piecewise geodesic curve
rgl.sphgrid(deggap = 15, col.long = "skyblue", col.lat = "skyblue")
spheres3d(x = 0, y = 0, z = 0, radius = 1, col = "grey", alpha = 0.05)
pch3d(control_points, col = "blue", cex = 0.2, pch = 19)
lines3d(gamma_example, col = "red", lty = 1, lwd = 2)
```

Penalized piecewise geodesic curve fitting

The main function `penalized_linear_spherical_spline()` fits a penalized piecewise geodesic curve to the given spherical data as illustrated in Section 3. As implied by the name of the function, this approach can also be understood as using a penalized linear spline method on the sphere. The crucial aspect here is the selection of knots that capture the points of change within the given time domain. The strategy proposed in this paper involves generating a sufficient number of initial knots using the `knots_quantile()` function and then adaptively selecting knots and control points based on penalization. The initial control points corresponding to the generated knots are determined by the corresponding data points y_i on the given spherical surface.

After ensuring the flexibility of the model by setting the number of initial knots and control points sufficiently large, piecewise geodesic curves are computed for an increasing sequence of the complexity parameters $\lambda_1 < \cdots < \lambda_{\max}$. As the complexity parameter increases, we prune or delete the knots that are no longer active. If λ is small, then so is the amount of penalization, resulting in a wiggly fit. On the other hand, the fitted curve gets smoother as λ gets larger. If one wishes to consider as many possibilities as possible to find the optimal curve, it is possible to generate complexity parameters densely starting from very small values, and ensuring that $\lambda_{\max}$ is sufficiently large to yield a least squares geodesic curve. Among the fitted curves, we can select the optimal curve based on the BIC, thus obtaining a data-adaptive piecewise geodesic curve.

The `penalized_linear_spherical_spline()` function is designed to automatically perform all these steps when initial knots and initial control points are not specified, aiming to enhance convenience for researchers. If there are valid numbers of knots determined through exploratory data analysis and preliminary research, along with coordinates for initial control points, they can be used as arguments. Table 3 summarizes the arguments of the function.

Table 3: Summary of the arguments of the `penalized_linear_spherical_spline()` function.

Argument	Description
t	A numeric vector representing the time or location.
y	A matrix where each row represents a data point.
initial_control_points	An optional matrix specifying initial control points. Default is NULL.
dimension	An integer specifying the dimension of the spline.
initial_knots	An optional numeric vector specifying initial knots. Default is NULL.
lambdas	A numeric vector specifying the penalization parameters.
step_size	A numeric value specifying the step size for optimization. Default is 0.01.
maxiter	An integer specifying the maximum number of iterations. Default is 1000.
epsilon_iter	A numeric value specifying the convergence criterion for iterations. Default is 1e-05.
jump_eps	A numeric value specifying the threshold for pruning control points based on jump size. Default is 1e-02.
verbose	A logical value indicating whether to print progress information. Default is FALSE.

The function returns a list containing the fitted result for each complexity parameter, the dimension and BIC values associated with the complexity parameters required for model selection. The BIC values are stored in the last element of the list. It is possible to directly examine the BIC values and their corresponding fitted curves. Unless there is a specific reason, it is recommended to consider the λ that attains the minimum BIC value as the

optimal complexity parameter and select the corresponding fitted curve as the final fitted model. The usage of the function and the model selection process are illustrated in Section 5.

5 Applications

APW Data

The data considered in this section is the polar wander dataset presented in Kent and Irving (2010). They argued that previous estimates of Triassic and Jurassic paleolatitudes for North America tend to be biased because of inclination error in sedimentary rocks. They constructed a new composite APW path for Triassic through Paleogene based on igneous rocks. The 17 Triassic/Jurassic cratonic poles from other major cratons are rotated into North American coordinates and combined with the 14 observations from North America. We apply the proposed method to these 31 observations ranging in age from 243 to 144 Ma (millions of years ago), which covers the late Triassic and Jurassic periods.

The APW dataset is included in the **spheresmooth**. The data consists of the time, longitude, and latitude values for 31 observations. It can be loaded as follows.

```
dim(apw_spherical)
```

```
#> [1] 31  3
```

The APW data is represented in spherical coordinates. To apply the main function, it is necessary to convert the given data into Cartesian coordinates. We apply the `spherical_to_cartesian()` function to the second and third columns of the datasets, since the first column represents the time points.

```
apw_cartesian = spherical_to_cartesian(apw_spherical[, 2:3])
```

To ensure sufficient flexibility of the model, we set the initial dimension to 15, where the dimension refers to the number of knots or the corresponding number of control points. The knot sequence corresponding to the specified dimension is created using the knots_quantile() function. The grid for tuning the complexity parameter is determined as a sequence of 40 values ranging from 10^{-7} to 1 on a logarithmic scale. We compute the fitted curves using the `penalized_linear_spherical_spline()` function.

```
t = apw_spherical[, 1]
dimension = 15
initial_knots = knots_quantile(t, dimension = dimension)
lambda_seq = exp(seq(log(1e-07), log(1), length = 40))

fit = penalized_linear_spherical_spline(t = t, y = apw_cartesian,
                                        dimension = dimension,
                                        initial_knots = initial_knots,
                                        lambdas = lambda_seq)
```

As explained in the previous section, the returned result is a list containing the information about 40 fitted curves and the dimension and BIC values for model selection.

```
class(fit)
```

```
#> [1] "list"
```

```
length(fit)
```

```
#> [1] 42
```

```
fit$dimension_list

#>  [1] 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15
#> [26] 15 13 13 10 10  9  9  8  8  8  8  8  8  8  8

fit$bic_list

#>  [1] 95.27152 95.22637 95.19061 95.12282 95.08984 95.05878 95.02849 95.00298
#>  [9] 94.97865 94.96560 94.95472 80.58414 80.58430 80.58470 80.58565 80.58750
#> [17] 80.59240 80.60174 80.62544 80.68245 80.77479 80.96411 81.44382 82.18175
#> [25] 83.45483 85.39508 66.62308 67.99973 38.50011 39.57477 29.76234 30.82627
#> [33] 21.74930 32.81671 37.56793 46.46920 47.62672 51.38727 54.17926 54.60989
```

The best model according to the BIC can be selected as follows.

```
# choose a curve that minimizes the BIC
best_index = which.min(fit$bic_list)
best_index

#> [1] 33

# obtained control points for the piecewise geodesic curve
fit[[best_index]]$control_points

#>                [,1]        [,2]       [,3]
#> [1,] -0.48039447 -0.2266981 0.8472480
#> [2,] -0.11927705  0.1300343 0.9843089
#> [3,] -0.06869155  0.1967739 0.9780396
#> [4,]  0.01640654  0.2026538 0.9791130
#> [5,]  0.03231127  0.3512434 0.9357265
#> [6,] -0.03372888  0.5085848 0.8603510
#> [7,] -0.08310953  0.5246722 0.8472378
#> [8,] -0.45295783  0.3987197 0.7974032
```

The optimal complexity parameter is determined as

```
lambda_seq[best_index]

#> [1] 0.0554102
```

and the corresponding dimension is 8. Since we started with the initial dimension 15, the result illustrates the sparsity-inducing property of the proposed method.

The code begins by retrieving a world map using the `getMap()` function, which is then converted to an `sf` object (`worldMap_sf`) using `st_as_sf()` for easier handling in spatial analyses. Control points are transformed from Cartesian coordinates to spherical coordinates using the `cartesian_to_spherical` function, with the resulting coordinates converted to degrees (latitude and longitude) to create `cp_long_lat_df`. The latitude values are adjusted to range from 0 to 90 degrees instead of being measured from the center. The `apw_spherical` data is also converted into a data frame (`apw_spherical_df`), where the latitude and longitude values are transformed from radians to degrees, making them suitable for mapping. Then, a geodesic curve is fitted using the `piecewise_geodesic` function, generating a fitted curve (`fitted_geodesic_curve`). This fitted curve is converted from Cartesian to spherical coordinates (`fitted_cs`), and then transformed into a data frame (`fitted_cs_long_lat_df`) with latitude and longitude values in degrees. Finally, three data frames (`apw_spherical_df`, `cp_long_lat_df`, `fitted_cs_long_lat_df`) are converted to `sf` objects for further geospatial analysis, specifying the appropriate coordinate reference system (EPSG:4326) to facilitate geographic visualizations and operations.

```
worldMap = getMap()
worldMap_sf = st_as_sf(worldMap)

cp_best = cartesian_to_spherical(fit[[best_index]]$control_points)
cp_long_lat = cp_best * 180 / pi
cp_long_lat_df = data.frame(latitude = 90-cp_long_lat[, 1],
                            longitude = cp_long_lat[,2])

apw_spherical_df = data.frame(apw_spherical)
apw_spherical_df$latitude = 90 - apw_spherical_df$latitude * 180 / pi
apw_spherical_df$longitude = apw_spherical_df$longitude * 180 / pi

fitted_geodesic_curve = piecewise_geodesic(seq(0, 1, length = 2000),
                                            fit[[best_index]]$control_points,
                                            fit[[best_index]]$knots)
fitted_cs = cartesian_to_spherical(fitted_geodesic_curve)
fitted_cs_long_lat = fitted_cs * 180 / pi
fitted_cs_long_lat_df = data.frame(latitude = 90 - fitted_cs_long_lat[, 1],
                            longitude = fitted_cs_long_lat[, 2])

apw_spherical_df_sf = st_as_sf(apw_spherical_df,
                            coords = c("longitude", "latitude"), crs = 4326)
cp_long_lat_df_sf = st_as_sf(cp_long_lat_df,
                            coords = c("longitude", "latitude"), crs = 4326)
fitted_cs_long_lat_df_sf = st_as_sf(fitted_cs_long_lat_df,
                            coords = c("longitude", "latitude"), crs = 4326)
```

The optimal fitted curve is visualized in the left plot of Figure 5 using the following code. It presents the obtained APW path (red line) and the associated control points (blue rhombus-shaped points) with the observations in the geographic coordinates. The visualization was generated using the coord_sf() function with the +proj=ortho option, providing an orthographic projection of the spherical data. The fitted curve shows that the obtained APW path has a clockwise rotational trend.

```
worldmap = ggplot() +
  geom_sf(data = worldMap_sf, color = "grey", fill = "antiquewhite") +
  geom_sf(data = apw_spherical_df_sf, size = 0.8) +
  geom_sf(data = cp_long_lat_df_sf, color = "blue", shape = 23, size = 4) +
  geom_sf(data = fitted_cs_long_lat_df_sf, color = "red", size = 0.5) +
  xlab("longitude") +
  ylab("latitude") +
  scale_y_continuous(breaks = (-2:2) * 30) +
  scale_x_continuous(breaks = (-4:4) * 45) +
 coord_sf(crs = "+proj=ortho +lat_0=38 +lon_0=120 +y_0=0 +ellps=WGS84 +no_defs")
worldmap
```

A zoomed version of the plot obtained from the following code is presented in the right panel of Figure 5.

```
mar = 20
zoommap = ggplot() +
  geom_sf(data = worldMap_sf, color = "grey", fill = "antiquewhite") +
  geom_sf(data = apw_spherical_df_sf, size = 0.8) +
  geom_sf(data = cp_long_lat_df_sf, color = "blue", shape = 23, size = 4) +
  geom_sf(data = fitted_cs_long_lat_df_sf, color = "red", size = 0.5) +
  xlab("longitude") +
```
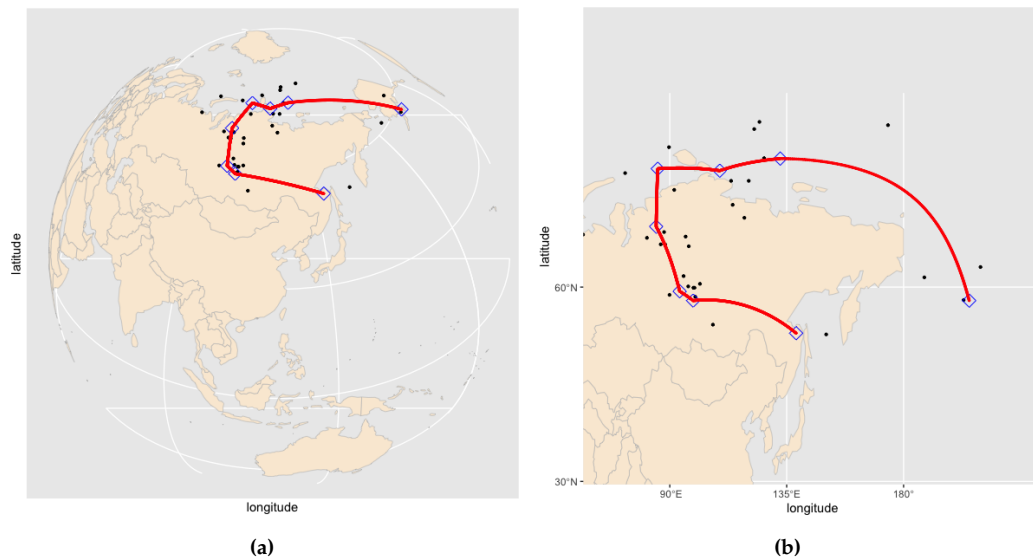
Figure 5: Plots of the APW path (red line) and the associated control points (blue points) obtained from the proposed method. The path goes from left to right in the plots. The left plot displays the path on the globe, and the right plot is a zoomed in version of the left plot on the projection map.

```
ylab("latitude") +
scale_y_continuous(breaks = (-2:2) * 30) +
scale_x_continuous(breaks = (-4:4) * 45) +
coord_sf(xlim = c(min(cp_long_lat_df$longitude) - mar,
                  max(cp_long_lat_df$longitude) + mar),
         ylim = c(min(cp_long_lat_df$latitude) - mar,
                  max(cp_long_lat_df$latitude) + mar))
zoommap
```

Goni Data

This section considers the tropical cyclone (TC) data provided by the Regional Specialized Meteorological Center (RSMC) Tokyo Typhoon Center. This data consists of the variables TC name, time, latitude, longitude, TC central pressure, and maximum sustained wind speed. To illustrate the proposed method and the **spheresmooth** package, we select the data concerning a cyclone named Goni observed from August 13th, 2015, to August 29th, 2015. The number of data points for Goni is 69, corresponding to each time point.

The Goni dataset is also included in the **spheresmooth** and can be loaded as follows.

```
goni_cartesian = spherical_to_cartesian(goni_spherical[, 2:3])
```

The code used for analysis is very similar to that used for computing the APW path. It involves transforming the data given in spherical coordinates to Cartesian coordinates, setting the initial dimension to 15, and fitting it with a sequence of complexity parameters. Subsequently, the optimal curve is selected using the BIC, and the coordinates of the corresponding control points are determined.

```
t = goni_spherical[, 1]
dimension = 15
initial_knots = knots_quantile(t, dimension = dimension)
lambda_seq = exp(seq(log(1e-07), log(1), length = 40))

fit = penalized_linear_spherical_spline(t = t, y = goni_cartesian,
```

```
                                          dimension = dimension,
                                          initial_knots = initial_knots,
                                          lambdas = lambda_seq)
# choose a curve that minimizes the BIC
best_index = which.min(fit$bic_list)
best_index

#> [1] 23

fit$dimension_list[best_index]

#> [1] 12

fit[[best_index]]$control_points

#>              [,1]      [,2]      [,3]
#>  [1,] -0.8572250 0.4708035 0.2085888
#>  [2,] -0.8205531 0.5235703 0.2292744
#>  [3,] -0.7429817 0.6032236 0.2899991
#>  [4,] -0.6430559 0.6947774 0.3221234
#>  [5,] -0.5554791 0.7659919 0.3235729
#>  [6,] -0.5088629 0.7970417 0.3252431
#>  [7,] -0.5041307 0.7884991 0.3523087
#>  [8,] -0.5073276 0.7575879 0.4107058
#>  [9,] -0.5258286 0.7266336 0.4421627
#> [10,] -0.5470298 0.6626324 0.5115434
#> [11,] -0.5418227 0.5904786 0.5981331
#> [12,] -0.4863508 0.5146343 0.7061263
```

The index of the curve that minimizes BIC is 23 with the corresponding dimension 12. The optimal complexity parameter is determined as

```
lambda_seq[best_index]

#> [1] 0.0008886238
```

We can visualize the fitted curve and obtain Figure 6 by executing code similar to that used in the previous section. The obtained path for Goni (red line) and the corresponding control points (blue rhombus-shaped points) are visualized along with the observations in Figure 6. Although the remaining control points may appear somewhat dense, in reality, these points are not redundant. The proposed penalization method utilizes the difference in velocity vectors to induce sparsity. Therefore, even control points that do not appear visually active on a map can provide valuable information for further analysis when viewed alongside time points, as they represent points where the velocity changes.

6 Summary

This paper introduced the **spheresmooth** package. We proposed a piecewise geodesic curve fitting method based on a velocity-based penalization scheme. The **spheresmooth** package implements the proposed method with the Riemannian block coordinate descent algorithm. It provides an automatic procedure for fitting a smooth path to a given set of noisy spherical data at known times. The **spheresmooth** package demonstrated its usefulness by applying the functions to the polar wander path data and tropical cyclone data. The methods presented in this paper not only advance spherical data analysis in general but also hold significant potential for application in astronomy, where spherical coordinate systems
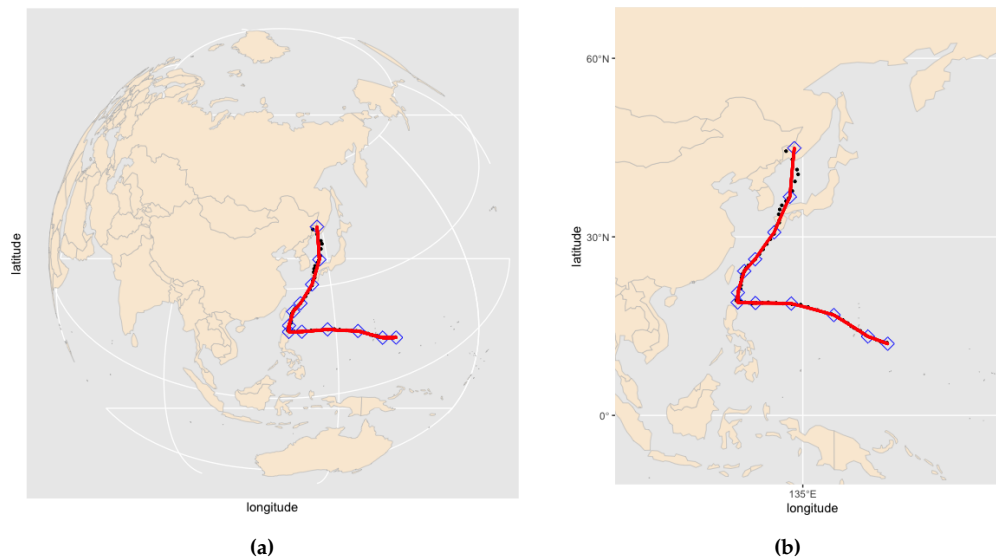
Figure 6: Plots of the Goni path (red line) and the associated control points (blue points) obtained from the proposed method. The path goes from left to right in the plots. The left plot displays the path on the globe, and the right plot is a zoomed in version of the left plot on the projection map.

and data are foundational. For example, these methods could be effectively utilized in analyzing celestial data, similar to the work conducted with CMB maps by the WMAP and Planck missions, demonstrating the versatility and applicability of our approach in handling complex, all-sky astronomical observations. We expect that the **spheresmooth** package will be helpful for applications in various fields, including statistics, machine learning, astronomy, cardiology, computer vision, physiology, and geophysics.

7 Acknowledgments

References

K.-Y. Bak, J.-K. Shin, and J.-Y. Koo. Intrinsic spherical smoothing method based on generalized Bézier curves and sparsity inducing penalization. *Journal of Applied Statistics*, 50(9): 1942–1961, 2023. [p1, 6, 7]

M. Camarinha, F. Silva Leite, and P. Crouch. Splines of class C^k on non-Euclidean spaces. *IMA Journal of Mathematical Control and Information*, 12(4):399–410, 1995. [p1]

M. P. do Carmo. *Differential Geometry of Curves and Surfaces*. Prentice-Hall, Englewood Cliffs, New Jersey, 1976. [p2]

M. P. do Carmo. *Riemannian Geometry*. Birkhäuser, 1992. [p2]

K. M. Górski, E. Hivon, A. J. Banday, B. D. Wandelt, F. K. Hansen, M. Reinecke, and M. Bartelmann. HEALPix: A framework for high-resolution discretization and fast analysis of data distributed on the sphere. *The Astrophysical Journal*, 622(2):759, 2005. [p2]

A. L. Gould. A regression technique for angular variates. *Biometrics*, pages 683–700, 1969. [p1]

P. E. Jupp and J. T. Kent. Fitting smooth paths to spherical data. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 36(1):34–46, 1987. [p1]

D. V. Kent and E. Irving. Influence of inclination error in sedimentary rocks on the Triassic and Jurassic apparent pole wander path for North America and implications for Cordilleran tectonics. *Journal of Geophysical Research: Solid Earth*, 115(B10), 2010. [p2, 14]

K.-R. Kim, I. L. Dryden, H. Le, and K. E. Severn. Smoothing splines on Riemannian manifolds, with applications to 3D shape space. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 83(1):108–132, 2021. [p1]

A. Kume, I. L. Dryden, and H. Le. Shape-space smoothing splines for planar landmark data. *Biometrika*, 94(3):513–528, 2007. [p1]

L. Machado, F. S. Leite, and K. Hüper. Riemannian means as solutions of variational problems. *LMS Journal of Computation and Mathematics*, 9:86–103, 2006. [p1]

D. Murdoch and D. Adler. *rgl: 3D Visualization Using OpenGL (R package version 1.3.1)*, 2024. URL https://CRAN.R-project.org/package=rgl. [p2]

L. Noakes, G. Heinzinger, and B. Paden. Cubic splines on curved spaces. *IMA Journal of Mathematical Control and Information*, 6(4):465–473, 1989. [p1]

A. Robotham. *sphereplot: Spherical Plotting (R package version 1.5.1)*, 2022. URL https://CRAN.R-project.org/package=sphereplot. [p2]

C. Samir, P. A. Absil, A. Srivastava, and E. Klassen. A gradient-descent method for curve fitting on Riemannian manifolds. *Foundations of Computational Mathematics*, 12(1):49–73, 2012. [p1]

G. Schwarz. Estimating the dimension of a model. *Annals of Statistics*, 6(2):461–464, 1978. [p7]

A. South. rworldmap: A new R package for mapping global data. *The R Journal*, 3(1):35–43, 2011. doi: 10.32614/RJ-2011-006. URL https://journal.r-project.org/articles/RJ-2011-006/. [p2]

J. Su, I. L. Dryden, E. Klassen, H. Le, and A. Srivastava. Fitting smoothing splines to time-indexed, noisy points on nonlinear manifolds. *Image and Vision Computing*, 30(6-7): 428–442, 2012. [p1]

R. Thompson and R. M. Clark. A robust least-squares Gondwanan apparent polar wander path and the question of palaeomagnetic assessment of Gondwanan reconstruction. *Earth and Planetary Science Letters*, 57(1):152–158, 1982. [p1]

H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. ISBN 978-3-319-24277-4. URL https://ggplot2.tidyverse.org. [p2]

Jae-Hwan Jhong
ChungBuk National University
Department of Information Statistics
1, Chungdae-ro, Seowon-gu, Cheongju-si, Chungcheongbuk-do, Republic of Korea
ORCiD: 0000-0003-2266-4986

Seyoung Lee
Sungshin Women's University

School of Mathematics, Statistics and Data Science
2, Bomun-ro 34da-gil, Seongbuk-gu, Seoul 02844, Republic of Korea

Ja-Yong Koo
Korea University
Department of Statistics
145, Anam-ro, Seongbuk-gu, Seoul 02841, Republic of Korea

Kwan-Young Bak
Sungshin Women's University
School of Mathematics, Statistics and Data Science
Data Science Center
2, Bomun-ro 34da-gil, Seongbuk-gu, Seoul 02844, Republic of Korea
ORCiD: 0000-0002-4541-160X
kybak@sungshin.ac.kr