Package: spheresmooth (via r-universe)

September 16, 2024

Type Package

Title Piecewise Geodesic Smoothing for Spherical Data

Version 0.1.0.9000

Description Fitting a smooth path to a given set of noisy spherical data observed at known time points. It implements a piecewise geodesic curve fitting method on the unit sphere based on a velocity-based penalization scheme. The proposed approach is implemented using the Riemannian block coordinate descent algorithm. To understand the method and algorithm, one can refer to Bak, K. Y., Shin, J. K., & Koo, J. Y. (2023) <doi:10.1080/02664763.2022.2054962> for the case of order 1. Additionally, this package includes various functions necessary for handling spherical data.

License GPL (>= 2)

Encoding UTF-8

LazyData true

RoxygenNote 7.3.1

Depends R (>= 2.10)

Suggests rgl, sphereplot

Repository https://kybak90.r-universe.dev

RemoteUrl https://github.com/kybak90/spheresmooth

RemoteRef HEAD

RemoteSha 034c48b2b644fbc2b6ef55714916891e5de7d194

Contents

apw_spherical 2
calculate_loss 2
Cartesian_to_Spherical 3
cross 3
cross_normalized 4

dot 5
edp 5
Exp 6
Geodesic 6
goni_spherical 7
knots_quantile 7
norm2 8
Normalize 8
penalized_linear_spherical_spline 9
piecewise_geodesic 10
spherical_dist 11
Spherical_to_Cartesian 12

Index **13**

`apw_spherical` *A polar wander dataset*

Description

A polar wander dataset presented in Kent and Irving (2010). 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. Our method is applied 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 first column represents the time points, and the remaining two columns provides the observed spherical coordinates.

`calculate_loss` *Calculate Loss Function*

Description

This function calculates the loss function based on the squared spherical distances between observed values and predicted values on the curve.

Usage

```
calculate_loss(y, gamma)
```

Arguments

`y`	Matrix of observed values.
`gamma`	Matrix of predicted values.

Value

Loss value.

Cartesian_to_Spherical *Convert Cartesian coordinates to cpherical coordinates*

Description

This function converts Cartesian coordinates to spherical coordinates.

Usage

```
Cartesian_to_Spherical(x)
```

Arguments

x	A matrix where each row represents a point in Cartesian coordinates.

Details

The Cartesian coordinates (x, y, z) are converted to spherical coordinates (theta, phi). Theta represents the inclination angle (0 to pi), and phi represents the azimuth angle (0 to 2*pi).

Value

A matrix where each row represents a point in spherical coordinates.

Examples

```
#example1
cartesian_points1 <- matrix(c(1/sqrt(3), 1/sqrt(3), 1/sqrt(3),-1/sqrt(3), 1/sqrt(3), -1/sqrt(3)),
  ncol = 3, byrow = TRUE)
Cartesian_to_Spherical(cartesian_points1)
#example2
cartesian_points2 <- matrix(c(1, 0, 0, 0, 1, 0, 0, 0, 1),ncol = 3, byrow = TRUE)
Cartesian_to_Spherical(cartesian_points2)
```

cross *Compute the cross product of two vectors*

Description

This function computes the cross product of two input vectors u and v.

Usage

```
cross(u, v)
```

Arguments

u	Numeric vector.
v	Numeric vector.

Value

Numeric vector representing the cross product of u and v.

Examples

```
cross(c(1,0,0), c(0,1,0))
```

---

cross_normalized *Compute the normalized cross product of two vectors*

Description

This function computes the cross product of two input vectors u and v, normalizes the result, and returns the normalized vector.

Usage

```
cross_normalized(u, v)
```

Arguments

u	Numeric vector.
v	Numeric vector.

Value

Numeric vector representing the normalized cross product of u and v.

Examples

```
cross_normalized(c(1,0,0), c(0,1,0))
```

dot — *Compute the dot product of two vectors*

Description

This function computes the dot product of two input vectors u and v.

Usage

```
dot(u, v)
```

Arguments

u	Numeric vector.
v	Numeric vector.

Value

Numeric value representing the dot product of u and v.

Examples

```
dot(c(1,2,3), c(4,5,6))
```

edp — *Compute the equal-distance projection of a point onto the xy plane*

Description

This function computes the equal-distance projection of a point p onto the xy plane.

Usage

```
edp(p)
```

Arguments

p	Numeric vector representing a point in Cartesian coordinates.

Value

Numeric vector representing the equal-distance projection of p onto the xy plane.

Exp — *Compute the exponential map on the unit sphere.*

Description

This function computes the exponential map on the unit sphere given a base point x and a vector v.

Usage

```
Exp(x, v)
```

Arguments

x	Numeric vector representing the base point.
v	Numeric vector representing a point.

Value

Numeric vector representing the result of the exponential map.

Examples

```
Exp(c(0,0,1), c(1,1,0))
```

Geodesic — *Compute the value of the geodesic curve connecting two points on the unit sphere for a given set of time points t*

Description

This function computes the value of the geodesic curve connecting two points p and q on the unit sphere at specified time points.

Usage

```
Geodesic(t, p, q, a, b)
```

Arguments

t	Numeric vector representing time points for the geodesic path.
p	Numeric vector representing the starting point on the sphere.
q	Numeric vector representing the ending point on the sphere.
a	Start time parameter.
b	End time parameter.

Value

Numeric matrix representing points along the geodesic path at specified time points.

Examples

```
Geodesic(c(0.25, 0.5, 0.75), c(1,0,0), c(0,1,0), 0, 1)
```

---

goni_spherical *A tropical cyclone dataset*

Description

A tropical cyclone dataset provided by the Regional Specialized Meteorological Center (RSMC) Tokyo Typhoon Center. We select a cyclone called 'Goni' observed over time on August, 2015. The first column represents the time points, and the remaining two columns provides the observed spherical coordinates.

---

knots_quantile *Generate knots for the piecewise geodesic curve based on the quantiles*

Description

This generates a sequence of knots for a given set of time points based on the quantiles.

Usage

```
knots_quantile(x, dimension, tiny = 1e-05)
```

Arguments

x	Numeric vector representing time points for the geodesic path.
dimension	Numeric vector the number of knots.
tiny	Numeric value representing a small constant that slightly expands the boundary.

Value

Numeric vector representing knots sequence in the time domain.

Examples

```
knots_quantile(seq(0, 1, length.out = 100), 10)
```

norm2 *Compute the L2 norm (Euclidean norm) of a vector*

Description

This function computes the L2 norm (Euclidean norm) of the input vector u.

Usage

```
norm2(u)
```

Arguments

u	Numeric vector.

Value

Numeric value representing the L2 norm of u.

Examples

```
norm2(c(1,2,3))
```

Normalize *Normalize a matrix row-wise*

Description

This function normalizes the rows of the input matrix x by dividing each row by its L2 norm (Euclidean norm).

Usage

```
Normalize(x)
```

Arguments

x	Numeric matrix.

Value

Numeric matrix with normalized rows.

Examples

```
Normalize(matrix(c(1,2,3,4,5,6), nrow = 2, byrow = TRUE))
```

`penalized_linear_spherical_spline` *Penalized Linear Spherical Spline*

Description

This function fits a penalized piecewise geodesic curve (linear spherical spline) to the given data.

Usage

```
penalized_linear_spherical_spline(
  t,
  y,
  initial_control_points = NULL,
  dimension,
  initial_knots,
  lambdas,
  step_size = 1,
  maxiter = 1000,
  epsilon_iter = 0.001,
  jump_eps = 1e-04,
  verbose = FALSE
)
```

Arguments

`t`	A numeric vector representing the time or location.
`y`	A matrix where each row represents a data point on the sphere.
`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 1.
`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-03.
`jump_eps`	A numeric value specifying the threshold for pruning control points based on jump size. Default is 1e-04.
`verbose`	A logical value indicating whether to print progress information. Default is FALSE.

Details

The goal is to find the optimal piecewise geodesic curve for the given spherical data while controlling model complexity through penalty terms. This function computes the optimal control points and knots for the given data and returns the fitted result. Internally, coordinate-wise gradient descent is used to minimize the loss function, and a penalty term is added to control the complexity of the model. The BIC (Bayesian Information Criterion) value is calculated according to the model's complexity to provide information for model selection. The function constructs piecewise curves using the piecewise_geodesic function and employs penalty terms to control the complexity of the model by updating control points and knots. To see how to use the function in practical applications, refer to the README or https://github.com/kybak90/spheresmooth.

Value

A list containing the fitted result for each complexity parameter and BIC values for model selection. One might choose the element that corresponds to the minimum BIC values as illustrated in the example.

`piecewise_geodesic` *Piecewise Geodesic*

Description

This function computes a piecewise geodesic path between control points.

Usage

```
piecewise_geodesic(t, control_points, knots)
```

Arguments

`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.

Details

This function calculates the piecewise geodesic curve between control points based on the provided knots. The geodesic curve is computed segment by segment between adjacent control points. It interpolates the path between control points in a geodesic manner, ensuring the shortest path along the surface.

Value

A matrix containing the piecewise geodesic path.

Examples

```
# `rgl` package and `sphereplot` pacakges are needed for the visualizaiton of the following example.
# Define control points and knots
library(rgl)
library(sphereplot)
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 = 45, 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)
```

`spherical_dist` *Calculate spherical distance between two vectors*

Description

This function calculates the spherical distance between two vectors.

Usage

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

Arguments

x	A numeric vector.
y	A numeric vector.

Value

The distance between vectors x and y.

Examples

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

`Spherical_to_Cartesian` *Convert spherical coordinates to Cartesian coordinates*

Description

This function converts spherical coordinates (theta, phi) to Cartesian coordinates.

Usage

```
Spherical_to_Cartesian(theta_phi)
```

Arguments

`theta_phi`	A matrix where each row contains the spherical coordinates (theta, phi) of a point.

Value

A matrix where each row contains the Cartesian coordinates (x, y, z) of a point.

Examples

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

Index

apw_spherical, 2

calculate_loss, 2
Cartesian_to_Spherical, 3
cross, 3
cross_normalized, 4

dot, 5

edp, 5
Exp, 6

Geodesic, 6
goni_spherical, 7

knots_quantile, 7

norm2, 8
Normalize, 8

penalized_linear_spherical_spline, 9
piecewise_geodesic, 10

spherical_dist, 11
Spherical_to_Cartesian, 12