

Package ‘trajr’

October 14, 2022

Type Package

Title Animal Trajectory Analysis

Version 1.4.0

Date 2020-12-17

Description A toolbox to assist with statistical analysis of 2-dimensional animal trajectories. It provides simple access to algorithms for calculating and assessing a variety of characteristics such as speed and acceleration, as well as multiple measures of straightness or tortuosity. McLean & Skowron Volponi (2018) <[doi:10.1111/eth.12739](https://doi.org/10.1111/eth.12739)>.

License MIT + file LICENSE

URL <https://github.com/JimMcL/trajr>

BugReports <https://github.com/JimMcL/trajr/issues>

Encoding UTF-8

LazyData true

Imports signal, utils, stats, graphics, plotrix, grDevices

RoxygenNote 7.1.1

Suggests knitr, rmarkdown, testthat, BH, readr, tcltk, sp, MASS

VignetteBuilder knitr

BuildVignettes true

NeedsCompilation no

Author Jim McLean [aut, cre]

Maintainer Jim McLean <jim_mclean@optusnet.com.au>

Depends R (>= 3.5.0)

Repository CRAN

Date/Publication 2020-12-17 13:10:02 UTC

R topics documented:

ElapsedTimeProgressBarFn	3
lines.Trajectory	3
plot.TrajDirectionAutocorrelations	4
plot.Trajectory	5
plot.TrajSpeedIntervals	6
points.Trajectory	7
TrajAcceleration	7
TrajAngles	8
TrajConvertTime	9
TrajDAMinMax	10
TrajDerivatives	11
TrajDirectionalChange	12
TrajDirectionAutocorrelations	13
TrajDistance	14
TrajDuration	14
TrajEmax	15
TrajExpectedSquareDisplacement	16
TrajFractalDimension	17
TrajFractalDimensionValues	18
TrajFromCoords	19
TrajGenerate	21
TrajGetFPS	23
TrajGetNCoords	23
TrajGetTimeUnits	24
TrajGetUnits	24
TrajInPolygon	25
TrajLength	26
TrajLogSequence	26
TrajMeanVectorOfTurningAngles	27
TrajMeanVelocity	28
TrajMerge	28
trajr	29
TrajRediscretize	30
TrajResampleTime	31
TrajReverse	32
TrajRotate	32
TrajsBuild	33
TrajScale	35
TrajSinuosity	36
TrajSinuosity2	37
TrajsMergeStats	38
TrajSmoothSG	39
TrajSpeedIntervals	40
TrajSplit	41
TrajSplitAtFirstCrossing	42
TrajsStatsReplaceNAs	43

<i>ElapsedTimeProgressBarFn</i>	3
TrajsStepLengths	44
TrajStepLengths	45
TrajStraightness	45
TrajTranslate	46
TrajVelocity	47
Index	49

ElapsedTimeProgressBarFn
A general purpose progress bar that reports elapsed time rather than number of items

Description

A general purpose progress bar that reports elapsed time rather than number of items

Usage

`ElapsedTimeProgressBarFn(numItems, reportFn)`

Arguments

<code>numItems</code>	Number of items to be processed
<code>reportFn</code>	A function used to report changing progress

Value

A function which should be called for each item as it is processed.

`lines.Trajectory` *Add Trajectory lines to a plot*

Description

The lines method for Trajectory objects.

Usage

```
## S3 method for class 'Trajectory'
lines(
  x,
  draw.start.pt = TRUE,
  start.pt.cex = 0.8,
  start.pt.pch = 16,
  start.pt.col = "black",
  turning.angles = NULL,
  ...
)
```

Arguments

x	An object of class "Trajectory", the trajectory to be plotted.
draw.start.pt	If TRUE, draws a dot at the start point of the trajectory.
start.pt.cex	Scale to apply when drawing the start point dot.
start.pt.pch	Pch (i.e. plot character, symbol or shape) to apply when drawing the start point dot.
start.pt.col	Colour to apply when drawing the start point dot.
turning.angles	If random or directed, draws step turning angles. directed assumes errors are relative to the first recorded step angle. random assumes errors are relative to the previous step.
...	Additional arguments are passed to lines .

plot.TrajDirectionAutocorrelations

Plot method for direction autocorrelation

Description

The plot method for TrajDirectionAutocorrelations objects. Plots the direction autocorrelation function as returned by a call to `link{TrajDirectionAutocorrelations}`, with an optional dot at the first local minimum.

Usage

```
## S3 method for class 'TrajDirectionAutocorrelations'
plot(
  x,
  firstMinWindowSize = 10,
  type = "l",
  ylab = expression("C(" * Delta * s * ")"),
  xlab = expression(Delta * s),
  ...
)
```

Arguments

x	Trajectory to be plotted.
firstMinWindowSize	If not NULL, specifies a window size used to calculate the first local minimum, which is then plotted as a point.
type, xlab, ylab	Defaults for plotting.
...	Additional arguments passed to plot .

plot.Trajectory *Plot method for trajectories*

Description

The plot method for Trajectory objects.

Usage

```
## S3 method for class 'Trajectory'
plot(
  x,
  add = FALSE,
  draw.start.pt = TRUE,
  start.pt.cex = 0.8,
  start.pt.pch = 16,
  start.pt.col = "black",
  turning.angles = NULL,
  xlim = grDevices::extendrange(x$x),
  ylim = grDevices::extendrange(x$y),
  xlab = ifelse(is.null(TrajGetUnits(x)), "x", sprintf("x (%s)", TrajGetUnits(x))),
  ylab = ifelse(is.null(TrajGetUnits(x)), "y", sprintf("y (%s)", TrajGetUnits(x))),
  asp = 1,
  ...
)
```

Arguments

<code>x</code>	An object of class "Trajectory", the trajectory to be plotted.
<code>add</code>	If TRUE, the trajectory is added to the current plot.
<code>draw.start.pt</code>	If TRUE, draws a dot at the start point of the trajectory.
<code>start.pt.cex</code>	Scale to apply when drawing the start point dot.
<code>start.pt.pch</code>	Pch (i.e. plot character, symbol or shape) to apply when drawing the start point dot.
<code>start.pt.col</code>	Colour to apply when drawing the start point dot.
<code>turning.angles</code>	If random or directed, draws step turning angles. <code>directed</code> assumes errors are relative to the first recorded step angle. <code>random</code> assumes errors are relative to the previous step.
<code>xlim, ylim, xlab, ylab, asp</code>	plotting parameters with useful defaults.
<code>...</code>	Additional arguments are passed to plot .

See Also

[TrajFromCoords](#)

Examples

```
set.seed(42)
trj <- TrajGenerate(angularErrorSd = 1.3)
plot(trj)
```

```
plot.TrajSpeedIntervals
```

Plot method for trajectory speed intervals

Description

Plots speed over time, with intervals of fast and/or slow speed highlighted.

Usage

```
## S3 method for class 'TrajSpeedIntervals'
plot(
  x,
  slowerThanColour = "red",
  fasterThanColour = "green",
  highlightColor = "#0000FF1E",
  xlab = sprintf("Time (%s)", TrajGetTimeUnits(attr(x, "trajectory"))),
  ylab = sprintf("Speed (%s/%s)", TrajGetUnits(attr(x, "trajectory")),
    TrajGetTimeUnits(attr(x, "trajectory"))),
  type = "l",
  ...
)
```

Arguments

x An object of class "SpeedIntervals", as created by [TrajSpeedIntervals](#).

slowerThanColour, fasterThanColour The colour of the horizontal line plotted at the "slower than" or "faster than" speed. Specify NULL to prevent the line from being plotted.

highlightColor Colour of the highlight rectangles.

xlab, ylab, type Plotting parameters with useful defaults.

... Additional arguments are passed to [plot](#).

See Also

[TrajSpeedIntervals](#)

points.Trajectory *Add Trajectory points to a plot*

Description

The points method for Trajectory objects.

Usage

```
## S3 method for class 'Trajectory'  
points(x, draw.start.pt = TRUE, turning.angles = NULL, ...)
```

Arguments

`x` An object of class "Trajectory", the trajectory to be plotted.
`draw.start.pt` If TRUE, draws a dot at the start point of the trajectory.
`turning.angles` If random or directed, draws step turning angles. directed assumes errors are relative to the first recorded step angle. random assumes errors are relative to the previous step.
`...` Additional arguments are passed to [points](#).

TrajAcceleration *Approximates the acceleration of a trajectory*

Description

Returns an approximation of the acceleration of a trajectory at each point using the second-order central finite differences.

Usage

```
TrajAcceleration(trj)
```

Arguments

`trj` Trajectory whose acceleration is to be calculated.

Details

'trajr' trajectories, which consist of straight line displacements between sampled locations, do not contain enough information to correctly derive velocity or acceleration. Since we have to assume a constant velocity at each step, the first derivative is discontinuous. Acceleration, therefore, is zero during each step and infinite at each change of velocity. The approximation implemented by this function assumes that acceleration occurs over a period of time: half the duration of the previous step plus half the duration of the next step.

Value

Vector of complex numbers. The modulus ($\text{Mod}(a)$) is the magnitude of the acceleration at each point, and the argument ($\text{Arg}(a)$) is the direction of the acceleration. The vector has an attribute, `trj`, with the trajectory as its value. The first and last values will always be NA, since acceleration cannot be estimated for those points.

See Also

[TrajVelocity](#) for calculating velocity, [TrajResampleTime](#) and [TrajRediscretize](#) to resample a trajectory to fixed time or length steps.

Examples

```
# A function to plot acceleration as arrows (scaled in length)
AccArrows <- function(acc, scale = .001, trj = attr(acc, "trj"), ...) {
  graphics::arrows(trj$x, trj$y, trj$x + Re(acc) * scale, trj$y + Im(acc) * scale, ...)
}

# Generate and plot a random trajectory
set.seed(101)
trj <- TrajGenerate(30)
plot(trj)

# Calculate acceleration
acc <- TrajAcceleration(trj)
# Plot acceleration as red arrows at each point. They need to be scaled down to
# fit in the plot, and the arrowhead lengths need to be shortened to look good
AccArrows(acc, scale = .001, col = "red", length = .1)
```

TrajAngles

Turning angles of a Trajectory

Description

Calculates the step angles (in radians) of each segment, either relative to the previous segment or relative to the specified compass direction.

Usage

```
TrajAngles(trj, lag = 1, compass.direction = NULL)
```

Arguments

<code>trj</code>	the trajectory whose whose angles are to be calculated.
<code>lag</code>	Angles between every lag'th segment are calculated. Only applies to non-directed walks, i.e. <code>compass.direction</code> is NULL.
<code>compass.direction</code>	If not NULL, step angles are calculated relative to this angle (in radians), otherwise they are calculated relative to the previous step angle.

Details

Note that since turning angles are circular quantities, i.e. $360^\circ == 0^\circ$, it is incorrect to treat them as linear quantities. In particular, do not calculate arithmetic means or standard deviations of turning angles. See Batschelet, (1981) for a detailed explanation and techniques for dealing with circular quantities.

Value

Step angles in radians, normalised so that $-\pi < \text{angle} \leq \pi$.

References

Batschelet, E. (1981). Circular statistics in biology. ACADEMIC PRESS, 111 FIFTH AVE., NEW YORK, NY 10003, 1981, 388.

See Also

[TrajStepLengths](#), [TrajMeanVectorOfTurningAngles](#)

TrajConvertTime	<i>Converts a delimited time string to a numeric value</i>
-----------------	--

Description

Time values may be imported in a format which is not immediately usable by 'trajr'. This function converts times that are specified as a number of delimited fields to a single numeric value. The default parameter values handle a value with 4 colon-separated values, which are hours, minutes, seconds and milliseconds, eg: "0:01:04:108" represents 1 minute, 4 seconds and 108 milliseconds, or 64.108 seconds.

Usage

```
TrajConvertTime(time, sep = ":", factors = c(60 * 60, 60, 1, 0.001))
```

Arguments

time	A character string containing the time value to be converted.
sep	Field separator.
factors	Vector of numeric factors to be applied to each field, in the order they occur within 'time'. The default assumes 4 fields containing numeric hours, minutes, seconds and milliseconds.

Details

Note that the base R `strptime` can be used to convert time values in more complex date/time formats, but it does not handle millisecond fields.

Value

‘time’ converted to a numeric value.

See Also

[strptime](#)

Examples

```
time <- c("0:00:00:029", "0:01:00:216", "0:02:01:062", "1:00:02:195", "1:06:03:949", "1:42:04:087")
seconds <- TrajConvertTime(time)
```

TrajDAMinMax

First direction autocorrelation minimum/maximum

Description

Determines the coordinates of the first local minimum/maximum of C in the direction autocorrelation function of a trajectory as returned by [TrajDirectionAutocorrelations](#). The end point is excluded from consideration as a minimum, similarly the start point will not be returned as a maximum. If the trajectory does not oscillate in direction, there will not be a local minimum/maximum, and NULL is returned.

Usage

```
TrajDAFindFirstMinimum(corr, windowSize = 10)
```

```
TrajDAFindFirstMaximum(corr, windowSize = 10)
```

Arguments

corr	A TrajDirectionAutocorrelations object, i.e. the direction autocorrelation of a trajectory.
windowSize	Size of window used to define what constitutes a local minimum/maximum.

Value

Numeric vector with 2 values, deltaS and C, or NULL if there is no local minimum/maximum.

See Also

[TrajDirectionAutocorrelations](#)

Examples

```
set.seed(42)
trj <- TrajGenerate(600, angularErrorSd = 1)
smoothed <- TrajSmoothSG(trj, 3, 11)

# Resample to fixed path length
resampled <- TrajRediscretize(smoothed, 1)
# Calculate direction autocorrelation for resampled trajectory
corr <- TrajDirectionAutocorrelations(resampled, 100)
# Extract first local minimum from autocorrelation
minPt <- TrajDAFindFirstMinimum(corr, 20)

# Plot the autocorrelation function
plot(corr, type = 'l')
# Plot a red dot with a black outline at the first minimum
points(minPt["deltaS"], minPt["C"], pch = 16, col = "red", lwd = 2)
points(minPt["deltaS"], minPt["C"], col = "black", lwd = 2)
```

TrajDerivatives

Calculates trajectory speed and change of speed

Description

Calculates speed and change of speed along a trajectory over time. These are the first and second order derivatives of distance travelled over time. Noisy trajectories should be smoothed before being passed to this function, as noise is effectively amplified when taking derivatives.

Usage

```
TrajDerivatives(trj)
```

Arguments

`trj` Trajectory whose speed and change in speed is to be calculated.

Details

The value returned as acceleration is *not* technically acceleration. In mechanics, acceleration is a vector. This value is a scalar quantity: change of speed, which is sometimes known informally as acceleration. This value corresponds to the acceleration in a 1-dimensional trajectory, with the sign indicating the direction of acceleration relative to the current direction of velocity. See [TrajAcceleration](#) for an approximation of (vector) acceleration, and [TrajVelocity](#) for an approximation of velocity.

Value

A list with components:

speed	numeric vector, speed between each pair of trajectory points, i.e. the speed of each step.
speedTimes	numeric vector, times corresponding to values in speed, i.e. the time from the start of the trajectory to the end of each step.
acceleration	numeric vector, change in speed between steps. Despite the name, this is not acceleration as defined by mechanics.
accelerationTimes	numeric vector, time from the start of the trajectory to the end of the second step in each pair.

See Also

[TrajSpeedIntervals](#) for analysing intervals of low or high speed within the trajectory. [TrajSmoothSG](#) for smoothing a trajectory. [TrajAcceleration](#) for calculating acceleration, and [TrajVelocity](#) for calculating velocity.

TrajDirectionalChange *Directional change (DC)*

Description

Calculates the time variation of directional change (DC) of a trajectory *sensu* Kitamura & Imafuku (2015). Directional change is defined as the angular change (in degrees) between any two points in the trajectory, divided by the time difference between the two points.

Usage

```
TrajDirectionalChange(trj, nFrames = 1)
```

Arguments

trj	Track to calculate DC for.
nFrames	Frame delta to process: if 1, every frame is processed, if 2, every 2nd frame is processed, and so on. Default is 1.

Details

This function returns the DC for each pair of consecutive points. Kitamura & Imafuku (2015) used the mean and the standard deviation of DC for portions of trajectories as index values of nonlinearity and irregularity respectively.

Value

The directional change (DC) in degrees between every pair of consecutive points in the trajectory, i.e. the returned vector will have length $(nrow(trj) - 1)$.

References

Kitamura, T., & Imafuku, M. (2015). Behavioural mimicry in flight path of Batesian intraspecific polymorphic butterfly *Papilio polytes*. *Proceedings of the Royal Society B: Biological Sciences*, 282(1809). doi:10.1098/rspb.2015.0483

Examples

```
set.seed(42)
trj <- TrajGenerate()
SD = mean(TrajDirectionalChange(trj))
SDDC = sd(TrajDirectionalChange(trj))
```

TrajDirectionAutocorrelations

Direction autocorrelation

Description

Calculates the autocorrelation of the track for Δs ranging from 1 to `deltaSMax`, based on Shamble et al. (2017). `trj` must have a constant step length (see [TrajRediscretize](#)) i.e. all segments in the trajectory must be the same length. `deltaS` is specified in number of segments. Call [TrajDAFindFirstMinimum](#) to locate the first local minimum which may be used to characterise directional periodicity in a trajectory (note that the first local minimum may not exist).

Usage

```
TrajDirectionAutocorrelations(trj, deltaSMax = round(nrow(trj)/4))
```

Arguments

<code>trj</code>	The trajectory to calculate the directional autocorrelations for.
<code>deltaSMax</code>	Maximum delta s to calculate, default is 1/4 the number of segments in the trajectory.

Value

A data frame with class `TrajDirectionAutocorrelations` and 2 columns, `deltaS` and `C`. Plotting this object displays a graph of the direction autocorrelation function, optionally with the location of the first local minimum marked

References

Shamble, P. S., Hoy, R. R., Cohen, I., & Beatus, T. (2017). Walking like an ant: a quantitative and experimental approach to understanding locomotor mimicry in the jumping spider *Myrmarachne formicaria*. *Proceedings of the Royal Society B: Biological Sciences*, 284(1858). doi:10.1098/rspb.2017.0308

See Also

[TrajDAFindFirstMinimum](#), [plot.TrajDirectionAutocorrelations](#)

TrajDistance	<i>Trajectory distance</i>
--------------	----------------------------

Description

Calculates the distance between the start and end of a trajectory (or a portion of a trajectory). Also called the diffusion distance, net distance, displacement, or bee-line from start to finish.

Usage

```
TrajDistance(trj, startIndex = 1, endIndex = nrow(trj))
```

Arguments

trj	Trajectory whose distance is to be calculated.
startIndex	Index of the starting point.
endIndex	Index of the ending point.

Value

Numeric distance from the start to the end of the trajectory.

TrajDuration	<i>Trajectory duration</i>
--------------	----------------------------

Description

Calculates the temporal duration of a trajectory (or a portion of a trajectory).

Usage

```
TrajDuration(trj, startIndex = 1, endIndex = nrow(trj))
```

Arguments

trj	Trajectory whose duration is to be calculated.
startIndex	Index of the starting point.
endIndex	Index of the ending point.

Value

Numeric duration of the trajectory, in time units.

See Also

[TrajGetTimeUnits](#)

TrajEmax	<i>Trajectory straightness index, E-max</i>
----------	---

Description

E_{max}, the maximum expected displacement, is a single-valued measure of straightness defined by (Cheung, Zhang, Stricker, & Srinivasan, 2007). E_{max-a} is a dimensionless, scale-independent measure of the maximum possible expected displacement. E_{max-b} is E_{max-a} * mean step length, and gives the maximum possible expected displacement in spatial units. Values closer to 0 are more sinuous, while larger values (approaching infinity) are straighter.

Usage

```
TrajEmax(trj, eMaxB = FALSE, compass.direction = NULL)
```

Arguments

trj	Trajectory to be analysed.
eMaxB	If TRUE, calculates and returns E _{max-b} , otherwise returns E _{max-a} .
compass.direction	if not NULL, turning angles are calculated for a directed walk, assuming the specified compass direction (in radians). Otherwise, a random walk is assumed.

Value

E_{max} (-a or -b) for trj.

References

Cheung, A., Zhang, S., Stricker, C., & Srinivasan, M. V. (2007). Animal navigation: the difficulty of moving in a straight line. *Biological Cybernetics*, 97(1), 47-61. doi:10.1007/s00422-007-0158-0

TrajExpectedSquareDisplacement

Trajectory expected square displacement

Description

Calculates the expected square displacement for a trajectory assuming it is a correlated random walk, using the formula in Kareiva & Shigesada, (1983).

Usage

```
TrajExpectedSquareDisplacement(  
  trj,  
  n = nrow(trj),  
  eqn1 = TRUE,  
  compass.direction = NULL  
)
```

Arguments

trj	A Trajectory.
n	Number of steps to calculate.
eqn1	If TRUE, calculate using equation 1, otherwise using equation 2. Equation 2 applies when the mean of turning angles is 0, i.e. turns are unbiased.
compass.direction	If not NULL, step angles are calculated relative to this angle (in radians), otherwise they are calculated relative to the previous step angle.

Details

Note that Cheung, Zhang, Stricker, and Srinivasan (2007) define an alternative formulation for expected maximum displacement, Emax (see [TrajEmax](#)).

References

Cheung, A., Zhang, S., Stricker, C., & Srinivasan, M. V. (2007). Animal navigation: the difficulty of moving in a straight line. *Biological Cybernetics*, 97(1), 47-61. doi:10.1007/s00422-007-0158-0

Kareiva, P. M., & Shigesada, N. (1983). Analyzing insect movement as a correlated random walk. *Oecologia*, 56(2), 234-238. doi:10.1007/bf00379695

See Also

[TrajEmax](#)

Examples

```

set.seed(1)
# A random walk
trj <- TrajGenerate(200)
smoothed <- TrajSmoothSG(trj)

# Calculate actual squared displacement at all points along the trajectory
sd2 <- sapply(2:nrow(smoothed), function(n) TrajDistance(smoothed, 1, n) ^ 2)
# Calculate expected squared displacement
ed2_1 <- sapply(2:nrow(smoothed), function(n) TrajExpectedSquareDisplacement(smoothed, n, TRUE))
ed2_2 <- sapply(2:nrow(smoothed), function(n) TrajExpectedSquareDisplacement(smoothed, n, FALSE))

# Plot expected against actual. According to Kareiva & Shigesada, (1983), if actual
# (approximately) matches expected, the trajectory is probably a correlated random walk
par(mar = c(5, 5, 0.1, 0.1) + .1)
plot(2:nrow(smoothed), sd2, type = 'l', pch = 16, cex = .2, lwd = 2,
     xlab = 'Number of consecutive moves',
     ylab = expression('Squared displacement, ' * R[n]^2))
lines(2:nrow(smoothed), ed2_1, col = "grey", lwd = 2)
lines(2:nrow(smoothed), ed2_2, col = "pink", lwd = 2)

legend("bottomright",
      c(expression("Actual displacement"^2),
        expression("Expected displacement"^2 * " (eqn 1)"),
        expression("Expected displacement"^2 * " (eqn 2)")),
      col = c('black', 'grey', 'pink'), lwd = 2,
      inset = c(0.01, 0.02))

```

TrajFractalDimension *Fractal dimension of a trajectory*

Description

Calculates the fractal dimension (D) of a trajectory using the 'dividers' method (Sugihara & May, 1990). By default, overestimation of D is compensated for as recommended by Nams (2006), by walking the dividers backwards and forwards, and by estimating the remaining path length at the end of the last step.

Usage

```
TrajFractalDimension(trj, stepSizes, adjustD = TRUE, dMean = TRUE)
```

Arguments

trj	Trajectory to calculate fractal dimension for.
stepSizes	Vector of step sizes (aka divider sizes) used to calculate path lengths.
adjustD	If TRUE, path length is adjusted for truncation error (Nams, 2006).

dMean If TRUE, the fractal dimension is calculated starting from the beginning of the trajectory, then re-calculated starting from the end and moving backwards. The value returned is the mean of the two fractal dimensions (Nams, 2006).

Details

Fractal dimension may be meaningless for animal trajectories as they may not be true fractal curves - see Benhamou (2004) and Turchin (1996), although it may be useful for studies involving differences in behaviour at different spatial scales (Nams, 2006).

You can test whether a trajectory is a fractal curve for a range of step sizes using the [TrajFractalDimensionValues](#) function. The example code in its documentation demonstrates how to plot path length for a range of step sizes. If the plotted points lie along straight line, then the trajectory is a fractal curve for that range of step sizes. However, typical trajectories result in a curve rather than a straight line.

If you decide to use fractal dimension despite the warnings of Benhamou (2004) and Turchin (1996), try to select a biologically meaningful range of step sizes (and be prepared to justify your choice). If comparing fractal dimensions across trajectories, be consistent in your choice of step sizes.

Value

The fractal dimension of the trajectory for the given step sizes.

References

- Benhamou, S. (2004). How to reliably estimate the tortuosity of an animal's path. *Journal of Theoretical Biology*, 229(2), 209-220. doi:10.1016/j.jtbi.2004.03.016
- Nams, V. O. (2006). Improving Accuracy and Precision in Estimating Fractal Dimension of Animal movement paths. *Acta Biotheoretica*, 54(1), 1-11. doi:10.1007/s10441-006-5954-8
- Sugihara, G., & M. May, R. (1990). Applications of fractals in ecology. *Trends in Ecology & Evolution*, 5(3), 79-86. doi:10.1016/0169-5347(90)90235-6
- Turchin, P. (1996). Fractal Analyses of Animal Movement: A Critique. *Ecology*, 77(7), 2086-2090. doi:10.2307/2265702

See Also

[TrajLogSequence](#) to create a logarithmically spaced sequence, [TrajFractalDimensionValues](#) for the function used internally to calculate a range of path lengths for different step sizes, [TrajEmax](#) and [TrajSinuosity2](#) for some alternate measures of trajectory tortuosity.

TrajFractalDimensionValues

Fractal dimension calculation

Description

Calculates path length ($L(\delta)$) for a range of step sizes (δ). For a fractal (i.e. scale independent) curve, $\log(L(\delta))$ grows linearly as $\log(\delta)$ grows smaller. In other words, if the points returned by this function lie on a straight line in a log-log plot, trj is a fractal curve.

Usage

```
TrajFractalDimensionValues(trj, stepSizes, adjustD = TRUE)
```

Arguments

trj	Trajectory to calculate fractal dimension for.
stepSizes	Vector of step sizes used to calculate path lengths.
adjustD	If TRUE, path length is adjusted to reduce truncation error (Nams, 2006).

Value

Data frame with columns stepsize (δ) and pathlength ($(L(\delta))$).

References

Nams, V. O. (2006). Improving Accuracy and Precision in Estimating Fractal Dimension of Animal movement paths. *Acta Biotheoretica*, 54(1), 1-11. doi:10.1007/s10441-006-5954-8

See Also

[TrajFractalDimension](#) for fractal dimension calculation.

Examples

```
set.seed(42)
trj <- TrajGenerate()
muL <- mean(TrajStepLengths(trj))
# Use 20 step sizes from 1/2 mean step length to 5 * mean step length.
# For real use, biologically meaningful step sizes should be used.
stepSizes <- TrajLogSequence(0.5 * muL, 5 * muL, 20)
plot(TrajFractalDimensionValues(trj, stepSizes), log = "xy", pch = 16, cex = .5)
```

TrajFromCoords

Create a Trajectory Object

Description

TrajFromCoords creates a new trajectory object from a set of 2-dimensional cartesian coordinates, times and some metadata. The coordinates are sometimes referred to as "relocations".

Usage

```
TrajFromCoords(
  track,
  xCol = 1,
  yCol = 2,
  timeCol = NULL,
  fps = 50,
  spatialUnits = "m",
  timeUnits = "s"
)
```

Arguments

track	data frame containing cartesian coordinates and optionally times for the points in the trajectory.
xCol	Name or index of the x column in track (default 1).
yCol	Name or index of the y column in track (default 2).
timeCol	optional name or index of the column which contains coordinate times.
fps	Frames per second - used to calculate relative coordinate times if track does not contain a time column. Time intervals between coordinate are assumed to be constant through the entire track.
spatialUnits	Abbreviation for the x and y units.
timeUnits	Abbreviation for the units that time is recorded in.

Details

If timeCol is specified, track[, timeCol] is expected to contain the time (in some numeric units) of each coordinate. Otherwise, times are calculated for each point as (coord - 1) / fps where coord is the index of the point; in other words, sampling at constant time intervals is assumed. Time values require conversion if they are not numeric. It may be possible to use 'strptime' for this purpose, or [TrajConvertTime](#) can be used to convert multiple field time values.

x and y must be square units. Longitude and latitude are not suitable for use as x and y values, since in general, 1° lat != 1° lon. To create a trajectory from positions in latitude and longitude, it is first necessary to transform the positions to a suitable spatial projection such as UTM (possibly by using spTransform from the rgdal package).

Value

An object with class "Trajectory", which is a data.frame with the following components:

x	X coordinates of trajectory points.
y	Y coordinates of trajectory points.
time	Time (in timeUnits) for each point. if timeCol is specified, values are track[, timeCol], otherwise values are calculated from fps.
displacementTime	Relative frame/observation times, with frame/observation 1 at time 0.

polar	Coordinates represented as complex numbers, to simplify working with segment angles.
displacement	Displacement vectors (represented as complex numbers) between each pair of consecutive points.

Examples

```

coords <- data.frame(x = c(1, 1.5, 2, 2.5, 3, 4),
                    y = c(0, 0, 1, 1, 2, 1),
                    times = c(0, 1, 2, 3, 4, 5))
trj <- TrajFromCoords(coords)

par(mar = c(4, 4, 0.5, 0.5) + 0.1)
plot(trj)

```

TrajGenerate	<i>Generate a random trajectory</i>
--------------	-------------------------------------

Description

Generates a trajectory. If `random` is `TRUE`, the trajectory will be a correlated random walk/idiothetic directed walk (Kareiva & Shigesada, 1983), corresponding to an animal navigating without a compass (Cheung, Zhang, Stricker, & Srinivasan, 2008). If `random` is `FALSE`, it will be a directed walk/allothetic directed walk/oriented path, corresponding to an animal navigating with a compass (Cheung, Zhang, Stricker, & Srinivasan, 2007, 2008).

Usage

```

TrajGenerate(
  n = 1000,
  random = TRUE,
  stepLength = 2,
  angularErrorSd = 0.5,
  angularErrorDist = function(n) stats::rnorm(n, sd = angularErrorSd),
  linearErrorSd = 0.2,
  linearErrorDist = function(n) stats::rnorm(n, sd = linearErrorSd),
  fps = 50,
  ...
)

```

Arguments

<code>n</code>	Number of steps in the trajectory.
<code>random</code>	If <code>TRUE</code> , a random search trajectory is returned, otherwise a directed trajectory (with <code>direction = 0</code> radians) is returned.

stepLength	Mean length of each step in the trajectory, in arbitrary length units.
angularErrorSd	Standard deviation of angular errors in radians.
angularErrorDist	Function which accepts a single argument - the number of values to return, and generates random deviates according to some distribution. The returned values are added to the previous step angle (when random == TRUE), or to 0 (is random == FALSE) to generate the step angle for each step in the trajectory. If the mean of the returned values is not zero, the walk will be biased.
linearErrorSd	Standard deviation of linear step length errors.
linearErrorDist	Function which accepts a single argument - the number of values to return, and generates random deviates according to some distribution. The returned values are added to stepLength to generate the lengths of each step.
fps	Simulated frames-per-second - used to generate times for each point in the trajectory.
...	Additional arguments are passed to TrajFromCoords .

Details

By default, for both random and directed walks, errors are normally distributed, unbiased, and independent of each other, so are *simple directed walks* in the terminology of Cheung, Zhang, Stricker, & Srinivasan, (2008). This behaviour may be modified by specifying alternative values for the angularErrorDist and/or linearErrorDist parameters.

The initial angle (for a random walk) or the intended direction (for a directed walk) is 0 radians. To change the initial angle or intended direction, call [TrajRotate](#) on the new trajectory. The starting position is (0, 0). To change the starting position, call [TrajTranslate](#) on the new trajectory.

Value

A new Trajectory with n segments and n + 1 coordinate pairs.

References

- Kareiva, P. M., & Shigesada, N. (1983). Analyzing insect movement as a correlated random walk. *Oecologia*, 56(2), 234-238. doi:10.1007/bf00379695
- Cheung, A., Zhang, S., Stricker, C., & Srinivasan, M. V. (2007). Animal navigation: the difficulty of moving in a straight line. *Biological Cybernetics*, 97(1), 47-61. doi:10.1007/s00422-007-0158-0
- Cheung, A., Zhang, S., Stricker, C., & Srinivasan, M. V. (2008). Animal navigation: general properties of directed walks. *Biological Cybernetics*, 99(3), 197-217. doi:10.1007/s00422-008-0251-z

Examples

```
# Generate a 1000 step correlated random walk
trj <- TrajGenerate()
plot(trj, main = "Correlated walk")
```

```

# Generate a 1000 step levy flight - paths lengths follow a cauchy distribution
trj <- TrajGenerate(linearErrorDist = rcauchy)
plot(trj, main = "Levy flight")

# Generate a short directed trajectory
trj <- TrajGenerate(n = 20, random = FALSE)
plot(trj, main = "Directed walk")

# Generate an uncorrelated random walk
trj <- TrajGenerate(500, angularErrorDist = function(n) runif(n, -pi, pi))
plot(trj, main = "Uncorrelated walk")

# Generate a walk directed northwards, starting from (200, 300),
# with a mean step length of 200. The initially generated trajectory
# is directed to angle 0, with starting point (0, 0)
trj <- TrajGenerate(n = 20, stepLength = 200, random = FALSE)
# Rotate 90 degrees about (0, 0) (i.e. from east to north)
trj <- TrajRotate(trj, pi / 2, relative = FALSE)
# Translate to desired starting point
trj <- TrajTranslate(trj, 200, 300)

```

TrajGetFPS

Trajectory frames-per-second

Description

Returns the frames-per-second recorded for this trajectory.

Usage

```
TrajGetFPS(trj)
```

Arguments

trj	Trajectory to query
-----	---------------------

TrajGetNCoords

Trajectory number of coordinates

Description

Returns the number of coordinates recorded for this trajectory, i.e. 1 more than the number of steps.

Usage

```
TrajGetNCoords(trj)
```

Arguments

trj Trajectory to query

TrajGetTimeUnits *Trajectory temporal units*

Description

Returns the temporal units specified for a scaled trajectory.

Usage

TrajGetTimeUnits(trj)

Arguments

trj Trajectory to query

See Also

[TrajFromCoords](#), [TrajGetUnits](#).

TrajGetUnits *Trajectory spatial units*

Description

Returns the spatial units specified for a scaled trajectory.

Usage

TrajGetUnits(trj)

Arguments

trj Trajectory to query

See Also

[TrajScale](#), [TrajGetTimeUnits](#).

`TrajInPolygon`*Test whether each of the points in a trajectory lie inside a polygon*

Description

Simply a wrapper around [point.in.polygon](#). The `sp` package must be installed for this function to be called. `sp` is not automatically installed as a dependency of `trajr`.

Usage

```
TrajInPolygon(trj, boundary)
```

Arguments

<code>trj</code>	Trajectory to test
<code>boundary</code>	A polygon defining the region to be tested against. Can be any structure that xy.coords can handle, such as a data frame with <code>x</code> and <code>y</code> columns.

Value

Integer array with a value for each point in the trajectory. Values are: 0: point is strictly exterior to boundary; 1: point is strictly interior to boundary; 2: point lies on the relative interior of an edge of boundary; 3: point is a vertex of boundary

See Also

[point.in.polygon](#), [xy.coords](#)

Examples

```
# Square arena
boundary <- data.frame(x = c(-10, 10, 10, -10), y = c(-10, -10, 10, 10))

# Generate a random trajectory
set.seed(1)
trj <- TrajGenerate(n = 10, stepLength = 2, angularErrorSd = .15)

# Test which points lie inside the boundary
print(TrajInPolygon(trj, boundary))
## [1] 1 1 1 1 1 1 0 0 0 0
```

TrajLength	<i>Trajectory length</i>
------------	--------------------------

Description

Calculates the cumulative length of a trajectory (or a portion of a trajectory), which is the total distance travelled along the trajectory.

Usage

```
TrajLength(trj, startIndex = 1, endIndex = nrow(trj))
```

Arguments

trj	Trajectory whose length is to be calculated.
startIndex	Index of the starting point.
endIndex	Index of the ending point.

Value

Numeric length of the trajectory.

See Also

[TrajStepLengths](#)

TrajLogSequence	<i>Logarithmically spaced sequence</i>
-----------------	--

Description

Convenience function to return a sequence of points which are regularly spaced when plotted on a logarithmic axis.

Usage

```
TrajLogSequence(from, to, length.out)
```

Arguments

from	Starting value of the sequence.
to	End (maximal) value of the sequence.
length.out	Desired length of the sequence (non-negative). Rounded up if fractional.

See Also

[seq](#)

TrajMeanVectorOfTurningAngles

Mean vector of turning angles

Description

Returns the mean vector of the turning angles, as defined by Batschelet, (1981). A unit vector is created for each turning angle in the trajectory, and the centre-of-mass/mean vector is returned.

Usage

```
TrajMeanVectorOfTurningAngles(trj, compass.direction = NULL)
```

Arguments

`trj` Trajectory object.

`compass.direction`

If not NULL, step angles are calculated relative to this angle (in radians), otherwise they are calculated relative to the previous step angle.

Details

According to Batschelet (1981), r may serve as a straightness index ranging from 0 to 1, where r is the length of the mean vector of turning angles of a trajectory with constant step length. Values of r near 1 indicating straighter paths. Hence, $r = \text{Mod}(\text{TrajMeanVectorOfTurningAngles}(\text{trj}))$, assuming that `trj` has a constant step length (e.g. has been rediscritized).

Value

A complex number r which represents the mean vector, $\text{Mod}(r)$ is the length of the mean vector which varies between 0 and 1, $\text{Arg}(r)$ is the angle.

References

Batschelet, E. (1981). Circular statistics in biology. ACADEMIC PRESS, 111 FIFTH AVE., NEW YORK, NY 10003, 1981, 388.

See Also

[TrajStraightness](#), [TrajAngles](#), [TrajRediscritize](#) for resampling a trajectory to a constant step length, [TrajResampleTime](#) for resampling a trajectory to a constant step time.

TrajMeanVelocity	<i>Trajectory mean velocity</i>
------------------	---------------------------------

Description

Calculates the mean or net velocity of a trajectory (or a portion of a trajectory). This is the velocity from the start point to the end point, ignoring the path that was taken.

Usage

```
TrajMeanVelocity(trj, startIndex = 1, endIndex = nrow(trj))
```

Arguments

trj	Trajectory whose duration is to be calculated.
startIndex	Index of the starting point.
endIndex	Index of the ending point.

Value

Numeric duration of the trajectory, in time units.

See Also

[TrajGetTimeUnits](#)

TrajMerge	<i>Combine multiple trajectories into a single whole trajectory</i>
-----------	---

Description

This is the inverse of [TrajSplit](#).

Usage

```
TrajMerge(parts)
```

Arguments

parts	A list containing one or more trajectories. The trajectories are concatenated together in order.
-------	--

Value

A single trajectory.

See Also[TrajSplit](#)**Examples**

```
trj <- TrajGenerate(n = 20)
ntrj <- TrajMerge(TrajSplit(trj, c(3, 9, 20)))
print(all(trj == ntrj))
## [1] TRUE
```

trajr

trajr: trajectory analysis in R

Description

A toolkit for the statistical analysis of 2-dimensional animal trajectories.

Details

Trajr operates on trajectories which are arrays of x and y coordinates. It can be used to calculate characteristics such as velocity and acceleration, as well as various measures of straightness or tortuosity. It also provides various convenience functions to assist with operating on multiple trajectories.

Trajr does not perform object tracking from videos, it operates on existing arrays of coordinates.

Trajectory creation

Most trajr functions operate on a Trajectory object. Trajectories are created by calling [TrajFromCoords](#). The function [TrajsBuild](#) allows you to create multiple Trajectories by reading their coordinates from files.

Typically, trajectories require smoothing to remove high frequency noise; see [TrajSmoothSG](#). Some methods require trajectories which have been resampled to a constant step length (*rediscretized*); see [TrajRediscretize](#).

You can create a random trajectory by calling [TrajGenerate](#).

Trajectory analysis

[TrajDerivatives](#) calculates the speed and acceleration of a trajectory (see also [TrajSpeedIntervals](#)).

Multiple algorithms for assessing straightness or tortuosity are available, see [TrajDirectionalChange](#), [TrajDirectionAutocorrelations](#), [TrajEmax](#), [TrajFractalDimension](#), [TrajSinuosity](#), and [TrajStraightness](#).

Other functions

Other functions provide information about trajectories, such as [TrajStepLengths](#), [TrajGetNCoords](#), [TrajGetUnits](#), [TrajGetTimeUnits](#), [TrajReverse](#), [TrajDuration](#), [TrajMeanVelocity](#), or allow some manipulations of trajectories, such as [TrajScale](#), [TrajReverse](#), [TrajTranslate](#), and [TrajRotate](#).

Trajr also provides the capability to plot a Trajectory and the results of some analyses.

TrajRediscretize	<i>Resample a trajectory to a constant step length</i>
------------------	--

Description

Constructs a new trajectory by resampling the input trajectory to a fixed step (or segment) length. Timing of frames is lost, so speed and acceleration cannot be calculated on a rediscretized trajectory.

Usage

```
TrajRediscretize(trj, R)
```

Arguments

trj	The trajectory to be resampled.
R	rediscretization step length, in the spatial units of trj.

Details

Based on the appendix in Bovee and Benhamou, (1988).

Value

A new trajectory with a constant segment length which follows trj.

References

Bovee, P., & Benhamou, S. (1988). Spatial analysis of animals' movements using a correlated random walk model. *Journal of Theoretical Biology*, 131(4), 419-433. doi:10.1016/S0022-5193(88)80038-9

TrajResampleTime	<i>Resample a trajectory to a constant time interval.</i>
------------------	---

Description

Constructs a new trajectory by resampling the input trajectory to a fixed time interval. Points are linearly interpolated along the trajectory. Spatial and time units are preserved.

Usage

```
TrajResampleTime(trj, stepTime, newFps = NULL)
```

Arguments

trj	The trajectory to be resampled.
stepTime	The resampled trajectory step time. Each step in the new trajectory will have this duration.
newFps	Value to be stored as the FPS value in the new trajectory (see TrajGetFPS). It is not otherwise used by this function.

Value

A new trajectory with a constant time interval for each step. Points in the new trajectory are calculated by linearly interpolating along trj.

Examples

```
# Simulate a trajectory with steps every 5 hours
set.seed(46)
trj <- TrajGenerate(10, stepLength = 5, fps = 1/5, timeUnits = "hours", linearErrorSd = .8)

# Resample to 1 hour steps
resampled <- TrajResampleTime(trj, 1)

par(mar = c(5, 4, .5, .5))
plot(trj, lwd = 2)
points(trj, pch = 16)
points(resampled, col = "red", draw.start.pt = FALSE)
```

TrajReverse	<i>Reverse a trajectory</i>
-------------	-----------------------------

Description

Reverses the direction of a trajectory, so that the starting point becomes the last point and vice versa.

Usage

```
TrajReverse(trj)
```

Arguments

trj	The Trajectory to be reversed.
-----	--------------------------------

Value

A copy of trj with direction reversed.

TrajRotate	<i>Rotate a trajectory</i>
------------	----------------------------

Description

Rotates a trajectory by angle (when relative is FALSE), or so that $\text{angle}(\text{finish} - \text{start}) == \text{angle}$ (when relative is TRUE).

Usage

```
TrajRotate(trj, angle = 0, origin = c(0, 0), relative = TRUE)
```

Arguments

trj	The trajectory to be rotated.
angle	The angle in radians between the first and last points in the rotated trajectory.
origin	Trajectory is rotated about this point.
relative	If TRUE, angle is the angle (after rotation) from the start to the end point of the trajectory. If FALSE, the trajectory is rotated about its start point by angle.

Value

A new trajectory which is a rotated version of the input trajectory.

 TrajsBuild

Construct multiple trajectories

Description

Reads multiple trajectories from files, performs some basic sanity checks on them, and optionally smooths and scales them. Attempts to collect and report errors for multiple trajectories in a single call.

Usage

```
TrajsBuild(
  fileNames,
  fps = NULL,
  scale = NULL,
  spatialUnits = NULL,
  timeUnits = NULL,
  csvStruct = list(x = 1, y = 2, time = NULL),
  smoothP = 3,
  smoothN = 41,
  translateToOrigin = FALSE,
  rootDir = NULL,
  csvReadFn = function(filename, ...) utils::read.csv(filename, stringsAsFactors =
    FALSE, ...),
  ...
)
```

Arguments

fileNames	Vector of the names of CSV files containing trajectory coordinates. All of the files must have the same columns. All file names must be unique. If rootDir is not NULL, then the file names are treated as regular expressions.
fps	Vector of frames-per-second values corresponding to the trajectories in fileNames. If length is 1, it is repeated to length(fileNames).
scale	Vector of scale values corresponding to the trajectories in fileNames. May be specified as character expressions (e.g. "1 / 1200") rather than numeric values. If NULL, the trajectories will not be scaled. If length is 1, it is repeated to length(fileNames).
spatialUnits	Abbreviated name of spatial coordinate units after scaling, e.g. "m".
timeUnits	Abbreviated name of temporal units, e.g. "s".
csvStruct	A list which identifies the columns in each CSV file which contain x-, y-, and optionally time-values.
smoothP	Filter order to be used for Savitzky-Golay smoothing (see TrajSmoothSG). If NA, no smoothing is performed.

smoothN	Filter length to be used for Savitzky-Golay smoothing (must be odd, see TrajSmoothSG). If NA, no smoothing is performed.
translateToOrigin	If TRUE, each trajectory is translated so that its starting point is at (0, 0).
rootDir	Optional name of a top level directory which contains the CSV files. If rootDir is not NULL, the CSV files may be located anywhere within rootDir or its sub-directories.
csvReadFn	Function used to read the CSV files. Required to accept arguments filename, ..., and return a data frame of coordinates, or a list of multiple data frames (see read.csv , read.csv2). The default function calls read.csv with argument stringsAsFactors = FALSE.
...	Additional arguments passed to csvReadFn.

Details

If rootDir is not null, it should be the name of a directory which is searched for the files in fileNames. The found files are then used as the list of files to be read in. This may be useful when the names of the files are known, but their exact location within a directory structure is not known.

For each file name in fileNames, reads the file by calling csvReadFn to obtain a set of coordinates and optionally times. A Trajectory is then constructed by passing the coordinates to [TrajFromCoords](#), passing in the appropriate fps value, and x, y and time column names/indices from csvStruct. If scale is not NULL, the trajectory is scaled by calling [TrajScale](#). If smoothP and smoothN are not NULL, the trajectory is smoothed by calling [TrajSmoothSG](#).

Value

A list of trajectories.

See Also

[read.csv](#), [TrajFromCoords](#), [TrajScale](#), [TrajSmoothSG](#), [TrajTranslate](#)

Examples

```
## Not run:
# Names of CSV files containing trajectory coordinates
fileNames <- c('xy001.csv', 'xy003.csv', 'xy004.csv')
# The files are all located somewhere under this directory
rootDir <- '.'
# Scale for these files is 1 / pixels per metre
scale <- c('1/1200', '1/1350', '1/1300')
# Files have columns y, x
csvStruct <- list(x = 2, y = 1)
# Apply default smoothing, and the files are formatted as conventional CSV,
# so there's no need to specify csvReadFn
trjs <- TrajsBuild(fileNames, fps = 50, scale = scale, units = "m",
                  csvStruct = csvStruct, rootDir = rootDir)
```

```
## End(Not run)
```

TrajScale	<i>Scale a trajectory</i>
-----------	---------------------------

Description

Scales the cartesian coordinates in a trajectory, for example, to convert units from pixels to metres.

Usage

```
TrajScale(trj, scale, units, yScale = scale)
```

Arguments

trj	The trajectory to be scaled.
scale	Scaling factor to be applied to the trajectory coordinates.
units	Character specifying the spatial units after scaling, e.g. "m" or "metres"
yScale	Optional scaling factor to be applied to the y-axis, which may be specified if the original coordinates are not square. Defaults to scale.

Value

new scaled trajectory.

Examples

```
set.seed(42)
trj <- TrajGenerate()
# original trajectory units are pixels, measured as having
# 47 pixels in 10 mm, so to convert to metres, scale the
# trajectory by the appropriate factor, i.e. (size in metres) / (size in pixels).
scale <- .01 / 47
scaled <- TrajScale(trj, scale, "m")
```

`TrajSinuosity`*Sinuosity of a trajectory*

Description

Calculates the sinuosity of a (constant step length) trajectory as defined by Bovet & Benhamou (1988), which is: $S = 1.18\sigma/\sqrt{q}$ where σ is the standard deviation of the step turning angles and q is the mean step length. A corrected sinuosity index is available as the function [TrajSinuosity2](#) which handles a wider range of variations in step angles.

Usage

```
TrajSinuosity(trj, compass.direction = NULL)
```

Arguments

`trj` Trajectory to calculate sinuosity of.
`compass.direction` if not NULL, turning angles are calculated for a directed walk, assuming the specified compass direction (in radians). Otherwise, a random walk is assumed.

Details

If your trajectory does not have a constant step length, it should be `_rediscretized_` by calling [TrajRediscretize](#) before calling this function.

Value

The sinuosity of `trj`.

References

Bovet, P., & Benhamou, S. (1988). Spatial analysis of animals' movements using a correlated random walk model. *Journal of Theoretical Biology*, 131(4), 419-433. doi:10.1016/S0022-5193(88)80038-9

See Also

[TrajAngles](#) for the turning angles in a trajectory, [TrajStepLengths](#) for the step lengths, [TrajSinuosity2](#) for a corrected version of sinuosity, and [TrajRediscretize](#) for resampling to a constant step length.

TrajSinuosity2	<i>Sinuosity of a trajectory</i>
----------------	----------------------------------

Description

Calculates the sinuosity of a trajectory as defined by Benhamou (2004), equation 8. This is a corrected version of the sinuosity index defined in Boveé & Benhamou (1988), which is suitable for a wider range of turning angle distributions, and does not require a constant step length.

Usage

```
TrajSinuosity2(trj, compass.direction = NULL)
```

Arguments

`trj` A Trajectory object.
`compass.direction`
if not NULL, turning angles are calculated for a directed walk, assuming the specified compass direction (in radians). Otherwise, a random walk is assumed.

Details

This function implements the formula

$$S = 2[p(((1 + c)/(1 - c)) + b^2)]^{-0.5}$$

where c is the mean cosine of turning angles, and b is the coefficient of variation of the step length.

References

Benhamou, S. (2004). How to reliably estimate the tortuosity of an animal's path. *Journal of Theoretical Biology*, 229(2), 209-220. doi:10.1016/j.jtbi.2004.03.016

See Also

[TrajSinuosity](#) for the uncorrected sinuosity index.

TrajsMergeStats	<i>Merge trajectory characteristics</i>
-----------------	---

Description

Builds a data frame by combining rows of statistical values for multiple trajectories. The statistics for each trajectory are defined by the caller in a user defined function - see the example for one way to achieve this.

Usage

```
TrajsMergeStats(
  trjs,
  statsFn,
  progressBar = c("none", "text", "win", "tk"),
  ...
)
```

Arguments

trjs	List of trajectories to be characterised.
statsFn	Function to calculate statistics of interest for a single trajectory.
progressBar	Displays an optional progressbar, which may be helpful if processing is very slow. The progressbar is displayed by printing to the console, by using <code>winProgressBar</code> or <code>tkProgressBar</code> , if <code>progressBar</code> is "text", "win" or "tk" respectively. The default is no progressbar (value "none"). The "win" progressbar is only available on Windows.
...	Additional arguments passed to <code>statsFn</code> .

Note

Any NULL valued statistics are converted to NAs.

Examples

```
## Not run:

# Define a function which calculates some statistics
# of interest for a single trajectory
characteriseTrajectory <- function(trj) {
  # Measures of speed
  derivs <- TrajDerivatives(trj)
  mean_speed <- mean(derivs$speed)
  sd_speed <- sd(derivs$speed)

  # Resample to constant step length.
  # Step length must be appropriate for the trajectory
```

```

resampled <- TrajRediscretize(trj, 2)

# Measures of straightness
sinuosity <- TrajSinuosity2(resampled)
Emax <- TrajEmax(resampled)

# Periodicity
resampled <- TrajRediscretize(trj, .001)
corr <- TrajDirectionAutocorrelations(resampled, round(nrow(resampled) / 4))
first_min <- TrajDAFindFirstMinimum(corr)

# Return a list with all of the statistics for this trajectory
list(mean_speed = mean_speed,
      sd_speed = sd_speed,
      sinuosity = sinuosity,
      Emax = Emax,
      first_min_deltaS = first_min[1],
      first_min_C = first_min[2])
}

trjs <- TrajsBuild(filenamees)
stats <- TrajsMergeStats(trjs, characteriseTrajectory)

## End(Not run)

```

TrajSmoothSG

Smooth a trajectory using a Savitzky-Golay filter

Description

Smooths a trajectory using a Savitzky-Golay smoothing filter.

Usage

```
TrajSmoothSG(trj, p = 3, n = p + 3 - p%%2, ...)
```

Arguments

trj	The trajectory to be smoothed.
p	polynomial order (passed to sgolayfilt).
n	Filter length (or window size), must be an odd number. Passed to sgolayfilt .
...	Additional arguments are passed to sgolayfilt .

Details

Consider carefully the effects of smoothing a trajectory with temporal gaps in the data. If the smoothed trajectory is used to derive speed and/or acceleration, it may be advisable to fill in the gaps before smoothing, possibly by calling `TrajResampleTime`.

Value

A new trajectory which is a smoothed version of the input trajectory.

See Also

[sgolayfilt](#)

Examples

```
set.seed(3)
trj <- TrajGenerate(500, random = TRUE, angularErrorSd = .25)
smoothed <- TrajSmoothSG(trj, 3, 31)
plot(trj)
plot(smoothed, col = "red", add = TRUE)
```

TrajSpeedIntervals *Calculate speed time intervals*

Description

Calculates and returns a list of time intervals during which speed is slower and/or faster than specified values. Speed is calculated by taking the modulus of velocity ([TrajVelocity](#)).

Usage

```
TrajSpeedIntervals(
  trj,
  fasterThan = NULL,
  slowerThan = NULL,
  interpolateTimes = TRUE,
  diff = c("backward", "central", "forward")
)
```

Arguments

trj	Trajectory to be analysed.
fasterThan, slowerThan	If not NULL, intervals will cover time periods where speed exceeds/is lower than this value.
interpolateTimes	If TRUE, times will be linearly interpolated between frames.
diff	Method used to calculate speed, see TrajVelocity for details. The default is "backward" to maintain backwards compatibility; in general, "central" provides a more accurate estimate of velocity.

Value

A data frame of class "TrajSpeedIntervals", each row is an interval, columns are:

startFrame	Indices of frames at the start of each interval.
stopFrame	Indices of frames at the end of each interval.
startTime	Time since start of trajectory at the start of each interval.
stopTime	Time since start of trajectory at the end of each interval
duration	Duration of each interval.

The data frame will also have non-standard attributes:

trajectory	Value of the trj argument.
slowerThan	Value of the slowerThan argument.
fasterThan	Value of the fasterThan argument.
speed	Data frame with columns speed and time.
derivs	Value returned by calling TrajDerivatives(trj). Provided for backwards-compatibility; use of speed is now preferred.

See Also

[TrajVelocity](#) for calculating trajectory velocity, [plot.TrajSpeedIntervals](#) for plotting speed over time with intervals highlighted.

Examples

```
# Plot speed, highlighting intervals where speed drops below 50 units/sec
set.seed(4)
trj <- TrajGenerate(200, random = TRUE)
smoothed <- TrajSmoothSG(trj, 3, 101)
intervals <- TrajSpeedIntervals(smoothed, diff = "central", slowerThan = 50, fasterThan = NULL)
plot(intervals)

# Report the duration of the longest period of low speed
cat(sprintf("Duration of the longest low-speed interval was %g secs\n", max(intervals$duration)))
```

TrajSplit

Split a trajectory into multiple sections

Description

Every point in trj will belong to exactly one of the returned sections. Note that this function will happily create single point trajectories.

Usage

```
TrajSplit(trj, idx)
```

Arguments

trj	The trajectory to be split
idx	Indices of splits. Each new section starts at one of these indices.

Value

A list containing one or more trajectories. The first trajectory in the list contains the first points from trj. Remaining trajectories contain the points starting from each of the idx values, in ascending order.

See Also

[TrajMerge](#), [TrajSplitAtFirstCrossing](#)

TrajSplitAtFirstCrossing

Split a trajectory into two parts, separated at the first boundary crossing

Description

This is basically a wrapper around [TrajInPolygon](#) and [TrajSplit](#).

Usage

```
TrajSplitAtFirstCrossing(trj, boundary)
```

Arguments

trj	The trajectory to split.
boundary	A polygon defining the boundary. Can be any structure that xy.coords can handle, such as a data frame with x and y columns.

Value

A list with 1 or 2 elements. If trj lies entirely inside or outside boundary, then the list simply contains trj. If trj crosses the boundary, then the list contains 2 trajectories. The first is the longest part of trj that lies entirely inside or outside boundary, and the second is the remainder of trj.

See Also

[TrajInPolygon](#), [TrajSplit](#)

Examples

```

# Square arena
boundary <- data.frame(x = c(-10, 10, 10, -10), y = c(-10, -10, 10, 10))

# Generate a random trajectory
set.seed(1)
trj <- TrajGenerate(n = 8, stepLength = 3, angularErrorSd = .4)
# Split the trajectory where it crosses the boundary
l <- TrajSplitAtFirstCrossing(trj, boundary)

# Plot the boundary and the two trajectories
plot(NULL, xlim = range(c(boundary$x, trj$x)), ylim = range(c(boundary$y, trj$y)), asp = 1)
polygon(boundary, border = "brown", lwd = 2)
lines(l[[1]], col = "#2040ff80", lwd = 3)
lines(l[[2]], col = "#ff204080", lwd = 3)

```

TrajsStatsReplaceNAs *Replace NAs in a data frame*

Description

Replaces NAs in a single column of a data frame with an imputed uninformative numeric replacement value, so that a principal component analysis can be applied without discarding data. Optionally adds a new "flag" column which contains 1 for each row which originally contained NA, otherwise 0.

Usage

```

TrajsStatsReplaceNAs(
  df,
  column,
  replacementValue = mean(df[, column], na.rm = TRUE),
  flagColumn = NULL
)

```

Arguments

<code>df</code>	Data frame to be adjusted.
<code>column</code>	Name or index of the column to be adjusted.
<code>replacementValue</code>	Numeric value to use instead of NA.
<code>flagColumn</code>	If not NULL, specifies the name of a new column to be added to the data frame, with value 0 for non-NA rows, 1 for NA rows. The column is added regardless of whether there are any NAs in the data.

Value

A copy of df with NAs replaced in column.

See Also

[prcomp](#)

Examples

```
df <- data.frame(x = c(1, 2, 3), y = c(NA, 5, 6), z = c(NA, NA, 9))
# Eliminate NAs in y, add a flag column, ignore other NAs
df <- TrajsStatsReplaceNAs(df, "y", flagColumn = "y.was.NA")
print(df)
```

TrajsStepLengths

Step lengths of multiple trajectories

Description

Returns the lengths of all of the steps in a list of trajectories

Usage

```
TrajsStepLengths(trjs)
```

Arguments

trjs A list of Trajectory objects.

Value

A numeric vector containing the lengths of every step in every trajectory.

Examples

```
## Not run:
trjs <- TrajsBuild(fileNames, scale = scale, units = "m")
# Print a summary about the step sizes across all trajectories
print(summary(TrajsStepLengths(trjs)))

## End(Not run)
```

TrajStepLengths	<i>Trajectory step lengths</i>
-----------------	--------------------------------

Description

Returns the lengths of each step in a trajectory.

Usage

```
TrajStepLengths(trj)
```

Arguments

trj Trajectory to query.

See Also

[TrajLength](#)

TrajStraightness	<i>Straightness of a Trajectory</i>
------------------	-------------------------------------

Description

Calculates the straightness index of a trajectory, D/L , where D is the beeline distance between the first and last points in the trajectory, and L is the path length travelled (Batschelet, 1981). Benhamou (2004) considers the straightness index to be a reliable measure of the efficiency of a directed walk, but inapplicable to random trajectories. The straightness index of a random walk tends towards zero as the number of steps increases, hence should only be used to compare the tortuosity of random walks consisting of a similar number of steps.

Usage

```
TrajStraightness(trj)
```

Arguments

trj Trajectory to calculate straightness of.

Details

The straightness index is also known as the net-to-gross displacement ratio. According to Batschelet (1981), this value (termed d) is an approximation of r , which is the length of the mean vector of turning angles of a constant step-length trajectory (see [TrajMeanVectorOfTurningAngles](#) and [TrajRediscretize](#) for creating a constant step-length trajectory).

Value

The straightness index of `trj`, which is a value between 0 (infinitely tortuous) to 1 (a straight line).

References

Batschelet, E. (1981). Circular statistics in biology. ACADEMIC PRESS, 111 FIFTH AVE., NEW YORK, NY 10003, 1981, 388.

Benhamou, S. (2004). How to reliably estimate the tortuosity of an animal's path. *Journal of Theoretical Biology*, 229(2), 209-220. doi:10.1016/j.jtbi.2004.03.016

See Also

[TrajDistance](#) for trajectory distance (or displacement), and [TrajLength](#) for trajectory path length.

TrajTranslate	<i>Translate a trajectory</i>
---------------	-------------------------------

Description

Shifts an entire trajectory by the specified delta x and y.

Usage

```
TrajTranslate(trj, dx, dy, dt = 0)
```

Arguments

<code>trj</code>	The Trajectory to be translated.
<code>dx</code>	Delta x.
<code>dy</code>	Delta y.
<code>dt</code>	Delta time.

Value

A new trajectory which is a translated version of the input trajectory.

Examples

```
# Shift a trajectory so that its origin is (10, 15).
# Note that trajectories created by TrajGenerate always start at (0, 0)
set.seed(42)
trj <- TrajGenerate()
trj <- TrajTranslate(trj, 10, 15)

# Translate a trajectory so its origin (0, 0) and it starts at time 0
trj <- TrajTranslate(trj, -trj$x[1], -trj$y[1], -trj$time[1])
```

TrajVelocity	<i>Velocity of a trajectory</i>
--------------	---------------------------------

Description

The velocity is approximated at each point of the trajectory using first-order finite differences. Central, forward or backward differences can be used. Central differences yield a more accurate approximation if the velocity is smooth. As a practical guide, if velocity doesn't change much between steps, use central differences. If it changes substantially (and not just as an artifact of recording noise), then use either forward or backward differences.

Usage

```
TrajVelocity(trj, diff = c("central", "forward", "backward"))
```

Arguments

trj	Trajectory whose velocity is to be calculated.
diff	Type of difference to be calculated, one of "central" (the default), "forward" or "backward".

Details

Intuitively, think of the central difference velocity at a point as the mean of the velocities of the two adjacent steps. Forward difference velocity is the velocity of the step starting at the point. Backward difference is the velocity of the step ending at the point.

Value

A vector of complex numbers representing the velocity at each point along the trajectory. The modulus ($\text{Mod}(v)$) is the magnitude of the velocity, i.e. the speed; the argument ($\text{Arg}(v)$) is the direction of the velocity; the real part ($\text{Re}(v)$) is velocity in the X direction; and the imaginary part ($\text{Im}(v)$) is velocity in the Y direction. The vector has an attribute, `trj`, with the trajectory as its value. If `diff` is "central", the first and last velocity values are NA since velocity cannot be calculated for them. If `diff` is "forward", the last value will be NA, and if `diff` is "backward", the first value will be NA.

See Also

[TrajAcceleration](#) for calculating acceleration; [TrajResampleTime](#) and [TrajRediscretize](#) to resample a trajectory to fixed time or length steps; [TrajSpeedIntervals](#) for calculating when speed crosses some threshold; Finite differences on [Wikipedia](#).

Examples

```
set.seed(11)
trj <- TrajGenerate(100)
# calculate velocity
vel <- TrajVelocity(trj)

# Obtain speed over time, with NAs removed
speed <- na.omit(data.frame(speed = Mod(vel), time = trj$time))

plot(speed ~ time, speed, type = 'l')
```


Index

ElapsedTimeProgressBarFn, 3

lines, 4

lines.Trajectory, 3

plot, 4–6

plot.TrajDirectionAutocorrelations, 4, 14

plot.Trajectory, 5

plot.TrajSpeedIntervals, 6, 41

point.in.polygon, 25

points, 7

points.Trajectory, 7

prcomp, 44

read.csv, 34

read.csv2, 34

seq, 26

sgolayfilt, 39, 40

strptime, 10

tkProgressBar, 38

TrajAcceleration, 7, 11, 12, 47

TrajAngles, 8, 27, 36

TrajConvertTime, 9, 20

TrajDAFindFirstMaximum (TrajDAMinMax), 10

TrajDAFindFirstMinimum, 13, 14

TrajDAFindFirstMinimum (TrajDAMinMax), 10

TrajDAMinMax, 10

TrajDerivatives, 11, 29

TrajDirectionalChange, 12, 29

TrajDirectionAutocorrelations, 10, 13, 29

TrajDistance, 14, 46

TrajDuration, 14, 30

TrajEmax, 15, 16, 18, 29

TrajExpectedSquareDisplacement, 16

TrajFractalDimension, 17, 19, 29

TrajFractalDimensionValues, 18, 18

TrajFromCoords, 5, 19, 22, 24, 29, 34

TrajGenerate, 21, 29

TrajGetFPS, 23, 31

TrajGetNCoords, 23, 30

TrajGetTimeUnits, 15, 24, 24, 28, 30

TrajGetUnits, 24, 24, 30

TrajInPolygon, 25, 42

TrajLength, 26, 45, 46

TrajLogSequence, 18, 26

TrajMeanVectorOfTurningAngles, 9, 27, 45

TrajMeanVelocity, 28, 30

TrajMerge, 28, 42

trajr, 29

TrajRediscretize, 8, 13, 27, 29, 30, 36, 45, 47

TrajResampleTime, 8, 27, 31, 47

TrajReverse, 30, 32

TrajRotate, 22, 30, 32

TrajsBuild, 29, 33

TrajScale, 24, 30, 34, 35

TrajSinuosity, 29, 36, 37

TrajSinuosity2, 18, 36, 37

TrajsMergeStats, 38

TrajSmoothSG, 12, 29, 33, 34, 39

TrajSpeedIntervals, 6, 12, 29, 40, 47

TrajSplit, 28, 29, 41, 42

TrajSplitAtFirstCrossing, 42, 42

TrajsStatsReplaceNAs, 43

TrajsStepLengths, 44

TrajStepLengths, 9, 26, 30, 36, 45

TrajStraightness, 27, 29, 45

TrajTranslate, 22, 30, 34, 46

TrajVelocity, 8, 11, 12, 40, 41, 47

xy.coords, 25, 42