

Package: animetric (via r-universe)

October 14, 2026

Type Package

Title An R package for calculating movement-based metrics

Version 0.5.0.9006

Description An R package for calculating movement-based metrics.

License MIT + file LICENSE

URL <https://animovement.dev/animetric/>,
<https://github.com/animovement/animetric/>

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

Encoding UTF-8

Language en-GB

Depends R ($\geq$ 4.1.0)

Imports anicore ($\geq$ 0.8.0.9004), anispace ($\geq$ 0.3.0.9006), cli,
data.table, dplyr ($\geq$ 1.1.0), lifecycle, purrr, rlang

Suggests testthat ($\geq$ 3.0.0)

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

Config/testthat/edition 3

Config/Needs/website animovement/animovementtemplate, knitr

Roxygen list(markdown = TRUE)

Config/roxygen2/version 8.1.0

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

Date/Publication 2026-10-14 12:03:49 UTC

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

RemoteRef HEAD

RemoteSha 0aefeb6c6f8e1b6154a39014dd2f70417b4f169c

Contents

add_kinematics	2
add_nnd	6
add_orientation	8
add_point	10
add_tortuosity	11
compute_emax	14
compute_gradient	14
compute_min_step	15
compute_nnd	16
compute_point	18
compute_sinusosity	19
compute_straightness	20
differentiate	20
summarise_aniframe	21
summarise_path	23

Index	26
--------------	-----------

add_kinematics	<i>Add kinematic measures to trajectory data</i>
----------------	--

Description

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

Usage

```
add_kinematics(data, vertical = NULL, min_step = "auto")
```

Arguments

data	An anipoint with position coordinates (x, x/y or x/y/z for Cartesian; rho/phi for polar; rho/phi/z for cylindrical; rho/phi/theta for spherical) and an index. The axes and the index are read from the frame's declared variables, so the columns can have any name.
vertical	[Experimental] For 3D data, the axis that points up in the world, against gravity: one of "x", "y" or "z", or with a minus sign ("-y") when that axis points down. It defines the horizontal plane that course is measured in. The frame's axis_directions cannot supply it, since they are relative to the camera: in a recording filmed from above, the axis pointing at the camera is the vertical one. A metadata field for it is proposed in animovement/anicore#172. NULL (the default) gives only the measures that need no vertical. Ignored for 1D and 2D data, where course is measured

in the plane of the data. Experimental until that proposal is settled: the argument may come to default to the declared vertical, and to mean something for 2D side views.

min_step **[Experimental]** The shortest step, in the frame's spatial unit, whose direction counts. Where a point moves less than this per sample, its direction of travel is set by tracking noise rather than by where the animal is going, so the direction is treated as undefined: **course** is NA there, and the row adds no turning. One of:

- "auto" (the default)** Three times the tracking noise, estimated separately for each trajectory, and at most half its median step. See Details, and `compute_min_step()` for the value it chooses.
- a number** A threshold of your own, such as the tracking precision in the frame's spatial unit.
- 0** Every direction counts, however short the step.

It affects only the direction measures (**course** and the turning measures), not **speed** or **cumulative_distance**.

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. Course is unwrapped before it is differentiated, so the turning rate has no jump at $\pm\pi$. The turning speed in 3D needs no angle to unwrap, and has no singularity when travel is vertical.

The minimum step:

When a point barely moves, the direction between successive positions is set by sub-pixel tracking noise, and swings at random from frame to frame. Every swing is turning to the measures above, so a resting animal accumulates turning as fast as a running one. **min_step** sets the step below which the direction is not trusted.

The step of a row is the distance its velocity implies moving per sample: **speed** times the sampling interval, which for the central differences used here is half the distance between the positions either side. Course is measured against the horizontal part of the step alone. With **min_step = "auto"**, the threshold is three times the positional noise, σ , estimated for each trajectory from its second differences $p_{i+1} - 2p_i + p_{i-1}$. For white noise of standard deviation σ on each axis these have standard deviation $\sqrt{6}\sigma$. Only their component along the direction of travel is used, which a path turning at constant speed leaves at zero, and their spread is measured with `stats::mad()`, which a minority of large values barely moves. A point standing still under white noise makes steps longer than 3σ with a probability of about 0.01% (0.04% in 3D), so the threshold removes the directions of a stationary point while keeping those of real movement.

The threshold is also at most half the trajectory's median step (among steps that have a direction), so most steps always keep theirs. This matters when the sampling is coarse for the movement: a random walk recorded step by step has second differences as large as its steps, and the estimate would take the movement itself for noise.

The estimate assumes the sampling is fast compared with changes in speed, as it is for video tracking. Two cases call for setting `min_step` yourself:

- **Smoothed positions.** Smoothing turns noise into slow wander, which second differences see less of, so the estimate is smaller than the jitter left in the data, and the threshold removes only part of it.
- **Coarse sampling,** such as GPS fixes minutes apart, where speed changes a lot between samples. The estimate then includes real movement, so the threshold may be too large; use the positional precision instead, or 0.

The threshold is not stored in the result. `compute_min_step()` returns the one "auto" chooses for each trajectory, with the noise estimate, and a number given as `min_step` is used as it is.

Value

An anipoint in the same coordinate system as the input, with added kinematic measures. Translational measures are computed for 1D, 2D and 3D data:

`v_x`, `v_y`, `v_z` Velocity components, named by axis role whatever the input columns are called.

`a_x`, `a_y`, `a_z` Acceleration components, the second derivatives of the positions.

`speed` The size of the velocity.

`acceleration` The signed rate of change of `speed`: positive when speeding up, negative when slowing down. This is the tangential acceleration, along the path. It is not the size of the acceleration vector (`a_x`, `a_y`), which also includes the part that turns the path: an animal rounding a corner at constant speed has an `acceleration` of 0 but non-zero `a_x` and `a_y`.

`cumulative_distance` Distance travelled since the first row: the straight-line steps between successive positions, summed. A step to or from a missing position adds nothing.

For 2D and 3D data, measures of how the direction of travel changes are added too:

`turning_speed` How fast the direction of travel turns, in any direction: curvature times speed. In 2D it is the size of the turning rate. In 3D it also counts climbing and diving, and is the angle between the velocities either side of each row, over the time between them.

`cumulative_turning` Turning accumulated since the first row: the angle between successive velocities, summed. A turn made while stationary, or while moving less than `min_step`, is counted when the animal moves off, as the angle between the directions before and after.

In 2D, and in 3D when `vertical` is given, the measures that need a direction to count from:

course Direction of travel in the horizontal plane (in 2D, the plane of the data): `atan2(v_y, v_x)` in 2D. It is NA where there is no horizontal movement, or less than `min_step` per sample, since the direction is then undefined. `course_unwrapped` is the same, without jumps at $\pm\pi$.

course_elevation 3D only: the angle of travel above the horizontal plane, from $-\pi/2$ (straight down) to $\pi/2$ (straight up).

turning_rate Signed rate of change of the course, the derivative of `course_unwrapped`. In 3D it counts horizontal turning only.

turning_acceleration Rate of change of the turning rate.

These describe the path, not the body: `course` is where the animal is going, not where it is facing, and the two differ for any animal that does not move nose-first. The names `heading` and `angular velocity` are kept for body orientation. 1D data gets translational measures only.

Angular measures are in the frame's declared `unit_angle`, radians or degrees; turning rate and acceleration are per unit of the index. Signed angles (`course`, `turning_rate`) follow the frame's own convention: `course` counts from the x axis toward the y axis, which is counter-clockwise when `anicore::get_angle_direction()` says so (y pointing up, as `aniread` leaves image data) and clockwise in a frame whose y points down. To change convention, change the coordinates, for example with `anicore::reflect_axis()`, and the angles follow. In 3D, `course` counts about **vertical** by the right-hand rule, from the next axis in the cycle x, y, z: from x toward y when z is vertical, from z toward x when y is, from y toward z when x is.

See Also

`add_tortuosity()` for windowed measures of how winding the path is, and `summarise_path()` for measures of each whole trajectory.

Examples

```
# 2D Cartesian data
traj_2d <- data.frame(time = 0:10, x = rnorm(11), y = rnorm(11)) |>
  anicore::as_anipoint()
kinematics_2d <- add_kinematics(traj_2d)

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

# 3D data with z pointing up: course and elevation of travel
traj_3d <- data.frame(
  time = 0:10,
  x = cos(0:10 / 2),
  y = sin(0:10 / 2),
  z = 0:10 / 10
) |>
  anicore::as_anipoint()
kinematics_3d <- add_kinematics(traj_3d, vertical = "z")
```

```
# A threshold of your own, in the frame's spatial unit, or 0 to count
# every direction however short the step
kinematics_2d <- add_kinematics(traj_2d, min_step = 0.1)
```

add_nnd

Add the distance to the *n*-th nearest neighbour

Description

Computes, for each point, the distance to the nearest point belonging to a *different* entity — typically a different individual at the same moment.

Which columns carry time and position is read from the anipoint's declared variables (see `anicore::get_variables()`). The identity columns are assigned roles by you, explicitly, because "another animal" and "another point on this animal" are different questions and the data cannot tell which one you mean.

Usage

```
add_nnd(data, across, n = 1L, within = NULL, focal = NULL, neighbour = NULL)
```

Arguments

data	An anipoint.
across	Column whose value must differ between a point and its neighbour.
n	Which neighbour to return (1 = nearest, 2 = second nearest). Ranked by entity, not by point: with n = 2, the result is the closest point on the second-nearest entity.
within	Identity columns that must match, added to the temporal context.
focal	Named list restricting which points are measured from, e.g. <code>list(keypoint = "nose")</code> . <code>NULL</code> measures from every point.
neighbour	Named list restricting which points may be returned as a neighbour, e.g. <code>list(keypoint = "tail")</code> .

Details

Every identity variable has one of three roles:

- **across** — its value must *differ* between a point and its neighbour. This is what "another" means: **"individual"** for the nearest other animal, **"keypoint"** for the nearest other point on the same animal.
- **within** — its value must *match*. Added to the temporal context, which always applies: points are never compared across timepoints, observations, sessions or trials.
- unnamed — unconstrained. Any value may match any other, which is what makes the default any-keypoint-to-any-keypoint.

focal and **neighbour** then restrict which points are measured *from* and which are eligible to be measured *to*. Both are named lists of column to permitted values, and they are independent, so asymmetric questions like nose-to-tail are expressible.

Value

The input anipoint with added columns, each named `nnd_`, then the neighbour rank `n`, then what the column holds:

- `nnd_<n>_<across>` — which entity the `n`-th nearest neighbour belongs to (e.g. `nnd_1_individual`)
- `nnd_<n>_<variable>` — the neighbour's value for each unconstrained identity variable (e.g. `nnd_1_keypoint`)
- `nnd_<n>_distance` — the distance to it

The rank is always there, 1 included, so the nearest and second-nearest neighbours can sit side by side: call `add_nnd()` once with `n = 1` and again with `n = 2`. Because the rank sits between two underscores and is only digits, the names split unambiguously with `"^nnd_(\\d+)_(\\.+)$"`, even when the part after it contains underscores, as an `across` column named `track_id` gives `nnd_1_track_id`.

See Also

[compute_nnd\(\)](#) for the vector-level function.

Examples

```
data <- anicore::example_anipoint(
  n_obs = 5,
  n_individuals = 3,
  n_keypoints = 3
)

# Nearest other individual, any keypoint to any keypoint
data |> add_nnd(across = "individual")

# Whose neck is my head nearest to?
data |> add_nnd(
  across = "individual",
  focal = list(keypoint = "head"),
  neighbour = list(keypoint = "neck")
)

# Nearest keypoint within each individual
data |> add_nnd(across = "keypoint", within = "individual")

# Each keypoint to the same keypoint on the nearest other individual
data |> add_nnd(across = "individual", within = "keypoint")

# Nearest and second-nearest other individual, side by side
data |>
  add_nnd(across = "individual", n = 1) |>
  add_nnd(across = "individual", n = 2)
```

add_orientation	<i>Declare orientation from the positions of points</i>
-----------------	---

Description

[Experimental]

Works out which way a body faces from where its points are, and declares it as the frame's orientation: **heading** in 2D, a unit quaternion (**qw**, **qx**, **qy**, **qz**) in 3D. Once declared, it is a proper **where** variable that later steps can use, such as egocentric alignment by orientation in anispace.

The body frame follows anicore's: an orientation of zero faces **+x**.

Along the body (default) **from** -> **to** is the forward axis, e.g. tail to head.

Across the body (**perpendicular = TRUE**) **from** -> **to** runs from the body's right to its left, e.g. right eye to left eye, and forward is perpendicular to it.

In 3D, two points give a direction but not the roll about it, so a third, **plane**, is needed. Any point off the line through **from** and **to** will do: it fixes which way the body's y axis points (along the body), or its forward axis (across the body). A point on the animal's left makes y its left; a point on its back makes y point dorsally instead. Either way the orientation is fully defined; only what its roll is called depends on the choice.

Usage

```
add_orientation(
  data,
  from,
  to,
  plane = NULL,
  perpendicular = FALSE,
  attach_to = NULL,
  level = NULL,
  name = NULL,
  overwrite = FALSE
)
```

Arguments

data	A 2D or 3D anipoint with Cartesian coordinates.
from, to	Members of level defining the axis: forward, or with perpendicular = TRUE , right to left.
plane	In 3D, a member of level off the line through from and to , fixing the roll about it. Not used in 2D.
perpendicular	Whether from -> to runs across the body, from right to left, rather than along it.

attach_to	Members of level that get the orientation. NULL , the default, gives it to every member of the subject: the body's orientation. Name some to attach different orientations to different parts, e.g. a head orientation to the head's points and a body orientation to the rest.
level	The identity variable from , to , plane and attach_to name members of. Defaults to the frame's only one; a frame declaring several has to be told.
name	The orientation column: one name in 2D, four in 3D (w , x , y , z components). Defaults to the columns of an orientation already declared, or else "heading" in 2D and c("qw", "qx", "qy", "qz") in 3D.
overwrite	Whether to replace orientation values already present in the rows being written, or an orientation declared in other columns.

Details

The orientation is computed for each subject at each moment: each combination of the frame's other identity variables, its temporal keys and its index. Where a defining point is missing, or the points coincide, it is **NA**. Rows not named by **attach_to** keep what they had, so successive calls can build up orientations for different parts in the one declared column.

In 2D, **heading** is in the frame's **unit_angle**, counting from the **x** axis toward the **y** axis, in $(-\pi, \pi]$. Across the body, forward is the **from** → **to** axis turned a quarter turn, so that **to** is on the body's left. In 3D the quaternion maps the body's axes into the frame's, and is built with [anispace::quat_from_vectors\(\)](#).

Value

The anipoint, with the orientation columns written and declared.

Examples

```
af <- anicore::example_anipoint(n_obs = 5, n_individuals = 2, n_keypoints = 3)

# The body faces from the shoulder to the head
add_orientation(af, from = "shoulder_right", to = "head", level = "keypoint")

# Attached to the head only
add_orientation(
  af,
  from = "shoulder_right",
  to = "head",
  attach_to = "head",
  level = "keypoint"
)
```

add_point

Add a derived point to an anipoint

Description

Derives a new member of an identity level at each moment, from the members it collapses, and appends it to the frame. The rest of the data is returned untouched. The centroid of an animal's keypoints is the usual case; **method** chooses how the point is derived.

Which levels are collapsed is the caller's choice. On pose data for a team, collapsing **"keypoint"** gives each player a point of their own; **across = "individual"** gives one point per keypoint across the players; and collapsing both gives the single point the whole team occupies.

A level that did not actually vary keeps its value rather than taking the new member's name — an individual's strain is still its strain, since nothing was derived over it.

The new member is an ordinary member of its level: **add_kinematics()** gives it kinematics, and **summarise_aniframe()** summarises it alongside the tracked points.

Usage

```
add_point(
  data,
  across = NULL,
  method = "centroid",
  include = NULL,
  exclude = NULL,
  name = NULL
)
```

Arguments

data	An anipoint with Cartesian coordinates.
across	Identity variables to collapse — the dimensions the new point is derived over. Required when the frame declares more than one identity variable, since their order is not a hierarchy and there is no finest one to assume; with a single identity variable, that one is the default. Collapsing every level gives a single point per position.
method	How each coordinate is derived from the members' values: "centroid" The mean. "median" The median, per axis: robust to a single stray point. "weighted" The mean weighted by confidence , so poorly tracked points count for less. Needs a confidence column. a function Applied to each axis's values, with missing values removed, returning one number; e.g. <code>\(x) mean(x, trim = 0.1)</code> . Missing values are ignored; a moment where every member is missing gives NA.

<code>include, exclude</code>	Values of the collapsed level to keep or leave out. Only meaningful when one level is collapsed. A midpoint is the centroid of two members: <code>include = c("ear_l", "ear_r")</code> .
<code>name</code>	Name for the new member. Defaults to "centroid", "median" or "weighted_centroid" after <code>method</code> ; required when <code>method</code> is a function.

Details

A declared orientation is derived too: the circular mean of the members' `yaw`, in the frame's `unit_angle`, or in 3D the mean of their quaternions (`anispace::quat_mean()`). It is weighted by `confidence` when `method` is "weighted", and unweighted otherwise. The new member has no confidence of its own, so its `confidence` is NA.

Value

The anipoint, with the new member appended as extra rows. The collapsed identity column comes back as a factor, since it now holds a named member that an integer column could not.

See Also

`compute_point()` for the new member's rows alone.

Examples

```
af <- anicore::example_anipoint(n_obs = 20, n_individuals = 2, n_keypoints = 3)

# Each animal gains a centroid
add_point(af, across = "keypoint")

# A median instead, robust to a stray keypoint
add_point(af, across = "keypoint", method = "median")

# The midpoint of two keypoints
add_point(af, across = "keypoint", include = c("head", "neck"), name = "neck_head")

# A custom rule
add_point(af, across = "keypoint", method = \(x) mean(x, trim = 0.1), name = "trimmed")

# One point per keypoint, across the animals
add_point(af, across = "individual")
```

Description

Computes how winding the path is (straightness, sinuosity and E_max) over a window centred on each row, and returns the frame with a column for each. Everything they need is computed internally and not added, so only the three measures appear.

Usage

```
add_tortuosity(data, window_width = 11L, step_length = "auto")
```

Arguments

data	A Cartesian anipoint.
window_width	Size of the sliding window, in observations (default 11L). Should be an odd number ≥ 3 for symmetric centring.
step_length	[Experimental] The step length the path is rediscrctised at for sinuosity and E_max, in the frame's spatial unit (default "auto"). "auto" takes each trajectory's mean step, weighted by step length (see Details). A positive number rediscrctises every trajectory at that step, so that their sinuosity can be compared at one scale. Straightness does not use it.

Details

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

Works on 1D, 2D and 3D Cartesian data, reading the axes from the frame's declared variables.

The window is centred on each timepoint. Near the ends of a trajectory, where the window would run past the first or last observation, the metrics are NA.

Straightness is the distance between the positions at the ends of the window over the distance travelled between them. It is NA for a window that holds a missing position: how far the animal moved across the gap is not known, so neither is the distance travelled.

Sinuosity and E_max come from the turning angles of the path rediscrctised to a constant step length, as Benhamou (2004) defines sinuosity: walking along the path, a new point is placed wherever it first leaves a circle of that radius around the last one (Bovet & Benhamou 1988). By default (**step_length** = "auto") the step length is the trajectory's mean step between rows, weighted by step length: the average step over the distance travelled, which time spent still does not shorten. Sinuosity and E_max describe the path at the scale of that step, and the automatic step differs between trajectories, so to compare trajectories, give them all the same **step_length**. Tracking jitter that stays within the circle while an animal is still gives no steps and no turning, where turning angles between successive frames would be dominated by it. Each window takes the turning at the rediscrctised points the path reaches within it, and is NA when there are none: where the animal moved less than a step, its path has no sinuosity to measure. The path is rediscrctised once per trajectory, and missing positions break it into stretches that are rediscrctised separately.

Value

The input anipoint with three columns added, named with the window width, so that `window_width = 11` gives:

straightness_11 Straightness index (D/L), from 0 to 1. NA where a position in the window is missing.

sinuosity_11 Corrected sinuosity index (Benhamou 2004).

e_max_11 Maximum expected displacement (dimensionless).

The width in the name keeps these windowed measures apart from the whole-path **straightness**, **sinuosity** and **e_max** of `summarise_path()`, and lets several window widths sit side by side.

References

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

Bovet, P., & Benhamou, S. (1988). Spatial analysis of animals' movements using a correlated random walk model. *Journal of Theoretical Biology*, 131(4), 419-433. doi:10.1016/S00225193(88)800389.

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. doi:10.1016/j.jtbi.2004.03.016.

Cheung, A., Zhang, S., Stricker, C., & Srinivasan, M. V. (2007). Animal navigation: the difficulty of moving in a straight line. *Biological Cybernetics*, 97(1), 47-61. doi:10.1007/s0042200701580.

See Also

- `add_kinematics()` for speed, course and the turning measures.
- `summarise_path()` for the same measures over each whole trajectory.

Examples

```
data <- anicore::example_anipoint(n_obs = 30, n_individuals = 1, n_keypoints = 1)

data |>
  add_tortuosity(window_width = 11)

# Several window widths side by side
data |>
  add_tortuosity(window_width = 5) |>
  add_tortuosity(window_width = 11)

# Sinuosity of every trajectory at one scale, in the frame's spatial unit
data |>
  add_tortuosity(window_width = 11, step_length = 0.5)
```

<code>compute_emax</code>	<i>Compute $E_{\max}$ (maximum expected displacement) from pre-computed vectors</i>
---------------------------	--

Description

Compute $E_{\max}$ (maximum expected displacement) from pre-computed vectors

Usage

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

Arguments

<code>mean_cos_turning</code>	Numeric vector of mean cosine of turning angles.
<code>mean_step_length</code>	Numeric vector of mean step lengths (required if <code>dimensional = TRUE</code>).
<code>dimensional</code>	Logical. If <code>TRUE</code> , 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.

Examples

```
compute_emax(mean_cos_turning = 0.8)

# The dimensional form also accounts for step length
compute_emax(0.8, mean_step_length = 1.2, dimensional = TRUE)
```

<code>compute_gradient</code>	<i>Compute numerical derivatives on possibly uneven grids</i>
-------------------------------	---

Description

This helper implements a finite-difference scheme based on the **Fornberg formula** for interior points when the spacing between coordinates is non-uniform. End points use first-order forward/backward differences.

Usage

```
compute_gradient(values, coords = NULL)
```

Arguments

values	Numeric vector of function values.
coords	Numeric vector of the corresponding coordinate values. Must be the same length as values .

Value

A numeric vector of the same length as **values** containing the estimated first derivative at each point.

References

Fornberg, B. (1988). Generation of finite difference formulas on arbitrarily spaced grids. *Mathematics of Computation*, 51(184), 699-706. doi:10.1090/S00255718198809350770

Examples

```
# The derivative of x^2 is 2x, which the interior points recover exactly
compute_gradient(c(0, 1, 4, 9, 16))

# The spacing between coordinates need not be even
compute_gradient(c(0, 1, 4, 9), coords = c(0, 1, 3, 6))
```

<code>compute_min_step</code>	<i>The minimum step "auto" chooses for each trajectory</i>
-------------------------------	--

Description**[Experimental]**

Returns the threshold that `min_step = "auto"` uses in `add_kinematics()` and `summarise_path()`, one per trajectory, with the positional noise it is estimated from. Use it to see whether the automatic threshold suits your data, to report it, or to choose a threshold of your own relative to it.

The threshold is computed by the same code that `add_kinematics()` uses, so the two always agree. See the section "The minimum step" in `add_kinematics()` for how it is estimated and when to set it yourself.

Usage

```
compute_min_step(data)
```

Arguments

data	An anipoint with two or three spatial axes, grouped one trajectory per group, as it is by its declared keys. Any coordinate system works; non-Cartesian frames are converted to Cartesian for the computation.
-------------	--

Value

A data frame with one row per trajectory: the key columns, then

- **positional_noise**: the estimated standard deviation of the tracking noise on each axis, in the frame's spatial unit. 0 when the trajectory is too short, or moves too little, to estimate it.
- **min_step**: the threshold "auto" uses, in the frame's spatial unit: three times **positional_noise**, or half the trajectory's median step if that is smaller. Where it is less than three times **positional_noise**, the cap on the median step set it.

See Also

[add_kinematics\(\)](#), whose **min_step** takes the threshold.

Examples

```
af <- anicore::example_anipoint(n_obs = 50, n_individuals = 2, n_keypoints = 1)
compute_min_step(af)

# One threshold for every trajectory, so their turning is filtered alike
thresholds <- compute_min_step(af)
add_kinematics(af, min_step = max(thresholds$min_step))
```

compute_nnd

Compute nearest neighbour distances within one group

Description

Low-level function behind [add_nnd\(\)](#), operating on plain vectors for one group of comparable points (typically one timepoint).

Usage

```
compute_nnd(
  x,
  y,
  z = NULL,
  across,
  n = 1L,
  is_focal = NULL,
  is_candidate = NULL,
  labels = NULL
)
```

Arguments

x, y	Numeric vectors of coordinates.
z	Numeric vector of coordinates, or NULL for 2D data.
across	Vector whose value must differ between a focal point and its neighbour — the thing being ranked, e.g. individual identity.
n	Which neighbour to return (1 = nearest, 2 = second nearest).
is_focal	Logical vector marking which points to measure from, or NULL for all of them. Non-focal points get NA results.
is_candidate	Logical vector marking which points may be returned as a neighbour, or NULL for all of them.
labels	Named list of vectors describing each point. The matched neighbour's value is reported for each, as nnd_<name> .

Details

For each focal point, candidates are ranked by the value of **across**: the closest candidate point is found for each distinct value, those values are ranked by that distance, and the **n**-th is returned. So "second nearest" means the second nearest *entity*, not the second nearest point.

Value

A tibble with **nnd_across** (the value of **across** for the matched neighbour), one **nnd_<name>** column per entry of **labels**, and **nnd_distance**.

See Also

[add_nnd\(\)](#) for the anipoint-level function.

Examples

```
# Nearest point belonging to a different individual
compute_nnd(
  x = c(0, 1, 2),
  y = c(0, 1, 0),
  across = c("a", "b", "c")
)

# The second nearest, with the neighbour's label reported back
compute_nnd(
  x = c(0, 1, 2),
  y = c(0, 1, 0),
  across = c("a", "b", "c"),
  n = 2L,
  labels = list(individual = c("a", "b", "c"))
)
```

compute_point	<i>Compute a derived point of an identity level</i>
---------------	---

Description

The point `add_point()` appends, on its own: one row per position of every other identity variable, derived from the members of the collapsed one.

Usage

```
compute_point(
  data,
  across = NULL,
  method = "centroid",
  include = NULL,
  exclude = NULL,
  name = NULL
)
```

Arguments

data	An anipoint with Cartesian coordinates.
across	Identity variables to collapse — the dimensions the new point is derived over. Required when the frame declares more than one identity variable, since their order is not a hierarchy and there is no finest one to assume; with a single identity variable, that one is the default. Collapsing every level gives a single point per position.
method	How each coordinate is derived from the members' values: "centroid" The mean. "median" The median, per axis: robust to a single stray point. "weighted" The mean weighted by confidence , so poorly tracked points count for less. Needs a confidence column. a function Applied to each axis's values, with missing values removed, returning one number; e.g. <code>\(x) mean(x, trim = 0.1)</code> . Missing values are ignored; a moment where every member is missing gives NA.
include, exclude	Values of the collapsed level to keep or leave out. Only meaningful when one level is collapsed. A midpoint is the centroid of two members: <code>include = c("ear_l", "ear_r")</code> .
name	Name for the new member. Defaults to "centroid", "median" or "weighted_centroid" after method ; required when method is a function.

Value

An anipoint containing only the new member. Its **confidence**, if the frame has one, is NA.

Examples

```
af <- anicore::example_anipoint(n_obs = 20, n_individuals = 2, n_keypoints = 3)

# The centroid of each animal's keypoints
compute_point(af, across = "keypoint")

# Their median
compute_point(af, across = "keypoint", method = "median")
```

compute_sinusosity	<i>Compute sinusosity index from precomputed vectors</i>
--------------------	--

Description

Compute sinusosity index from precomputed vectors

Usage

```
compute_sinusosity(
  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 sinusosity values

Examples

```
compute_sinusosity(mean_step_length = 1.2, mean_cos_turning = 0.8)

# The original formulation, for comparison with older work
compute_sinusosity(1.2, 0.8, method = "original")
```

`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)

Examples

```
# Straight-line displacement over the distance actually travelled
compute_straightness(5, 10)

# A perfectly straight path scores 1
compute_straightness(10, 10)
```

`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.

References

Fornberg, B. (1988). Generation of finite difference formulas on arbitrarily spaced grids. *Mathematics of Computation*, 51(184), 699-706. doi:10.1090/S00255718198809350770

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)
```

summarise_aniframe	<i>Summarise the time series of an aniframe</i>
--------------------	---

Description

[Experimental]

Summarises the per-row measures of a frame over each group: a measure of central tendency and of dispersion for each, one row per group. It describes the measures the frame already carries — the output of `add_kinematics()`, `add_tortuosity()` or your own `mutate()` — and computes no new ones. Any grouping is allowed, since a median of rows means the same whether the rows are one keypoint's or a whole animal's.

Properties of a trajectory as a whole, such as how far it ended up from where it started, are not a statistic of any column; see `summarise_path()`.

Each frame class has its own default set of measures:

anipoint speed, acceleration, turning_speed, turning_rate, turning_acceleration, course_elevation, the windowed measures of `add_tortuosity()` at every window width present (`straightness_11`, `sinuosity_11`, `e_max_11`, ...), and confidence; circular statistics for course and for a declared yaw, reported as `*_heading`.

anisegment length and confidence.

anijoint angle, with circular statistics, and confidence.

Only the measures present are summarised. The windowed measures are recognised by their names, `straightness`, `sinuosity` or `e_max` followed by `_` and a window width, so a frame with several widths has each summarised. The unsuffixed `straightness`, `sinuosity` and `emax` are summarised too. Velocity and acceleration components, `course_unwrapped`, and the running totals `cumulative_distance` and `cumulative_turning` are left out by default; `summarise_path()` reports the totals.

Angles are summarised with circular statistics (`anicore::circ_mean()` and its siblings), so that the mean of 350 and 10 degrees is 0 rather than 180. They are computed in radians and reported in the frame's `unit_angle`. Only the angles `animetric` knows of are treated as circular; any other column is summarised as a linear quantity.

Usage

```
summarise_aniframe(data, ...)

summarize_aniframe(data, ...)

## S3 method for class 'anipoint'
summarise_aniframe(
  data,
  cols = NULL,
  measures = c("median_mad", "mean_sd"),
  ...
)

## S3 method for class 'anisegment'
summarise_aniframe(
  data,
  cols = NULL,
  measures = c("median_mad", "mean_sd"),
  ...
)

## S3 method for class 'anijoint'
summarise_aniframe(
  data,
  cols = NULL,
  measures = c("median_mad", "mean_sd"),
  ...
)

## S3 method for class 'anievent'
summarise_aniframe(data, ...)

## Default S3 method:
summarise_aniframe(data, ...)
```

Arguments

data	An anipoint, anisegment or anijoint.
...	Not used.
cols	Character vector of columns to summarise, in place of the class's default set. A known angle among them is still summarised with circular statistics.
measures	Measures of central tendency and dispersion: "median_mad" (default) or "mean_sd".

Value

A data frame with one row per group: the grouping columns, then `<measure>_<column>` for each measure, e.g. `median_speed` and `mad_speed`.

See Also

[summarise_path\(\)](#) for whole-trajectory measures.

Examples

```
kin <- add_kinematics(
  anicore::example_anipoint(n_obs = 20, n_individuals = 1, n_keypoints = 1)
)
summarise_aniframe(kin)

# Mean and standard deviation instead of median and MAD
summarise_aniframe(kin, measures = "mean_sd")

# Only some measures
summarise_aniframe(kin, cols = c("speed", "course"))
```

<code>summarise_path</code>	<i>Summarise each trajectory as a whole</i>
-----------------------------	---

Description

Measures of a path that are not a statistic of any per-row column: how long it was, where it ended up relative to where it started, and how directly it got there. Each needs one trajectory per group, since the distance between the ends of several pooled trajectories describes nothing.

Positions are all it needs: the kinematics are computed internally, so [add_kinematics\(\)](#) need not be run first. Any coordinate system works; non-Cartesian frames are converted to Cartesian for the computation.

For the distribution of per-row measures over each group, such as median speed or the median of windowed straightness, see [summarise_aniframe\(\)](#).

Usage

```
summarise_path(data, min_step = "auto", step_length = "auto")
```

```
summarize_path(data, min_step = "auto", step_length = "auto")
```

Arguments

data	An anipoint, grouped one trajectory per group, as it is by its declared keys.
min_step	[Experimental] The shortest step whose direction counts toward total_turning , as in add_kinematics() (default "auto"). 0 counts every direction, however short the step.
step_length	[Experimental] The step length the path is rediscretised at for sinuosity and e_max , in the frame's spatial unit (default "auto"). "auto" takes each trajectory's mean step, weighted by step length (see Details). A positive number rediscretises every trajectory at that step, so that their sinuosity can be compared at one scale.

Details

Sinuosity is defined for a path of constant step length (Benhamou 2004), so **sinuosity** and **e_max** come from the turning angles of the path rediscretised to one: walking along the path, a new point is placed wherever it first leaves a circle of that radius around the last one (Bovet & Benhamou 1988). By default (**step_length** = "auto") the step length is the trajectory's mean step between rows, weighted by step length: the average step over the distance travelled, which time spent still does not shorten. Sinuosity and **E_max** describe the path at the scale of that step, and the automatic step differs between trajectories: a faster animal gets a longer one. To compare trajectories, give them all the same **step_length**. Tracking jitter that stays within the circle while an animal is still gives no steps and no turning, where turning angles between successive frames would be dominated by it. Missing positions break the path into stretches that are rediscretised separately.

Value

A data frame with one row per trajectory:

- **total_distance**: distance travelled, the last value of [add_kinematics\(\)](#)'s **cumulative_distance**.
- **total_turning**: turning of the direction of travel, summed (2D and 3D), in the frame's **unit_angle**: the last value of [add_kinematics\(\)](#)'s **cumulative_turning**, so steps shorter than **min_step** add none.
- **net_displacement**: straight-line distance from start to end.
- **straightness**: net displacement over distance travelled, from 0 to 1.
- **sinuosity**: corrected sinuosity index (Benhamou 2004), of the path rediscretised to a constant step length (see Details).
- **e_max**: maximum expected displacement (dimensionless), from the same rediscretised path.

The windowed measures of `add_tortuosity()` carry the window width in their names (`straightness_11`), so they never collide with these.

References

- Benhamou, S. (2004). How to reliably estimate the tortuosity of an animal's path. *Journal of Theoretical Biology*, 229(2), 209-220. doi:[10.1016/j.jtbi.2004.03.016](https://doi.org/10.1016/j.jtbi.2004.03.016).
- Bovet, P., & Benhamou, S. (1988). Spatial analysis of animals' movements using a correlated random walk model. *Journal of Theoretical Biology*, 131(4), 419-433. doi:[10.1016/S00225193\(88\)800389](https://doi.org/10.1016/S00225193(88)800389).

Examples

```
traj <- anicore::example_anipoint(n_obs = 20, n_individuals = 1, n_keypoints = 1)
summarise_path(traj)

# Count every direction toward total_turning, however short the step
summarise_path(traj, min_step = 0)

# Sinuosity at a step of your own, in the frame's spatial unit, the same
# for every trajectory
summarise_path(traj, step_length = 0.5)
```

Index

- * **kinematics**
 - compute_gradient, 14
- add_kinematics, 2
- add_kinematics(), 10, 13, 15, 16, 21, 23, 24
- add_nnd, 6
- add_nnd(), 16, 17
- add_orientation, 8
- add_point, 10
- add_point(), 18
- add_tortuosity, 11
- add_tortuosity(), 5, 21, 25
- anicore::circ_mean(), 22
- anicore::get_angle_direction(), 5
- anicore::get_variables(), 6
- anicore::reflect_axis(), 5
- anispace::quat_from_vectors(), 9
- anispace::quat_mean(), 11
- compute_emax, 14
- compute_gradient, 14
- compute_min_step, 15
- compute_min_step(), 3, 4
- compute_nnd, 16
- compute_nnd(), 7
- compute_point, 18
- compute_point(), 11
- compute_sinusosity, 19
- compute_straightness, 20
- differentiate, 20
- stats::mad(), 3
- summarise_aniframe, 21
- summarise_aniframe(), 10, 23
- summarise_path, 23
- summarise_path(), 5, 13, 15, 21–23
- summarize_aniframe
 - (summarise_aniframe), 21
- summarize_path (summarise_path), 23