Package ‘tectonicr’

May 27, 2024

Title  Analyzing the Orientation of Maximum Horizontal Stress

Version  0.3.2


License  GPL (>= 3)

URL  https://tobiste.github.io/tectonicr/

BugReports  https://github.com/tobiste/tectonicr/issues

Depends  R (>= 4.1.0)

Imports  boot, dplyr, ggplot2, methods, RColorBrewer, sf, smother, spatstat.explore, spatstat.geom, spatstat.univar, spatstat.utils, terra, tidyr, viridis, zoo

Suggests  ggforce, knitr, rmarkdown, roxygen2, testthat (>= 3.0.0), tidyterra

VignetteBuilder  knitr

Config/testthat/edition  3

Encoding  UTF-8

Language  en-US

LazyData  true

RoxygenNote  7.3.1

NeedsCompilation  no

Author  Tobias Stephan [aut, cre] (<https://orcid.org/0000-0002-9290-014X>)

Maintainer  Tobias Stephan <tobias.stephan1@yahoo.com>

Repository  CRAN

Date/Publication  2024-05-27 16:10:02 UTC
R topics documented:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs_vel</td>
<td>3</td>
</tr>
<tr>
<td>angle-conversion</td>
<td>4</td>
</tr>
<tr>
<td>angle_vectors</td>
<td>5</td>
</tr>
<tr>
<td>axes</td>
<td>5</td>
</tr>
<tr>
<td>circle_stats</td>
<td>6</td>
</tr>
<tr>
<td>circular Dispersion_boot</td>
<td>8</td>
</tr>
<tr>
<td>circular_sd_error</td>
<td>9</td>
</tr>
<tr>
<td>compact_grid</td>
<td>10</td>
</tr>
<tr>
<td>confidence</td>
<td>11</td>
</tr>
<tr>
<td>conjugate_Q4</td>
<td>12</td>
</tr>
<tr>
<td>coordinates</td>
<td>13</td>
</tr>
<tr>
<td>coordinates2</td>
<td>14</td>
</tr>
<tr>
<td>coordinate_mod</td>
<td>15</td>
</tr>
<tr>
<td>cpm_models</td>
<td>15</td>
</tr>
<tr>
<td>deviation_norm</td>
<td>16</td>
</tr>
<tr>
<td>deviation_shmax</td>
<td>17</td>
</tr>
<tr>
<td>dispersion</td>
<td>18</td>
</tr>
<tr>
<td>distance_from_pb</td>
<td>20</td>
</tr>
<tr>
<td>distance_mod</td>
<td>21</td>
</tr>
<tr>
<td>dist_greatcircle</td>
<td>21</td>
</tr>
<tr>
<td>earth_radius</td>
<td>22</td>
</tr>
<tr>
<td>equivalent_rotation</td>
<td>23</td>
</tr>
<tr>
<td>est.kappa</td>
<td>24</td>
</tr>
<tr>
<td>euler_pole</td>
<td>24</td>
</tr>
<tr>
<td>euler_to_Q4</td>
<td>25</td>
</tr>
<tr>
<td>get_azimuth</td>
<td>25</td>
</tr>
<tr>
<td>get_distance</td>
<td>26</td>
</tr>
<tr>
<td>get_projected_pb_strike</td>
<td>27</td>
</tr>
<tr>
<td>get_relrot</td>
<td>27</td>
</tr>
<tr>
<td>iceland</td>
<td>28</td>
</tr>
<tr>
<td>is.euler</td>
<td>28</td>
</tr>
<tr>
<td>is.Q4</td>
<td>29</td>
</tr>
<tr>
<td>kernel_dispersion</td>
<td>29</td>
</tr>
<tr>
<td>kuiper_test</td>
<td>31</td>
</tr>
<tr>
<td>line_azimuth</td>
<td>32</td>
</tr>
<tr>
<td>mean_resultant_length</td>
<td>33</td>
</tr>
<tr>
<td>model_shmax</td>
<td>34</td>
</tr>
<tr>
<td>normalize_Q4</td>
<td>35</td>
</tr>
<tr>
<td>norm_chisq</td>
<td>35</td>
</tr>
<tr>
<td>nuvel1</td>
<td>36</td>
</tr>
<tr>
<td>nuvel1_plates</td>
<td>37</td>
</tr>
<tr>
<td>parse_wsm</td>
<td>38</td>
</tr>
<tr>
<td>pb2002</td>
<td>38</td>
</tr>
<tr>
<td>plates</td>
<td>39</td>
</tr>
<tr>
<td>PoR2Geo_azimuth</td>
<td>40</td>
</tr>
<tr>
<td>PoR_coordinates</td>
<td>41</td>
</tr>
</tbody>
</table>
**Index**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs_vel</td>
<td><strong>Absolute Plate Velocity</strong></td>
</tr>
</tbody>
</table>

**Description**

Calculates the absolute angular velocity of plate motion
Usage

```
abs_vel(w, alpha, r = earth_radius())
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>w</code></td>
<td>Angular velocity or rate or angle of rotation</td>
</tr>
<tr>
<td><code>alpha</code></td>
<td>Angular distance to Euler pole or small circle around Euler pole</td>
</tr>
<tr>
<td><code>r</code></td>
<td>Radius. Default is WGS84 Earth’s radius (6371.009 km)</td>
</tr>
</tbody>
</table>

Value

numeric (unit of velocity: km/Myr)

See Also

`earth_radius()`

Examples

```
abs_vel(0.21, 0)
abs_vel(0.21, 45)
abs_vel(0.21, 90)
```

---

### angle-conversion

#### Degrees to Radians

**Description**

Helper functions to transform between angles in degrees and radians.

**Usage**

```
rad2deg(rad)
deg2rad(deg)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>rad</code></td>
<td>(array of) angles in radians.</td>
</tr>
<tr>
<td><code>deg</code></td>
<td>(array of) angles in degrees.</td>
</tr>
</tbody>
</table>

**Value**

numeric. angle in degrees or radians.

**Examples**

```
deg2rad(seq(-90, 90, 15))
rad2deg(seq(-pi / 2, pi / 2, length = 13))
```
angle_vectors  

**Angle Between Two Vectors**

**Description**
Calculates the angle between two vectors

**Usage**

```r
angle_vectors(x, y)
```

**Arguments**

- `x, y` Vectors in Cartesian coordinates. Can be vectors of three numbers or a matrix of 3 columns (x, y, z)

**Value**
numeric. angle in degrees

**Examples**

```r
t <- c(1, -2, 3)
v <- c(-2, 1, 1)
angle_vectors(t, v)
```

---

axes  

**Plot axes**

**Description**
Plot axes

**Usage**

```r
axes(
    x,
    y,
    angle,
    radius = 0.5,
    arrow.code = 1,
    arrow.length = 0,
    add = FALSE,
    ...
)
```
circle_stats

Summary Statistics of Circular Data

Description

Calculate the (weighted median) and standard deviation of orientation data.

Usage

circular_mean(x, w = NULL, axial = TRUE, na.rm = TRUE)
circular_var(x, w = NULL, axial = TRUE, na.rm = TRUE)
circular_sd(x, w = NULL, axial = TRUE, na.rm = TRUE)
circular_median(x, w = NULL, axial = TRUE, na.rm = TRUE)
circular_quantiles(x, w = NULL, axial = TRUE, na.rm = TRUE)
circular_IQR(x, w = NULL, axial = TRUE, na.rm = TRUE)
Arguments

- **x**: numeric vector. Values in degrees.
- **w**: (optional) Weights. A vector of positive numbers and of the same length as x.
- **axial**: logical. Whether the data are axial, i.e., pi-periodical (TRUE, the default) or directional, i.e., 2π-periodical (FALSE).
- **na.rm**: logical value indicating whether NA values in x should be stripped before the computation proceeds.

Value

numeric vector

Note

Weighting may be the reciprocal of the data uncertainties.

Weightings have no effect on quasi-median and quasi-quantiles if length(x) %% 2 != 1 and length(x) %% 4 == 0, respectively.

References


Examples

```r
x <- rvm(10, 0, 100) %% 180
unc <- stats::runif(100, 0, 10)
circular_mean(x, 1 / unc)
circular_var(x, 1 / unc)
circular_sd(x, 1 / unc)
circular_median(x, 1 / unc)
circular_quantiles(x, 1 / unc)
circular_IQR(x, 1 / unc)

data("san_andreas")
circular_mean(san_andreas$azi)
circular_mean(san_andreas$azi, 1 / san_andreas$unc)
circular_median(san_andreas$azi)
circular_median(san_andreas$azi, 1 / san_andreas$unc)
circular_quantiles(san_andreas$azi)
circular_quantiles(san_andreas$azi, 1 / san_andreas$unc)
circular_var(san_andreas$azi)
circular_var(san_andreas$azi, 1 / san_andreas$unc)
```
circular_dispersion_boot

Bootstrapped estimates for circular dispersion

Description

Calculates bootstrapped estimates of the circular dispersion, its standard error and its confidence interval.

Usage

circular_dispersion_boot(
x, y = NULL, w = NULL, w.y = NULL, R = 1000, conf.level = 0.95, ...
)

Arguments

x numeric values in degrees.
y numeric. The angle(s) about which the angles x disperse (in degrees).
w, w.y (optional) Weights for x and y, respectively. A vector of positive numbers and of the same length as x.
R The number of bootstrap replicates. positive integer (1000 by default).
conf.level Level of confidence: \((1 - \alpha\%) / 100\). (0.95 by default).
... optional arguments passed to \texttt{boot::boot()}

Value

list containing:

MLE the maximum likelihood estimate of the circular dispersion
sde standard error of MLE
CI lower and upper limit of the confidence interval of MLE
circular_sd_error

See Also

circular_dispersion()

Examples

data("nuvel1")
PoR <- subset(nuvel1, nuvel1$plate.rot == "na")
sa.por <- PoR_shmax(san_andreas, PoR, "right")
circular_dispersion(sa.por$azi.PoR, y = 135, w = 1 / san_andreas$unc)
circular_dispersion_boot(sa.por$azi.PoR, y = 135, w = 1 / san_andreas$unc, R = 1000)

Description

Measure of the chance variation expected from sample to sample in estimates of the mean direction. The approximated standard error of the mean direction is computed by the mean resultant length and the MLE concentration parameter \( \kappa \).

Usage

circular_sd_error(x, w = NULL, axial = TRUE, na.rm = TRUE)

Arguments

x numeric vector. Values in degrees.
w (optional) Weights. A vector of positive numbers and of the same length as x.
axial logical. Whether the data are axial, i.e. pi-periodical (TRUE, the default) or directional, i.e. 2\(\pi\)-periodical (FALSE).
na.rm logical value indicating whether NA values in x should be stripped before the computation proceeds.

Value

Angle in degrees

References


See Also

mean_resultant_length().circular_mean()
compact_grid

### Examples

```
# Example data from Davis (1986), pp. 316
finland_stria <- c(23, 27, 53, 58, 64, 83, 85, 88, 93, 99, 100, 105, 113,
                   113, 114, 117, 121, 123, 125, 126, 126, 127, 128, 128, 129, 132,
                   132, 132, 134, 135, 137, 144, 145, 145, 146, 153, 155, 155, 155, 157, 163,
                   165, 171, 172, 179, 181, 186, 190, 212)
circular_sd_error(finland_stria, axial = FALSE)
```

data(san_andreas)
data("nuvel1")
PoR <- subset(nuvel1, nuvel1$plate.rot == "na")
sa.por <- PoR_shmax(san_andreas, PoR, "right")
circular_sd_error(sa.por$azi.PoR, w = 1 / san_andreas$unc)

---

### compact_grid

**Compact smoothed stress field**

**Description**

Filter smoothed stress field containing a range of search radii or kernel half widths to find smallest wavelength (R) with the least circular sd. or dispersion for each coordinate, respectively.

**Usage**

```
compact_grid(x, type = c("stress", "dispersion"))
```

**Arguments**

- **x**: output of `stress2grid()`, `PoR_stress2grid()`, or `kernel_dispersion()`
- **type**: character. Type of the grid x. Either "stress" (when input is `stress2grid()`) or `PoR_stress2grid()`) or "dispersion" (when input is `kernel_dispersion()`).

**Value**

sf object

**See Also**

`stress2grid()`, `PoR_stress2grid()`, `kernel_dispersion()`

**Examples**

```
data("san_andreas")
res <- stress2grid(san_andreas)
compact_grid(res)
```
### Description

Probabilistic limit on the location of the true or population mean direction, assuming that the estimation errors are normally distributed.

### Usage

```r
confidence_angle(x, conf.level = 0.95, w = NULL, axial = TRUE, na.rm = TRUE)
confidence_interval(x, conf.level = 0.95, w = NULL, axial = TRUE, na.rm = TRUE)
```

### Arguments

- **x**: numeric vector. Values in degrees.
- **conf.level**: Level of confidence: \((1 - \alpha\%)/100\). (0.95 by default).
- **w**: (optional) Weights. A vector of positive numbers and of the same length as `x`.
- **axial**: logical. Whether the data are axial, i.e. \(\pi\)-periodical (TRUE, the default) or directional, i.e. \(2\pi\)-periodical (FALSE).
- **na.rm**: logical value indicating whether NA values in `x` should be stripped before the computation proceeds.

### Details

The confidence angle gives the interval, i.e. plus and minus the confidence angle, around the mean direction of a particular sample, that contains the true mean direction under a given level of confidence.

### Value

Angle in degrees

### References


### See Also

`mean_resultant_length()`, `circular_sd_error()`
conjugate_Q4

Conjugation of a Quaternion

Description

Inverse rotation given by conjugated quaternion

Usage

conjugate_Q4(q, normalize = FALSE)

Arguments

q

object of class "quaternion"

normalize

logical. Whether a quaternion normalization should be applied (TRUE) or not (FALSE, the default).

Value

object of class "quaternion"


**Description**

Converts vector between Cartesian and geographical coordinate systems

**Usage**

- `cartesian_to_geographical(n)`
- `geographical_to_cartesian(p)`
- `geographical_to_spherical(p)`

**Arguments**

- `n` Cartesian coordinates (x, y, z) as vector
- `p` Geographical coordinates (latitude, longitude) as vector

**Value**

Functions return a (2- or 3-dimensional) vector representing a point in the requested coordinate system.

**See Also**

- `cartesian_to_spherical()` and `spherical_to_cartesian()` for conversions to spherical coordinates

**Examples**

```r
n <- c(1, -2, 3)
cartesian_to_geographical(n)
p <- c(50, 10)
geographical_to_cartesian(p)
```
Description

Converts vector between Cartesian and spherical coordinate systems

Usage

cartesian_to_spherical(n)
spherical_to_cartesian(p)
spherical_to_geographical(p)

Arguments

n  Cartesian coordinates (x, y, z) as three-column vector
p  Spherical coordinates (colatitude, azimuth) as two-column vector

Value

Functions return a (2- or 3-dimensional) vector representing a point in the requested coordinate system.

See Also

cartesian_to_geographical() and geographical_to_cartesian() for conversions to geographical coordinates

Examples

n <- c(1, -2, 3)
cartesian_to_spherical(n)
p <- c(50, 10)
spherical_to_cartesian(p)
**coordinate_mod**

**Coordinate Correction**

**Description**

Corrects the longitudes or latitudes to value between -180.0 and 180.0 or -90 and 90 degree

**Usage**

`longitude_modulo(x)`

`latitude_modulo(x)`

**Arguments**

- **x** Longitude(s) or latitude(s) in degrees

**Value**

numeric

**Examples**

`longitude_modulo(-361 + 5 * 360)`

`latitude_modulo(-91 + 5 * 180)`

**cpm_models**

**Global model of current plate motions**

**Description**

Compilation of global models for current plate motions, including NNR-NUVEL1A (DeMets et al., 1990), NNR-MORVEL56 (Argus et al., 2011), REVEL (Sella et al., 2002), GSRM2.1 (Kreemer et al., 2014) HS-NUVEL1A (Gripp and Gordon, 2002), and PB2002 (Bird, 2003)

**Usage**

`data('cpm_models')`
deviation_norm

Format

An object of class data.frame

plate.name  The rotating plate
plate.rot  The abbreviation of the plate’s name
lat,lon  Coordinates of the Pole of Rotation
angle  The amount of rotation (angle in 1 Myr)
plate.fix  The anchored plate, i.e. plate.rot moves relative to plate.fix
model  Model for current global plate motion

References


Examples

data("cpm_models")
head("cpm_models")

deviation_norm(x, y = NULL)

normalize_angle between two directions

Description

Normalizes the angle between two directions to the acute angle in between, i.e. angles between 0 and 90°

Usage

deviation_norm(x, y = NULL)
deviation_shmax

Arguments

x, y

Minuend and subtrahend. Both numeric vectors of angles in degrees. If y is missing, it treats x as difference. If not, length of subtrahend y is either 1 or equal to length(x).

Value

numeric vector, acute angles between two directions, i.e. values between 0 and 90°

Author(s)

Tobias Stephan

Examples

deviation_norm(175, 5)
deviation_norm(c(175, 95, 0), c(5, 85, NA))
deviation_norm(c(-5, 85, 95, 175, 185, 265, 275, 355, 365))

deviation_shmax

Deviation of Observed and Predicted Directions of Maximum Horizontal Stress

Description

Calculate the angular difference between the observed and modeled direction of maximum horizontal stresses ($\sigma_{Hmax}$) along great circles, small circles, and loxodromes of the relative plate motion’s Euler pole

Usage

deviation_shmax(prd, obs)

Arguments

prd data.frame containing the modeled azimuths of $\sigma_{Hmax}$, i.e. the return object from model_shmax()

obs Numeric vector containing the observed azimuth of $\sigma_{Hmax}$, same length as prd

Value

An object of class data.frame

dev.gc Deviation of observed stress from modeled $\sigma_{Hmax}$ following great circles
dev.sc Small circles
dev.ld.cw Clockwise loxodromes
dev.ld.ccw Counter-clockwise loxodromes
dispersion

Author(s)
Tobias Stephan

References

See Also
model_shmax() to calculate the theoretical direction of $\sigma_{Hmax}$.

Examples
data("nuvel1")
# North America relative to Pacific plate:
PoR <- subset(nuvel1, nuvel1$plate.rot == "na")

# the point where we want to model the SHmax direction:
point <- data.frame(lat = 45, lon = 20)

prd <- model_shmax(point, PoR)
deviation_shmax(prd, obs = 90)

dispersion
Circular distance and dispersion

Description
Circular distance between two angles and circular dispersion of angles about a specified angle.

Usage
circular_distance(x, y, axial = TRUE, na.rm = TRUE)
circular_dispersion(
  x,
  y = NULL,
  w = NULL,
  w.y = NULL,
  norm = FALSE,
  axial = TRUE,
  na.rm = TRUE
)
Arguments

- **x, y** vectors of numeric values in degrees. `length(y)` is either 1 or `length(x)`
- **axial** logical. Whether the data are axial, i.e. pi-periodical (TRUE, the default) or directional, i.e. 2π-periodical (FALSE).
- **na.rm** logical. Whether NA values in x should be stripped before the computation proceeds.
- **w, w.y** (optional) Weights. A vector of positive numbers and of the same length as x. w.y is the (optional) weight of y.
- **norm** logical. Whether the dispersion should be normalized by the maximum possible angular difference.

Value

- `circular_distance` returns a numeric vector of positive numbers, `circular_dispersion` returns a positive number.

Note

If from is NULL, than the circular variance is returned.

References


See Also

- `circular_mean()`
- `circular_var()`

Examples

```r
a <- c(0, 2, 359, 6, 354)
circular_distance(a, 10) # distance to single value

b <- a + 90
circular_distance(a, b) # distance to multiple values

data("nuvel1")
PoR <- subset(nuvel1, nuvel1$plate.rot == "na")
sa.por <- PoR_shmax(san_andreas, PoR, "right")
circular Dispersion(sa.por$azi.PoR, y = 135)
circular Dispersion(sa.por$azi.PoR, y = 135, w = 1 / san_andreas$unc)
```
**distance_from_pb**  
*Distance from plate boundary*

**Description**

Absolute distance of data points from the nearest plate boundary in degree

**Usage**

```r
distance_from_pb(x, PoR, pb, tangential = FALSE, km = FALSE, ...)
```

**Arguments**

- `x`, `pb` sf objects of the data points and the plate boundary geometries in the geographical coordinate system
- `PoR` Pole of Rotation. "data.frame" or object of class "euler.pole" containing the geographical coordinates of the Pole of Rotation
- `tangential` Logical. Whether the plate boundary is a tangential boundary (TRUE) or an inward and outward boundary (FALSE, the default).
- `km` Logical. Whether the distance is expressed in kilometers (TRUE) or in degrees (FALSE, the default).
- `...` optional arguments passed to `smoothr::densify()`

**Details**

The distance to the plate boundary is the longitudinal or latitudinal difference between the data point and the plate boundary (along the closest latitude or longitude) for inward/outward or tangential plate boundaries, respectively.

**Value**

Numeric vector of the great circle distances

**References**


**Examples**

```r
data("nuvel1")
na_pa <- subset(nuvel1, nuvel1$plate.rot == "na")

data("plates")
plate_boundary <- subset(plates, plates$pair == "na-pa")
data("san_andreas")
```
```
res <- distance_from_pb(
  x = san_andreas, PoR = na_pa, pb = plate_boundary, tangential = TRUE
)
head(res)

res.km <- distance_from_pb(
  x = san_andreas, PoR = na_pa, pb = plate_boundary, tangential = TRUE, km = TRUE
)
range(res.km)
```

---

**distance_mod**

Normalize angular distance on a sphere distance

**Description**

Helper function to express angular distance on the sphere in the range of 0 to 180 degrees

**Usage**

```r
distance_mod(x)
```

**Arguments**

- **x**
  - numeric, angular distance (in degrees)

**Value**

numeric vector

---

**dist.greatcircle**

Distance between points

**Description**

Returns the great circle distance between a location and all grid point in km

**Usage**

```r
dist.greatcircle(
  lat1, lon1, lat2, lon2,
  r = earth_radius(),
  method = c("haversine", "orthodrome", "vincenty", "euclidean")
)
```
Arguments

lat1, lon1 numeric vector. coordinate of point(s) 1 (degrees).
lat2, lon2 numeric vector. coordinates of point(s) 2 (degrees).
r numeric. radius of the sphere (default = 6371.0087714 km, i.e. the radius of the Earth)
method Character. Formula for calculating great circle distance, one of:
"haversine" great circle distance based on the haversine formula that is optimized for 64-bit floating-point numbers (the default)
"orthodrome" great circle distance based on the spherical law of cosines
"vincenty" distance based on the Vincenty formula for an ellipsoid with equal major and minor axes
"euclidean" Euclidean distance (not great circle distance!)

Value

numeric vector with length equal to length(lat1)

See Also

orthodrome(), haversine(), vincenty()

Examples

dist_greatcircle(lat1 = 20, lon1 = 12, lat2 = c(50, 30), lon2 = c(40, 32))
dist_greatcircle(
  lat1 = 20, lon1 = 12, lat2 = c(50, 30), lon2 = c(40, 32),
  method = "orthodrome"
)
dist_greatcircle(
  lat1 = 20, lon1 = 12, lat2 = c(50, 30), lon2 = c(40, 32),
  method = "vincenty"
)
dist_greatcircle(
  lat1 = 20, lon1 = 12, lat2 = c(50, 30), lon2 = c(40, 32),
  method = "euclidean"
)

earth_radius

Description

IERS mean radius of Earth in km (based on WGS 84)

Usage

earth_radius()
equivalent_rotation

Value
numeric value

---

Description
Transforms a sequence of rotations into a new reference system

Usage
equivalent_rotation(x, fixed, rot)

Arguments
- x: Object of class "data.frame" containing the Euler poles of plate rotations:
  - plate.rot: Moving plate
  - lat, lon: coordinates of Euler pole
  - angle: Angle of rotation
  - plate.fix: Fixed plate
- fixed: plate that will be regarded as fixed. Has to be one out of x$plate.fix
- rot: (optional) plate that will be regarded as rotating. Has to be one out of x$plate.rot.

Value
sequence of plate rotations in new reference system. Same object class as x

See Also
relative_rotation()

Examples
data(nuvel1) # load the NUVEL1 rotation parameters

# all nuvel1 rotation equivalent to fixed Africa:
equivalent_rotation(nuvel1, fixed = "af")
# relative plate motion between Eurasia and India:
equivalent_rotation(nuvel1, "eu", "in")
### est.kappa

*Concentration parameter of von Mises distribution*

**Description**

Computes the maximum likelihood estimate of \( \kappa \), the concentration parameter of a von Mises distribution, given a set of angular measurements.

**Usage**

```r
est.kappa(x, w = NULL, bias = FALSE, ...)
```

**Arguments**

- `x`: numeric. angles in degrees
- `w`: numeric. weightings
- `bias`: logical parameter determining whether a bias correction is used in the computation of the MLE. Default for bias is `FALSE` for no bias correction.
- `...`: optional parameters passed to `circular_mean()`

**Value**

numeric.

**Examples**

```r
est.kappa(rvm(100, 90, 10), w = 1 / runif(100, 0, 10))
```

---

### euler_pole

*Euler pole object*

**Description**

Creates an object of the orientation of the Euler pole axis

**Usage**

```r
euler_pole(x, y, z = NA, geo = TRUE, angle = NA)
```

**Arguments**

- `x`: latitude or x coordinate of Euler pole axis
- `y`: longitude or y
- `z`: z coordinate
- `geo`: logical, `TRUE` (the default) if Euler pole axis is given in geographical coordinates (latitude, longitude). `FALSE` if given in Cartesian coordinates \((x, y, z)\)
- `angle`: (optional) Angle of rotation in degrees (CCW rotation if angle is positive)
**euler_to_Q4**

**Value**

An object of class "euler.pole" containing the Euler pole axis in both geographical and Cartesian coordinates and the angle of rotation in radians.

**Examples**

euler_pole(90, 0, angle = 45)
euler_pole(0, 0, 1, geo = FALSE)

---

**get_azimuth**

**Azimuth Between two Points**

**Description**

Calculate initial bearing (or forward azimuth/direction) to go from point a to point b following great circle arc on a sphere.

**Usage**

get_azimuth(lat_a, lon_a, lat_b, lon_b)

**Arguments**

- **lat_a, lat_b** Numeric. Latitudes of a and b (in degrees).
- **lon_a, lon_b** Numeric. Longitudes of a and b (in degrees).
Details

`get_azimuth()` is based on the spherical law of tangents. This formula is for the initial bearing (sometimes referred to as forward azimuth) which if followed in a straight line along a great circle arc will lead from the start point `a` to the end point `b`.

\[
\theta = \arctan \frac{\sin \Delta \lambda \cos \psi_2, \cos \psi_1 \sin \psi_1 - \sin \psi_1 \cos \psi_2 \cos \Delta \lambda}{\cos \psi_1 \sin \psi_1 - \sin \psi_1 \cos \psi_2 \cos \Delta \lambda}
\]

where \(\psi_1, \lambda_1\) is the start point, \(\psi_2, \lambda_2\) the end point (\(\Delta \lambda\) is the difference in longitude).

Value

numeric. Azimuth in degrees

References

http://www.movable-type.co.uk/scripts/latlong.html

Examples

berlin <- c(52.517, 13.4) # Berlin
tokyo <- c(35.7, 139.767) # Tokyo
get_azimuth(berlin[1], berlin[2], tokyo[1], tokyo[2])

get_distance

Helper function to Distance from plate boundary

Description

Helper function to Distance from plate boundary

Usage

get_distance(lon, lat, pb.coords, tangential, km)

Arguments

lon, lat numeric vectors
pb.coords matrix
tangential, km logical

See Also

distance_from_pb()
get_projected_pb_strike

Helper function to get Distance from plate boundary

Description
Helper function to get Distance from plate boundary

Usage
get_projected_pb_strike(lon, lat, pb.coords, pb.bearing, tangential)

Arguments
- lon, lat, pb.bearing: numeric vectors
- pb.coords: matrix
- tangential: logical

See Also
- projected_pb_strike()

get_relrot

Helper function to Equivalent rotation

Description
Helper function to Equivalent rotation

Usage
get_relrot(plate.rot, lat, lon, angle, fixed, fixed.ep)

Arguments
- plate.rot, fixed: character or numeric
- lat, lon, angle: numeric
- fixed.ep: data.frame

See Also
- equivalent_rotation()
Example stress data for outward-moving displaced plate boundary

Description

Usage
data('iceland')

Format
An object of class "sf"

Source
https://www.world-stress-map.org/

References

Examples
data("iceland")
head(iceland)

Check if object is euler.pole

Description
Check if object is euler.pole

Usage
is.euler(x)

Arguments
x object of class "euler.pole"
is.Q4

Value

logical

is.Q4 Check if object is quaternion

Description

Check if object is quaternion

Usage

is.Q4(x)

Arguments

x object of class "quaternion"

Value

logical

kernel_dispersion Adaptive Kernel Dispersion

Description

Stress field and wavelength analysis using circular dispersion (or other statistical estimators for dispersion)

Usage

kernel_dispersion(x,
    stat = c("dispersion", "nchisq", "rayleigh"),
    grid = NULL,
    lon_range = NULL,
    lat_range = NULL,
    gridsize = 2.5,
    min_data = 3,
    threshold = 1,
    arte_thres = 200,
    dist_threshold = 0.1,
    R_range = seq(100, 2000, 100),
    ...
)
**Arguments**

- `x`: sf object containing
  - `azi`: Azimuth in degree
  - `unc`: Uncertainties of azimuth in degree
  - `prd`: Predicted value for azimuth

- `stat`: The measurement of dispersion to be calculated. Either "dispersion" (default), "nchisq", or "rayleigh" for circular dispersion, normalized Chi-squared test statistic, or Rayleigh test statistic.

- `grid`: (optional) Point object of class sf.
  - `lon_range`, `lat_range`: (optional) numeric vector specifying the minimum and maximum longitudes and latitudes (are ignored if "grid" is specified).
  - `gridsize`: Numeric. Target spacing of the regular grid in decimal degree. Default is 2.5. (is ignored if "grid" is specified)
  - `min_data`: Integer. Minimum number of data per bin. Default is 3
  - `threshold`: Numeric. Threshold for stat value (default is 1)
  - `arte_thres`: Numeric. Maximum distance (in km) of the grid point to the next data point. Default is 200
  - `dist_threshold`: Numeric. Distance weight to prevent overweight of data nearby (0 to 1). Default is 0.1
  - `R_range`: Numeric value or vector specifying the (adaptive) kernel half-width(s) as search radius (in km). Default is `seq(50, 1000, 50)`

- `...`: optional arguments to `dist_greatcircle()`

**Value**

- sf object containing
  - `lon`,`lat`: longitude and latitude in degree
  - `stat`: output of function defined in `stat`
  - `R`: The search radius in km.
  - `mdr`: Mean distance of datapoints per search radius
  - `N`: Number of data points in search radius

**See Also**

- `circular_dispersion()`, `norm_chisq()`, `rayleigh_test()`

**Examples**

```R
data("nuvel1")
PoR <- subset(nuvel1, nuvel1$plate.rot == "na")
san_andreas_por <- san_andreas
san_andreas_por$azi <- PoR_shmax(san_andreas, PoR, "right")$azi.PoR
san_andreas_por$prd <- 135
kernel_dispersion(san_andreas_por)
```
**Kuiper Test of Circular Uniformity**

**Description**

Kuiper's test statistic is a rotation-invariant Kolmogorov-type test statistic. The critical values of a modified Kuiper's test statistic are used according to the tabulation given in Stephens (1970).

**Usage**

```r
kuiper_test(x, alpha = 0, axial = TRUE)
```

**Arguments**

- `x`: numeric vector containing the circular data which are expressed in degrees
- `alpha`: Significance level of the test. Valid levels are 0.01, 0.05, and 0.1. This argument may be omitted (NULL, the default), in which case, a range for the p-value will be returned.
- `axial`: logical. Whether the data are axial, i.e. $\pi$-periodical (TRUE, the default) or circular, i.e. $2\pi$-periodical (FALSE).

**Details**

If `statistic > p.value`, the null hypothesis is rejected. If not, randomness (uniform distribution) cannot be excluded.

**Value**

list containing the test statistic `statistic` and the significance level `p.value`.

**Examples**

```r
# Example data from Mardia and Jupp (2001), pp. 93
pidgeon_homing <- c(55, 60, 65, 95, 100, 110, 260, 275, 285, 295)
kuiper_test(pidgeon_homing, alpha = .05)

# San Andreas Fault Data:
data(san_andreas)
data("nuvel1")
PoR <- subset(nuvel1, nuvel1$plate.rot == "na")
sa.por <- PoR_shmax(san_andreas, PoR, "right")
kuiper_test(sa.por$azi.PoR, alpha = .05)
```
line_azimuth

Extract azimuths of line segments

Description

Extract azimuths of line segments

Usage

line_azimuth(x)

lines_azimuths(x)

Arguments

x sf object of type "LINESTRING" or "MULTILINESTRING"

Details

It is recommended to perform line_azimuth() on single line objects, i.e. type "LINESTRING", instead of "MULTILINESTRING". This is because the azimuth of the last point of a line will be calculated to the first point of the next line otherwise. This will cause a warning message. For MULTILINESTRING objects, use lines_azimuths().

Value

sf object of type "POINT" with the columns and entries of the first row of x

Examples

data("plates")
subset(plates, pair == "af-eu") |> smoothr::densify() |> line_azimuth()

## Not run:
lines_azimuths(plates)

## End(Not run)
Description

Measure of spread around the circle. It should be noted that: If R=0, then the data is completely spread around the circle. If R=1, the data is completely concentrated on one point.

Usage

mean_resultant_length(x, w = NULL, na.rm = TRUE)

Arguments

x numeric vector. Values in degrees, for which the mean, median or standard deviation are required.
w (optional) Weights. A vector of positive numbers, of the same length as x.
na.rm logical value indicating whether NA values in x should be stripped before the computation proceeds.

Value

numeric.

References


Examples

# Example data from Davis (1986), pp. 316
finland_stria <- c(23, 27, 53, 58, 64, 83, 85, 88, 93, 99, 100, 105, 113,
                   113, 114, 117, 121, 123, 125, 126, 126, 127, 127, 128, 128, 129, 132,
                   132, 132, 134, 135, 137, 144, 145, 145, 146, 153, 155, 155, 155, 157, 163,
                   165, 171, 172, 179, 181, 186, 190, 212)
mean_resultant_length(finland_stria, w = NULL, na.rm = FALSE) # 0.800
model_shmax

*Theoretical Direction of Maximum Horizontal Stress in the geographical reference system.*

**Description**

Models the direction of maximum horizontal stress $\sigma_{H_{\text{max}}}$ along great circles, small circles, and loxodromes at a given point or points according to the relative plate motion in the geographical coordinate reference system.

**Usage**

```r
model_shmax(df, euler)
```

**Arguments**

- `df` data.frame containing the coordinates of the point(s) (lat, lon).
- `euler` "data.frame" or object of class "euler.pole" containing the geographical coordinates of the Euler pole

**Details**

$\sigma_{H_{\text{max}}}$ following *great circles* is the (initial) bearing between the given point and the pole of relative plate motion. $\sigma_{H_{\text{max}}}$ along *small circles*, clockwise, and counter-clockwise *loxodromes* is 90°, +45°, and 135° (-45°) to this great circle bearing, respectively.

**Value**

Data frame

- `ge` Azimuth of modeled $\sigma_{H_{\text{max}}}$ following great circles
- `sc` Small circles
- `ld.cw` Clockwise loxodromes
- `ld.ccw` Counter-clockwise loxodromes

**Author(s)**

Tobias Stephan

**References**


**See Also**

`deviation_shmax()` to compute the deviation of the modeled direction from the observed direction of $\sigma_{H_{\text{max}}}$. `PoR_shmax()` to calculate the azimuth of $\sigma_{H_{\text{max}}}$ in the pole of rotation reference system.
Examples

data("nuvel1")
# North America relative to Pacific plate:
euler <- subset(nuvel1, nuvel1$plate.rot == "na")

# the point where we want to model the SHmax direction:
point <- data.frame(lat = 45, lon = 20)

model_shmax(point, euler)

normalize_Q4

**Quaternion normalization**

Description

Quaternion normalization

Usage

normalize_Q4(q)

Arguments

q

quaternion

Value

object of class "quaternion"

norm_chisq

**Normalized Chi-Squared Test for Circular Data**

Description

A quantitative comparison between the predicted and observed directions of \(\sigma_{H_{\text{max}}}\) is obtained by the calculation of the average azimuth and by a normalized \(\chi^2\) test.

Usage

norm_chisq(obs, prd, unc)

Arguments

obs

Numeric vector containing the observed azimuth of \(\sigma_{H_{\text{max}}}\), same length as prd

prd

Numeric vector containing the modeled azimuths of \(\sigma_{H_{\text{max}}}\), i.e. the return object from model_shmax()

unc

Uncertainty of observed \(\sigma_{H_{\text{max}}}\), either a numeric vector or a number
The normalized $\chi^2$ test is

$$Norm\chi^2_i = \frac{\sum_{i=1}^{M} \left( \frac{\alpha_i - \alpha_{\text{predict}}}{\sigma_i} \right)^2}{\sum_{i=1}^{M} \left( \frac{90}{\sigma_i} \right)^2}$$

The value of the chi-squared test statistic is a number between 0 and 1 indicating the quality of the predicted $\sigma_{Hmax}$ directions. Low values ($\leq 0.15$) indicate good agreement, high values ($> 0.7$) indicate a systematic misfit between predicted and observed $\sigma_{Hmax}$ directions.

**Value**

Numeric vector

**References**


**Examples**

```r
data("nuvel1")
PoR <- subset(nuvel1, nuvel1$plate.rot == "na") # North America relative to Pacific plate
data(san_andreas)
point <- data.frame(lat = 45, lon = 20)
prd <- model_shmax(point, PoR)
norm_chisq(obs = c(50, 40, 42), prd$sc, unc = c(10, NA, 5))
data(san_andreas)
prd2 <- PoR_shmax(san_andreas, PoR, type = "right")
norm_chisq(obs = prd2$azi.PoR, 135, unc = san_andreas$unc)
```

**nuvel1**

*NUVEL-1 Global model of current plate motions*

**Description**

NNR-NUVEL-1 global model of current plate motions by DeMets et al. 1990

**Usage**

data('nuvel1')
nuvel1_plates

Format
An object of class `data.frame`

- `plate.name` The rotating plate
- `plate.rot` The abbreviation of the plate’s name
- `lat,lon` Coordinates of the Pole of Rotation
- `angle` The amount of rotation (angle in 1 Myr)
- `plate.fix` The anchored plate, i.e. `plate.rot` moves relative to `plate.fix`
- `source` Reference to underlying study

References

Examples
```r
data("nuvel1")
head("nuvel1")
```

---

nuvel1_plates  Plate Boundaries on the Earth

Description
Global set of present plate boundaries on the Earth based on NUVEL-1 model by DeMets et al. 1990

Usage
```r
data('nuvel1_plates')
```

Format
An object of class `sf`

References

Examples
```r
data("nuvel1_plates")
head("nuvel1_plates")
```
parse_wsm

**Numerical values to World Stress Map Quality Ranking**

**Description**

Assigns numeric values of the precision of each measurement to the categorical quality ranking of the World Stress Map (A, B, C, D).

**Usage**

`parse_wsm_quality(x)`

`quantise_wsm_quality(x)`

**Arguments**

`x` Either a string or a character vector of WSM quality ranking

**Value**

"integer" or vector of type "integer"

**References**


**Examples**

`parse_wsm_quality(c("A", "B", "C", "D", NA))`
`data("san_andreas")`
`parse_wsm_quality(san_andreas$quality)`

---

**pb2002**

*Global model of current plate motions*

**Description**

PB2002 global model of current plate motions by Bird 2003

**Usage**

`data('pb2002')`
plates

Format

An object of class data.frame

plate.name  The rotating plate
plate.rot   The abbreviation of the plate’s name
lat,lon     Coordinates of the Pole of Rotation
angle       The amount of rotation (angle in 1 Myr)
plate.fix   The anchored plate, i.e. plate.rot moves relative to plate.fix
source      Reference to underlying study

References


Examples

data("pb2002")
head("pb2002")

plates                  Plate Boundaries on the Earth

Description

Global set of present plate boundaries on the Earth based on PB2002 model by Bird (2003). Contains the plate boundary displacement types such as inward, outward, or tangentially displacement.

Usage

data('plates')

Format

An object of class sf

References


Examples

data("plates")
head("plates")
PoR2Geo_azimuth

Azimuth conversion from PoR to geographical coordinate reference system

Description
Conversion of PoR azimuths into geographical azimuths

Usage
PoR2Geo_azimuth(x, PoR)

Arguments
x data.frame containing the PoR equivalent azimuths (azi.PoR), and either the geographical coordinates of the point(s) or the PoR-equivalent coordinates.
PoR data.frame containing the geographical location of the Euler pole (lat, lon)

Value
numeric vector of transformed azimuths (in degrees)

References

See Also
PoR_shmax()

Examples
data("nuvel1")
# North America relative to Pacific plate:
PoR <- subset(nuvel1, nuvel1$plate.rot == "na")
data("san_andreas")
head(san_andreas$azi)
san_andreas$azi.PoR <- PoR_shmax(san_andreas, PoR)
res.geo <- PoR2Geo_azimuth(san_andreas, PoR)
head(res.geo)
**PoR_coordinates**

**Description**

Retrieve the PoR equivalent coordinates of an object

**Usage**

```r
PoR_coordinates(x, PoR)
```

**Arguments**

- `x` sf or data.frame containing lat and lon coordinates (lat, lon)
- `PoR` Pole of Rotation. "data.frame" or object of class "euler.pole" containing the geographical coordinates of the Euler pole

**Value**

data.frame with the PoR coordinates (lat.PoR, lon.PoR)

**Examples**

```r
data("nuvel1")
por <- subset(nuvel1, nuvel1$plate.rot == "na") # North America relative to Pacific plate
data("san_andreas")
san_andreas.por_sf <- PoR_coordinates(san_andreas, por)
head(san_andreas.por_sf)
san_andreas.por_df <- PoR_coordinates(sf::st_drop_geometry(san_andreas), por)
head(san_andreas.por_df)
```

---

**PoR_crs**

**PoR coordinate reference system**

**Description**

Create the reference system transformed in Euler pole coordinate

**Usage**

```r
PoR_crs(x)
```

**Arguments**

- `x` "data.frame" or "euler.pole" object containing the geographical coordinates of the Euler pole
Details

The PoR coordinate reference system is oblique transformation of the geographical coordinate system with the Euler pole coordinates being the translation factors.

Value

Object of class \texttt{crs}

See Also

\texttt{sf::st\_crs()}

Examples

\begin{verbatim}
data("nuvel1")
por <- subset(nuvel1, nuvel1$plate.rot == "na") # North America relative to Pacific plate
PoR_crs(por)
\end{verbatim}

---

Description

Plot data in PoR map

Usage

\begin{verbatim}
PoR_map(
x, PoR,
 pb = NULL,
 type = c("none", "in", "out", "right", "left"),
 deviation = FALSE,
 ... )
\end{verbatim}

Arguments

\begin{itemize}
\item \textit{x, pb} \hspace{1cm} \texttt{sf} objects of the data points and the plate boundary geometries in the geographical coordinate system
\item \textit{PoR} \hspace{1cm} Pole of Rotation. \texttt{"data.frame"} or object of class \texttt{"euler\_pole"} containing the geographical coordinates of the Pole of Rotation
\item \textit{type} \hspace{1cm} Character. Type of plate boundary (optional). Can be \texttt{"out"}, \texttt{"in"}, \texttt{"right"}, or \texttt{"left"} for outward, inward, right-lateral, or left-lateral moving plate boundaries, respectively. If \texttt{"none"} (the default), only the PoR-equivalent azimuth is returned.
\end{itemize}
\textbf{PoR\_shmax}  

\texttt{PoR\_shmax}  

\texttt{deviation} logical. Whether the data should be color-coded according to the deviation from the prediction, or according to the stress regime?  

\texttt{Value}  

\texttt{plot}  

\texttt{See Also}  

\texttt{PoR\_shmax()}, \texttt{axes()}, \texttt{tectonicr.colors()}  

\textbf{Examples}  

\begin{verbatim}  
data("nuvel1")  
na_pa <- subset(nuvel1, nuvel1$plate.rot == "na")  

data("plates")  
plate_boundary <- subset(plates, plates$pair == "na-pa")  

data("san_andreas")  
PoR_map(san_andreas, PoR = na_pa, pb = plate_boundary, type = "right", deviation = TRUE)  
\end{verbatim}  

\textbf{Description}  

Models the direction of maximum horizontal stress $\sigma_{Hmax}$ in the Euler pole (Pole of Rotation) coordinate reference system. When type of plate boundary is given, it also gives the deviation from the theoretically predicted azimuth of $\sigma_{Hmax}$, the deviation, and the normalized $\chi^2$ statistics.  

\textbf{Usage}  

\texttt{PoR\_shmax(df, PoR, type = c("none", "in", "out", "right", "left"))}  

\textbf{Arguments}  

\begin{itemize}  
\item \texttt{df} data.frame containing the coordinates of the point(s) (lat, lon), the direction of $\sigma_{Hmax}$ azi and its standard deviation unc (optional)  
\item \texttt{PoR} "data.frame" or object of class "euler.pole" containing the geographical coordinates of the Euler pole  
\item \texttt{type} Character. Type of plate boundary (optional). Can be "out", "in", "right", or "left" for outward, inward, right-lateral, or left-lateral moving plate boundaries, respectively. If "none" (the default), only the PoR-equivalent azimuth is returned.  
\end{itemize}
Details

The azimuth of $\sigma_{Hmax}$ in the pole of rotation reference system is approximate 0 (or 180), 45, 90, 135 degrees if the stress is sourced by an outward, sinistral, inward, or dextral moving plate boundary, respectively. Directions of $\sigma_{Hmax}$ with respect to the four plate boundary types.

Value

Either a numeric vector of the azimuths in the transformed coordinate system (in degrees), or a "data.frame" with:

- azi.PoR the transformed azimuths (in degrees),
- prd the predicted azimuths (in degrees),
- dev the deviation between the transformed and the predicted azimuth (in degrees),
- nchisq the Norm $\chi^2$ test statistic, and
- cdist the angular distance between the transformed and the predicted azimuth.

References


See Also

- model_shmax() to compute the theoretical direction of $\sigma_{Hmax}$ in the geographical reference system.
- deviation_shmax() to compute the deviation of the modeled direction from the observed direction of $\sigma_{Hmax}$.
- norm_chisq() to calculate the normalized $\chi^2$ statistics.
- circular_distance() to calculate the angular distance.

Examples

```r
data("nuvel1")
# North America relative to Pacific plate:
PoR <- subset(nuvel1, nuvel1$plate.rot == "na")
data("san_andreas")
res <- PoR_shmax(san_andreas, PoR, type = "right")
head(res)
```

PoR_stress2grid

Spatial interpolation of SHmax in PoR coordinate reference system

Description

The data is transformed into the PoR system before the interpolation. The interpolation grid is returned in geographical coordinates and azimuths.
PoR_stress2grid

Usage

PoR_stress2grid(
  x,
  PoR,
  grid = NULL,
  PoR_grid = TRUE,
  lon_range = NULL,
  lat_range = NULL,
  gridsize = 2.5,
  ...
)

Arguments

  x               sf object containing
  azi  SHmax in degree
  unc  Uncertainties of SHmax in degree
  type  Methods used for the determination of the orientation of SHmax
  PoR  Pole of Rotation. "data.frame" or object of class "euler.pole" containing
       the geographical coordinates of the Euler pole
  grid           (optional) Point object of class sf.
  PoR_grid      logical. Whether the grid should be generated based on the coordinate range in
                 the PoR (TRUE, the default) CRS or the geographical CRS (FALSE). Is ignored if
                 grid is specified.
  lon_range, lat_range
                 (optional) numeric vector specifying the minimum and maximum longitudes
                 and latitudes (are ignored if "grid" is specified).
  gridsize      Numeric. Target spacing of the regular grid in decimal degree. Default is 2.5 (is
                 ignored if grid is specified)
  ...           Arguments passed to stress2grid()

Details

Stress field and wavelength analysis in PoR system and back-transformed

Value

  sf object containing
  lon,lat  longitude and latitude in geographical CRS (in degrees)
  lon.PoR,lat.PoR  longitude and latitude in PoR CRS (in degrees)
  azi  geographical mean SHmax in degree
  azi.PoR  PoR mean SHmax in degree
  sd  Standard deviation of SHmax in degrees
  R  Search radius in km
  mdr  Mean distance of datapoints per search radius
  N  Number of data points in search radius
See Also

stress2grid(), compact_grid()

Examples

data("san_andreas")
data("nuvel1")
PoR <- subset(nuvel1, nuvel1$plate.rot == "na")
PoR_stress2grid(san_andreas, PoR)

por_transformation_df Conversion between spherical PoR to geographical coordinate system

Description

Transformation from spherical PoR to geographical coordinate system and vice versa

Usage

geographical_to_PoR(x, PoR)
PoR_to_geographical(x, PoR)

Arguments

x "data.frame" containing lat and lon coordinates of a point in the geographical CRS or the lat.PoR, lon.PoR of the point in the PoR CRS.
PoR Pole of Rotation. "data.frame" or object of class "euler.pole" containing the geographical coordinates of the Euler pole

Value

"data.frame" with the transformed coordinates (lat.PoR and lon.PoR for PoR CRS, or lat and lon for geographical CRS).

Examples

data("nuvel1")
por <- subset(nuvel1, nuvel1$plate.rot == "na") # North America relative to Pacific plate
data("san_andreas")
san_andreas.por <- geographical_to_PoR(san_andreas, por)
head(san_andreas.por)
head(PoR_to_geographical(san_andreas.por, por))
por_transformation_quat

Conversion between PoR to geographical coordinate system using quaternions

Description
Helper function for the transformation from PoR to geographical coordinate system or vice versa.

Usage
geographical_to_PoR_quat(x, PoR)
PoR_to_geographical_quat(x, PoR)

Arguments
x, PoR two-column vectors containing the lat and lon coordinates

Value
two-element numeric vector

Examples
ep.geo <- c(20, 33)
q.geo <- c(10, 45)
q.por <- geographical_to_PoR_quat(q.geo, ep.geo)
q.por
PoR_to_geographical_quat(q.por, ep.geo)

por_transformation_sf
Conversion between PoR to geographical coordinates of spatial data

Description
Transform spatial objects from PoR to geographical coordinate reference system and vice versa.

Usage
PoR_to_geographical_sf(x, PoR)
geographical_to_PoR_sf(x, PoR)
Arguments

- **x**  
  sf, SpatRast, or Raster* object of the data points in geographical or PoR coordinate system
- **PoR**  
  Pole of Rotation. "data.frame" or object of class "euler.pole" containing the geographical coordinates of the Euler pole

Details

The PoR coordinate reference system is oblique transformation of the geographical coordinate system with the Euler pole coordinates being the translation factors.

Value

sf or SpatRast object of the data points in the transformed geographical or PoR coordinate system

Examples

```r
data("nuvel1")
PoR <- subset(nuvel1, nuvel1$plate.rot == "na") # North America relative to Pacific plate
data("san_andreas")
san_andreas.por <- geographical_to_PoR_sf(san_andreas, PoR)
PoR_to_geographical_sf(san_andreas.por, PoR)
```

Description

The maximum error in the model’s predicted azimuth given the Pole of rotations uncertainty and distance of the data point to the pole.

Usage

`prd_err(dist_PoR, sigma_PoR = 1)`

Arguments

- **dist_PoR**  
  Distance to Euler pole (great circle distance, in degree)
- **sigma_PoR**  
  uncertainty of the position of the Pole of rotation (in degree).

Value

numeric vector. The maximum error for azimuths prediction (in degree)

References

See Also

PoR_shmax() and model_shmax() for the model’s prediction, and orthodrome() for great circle distances.

Examples

prd_err(67, 1)

<table>
<thead>
<tr>
<th>product_Q4</th>
<th>Product of quaternions</th>
</tr>
</thead>
</table>

Description

Helper function for multiplication of two quaternions. Concatenation of two rotations R1 followed by R2

Usage

product_Q4(q1, q2, normalize = FALSE)

Arguments

q1, q2  
two objects of class “quaternion”. first rotation R1 expressed by q1 followed by second rotation R2 expressed by q2

normalize  
logical. Whether a quaternion normalization should be applied (TRUE) or not (FALSE, the default).

Value

object of class “quaternion”

Note

Multiplication is not commutative.
projected_pb_strike

*Strike of the plate boundary projected on data point*

**Description**

The fault’s strike in the PoR CRS projected on the data point along the predicted stress trajectories.

**Usage**

```r
projected_pb_strike(x, PoR, pb, tangential = FALSE, ...)
```

**Arguments**

- `x`, `pb` sf objects of the data points and the plate boundary geometries in the geographical coordinate system
- `PoR` Pole of rotation. "data.frame" or object of class "euler.pole" containing the geographical coordinates of the Euler pole
- `tangential` Logical. Whether the plate boundary is a tangential boundary (TRUE) or an inward and outward boundary (FALSE, the default).
- `...` optional arguments passed to `smoothr::densify()`

**Details**

Useful to calculate the beta angle, i.e. the angle between SHmax direction (in PoR CRS!) and the fault’s strike (in PoR CRS). The beta angle is the same in geographical and PoR coordinates.

**Value**

Numeric vector of the strike direction of the plate boundary (in degree)

**Note**

The algorithm calculates the great circle bearing between line vertices. Since transform plate boundaries represent small circle lines in the PoR system, this great-circle azimuth is only an approximation of the true (small-circle) azimuth.

**Examples**

```r
data("nuvel1")
na_pa <- subset(nuvel1, nuvel1$plate.rot == "na")

data("plates")
plate_boundary <- subset(plates, plates$pair == "na-pa")

data("san_andreas")
res <- projected_pb_strike(
  x = san_andreas, PoR = na_pa, pb = plate_boundary, tangential = TRUE
)
```
head(res)
head(san_andreas$azi - res) # beta angle

---

**Q4_to_euler**

**Euler angle/axis from quaternion**

**Description**

Euler angle/axis from quaternion

**Usage**

Q4_to_euler(q)

**Arguments**

- `q`: object of class "quaternion"

**Value**

"euler.pole" object

---

**quick_plot**

**Plotting stress analysis results**

**Description**

Creates a set of plots including the azimuth as a function of the distance to the plate boundary, the Norm Chi-squared as a function of the distance to the plate boundary, the circular distance (and dispersion) a function of the distance to the plate boundary, and a rose diagram of the frequency distribution of the azimuths.

**Usage**

quick_plot(azi, distance, prd, unc = NULL, regime, width = 51)

**Arguments**

- `azi`: numeric. Azimuth of $\sigma_{Hmax}$
- `distance`: numeric. Distance to plate boundary
- `prd`: numeric. the predicted direction of $\sigma_{Hmax}$
- `unc`: numeric. Uncertainty of observed $\sigma_{Hmax}$, either a numeric vector or a number
- `regime`: character vector. The stress regime (following the classification of the World Stress Map)
- `width`: integer. window width (in number of observations) for moving average of the azimuths, circular dispersion, and Norm Chi-square statistics. If `NULL`, an optimal width will be estimated.
Details

Plot 1 shows the transformed azimuths as a function of the distance to the plate boundary. The red line indicates the rolling circular mean, stippled red lines indicate the 95% confidence interval about the mean.

Plot 2 shows the normalized $\chi^2$ statistics as a function of the distance to the plate boundary. The red line shows the rolling $\chi^2$ statistic.

Plot 3 shows the circular distance of the transformed azimuths to the predicted azimuth, as a function of the distance to the plate boundary. The red line shows the rolling circular dispersion about the prediction.

Plot 4 gives the rose diagram of the transformed azimuths.

Value

four R base plots

See Also

PoR_shmax(), distance_from_pb(), circular_mean(), circularDispersion(), confidence_angle(), norm_chisq(), weighted_rayleigh()

Examples

data("nuvel1")
na_pa <- subset(nuvel1, nuvel1$plate.rot == "na")
data("plates")
plate_boundary <- subset(plates, plates$pair == "na-pa")
data("san_andreas")
res <- PoR_shmax(san_andreas, na_pa, "right")
d <- distance_from_pb(san_andreas, na_pa, plate_boundary, tangential = TRUE)
quick_plot(res$azi.PoR, d, res$prd, san_andreas$unc, san_andreas$regime)

raster_transformation

Conversion between PoR to geographical coordinate reference system of raster data

Description

Helper function to transform raster data set from PoR to geographical coordinates

Usage

geographical_to_PoR_raster(x, PoR)

PoR_to_geographical_raster(x, PoR)
Arguments

<table>
<thead>
<tr>
<th>x</th>
<th>numeric vector. Values in degrees</th>
</tr>
</thead>
<tbody>
<tr>
<td>mu</td>
<td>(optional) The specified or known mean direction (in degrees) in alternative hypothesis</td>
</tr>
<tr>
<td>axial</td>
<td>logical. Whether the data are axial, i.e. $\pi$-periodical (TRUE, the default) or directional, i.e. $2\pi$-periodical (FALSE).</td>
</tr>
</tbody>
</table>

Details

$H_0$: angles are randomly distributed around the circle.

$H_1$: angles are from unimodal distribution with unknown mean direction and mean resultant length (when mu is NULL. Alternatively (when mu is specified), angles are uniformly distributed around a specified direction.

If statistic > p.value, the null hypothesis is rejected, i.e. the length of the mean resultant differs significantly from zero, and the angles are not randomly distributed.

Value

a list with the components:

statistic mean resultant length
p.value significance level of the test statistic
p.value2 modified significance level (Cordeiro and Ferrari, 1991)
Note

Although the Rayleigh test is consistent against (non-uniform) von Mises alternatives, it is not consistent against alternatives with $p = 0$ (in particular, distributions with antipodal symmetry, i.e. axial data). Tests of non-uniformity which are consistent against all alternatives include Kuiper’s test (`kuiper_test()`) and Watson’s $U^2$ test (`watson_test()`).

References


See Also

`mean_resultant_length()`, `circular_mean()`, `norm_chisq()`, `kuiper_test()`, `watson_test()`

Examples

```r
# Example data from Mardia and Jupp (2001), pp. 93
pidgeon_homing <- c(55, 60, 65, 95, 100, 110, 260, 275, 285, 295)
rayleigh_test(pidgeon_homing, axial = FALSE)

# Example data from Davis (1986), pp. 316
finland_stria <- c(
    23, 27, 53, 58, 64, 83, 85, 88, 93, 99, 100, 105, 113,
    113, 114, 117, 121, 123, 125, 126, 126, 127, 127, 128, 128, 129, 132,
    132, 134, 135, 137, 144, 145, 145, 146, 153, 155, 155, 157, 163,
    165, 171, 172, 179, 181, 186, 190, 212
)
rayleigh_test(finland_stria, axial = FALSE)
rayleigh_test(finland_stria, mu = 105, axial = FALSE)

# Example data from Mardia and Jupp (2001), pp. 99
atomic_weight <- c(
    rep(0, 12), rep(3.6, 1), rep(36, 6), rep(72, 1),
    rep(108, 2), rep(169.2, 1), rep(324, 1)
)
rayleigh_test(atomic_weight, 0, axial = FALSE)

# San Andreas Fault Data:
data(san_andreas)
rayleigh_test(san_andreas$azi)
data("nuvel1")
PoR <- subset(nuvel1, nuvel1$plate.rot == "na")
sa.por <- PoR_shmax(san_andreas, PoR, "right")
rayleigh_test(sa.por$azi.PoR, mu = 135)
```
relative_rotation

Relative rotation between two rotations

Description

Calculates the relative rotation between two rotations, i.e. the difference from rotation 1 to rotation 2.

Usage

relative_rotation(r1, r2)

Arguments

r1, r2  
Objects of class "euler.pole". First rotation is r1, followed rotation r2.

Value

list. Euler axes (geographical coordinates) and Euler angles (in degrees)

References


See Also

euler_pole() for class "euler.pole"

Examples

a <- euler_pole(90, 0, angle = 45)
b <- euler_pole(0, 0, 1, geo = FALSE, angle = -15)
relative_rotation(a, b)
relative_rotation(b, a)
Apply Rolling Functions using Circular Statistical Tests for Uniformity

Description

A generic function for applying a function to rolling margins of an array.

Usage

roll_normchisq(
  obs,
  prd,
  unc = NULL,
  width = NULL,
  by.column = FALSE,
  partial = TRUE,
  fill = NA,
  ...
)

roll_rayleigh(
  obs,
  prd,
  unc = NULL,
  width = NULL,
  by.column = FALSE,
  partial = TRUE,
  fill = NA,
  ...
)

roll_dispersion(
  x,
  y,
  w = NULL,
  w.y = NULL,
  width = NULL,
  by.column = FALSE,
  partial = TRUE,
  fill = NA,
  ...
)

roll_confidence(
  x,
  conf.level = 0.95,
  w = NULL,
axial = TRUE,
width = NULL,
by.column = FALSE,
partial = TRUE,
fill = NA,
...
)

roll_dispersion_CI(
  x,
  y,
  w = NULL,
  w.y = NULL,
  R,
  conf.level = 0.95,
  width = NULL,
  by.column = FALSE,
  partial = TRUE,
  fill = NA,
  ...
)

roll_dispersion_sde(
  x,
  y,
  w = NULL,
  w.y = NULL,
  R,
  conf.level = 0.95,
  width = NULL,
  by.column = FALSE,
  partial = TRUE,
  fill = NA,
  ...
)

Arguments

obs  Numeric vector containing the observed azimuth of $\sigma_{H_{max}}$, same length as prd
prd  Numeric vector containing the modeled azimuths of $\sigma_{H_{max}}$, i.e. the return object from model_shmax()
unc Uncertainty of observed $\sigma_{H_{max}}$, either a numeric vector or a number
width integer specifying the window width (in numbers of observations) which is aligned to the original sample according to the align argument. If NULL, an optimal width is estimated.
by.column logical. If TRUE, FUN is applied to each column separately.
partial logical or numeric. If FALSE then FUN is only applied when all indexes of the rolling window are within the observed time range. If TRUE (default), then the
subset of indexes that are in range are passed to FUN. A numeric argument to partial can be used to determine the minimal window size for partial computations. See below for more details.

fill

da three-component vector or list (recycled otherwise) providing filling values at the left/within/to the right of the data range. See the fill argument of `zoo::na.fill()` for details

... optional arguments passed to `zoo::rollapply()`

x, y numeric. Directions in degrees

w, w.y (optional) Weights of x and y, respectively. A vector of positive numbers and of the same length as x.

conf.level Level of confidence: \((1 - \alpha\%)/100\). (0.95 by default).

axial logical. Whether the data are axial, i.e. \(\pi\)-periodical (TRUE, the default) or directional, i.e. \(2\pi\)-periodical (FALSE).

R The number of bootstrap replicates.

Value

numeric vector with the test statistic of the rolling test. roll_displacement_CI returns a 2-column matrix with the lower and the upper confidence limits

Note

If the rolling functions are applied to values that are a function of distance it is recommended to sort the values first.

Examples

data("plates")
plate_boundary <- subset(plates, plates$pair == "na-pa")
data("san_andreas")
PoR <- subset(nuvel1, nuvel1$plate.rot == "na")
distance <- distance_from_pb(
  x = san_andreas,
  PoR = PoR,
  pb = plate_boundary,
  tangential = TRUE
)
dat <- san_andreas[order(distance), ]
dat.PoR <- PoR_shmax(san_andreas, PoR, "right")
roll_normchisq(dat.PoR$azi.PoR, 135, dat$unc)
roll_rayleigh(dat.PoR$azi.PoR, prd = 135, unc = dat$unc)
roll_displacement(dat.PoR$azi.PoR, y = 135, w = 1 / dat$unc)
roll_confidence(dat.PoR$azi.PoR, w = 1 / dat$unc)
roll_displacement_CI(dat.PoR$azi.PoR, y = 135, w = 1 / dat$unc, R = 10)
Apply Rolling Functions using Circular Statistics

Description

A generic function for applying a function to rolling margins of an array along an additional value.

Usage

```r
distroll_circstats(
  x,
  distance,
  FUN,
  width = NULL,
  min_n = 2,
  align = c("right", "center", "left"),
  w = NULL,
  sort = TRUE,
  ...
)

distroll_confidence(
  x,
  distance,
  w = NULL,
  width = NULL,
  min_n = 2,
  align = c("right", "center", "left"),
  sort = TRUE,
  ...
)

distroll Dispersion( 
  x,
  y,
  w = NULL,
  w.y = NULL,
  distance,
  width = NULL,
  min_n = 2,
  align = c("right", "center", "left"),
  sort = TRUE,
  ...
)

distroll Dispersion SDE( 
  x,
  ...
)
```
Arguments

- **x**, **y** vectors of numeric values in degrees. length(y) is either 1 or length(x)
- **distance** numeric. the independent variable along the values in x are sorted, e.g. the plate boundary distances
- **FUN** the function to be applied
- **width** numeric. the range across distance on which FUN should be applied on x. If NULL, then width is a number that separates the distances in 10 equal groups.
- **min_n** integer. The minimum values that should be considered in FUN (2 by default), otherwise NA.
- **align** specifies whether the index of the result should be left- or right-aligned or centered (default) compared to the rolling window of observations. This argument is only used if width represents widths.
- **w** numeric. the weighting for x
- **sort** logical. Should the values be sorted after distance prior to applying the function (TRUE by default).
- **...** optional arguments to FUN
- **w.y** numeric. the weighting for y

Value
two-column vectors of (sorted) x and the rolled statistics along distance.

Examples
data("plates")
plate_boundary <- subset(plates, plates$pair == "na-pa")
data("san_andreas")
PoR <- subset(nuvel1, nuvel1$plate.rot == "na")
san_andreas$distance <- distance_from_pb(
  x = san_andreas,
  PoR = PoR,
  pb = plate_boundary,
  tangential = TRUE
)
dat <- san_andreas |> cbind(PoR_shmax(san_andreas, PoR, "right"))
distroll_circstats(dat$azi.PoR, distance = dat$distance, w = 1 / dat$unc, FUN = circular_mean)
distroll_confidence(dat$azi.PoR, distance = dat$distance, w = 1 / dat$unc)
distroll_dispersion(dat$azi.PoR, y = 135, distance = dat$distance, w = 1 / dat$unc)
distroll_dispersion_sde(dat$azi.PoR, y = 135, distance = dat$distance, w = 1 / dat$unc, R = 100)

---

**Description**

A generic function for applying a function to rolling margins of an array.

**Usage**

```r
roll_circstats(
  x,
  w = NULL,
  FUN,
  axial = TRUE,
  na.rm = TRUE,
  width = NULL,
  by.column = FALSE,
  partial = TRUE,
  fill = NA,
  ...
)
```

**Arguments**

- **x**: numeric vector. Values in degrees.
- **w**: (optional) Weights. A vector of positive numbers and of the same length as x.
- **FUN**: the function to be applied
- **axial**: logical. Whether the data are axial, i.e. pi-periodical (TRUE, the default) or directional, i.e. 2π-periodical (FALSE).
- **na.rm**: logical value indicating whether NA values in x should be stripped before the computation proceeds.
- **width**: integer specifying the window width (in numbers of observations) which is aligned to the original sample according to the align argument. If NULL, an optimal width is calculated.
- **by.column**: logical. If TRUE, FUN is applied to each column separately.
- **partial**: logical or numeric. If FALSE then FUN is only applied when all indexes of the rolling window are within the observed time range. If TRUE (default), then the subset of indexes that are in range are passed to FUN. A numeric argument to partial can be used to determine the minimal window size for partial computations. See below for more details.
fill

a three-component vector or list (recycled otherwise) providing filling values at the left/within/to the right of the data range. See the fill argument of \texttt{zoo::na.fill()} for details.

... optional arguments passed to \texttt{zoo::rollapply()}

Value

numeric vector with the results of the rolling function.

Note

If the rolling statistics are applied to values that are a function of distance it is recommended to sort the values first.

Examples

data("plates")
plate_boundary <- subset(plates, plates$pair == "na-pa")
data("san_andreas")
PoR <- subset(nuvel1, nuvel1$plate.rot == "na")
distance <- distance_from_pb(
  x = san_andreas,
  PoR = PoR,
  pb = plate_boundary,
  tangential = TRUE
)
dat <- san_andreas[order(distance), ]
roll_circstats(dat$azi, w = 1 / dat$unc, circular_mean, width = 51)

---

**rose**  

*Rose Diagram*

Description

Plots a rose diagram (rose of directions), the analogue of a histogram or density plot for angular data.

Usage

```
rose(x,
     weights = NULL,
     binwidth = NULL,
     bins = NULL,
     axial = TRUE,
     equal_area = TRUE,
     clockwise = TRUE,
     muci = TRUE,
)```

---
```r
round_binwidth = 0,
mtext = "N",
main = NULL,
sub = NULL,
at = seq(0, 360 - 45, 45),
col = "grey",
dots = FALSE,
dot_pch = 1,
dot_cex = 1,
dot_col = "grey",
...
)

Arguments

x Data to be plotted. A numeric vector containing angles (in degrees).
weights Optional vector of numeric weights associated with x.
binwidth The width of the bins (in degrees).
bins number of arcs to partition the circle width. Overridden by binwidth.
axial Logical. Whether data are uniaxial (axial=FALSE) or biaxial (TRUE, the default).
equal_area Logical. Whether the radii of the bins are proportional to the frequencies (equal_area=FALSE, i.e. equal-angle) or proportional to the square-root of the frequencies (equal_area=TRUE, the default).
clockwise Logical. Whether angles increase in the clockwise direction (clockwise=TRUE, the default) or anti-clockwise, counter-clockwise direction (FALSE).
muci logical. Whether the mean and its 95% CI are added to the plot or not.
round_binwidth integer. Number of decimal places of bin width (0 by default).
mtext character. String to be drawn at the top margin of the plot ("N" by default)
main, sub Character string specifying the title and subtitle of the plot. If sub = NULL, it will show the bin width.
at Optional vector of angles at which tick marks should be plotted. Set at=numeric(0) to suppress tick marks.
col fill color of bins
dots logical. Whether a circular dot plot should be added (FALSE is the default).
dot_cex, dot_pch, dot_col Plotting arguments for circular dot plot
...
Additional arguments passed to spatstat.explore::rose().

Value

A window (class "owin") containing the plotted region.
Note

If bins and binwidth are NULL, an optimal bin width will be calculated using Scott (1979):

\[ w_b = \frac{R}{n^{\frac{1}{3}}} \]

with \( n \) being the length of \( x \), and the range \( R \) being either 180 or 360 degree for axial or directional data, respectively.

If "axial" == TRUE, the binwidth is adjusted to guarantee symmetrical fans.

Examples

```r
x <- rvm(100, mean = 90, k = 5)
rose(x, axial = FALSE, border = TRUE)

data("san_andreas")
rose(san_andreas$azi, dots = TRUE, main = "dot plot")
rose(san_andreas$azi, weights = 1 / san_andreas$unc, main = "weighted")
```

---

rose_geom | Lines and fans in rose diagram

Description

Lines and fans in rose diagram

Usage

```r
rose_line(x, radius = 1, axial = TRUE, add = TRUE, ...)
rose_fan(x, d, radius = 1, axial = TRUE, add = TRUE, ...)
```

Arguments

- **x**: angles in degrees
- **radius**: of the rose diagram
- **axial**: Logical. Whether x are uniaxial (axial=FALSE) or biaxial (TRUE, the default).
- **add**: logical. Add to existing plot?
- **...**: optional arguments passed to `graphics::segments()` or `graphics::polygon()`
- **d**: width of a fan (in degrees)

Value

No return value, called for side effects
**Examples**

```r
a <- c(0, 10, 45)
radius <- c(.7, 1, .2)
lwd <- c(2, 1, .75)
col <- c(1, 2, 3)
rose_line(c(0, 10, 45), radius = radius, axial = FALSE, add = FALSE, lwd = lwd, col = col)
```

---

**rose_stats**

*Show Average Direction and Spread in Rose Diagram*

**Description**

Adds the average direction (and its spread) to an existing rose diagram.

**Usage**

```r
rose_stats(
  x,
  weights = NULL,
  axial = TRUE,
  avg = c("mean", "median"),
  spread = c("CI", "sd", "IQR"),
  avg.col = "#85112AFF",
  avg.lty = 2,
  avg.lwd = 1.5,
  spread.col = ggplot2::alpha("#85112AFF", 0.2),
  spread.border = FALSE,
  spread.lty = NULL,
  spread.lwd = NULL,
  add = TRUE,
  ...
)
```

**Arguments**

- **x**: Data to be plotted. A numeric vector containing angles (in degrees).
- **weights**: Optional vector of numeric weights associated with x.
- **axial**: Logical. Whether data are uniaxial (axial=FALSE) or biaxial (TRUE, the default).
- **avg**: character. The average estimate for x. Either the circular mean ("mean", the default) or the circular Quasi Median ("median")
- **spread**: character. The measure of spread to be plotted as a fan. Either the 95% confidence interval ("CI", the default), the circular standard deviation ("sd"), or the Quasi interquartile range on the circle ("IQR"). NULL if no fan should be drawn.
- **avg.col**: color for the average line
- **avg.lty**: line type of the average line
avg.lwd  line width of the average line  
spread.col  color of the spread fan  
spread.border  logical. Whether to draw a border of the fan or not.  
spread.lty  line type of the spread fan’s border  
spread.lwd  line width of the spread fan’s border  
add  logical.  
...  optional arguments to rose_baseplot() if add is FALSE.

Value

No return value, called for side effects

See Also

rose() for plotting the rose diagram, and circular_mean(), circular_median(), confidence_interval(), circular_sd(), circular_IQR() for statistical parameters.

Examples

data("san_andreas")
rose(san_andreas$azi, weights = 1 / san_andreas$unc, muci = FALSE)
rose_stats(san_andreas$azi, weights = 1 / san_andreas$unc, avg = "median", spread = "IQR")

---

**rotation_Q4**  
*Rotation of a vector by a quaternion*

Description

Rotation of a vector by a quaternion

Usage

`rotation_Q4(q, p)`

Arguments

- `q`  
object of class "quaternion"
- `p`  
three-column vector (Cartesian coordinates) of unit length

Value
	hree-column vector (Cartesian coordinates) of unit length
**san_andreas**

*Example stress data for tangentially displaced plate boundary*

**Description**


**Usage**

```r
data('san_andreas')
```

**Format**

An object of class "sf"

**Source**

[https://www.world-stress-map.org/](https://www.world-stress-map.org/)

**References**


**Examples**

```r
data("san_andreas")
head(san_andreas)
```

---

**spec_atan**

*Quadrant-specific inverse of the tangent*

**Description**

Returns the quadrant specific inverse of the tangent

**Usage**

```r
atan2_spec(x, y)
atan2d_spec(x, y)
```
spherical_angle

Arguments

x, y  dividend and divisor that comprise the sum of sines and cosines, respectively.

Value

numeric.

References


spherical_angle  Angle along great circle on spherical surface

Description

Smallest angle between two points on the surface of a sphere, measured along the surface of the sphere

Usage

orthodrome(lat1, lon1, lat2, lon2)

haversine(lat1, lon1, lat2, lon2)

vincenty(lat1, lon1, lat2, lon2)

Arguments

lat1, lat2  numeric vector. latitudes of point 1 and 2 (in radians)

lon1, lon2  numeric vector. longitudes of point 1 and 2 (in radians)

Details

"orthodrome"  based on the spherical law of cosines

"haversine"  uses haversine formula that is optimized for 64-bit floating-point numbers

"vincenty"  uses Vincenty formula for an ellipsoid with equal major and minor axes

Value

numeric. angle in radians
stress2grid

References

- [http://www.movable-type.co.uk/scripts/latlong.html](http://www.movable-type.co.uk/scripts/latlong.html)
- [http://www.edwilliams.org/avform147.htm](http://www.edwilliams.org/avform147.htm)

Examples

```r
berlin <- c(52.52, 13.41)
calgary <- c(51.04, -114.072)
orthodrome(berlin[1], berlin[2], calgary[1], calgary[2])
haversine(berlin[1], berlin[2], calgary[1], calgary[2])
v Vincenty(berlin[1], berlin[2], calgary[1], calgary[2])
```

stress2grid

*Spatial interpolation of SHmax*

Description

Stress field interpolation and wavelength analysis using a kernel (weighted) mean/median and standard deviation/IQR of stress data

Usage

```r
stress2grid(
x,
stat = c("mean", "median"),
grid = NULL,
lon_range = NULL,
lat_range = NULL,
gridsize = 2.5,
min_data = 3,
threshold = 25,
arte_thres = 200,
method_weighting = FALSE,
quality_weighting = TRUE,
dist_weight = c("inverse", "linear", "none"),
idp = 1,
qp = 1,
mp = 1,
dist_threshold = 0.1,
R_range = seq(50, 1000, 50),
...
)
```
Arguments

- **x**: sf object containing
  - **azi**: SHmax in degree
  - **unc** (optional): Uncertainties of SHmax in degree
  - **type** (optional): Methods used for the determination of the direction of SHmax

- **stat**: Whether the direction of interpolated SHmax is based on the circular mean and standard deviation ("mean", the default) or the circular median and interquartile range ("median")

- **grid**: (optional) Point object of class sf.

- **lon_range, lat_range**: (optional) Numeric vector specifying the minimum and maximum longitudes and latitudes (ignored if grid is specified).

- **gridsize**: Numeric. Target spacing of the regular grid in decimal degree. Default is 2.5. (is ignored if grid is specified)

- **min_data**: Integer. Minimum number of data per bin. Default is 3

- **threshold**: Numeric. Threshold for deviation of direction. Default is 25

- **arte_thres**: Numeric. Maximum distance (in km) of the grid point to the next data point. Default is 200

- **method_weighting**: Logical. If a method weighting should be applied: Default is FALSE. If FALSE, overwrites mp.

- **quality_weighting**: Logical. If a quality weighting should be applied: Default is TRUE. If FALSE, overwrites qp.

- **dist_weight**: Distance weighting method which should be used. One of "none", "linear", or "inverse" (the default).

- **idp, qp, mp**: Numeric. The weighting power of inverse distance, quality and method. Default is 1. The higher the value, the more weight it will put. When set to 0, no weighting is applied. idp is only effective if inverse distance weighting (dist_weight="inverse") is applied.

- **dist_threshold**: Numeric. Distance weight to prevent overweight of data nearby (0 to 1). Default is 0.1

- **R_range**: Numeric value or vector specifying the kernel half-width(s), i.e. the search radius (in km). Default is seq(50, 1000, 50)

Details

This is a modified version of the MATLAB script "stress2grid"

Value

- sf object containing
stress_analysis

lon, lat  longitude and latitude in degrees
azi  Mean SHmax in degree
sd  Standard deviation of SHmax in degrees
R  Search radius in km
mdr  Mean distance of datapoints per search radius
N  Number of data points in search radius

Source

https://github.com/MorZieg/Stress2Grid

References


See Also

dist_greatcircle(), PoR_stress2grid(), compact_grid(), circular_mean(), circular_median(), circular_sd()

Examples

data("san_andreas")
stress2grid(san_andreas, stat = "median")

stress_analysis  Quick analysis of a stress data set

Description

Returns the converted azimuths, distances to the plate boundary, statistics of the model, and some plots.

Usage

stress_analysis(
  x,
  PoR,
  type = c("none", "in", "out", "right", "left"),
  pb,
  plot = TRUE,
  ...
)

Arguments

x  data.frame or sf object containing the coordinates of the point(s) (lat, lon), the direction of $\sigma_{Hmax}$ azi and its standard deviation unc (optional)

PoR  Pole of Rotation. data.frame or object of class "euler.pole" containing the geographical coordinates of the Euler pole

type  Character. Type of plate boundary (optional). Can be "out", "in", "right", or "left" for outward, inward, right-lateral, or left-lateral moving plate boundaries, respectively. If "none" (the default), only the PoR-equivalent azimuth is returned.

pb  (optional) sf object of the plate boundary geometries in the geographical coordinate system

plot  (logical). Whether to produce a plot additional to output.

...  optional arguments to distance_from_pb()

Value

list containing the following values:

results  data.frame showing the the coordinate and azimuth conversions (lat.PoR, lon.PoR, and azi.PoR), the predicted azimuths (prd), deviation angle from predicted (dev), circular distance (cdist), misfit to predicted stress direction (nchisq) and, if given, distance to tested plate boundary (distance)

stats  array with circular (weighted) mean, circular standard deviation, circular variance, circular dispersion, the 95% confidence angle, and the normalized Chi-squared test statistic

test  list containing the test results of the (weighted) Rayleigh test against the uniform distribution about the predicted orientation.

See Also

PoR_shmax(), distance_from_pb(), norm_chisq(), quick_plot()

Examples

data("nuvel1")
na_pa <- subset(nuvel1, nuvel1$plate.rot == "na")
data("plates")
plate_boundary <- subset(plates, plates$pair == "na-pa")
data("san_andreas")
stress_analysis(san_andreas, na_pa, type = "right", plate_boundary, plot = TRUE)
stress_colors

**Color palette for stress regime**

**Description**

Color palette for stress regime

**Usage**

stress_colors()

**Value**

function

**Examples**

stress_colors()

---

stress_paths

**Theoretical Plate Tectonic Stress Paths**

**Description**

Construct $\sigma_{Hmax}$ lines that are following small circles, great circles, or loxodromes of an Euler pole for the relative plate motion.

**Usage**

eulerpole_paths(x, type = c("sc", "gc", "ld"), n = 10, angle, cw)
eulerpole_smallcircles(x, n = 10)
eulerpole_greatcircles(x, n = 10)
eulerpole_loxodromes(x, n = 10, angle = 45, cw)

**Arguments**

- **x** Either an object of class "euler.pole" or "data.frame" containing coordinates of Euler pole in lat, lon, and rotation angle (optional).
- **type** Character string specifying the type of curves to export. Either "sm" for small circles (default), "gc" for great circles, or "ld" for loxodromes.
- **n** Number of equally spaced curves; $n = 10$ by default (angular distance between curves: $180 / n$)
angle Direction of loxodromes; angle = 45 by default.
cw logical. Sense of loxodromes: TRUE for clockwise loxodromes (left-lateral displaced plate boundaries). FALSE for counterclockwise loxodromes (right-lateral displaced plate boundaries).

Details

Maximum horizontal stress can be aligned to three types of curves related to relative plate motion:

**Small circles** Lines that have a constant distance to the Euler pole. If x contains angle, output additionally gives absolute velocity on small circle (degree/Myr -> km/Myr).

**Great circles** Paths of the shortest distance between the Euler pole and its antipodal position.

**Loxodromes** Lines of constant bearing, i.e. curves cutting small circles at a constant angle.

Value

sf object

Author(s)

Tobias Stephan

Examples

data("nuvel1")
por <- subset(nuvel1, nuvel1$plate.rot == "na") # North America relative to # Pacific plate
eulerpole_smallcircles(por)
eulerpole_greatcircles(por)
eulerpole_loxodromes(x = por, angle = 45, n = 10, cw = FALSE)
eulerpole_loxodromes(x = por, angle = 30, cw = TRUE)
eulerpole_smallcircles(data.frame(lat = 30, lon = 10))

superimposed_shmax SHmax direction resulting from multiple plate boundaries

Description

Calculates a $\sigma_{H_{max}}$ direction at given coordinates, sourced by multiple plate boundaries. This first-order approximation is the circular mean of the superimposed theoretical directions, weighted by the rotation rates of the underlying PoRs.

Usage

superimposed_shmax(df, PoRs, types, absolute = TRUE, PoR_weighting = NULL)
tectonicr.colors

Arguments

- **df**: data.frame containing the coordinates of the point(s) (lat, lon), and the direction of $\sigma_{Hmax}$ azi (in degrees)
- **PoRs**: multirow data.frame or "euler.pole" object that must contain lat, lon and angle
- **types**: character vector with length equal to number of rows in PoRs. Type of plate boundary. Must be "out", "in", "right", or "left" for outward, inward, right-lateral, or left-lateral moving plate boundaries, respectively.
- **absolute**: logical. Whether the resultant azimuth should be weighted using the absolute rotation at the points or the angular rotation of the PoRs.
- **PoR_weighting**: (optional) numeric vector with length equal to number of rows in PoRs. Extra weightings for the used PoRs.

Value

numeric. Resultant azimuth in degrees and geographical CRS

See Also

model.shmax()

Examples

```r
data(san_andreas)
data(nuvel1)
pors <- subset(nuvel1, plate.rot %in% c("eu", "na"))
superimposed.shmax(san_andreas, pors, types = c("in", "right"), PoR_weighting = c(2, 1))
```

tectonicr.colors

Colors for input variables

Description

assigns colors to continuous or categorical values for plotting

Usage

tectonicr.colors(
  x,
  n = 10,
  pal = NULL,
  categorical = FALSE,
  na.value = "grey",
  ...
)
Arguments

- **x**: values for color assignment
- **n**: integer. number of colors for continuous colors (i.e. `categorical = FALSE`).
- **pal**: either a named vector specifying the colors for categorical values, or a color function. If NULL, default colors are `RColorBrewer::brewer.pal()` (categorical = TRUE) and `viridis::viridis()` (categorical = FALSE).
- **categorical**: logical.
- **na.value**: color for NA values (categorical).
- **...**: optional arguments passed to palette function

Value

named color vector

Examples

```r
val1 <- c("N", "S", "T", "T", NA)
tectonicr.colors(val1, categorical = TRUE)
tectonicr.colors(val1, pal = stress_colors(), categorical = TRUE)

val2 <- runif(10)
tectonicr.colors(val2, n = 5)
```

---

tibet **Example stress data for inward-moving displaced plate boundary**

Description

Subset of the World Stress Map (WSM) compilation of information on the crustal present-day stress field (Version 1.1. 2019). Subset contains stress data of the Himalaya and Tibetan plateau

Usage

```r
data('tibet')
```

Format

An object of class "sf"

Source

https://www.world-stress-map.org/
References

Examples

data("tibet")
head(tibet)

---

vcross

Vector cross product

Description
Vector or cross product

Usage

vcross(x, y)

Arguments

x, y numeric vectors of length 3

Value
numeric vector of length 3

Examples

vcross(c(1, 2, 3), c(4, 5, 6))

---

vonmises

The von Mises Distribution

Description
Density, distribution function, and random generation for the circular normal distribution with mean and kappa.
Usage

```
rvm(n, mean, kappa)
dvm(theta, mean, kappa)
pvm(theta, mean, kappa)
```

Arguments

- **n**: number of observations in degrees
- **mean**: mean in degrees
- **kappa**: concentration parameter
- **theta**: angular value in degrees

Value

numeric vector.

Examples

```
x <- rvm(100, mean = 90, k = 100)
dvm(x, mean = 90, k = 100)
```

---

**watson_test**

*Watson’s \( U^2 \) Test of Circular Uniformity*

Description

Watson’s test statistic is a rotation-invariant Cramer - von Mises test

Usage

```
watson_test(  
x,  
alpha = 0,  
dist = c("uniform", "vonmises"),  
axial = TRUE,  
mu = NULL  
)
```
**weighted_rayleigh**

Weighted Goodness-of-fit Test for Circular Data

**Description**

Weighted version of the Rayleigh test (or V0-test) for uniformity against a distribution with a priori expected von Mises concentration.

**Arguments**

- **x**
  - numeric vector. Values in degrees
- **alpha**
  - Significance level of the test. Valid levels are 0.01, 0.05, and 0.1. This argument may be omitted (NULL, the default), in which case, a range for the p-value will be returned.
- **dist**
  - Distribution to test for. The default, "uniform", is the uniform distribution. "vonmises" tests the von Mises distribution.
- **axial**
  - logical. Whether the data are axial, i.e. \( \pi \)-periodical (TRUE, the default) or circular, i.e. 2\( \pi \)-periodical (FALSE).
- **mu**
  - (optional) The specified mean direction (in degrees) in alternative hypothesis

**Details**

If statistic \( > p\_value \), the null hypothesis is rejected. If not, randomness (uniform distribution) cannot be excluded.

**Value**

list containing the test statistic statistic and the significance level p.value.

**References**


**Examples**

# Example data from Mardia and Jupp (2001), pp. 93
pidgeon_homing <- c(55, 60, 65, 95, 100, 110, 260, 275, 285, 295)
watson_test(pidgeon_homing, alpha = .05)

# San Andreas Fault Data:
data(san_andreas)
data("nuvel1")
PoR <- subset(nuvel1, nuvel1$plate.rot == "na")
sa.por <- PoR_shmax(san_andreas, PoR, "right")
watson_test(sa.por$azi.PoR, alpha = .05)
watson_test(sa.por$azi.PoR, alpha = .05, dist = "vonmises")
Usage

weighted_rayleigh(x, mu = NULL, w = NULL, axial = TRUE)

Arguments

x numeric vector. Values in degrees
mu The *a priori* expected direction (in degrees) for the alternative hypothesis.
w numeric vector weights of length length(x). If NULL, the non-weighted Rayleigh test is performed.
axial logical. Whether the data are axial, i.e. \(\pi\)-periodical (TRUE, the default) or directional, i.e. \(2\pi\)-periodical (FALSE).

Details

The Null hypothesis is uniformity (randomness). The alternative is a distribution with a specified mean direction (\(\text{prd}\)). If statistic > p.value, the null hypothesis is rejected. If not, the alternative cannot be excluded.

Value

a list with the components:

- **statistic** Test statistic
- **p.value** significance level of the test statistic

See Also

rayleigh_test()

Examples

```r
# Load data
data("cpm_models")
data(san_andreas)
PoR <- equivalent_rotation(subset(cpm_models, model == "NNR-MORVEL56"), "na", "pa")
sa.por <- PoR_shmax(san_andreas, PoR, "right")
data("iceland")
PoR.ice <- equivalent_rotation(subset(cpm_models, model == "NNR-MORVEL56"), "eu", "na")
ic.e.por <- PoR_shmax(iceland, PoR.ice, "out")
data("tibet")
PoR.tib <- equivalent_rotation(subset(cpm_models, model == "NNR-MORVEL56"), "eu", "in")
tibet.por <- PoR_shmax(tibet, PoR.tib, "in")

# GOF test:
weighted_rayleigh(tibet.por$azi.PoR, mu = 90, w = 1 / tibet$unc)
weighted_rayleigh(ic.e.por$azi.PoR, mu = 0, w = 1 / iceland$unc)
weighted_rayleigh(sa.por$azi.PoR, mu = 135, w = 1 / san_andreas$unc)
```
Index

* datasets
  cpm_models, 15
  iceland, 28
  nuvel1, 36
  nuvel1_plates, 37
  pb2002, 38
  plates, 39
  san_andreas, 67
  tibet, 76

abs_vel, 3
angle-conversion, 4
angle_vectors, 5
atan2_spec(spec_atan), 67
atan2d_spec(spec_atan), 67
axes, 5
axes(), 43

boot::boot(), 8

cartesian_to_geographical coordinates, 13
cartesian_to_geographical(), 14
cartesian_to_spherical(coordinates2), 14
cartesian_to_spherical(), 13
circle_stats, 6
circular_dispersion(dispersion), 18
circular_dispersion(), 9, 30, 52
circular_dispersion_boost, 8
circular_distance(dispersion), 18
circular_distance(), 44
circular_IQR(circle_stats), 6
circular_IQR(), 66
circular_mean(circle_stats), 6
circular_mean(), 9, 19, 52, 54, 66, 71
circular_median(circle_stats), 6
circular_median(), 66, 71
circular_quantiles(circle_stats), 6
circular_sd(circle_stats), 6
circular_sd(), 66, 71
circular_sd_error, 9
circular_sd_error(), 11
circular_var(circle_stats), 6
circular_var(), 19
compact_grid, 10
compact_grid(), 46, 71
colour(), 3

deg2rad(angle-conversion), 4
deviation_norm, 16
deviation_shmax, 17
deviation_shmax(), 34, 44
dispersion, 18
dist_greatcircle, 21
dist_greatcircle(), 30, 70, 71
distance_from_pb, 20
distance_from_pb(), 26, 52, 72
distance_mod, 21
distroll_circstats(rolling_test_dist), 59
distroll_confidence(rolling_test_dist), 59
distroll_distance(rolling_test_dist), 59
distroll_distance_sde(rolling_test_dist), 59
dvm(vonmises), 77

earth_radius, 22
earth_radius(), 4

eightyone
equivalent_rotation, 23
equivalent_rotation(), 27
est.kappa, 24
euler_pole, 24
euler_pole(), 55
euler_to_Q4, 25
eulerpole_greatcircles (stress_paths), 73
eulerpole_loxodromes (stress_paths), 73
eulerpole_paths (stress_paths), 73

geographical_to_cartesian
(coordinates), 13
geographical_to_cartesian(), 14
geographical_to_PoR
(por_transformation_df), 46
geographical_to_PoR_quat
(por_transformation_quat), 47
good just

get_distance, 26
good just

haversine (spherical_angle), 68
haversine(), 22

iceland, 28
is.euler, 28
is.Q4, 29

kernel_dispersion, 29
kernel_dispersion(), 10
kuiper_test, 31
kuiper_test(), 54

latitude_modulo (coordinate_mod), 15
line_azimuth, 32

lines_azimuths (line_azimuth), 32
longitude_modulo (coordinate_mod), 15

mean_resultant_length, 33
mean_resultant_length(), 9, 11, 54
model_shmax, 34
model_shmax(), 18, 44, 49, 75

norm_chisq, 35
norm_chisq(), 30, 44, 52, 54, 72
normalize_Q4, 35
nuvel1, 36
nuvel1_plates, 37

orthodrome (spherical_angle), 68
orthodrome(), 22, 49

parse_wsm, 38
parse_wsm_quality (parse_wsm), 38
pb2002, 38
plates, 39
PoR2Geo_azimuth, 40
PoR_coordinates, 41
PoR_crs, 41
PoR_map, 42
PoR_shmax, 43
PoR_shmax(), 34, 40, 43, 49, 52, 72
PoR_stress2grid, 44
PoR_stress2grid(), 10, 71
PoR_to_geographical
(por_transformation_df), 46
PoR_to_geographical_quat
(por_transformation_quat), 47
PoR_to_geographical_raster
(raster_transformation), 52
PoR_to_geographical_sf
(por_transformation_sf), 47

product_Q4, 49
projected_pb_strike, 50
projected_pb_strike(), 27
pvm (vonmises), 77

Q4_to_euler, 51
quantise_wsm_quality (parse_wsm), 38
quaternion (relative_rotation), 55
INDEX

quick_plot, 51
quick_plot(), 72

rad2deg (angle-conversion), 4
raster_transformation, 52
rayleigh_test, 53
rayleigh_test(), 30, 80
relative_rotation, 55
relative_rotation(), 23
roll_circstats, 61
roll_confidence (rolling_test), 56
roll_dispersion (rolling_test), 56
roll_dispersion_CI (rolling_test), 56
roll_dispersion_sde (rolling_test), 56
roll_normchisq (rolling_test), 56
roll_rayleigh (rolling_test), 56
rolling_test, 56
rolling_test_dist, 59
rose, 62
rose(), 66
rose_fan (rose_geom), 64
rose_geom, 64
rose_line (rose_geom), 64
rose_stats, 65
rotation (relative_rotation), 55
rotation_Q4, 66
rvm (vonmises), 77

san_andreas, 67
sf::st_crs(), 42
smoothr::densify(), 20, 50
spatstat.explore::rose(), 63
spec_atan, 67
spherical_angle, 68
spherical_to_cartesian (coordinates2), 14
spherical_to_cartesian(), 13
spherical_to_geographical
(coordinates2), 14
stress2grid, 69
stress2grid(), 10, 45, 46
stress_analysis, 71
stress_colors, 73
stress_paths, 73
superimposed_shmax, 74

tectonicr.colors, 75
tectonicr.colors(), 43
tibet, 76

vcross, 77
vincenty (spherical_angle), 68
vincenty(), 22
vonmises, 77

watson_test, 78
watson_test(), 54
weighted_rayleigh, 79
weighted_rayleigh(), 52

zoo::na.fill(), 58, 62
zoo::rollapply(), 58, 62