Package: animetric (via r-universe)

June 3, 2026

Type Package

Title An R package for calculating movement-based metrics

Version 0.3.2

Description An R package for calculating movement-based metrics.

License MIT + file LICENSE

URL http://animovement.dev/animetric/, https://github.com/animovement/animetric/

BugReports https://github.com/animovement/animetric/issues

Encoding UTF-8

LazyData true

Depends R (>= 4.1.0)

Imports aniframe, anispace, cli, dplyr, purrr, rlang

Suggests circular, data.table, knitr, sf, testthat (>= 3.0.0)

Additional_repositories https://animovement.r-universe.dev

Config/testthat/edition 3

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

VignetteBuilder knitr

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

Date/Publication 2026-01-13 19:15:41 UTC

RemoteUrl https://github.com/animovement/animetric

RemoteRef HEAD

RemoteSha 288036ea63fbced419e3b0883fba4ccfbee6b3f5

Contents

`calculate_kinematics` *Calculate kinematic measures from trajectory data*

Description

Computes translational and rotational kinematic measures from movement data. Handles data in any coordinate system by automatically converting to Cartesian for calculations, then converting back to the original system.

Usage

```
calculate_kinematics(data)
```

Arguments

`data`	An aniframe with position coordinates (x/y or x/y/z for Cartesian; rho/phi for polar; rho/phi/z for cylindrical; rho/phi/theta for spherical) and a time column

Details

The function preserves the original coordinate system by:

1. Detecting the input coordinate system from metadata
2. Converting to Cartesian if necessary
3. Computing kinematics in Cartesian space
4. Converting back to the original coordinate system

All kinematic calculations are performed using numerical differentiation via the `differentiate` function. Angles are unwrapped to handle discontinuities at $\pm$.

Value

An aniframe in the same coordinate system as the input, with added kinematic measures. For 2D data, includes translational kinematics (velocity components, speed, acceleration, path length) and rotational kinematics (heading, angular velocity, angular speed, angular acceleration). For 3D data, includes translational kinematics only (rotational measures for 3D are not yet implemented).

Examples

```
## Not run:
# 2D Cartesian data
traj_2d <- data.frame(time = 0:10, x = rnorm(11), y = rnorm(11)) |>
  as_aniframe()
kinematics_2d <- calculate_kinematics(traj_2d)

# Polar data (automatically converted and converted back)
traj_polar <- aniframe::map_to_polar(traj_2d)
kinematics_polar <- calculate_kinematics(traj_polar)

## End(Not run)
```

`calculate_nnd` *Calculate distance to n-th nearest neighbour*

Description

Computes the distance from each point to the n-th nearest point belonging to a different individual. Optionally filter by keypoint on neighbouring individuals.

Usage

```
calculate_nnd(data, n = 1L, keypoint_neighbour = NULL)
```

Arguments

`data`	An aniframe with x, y (and optionally z) coordinates and individual identifiers.
`n`	Which neighbour (1 = nearest, 2 = second nearest, etc.).
`keypoint_neighbour`	Keypoint(s) on other individuals to consider as potential neighbours. Can be a single keypoint or a character vector. If NULL (default), considers all keypoints.

Value

The input aniframe with additional columns:

- `nnd_distance`: distance to the n-th nearest neighbour
- `nnd_individual`: individual ID of the n-th nearest neighbour
- `nnd_keypoint`: keypoint of the neighbour (if keypoint column exists)

Examples

```
## Not run:
# Distance to nearest individual (any keypoint to any keypoint)
data |> calculate_nnd()

# Distance to nearest nose of another individual
data |> calculate_nnd(keypoint_neighbour = "nose")

# Nose-to-nose distance
data |>
  calculate_nnd(keypoint_neighbour = "nose") |>
  dplyr::filter(keypoint == "nose")

# Minimum distance between individuals (across all keypoints)
data |>
  calculate_nnd() |>
  dplyr::group_by(session, trial, time, individual) |>
  dplyr::slice_min(nnd_distance, n = 1)

## End(Not run)
```

`calculate_tortuosity` *Calculate tortuosity metrics over sliding windows*

Description

Computes multiple tortuosity metrics (straightness, sinuosity, E_max) over sliding windows, returning a value at each timepoint.

Usage

```
calculate_tortuosity(data, window_width = 11L)
```

Arguments

`data`	An aniframe with position coordinates and time. Velocity and heading columns will be computed if not already present.
`window_width`	Size of the sliding window (number of observations). Should be an odd number >= 3 for symmetric centering.

Details

If required kinematic columns are missing, the function will compute them automatically by calling the appropriate helper functions.

Straightness is appropriate for directed/goal-oriented movement, while sinuosity and E_max are appropriate for random search paths.

For 2D data, heading is derived from the velocity vector, which provides smoother estimates than raw position differences.

For 3D data, turning angles are computed as the angle between consecutive velocity vectors using the dot product.

The window is centered on each timepoint. At path edges, metrics are computed from available data within the truncated window.

Value

The input aniframe with additional columns:

straightness Straightness index (D/L), ranges 0-1

sinuosity Corrected sinuosity index (Benhamou 2004)

emax Maximum expected displacement (dimensionless)

References

Batschelet, E. (1981). Circular statistics in biology. Academic Press.

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.

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

See Also

- `calculate_kinematics()` for computing velocity and heading

Examples

```
## Not run:
# Kinematics computed automatically if missing
data |>
  calculate_tortuosity(window_width = 11)

# Or with kinematics already computed
data |>
  calculate_kinematics() |>
  calculate_tortuosity(window_width = 11)

## End(Not run)
```

compute_centroid *Compute centroid from keypoints*

Description

Calculates the mean position of selected keypoints at each time point. The centroid is computed for each combination of grouping variables (individual, time, trial/session if present).

Usage

```
compute_centroid(
  data,
  include_keypoints = NULL,
  exclude_keypoints = NULL,
  centroid_name = "centroid"
)
```

Arguments

`data`	An aniframe with Cartesian coordinates (x, y, and/or z columns).
`include_keypoints`	Character vector of keypoints to include in centroid calculation. If NULL (default), all keypoints are used unless `exclude_keypoints` is specified. Mutually exclusive with `exclude_keypoints`.
`exclude_keypoints`	Character vector of keypoints to exclude from centroid calculation. If NULL (default), no keypoints are excluded. Mutually exclusive with `include_keypoints`.
`centroid_name`	Name for the new centroid keypoint. Default is "centroid".

Value

An aniframe containing only the centroid keypoint. Coordinate values are the mean of selected keypoints (with NA values removed). Confidence is set to NA. Missing coordinate dimensions return NA.

compute_emax *Compute E_max (maximum expected displacement) from pre-computed vectors*

Description

Compute E_max (maximum expected displacement) from pre-computed vectors

Usage

```
compute_emax(mean_cos_turning, mean_step_length = NULL, dimensional = FALSE)
```

Arguments

`mean_cos_turning`	Numeric vector of mean cosine of turning angles.
`mean_step_length`	Numeric vector of mean step lengths (required if `dimensional = TRUE`).
`dimensional`	Logical. If `TRUE`, returns E_max in spatial units; otherwise returns the dimensionless ratio.

Value

Numeric vector of E_max values (same length as `mean_cos_turning`), with `NA` for invalid inputs and `Inf` for perfectly straight paths.

`compute_nnd` *Compute nearest neighbour distances for a single time point*

Description

Low-level function that computes distances to the nth nearest individual. For each focal point, finds the closest point belonging to the nth nearest individual (ranked by minimum distance). Called by `calculate_nnd()` for each time point.

Usage

```
compute_nnd(
  x,
  y,
  z = NULL,
  individual,
  keypoint = NULL,
  n = 1L,
  keypoint_neighbour = NULL
)
```

Arguments

`x`	Numeric vector of x coordinates.
`y`	Numeric vector of y coordinates.
`z`	Numeric vector of z coordinates, or NULL for 2D data.
`individual`	Factor or vector identifying which individual each point belongs to.
`keypoint`	Factor or vector identifying keypoint labels, or NULL if no keypoints.
`n`	Which individual to find (1 = nearest, 2 = second nearest, etc.).

`keypoint_neighbour`
: Character vector of keypoint(s) to consider as valid neighbours, or NULL to consider all.

Value

A tibble with columns:

- `nnd_individual`: individual ID of the n-th nearest individual
- `nnd_keypoint`: keypoint of the closest point on that individual (only if `keypoint` is not NULL)
- `nnd_distance`: distance to the closest point on the n-th nearest individual

See Also

`calculate_nnd()` for the user-facing aniframe function

`compute_sinuosity` *Compute sinuosity index from precomputed vectors*

Description

Compute sinuosity index from precomputed vectors

Usage

```
compute_sinuosity(
  mean_step_length,
  mean_cos_turning,
  method = c("corrected", "original")
)
```

Arguments

`mean_step_length`
: Numeric vector of mean step lengths within window

`mean_cos_turning`
: Numeric vector of mean cosine of turning angles

`method`
: Either ”corrected” (Benhamou 2004) or ”original” (Bovet & Benhamou 1988)

Value

Numeric vector of sinuosity values

compute_straightness *Compute straightness index from precomputed vectors*

Description

Compute straightness index from precomputed vectors

Usage

```
compute_straightness(displacement, path_length)
```

Arguments

displacement	Numeric vector of net displacements (D)
path_length	Numeric vector of path lengths (L)

Value

Numeric vector of straightness values (D/L)

differentiate *Differentiate a numeric series (optionally repeatedly)*

Description

Wrapper around `compute_gradient()` that optionally applies the gradient operator multiple times (`order`). If no explicit time vector is supplied, a simple index sequence is used.

Usage

```
differentiate(x, time = NULL, order = 1)
```

Arguments

x	Numeric vector of observations.
time	Optional numeric vector of timestamps. Must be the same length as `x`. If `NULL`, `seq_along(x)` is used.
order	Integer 1 indicating how many times the differentiation should be applied. Defaults to a single derivative.

Details

The function computes the first-order derivative using the Fornberg-based scheme implemented in `compute_gradient()`. When `order > 1`, the gradient is applied iteratively to the result of the previous iteration.

Value

Numeric vector of the same length as `x` containing the differentiated values.

Examples

```
# Simple equally spaced case
y <- sin(seq(0, 2 * pi, length.out = 10))
differentiate(y)

# Uneven time stamps
t <- c(0, 0.9, 2.1, 3.8, 5.0, 5.2, 5.6, 6.7, 7.2, 8.9)
differentiate(y, time = t, order = 2)
```

`is_aniframe_kin` *Check if object is an aniframe_kin*

Description

Check if object is an aniframe_kin

Usage

```
is_aniframe_kin(x)
```

Arguments

`x`	An object to test

Value

Logical: TRUE if x inherits from aniframe

`mean_angle` *Compute the circular mean of angles*

Description

Returns the mean direction of a vector of angles in radians, wrapped to [0, 2*pi).

Usage

```
mean_angle(ang)
```

Arguments

`ang`	Numeric vector of angles in radians.

Value

Circular mean in [0, 2*pi).

Examples

```
mean_angle(c(pi/2, 1.5 * pi))
mean_angle(c(0, pi))
```

---

median_angle *Compute the circular median of angles*

Description

Returns the median direction of a vector of angles in radians, wrapped to [0, 2*pi).

Usage

```
median_angle(ang)
```

Arguments

ang	Numeric vector of angles in radians.

Value

Circular median in [0, 2*pi).

Examples

```
median_angle(c(pi/2, 1.5 * pi))
median_angle(c(0, pi))
```

---

summarise_aniframe *Summarise an aniframe*

Description

Calculate summary statistics for aniframe data by dispatching to specialised summary functions.

Usage

```
summarise_aniframe(
  data,
  type = c("kinematics", "tortuosity"),
  measures = c("median_mad", "mean_sd")
)

summarize_aniframe(
  data,
  type = c("kinematics", "tortuosity"),
  measures = c("median_mad", "mean_sd")
)
```

Arguments

`data`	A kinematics aniframe (output of `calculate_kinematics()`)
`type`	Character vector of summary types. Options are `"kinematics"` and `"tortuosity"`. Default is both.
`measures`	Measures of central tendency and dispersion for kinematics. Options are `"median_mad"` (default) and `"mean_sd"`.

Value

A summarised data frame with one row per group.

See Also

`summarise_kinematics()`, `summarise_tortuosity()`

`summarise_keypoints` *Summarize keypoint data*

Description

Creates summary statistics across multiple keypoints at each time point. Currently supports computing centroids from selected keypoints. Future functionality will include polygonal summaries.

Usage

```
summarise_keypoints(
  data,
  keypoints = "all",
  name = "centroid",
  add_area = FALSE
)
```

Arguments

`data`	An aniframe containing keypoint data.
`keypoints`	Character vector of keypoint names to summarize, or "all" to use all keypoints in the data. Default is "all".
`name`	Character string for the name of the new summary keypoint. Default is "centroid".
`add_area`	Logical indicating whether to compute area (not yet implemented). Default is FALSE.

Value

An aniframe with the original data plus the new summary keypoint.

`summarise_kinematics` *Calculate kinematic summary statistics*

Description

Calculate central tendency and dispersion for translational and rotational kinematics.

Usage

```
summarise_kinematics(
  data,
  measures = c("median_mad", "mean_sd"),
  .check = TRUE
)

summarize_kinematics(
  data,
  measures = c("median_mad", "mean_sd"),
  .check = TRUE
)
```

Arguments

`data`	A kinematics aniframe (output of `calculate_kinematics()`)
`measures`	Measures of central tendency and dispersion for kinematics. Options are `"median_mad"` (default) and `"mean_sd"`.
`.check`	Whether to validate input. Set to `FALSE` when called from `summarise_aniframe()` to avoid redundant checks.

Value

A summarised data frame with one row per group containing central tendency and dispersion measures (prefixed with median_/mad_ or mean_/sd_)

- Speed, acceleration
- Angular speed, velocity, acceleration (2D only)
- Heading (2D only, using circular statistics)

`summarise_tortuosity` *Calculate tortuosity summary statistics*

Description

Calculate path length, displacement, and tortuosity metrics.

Usage

```
summarise_tortuosity(data)

summarize_tortuosity(data)
```

Arguments

`data`	A kinematics aniframe (output of `calculate_kinematics()`)

Value

A summarised data frame with one row per group containing:

- `total_path_length`: Total distance traveled
- `total_angular_path_length`: Total angular distance (2D only)

Tortuosity metrics:

- `net_displacement`: Straight-line distance from start to end
- `straightness`: Ratio of net displacement to path length (0-1)
- `sinuosity`: Corrected sinuosity index (Benhamou 2004)
- `emax`: Maximum expected displacement (dimensionless)

References

Benhamou, S. (2004). How to reliably estimate the tortuosity of an animal's path. Journal of Theoretical Biology, 229(2), 209-220.

Index