Package ‘fishmechr’

May 8, 2026

Title Computes Kinematic Parameters for Swimming

Version 1.0.3

Description Processes tracked points on a fish's body and uses them to estimate standard kinematic parameters such as tail beat amplitude and frequency, body wavelength and wavespeed. As part of this, it also estimates the location of the center of mass and the principal swimming axis. The techniques are described in detail in the main vignette and are published in the book chapter Hawkins, O.H., Di Santo, V., Tytell, Eric.D., 2025. ``Biomechanics and energetics of swimming'', in: Higham, T.E., Lauder, G.V. (Eds.), Integrative Fish Biomechanics, Fish Physiology. Academic Press. <doi:10.1016/bs.fp.2025.06.003>.

License CC BY 4.0

Encoding UTF-8

RoxygenNote 7.3.3

Depends R (>= 4.4.0)

LazyData true

Imports assertthat, cli, dplyr, gsignal, pracma, rlang, tibble, tidyr, tidyselect

Suggests ggplot2, knitr, patchwork, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

URL `https://tytell.github.io/fishmechr/`, `https://github.com/tytell/fishmechr`

BugReports `https://github.com/tytell/fishmechr/issues`

NeedsCompilation no

Author Eric Tytell [aut, cre, cph] (ORCID: <`https://orcid.org/0000-0002-6603-9448`>), Olivia Hawkins [ctb] (ORCID: <`https://orcid.org/0000-0001-9373-5919`>), National Science Foundation [fnd]

Maintainer Eric Tytell <`eric.tytell@tufts.edu`>

Repository CRAN

Date/Publication 2026-03-29 15:40:02 UTC

Contents

`apply_filter` *Apply a filter constructed with build_filter*

Description

Wrapper function for `gsignal::filtfilt` to apply a filter to a dataset. Potentially skips NAs in the data set (see `skip_na()` for details).

Usage

```
apply_filter(filt, x, na.skip = TRUE)
```

Arguments

`filt`	Filter as returned from `build_filter()` or `gsignal::butter()`
`x`	Vector of data to filter
`na.skip`	(TRUE or FALSE) to skip NAs in the data set.

Value

Filtered data set

Examples

```
filt <- build_filter(hi = 5, sampfreq = 100)
# 2 Hz signal sampled at 100 Hz with high-frequency noise and two NA gaps
x <- sin(2 * pi * (1:100) / 100 * 2) + rnorm(100, sd = 0.3)
x[c(30, 31, 70)] <- NA
apply_filter(filt, x)
```

`arclength` *Calculate arc length along a 2D curve*

Description

Computes the straight-line distance between points on the curve and then adds them up to get the arc length.

Usage

```
arclength(x, y, na.skip = FALSE)
```

Arguments

`x, y`	Coordinates of the curve.
`na.skip`	(TRUE or FALSE) If TRUE, skip NAs and compute arc length for the non-NA points, returning NA for the NA positions. If FALSE (default), return all NAs if any input point is NA.

Value

Arc length along the curve.

Examples

```
# compute arc length in each frame from the lamprey data set
library(dplyr)
lampreydata |>
  group_by(frame) |>
  mutate(arclen = arclength(mxmm, mymm))
```

`build_filter` *Constructs a smoothing filter*

Description

Wrapper function for `gsignal::butter` to construct a bandpass (or low or high pass) filter at a particular frequency. Uses the Sos form for the filter, which works better numerically particularly for very low frequencies.

Usage

```
build_filter(lo = NULL, hi = NULL, sampfreq, order = 13)
```

Arguments

`lo`	Lower frequency cutoff in Hz. If only `lo` is specified, then `build_filter` constructs a high pass filter
`hi`	Upper frequency cutoff in Hz. If only `hi` is specified, then `build_filter` constructs a high pass filter
`sampfreq`	Sampling frequency for the data in Hz.
`order`	(optional) Order for the filter

Value

Filter parameters in Sos form

Examples

```
# Low pass filter with a cutoff at 0.5Hz for data sampled at 100Hz
lopass <- build_filter(lo=0.5, sampfreq=100)

# Band pass filter that passes frequencies between 0.5 and 10Hz
bandpass <- build_filter(lo=0.5, hi=10, sampfreq=100)
```

`curvature` *Estimates curvature for a single curve*

Description

Estimates curvature either directly through derivatives of the x and y coordinates relative to arc length, or as the derivative of segment angle relative to arc length.

Usage

```
curvature(s, x, y, method = "angle")
```

Arguments

`s`	Arc length along the body.
`x, y`	Coordinates of each point along the body
`method`	("xy" or "angle") for the direct formula or for the angle derivative

Details

Assumes that points are in order along the curve from head to tail.

Value

Curvature.

`deriv` *Estimate first or second derivatives for dy/dx.*

Description

Uses central differencing where possible.

Usage

```
deriv(x, y, ord = 1, method = "direct", ends = "forwardback")
```

Arguments

`x`	x variable. Does not need to be evenly spaced.
`y`	y variable.
`ord`	Order of the derivative (1 or 2).
`method`	Method for taking second derivatives. Either • 'direct' (default) Uses a direct formula, based on a central difference of forward and backward differences, from https://mathformeremortals.wordpress.com/2013/01/12/a-numerical-second-derivative-from-three-points/ • 'repeat' Repeat two first derivatives.
`ends`	('forwardback', 'NA', or 'drop') How to handle the endpoints where central differencing is not possible. `'forwardback'` (default) uses forward differencing at the first point and backward differencing at the last. `'NA'` sets endpoints to NA. `'drop'` removes the endpoints.

Value

Derivative of y relative to x.

`find_gaps_df` *Find gaps in a data series*

Description

Looks for NAs (gaps) in a data series and counts the number of NAs. Useful for `smooth_points_df`, which can fill gaps up to a certain length. 0 means no gap, 1 means a single NA, and so forth.

Usage

```
find_gaps_df(.data, cols, .frame = frame, .out = c("gaplen"))
```

Arguments

`.data`	Data frame containing the midlines.
`cols`	Columns containing the components to be smoothed. Often these will be the x and y coordinates of the midline.
`.frame`	Column identifying frames (default is `frame`)
`.out`	Name of the column to contain the length of the gap.

Value

The data frame with a new column.

Examples

```
# create a data frame with two NA gaps of different lengths
df <- data.frame(frame = 1:10,
                 x = c(1, 2, NA, NA, 5, 6, 7, NA, 9, 10))
find_gaps_df(df, x)
```

`fishwidth` *Fish body width profiles*

Description

Body width (diameter) as a function of fractional arc length for an eel (*Anguilla rostrata*) and a sea lamprey (*Petromyzon marinus*). Used for estimating the center of mass via `get_midline_center_df()`.

Usage

```
fishwidth
```

Format

A data frame with 20 rows and 3 columns:

s Arc length as a fraction of body length (0 = head, 1 = tail)

eelwidth Body width of the eel as a fraction of body length

ammowidth Body width of the lamprey as a fraction of body length

---

`get_body_cycle_numbers_df` *Gets oscillation cycle numbers for a midline data set*

Description

Extracts the phase for a specific point along the body (defined by `pointval`) and uses `get_cycle_numbers()` to get the cycle number. Joins the cycle number data with the original data set so that cycle number is defined for each point along the body

Usage

```
get_body_cycle_numbers_df(
  .data,
  ph,
  pointval,
  .frame = frame,
  .point = point,
  .out = NULL,
  overwrite = TRUE,
  ...
)
```

Arguments

`.data`	Data frame containing the midlines
`ph`	Phase variable
`pointval`	Specific point on the body to use to define the phase and and cycle number
`.frame, .point`	Columns identifying frames and points (defaults are `frame` and `point`)
`.out`	Name of the output column. Needs to have one element specifying the name for the cycle variable, or as a named list with an element named `cycle`). Default is 'cycle'
`overwrite`	TRUE or FALSE to overwrite existing columns, if present.
`...`	Extra parameters to supply to `get_cycle_numbers()`

Value

A data frame containing a new column with the cycle numbers named 'cycle' or as specified in .out.

Examples

```
library(dplyr)
# get phase at each frame for one point, then label each complete tail-beat cycle
# this is a minimal example; check the vignette for more details. In particular,
# you generally need to do much more smoothing on the data set for it to give
# good output
lampreydata |>
   group_by(frame) |>
   mutate(arclen = arclength(mxmm, mymm),
          curve_ang = curvature(arclen, mxmm, mymm)) |>
   group_by(point) |>
   mutate(ph_c = hilbert_phase(curve_ang)) |>
   ungroup() |>
   get_body_cycle_numbers_df(ph_c, pointval = 18)
```

`get_cycle_numbers` *Gets cycle numbers from a phase variable*

Description

Given a phase variable that increases, each time the phase passes through 2 k pi, a new cycle starts. This function unwraps the phase, so that it increases steadily, then takes the floor of the phase divided by 2 pi (or another modulus), so that it gets an integer for each cycle.

Usage

```
get_cycle_numbers(
  ph,
  unwrap = FALSE,
  mod = 2 * pi,
  exclude_partial_cycles = TRUE
)
```

Arguments

`ph`	Phase variable
`unwrap`	(TRUE or FALSE) Unwrap the phase variable. Note that the function will not work unless the phase is unwrapped, so you should only set unwrap to FALSE if the phase has been unwrapped earlier.
`mod`	Modulus for the phase. Default is `2*pi`.
`exclude_partial_cycles`	(TRUE or FALSE) Exclude cycles in which the phase does not advance from close to 0 to close to 2pi.

Details

Optionally, it will try to exclude partial cycles, setting the cycle number to NA for cycles that do not start at a phase close to 0 and progress to a phase close to 2 pi. "Close" here is defined based on the average change in phase.

Value

Integer cycle numbers with the same size as `ph`

See Also

`get_body_cycle_numbers_df()`

Examples

```
# example phase that advances by slightly more than 3 cycles, modulo 2pi
ph <- seq(0, 20, by = pi/10) %% (2*pi)
get_cycle_numbers(ph, unwrap=TRUE)
```

`get_frequency` *Estimates the cycle frequency based on time and phase*

Description

Estimates frequency based on time and phase by one of two methods:

- 'deriv' Takes the derivative of phase vs time to get frequency
- 'slope' Fits a line to phase vs time and uses the slope as an estimate of frequency.

Usage

```
get_frequency(
  t,
  ph,
  unwrap = FALSE,
  method = "deriv",
  mod = 2 * pi,
  check_reasonableness = TRUE
)
```

Arguments

`t`	Time
`ph`	Phase
`unwrap`	(TRUE or FALSE) Unwrap the phase. Note that the function will not work unless the phase is unwrapped, so you should only set unwrap to FALSE if the phase has been unwrapped earlier.
`method`	('deriv' or 'slope') Method to estimate the frequency. 'deriv' takes the derivative of phase vs time; 'slope' fits a line to phase vs time and uses the slope.
`mod`	Modulus for the phase. Default is `2*pi`.
`check_reasonableness`	Runs some simple checks to make sure the data make sense. Checks to make sure phase is mostly increasing and warns if it isn't.

Value

The frequency estimate, either as a vector the same size as `ph` (for the 'deriv' algorithm) or as a single value (for the 'slope' algorithm)

Examples

```
t <- seq(0, 3, by=0.1)
# example phase that has a frequency of exactly 2.3 Hz
ph <- 2*pi*(t * 2.3)

get_frequency(t, ph)
```

`get_midline_center_df` *Gets the center of a midline for many midlines in a data frame*

Description

Estimates the center of a midline based on mass distribution, volume distribution, or body width.

Usage

```
get_midline_center_df(
  .data,
  arclen,
  x,
  y,
  mass,
  width,
  height,
  .out = NULL,
  .frame = frame,
  .point = point,
  excludepoints = c(),
  method = "mutate",
  overwrite = TRUE
)
```

Arguments

`.data`	Data frame containing the midlines.
`arclen`	The column containing the arc length. See `arclength()`
`x, y`	Columns containing the x and y coordinates of the midline. There should be N points.
`mass`	(optional) Column containing the mass of each segment, with N-1 segments.
`width`	(optional) Column containing the horizontal plane width of the body at each midline point (N points)
`height`	(optional) Column containing the dorso-ventral height of the body at each midline point (N points)
`.out`	Names of the output columns. Needs to have two elements specifying the names for the x and y coordinates of center position. Or it can be a named list containing at least some of the elements `xctr` and `yctr`. If the return elements aren't in the named list, the defaults are 'xcom' and 'ycom')
`.frame, .point`	Columns identifying frames and points (defaults are `frame` and `point`)
`excludepoints`	Exclude these points when estimating center. Some points (like the tip of the tail) have relatively little mass and are hard to track, so can introduce errors.
`method`	'mutate' or 'summarize'. If summarize, returns one center position for each frame. If mutate, returns a same center position repeated for each point in a frame.
`overwrite`	TRUE or FALSE to overwrite existing columns, if present.

Details

Given a mass distribution, it produces an estimates of the true center of mass. If given the body width and height, it assumes that the body has an oval cross section with varying width and height, and it estimates the volume distribution. This method will give a good estimate of the center of mass if the body has close to uniform density. If given just the width, it uses the width to estimate a weight average centroid position.

Value

A data frame containing the original variables along with xcom, ycom (or names as specified in `.out`). The center of each midline in each frame.

Examples

```
library(dplyr)
lampreydata |>
  group_by(frame) |>
  mutate(arclen = arclength(mxmm, mymm),
         width = interpolate_width(fishwidth$s, fishwidth$ammowidth, arclen)) |>
  ungroup() |>
  get_midline_center_df(arclen, mxmm, mymm, width = width)
```

`get_primary_swimming_axis`

Gets the main swimming axis from a midline

Description

Computes the main swimming axis of a midline as a unit vector, using the singular value decomposition (`svd()`). This only works well if the midlines are centered around zero, so it optionally subtracts off the mean of x and y. For more sophisticated centering algorithms, see `get_midline_center_df()`.

Usage

```
get_primary_swimming_axis(x, y, center = TRUE, na.rm = FALSE)
```

Arguments

`x, y`	Coordinates of the midline
`center`	(TRUE or FALSE) Subtract the mean from the x and y coordinates
`na.rm`	(default FALSE) Remove NA points before computing the SVD

Value

A data frame with the following columns:

- `swimaxis_x`, `swimaxis_y` x and y components of the swimming axis vector
- `swimaxis_xctr`, `swimaxis_yctr` Mean x and y values that were subtracted before running the SVD

See Also

`get_midline_center_df()`

Examples

```
library(dplyr)
library(tidyr)
# run the algorithm across multiple midlines at different times
lampreydata |>
  group_by(t) |>
  summarize(swimaxis = get_primary_swimming_axis(mxmm, mymm)) |>
  unnest(swimaxis)
```

`get_primary_swimming_axis_df` *Gets the primary swimming axis for many midlines*

Description

Processes midlines from many frames of a video

Usage

```
get_primary_swimming_axis_df(
  .data,
  tm,
  x,
  y,
  cutoff = NULL,
  overwrite = TRUE,
  .out = NULL,
  .frame = frame,
  .point = point,
  check_reasonableness = TRUE,
  na.rm = FALSE
)
```

Arguments

`.data`	Data frame containing the midline data
`tm`	Column containing the time data. If a cutoff frequency is passed in, then this variable will be used to get the sampling frequency.
`x, y`	Columns containing the x and y coordinates of each point along the midline.
`cutoff`	(optional) If this parameter is included, smooth the swimming axis data with a low-pass filter with a cutoff at this frequency.
`overwrite`	(default TRUE). Overwrite output columns if they exist
`.out`	Names of the output columns. Needs to have four elements specifying the names for the x and y coordinates of the swim axis and the parallel and perpendicular components of the excursion, in that order. Or it can be a named list containing at

	least some of the elements `swimaxis_x`, `swimaxis_y`, `exc_x`, `exc`, in any order. If the return elements aren't in the named list, the defaults are 'swimaxis_x', 'swimaxis_y', 'exc_x', and 'exc')
`.frame, .point`	Columns identifying frames and points (defaults are `frame` and `point`)
`check_reasonableness`	(default TRUE). Run some checks that the data are reasonable before processing.
`na.rm`	(default FALSE) Remove NA points before computing the SVD

Details

Uses `get_primary_swimming_axis()` to compute the swimming axis for a midline. Then optionally smooths the axis using a Butterworth filter, and then projects the midlines on to the new time-varying axes.

Value

A data frame containing the original variables along with

- XX_ctr,YY_ctr: The center of each midline at each time, where XX and YY are the original names of the x and y coordinates.
- exc,exc_x: The new midlines centered and projected on to the swimming direction and the perpendicular axis. `exc` is useful as the lateral excursion of the swimming undulation.

Examples

```
library(dplyr)
# subtract the center of mass, then estimate the primary swimming axis
lampreydata |>
  group_by(frame) |>
  mutate(arclen = arclength(mxmm, mymm)) |>
  ungroup() |>
  get_midline_center_df(arclen, mxmm, mymm) |>
  mutate(mxmm_ctr = mxmm - xcom, mymm_ctr = mymm - ycom) |>
  get_primary_swimming_axis_df(t, mxmm_ctr, mymm_ctr)
```

`get_volume` *Gets the volume of segments of a cylindrical body with elliptical cross section*

Description

Used for estimating the center of mass of a fish. If we know the width and height profile, and we assume that the cross section is elliptical, then we can estimate the volume of each segment as the volume of a truncated elliptical cone.

Usage

```
get_volume(arclen, width, height)
```

Arguments

`arclen, width, height`	Arc length, width and height. Should have the same units. N points

Details

The formula for such a cone is V = pi s (w h + 1/2 dw h + 1/2 dh w + 1/3 dw dh) where s is the length of the cone, w and h are the half width and height, and dw and dh are the difference in width or height from one end to the other (e.g., dw = w(s) - w(0) if w is a function of s)

Value

Volume of each segment (N-1 values). Last value will be NA

Examples

```
# volume of lamprey body segments using measured width and estimated height
h <- seq(0.05, 0.03, length.out = nrow(fishwidth))  # height tapers head to tail
get_volume(fishwidth$s, fishwidth$ammowidth, h)
```

`get_wavelength` *Computes the body wavelength based on the phase at each point and the arc length*

Description

Computes the spatial derivative in phase relative to arc length using several different possible methods:

- 'deriv' Computes the derivative directly using `deriv()`
- 'slope' Fits a line to the phase relative to arc length and uses the slope of that line as an estimate of the wavelength
- 'cycle' Looks for pairs of points along the body where the phase differs by a full cycle. The arc length between those points is one wavelength.
- 'halfcycle' Looks for pairs of points along the body where the phase differs by one half cycle. The arc length between those points is half of a wavelength.

Usage

```
get_wavelength(
  arclen,
  ph,
  unwrap = TRUE,
  method = "deriv",
  ignore_arclen_vals = NULL,
  sort_arclen = FALSE,
  check_reasonableness = TRUE,
```

```
  mod = 2 * pi,
  traveling_wave_dir = -1
)
```

Arguments

`arclen`	Arc length
`ph`	Phase
`unwrap`	(TRUE or FALSE) Unwrap the phase along the body
`method`	('deriv', 'slope', 'cycle', 'halfcycle') as explained above
`ignore_arclen_vals`	NULL or a function that returns TRUE or FALSE for certain values of arclength where the phase estimate is not reliable. This is often used to exclude points near the head (e.g., `ignore_arclen_vals = \(s) s < 0.3`)
`sort_arclen`	(TRUE or FALSE) Sort the phase values by arc length before computing the wavelength. Useful if arc lengths arrive out of order. (default = FALSE)
`check_reasonableness`	(TRUE or FALSE) Check that the phase decreases along the body as expected for a head-to-tail traveling wave, and warn if it does not. (default = TRUE)
`mod`	Modulus for the phase variable
`traveling_wave_dir`	(1 or -1) Defines the direction of the traveling wave. For a normal head-to-tail traveling wave, use -1 (default)

Value

The wavelength as a vector the same size as the phase variable

Examples

```
s <- seq(0, 1, by=0.1)
# artificial data with a wavelength of exactly 0.6
ph <- 2*pi*((1 - s) / 0.6)

get_wavelength(s, ph, ignore_arclen_vals = \(s) s < 0.3)
```

`hilbert_phase` *Compute phase of an oscillation using the Hilbert transform*

Description

Given a value that oscillates (x), first computes the analytic signal using the Hilbert transform, then the phase of that signal.

Usage

```
hilbert_phase(x, na.skip = TRUE, unwrap = TRUE, check_reasonableness = TRUE)
```

Arguments

`x`	Oscillating signal
`na.skip`	TRUE or FALSE to skip NAs. See `skip_na()` for differences with na.omit (default = TRUE)
`unwrap`	TRUE or FALSE to unwrap the phase signal so that it increases smoothly over time (default = TRUE)
`check_reasonableness`	TRUE or FALSE to run some checks on the input data to make sure that the output will be reasonable (default = TRUE)

Value

Phase (mod 2pi)

Examples

```
t <- seq(0, 4, by=0.1)
# signal that increases in frequency over time
x <- cos(2*pi*t*(2.2 + 0.2*t))
ph <- hilbert_phase(x)
```

`interpolate_peak_location`

Interpolate the location of a peak based on three points

Description

Uses a parabolic approximation to determine the location of a peak from 3 points.

Usage

```
interpolate_peak_location(y, x = c(-1, 0, 1))
```

Arguments

`y`	3 points on the curve, where the peak/trough should be at `y[2]`, so that `y[1]` and `y[3]` are lower/higher, respectively, than `y[2]`
`x`	x coordinates. Defaults to `x = c(-1, 0, 1)`, so that the output is an offset of the peak location from what was originally detected. It could also be the true `x` coordinates, which is important if they're unevenly spaced.

Value

The location of the peak, either as an offset (with default `x`) or as a true `x` coordinate

Examples

```
y <- c(1, 2, 0.5)
interpolate_peak_location(y)
```

`interpolate_points_df` *Interpolates and smooths a 2D curve at new arc length*

Description

For a 2D curve with (x,y) coordinates parameterized by the arc length, interpolate new (x,y) coordinates at new arc lengths. Smooth the input data with a smoothing spline.

Usage

```
interpolate_points_df(
  .data,
  arclen,
  x,
  y,
  arclen_out = NULL,
  spar = 0.8,
  tailmethod = "extrapolate",
  fill_gaps = 0,
  .suffix = "_s",
  .out = NULL,
  overwrite = TRUE,
  .frame = frame,
  .point = point
)
```

Arguments

`.data`	Data frame
`arclen`	Name of the input arc length column in `.data`
`x, y`	Name of the columns that contain the coordinates of points on the curve
`arclen_out`	Vector containing the new arc length
`spar`	Smoothing parameter (ranges from 0 for no smoothing to 1 for high smoothing; see `smooth.spline()` for more details.)
`tailmethod`	('keep', 'extrapolate', or 'NA') Methods to estimate the position of the tail tip if the last value of `arclength_out` is longer than the maximum arc length in the current frame. • 'keep' to keep the existing tail point, even if it is not at the requested arc length • 'extrapolate' to extrapolate a tail tip position, assuming that the curve continues straight

- 'NA' to use replace the tail point with NA in this case.

`fill_gaps`	Fill internal missing points of this size or smaller. (0, default, means no filling; 1 means to fill single missing points)
`.suffix`	(default = '_s') Suffix to append to the names of the arclen, x, and y columns after smoothing and interpolation.
`.out`	Names of the output columns. Defaults are (arclen = 'arclen_s', xs = 'xs', ys = 'ys'). Overrides the `.suffix` parameter if it is included.
`overwrite`	TRUE or FALSE to overwrite existing columns
`.frame`	Name of the frame variable in the data frame
`.point`	Name of the point variable in the data frame

Details

Operates on each frame (as defined in the `.frame` parameter) individually.

Value

A data frame with updated columns for the smoothed and iterpolated arc length, x and y coordinates.

Examples

```
library(dplyr)
# compute arc length, then interpolate all midlines to 20 evenly-spaced points
lampreydata |>
  group_by(frame) |>
  mutate(arclen = arclength(mxmm, mymm)) |>
  ungroup() |>
  interpolate_points_df(arclen, mxmm, mymm)
```

`interpolate_points_frame`

Interpolates x and y points on a curve to different arc lengths

Description

For a 2D curve with (x,y) coordinates parameterized by the arc length, interpolate new (x,y) coordinates at new arc lengths. Smooth the input data with a smoothing spline.

Usage

```
interpolate_points_frame(arclen, x, y, arclen_out, spar = 0.1, fill_gaps = 0)
```

Arguments

`arclen`	Input arc length
`x, y`	Coordinates of points on the curve
`arclen_out`	New arc length
`spar`	Smoothing parameter (ranges from 0 for no smoothing to 1 for high smoothing; see `smooth.spline()` for more details.)
`fill_gaps`	Fill internal missing points of this size or smaller. (0, default, means no filling; 1 means to fill single missing points)

Value

A tibble containing the new interpolated and smoothed x and y coordinates as columns `xs` and `ys`

Examples

```
# get one frame of lamprey midline data
df1 <- lampreydata[lampreydata$frame == 3, ]
# compute arc length along the midline
s <- with(df1, arclength(mxmm, mymm))
# define 20 evenly-spaced output arc lengths from head to tail
s_new <- seq(0, max(s, na.rm = TRUE), length.out = 20)
# interpolate and smooth the midline at the new arc lengths
with(df1, interpolate_points_frame(s, mxmm, mymm, s_new))
```

`interpolate_width` *Interpolates and scales fish body width*

Description

Interpolates the width for a new arc length and scales it based on body length. Assumes that the input width and arc length have the same units (they could be in fractions of body length, cm, or pixels, as long as they are the same). Once the width is estimated at the new arc length, scales it based on the new maximum length.

Usage

```
interpolate_width(arclen0, width0, arclen, scale_to_body_length = TRUE)
```

Arguments

`arclen0`	Arc length for the width measurement. The first value should be at the head and the last value should be at the tail tip.
`width0`	Width measurement. Should have the same units as `arclen0`
`arclen`	New arc length
`scale_to_body_length`	TRUE or FALSE to scale the interpolated width by multiplying by body length. This only works if `arclen` is in real units (like cm) so that the last value in `arclen` is equal to the total length of the fish.

Details

Width here is defined as the distance from one side of the body to the other (like a diameter), not from the center to a side (like a radius).

Value

Width at the new values of arc length, scaled for the new length

---

`lampreydata` *Lamprey midline data*

Description

Midline tracking data for a sea lamprey (*Petromyzon marinus*) swimming steadily. Contains 20 evenly-spaced points along the body from head to tail, across 80 frames.

Usage

```
lampreydata
```

Format

A data frame with 1600 rows and 5 columns:

t	Time in seconds
frame	Frame number
point	Point index along the body (1 = head, 20 = tail)
mxmm	x coordinate in mm
mymm	y coordinate in mm

`peak_phase` *Compute phase of an oscillation by locating peaks and zero crossings.*

Description

For an oscillatory signal, we can define a positive peak as having phase 0 and a negative peak as having phase pi, with the zero crossings or intermediate points having phase pi/2 and 3pi/2. Then, if we find those peaks and zero crossings, we can interpolate to find the phase at any point.

Usage

```
peak_phase(
  x,
  unwrap = TRUE,
  check_reasonableness = TRUE,
  check_ordering = TRUE,
  interpolate_peaks = TRUE,
  interpolate_zeros = TRUE,
  zero_mode = "midpoint",
  ...
)
```

Arguments

`x`	Oscillatory signal
`unwrap`	TRUE or FALSE to unwrap the phase signal so that it increases smoothly over time (default = TRUE)
`check_reasonableness`	TRUE or FALSE to run some checks on the input data to make sure that the output will be reasonable (default = TRUE)
`check_ordering`	TRUE or FALSE to check the order of peaks and zeros. A good signal should have a positive peak followed by a downward zero, then a negative peak followed by an upward zero. (default = TRUE)
`interpolate_peaks`	TRUE or FALSE to interpolate the locations of peaks using a 3-point parabola (see `interpolate_peak_location()`). (default = TRUE)
`interpolate_zeros`	TRUE or FALSE to interpolate the locations of zero crossings linearly (default = TRUE)
`zero_mode`	'midpoint', 'zero', or 'none' or NA. Define zero crossings as the point halfway between a positive and a negative peak ('midpoint'), or as a genuine zero crossing ('zero'). If 'none' or NA, do not detect zeros.
`...`	Other parameters to supply to `pracma::findpeaks()`. 'minpeakdistance', the minimum number of index values between peaks, is often the most useful.

Details

Uses `pracma::findpeaks()` to locate peaks.

Value

The phase of the oscillatory signal.

Examples

```
t <- seq(0, 4, by=0.1)
# signal that increases in frequency over time
```

```
x <- cos(2*pi*t*(2.2 + 0.2*t))
ph <- peak_phase(x)
```

`pivot_kinematics_longer` *Pivots a kinematics dataset into long format*

Description

Converts a wide-format data frame where each point's variables are separate columns (e.g. `head.x`, `head.y`, `tail.x`, `tail.y`) into a long format with one row per point per frame. The point column is returned as a factor with levels in the order given by `pointnames`.

Usage

```
pivot_kinematics_longer(df, pointnames, point_to = "point", sep = "\\.")
```

Arguments

`df`	The data frame
`pointnames`	The names of the points, in order from head to tail
`point_to`	The name of the column to put the point names in
`sep`	The separator between the point name and variable name, as a regular expression (default = `"\\."`)

Value

The long data set

Examples

```
df <- data.frame(
  frame = 1:3,
  head.x = c(0, 0, 0), head.y = c(0, 1, 2),
  tail.x = c(5, 5, 5), tail.y = c(0, 1, 2)
)
pivot_kinematics_longer(df, c("head", "tail"))
```

`skip_na` *Skip NAs when running a function on a vector*

Description

`skip_na()` is a helper function related to `na.omit()`. It runs a function `f` on a vector that may contain NAs or NaNs, skipping all the NAs, and returns the results as a vector of the same length as `x` with the NAs in the same places.

Usage

```
skip_na(x, f, min.len = 1, ...)
```

Arguments

`x`	A vector that may have NAs
`f`	The function to run on the vector
`min.len`	Minimum number of non-NA values required to run `f` (default 1)
`...`	Other parameters to supply to the function

Value

A vector with the same length as `x` with NAs in the same places

Examples

```
x <- c(1,2,3,NA,4,5,6,NA,7,8)
skip_na(x, cumsum)
# should return a vector the same length as x with NAs in position 4 and 8
```

`smooth_point` *Applies a smoothing spline to a data series, potentially with gaps*

Description

Builds a smoothing spline for y(x), where `y` may contain gaps (NAs). Ignores the NAs. The uses the spline to interpolate values at the same `x` coordinates, potentially filling in the gaps.

Usage

```
smooth_point(x, y, spar, goodout = NULL)
```

Arguments

`x`	x coordinate
`y`	y coordinate, potentially with NAs
`spar`	Smoothing parameter (see `smooth.spline()`)
`goodout`	Logical vector of where in the x coordinate to interpolate

Value

The smoothed values

Examples

```
# smooth a noisy sine wave with two missing points
x <- 1:20
y <- sin(2 * pi * x / 10) + rnorm(20, sd = 0.1)
y[c(9, 10)] <- NA
smooth_point(x, y, spar = 0.5)
```

`smooth_points_df` *Smooths locations of points over time*

Description

Smooths columns specified in `cols` for each individual point over time, using a smoothing spline. Optionally fills in gaps of less than `fillgaps` frames.

Usage

```
smooth_points_df(
  .data,
  cols,
  spar,
  .out = NULL,
  .frame = frame,
  .point = point,
  fillgaps = 0
)
```

Arguments

`.data`	Data frame containing the midlines.
`cols`	Columns containing the components to be smoothed. Often these will be the x and y coordinates of the midline.
`spar`	Smoothing parameter passed to `smooth.spline()`. Values range from 0 (no smoothing) to 1 (heavy smoothing).

`.out`	Names of the output columns. Should either be a list with the same number of elements as cols, or a glue specification as in `across` for the `.names` parameter. The default (NULL) means that the output columns will have the same name as the original column with an 's' appended at the end.
`.frame`, `.point`	Columns identifying frames and points (defaults are `frame` and `point`)
`fillgaps`	Longest gap to interpolate over. default is 0, which means not to fill gaps

Value

A data frame containing the smoothed points

Examples

```
library(dplyr)
# smooth x and y coordinates of the lamprey midline over time
lampreydata |>
  smooth_points_df(c(mxmm, mymm), spar = 0.6)
```

`xmucosusdata` *Prickleback tracking data*

Description

Tracking data for a a swimming rock prickleback, *Xiphister mucosus*. The data was tracked using Sleap (https://sleap.ai/) and comes out in the following format. `frame_idx` is the frame number, and each point along the body is identified with the point name and `.x`, `.y`, the coordinate, and `.score`, which is a measure of the estimated accuracy of the point. All of the points together are also given a score (`instance.score`).

Usage

```
xmucosusdata
```

Format

A data frame with 711 rows and 27 columns. Columns follow the pattern `<keypoint>.x`, `<keypoint>.y`, and `<keypoint>.score` for each tracked keypoint, plus `frame_idx`, `instance.score`, and `track`. Keypoints are: `Snout`, `BP1–BP6`, and `Tail`.

`zebrafish_shape` *Zebrafish body shape*

Description

Width and height profile of a zebrafish body as a function of fractional arc length. Used for estimating volumes and centers of mass.

Usage

```
zebrafish_shape
```

Format

A data frame with 20 rows and 3 columns:

s Arc length as a fraction of body length (0 = head, 1 = tail)

width Lateral body width as a fraction of body length

height Dorso-ventral body height as a fraction of body length

---

`zfishdata` *Zebrafish keypoint tracking data*

Description

Keypoint tracking data for zebrafish (*Danio rerio*) swimming at several speeds. Keypoints are tracked using DeepLabCut and include body landmarks and fin tips.

Usage

```
zfishdata
```

Format

A data frame with 1056 rows and 46 columns. Most columns follow the pattern `<keypoint>.x`, `<keypoint>.y`, and `<keypoint>.score` for each tracked keypoint. Additional columns:

- `fn` Source file name
- `frame_idx` Frame index within the trial
- `id` Fish identifier
- `speed` Swimming speed in cm/s
- `datetime` Date and time of the trial
- `instance.score` Overall instance detection score
- `track` Track identifier

`zfish_goodframes` *Zebrafish good frame ranges*

Description

Start and end frame indices for the usable portions of each zebrafish swimming trial in `zfishdata`. Some trials contain multiple blocks of good frames.

Usage

```
zfish_goodframes
```

Format

A data frame with 8 rows and 4 columns:

File File name of the corresponding trial in `zfishdata`

Start First usable frame index

End Last usable frame index

Block Block number within the trial

Index