Package 'QuAnTeTrack'

May 21, 2025

Title Quantitative Analysis of Tetrapod Trackways

Version 0.1.0

Description A quantitative and automated tool to extract (palaeo)biological information (i.e., measurements, velocities, similarity metrics, etc.) from the analysis of tetrapod trackways. Methods implemented in the package draw from several sources, including Alexander (1976) <doi:10.1038/261129a0>, Batschelet (1981, ISBN:9780120810505), Benhamou (2004) <doi:10.1016/j.jtbi.2004.03.016>, Bovet and Benhamou (1988) <doi:10.1016/S0022-5193(88)80038-9>, Cheung et al. (2007) <doi:10.1007/s00422-007-0158-0>, Cheung et al. (2008) <doi:10.1007/s00422-008-0251-z>, Cleasby et al. (2019) <doi:10.1007/s00265-019-2761-1>, Farlow et al. (1981) <doi:10.1038/294747a0>, Ostrom (1972) <doi:10.1016/0031-0182(72)90049-1>, Rohlf (2008) <https://sbmorphometrics.org/>, Rohlf (2009) <https://sbmorphometrics.org/>, Ruiz and Torices (2013) <doi:10.1080/10420940.2012.759115>, Scrucca et al. (2016) <doi:10.32614/RJ-2016-021>, Thulborn and Wade (1984) <https://www.museum.qld.gov.au/collections-and-research/memoirs/nature-21/mqm-n21-2-11-thulborn-wade>.

License CC0

Depends R (>= 3.5)

Imports berryFunctions (>= 1.21.14), car, dplyr, dtw, dunn.test, emmeans, geomorph (>= 4.0.3), ggplot2 (>= 3.3.6), ggrepel (>= 0.9.1), gridExtra, magrittr, mclust, NISTunits (>= 1.0.1), schoolmath (>= 0.4.1), shotGroups (>= 0.8.1), SimilarityMeasures (>= 1.4), splancs (>= 2.1.43), stringr (>= 1.4.0), trajr (>= 1.4.0)

Encoding UTF-8

LazyData true

RoxygenNote 7.3.2

Suggests knitr, rgl, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

URL https://github.com/MacroFunUV/QuAnTeTrack,
https://macrofunuv.github.io/QuAnTeTrack/

BugReports https://github.com/MacroFunUV/QuAnTeTrack/issues

VignetteBuilder knitr

NeedsCompilation no

Author Humberto G Ferrón [aut, cre, cph] (ORCID:
<https://orcid.org/0000-0003-2254-8424>)

Maintainer Humberto G Ferrón <humberto.ferron@uv.es>

Repository CRAN

Date/Publication 2025-05-21 15:40:02 UTC

Contents

`cluster_track` *Cluster tracks based on movement parameters*

Description

`cluster_track()` clusters trajectories based on various movement and velocity parameters calculated for each track.

Usage

```
cluster_track(data, veltrack, variables)
```

Arguments

`data`	A `track` R object, which is a list consisting of two elements: • `Trajectories`: A list of interpolated trajectories, where each trajectory is a series of midpoints between consecutive footprints. • `Footprints`: A list of data frames containing footprint coordinates, metadata (e.g., image reference, ID), and a marker indicating whether the footprint is actual or inferred.
`veltrack`	A `track velocity` R object consisting of a list of lists, where each sublist contains the computed parameters for a corresponding track.
`variables`	A character vector specifying the movement parameters to be used in the clustering analysis. Valid parameter names include: `"TurnAng"`, `"sdTurnAng"`, `"Distance"`, `"Length"`, `"StLength"`, `"sdStLength"`, `"Sinuosity"`, `"Straightness"`, `"Velocity"`, `"sdVelocity"`, `"MaxVelocity"`, `"MinVelocity"`.

Details

The `cluster_track()` function performs a model-based clustering analysis on track parameters using the `Mclust()` function from the **mclust** package.

The function first filters out tracks with fewer than four steps, as these tracks may not provide reliable movement data. It then calculates various movement parameters for each remaining track, including turning angles, distances, lengths, sinuosity, straightness, and velocities. Finally, the selected movement parameters are used as input for clustering the tracks.

If only one parameter is selected, the clustering is performed using equal variance (`"E"`) and variable variance (`"V"`) Gaussian models. If more than one parameter is selected, all Gaussian models available in `mclust.options("emModelNames")` are considered.

The following movement parameters can be included in the clustering:

- `"TurnAng"`: Turning angles for the track, measured in degrees. This measures how much the direction of movement changes at each step.
- `"sdTurnAng"`: The standard deviation of the turning angles, indicating how variable the turning angles are across the track.

- "Distance": The total distance covered by the track, calculated as the sum of the straight-line distances between consecutive points (in meters).
- "Length": The overall length of the track, a straight-line distance between the starting and ending points (in meters).
- "StLength": Step lengths for each step of the track, representing how far the object moved between two consecutive points (in meters).
- "sdStLength": The standard deviation of the step lengths, showing how consistent the steps are in length.
- "Sinuosity": A measure of the track's winding nature, calculated as the ratio of the actual track length to the straight-line distance (dimensionless).
- "Straightness": The straightness of the track, calculated as the straight-line distance divided by the total path length (dimensionless).
- "Velocity": The average velocity of the track, calculated as the total distance divided by the time elapsed between the first and last footprint (in meters per second).
- "sdVelocity": The standard deviation of the velocity, indicating how much the velocity fluctuates throughout the track.
- "MaxVelocity": The maximum velocity achieved during the track, identifying the fastest point (in meters per second).
- "MinVelocity": The minimum velocity during the track, identifying the slowest point (in meters per second).

The cluster_track() function has biological relevance in identifying groups of tracks with similar movement parameters, providing insights into ecological and behavioral patterns. By clustering tracks based on characteristics such as sinuosity, velocity, and turning angles, it allows detecting movement patterns associated with specific behaviors. This can help identify tracks potentially made by individuals moving together, which is useful for investigating hypotheses on gregarious behavior, predation strategies, or coordinated movement. Additionally, clustering serves as a preliminary step before similarity tests and simulations, refining track selection and improving hypothesis testing in movement ecology studies.

Value

A track clustering R object consisting of a list containing the following elements:

- matrix: A data frame containing the movement parameters calculated for each track.
- clust: An Mclust object containing the results of the model-based clustering analysis. This object provides the optimal (according to BIC) mixture model estimation. The output components are:
  - call: The matched call.
  - data: The input data matrix.
  - modelName: A character string denoting the model at which the optimal BIC occurs.
  - n: The number of observations in the data.
  - d: The dimension of the data.
  - G: The optimal number of mixture components.
  - BIC: All BIC values.

- `loglik`: The log-likelihood corresponding to the optimal BIC.
- `df`: The number of estimated parameters.
- `bic`: BIC value of the selected model.
- `icl`: ICL value of the selected model.
- `hypvol`: The hypervolume parameter for the noise component if required, otherwise set to NULL.
- `parameters`: A list with the following components:
  - `pro`: A vector whose k^{th} component is the mixing proportion for the k^{th} component of the mixture model. If missing, equal proportions are assumed.
  - `mean`: The mean for each component. If there is more than one component, this is a matrix whose k^{th} column is the mean of the k^{th} component of the mixture model.
- `variance`: A list of variance parameters for the model. The components of this list depend on the model specification. See the help file for mclustVariance for details.
- `z`: A matrix whose i,k^{th} entry is the probability that observation i in the test data belongs to the k^{th} class.
- `classification`: The classification corresponding to z, i.e., map(z).
- `uncertainty`: The uncertainty associated with the classification.

Logo

Author(s)

Humberto G. Ferrón

humberto.ferron@uv.es

Macroevolution and Functional Morphology Research Group (www.macrofun.es)

Cavanilles Institute of Biodiversity and Evolutionary Biology

Calle Catedrático José Beltrán Martínez, nº 2

46980 Paterna - Valencia - Spain

Phone: +34 (9635) 44477

References

Alexander, R. M. (1976). Estimates of speeds of dinosaurs. Nature, 261(5556), 129-130.

Ruiz, J., & Torices, A. (2013). Humans running at stadiums and beaches and the accuracy of speed estimations from fossil trackways. Ichnos, 20(1), 31-35.

Scrucca L., Fop M., Murphy T. B., & Raftery A. E. (2016) mclust 5: clustering, classification and density estimation using Gaussian finite mixture models. The R Journal, 8(1), 289-317.

See Also

`track_param`, `velocity_track`, `Mclust`

Examples

```
# Example 1: Cluster MountTom tracks using TurnAng and Velocity
H_mounttom <- c(
  1.380, 1.404, 1.320, 1.736, 1.364, 1.432, 1.508, 1.768, 1.600,
  1.848, 1.532, 1.532, 0.760, 1.532, 1.688, 1.620, 0.636, 1.784,
  1.676, 1.872, 1.648, 1.760, 1.612
) # Hip heights for MountTom tracks
veltrack_MountTom <- velocity_track(MountTom, H = H_mounttom)
result1 <- cluster_track(MountTom, veltrack_MountTom,
  variables = c("TurnAng", "Velocity")
)
result1$clust$classification

# Example 2: Cluster MountTom tracks using Sinuosity and Step Length
result2 <- cluster_track(MountTom, veltrack_MountTom,
  variables = c("Sinuosity", "StLength")
)
plot(result2$clust)

# Example 3: Cluster MountTom tracks using Maximum and Minimum Velocity
result3 <- cluster_track(MountTom, veltrack_MountTom,
  variables = c("MaxVelocity", "MinVelocity")
)
result3$clust$classification

# Example 4: Cluster MountTom tracks using Straightness
result4 <- cluster_track(MountTom, veltrack_MountTom, variables = "Straightness")
result4$clust$classification

# Example 5: Cluster PaluxyRiver tracks using Distance and Straightness
H_paluxyriver <- c(3.472, 2.200) # Hip heights for PaluxyRiver tracks
Method_paluxyriver <- c("A", "B") # Different methods for different tracks
veltrack_PaluxyRiver <- velocity_track(PaluxyRiver,
  H = H_paluxyriver,
  method = Method_paluxyriver
)
result5 <- cluster_track(PaluxyRiver, veltrack_PaluxyRiver,
  variables = c("Distance", "Straightness")
)
result5$matrix
result5$clust$classification

# Example 6: Cluster PaluxyRiver tracks using Length and SD of Velocity
result6 <- cluster_track(PaluxyRiver, veltrack_PaluxyRiver,
  variables = c("Length", "sdVelocity")
)
plot(result6$clust)

# Example 7: Cluster PaluxyRiver tracks using TurnAng and SD of TurnAng
result7 <- cluster_track(PaluxyRiver, veltrack_PaluxyRiver,
  variables = c("TurnAng", "sdTurnAng")
)
```

```
result7$clust$classification

# Example 8: Cluster PaluxyRiver tracks using Sinuosity
result8 <- cluster_track(PaluxyRiver, veltrack_PaluxyRiver,
  variables = c("Sinuosity")
)
result8$clust$classification
```

`combined_prob` *Calculate combined probabilities of similarity or intersection metrics of tracks*

Description

`combined_prob()` calculates the combined probabilities of similarity and intersection metrics derived from different models. The function uses simulation data to extract *p*-values, providing insight into the significance of combined metrics across various similarity assessments.

Usage

```
combined_prob(data, metrics = NULL)
```

Arguments

`data`	A `track` R object, which is a list consisting of two elements: • `Trajectories`: A list of interpolated trajectories, where each trajectory is a series of midpoints between consecutive footprints. • `Footprints`: A list of data frames containing footprint coordinates, metadata (e.g., image reference, ID), and a marker indicating whether the footprint is actual or inferred.
`metrics`	A list of `track similarity` and/or `track intersection` R objects derived from different tests. All tests must be based on the same number of simulations.

Details

The `combined_prob()` function combines *p*-values derived from multiple similarity metric tests and intersection tests. It calculates the combined *p*-values by assessing the probability of observing the combined metrics across simulated datasets. This function is particularly useful for comparing multiple models and evaluating their collective performance in terms of *p*-values.

Value

A list containing:

`P_values (model names)`
: A matrix of *p*-values for the combined metrics across all trajectories. Each entry represents the probability of observing the combined metrics between the corresponding pair of trajectories.

`P_values_combined (model names)`
: A numeric value representing the overall probability of observing the combined metrics, across all pairs of trajectories.

Logo

Author(s)

Humberto G. Ferrón

humberto.ferron@uv.es

Macroevolution and Functional Morphology Research Group (www.macrofun.es)

Cavanilles Institute of Biodiversity and Evolutionary Biology

Calle Catedrático José Beltrán Martínez, nº 2

46980 Paterna - Valencia - Spain

Phone: +34 (9635) 44477

See Also

`tps_to_track`, `simulate_track`, `track_intersection`, `simil_DTW_metric`, `simil_Frechet_metric`

Examples

```
# Example 1: "Directed" model and similarity metrics.
s1 <- simulate_track(PaluxyRiver, nsim = 3, model = "Directed")
DTW1 <- simil_DTW_metric(PaluxyRiver, test = TRUE, sim = s1, superposition = "None")
Frechet1 <- simil_Frechet_metric(PaluxyRiver, test = TRUE, sim = s1, superposition = "None")
int1 <- track_intersection(PaluxyRiver, test = TRUE, H1 = "Lower", sim = s1,
  origin.permutation = "None")
combined_prob(PaluxyRiver, metrics = list(DTW1, Frechet1, int1))

# Example 2: "Constrained" model and similarity metrics.
s2 <- simulate_track(PaluxyRiver, nsim = 3, model = "Constrained")
DTW2 <- simil_DTW_metric(PaluxyRiver, test = TRUE, sim = s2,
  superposition = "None")
Frechet2 <- simil_Frechet_metric(PaluxyRiver, test = TRUE, sim = s2,
  superposition = "None")
int2 <- track_intersection(PaluxyRiver, test = TRUE, H1 = "Lower", sim = s2,
  origin.permutation = "Min.Box")
combined_prob(PaluxyRiver, metrics = list(DTW2, Frechet2, int2))

# Example 3: "Unconstrained" model and similarity metrics.
s3 <- simulate_track(PaluxyRiver, nsim = 3, model = "Unconstrained")
DTW3 <- simil_DTW_metric(PaluxyRiver, test = TRUE, sim = s3,
  superposition = "None")
Frechet3 <- simil_Frechet_metric(PaluxyRiver, test = TRUE, sim = s3,
  superposition = "None")
int3 <- track_intersection(PaluxyRiver, test = TRUE, H1 = "Lower", sim = s3,
  origin.permutation = "Conv.Hull")
```

```
combined_prob(PaluxyRiver, metrics = list(DTW3, Frechet3, int3))
```

`mode_velocity` *Test for steady, acceleration, or deceleration along trajectories*

Description

`mode_velocity()` evaluates the trend in velocity along each trajectory by applying Spearman's rank correlation test. The function classifies the trend into "acceleration", "deceleration", or "steady" based on the correlation and the *p*-value.

Usage

```
mode_velocity(trackvel)
```

Arguments

`trackvel`	A `track` `velocity` object where each element corresponds to a track and contains a vector of velocity or relative stride length data.

Details

The `mode_velocity()` function performs the following operations:

- **Spearman's Rank Correlation Test:**
  - This non-parametric test assesses the strength and direction of a monotonic relationship between two variables. It does not require assumptions about the normality of data or a linear relationship between velocity and step number.
  - It uses ranks rather than raw values, making it robust to outliers and suitable for detecting general trends (acceleration or deceleration) in velocity data.
- **Function Operation:**
  - For each trajectory in the `trackvel` list, the function calculates the Spearman correlation coefficient and the associated *p*-value between velocity and step number.
  - Based on the *p*-value and correlation coefficient, it classifies the trend as "acceleration", "deceleration", or "steady".
  - If a trajectory contains fewer than 3 steps, the function returns a message indicating insufficient data for correlation analysis.
- **Advantages:**
  - The non-parametric nature allows flexibility with data distributions and reduced sensitivity to outliers compared to parametric tests.
  - Effective for detecting monotonic trends (either increasing or decreasing) when the correlation is statistically significant.
- **Limitations:**

- May be unreliable with very small sample sizes (e.g., fewer than 3 steps), providing potentially non-informative results.
- Does not capture the magnitude of change or provide detailed insights into the rate of acceleration or deceleration.
- Identifies monotonic trends based on statistical significance but does not distinguish between different types of monotonic relationships (e.g., steady acceleration vs. abrupt changes).

Interpretation of Results:

- **Acceleration:** If the *p*-value is less than 0.05 and the Spearman correlation coefficient is positive.
- **Deceleration:** If the *p*-value is less than 0.05 and the Spearman correlation coefficient is negative.
- **Steady:** If the *p*-value is greater than or equal to 0.05, indicating no significant monotonic relationship.

Usage Considerations:

- Ensure that each trajectory in `trackvel` has a sufficient number of steps for meaningful analysis.
- For more detailed analysis of velocity trends, consider complementary methods such as linear or non-linear regression, or specialized change point detection techniques.

Value

A list where each element corresponds to a trajectory from the input `trackvel` and contains:

- **correlation:** The result of the Spearman correlation test, including the correlation coefficient and *p*-value.
- **trend:** A classification of the trend as "Acceleration", "Deceleration", or "Steady" based on the *p*-value and the correlation coefficient.
- If a trajectory has fewer than 3 steps, the entry contains the message "Less than three steps."

Logo

Author(s)

Humberto G. Ferrón

humberto.ferron@uv.es

Macroevolution and Functional Morphology Research Group (www.macrofun.es)

Cavanilles Institute of Biodiversity and Evolutionary Biology

Calle Catedrático José Beltrán Martínez, nº 2

46980 Paterna - Valencia - Spain

Phone: +34 (9635) 44477

See Also

tps_to_track, velocity_track, plot_velocity

Examples

```
# Example 1: Test for Steady, Acceleration, or Deceleration in MountTom dataset.

# Hip heights for each track in the MountTom dataset
H_mounttom <- c(
  1.380, 1.404, 1.320, 1.736, 1.364, 1.432, 1.508, 1.768, 1.600,
  1.848, 1.532, 1.532, 0.760, 1.532, 1.688, 1.620, 0.636, 1.784,
  1.676, 1.872, 1.648, 1.760, 1.612
)

# Calculate velocities using the default Method "A"
V_mounttom <- velocity_track(MountTom, H = H_mounttom)

# Test for Steady, Acceleration, or Deceleration
mode_velocity(V_mounttom)

# Example 2: Test for Steady, Acceleration, or Deceleration in PaluxyRiver dataset.

# Hip heights for each track in the PaluxyRiver dataset
H_paluxyriver <- c(3.472, 2.200)

# Specify different methods for different tracks
Method_paluxyriver <- c("A", "B")

# Calculate velocities using specified methods
V_paluxyriver <- velocity_track(PaluxyRiver,
  H = H_paluxyriver,
  method = Method_paluxyriver
)

# Test for Steady, Acceleration, or Deceleration
mode_velocity(V_paluxyriver)
```

MountTom *MountTom Dinosaur Track Dataset*

Description

A 'track' R object representing dinosaur tracks from the Mount Tom site.

Usage

```
MountTom
```

Format

A list consisting of two elements:

- **Trajectories**: A list of interpolated trajectories, where each trajectory is a series of midpoints between consecutive footprints.
- **Footprints**: A list of data frames containing footprint coordinates, metadata (e.g., image reference, ID), and a marker indicating whether the footprint is actual or inferred.

Source

Ostrom, J. H. (1972). Were some dinosaurs gregarious?. Palaeogeography, Palaeoclimatology, Palaeoecology, 11(4), 287-301.

`PaluxyRiver` *PaluxyRiver Dinosaur Track Dataset*

Description

A 'track' R object representing dinosaur tracks from the Paluxy River site.

Usage

```
PaluxyRiver
```

Format

A list consisting of two elements:

- **Trajectories**: A list of interpolated trajectories, where each trajectory is a series of midpoints between consecutive footprints.
- **Footprints**: A list of data frames containing footprint coordinates, metadata (e.g., image reference, ID), and a marker indicating whether the footprint is actual or inferred.

Source

Farlow, J. O., O'Brien, M., Kuban, G. J., Dattilo, B. F., Bates, K. T., Falkingham, P. L., & Piñuela, L. (2012). Dinosaur Tracksites of the Paluxy River Valley (Glen Rose Formation, Lower Cretaceous), Dinosaur Valley State Park, Somervell County, Texas. In Proceedings of the V International Symposium about Dinosaur Palaeontology and their Environment (pp. 41-69). Burgos: Salas de los Infantes.

plot_direction *Plot direction data in tracks.*

Description

plot_direction() generates different types of plots to visualize the direction data from track R objects. The function allows for the creation of boxplots, polar histograms of step directions, polar histograms of average directions per track, and faceted polar histograms.

Usage

```
plot_direction(
  data,
  plot_type = "boxplot",
  angle_range = 30,
  y_labels_position = 90,
  y_breaks_manual = NULL
)
```

Arguments

data	A track R object, which is a list consisting of two elements: • Trajectories: A list of interpolated trajectories, where each trajectory is a series of midpoints between consecutive footprints. • Footprints: A list of data frames containing footprint coordinates, metadata (e.g., image reference, ID), and a marker indicating whether the footprint is actual or inferred.
plot_type	A character string indicating the type of plot to generate. The options are "boxplot", "polar_steps", "polar_average", and "faceted". Default is "boxplot".
angle_range	A numeric value specifying the width of the bins (in degrees) used for polar plots. Default is 30 degrees.
y_labels_position	A numeric value specifying the position (in degrees) of the y-axis labels in the polar plots. Default is 90 degrees.
y_breaks_manual	A numeric vector specifying manual breaks for the y-axis in polar plots. If NULL, the breaks are calculated automatically. Default is NULL.

Details

The plot_direction() function provides four types of plots:

- "boxplot": A boxplot showing the distribution of step direction values for each track.
- "polar_steps": A polar plot showing the frequency of step in different direction bins.

- "polar_average": A polar plot showing the frequency of average directions per track in different direction bins.
- "faceted": A polar plot similar to "polar_steps" but faceted by track.

The angle_range parameter defines the bin width for the polar plots, and y_labels_position allows for adjusting the position of y-axis labels. The y_breaks_manual parameter lets users manually specify the breaks on the y-axis for finer control over the appearance of the polar plots.

Value

A ggplot object that displays the specified plot type. The **ggplot2** package is used for plotting.

Logo

Author(s)

Humberto G. Ferrón

humberto.ferron@uv.es

Macroevolution and Functional Morphology Research Group (www.macrofun.es)

Cavanilles Institute of Biodiversity and Evolutionary Biology

Calle Catedrático José Beltrán Martínez, nº 2

46980 Paterna - Valencia - Spain

Phone: +34 (9635) 44477

See Also

tps_to_track, test_direction

Examples

```
# Example 1: Boxplot of Direction Data in MountTom Dataset
plot_direction(MountTom, plot_type = "boxplot")

# Example 2: Polar Plot of Step Directions in MountTom Dataset
plot_direction(MountTom, plot_type = "polar_steps")

# Example 3: Polar Plot of Average Directions Per Track in MountTom Dataset
plot_direction(MountTom, plot_type = "polar_average")

# Example 4: Faceted Polar Plot of Step Directions in MountTom Dataset
plot_direction(MountTom, plot_type = "faceted")

# Example 5: Polar Plot with Custom Angle Range in MountTom Dataset
plot_direction(MountTom, plot_type = "polar_steps", angle_range = 90)

# Example 6: Polar Plot with Custom Y-Axis Labels and Breaks in MountTom Dataset
plot_direction(MountTom,
```

```
  plot_type = "polar_steps", y_labels_position = 0,
  y_breaks_manual = c(0, 15, 30, 45, 60)
)

# Example 7: Boxplot of Direction Data in PaluxyRiver Dataset
plot_direction(PaluxyRiver, plot_type = "boxplot")

# Example 8: Polar Plot of Step Directions in PaluxyRiver Dataset
plot_direction(PaluxyRiver, plot_type = "polar_steps")

# Example 9: Polar Plot of Average Directions Per Track with Custom Breaks in PaluxyRiver Dataset
plot_direction(PaluxyRiver,
  plot_type = "polar_average",
  y_breaks_manual = c(1, 2)
)

# Example 10: Faceted Polar Plot of Step Directions in PaluxyRiver Dataset
plot_direction(PaluxyRiver, plot_type = "faceted")

# Example 11: Polar Plot of Average Directions Per Track with Custom Breaks in PaluxyRiver Dataset
plot_direction(PaluxyRiver,
  plot_type = "polar_average",
  y_breaks_manual = c(1, 2)
)

# Example 12: Polar Plot with Custom Y-Axis Labels in PaluxyRiver Dataset
plot_direction(PaluxyRiver,
  plot_type = "polar_steps",
  y_labels_position = -90
)
```

`plot_sim` *Plot Simulated Tracks*

Description

`plot_sim()` creates a plot that visualizes both simulated and actual movement trajectories. This function is useful for comparing the simulated tracks generated by `simulate_track()` with the observed trajectories to evaluate how well the simulation models represent real movement patterns.

Usage

```
plot_sim(
  data,
  sim,
  colours_sim = NULL,
  alpha_sim = NULL,
  lwd_sim = NULL,
  colours_act = NULL,
```

```
  alpha_act = NULL,
  lwd_act = NULL
)
```

Arguments

data	A track R object, which is a list consisting of two elements: • Trajectories: A list of interpolated trajectories, where each trajectory is a series of midpoints between consecutive footprints. • Footprints: A list of data frames containing footprint coordinates, metadata (e.g., image reference, ID), and a marker indicating whether the footprint is actual or inferred.
sim	A track simulation R object, where each object is a list of simulated trajectories stored as track R objects.
colours_sim	A vector of colors for plotting each set of simulated trajectories. If NULL, the default color will be black ("#000000").
alpha_sim	A numeric value between 0 and 1 for the transparency level of simulated trajectories. The default is 0.1.
lwd_sim	A numeric value for the line width of the simulated trajectory lines. The default is 0.5.
colours_act	A vector of colors for plotting actual trajectories. If NULL, the default color will be black ("#000000").
alpha_act	A numeric value between 0 and 1 for the transparency level of actual trajectories. The default is 0.6.
lwd_act	A numeric value for the line width of the actual trajectory lines. The default is 0.8.

Details

The function uses **ggplot2** to create a plot with the following components:

- Simulated trajectories are displayed with paths colored according to the colours_sim parameter, with the specified transparency alpha_sim and line width lwd_sim.
- Actual trajectories are overlaid in the colors specified by colours_act, with a transparency level alpha_act and line width lwd_act to provide a clear comparison.

Value

A ggplot object displaying the simulated and actual trajectories.

Logo

Author(s)

Humberto G. Ferrón

humberto.ferron@uv.es

Macroevolution and Functional Morphology Research Group (www.macrofun.es)

Cavanilles Institute of Biodiversity and Evolutionary Biology

Calle Catedrático José Beltrán Martínez, nº 2

46980 Paterna - Valencia - Spain

Phone: +34 (9635) 44477

See Also

tps_to_track, simulate_track

Examples

```
# Example 1: Simulate tracks using data from the Paluxy River
# Default model (Unconstrained movement)
simulated_tracks <- simulate_track(PaluxyRiver, nsim = 3)

# Plot simulated tracks with default settings and actual tracks
plot_sim(PaluxyRiver, simulated_tracks)

# Example 2: Simulate tracks using the "Directed" model, representing movement toward a
# resource
simulated_tracks_directed <- simulate_track(PaluxyRiver, nsim = 3, model = "Directed")

# Plot simulated tracks with specific colors and transparency for "Directed" model
plot_sim(PaluxyRiver, simulated_tracks_directed,
  colours_sim = c("#E69F00", "#56B4E9"),
  alpha_sim = 0.4, lwd_sim = 1, colours_act = c("black", "black"), alpha_act = 0.7,
  lwd_act = 2
)

# Example 3: Simulate tracks using the "Constrained" model, representing movement along
# a feature
simulated_tracks_constrained <- simulate_track(PaluxyRiver, nsim = 3, model = "Constrained")

# Plot simulated tracks with a different color scheme and width for "Constrained" model
plot_sim(PaluxyRiver, simulated_tracks_constrained,
  colours_sim = c("#E69F00", "#56B4E9"),
  alpha_sim = 0.6, lwd_sim = 0.1, alpha_act = 0.5, lwd_act = 2
)

# Example 4: Simulate tracks using the "Unconstrained" model (random exploratory
# movement)
simulated_tracks_unconstrained <- simulate_track(PaluxyRiver, nsim = 3, model = "Unconstrained")

# Plot simulated tracks with default colors and increased transparency for "Unconstrained"
# model
```

```
plot_sim(PaluxyRiver, simulated_tracks_unconstrained,
  colours_sim = c("#E69F00", "#56B4E9"),
  alpha_sim = 0.2, lwd_sim = 1, colours_act = c("#E69F00", "#56B4E9"), alpha_act = 0.9,
  lwd_act = 2
)

# Subsetting trajectories with four or more steps in the Mount Tom dataset
sbMountTom <- subset_track(MountTom, tracks = c(1, 2, 3, 4, 7, 8, 9, 13, 15, 16, 18))

# Example 5: Simulate tracks using data from Mount Tom
simulated_tracks_mt <- simulate_track(sbMountTom, nsim = 3)

# Plot simulated tracks with default settings and actual tracks from Mount Tom
plot_sim(sbMountTom, simulated_tracks_mt)

# Example 6: Simulate tracks using the "Directed" model for Mount Tom
simulated_tracks_mt_directed <- simulate_track(sbMountTom, nsim = 3, model = "Directed")

# Plot simulated tracks with specific colors and transparency for "Directed" model for Mount
# Tom
plot_sim(sbMountTom, simulated_tracks_mt_directed, colours_sim = c(
  "#E69F00", "#56B4E9",
  "#009E73", "#F0E442", "#0072B2", "#D55E00", "#CC79A7", "#999999", "#F4A300",
  "#6C6C6C", "#1F77B4"
), alpha_sim = 0.3, lwd_sim = 1.5, alpha_act = 0.8, lwd_act = 2)

# Example 7: Simulate tracks using the "Constrained" model for Mount Tom
simulated_tracks_mt_constrained <- simulate_track(sbMountTom, nsim = 3, model = "Constrained")

# Plot simulated tracks with different color scheme and increased line width for "Constrained"
# model
plot_sim(sbMountTom, simulated_tracks_mt_constrained, colours_sim = c(
  "#E41A1C", "#377EB8",
  "#4DAF4A", "#FF7F00", "#F781BF", "#A65628", "#FFFF33", "#8DD3C7", "#FB8072",
  "#80BF91", "#F7F7F7"
), alpha_sim = 0.5, lwd_sim = 0.2, alpha_act = 0.6, lwd_act = 2)

# Example 8: Simulate tracks using the "Unconstrained" model for Mount Tom
simulated_tracks_mt_unconstrained <- simulate_track(sbMountTom, nsim = 3, model = "Unconstrained")

# Plot simulated tracks with a different color scheme and transparency for "Unconstrained" model
plot_sim(sbMountTom, simulated_tracks_mt_unconstrained, colours_sim = c(
  "#6BAED6", "#FF7F00",
  "#1F77B4", "#D62728", "#2CA02C", "#9467BD", "#8C564B", "#E377C2", "#7F7F7F",
  "#BCBD22", "#17BECF"
), alpha_sim = 0.2, lwd_sim = 0.5, colours_act = c(
  "#6BAED6",
  "#FF7F00", "#1F77B4", "#D62728", "#2CA02C", "#9467BD", "#8C564B", "#E377C2",
  "#7F7F7F", "#BCBD22", "#17BECF"
), alpha_act = 1, lwd_act = 2)
```

plot_track *Plot tracks and footprints*

Description

plot_track() visualizes track and footprint data in various ways, allowing for the plotting of trajectories, footprints, or both combined, with customizable aesthetics.

Usage

```
plot_track(
  data,
  plot = "FootprintsTracks",
  colours = NULL,
  cex.f = NULL,
  shape.f = NULL,
  alpha.f = NULL,
  cex.t = NULL,
  alpha.t = NULL,
  plot.labels = NULL,
  labels = NULL,
  box.p = NULL,
  cex.l = NULL,
  alpha.l = NULL
)
```

Arguments

data	A track R object, which is a list consisting of two elements: • Trajectories: A list of interpolated trajectories, where each trajectory is a series of midpoints between consecutive footprints. • Footprints: A list of data frames containing footprint coordinates, metadata (e.g., image reference, ID), and a marker indicating whether the footprint is actual or inferred.
plot	Type of plot to generate. Options are "FootprintsTracks" (default), "Tracks", or "Footprints". Determines what elements are included in the plot.
colours	A vector of colors to be used for different tracks. If NULL, defaults to black. The length of this vector should match the number of tracks in the data.
cex.f	The size of the footprint points. Default is 2.5.
shape.f	A vector of shapes to be used for footprints in different tracks. If NULL, defaults to 19 (solid circle). The length of this vector should match the number of tracks in the data.
alpha.f	The transparency of the footprint points. Default is 0.5.
cex.t	The size of the track lines. Default is 0.5.

alpha.t	The transparency of the track lines. Default is 1.
plot.labels	Logical indicating whether to add labels to each track. Default is FALSE.
labels	A vector of labels for each track. If NULL, labels are automatically generated from track names.
box.p	Padding around label boxes, used only if plot.labels is TRUE. Adjusts the spacing around the label text.
cex.l	The size of the labels. Default is 3.88.
alpha.l	The transparency of the labels. Default is 0.5.

Value

A ggplot object that displays the specified plot type, including tracks, footprints, or both, from track R objects. The **ggplot2** package is used for plotting.

Logo

Author(s)

Humberto G. Ferrón

humberto.ferron@uv.es

Macroevolution and Functional Morphology Research Group (www.macrofun.es)

Cavanilles Institute of Biodiversity and Evolutionary Biology

Calle Catedrático José Beltrán Martínez, nº 2

46980 Paterna - Valencia - Spain

Phone: +34 (9635) 44477

See Also

tps_to_track

Examples

```
# Example 1: Basic Plot with Default Settings - MountTom Dataset
plot_track(MountTom)

# Example 2: Basic Plot with Default Settings - PaluxyRiver Dataset
plot_track(PaluxyRiver)

# Example 3: Plot Tracks Only - MountTom Dataset
plot_track(MountTom, plot = "Tracks")

# Example 4: Plot Footprints Only - PaluxyRiver Dataset
plot_track(PaluxyRiver, plot = "Footprints")

# Example 5: Custom Colors for Tracks - MountTom Dataset
```

```
custom_colors <- c(
  "#008000", "#0000FF", "#FF0000", "#800080", "#FFA500", "#FFC0CB", "#FFFF00",
  "#00FFFF", "#A52A2A", "#FF00FF", "#808080", "#000000", "#006400", "#00008B",
  "#8B0000", "#FF8C00", "#008B8B", "#A9A9A9", "#000080", "#808000", "#800000",
  "#008080", "#FFD700"
)
plot_track(MountTom, colours = custom_colors)

# Example 6: Larger Footprints and Track Lines - PaluxyRiver Dataset
plot_track(PaluxyRiver, cex.f = 5, cex.t = 2)

# Example 7: Semi-Transparent Footprints and Tracks - MountTom Dataset
plot_track(MountTom, alpha.f = 0.5, alpha.t = 0.5)

# Example 8: Different Shapes for Footprints - PaluxyRiver Dataset
plot_track(PaluxyRiver, shape.f = c(16, 17))

# Example 9: Plot with Labels for Tracks - MountTom Dataset
labels <- paste("Track", seq_along(MountTom[[1]]))
plot_track(MountTom, plot.labels = TRUE, labels = labels, cex.l = 4, box.p = 0.3, alpha.l = 0.7)

# Example 10: Custom Colors and Shapes for Footprints Only - PaluxyRiver Dataset
plot_track(PaluxyRiver, plot = "Footprints", colours = c("purple", "orange"), shape.f = c(15, 18))

# Example 11: Larger Line Size & Custom Colors for Tracks Only - MountTom Dataset
plot_track(MountTom, plot = "Tracks", cex.t = 1.5, colours = custom_colors)

# Example 12: Black Footprints and Tracks with Labels - PaluxyRiver Dataset
plot_track(PaluxyRiver,
  colours = NULL, shape.f = c(16, 16), plot.labels = TRUE,
  labels = c("Saurpod", "Theropod"), cex.l = 2, alpha.l = 0.5
)
```

`plot_velocity` *Plot trajectories colored by velocity or relative stride length*

Description

`plot_velocity()` creates a plot of trajectories, colored by either velocity or relative stride length from `track` and `track velocity` R objects. The function uses **ggplot2** package for visualization and allows customization of line width and color gradients.

Usage

```
plot_velocity(
  data,
  trackvel,
  param = NULL,
  lwd = NULL,
```

```
  colours = NULL,
  legend = NULL
)
```

Arguments

data	A track R object, which is a list consisting of two elements: • Trajectories: A list of interpolated trajectories, where each trajectory is a series of midpoints between consecutive footprints. • Footprints: A list of data frames containing footprint coordinates, metadata (e.g., image reference, ID), and a marker indicating whether the footprint is actual or inferred.
trackvel	A track velocity R object consisting of a list where each element corresponds to a track and contains velocity or relative stride length data.
param	A character string specifying the parameter to plot. Options are: • "V" for velocity. • "RSL" for relative stride length. If NULL, the default value "V" will be used.
lwd	Numeric. Line width for the plotted trajectories. Default is 1.
colours	A vector of colors to use for the gradient. Default is a predefined set of colors.
legend	Logical. If TRUE, the legend will be shown. If FALSE, the legend will be removed. Default is TRUE.

Details

The function creates a plot where each trajectory is colored based on the specified parameter ("V" for velocity or "RSL" for relative stride length). The **ggplot2** package is used for plotting.

The color gradient for the parameter is applied using scale_color_gradientn(). The color palette can be customized via the colours argument.

Value

A ggplot object showing the trajectories colored by the specified parameter.

Logo

Author(s)

Humberto G. Ferrón

humberto.ferron@uv.es

Macroevolution and Functional Morphology Research Group (www.macrofun.es)

Cavanilles Institute of Biodiversity and Evolutionary Biology

Calle Catedrático José Beltrán Martínez, nº 2

46980 Paterna - Valencia - Spain

Phone: +34 (9635) 44477

See Also

tps_to_track, velocity_track, scale_color_gradientn

Examples

```
# Example 1: Plot Trajectories Colored by Velocity with Default Settings (MountTom dataset)

# Hip heights for each track in the MountTom dataset
H_mounttom <- c(
  1.380, 1.404, 1.320, 1.736, 1.364, 1.432, 1.508, 1.768, 1.600, 1.848,
  1.532, 1.532, 0.760, 1.532, 1.688, 1.620, 0.636, 1.784, 1.676, 1.872,
  1.648, 1.760, 1.612
)

# Calculate velocities using the default Method "A"
V_mounttom <- velocity_track(MountTom, H = H_mounttom)

# Plot trajectories colored by velocity
plot1 <- plot_velocity(MountTom, V_mounttom, param = "V")
print(plot1)

# Example 2: Plot Trajectories Colored by Relative Stride Length with Default Settings
# (PaluxyRiver dataset)

# Hip heights for each track in the PaluxyRiver dataset
H_paluxyriver <- c(3.472, 2.200)

# Specify different methods for different tracks
Method_paluxyriver <- c("A", "B")

# Calculate velocities using specified methods
V_paluxyriver <- velocity_track(PaluxyRiver, H = H_paluxyriver, method = Method_paluxyriver)

# Plot trajectories colored by relative stride length
plot2 <- plot_velocity(PaluxyRiver, V_paluxyriver, param = "RSL")
print(plot2)

# Example 3: Plot Trajectories Colored by Velocity with Custom Line Width and Colors
# (MountTom dataset)

# Custom colors and line width
custom_colours <- c("blue", "green", "yellow", "red")
custom_lwd <- 2

# Plot trajectories with custom colors and line width
plot3 <- plot_velocity(MountTom, V_mounttom,
  param = "V", lwd = custom_lwd,
  colours = custom_colours
)
print(plot3)

# Example 4: Plot Trajectories Colored by Relative Stride Length with Custom Line Width
```

```
# and No Legend (PaluxyRiver dataset)

# Custom colors and line width
custom_colours_rsl <- c("purple", "orange", "pink", "gray")
custom_lwd_rsl <- 1.5

# Plot trajectories with custom colors, line width, and no legend
plot4 <- plot_velocity(PaluxyRiver, V_paluxyriver,
  param = "RSL", lwd = custom_lwd_rsl,
  colours = custom_colours_rsl, legend = FALSE
)
print(plot4)
```

`simil_DTW_metric` *Similarity metric using Dynamic Time Warping (DTW)*

Description

`simil_DTW_metric()` computes similarity metrics between two or more trajectories using Dynamic Time Warping (DTW). It allows for different superposition methods to align trajectories before calculating the DTW metric. The function also supports testing with simulations to calculate *p*-values for the DTW distance metrics.

Usage

```
simil_DTW_metric(data, test = NULL, sim = NULL, superposition = NULL)
```

Arguments

`data`	A `track` R object, which is a list consisting of two elements: • `Trajectories`: A list of interpolated trajectories, where each trajectory is a series of midpoints between consecutive footprints. • `Footprints`: A list of data frames containing footprint coordinates, metadata (e.g., image reference, ID), and a marker indicating whether the footprint is actual or inferred.
`test`	Logical; if `TRUE`, the function compares the observed DTW distances against simulated trajectories and calculates *p*-values. Default is `FALSE`.
`sim`	A `track simulation` R object consisting of a list of simulated trajectories to use for comparison when `test = TRUE`.
`superposition`	A character string indicating the method used to align trajectories. Options are `"None"`, `"Centroid"`, or `"Origin"`. Default is `"None"`.

Details

The simil_DTW_metric() function calculates the similarity between trajectories using the Dynamic Time Warping (DTW) algorithm from the **dtw** package. The dtw() function is used with the dist.method argument set to "Euclidean" for computing the local distances between points in the trajectories.

DTW aligns two time series by minimizing the cumulative distance between their points, creating an optimal alignment despite variations in length or temporal distortions. The algorithm constructs a distance matrix where each element represents the cost of aligning points between the two series and finds a warping path through this matrix that minimizes the total distance. The warping path is contiguous and monotonic, starting from the bottom-left corner and ending at the top-right corner (Cleasby et al., 2019).

DTW measures are non-negative and unbounded, with larger values indicating greater dissimilarity between the time series. This method has been used in various contexts, including ecological studies to analyze and cluster trajectory data (Cleasby et al., 2019).

Potential limitations and biases of DTW include sensitivity to noise and outliers, computational complexity, and the need for appropriate distance metrics. Additionally, DTW may not always account for all structural differences between trajectories and can be biased by the chosen alignment constraints. While DTW can handle trajectories of different lengths due to its elastic nature, having trajectories of similar lengths can improve the accuracy and interpretability of the similarity measure. Similar lengths result in a more meaningful alignment and can make the computation more efficient. When trajectories differ significantly in length, preprocessing or normalization might be necessary, and careful analysis is required to understand the alignment path. The function's flexibility in handling different lengths allows it to be applied in various contexts. However, large differences in trajectory lengths might introduce potential biases that should be considered when interpreting the results.

The function offers three different superposition methods to align the trajectories before DTW() is applied:

- "None": No superposition is applied.
- "Centroid": Trajectories are shifted to align based on their centroids.
- "Origin": Trajectories are shifted to align based on their starting point.

If test = TRUE, the function can compute *p*-values by comparing the observed DTW distances with those generated from a set of simulated trajectories. The *p*-values are calculated for both individual trajectory pairs and for the entire set of trajectories.

Value

A track similarity R object consisting ofa list containing the following elements:

DTW_distance_metric
: A matrix containing the pairwise DTW distances between trajectories.

DTW_distance_metric_p_values
: (If test is TRUE) A matrix containing the *p*-values for the pairwise DTW distances.

DTW_metric_p_values_combined
: (If test is TRUE) The overall *p*-value for the combined DTW distances.

`DTW_distance_metric_simulations`
: (If `test` is `TRUE`) A list of DTW distance matrices from each simulated dataset.

Logo

Author(s)

Humberto G. Ferrón

humberto.ferron@uv.es

Macroevolution and Functional Morphology Research Group (www.macrofun.es)

Cavanilles Institute of Biodiversity and Evolutionary Biology

Calle Catedrático José Beltrán Martínez, nº 2

46980 Paterna - Valencia - Spain

Phone: +34 (9635) 44477

References

Cleasby, I. R., Wakefield, E. D., Morrissey, B. J., Bodey, T. W., Votier, S. C., Bearhop, S., & Hamer, K. C. (2019). Using time-series similarity measures to compare animal movement trajectories in ecology. Behavioral Ecology and Sociobiology, 73, 1-19.

See Also

`tps_to_track`, `simulate_track`, `dtw`

Examples

```
# Example 1: Simulating tracks using the "Directed" model and comparing DTW distance
# in the PaluxyRiver dataset
s1 <- simulate_track(PaluxyRiver, nsim = 3, model = "Directed")
simil_DTW_metric(PaluxyRiver, test = TRUE, sim = s1, superposition = "None")

# Example 2: Simulating tracks using the "Constrained" model and comparing DTW distance
# in the PaluxyRiver dataset
s2 <- simulate_track(PaluxyRiver, nsim = 3, model = "Constrained")
simil_DTW_metric(PaluxyRiver, test = TRUE, sim = s2, superposition = "None")

# Example 3: Simulating tracks using the "Unconstrained" model and comparing DTW distance
# in the PaluxyRiver dataset
s3 <- simulate_track(PaluxyRiver, nsim = 3, model = "Unconstrained")
simil_DTW_metric(PaluxyRiver, test = TRUE, sim = s3, superposition = "None")

# Example 4: Simulating and comparing DTW distance in the MountTom dataset using the
# "Centroid" superposition method
sbMountTom <- subset_track(MountTom, tracks = c(1, 2, 3, 4, 7, 8, 9, 13, 15, 16, 18))
s4 <- simulate_track(sbMountTom, nsim = 3)
simil_DTW_metric(sbMountTom, test = TRUE, sim = s4, superposition = "Centroid")
```

```
# Example 5: Simulating and comparing DTW distance in the MountTom dataset using the
# "Origin" superposition method
sbMountTom <- subset_track(MountTom, tracks = c(1, 2, 3, 4, 7, 8, 9, 13, 15, 16, 18))
s5 <- simulate_track(sbMountTom, nsim = 3)
simil_DTW_metric(sbMountTom, test = TRUE, sim = s5, superposition = "Origin")
```

`simil_Frechet_metric` *Similarity metric using Fréchet distance*

Description

`simil_Frechet_metric()` computes similarity metrics between two or more trajectories using the Fréchet distance. It allows for different superposition methods to align trajectories before calculating the Fréchet distance metrics. The function also supports testing with simulations to calculate *p*-values for the Fréchet distance metrics.

Usage

```
simil_Frechet_metric(data, test = FALSE, sim = NULL, superposition = "None")
```

Arguments

`data`	A `track` R object, which is a list consisting of two elements: • `Trajectories`: A list of interpolated trajectories, where each trajectory is a series of midpoints between consecutive footprints. • `Footprints`: A list of data frames containing footprint coordinates, metadata (e.g., image reference, ID), and a marker indicating whether the footprint is actual or inferred.
`test`	Logical; if `TRUE`, the function compares the observed Fréchet distances against simulated trajectories and calculates *p*-values. Default is `FALSE`.
`sim`	A `track simulation` R object consisting of a list of simulated trajectories to use for comparison when `test = TRUE`.
`superposition`	A character string indicating the method used to align trajectories. Options are `"None"`, `"Centroid"`, or `"Origin"`. Default is `"None"`.

Details

The `simil_Frechet_metric()` function calculates the similarity between trajectories using the `Frechet()` function from the **SimilarityMeasures** package.

The Fréchet distance is a measure of similarity between two curves or continuous trajectories, which takes into account both the order and location of points within the trajectories (Besse et al. 2015). The distance can be described by the analogy of a person walking a dog on an extendable leash (Aronov et al. 2006). Both the person and the dog move along their respective trajectories, with each able to adjust their speed but not retrace their steps. The Fréchet distance is the minimum leash length required to keep the dog connected to the person throughout the walk (Cleasby et al., 2019).

Unlike other trajectory comparison techniques, such as Dynamic Time Warping, the Fréchet distance focuses on the overall shape of the trajectories rather than matching specific points. As a result, it is sensitive to noise because all points of the compared trajectories are considered in its calculation. However, it can still be a powerful tool for trajectory clustering and comparison, particularly when shape is the primary concern (Cleasby et al., 2019).

Note that when comparing real trajectories that are very disparate or those simulated under an unconstrained method, the resulting trajectories may not be suitable for Fréchet distance calculations. In such cases, the Fréchet distance is returned as -1 to indicate an invalid measurement.

The function offers three different superposition methods to align the trajectories before `Frechet()` is applied:

- `"None"`: No superposition is applied.
- `"Centroid"`: Trajectories are shifted to align based on their centroids.
- `"Origin"`: Trajectories are shifted to align based on their starting point.

If `test = TRUE`, the function can compute *p*-values by comparing the observed Fréchet distances with those generated from a set of simulated trajectories. The *p*-values are calculated for both individual trajectory pairs and for the entire set of trajectories.

Value

A `track similarity` R object consisting ofa list containing the following elements:

`Frechet_distance_metric`
: A matrix containing the pairwise Frechet distances between trajectories.

`Frechet_distance_metric_p_values`
: (If `test` is `TRUE`) A matrix containing the *p*-values for the pairwise Frechet distances.

`Frechet_metric_p_values_combined`
: (If `test` is `TRUE`) The overall *p*-value for the combined Frechet distances.

`Frechet_distance_metric_simulations`
: (If `test` is `TRUE`) A list of Frechet distance matrices from each simulated dataset.

Logo

Author(s)

Humberto G. Ferrón

humberto.ferron@uv.es

Macroevolution and Functional Morphology Research Group (www.macrofun.es)

Cavanilles Institute of Biodiversity and Evolutionary Biology

Calle Catedrático José Beltrán Martínez, nº 2

46980 Paterna - Valencia - Spain

Phone: +34 (9635) 44477

References

Cleasby, I. R., Wakefield, E. D., Morrissey, B. J., Bodey, T. W., Votier, S. C., Bearhop, S., & Hamer, K. C. (2019). Using time-series similarity measures to compare animal movement trajectories in ecology. Behavioral Ecology and Sociobiology, 73, 1-19.

See Also

tps_to_track, simulate_track, Frechet

Examples

```
# Example 1: Simulating tracks using the "Directed" model and comparing Frechet distance
# in the PaluxyRiver dataset
s1 <- simulate_track(PaluxyRiver, nsim = 3, model = "Directed")
simil_Frechet_metric(PaluxyRiver, test = TRUE, sim = s1, superposition = "None")

# Example 2: Simulating tracks using the "Constrained" model and comparing Frechet distance
# in the PaluxyRiver dataset  using the "Centroid" superposition method
s2 <- simulate_track(PaluxyRiver, nsim = 3, model = "Constrained")
simil_Frechet_metric(PaluxyRiver, test = TRUE, sim = s2, superposition = "Centroid")

# Example 3: Simulating tracks using the "Unconstrained" model and comparing Frechet distance
# in the PaluxyRiver dataset using the "Origin" superposition method
s3 <- simulate_track(PaluxyRiver, nsim = 3, model = "Unconstrained")
simil_Frechet_metric(PaluxyRiver, test = TRUE, sim = s3, superposition = "Origin")
```

---

simulate_track *Simulate tracks using different models*

Description

simulate_track() simulates movement trajectories based on an original set of tracks. Three movement models are available for simulation, each reflecting different levels of constraint in movement patterns. These models can represent biological or environmental constraints, such as movement along coastlines, rivers, or towards resources like water or food.

Usage

```
simulate_track(data, nsim = NULL, model = NULL)
```

Arguments

data	A track R object, which is a list consisting of two elements:

- Trajectories: A list of interpolated trajectories, where each trajectory is a series of midpoints between consecutive footprints.

- `Footprints`: A list of data frames containing footprint coordinates, metadata (e.g., image reference, ID), and a marker indicating whether the footprint is actual or inferred.

`nsim` The number of simulations to run. Defaults to `1000` if not specified.

`model` The type of movement model to use. Options are `"Directed"`, `"Constrained"`, or `"Unconstrained"`. Defaults to `"Unconstrained"` if not provided.

Details

This function simulates movement trajectories based on the following models:

- **Directed**: This model simulates movement that follows a specific direction navigating with a compass (i.e., a directed walk/allothetic directed walk/oriented path) (Cheung et al., 2007, 2008). The trajectory is constrained by both the angular and linear properties of the original track, with minor deviations allowed to reflect natural variability.
  - **Angular constraints**: The trajectory closely follows a specific direction, maintaining the overall angular orientation of the original track. Deviations of consecutive steps are governed by the angular standard deviation calculated from the original track using `TrajAngles()`.
  - **Linear constraints**: Step lengths are constrained to the mean of the original track's step lengths, with variability allowed according to the standard deviation of step lengths computed with `TrajStepLengths()`.
  - **Starting direction**: Fixed to the original direction (overall angular orientation) of the track.

This model is ideal for simulating movement directed toward a specific resource or constrained by natural barriers, with a relatively fixed direction and minor deviations.

- **Constrained**: This model simulates movement that correspond to a correllated random walk/idiothetic directed walk (Kareiva & Shigesada, 1983), corresponding to an animal navigating without a compass (Cheung et al., 2008), while still maintaining certain angular and linear characteristics of the original track. It provides more flexibility than the Directed model but is not entirely random like the Unconstrained model.
  - **Angular constraints**: The trajectory does not follow a specific direction. Deviations of consecutive steps are governed by the angular standard deviation calculated from the original track using `TrajAngles()`.
  - **Linear constraints**: Step lengths are constrained to the mean of the original track's step lengths, with variability allowed according to the standard deviation of step lengths computed with `TrajStepLengths()`.
  - **Starting direction**: Fixed to the original direction (overall angular orientation) of the track.

This model is suitable for scenarios where movement is influenced by external constraints but allows for some degree of random exploration.

- **Unconstrained**: This model simulates movement that correspond to a correllated random walk/idiothetic directed walk (Kareiva & Shigesada, 1983), corresponding to an animal navigating without a compass (Cheung et al., 2008), while still maintaining certain angular and linear characteristics of the original track.

- **Angular constraints**: The trajectory does not follow a specific direction. Deviations of consecutive steps are governed by the angular standard deviation calculated from the original track using `TrajAngles()`.
- **Linear constraints**: Step lengths are constrained to the mean of the original track's step lengths, with variability allowed according to the standard deviation of step lengths computed with `TrajStepLengths()`.
- **Starting direction**: Randomly determined.

This model is suitable for simulating exploratory or dispersal behavior in open environments, where movement is random and not influenced by specific constraints.

Note: Simulations cannot be applied to trajectories with fewer than four steps as the standard deviations of angles and step lengths cannot be computed for such short trajectories. Consider using the `subset_track()` function to filter tracks with four or more steps.

The function utilizes the **trajr** package for key calculations:

- `TrajGenerate()`: Generates a new trajectory based on random or directed movement models, constrained by specified parameters.
- `TrajStepLengths()`: Calculates the step lengths (distances between consecutive points) of the original trajectory.
- `TrajAngles()`: Computes the angles between consecutive segments of the trajectory, used to maintain directional movement in constrained models.
- `TrajRotate()`: Rotates the trajectory by a specified angle to match the original direction or a random angle for unconstrained models.
- `TrajTranslate()`: Translates the simulated trajectory to start at the same geographic location as the original.

The `NISTdegTOradian()` function from the **NISTunits** package is used to convert angles from degrees to radians.

Value

A `track simulation` R object consisting of a list of simulated trajectories stored as `track` R objects.

Logo

Author(s)

Humberto G. Ferrón

humberto.ferron@uv.es

Macroevolution and Functional Morphology Research Group (www.macrofun.es)

Cavanilles Institute of Biodiversity and Evolutionary Biology

Calle Catedrático José Beltrán Martínez, nº 2

46980 Paterna - Valencia - Spain

Phone: +34 (9635) 44477

References

Cheung, A., Zhang, S., Stricker, C., & Srinivasan, M. V. (2007). Animal navigation: the difficulty of moving in a straight line. Biological cybernetics, 97, 47-61.

Cheung, A., Zhang, S., Stricker, C., & Srinivasan, M. V. (2008). Animal navigation: general properties of directed walks. Biological cybernetics, 99, 197-217.

See Also

tps_to_track, plot_sim, subset_track, TrajGenerate, TrajStepLengths, TrajAngles, TrajRotate, TrajTranslate, NISTdegTOradian

Examples

```
# Example 1: Simulate tracks using data from the Paluxy River
# Default model (Unconstrained movement)
simulated_tracks <- simulate_track(PaluxyRiver, nsim = 3)

# Example 2: Simulate tracks using the "Directed" model, representing movement
# toward a resource (e.g., water source)
simulated_tracks_directed <- simulate_track(PaluxyRiver, nsim = 3, model = "Directed")

# Example 3: Simulate tracks using the "Constrained" model, representing movement
# along a geographic feature (e.g., coastline)
simulated_tracks_constrained <- simulate_track(PaluxyRiver, nsim = 3, model = "Constrained")

# Example 4: Simulate tracks using the "Unconstrained" model (random exploratory movement)
simulated_tracks_unconstrained <- simulate_track(PaluxyRiver, nsim = 3, model = "Unconstrained")

# Subsetting trajectories with four or more steps in the MountTom dataset
sbMountTom <- subset_track(MountTom, tracks = c(1, 2, 3, 4, 7, 8, 9, 13, 15, 16, 18))

# Example 5: Simulate tracks using data from Mount Tom
# Default model (Unconstrained movement)
simulated_tracks_mt <- simulate_track(sbMountTom, nsim = 3)

# Example 6: Simulate tracks using the "Directed" model for Mount Tom, representing
# directed movement
simulated_tracks_mt_directed <- simulate_track(sbMountTom, nsim = 3, model = "Directed")

# Example 7: Simulate tracks using the "Constrained" model for Mount Tom, representing
# constrained movement
simulated_tracks_mt_constrained <- simulate_track(sbMountTom, nsim = 3, model = "Constrained")

# Example 8: Simulate tracks using the "Unconstrained" model for Mount Tom, representing
# random exploratory movement
simulated_tracks_mt_unconstrained <- simulate_track(sbMountTom, nsim = 3, model = "Unconstrained")
```

subset_track *Subset tracks*

Description

subset_track() is a function that subsets tracks from a list of track data based on the specified indices.

Usage

```
subset_track(data, tracks = NULL)
```

Arguments

data	A track R object, which is a list consisting of two elements: • Trajectories: A list of interpolated trajectories, where each trajectory is a series of midpoints between consecutive footprints. • Footprints: A list of data frames containing footprint coordinates, metadata (e.g., image reference, ID), and a marker indicating whether the footprint is actual or inferred.
tracks	A numeric vector specifying the indices of tracks to subset. The default is to include all tracks.

Details

This function subsets both the Trajectories and Footprints elements of the input data based on the provided vector of indices. It allows users to focus on a specific subset of tracks for further analysis or visualization, particularly when working with large datasets containing numerous tracks.

Value

A track R object that contains only the specified subset of tracks. The structure of the returned object mirrors the input structure but includes only the selected tracks.

Logo

Author(s)

Humberto G. Ferrón

humberto.ferron@uv.es

Macroevolution and Functional Morphology Research Group (www.macrofun.es)

Cavanilles Institute of Biodiversity and Evolutionary Biology

Calle Catedrático José Beltrán Martínez, nº 2

46980 Paterna - Valencia - Spain

Phone: +34 (9635) 44477

See Also

tps_to_track

Examples

```
# Example 1: Subset the first three tracks of MountTom dataset.
subset_data <- subset_track(MountTom, tracks = c(1:3))

# Example 2:  Subset the tracks at indices 5, 7, and 10.
subset_data <- subset_track(MountTom, tracks = c(5, 7, 10))
```

---

test_direction *Test for differences in direction means with pairwise comparisons*

Description

test_direction() evaluates differences in mean direction across different tracks using a specified statistical test. It includes options for ANOVA, Kruskal-Wallis test, and Generalized Linear Models (GLM), and checks for assumptions such as normality and homogeneity of variances. For datasets with more than two tracks, it performs pairwise comparisons to identify specific differences between tracks.

Usage

```
test_direction(data, analysis = NULL)
```

Arguments

data	A track R object, which is a list consisting of two elements: • Trajectories: A list of interpolated trajectories, where each trajectory is a series of midpoints between consecutive footprints. • Footprints: A list of data frames containing footprint coordinates, metadata (e.g., image reference, ID), and a marker indicating whether the footprint is actual or inferred.
analysis	A character string specifying the type of analysis: "ANOVA", "Kruskal-Wallis", or "GLM". Default is "ANOVA".

Details

The test_direction() function performs the following operations:

- **Condition Testing:**
  - **Normality:** Shapiro-Wilk test for normality on step direction data within each track.
  - **Homogeneity of Variances:** Levene's test for equal variances across tracks.
- **Statistical Analysis:**

  - **ANOVA:** Compares step mean directions across tracks, assuming normality and homogeneity of variances. Includes Tukey's HSD post-hoc test for pairwise comparisons.
  - **Kruskal-Wallis Test:** Non-parametric alternative to ANOVA for comparing step median directions across tracks when assumptions are violated. Includes Dunn's test for pairwise comparisons.
  - **GLM:** Generalized Linear Model with a Gaussian family for comparing step means if ANOVA assumptions are not met. Pairwise comparisons in the GLM are conducted using estimated marginal means (least-squares means) with the **emmeans** package, which computes differences between group means while adjusting for multiple comparisons using Tukey's method.
- **Direction Measurement:**
  - The direction is measured in degrees. The reference direction, or 0 degrees, is considered to be along the positive x-axis. Angles are measured counterclockwise from the positive x-axis, with 0 degrees pointing directly along this axis.

Value

A list with the results of the statistical analysis and diagnostic tests:

- `normality_results`: A matrix of test statistics and *p*-values from the Shapiro-Wilk test for each track, with rows for the test statistic and *p*-value, and columns for each track.
- `homogeneity_test`: The result of Levene's test, including the *p*-value for homogeneity of variances.
- `ANOVA` (If `analysis` is `"ANOVA"`): A list containing the ANOVA table and Tukey HSD post-hoc test results.
- `Kruskal_Wallis` (If `analysis` is `"Kruskal-Wallis"`): A list containing the Kruskal-Wallis test result and Dunn's test post-hoc results.
- `GLM` (If `analysis` is `"GLM"`): A summary of the GLM fit and pairwise comparisons.

Logo

Author(s)

Humberto G. Ferrón

humberto.ferron@uv.es

Macroevolution and Functional Morphology Research Group (www.macrofun.es)

Cavanilles Institute of Biodiversity and Evolutionary Biology

Calle Catedrático José Beltrán Martínez, nº 2

46980 Paterna - Valencia - Spain

Phone: +34 (9635) 44477

See Also

`tps_to_track`, `plot_direction`

Examples

```
# Example 1: Test for Differences in Direction Means with Pairwise Comparisons in MountTom dataset
test_direction(MountTom, analysis = "ANOVA")

# Example 2: Test for Differences in Direction Means with Pairwise Comparisons in MountTom dataset
test_direction(MountTom, analysis = "Kruskal-Wallis")

# Example 3: Test for Differences in Direction Means with Pairwise Comparisons in MountTom dataset
test_direction(MountTom, analysis = "GLM")

# Example 4: Test for Differences in Direction Means with Pairwise Comparisons in PaluxyRiver
# dataset
test_direction(PaluxyRiver, analysis = "ANOVA")

# Example 5: Test for Differences in Direction Means with Pairwise Comparisons in PaluxyRiver
# dataset
test_direction(PaluxyRiver, analysis = "Kruskal-Wallis")

# Example 6: Test for Differences in Direction Means with Pairwise Comparisons in PaluxyRiver
# dataset
test_direction(PaluxyRiver, analysis = "GLM")
```

---

test_velocity *Test for differences in velocity means with pairwise comparisons*

Description

`test_velocity()` evaluates differences in velocity means across different tracks using a specified statistical test. It includes options for ANOVA, Kruskal-Wallis test, and Generalized Linear Models (GLM), and checks for assumptions such as normality and homogeneity of variances. For datasets with more than two tracks, it performs pairwise comparisons to identify specific differences between tracks.

Usage

```
test_velocity(data, trackvel, plot = FALSE, analysis = NULL)
```

Arguments

`data`	A `track` R object, which is a list consisting of two elements: • `Trajectories`: A list of interpolated trajectories, where each trajectory is a series of midpoints between consecutive footprints. • `Footprints`: A list of data frames containing footprint coordinates, metadata (e.g., image reference, ID), and a marker indicating whether the footprint is actual or inferred.
`trackvel`	A `track velocity` R object consisting of a list where each element corresponds to a track and contains velocity or relative stride length data.

plot	A logical value indicating whether to plot a boxplot of velocities by track (default is FALSE).
analysis	A character string specifying the type of analysis: "ANOVA", "Kruskal-Wallis", or "GLM". Default is "ANOVA".

Details

The test_velocity function performs the following operations:

- **Condition Testing:**
  - **Normality:** Shapiro-Wilk test for normality on velocity data within each track.
  - **Homogeneity of Variances:** Levene's test for equal variances across tracks.
- **Statistical Analysis:**
  - **ANOVA:** Compares mean velocities across tracks, assuming normality and homogeneity of variances. Includes Tukey's HSD post-hoc test for pairwise comparisons.
  - **Kruskal-Wallis Test:** Non-parametric alternative to ANOVA for comparing median velocities across tracks when assumptions are violated. Includes Dunn's test for pairwise comparisons.
  - **GLM:** Generalized Linear Model with a Gaussian family for comparing means if ANOVA assumptions are not met. Pairwise comparisons in the GLM are conducted using estimated marginal means (least-squares means) with the **emmeans** package, which computes differences between group means while adjusting for multiple comparisons using Tukey's method.
- **Plotting:**
  - If plot is TRUE, a boxplot of velocities by track is generated.

Value

A list with the results of the statistical analysis and diagnostic tests:

- normality_results: A matrix of test statistics and *p*-values from the Shapiro-Wilk test for each track, with rows for the test statistic and *p*-value, and columns for each track.
- homogeneity_test: The result of Levene's test, including the *p*-value for homogeneity of variances.
- ANOVA (If analysis is "ANOVA"): A list containing the ANOVA table and Tukey HSD post-hoc test results.
- Kruskal_Wallis (If analysis is "Kruskal-Wallis"): A list containing the Kruskal-Wallis test result and Dunn's test post-hoc results.
- GLM (If analysis is "GLM"): A summary of the GLM fit and pairwise comparisons.
- plot (If plot is TRUE): A boxplot of velocities by track is generated and displayed.

Logo

Author(s)

Humberto G. Ferrón

humberto.ferron@uv.es

Macroevolution and Functional Morphology Research Group (www.macrofun.es)

Cavanilles Institute of Biodiversity and Evolutionary Biology

Calle Catedrático José Beltrán Martínez, nº 2

46980 Paterna - Valencia - Spain

Phone: +34 (9635) 44477

See Also

tps_to_track, velocity_track

Examples

```
# Example 1: Test for Differences in Velocity Means with Pairwise Comparisons in Trajectories
# in MountTom dataset.

# Hip heights for each track in the MountTom dataset
H_mounttom <- c(
  1.380, 1.404, 1.320, 1.736, 1.364, 1.432, 1.508, 1.768, 1.600, 1.848,
  1.532, 1.532, 0.760, 1.532, 1.688, 1.620, 0.636, 1.784, 1.676, 1.872,
  1.648, 1.760, 1.612
)

# Calculate velocities using the default Method "A"
V_mounttom <- velocity_track(MountTom, H = H_mounttom)

# Test for Differences in Velocity Means with Pairwise Comparisons
test_velocity(MountTom, V_mounttom)

# Example 2: Test for Differences in Velocity Means with Pairwise Comparisons in Trajectories
# in PaluxyRiver dataset.

# Hip heights for each track in the PaluxyRiver dataset
H_paluxyriver <- c(3.472, 2.200)

# Specify different methods for different tracks
Method_paluxyriver <- c("A", "B")

# Calculate velocities using specified methods
V_paluxyriver <- velocity_track(PaluxyRiver, H = H_paluxyriver, method = Method_paluxyriver)

# Test for Differences in Velocity Means with Pairwise Comparisons
test_velocity(PaluxyRiver, V_paluxyriver)
```

tps_to_track *Transform a *.tps file into a* track *R object*

Description

tps_to_track() reads a *.tps file containing footprint coordinates of one or several tracks and transforms it into a track R object.

Usage

```
tps_to_track(file, scale = NULL, missing = FALSE, NAs = NULL, R.L.side = NULL)
```

Arguments

file	A *.tps file containing (x,y) coordinates of footprints in tracks.
scale	A numeric value specifying the scale in meters per pixel.
missing	A logical value indicating whether there are missing footprints in any track to be interpolated: TRUE, or FALSE (the default).
NAs	A matrix with two columns indicating which missing footprints will be interpolated. The first column gives the number of the track containing missing footprints, and the second column gives the number of the footprint that is missing within this track. The number of rows is equal to the total number of missing footprints in the sample.
R.L.side	A character vector specifying the side of the first footprint of each track. Only needed if missing is set to TRUE. The length of the vector must be equal to the total number of tracks in the sample. • "L": first footprint corresponds to the left foot. • "R": first footprint corresponds to the right foot.

Details

It is highly recommended that the *.tps file is built using the TPS software (Rohlf 2008, 2009). Tracks with a different number of footprints (i.e., landmarks) are allowed. This function transforms the coordinates of the footprints of each track into a set of trajectory coordinates. Each point of the trajectory is calculated as:

$$Point_i(x, y) = (Footprint_i(x, y) + Footprint_{i+1}(x, y)/2$$

The number of points of the resulting trajectory is $n_{footprints} - 1$.

If missing is set to TRUE, missing footprints can be interpolated. This interpolation is based on adjacent footprints and the provided side information.

Value

A `track` R object, which is a list consisting of two elements:

- `Trajectories`: A list of interpolated trajectories, where each trajectory is a series of midpoints between consecutive footprints.
- `Footprints`: A list of data frames containing footprint coordinates, metadata (e.g., image reference, ID), and a marker indicating whether the footprint is actual or inferred.

Logo

Author(s)

Humberto G. Ferrón

humberto.ferron@uv.es

Macroevolution and Functional Morphology Research Group (www.macrofun.es)

Cavanilles Institute of Biodiversity and Evolutionary Biology

Calle Catedrático José Beltrán Martínez, nº 2

46980 Paterna - Valencia - Spain

Phone: +34 (9635) 44477

References

Farlow, J. O., O'Brien, M., Kuban, G. J., Dattilo, B. F., Bates, K. T., Falkingham, P. L., & Piñuela, L. (2012). Dinosaur Tracksites of the Paluxy River Valley (Glen Rose Formation, Lower Cretaceous), Dinosaur Valley State Park, Somervell County, Texas. In Proceedings of the V International Symposium about Dinosaur Palaeontology and their Environment (pp. 41-69). Burgos: Salas de los Infantes.

Ostrom, J. H. (1972). Were some dinosaurs gregarious?. Palaeogeography, Palaeoclimatology, Palaeoecology, 11(4), 287-301.

Rohlf, F. J. 2008. TPSUTIL. Version 1.40. Department of Ecology and Evolution, State University of New York. Available at https://sbmorphometrics.org/.

Rohlf, F. J. 2009. tpsDig. Version 2.14. Department of Ecology and Evolution, State University of New York. Available at https://sbmorphometrics.org/.

Examples

```
# Example 1: Tracks without missing footprints. Based on the Paluxy River
# dinosaur chase sequence (Farlow et al., 2011).

# Load the example TPS file provided in the QuAnTeTrack package.
# This TPS data includes footprint coordinates for different tracks,
# along with associated metadata.
tpsPaluxyRiver <- system.file("extdata", "PaluxyRiver.tps", package = "QuAnTeTrack")

# Call the tps_to_track function to convert the TPS data in the file
```

```
# into a track object. The 'scale' argument sets the scaling factor
# for the coordinates, and 'missing=FALSE' indicates that no landmarks
# are missing in the dataset.
tps_to_track(tpsPaluxyRiver, scale = 0.004341493, missing = FALSE, NAs = NULL)


# Example 2: Tracks with missing footprints. Based on dinosaur tracks from
# the Mount Tom (Ostrom, 1972).

# Load the example TPS file provided in the QuAnTeTrack package.
# This TPS data includes footprint coordinates for different tracks,
# along with associated metadata.
tpsMountTom <- system.file("extdata", "MountTom.tps", package = "QuAnTeTrack")

# Define a matrix representing the footprints that are missing from the dataset.
# In this example, the matrix 'NAs' specifies that footprint 7 is missing in track 3.
NAs <- matrix(c(7, 3), nrow = 1, ncol = 2)

# Call the tps_to_track function, which will convert the TPS data in the file
# to a track object. The 'scale' argument sets the scaling factor for the coordinates,
# 'missing' specifies whether missing footprints should be handled, 'NAs' provides
# the missing footprints matrix, and 'R.L.side' specifies which side (Right or Left)
# is the first footprint of each track.
tps_to_track(tpsMountTom,
  scale = 0.004411765, missing = TRUE, NAs = NAs,
  R.L.side = c(
    "R", "L", "L", "L", "R", "L", "R", "R", "L", "L", "L",
    "L", "L", "R", "R", "L", "R", "R", "L", "R", "R",
    "R", "R"
  )
)
```

`track_intersection` *Calculate intersection metrics in tracks*

Description

`track_intersection()` calculates the number of unique intersections between trajectories. The function also supports testing with simulations and different permutation procedures for the coordinates of the simulated trajectories' origins to compute *p*-values. This allows for a robust assessment of the intersection metrics, enabling users to evaluate the significance of the observed intersections in relation to simulated trajectories.

Usage

```
track_intersection(
  data,
  test = NULL,
  H1 = NULL,
```

```
  sim = NULL,
  origin.permutation = NULL,
  custom.coord = NULL
)
```

Arguments

data	A track R object, which is a list consisting of two elements: • Trajectories: A list of interpolated trajectories, where each trajectory is a series of midpoints between consecutive footprints. • Footprints: A list of data frames containing footprint coordinates, metadata (e.g., image reference, ID), and a marker indicating whether the footprint is actual or inferred.
test	Logical; if TRUE, the function compares the observed intersection metrics against. Default is FALSE.
H1	A character string specifying the alternative hypothesis to be tested. Options are "Lower" for testing whether the observed intersections are significantly lower than the simulated ones (e.g., coordinated or gregarious movement), or "Higher" for testing whether the observed intersections are significantly higher than the simulated ones (e.g., predatory or chasing events).
sim	A track simulation R object consisting of a list of simulated trajectories to use for comparison when test = TRUE.
origin.permutation	A character string specifying the method for permutation of the coordinates of the simulated trajectories' origins. Options include "None", "Min.Box", "Conv.Hull", or "Custom". Default is "None".
custom.coord	A matrix of custom coordinates that define the vertices of an area for permutation of the coordinates of the simulated trajectories' origins.

Details

The track_intersection() function is designed to calculate the number of unique intersections between trajectories and to evaluate their statistical significance through hypothesis testing based on simulated tracks. This process provides a robust framework for comparing observed intersections against those expected under random conditions, allowing users to test specific behavioral hypotheses related to the movement patterns of trackmakers.

Hypothesis testing is controlled by the H1 argument, which defines the **alternative hypothesis** to be evaluated. This argument is crucial for interpreting the statistical results, as it determines whether the function will test for **reduced** or **increased** intersection counts compared to simulated trajectories.

The H1 argument accepts two possible values:

- "Lower": This option tests whether the observed intersections are significantly lower than those generated by simulations. This scenario corresponds to hypotheses involving **coordinated or gregarious movement**, where animals moving in parallel or in a group (e.g., hunting packs or social gatherings) would produce **fewer intersections** than expected under random conditions.

- "Higher": This option tests whether the observed intersections are significantly higher than those generated by simulations. It applies to scenarios where **predatory or chasing interactions** are likely, such as when one trackmaker follows or crosses the path of another. This behavior results in **more intersections** than what would occur randomly.

The selection of the H1 argument must be consistent with the behavioral hypothesis being tested. For example, use "Lower" when investigating group movement or cooperative behavior, and "Higher" when analyzing predatory or competitive interactions. The function will automatically adjust the calculation of *p*-values to reflect the selected H1. If the argument is left NULL, an error will be triggered, indicating that users must explicitly specify the hypothesis to be tested.

The interpretation of the **combined *p*-value** returned by the function is directly influenced by the choice of H1, as it determines whether the statistical comparison aims to detect a **reduction** or an **increase** in intersection counts compared to the simulated dataset.

In addition to hypothesis testing, the track_intersection() function offers several options for altering the initial positions of simulated tracks through the origin.permutation argument. The available options include:

- "None": Simulated trajectories are not shifted and are compared based on their original starting positions.
- "Min.Box": Trajectories are randomly placed within the **minimum bounding box** surrounding the original starting points.
- "Conv.Hull": Trajectories are placed within the **convex hull** that encompasses all original starting points, providing a more precise representation of the area occupied by the tracks.
- "Custom": Allows users to define a specific region of interest by providing a matrix of coordinates (custom.coord) that specifies the vertices of the desired area. This option is particularly useful when certain spatial features or environmental conditions are known to constrain movement.

The choice of origin.permutation should reflect the nature of the behavioral hypothesis being tested. For example, using "None" is most appropriate when testing how intersections compare under scenarios where trackmakers originate from specific locations. In contrast, options like "Min.Box", "Conv.Hull", or "Custom" are suitable when evaluating how intersections would differ if the tracks originated from a broader or predefined area.

The track_intersection() function also allows for integration with similarity metrics computed using simil_DTW_metric() and simil_Frechet_metric(). This combination of intersection counts and similarity metrics can provide a more comprehensive analysis of how trackmakers interacted, whether their movements were coordinated or independent, and whether their interactions were consistent with the hypothesized behavioral patterns.

Overall, the selection of H1 and origin.permutation should be carefully considered in light of the specific hypotheses being tested. By combining intersection metrics with similarity measures, users can obtain a deeper understanding of the behavioral dynamics underlying the observed trackways.

Value

A track intersection R object consisting of a list containing the following elements:

`Intersection_metric`
: A matrix of unique intersection counts between trajectories. Each entry represents the number of unique intersection points between the corresponding pair of trajectories.

`Intersection_metric_p_values`
: (If `test = TRUE`) A matrix of *p*-values associated with the intersection metrics, calculated through permutations of simulated trajectory origins. Each entry reflects the probability of observing an intersection count as extreme as the observed one, given the null hypothesis of no difference.

`Intersection_metric_p_values_combined`
: (If `test = TRUE`) A numeric value representing the combined *p*-value for all intersections, indicating the overall significance of the intersection metrics across all pairs of trajectories.

`Intersection_metric_simulations`
: (If `test = TRUE`) A list containing matrices of intersection counts for each simulation iteration, allowing for further inspection of the distribution of intersections across multiple randomized scenarios.

Logo

Author(s)

Humberto G. Ferrón

humberto.ferron@uv.es

Macroevolution and Functional Morphology Research Group (www.macrofun.es)

Cavanilles Institute of Biodiversity and Evolutionary Biology

Calle Catedrático José Beltrán Martínez, nº 2

46980 Paterna - Valencia - Spain

Phone: +34 (9635) 44477

See Also

`tps_to_track`, `simulate_track`, `simil_DTW_metric`, `simil_Frechet_metric`

Examples

```
# Example 1: Intersection metrics in the PaluxyRiver dataset.
s1 <- simulate_track(PaluxyRiver, nsim = 5, model = "Directed")
int1 <- track_intersection(PaluxyRiver, test = TRUE, H1 = "Lower", sim = s1,
  origin.permutation = "None")
print(int1)

# Example 2: Using "Min.Box" origin permutation in PaluxyRiver dataset.
s2 <- simulate_track(PaluxyRiver, nsim = 5, model = "Constrained")
int2 <- track_intersection(PaluxyRiver, test = TRUE, H1 = "Lower", sim = s2,
  origin.permutation = "Min.Box")
```

```
print(int2)

# Example 3: Using "Conv.Hull" origin permutation in PaluxyRiver dataset.
s3 <- simulate_track(PaluxyRiver, nsim = 5, model = "Unconstrained")
int3 <- track_intersection(PaluxyRiver, test = TRUE, H1 = "Lower", sim = s3,
  origin.permutation = "Conv.Hull")
print(int3)

# Example 4: Using "Min.Box" origin permutation in MountTom subset.
sbMountTom <- subset_track(MountTom, tracks = c(1, 2, 3, 4, 7, 8, 9, 13, 15, 16, 18))
s4 <- simulate_track(sbMountTom, nsim = 5)
int4 <- track_intersection(sbMountTom, test = TRUE, H1 = "Higher", sim = s4,
  origin.permutation = "Min.Box")
print(int4)

# Example 5: Customized origin permutation in MountTom subset.
sbMountTom <- subset_track(MountTom, tracks = c(1, 2, 3, 4, 7, 8, 9, 13, 15, 16, 18))
s5 <- simulate_track(sbMountTom, nsim = 5)
area_origin <- matrix(c(50, 5, 10, 5, 10, 20, 50, 20), ncol = 2, byrow = TRUE)
int5 <- track_intersection(sbMountTom, test = TRUE, H1 = "Higher", sim = s5,
  origin.permutation = "Custom", custom.coord = area_origin)
print(int5)
```

`track_param` *Print track parameters*

Description

`track_param()` is a function to compute and print various parameters of tracks from a list of track data.

Usage

```
track_param(data)
```

Arguments

`data`	A `track` R object, which is a list consisting of two elements: • `Trajectories`: A list of interpolated trajectories, where each trajectory is a series of midpoints between consecutive footprints. • `Footprints`: A list of data frames containing footprint coordinates, metadata (e.g., image reference, ID), and a marker indicating whether the footprint is actual or inferred.

Details

This function calculates various movement parameters for each track in the provided data. It uses the following helper functions from the **trajr** (Animal Trajectory Analysis) package:

- `TrajAngles()`: Calculates the turning angles of the track.
- `TrajDistance()`: Calculates the total distance covered by the track.
- `TrajLength()`: Calculates the length of the track.
- `TrajStepLengths()`: Calculates the step lengths of the track.
- `TrajSinuosity2()`: Calculates the sinuosity of the track.
- `TrajStraightness()`: Calculates the straightness of the track.

Value

A list of lists, where each sublist contains the computed parameters for a corresponding track. The parameters included are:

- `Turning_angles`: A vector of turning angles for the track (in degrees).
- `Mean_turning_angle`: The mean of the turning angles (in degrees).
- `Standard_deviation_turning_angle`: The standard deviation of the turning angles (in degrees).
- `Distance`: The total distance covered by the track (in meters).
- `Length`: The length of the track (in meters).
- `Step_lengths`: A vector of step lengths for the track (in meters).
- `Mean_step_length`: The mean of the step lengths (in meters).
- `Standard_deviation_step_length`: The standard deviation of the step lengths (in meters).
- `Sinuosity`: The sinuosity of the track (dimensionless).
- `Straightness`: The straightness of the track (dimensionless).

The reference direction, or 0 degrees, is considered to be along the positive x-axis. This means that angles are measured counterclockwise from the positive x-axis, with 0 degrees (or 0 degrees) pointing directly along this axis. For a detailed explanation and appropriate methods for analyzing circular data, refer to Batschelet (1981).

Sinuosity is calculated according to Benhamou (2004), as defined in equation 8. The formula used here is a refined version of the sinuosity index presented by Bovet & Benhamou (1988), which is applicable to a broader range of turning angle distributions and does not require a constant step length.

The sinuosity is computed using the formula:

$$S = 2 \left[p \left(\frac{1+c}{1-c} + b^2 \right) \right]^{-0.5}$$

where:

p	is the mean step length (in meters),
c	is the mean cosine of turning angles (in radians), and

b is the coefficient of variation of the step length (in meters).

The straightness index is defined as the ratio D/L, where:

D is the beeline distance between the first and last points in the trajectory (in meters), and

L is the total path length traveled (in meters).

Straightness index is based on the method described by Batschelet (1981). According to Benhamou (2004), the straightness index serves as a reliable measure of the efficiency of a directed walk. However, it is not suitable for random trajectories, as the index for a random walk tends towards zero with increasing steps. Thus, it is recommended to use this measure to compare the tortuosity of random walks only if they consist of a similar number of steps.

Logo

Author(s)

Humberto G. Ferrón

humberto.ferron@uv.es

Macroevolution and Functional Morphology Research Group (www.macrofun.es)

Cavanilles Institute of Biodiversity and Evolutionary Biology

Calle Catedrático José Beltrán Martínez, nº 2

46980 Paterna - Valencia - Spain

Phone: +34 (9635) 44477

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:: straightness, sinuosity, or fractal dimension?. Journal of theoretical biology, 229(2), 209-220.

Bovet, P., & Benhamou, S. (1988). Spatial analysis of animals' movements using a correlated random walk model. Journal of theoretical biology, 131(4), 419-433.

See Also

tps_to_track

Examples

```
# Example 1:
track_param(PaluxyRiver)

# Example 2:
track_param(MountTom)
```

`velocity_track` *Calculate velocities and relative stride lengths for tracks*

Description

`velocity_track()` calculates the velocities and relative stride lengths for each step in a series of tracks, based on the step length, height at the hip, and gravity acceleration.

Usage

```
velocity_track(data, H, G = NULL, method = NULL)
```

Arguments

`data`	A `track` R object, which is a list consisting of two elements: • `Trajectories`: A list of interpolated trajectories, where each trajectory is a series of midpoints between consecutive footprints. • `Footprints`: A list of data frames containing footprint coordinates, metadata (e.g., image reference, ID), and a marker indicating whether the footprint is actual or inferred.
`H`	A numeric vector representing the height at the hip (in meters) for each track maker. The length of this vector should match the number of tracks in the data.
`G`	Gravity acceleration (in meters per second squared). Default is `9.8`.
`method`	A character vector specifying the method to calculate velocities for each track. Method `"A"` follows the approach from Alexander (1976), while method `"B"` is based on Ruiz & Torices (2013). If `NULL`, method `"A"` will be used for all tracks.

Details

The `velocity_track()` function calculates velocities using two methods:

Method A: Based on Alexander (1976), with the formula:

$$v = 0.25 \cdot \sqrt{G} \cdot S^{1.67} \cdot H^{-1.17}$$

- **v**: Velocity of the track-maker (in meters per second).
- **G**: Acceleration due to gravity (in meters per second squared), typically 9.81 m/s^2.
- **S**: Stride length, which is the distance between consecutive footprints (in meters).
- **H**: Height at the hip of the track-maker (in meters).
- The coefficients 0.25, 1.67, and -1.17 are derived from empirical studies. These coefficients adjust the formula to account for different animal sizes and gaits.

This method applies to a wide range of terrestrial vertebrates and is used to estimate velocity across different gaits.

Method B: Based on Ruiz & Torices (2013), with the formula:

$$v = 0.226 \cdot \sqrt{G} \cdot S^{1.67} \cdot H^{-1.17}$$

- **v**: Velocity of the track-maker (in meters per second).
- **G**: Acceleration due to gravity (in meters per second squared), typically 9.81 m/s^2.
- **S**: Stride length (in meters).
- **H**: Height at the hip of the track-maker (in meters).
- The oefficient 0.226 in method B is a refinement based on updated data for bipedal locomotion.

Based on Thulborn & Wade (1984), it is possible to identify the gaits of track-makers on the basis of relative stride length, as follows:

- **Walk**: $A/H < 2.0$; locomotor performance equivalent to walking in mammals.
- **Trot**: $2.0 \leq A/H \leq 2.9$; locomotor performance equivalent to trotting or racking in mammals.
- **Run**: $A/H > 2.9$; locomotor performance equivalent to cantering, galloping, or sprinting in mammals.

Value

A `track velocity` R object consisting of a list of lists, where each sublist contains the computed parameters for a corresponding track. The parameters included are:

- `Step_velocities`: A vector of velocities for each step in the track (in meters per second).
- `Mean_velocity`: The mean velocity across all steps in the track (in meters per second).
- `Standard_deviation_velocity`: The standard deviation of velocities across all steps in the track (in meters per second).
- `Maximum_velocity`: The maximum velocity among all steps in the track (in meters per second).
- `Minimum_velocity`: The minimum velocity among all steps in the track (in meters per second).
- `Step_relative_stride`: A vector of relative stride lengths for each step in the track (dimensionless).
- `Mean_relative_stride`: The mean relative stride length across all steps in the track (dimensionless).
- `Standard_deviation_relative_stride`: The standard deviation of relative stride lengths across all steps in the track (dimensionless).
- `Maximum_relative_stride`: The maximum relative stride length among all steps in the track (dimensionless).
- `Minimum_relative_stride`: The minimum relative stride length among all steps in the track (dimensionless).

Logo

Author(s)

Humberto G. Ferrón

humberto.ferron@uv.es

Macroevolution and Functional Morphology Research Group (www.macrofun.es)

Cavanilles Institute of Biodiversity and Evolutionary Biology

Calle Catedrático José Beltrán Martínez, nº 2

46980 Paterna - Valencia - Spain

Phone: +34 (9635) 44477

References

Alexander, R. M. (1976). Estimates of speeds of dinosaurs. Nature, 261(5556), 129-130.

Ruiz, J., & Torices, A. (2013). Humans running at stadiums and beaches and the accuracy of speed estimations from fossil trackways. Ichnos, 20(1), 31-35.

Thulborn, R. A., & Wade, M. (1984). Dinosaur trackways in the Winton Formation (mid-Cretaceous) of Queensland. Memoirs of the Queensland Museum, 21(2), 413-517.

See Also

tps_to_track

Examples

```
# Example 1: Calculate velocities for the MountTom dataset using default settings.
# H_mounttom contains hip heights for each track in the MountTom dataset.
# The function will use the default method "A" for all tracks.
# Hip heights are inferred as four times the footprint length, following Alexander's approach.
H_mounttom <- c(
  1.380, 1.404, 1.320, 1.736, 1.364, 1.432, 1.508, 1.768, 1.600,
  1.848, 1.532, 1.532, 0.760, 1.532, 1.688, 1.620, 0.636, 1.784, 1.676, 1.872,
  1.648, 1.760, 1.612
)
velocity_track(MountTom, H = H_mounttom)

# Example 2: Calculate velocities for the PaluxyRiver dataset using default settings.
# H_paluxyriver contains hip heights for each track in the PaluxyRiver dataset.
# The function will use the default method "A" for all tracks.
# Hip heights are inferred as four times the footprint length, following Alexander's approach.
H_paluxyriver <- c(3.472, 2.200)
velocity_track(PaluxyRiver, H = H_paluxyriver)

# Example 3: Calculate velocities for the PaluxyRiver dataset using different methods
# for velocity calculation. Method "A" is used for sauropods, which is more
# appropriate for quadrupedal dinosaurs. Method "B" is used for theropods, which
# is more appropriate for bipedal dinosaurs. Hip heights are inferred as four times
# the footprint length, following Alexander's approach.
H_paluxyriver <- c(3.472, 2.200)
Method_paluxyriver <- c("A", "B")
```

```
velocity_track(PaluxyRiver, H = H_paluxyriver, method = Method_paluxyriver)
```

Index