

Package: aniprocess (via r-universe)

October 17, 2026

Type Package

Title An R package for signal processing and filtering of movement data

Version 0.5.0.9003

Description An R package for signal processing and filtering of movement data.

License MIT + file LICENSE

Depends R ($\geq 4.1.0$)

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

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

Encoding UTF-8

Language en-GB

Imports anicore ($\geq 0.8.0.9003$), cli, data.table ($\geq 1.18.0$), dplyr ($\geq 1.1.0$), lifecycle, methods, rlang, signal, tidyselect

Suggests covr, stinepack, 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 15:24:22 UTC

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

RemoteRef HEAD

RemoteSha ccc9ad52a9a863f8a632b7c6c8034dce78ba3ff7

Contents

filter_across	2
filter_ccma	4
filter_deadband	6
filter_gaussian	9
filter_highpass	10
filter_highpass_fft	12
filter_kalman	15
filter_kalman_irregular	17
filter_lowpass	19
filter_lowpass_fft	21
filter_one_euro	23
filter_rollmean	25
filter_rollmedian	27
filter_sgolay	28
filter_triangular	30
filter_with	31
find_peaks	33
find_troughs	36
mask_na_across	39
mask_na_confidence	41
mask_na_excursion	42
mask_na_hampel	44
mask_na_range	46
mask_na_roi	47
mask_na_segment_length	48
mask_na_speed	52
mask_na_with	53
replace_na_across	54
replace_na_linear	55
replace_na_locf	56
replace_na_spline	57
replace_na_stine	58
replace_na_value	60
replace_na_with	61

Index	63
--------------	-----------

<code>filter_across</code>	<i>Apply a filter across an aniframe's spatial columns</i>
----------------------------	--

Description

The aniframe-level entry point to the `filter_*`() family. Applies a named filter to the frame's declared position columns, within its existing grouping.

Usage

```
filter_across(
  data,
  method = c("gaussian", "rollmean", "rollmedian", "triangular", "sgolay", "lowpass",
    "highpass", "lowpass_fft", "highpass_fft", "kalman", "kalman_irregular", "one_euro",
    "ccma", "deadband"),
  variables = NULL,
  ...,
  on_deltas = FALSE
)
```

Arguments

data	An aniframe.
method	Filter to apply. One of "gaussian", "rollmean", "rollmedian", "triangular", "sgolay", "lowpass", "highpass", "lowpass_fft", "highpass_fft", "kalman", "kalman_irregular", "one_euro", "ccma" or "deadband".
variables	Columns to filter, as a tidyselect expression. Defaults to the declared position columns.
...	Arguments passed to the underlying filter.
on_deltas	If TRUE, difference each column, filter the differences, and re-integrate from the original starting value. For trackball data, where the raw measurements are per-frame displacements and the coordinates were integrated from them, smoothing belongs on the displacements rather than on the integrated positions. A NA among the filtered differences counts as no movement when accumulating, and is restored as NA at its own position, so one missing step does not blank the rest of the series.

Details

This is the aniframe tier of the interface:

- `filter_gaussian()` and friends filter one vector — use them inside `dplyr::mutate()`.
- `filter_with()` does the same but picks the method by name.
- `filter_across()` applies a method to a whole aniframe.

Beyond looping over columns, it fills in what the frame already knows: `sampling_rate` comes from metadata for the methods that need it, and `"kalman_irregular"` takes its `times` from the index column (`anicore::get_index()`). Either can still be passed explicitly to override.

`"ccma"` and `"deadband"` are multivariate — each output coordinate depends on all of them — so they are applied jointly rather than column by column, and need a Cartesian frame. `on_deltas` does not apply to them.

Value

An aniframe of the same shape, with the selected columns filtered.

See Also

`filter_with()` for the vector-level generic.

Examples

```
tracking_data <- anicore::example_anipoint() |>
  anicore::set_metadata(sampling_rate = 30)

# sampling_rate is taken from the aniframe's metadata
filter_across(tracking_data, "lowpass", cutoff_freq = 5)

# restrict to some of the spatial columns
filter_across(tracking_data, "gaussian", variables = c(x, y), sigma = 2)
```

filter_ccma

Apply Curvature-Corrected Moving Average (CCMA)

Description**[Experimental]**

Smooths a trajectory while undoing the inward "corner-cutting" bias that a plain moving average introduces on curved paths.

Usage

```
filter_ccma(
  data,
  window_width_ma = 11,
  window_width_cc = 7,
  kernel = c("hanning", "uniform"),
  boundary = c("padding"),
  cc_mode = TRUE,
  na_action = c("linear", "spline", "stine", "locf", "value", "error"),
  keep_na = TRUE,
  ...
)
```

Arguments

data	A data frame of 2 or 3 numeric coordinate columns. The curvature math is Cartesian-specific (cross products, Euclidean norms, circumradius), so coordinates must be Cartesian. For a whole aniframe use <code>filter_across()</code> , which selects the declared position columns and rejects non-Cartesian frames.
window_width_ma	Integer width of the moving-average kernel (must be odd; even values are rounded up). Larger = more smoothing. Default 11.

<code>window_width_cc</code>	Integer width of the curvature-correction kernel (must be odd). Larger = smoother correction but uses curvature info from further away. Default 7.
<code>kernel</code>	Kernel shape for both stages. One of "hanning" (default; raised cosine) or "uniform" (boxcar).
<code>boundary</code>	Edge-handling strategy. Currently only "padding" (repeat the first and last point so output length equals input length).
<code>cc_mode</code>	If FALSE, returns just the moving-average result without curvature correction. Useful for comparison.
<code>na_action</code>	Method used to fill NA values <i>before</i> filtering, so the filter sees a complete series. One of "linear" (default), "spline", "stine", "locf", "value", or "error" to abort when NAs are present. Filling is internal: whether the filled values reach the output is controlled by <code>keep_na</code> .
<code>keep_na</code>	Logical. If TRUE (default), positions that were NA in the input are NA in the output — gaps stay gaps. If FALSE, the values used to fill those gaps are kept, so the output has fewer NAs than the input and genuinely-missing stretches come back as interpolated estimates.
<code>...</code>	Additional arguments passed to <code>replace_na_with()</code> (e.g. <code>value</code> , <code>min_gap</code> , <code>max_gap</code>).

Details

A plain moving average pulls each point toward the chord between its neighbours, which lies *inside* the curve — so smoothed circles shrink inward. CCMA (Steinecker & Wuensche, 2023) estimates how much shrinkage the moving average caused at each point — from the local curvature and the kernel — and pushes the result back outward by exactly that amount.

The algorithm has two stages:

1. **Moving average** of the spatial coordinates with a kernel of width `window_width_ma`.
2. **Curvature correction**: at each output position, sum a kernel of width `window_width_cc` of curvature-derived shifts and apply them outward in the curve's plane.

Because curvature is intrinsically multi-dimensional, this filter operates on all spatial coordinates jointly (unlike the per-column filters dispatched through `filter_across()`). It is most useful for smoothing curved 2D or 3D trajectories where a plain moving average visibly cuts corners; for general-purpose time-series smoothing reach for `filter_gaussian()` or `filter_sgolay()`.

Value

A data frame with the same names and shape as `data`, its coordinates smoothed.

Input shape

This is a **column-level** function: it takes a data frame of coordinate columns and returns one of the same shape. The aniframe tier is `filter_across()`, which applies it within

the frame's existing grouping so each individual / track / keypoint is smoothed as its own trajectory.

```
filter_across(af, "ccma") # a whole aniframe
data |> mutate(filter_ccma(pick(all_of(c("x", "y"))))) # the columns
```

`dplyr::pick()` supplies the columns and the result is spliced back over them.

CCMA is multivariate — each output coordinate depends on all of them — so it cannot be used with `dplyr::across()`, which passes one column at a time.

References

Steinecker, T. & Wuensche, H.-J. (2023). A Simple and Model-Free Path Filtering Algorithm for Smoothing and Accuracy. *2023 IEEE Intelligent Vehicles Symposium (IV)*, 1-7. doi:10.1109/IV55152.2023.10186704.

Reference Python implementation: <https://github.com/UniBwTAS/ccma>

Examples

```
# The column tier: a data frame of coordinates
coords <- data.frame(x = sin(seq(0, 6, length.out = 40)), y = seq(0, 6, length.out = 40))
head(filter_ccma(coords, window_width_ma = 11, window_width_cc = 7))

# The aniframe tier
af <- anicore::example_anipoint(n_obs = 60, n_individuals = 1, n_keypoints = 1)
filter_across(af, "ccma", window_width_ma = 11, window_width_cc = 7)
```

filter_deadband	<i>Apply a deadband filter</i>
-----------------	--------------------------------

Description

[Experimental]

Holds a still subject still. The output stays where it is until the tracked point has moved more than `threshold` away from it, so the small wandering of tracking noise on a stationary point is removed entirely rather than merely made smaller.

Usage

```
filter_deadband(data, threshold, mode = c("hold", "drag"), max_gap = Inf)
```

Arguments

data	A data frame of numeric coordinate columns — typically supplied by <code>dplyr::pick()</code> inside <code>dplyr::mutate()</code> . To filter a whole aniframe, use <code>filter_across()</code> .
threshold	A single non-negative number, in the units of the coordinates: how far the point has to move from the anchor before the output follows. 0 returns data unchanged.
mode	How the output follows once the threshold is crossed: "hold" (default) jumps to the point, "drag" moves just far enough to bring the point back to the edge of the dead zone. See Details.
max_gap	A whole number of rows, or <code>Inf</code> (default): the longest run of incomplete rows across which the anchor is kept. After a longer gap, the next complete row starts afresh.

Details

Tracking noise on a point that is not moving adds a little to its path every frame, so distance travelled is overestimated — the more so the longer the subject is still and the higher the sampling rate — and speed never reaches zero. A smoother scales that noise down but does not remove it. A deadband does: movement smaller than **threshold** is ignored.

The output keeps an *anchor*, starting at the first complete row. For each row, the distance from the anchor to the row's point is compared with **threshold**. While it is at most **threshold**, the output is the anchor. Once it is more, the anchor moves, in one of two ways:

- "hold" (default): the anchor jumps to the point. This is the "direct" *Minimal Distance Moved* of Noldus EthoVision XT, and the behaviour OptiTrack Motive describes for its rigid-body *Deadband Filter*, so results can be compared with theirs.
- "drag": the anchor is pulled towards the point until it is exactly **threshold** away, as if on a lead of that length. This is the play (backlash) operator of hysteresis theory, in its vector form (Krejčí, 1991). The output never jumps and trails the point by at most **threshold**, but sustained movement is shortened by **threshold** at each start and stop.

With several coordinate columns the distance is Euclidean, so the dead zone is a circle or sphere: a move registers by its length, whatever its direction. The axes should therefore share a unit, and be Cartesian.

Choosing a threshold:

threshold is in the units of the coordinates. It should exceed the noise on a still point — a few times its standard deviation, measured on a stretch where the subject is known to be still — and stay below the smallest movement that matters. Smoothing first, for instance with `filter_lowpass()`, lowers the noise and so lets the threshold be smaller.

What it does to derivatives:

A deadband is for distance travelled, immobility and bout detection, not for kinematics. In "hold" mode the speed is zero while the anchor holds and spikes when it jumps; in "drag" mode it is zero, then continuous. Neither is the subject's real speed.

The filter is recursive and causal: each output depends on the rows before it, so the result depends on where the series starts, and running it backwards in time gives a different answer.

Missing values:

A row with any coordinate missing is not a point: it is left as it is, and does not move the anchor. The next complete row is compared with the anchor as it stood before the gap, so a subject that is in the same place after a dropout stays exactly where it was. When a run of such rows is longer than `max_gap`, the anchor is reset instead: the next complete row is taken as it is, as at the start of the series.

Value

A data frame with the same names and shape as `data`.

Input shape

This is a **column-level** function: it takes a data frame of coordinate columns and returns one of the same shape. The aniframe tier is `filter_across()`, which applies it within the frame's existing grouping so each individual / track / keypoint is filtered as its own trajectory.

```
filter_across(af, "deadband", threshold = 2) # a whole aniframe
data |> mutate(filter_deadband(pick(all_of(c("x", "y"))), threshold = 2)) # the columns
```

The distance depends on all coordinates jointly, so this cannot be used with `dplyr::across()`, which passes one column at a time. For a single signal, pass a one-column frame.

References

- OptiTrack. Properties Pane: Rigid Body — Deadband Filter. *Motive documentation*. <https://docs.optitrack.com/motive-ui-panes/properties-pane/properties-pane-rigid-body>
- Noldus Information Technology. Smooth the tracks: Minimal Distance Moved methods. *EthoVision XT 19 reference manual*. <https://noldus.com/shared/resources/book/noldus-product-documentation/chapter/ethovision-xt/page/ethovision-xt-19-smooth-the-tracks-min>
- Hen, I., Sakov, A., Kafkafi, N., Golani, I., & Benjamini, Y. (2004). The dynamics of spatial behavior: how can robust smoothing techniques help? *Journal of Neuroscience Methods*, 133, 161–172. doi:10.1016/j.jneumeth.2003.10.013
- Krasnosel'skiĭ, M. A., & Pokrovskiĭ, A. V. (1989). *Systems with Hysteresis*. Springer. doi:10.1007/9783642613029
- Krejčí, P. (1991). Vector hysteresis models. *European Journal of Applied Mathematics*, 2, 281–292. doi:10.1017/S0956792500000541

See Also

`filter_lowpass()` and `filter_one_euro()`, which reduce noise rather than remove it, and work well before a deadband.

Examples

```
# A point that sits still with a little noise for 100 frames, then moves
# 10 units in 10 frames
set.seed(1)
coords <- data.frame(
  x = c(rep(0, 100), 1:10) + rnorm(110, sd = 0.05),
  y = rnorm(110, sd = 0.05)
)

# The still stretch comes out exactly still
held <- filter_deadband(coords, threshold = 0.2)
held[95:105, ]

# So the noise no longer adds to the path length
path_length <- function(d) sum(sqrt(diff(d$x)^2 + diff(d$y)^2))
path_length(coords)
path_length(held)

# "drag" follows without jumping, at most `threshold` behind
filter_deadband(coords, threshold = 0.2, mode = "drag")[95:105, ]

# The aniframe tier
af <- anicore::example_anipoint(n_obs = 60, n_individuals = 1, n_keypoints = 1)
filter_across(af, "deadband", threshold = 0.5)
```

filter_gaussian

Apply Gaussian Kernel Smoother

Description

Convolve a numeric vector with a discrete Gaussian kernel.

Usage

```
filter_gaussian(x, sigma = 1, window_width = NULL, keep_na = TRUE)
```

Arguments

x	Numeric vector to filter.
sigma	Standard deviation of the Gaussian kernel, in samples (frames). Must be positive.
window_width	Integer kernel width in samples. Must be positive and is forced to be odd. Defaults to $2 * \text{ceiling}(3 * \text{sigma}) + 1$, which truncates the kernel at ± 3 .
keep_na	Logical. If TRUE (default), positions that were NA in the input are NA in the output — gaps stay gaps. If FALSE , the values used to fill those gaps are kept, so the output has fewer NAs than the input and genuinely-missing stretches come back as interpolated estimates.

Details

The kernel is symmetric and centered: `weights[k] = dnorm(k, sd = sigma)` for `k` in `-half:half` (where `half = (window_width - 1) / 2`), renormalised to sum to 1. The output is the kernel-weighted moving average of `x`.

At each position, weights for kernel taps that would fall outside `x` or that align with NA values are excluded, and the remaining weights are renormalised. This means edges and isolated NAs are handled gracefully without contaminating the result. A position whose entire window is NA returns NA.

Larger `sigma` gives heavier smoothing. For movement data, typical values range from 0.5 (very light smoothing) to 5 (heavy smoothing).

Value

Filtered numeric vector, same length as `x`.

Examples

```
x <- c(1, 2, 3, 100, 5, 6, 7)
filter_gaussian(x, sigma = 1)
```

`filter_highpass`

Apply Butterworth Highpass Filter to Signal

Description

This function applies a highpass Butterworth filter to a signal using forward-backward filtering (`filtfilt`) to achieve zero phase distortion. The Butterworth filter is maximally flat in the passband, making it ideal for many signal processing applications.

Usage

```
filter_highpass(
  x,
  cutoff_freq,
  sampling_rate,
  order = 4,
  na_action = c("linear", "spline", "stine", "locf", "value", "error"),
  keep_na = TRUE,
  ...
)
```

Arguments

x	Numeric vector containing the signal to be filtered
cutoff_freq	Cutoff frequency in Hz. Frequencies above this value are passed, while frequencies below are attenuated. Should be between 0 and <code>sampling_rate/2</code> .
sampling_rate	Sampling rate of the signal in Hz. Must be at least twice the highest frequency component in the signal (Nyquist criterion).
order	Filter order (default = 4). Controls the steepness of frequency rolloff: • Higher orders give sharper cutoffs but may introduce more ringing • Lower orders give smoother transitions but less steep rolloff • Common values in practice are 2-8 • Values above 8 are rarely used due to numerical instability
na_action	Method used to fill NA values <i>before</i> filtering, so the filter sees a complete series. One of "linear" (default), "spline", "stine", "locf", "value", or "error" to abort when NAs are present. Filling is internal: whether the filled values reach the output is controlled by <code>keep_na</code> .
keep_na	Logical. If TRUE (default), positions that were NA in the input are NA in the output — gaps stay gaps. If FALSE, the values used to fill those gaps are kept, so the output has fewer NAs than the input and genuinely-missing stretches come back as interpolated estimates.
...	Additional arguments passed to <code>replace_na_with()</code> . Common options include: • <code>value</code>: Numeric value for replacement when <code>na_action = "value"</code> • <code>min_gap</code>: Minimum gap size to interpolate/fill • <code>max_gap</code>: Maximum gap size to interpolate/fill

Details

The Butterworth filter response falls off at $-6 \times \text{order}$ dB/octave. The cutoff frequency corresponds to the -3dB point of the filter's magnitude response.

Common Applications:

- Removing baseline drift: Use low cutoff (0.1-1 Hz)
- EMG analysis: Use moderate cutoff (10-20 Hz)
- Motion artifact removal: Use application-specific cutoff

Parameter Selection Guidelines:

- `cutoff_freq`: Choose based on the lowest frequency you want to preserve
- `order`: Same guidelines as `lowpass_filter`

Common values by field:

- ECG processing: `order=2`, `cutoff=0.5` Hz
- EEG analysis: `order=4`, `cutoff=1` Hz
- Mechanical vibrations: `order=2`, cutoff application-specific

Missing Value Handling: The function uses `replace_na_with()` internally for handling missing values. See `?replace_na_with` for detailed information about each method and its parameters. NAs can optionally be restored to their original positions after filtering using `keep_na = TRUE`.

Value

Numeric vector containing the filtered signal

References

Butterworth, S. (1930). On the Theory of Filter Amplifiers. *Wireless Engineer*, 7, 536-541.

See Also

[replace_na_with\(\)](#) for details on NA handling methods [filter_lowpass](#) for low-pass filtering

Examples

```
# Generate example signal with drift
t <- seq(0, 1, by = 0.001)
drift <- 0.5 * t # Linear drift
signal <- sin(2*pi*10*t) # 10 Hz signal
x <- signal + drift

# Add some NAs
x[sample(length(x), 10)] <- NA

# Basic filtering with linear interpolation for NAs
filtered <- filter_highpass(x, cutoff_freq = 2, sampling_rate = 1000)

# Using spline interpolation with max gap constraint
filtered <- filter_highpass(x, cutoff_freq = 2, sampling_rate = 1000,
                           na_action = "spline", max_gap = 3)

# Replace NAs with zeros before filtering
filtered <- filter_highpass(x, cutoff_freq = 2, sampling_rate = 1000,
                           na_action = "value", value = 0)

# Filter but keep NAs in their original positions
filtered <- filter_highpass(x, cutoff_freq = 2, sampling_rate = 1000,
                           na_action = "linear", keep_na = TRUE)
```

Description

This function implements a highpass filter using the Fast Fourier Transform (FFT). It provides a sharp frequency cutoff but may introduce ringing artifacts (Gibbs phenomenon).

Usage

```
filter_highpass_fft(
  x,
  cutoff_freq,
  sampling_rate,
  na_action = c("linear", "spline", "stine", "locf", "value", "error"),
  keep_na = TRUE,
  ...
)
```

Arguments

x	Numeric vector containing the signal to be filtered
cutoff_freq	Cutoff frequency in Hz. Frequencies above this value are passed, while frequencies below are attenuated. Should be between 0 and <code>sampling_rate/2</code> .
sampling_rate	Sampling rate of the signal in Hz. Must be at least twice the highest frequency component in the signal (Nyquist criterion).
na_action	Method used to fill NA values <i>before</i> filtering, so the filter sees a complete series. One of "linear" (default), "spline", "stine", "locf", "value", or "error" to abort when NAs are present. Filling is internal: whether the filled values reach the output is controlled by <code>keep_na</code> .
keep_na	Logical. If TRUE (default), positions that were NA in the input are NA in the output — gaps stay gaps. If FALSE , the values used to fill those gaps are kept, so the output has fewer NAs than the input and genuinely-missing stretches come back as interpolated estimates.
...	Additional arguments passed to <code>replace_na_with()</code> . Common options include: • <code>value</code>: Numeric value for replacement when <code>na_action = "value"</code> • <code>min_gap</code>: Minimum gap size to interpolate/fill • <code>max_gap</code>: Maximum gap size to interpolate/fill

Details

FFT-based filtering applies a hard cutoff in the frequency domain. This can be advantageous for:

- Precise frequency selection
- Batch processing of long signals
- Cases where sharp frequency cutoffs are desired

Common Applications:

- Removing baseline drift: Use low cutoff (0.1-1 Hz)
- EMG analysis: Use moderate cutoff (10-20 Hz)
- Motion artifact removal: Use application-specific cutoff

Limitations:

- May introduce ringing artifacts
- Assumes periodic signal (can cause edge effects)
- Less suitable for real-time processing

Missing Value Handling: The function uses `replace_na_with()` internally for handling missing values. See `?replace_na_with` for detailed information about each method and its parameters. NAs can optionally be restored to their original positions after filtering using `keep_na = TRUE`.

Value

Numeric vector containing the filtered signal

See Also

[replace_na_with\(\)](#) for details on NA handling methods [filter_lowpass_fft](#) for FFT-based low-pass filtering [filter_highpass](#) for Butterworth-based filtering

Examples

```
# Generate example signal with drift
t <- seq(0, 1, by = 0.001)
drift <- 0.5 * t # Linear drift
signal <- sin(2*pi*10*t) # 10 Hz signal
x <- signal + drift

# Add some NAs
x[sample(length(x), 10)] <- NA

# Basic filtering with linear interpolation for NAs
filtered <- filter_highpass_fft(x, cutoff_freq = 2, sampling_rate = 1000)

# Using spline interpolation with max gap constraint
filtered <- filter_highpass_fft(x, cutoff_freq = 2, sampling_rate = 1000,
                               na_action = "spline", max_gap = 3)

# Replace NAs with zeros before filtering
filtered <- filter_highpass_fft(x, cutoff_freq = 2, sampling_rate = 1000,
                               na_action = "value", value = 0)

# Filter but keep NAs in their original positions
filtered <- filter_highpass_fft(x, cutoff_freq = 2, sampling_rate = 1000,
                               na_action = "linear", keep_na = TRUE)

# Compare with Butterworth filter
```

```
butter_filtered <- filter_highpass(x, 2, 1000)
```

filter_kalman

Kalman Filter for Regular Time Series

Description

Implements a Kalman filter for regularly sampled time series data with automatic parameter selection based on sampling rate. The filter handles missing values (NA) and provides noise reduction while preserving real signal changes.

Usage

```
filter_kalman(
  x,
  sampling_rate,
  base_Q = NULL,
  R = NULL,
  initial_state = NULL,
  initial_P = NULL,
  keep_na = FALSE
)
```

Arguments

x	Numeric vector of measurements to be filtered.
sampling_rate	Numeric value specifying the sampling rate in Hz (frames per second).
base_Q	Optional. Process variance. If NULL, automatically calculated based on sampling_rate. Represents expected rate of change in the true state.
R	Optional. Measurement variance. If NULL, defaults to 0.1. Represents the noise level in your measurements.
initial_state	Optional. Initial state estimate. If NULL, uses first non-NA measurement.
initial_P	Optional. Initial state uncertainty. If NULL, calculated based on sampling_rate.
keep_na	Logical. If TRUE, positions that were NA in x are NA in the output. Defaults to FALSE, unlike the rest of the filter family: a Kalman filter's predict step is designed to carry the state estimate through missing observations, so inferring across gaps is the intended behaviour rather than an accident. Set TRUE when you want gaps left as gaps.

Details

The function implements a simple Kalman filter with a constant position model. When parameters are not explicitly provided, they are automatically configured based on the sampling rate:

- `base_Q` defaults to `var(x) / sampling_rate` (so the per-step process noise $Q = \text{base_Q} / \text{sampling_rate}$ shrinks at higher sampling rates, where consecutive samples are closer together).
- `R` defaults to `min(mean(diff(x)^2) / 2, var(x) / 4)` if there are enough observations, else 0.1.
- `initial_P` defaults to `var(x)` if there are enough observations, else 1.

Missing values (NA) are handled by relying on the prediction step without measurement updates.

Value

A numeric vector of the same length as `x` containing the filtered values.

Note

Parameter selection guidelines:

- Increase `R` or decrease `base_Q` for smoother output
- Decrease `R` or increase `base_Q` for more responsive output
- For high-frequency data (>100 Hz), consider reducing `base_Q`
- If you know your sensor's noise characteristics, set `R` to the square of the standard deviation

See Also

`filter_kalman_irregular` for handling irregularly sampled data

Examples

```
# Basic usage with 60 Hz data
x <- c(1, 1.1, NA, 0.9, 1.2, NA, 0.8, 1.1)
filtered <- filter_kalman(x, sampling_rate = 60)

# Custom parameters for more aggressive filtering
filtered_custom <- filter_kalman(x,
                                sampling_rate = 60,
                                base_Q = 0.001,
                                R = 0.2)
```

filter_kalman_irregular

Kalman Filter for Irregular Time Series with Optional Resampling

Description

Implements a Kalman filter for irregularly sampled time series data with optional resampling to regular intervals. Handles variable sampling rates, missing values, and automatically adjusts process variance based on time intervals.

Usage

```
filter_kalman_irregular(
  x,
  times,
  base_Q = NULL,
  R = NULL,
  initial_state = NULL,
  initial_P = NULL,
  resample = FALSE,
  resample_freq = NULL,
  keep_na = FALSE
)
```

Arguments

x	Numeric vector of measurements to be filtered.
times	Numeric vector of timestamps corresponding to x.
base_Q	Optional. Base process variance per second. If NULL, automatically calculated.
R	Optional. Measurement variance. If NULL, defaults to 0.1.
initial_state	Optional. Initial state estimate. If NULL, uses first non-NA measurement.
initial_P	Optional. Initial state uncertainty. If NULL, calculated from median sampling rate.
resample	Logical. Whether to return regularly resampled data (default: FALSE).
resample_freq	Numeric. Desired sampling frequency in Hz for resampling (required if resample=TRUE).
keep_na	Logical. If TRUE, positions that were NA in x are NA in the values reported on the original timestamps. Defaults to FALSE, for the reason given in filter_kalman() . When resample = TRUE this applies to original_values only — the resampled values sit on a different time grid, where input positions have no counterpart, so they are always interpolated from the complete filtered series.

Details

The function implements an adaptive Kalman filter that accounts for irregular sampling intervals. Process variance is scaled by the time difference between measurements, allowing proper uncertainty handling for variable sampling rates.

Key features:

- Handles irregular sampling intervals
- Scales process variance with time gaps
- Optional resampling to regular intervals
- Automatic parameter selection based on median sampling rate
- Missing value (NA) handling

When resampling, the function uses linear interpolation and warns if the requested sampling frequency exceeds twice the median original sampling rate (Nyquist frequency).

Value

If `resample=FALSE`: A numeric vector of filtered values corresponding to original timestamps
If `resample=TRUE`: A list containing:

- `time`: Vector of regular timestamps
- `values`: Vector of filtered values at regular timestamps
- `original_time`: Original irregular timestamps
- `original_values`: Filtered values at original timestamps

Note

Resampling considerations:

- Avoid resampling above twice the median original sampling rate
- Consider the physical meaning of your data when choosing `resample_freq`
- Be cautious of creating artifacts through high-frequency resampling

Parameter selection guidelines:

- `base_Q` controls the expected rate of change per second
- `R` should reflect your measurement noise level
- For slow-changing signals, reduce `base_Q`
- For noisy `x`, increase `R`

See Also

`filter_kalman` for regularly sampled data

Examples

```
# Example with irregular sampling
x <- c(1, 1.1, NA, 0.9, 1.2, NA, 0.8, 1.1)
times <- c(0, 0.1, 0.3, 0.35, 0.5, 0.8, 0.81, 1.0)

# Basic filtering with irregular samples
filtered <- filter_kalman_irregular(x, times)

# Filtering with resampling to 50 Hz
filtered_resampled <- filter_kalman_irregular(x, times,
                                              resample = TRUE,
                                              resample_freq = 50)

# Plot results
plot(times, x, type="p", col="blue")
lines(filtered_resampled$time, filtered_resampled$values, col="red")
```

filter_lowpass	<i>Apply Butterworth Lowpass Filter to Signal</i>
----------------	---

Description

This function applies a lowpass Butterworth filter to a signal using forward-backward filtering (filtfilt) to achieve zero phase distortion. The Butterworth filter is maximally flat in the passband, making it ideal for many signal processing applications.

Usage

```
filter_lowpass(
  x,
  cutoff_freq,
  sampling_rate,
  order = 4,
  na_action = c("linear", "spline", "stine", "locf", "value", "error"),
  keep_na = TRUE,
  ...
)
```

Arguments

x	Numeric vector containing the signal to be filtered
cutoff_freq	Cutoff frequency in Hz. Frequencies below this value are passed, while frequencies above are attenuated. Should be between 0 and <code>sampling_rate/2</code> .
sampling_rate	Sampling rate of the signal in Hz. Must be at least twice the highest frequency component in the signal (Nyquist criterion).
order	Filter order (default = 4). Controls the steepness of frequency rolloff:

	• Higher orders give sharper cutoffs but may introduce more ringing • Lower orders give smoother transitions but less steep rolloff • Common values in practice are 2-8 • Values above 8 are rarely used due to numerical instability
<code>na_action</code>	Method used to fill NA values <i>before</i> filtering, so the filter sees a complete series. One of "linear" (default), "spline", "stine", "locf", "value", or "error" to abort when NAs are present. Filling is internal: whether the filled values reach the output is controlled by <code>keep_na</code> .
<code>keep_na</code>	Logical. If TRUE (default), positions that were NA in the input are NA in the output — gaps stay gaps. If FALSE, the values used to fill those gaps are kept, so the output has fewer NAs than the input and genuinely-missing stretches come back as interpolated estimates.
<code>...</code>	Additional arguments passed to <code>replace_na_with()</code> . Common options include: • <code>value</code>: Numeric value for replacement when <code>na_action = "value"</code> • <code>min_gap</code>: Minimum gap size to interpolate/fill • <code>max_gap</code>: Maximum gap size to interpolate/fill

Details

The Butterworth filter response falls off at $-6 \times \text{order}$ dB/octave. The cutoff frequency corresponds to the -3dB point of the filter's magnitude response.

Parameter Selection Guidelines:

- `cutoff_freq`: Choose based on the frequency content you want to preserve
- `sampling_rate`: Should match your data collection rate
- `order`:
 - `order=2`: Gentle rolloff, minimal ringing (~12 dB/octave)
 - `order=4`: Standard choice, good balance (~24 dB/octave)
 - `order=6`: Steeper rolloff, some ringing (~36 dB/octave)
 - `order=8`: Very steep, may have significant ringing (~48 dB/octave) Note: For very low cutoff frequencies (<0.001 of Nyquist), order is automatically reduced to 2 to maintain stability.

Common values by field:

- Biomechanics: `order=2` or 4
- EEG/MEG: `order=4` or 6
- Audio processing: `order=2` to 8
- Mechanical vibrations: `order=2` to 4

Missing Value Handling: The function uses `replace_na_with()` internally for handling missing values. See `?replace_na_with` for detailed information about each method and its parameters. NAs can optionally be restored to their original positions after filtering using `keep_na = TRUE`.

Value

Numeric vector containing the filtered signal

References

Butterworth, S. (1930). On the Theory of Filter Amplifiers. Wireless Engineer, 7, 536-541.

See Also

[replace_na_with\(\)](#) for details on NA handling methods [filter_highpass](#) for high-pass filtering

Examples

```
# Generate example signal: 2 Hz fundamental + 50 Hz noise
t <- seq(0, 1, by = 0.001)
x <- sin(2*pi*2*t) + 0.5*sin(2*pi*50*t)

# Add some NAs
x[sample(length(x), 10)] <- NA

# Basic filtering with linear interpolation for NAs
filtered <- filter_lowpass(x, cutoff_freq = 5, sampling_rate = 1000)

# Using spline interpolation with max gap constraint
filtered <- filter_lowpass(x, cutoff_freq = 5, sampling_rate = 1000,
                          na_action = "spline", max_gap = 3)

# Replace NAs with zeros before filtering
filtered <- filter_lowpass(x, cutoff_freq = 5, sampling_rate = 1000,
                          na_action = "value", value = 0)

# Filter but keep NAs in their original positions
filtered <- filter_lowpass(x, cutoff_freq = 5, sampling_rate = 1000,
                          na_action = "linear", keep_na = TRUE)
```

filter_lowpass_fft	<i>Apply FFT-based Lowpass Filter to Signal</i>
--------------------	---

Description

This function implements a lowpass filter using the Fast Fourier Transform (FFT). It provides a sharp frequency cutoff but may introduce ringing artifacts (Gibbs phenomenon).

Usage

```
filter_lowpass_fft(
  x,
  cutoff_freq,
  sampling_rate,
  na_action = c("linear", "spline", "stine", "locf", "value", "error"),
  keep_na = TRUE,
  ...
)
```

Arguments

<code>x</code>	Numeric vector containing the signal to be filtered
<code>cutoff_freq</code>	Cutoff frequency in Hz. Frequencies below this value are passed, while frequencies above are attenuated. Should be between 0 and <code>sampling_rate/2</code> .
<code>sampling_rate</code>	Sampling rate of the signal in Hz. Must be at least twice the highest frequency component in the signal (Nyquist criterion).
<code>na_action</code>	Method used to fill NA values <i>before</i> filtering, so the filter sees a complete series. One of "linear" (default), "spline", "stine", "locf", "value", or "error" to abort when NAs are present. Filling is internal: whether the filled values reach the output is controlled by <code>keep_na</code> .
<code>keep_na</code>	Logical. If TRUE (default), positions that were NA in the input are NA in the output — gaps stay gaps. If FALSE, the values used to fill those gaps are kept, so the output has fewer NAs than the input and genuinely-missing stretches come back as interpolated estimates.
<code>...</code>	Additional arguments passed to <code>replace_na_with()</code> . Common options include: • <code>value</code>: Numeric value for replacement when <code>na_action = "value"</code> • <code>min_gap</code>: Minimum gap size to interpolate/fill • <code>max_gap</code>: Maximum gap size to interpolate/fill

Details

FFT-based filtering applies a hard cutoff in the frequency domain. This can be advantageous for:

- Precise frequency selection
- Batch processing of long signals
- Cases where sharp frequency cutoffs are desired

Limitations:

- May introduce ringing artifacts
- Assumes periodic signal (can cause edge effects)
- Less suitable for real-time processing

Missing Value Handling: The function uses `replace_na_with()` internally for handling missing values. See `?replace_na_with` for detailed information about each method and its parameters. NAs can optionally be restored to their original positions after filtering using `keep_na = TRUE`.

Value

Numeric vector containing the filtered signal

See Also

[replace_na_with\(\)](#) for details on NA handling methods [filter_highpass_fft](#) for FFT-based high-pass filtering [filter_lowpass](#) for Butterworth-based filtering

Examples

```
# Generate example signal with mixed frequencies
t <- seq(0, 1, by = 0.001)
x <- sin(2*pi*2*t) + sin(2*pi*50*t)

# Add some NAs
x[sample(length(x), 10)] <- NA

# Basic filtering with linear interpolation for NAs
filtered <- filter_lowpass_fft(x, cutoff_freq = 5, sampling_rate = 1000)

# Using spline interpolation with max gap constraint
filtered <- filter_lowpass_fft(x, cutoff_freq = 5, sampling_rate = 1000,
                              na_action = "spline", max_gap = 3)

# Replace NAs with zeros before filtering
filtered <- filter_lowpass_fft(x, cutoff_freq = 5, sampling_rate = 1000,
                              na_action = "value", value = 0)

# Filter but keep NAs in their original positions
filtered <- filter_lowpass_fft(x, cutoff_freq = 5, sampling_rate = 1000,
                              na_action = "linear", keep_na = TRUE)

# Compare with Butterworth filter
butter_filtered <- filter_lowpass(x, 5, 1000)
```

filter_one_euro

Apply the One Euro filter

Description

An adaptive low-pass filter that trades jitter against lag according to how fast the signal is moving: heavy smoothing while the subject is nearly still, light smoothing while it moves quickly.

Usage

```
filter_one_euro(
  x,
  sampling_rate,
  min_cutoff = 1,
  beta = 0,
  d_cutoff = 1,
  na_action = c("linear", "spline", "stine", "locf", "value", "error"),
  keep_na = TRUE,
  ...
)
```

Arguments

<code>x</code>	Numeric vector to filter.
<code>sampling_rate</code>	Sampling rate of the signal in Hz.
<code>min_cutoff</code>	Minimum cutoff frequency in Hz, the cutoff used when the signal is not moving. Lower means smoother but lagier. Default 1.
<code>beta</code>	Speed coefficient. 0 gives a plain low-pass filter at <code>min_cutoff</code> ; larger values raise the cutoff more sharply as the signal speeds up, cutting lag. Default 0.
<code>d_cutoff</code>	Cutoff frequency in Hz for the derivative estimate, which keeps noise in the derivative from driving the adaptation. Default 1.
<code>na_action</code>	Method used to fill NA values <i>before</i> filtering, so the filter sees a complete series. One of "linear" (default), "spline", "stine", "locf", "value", or "error" to abort when NAs are present. Filling is internal: whether the filled values reach the output is controlled by <code>keep_na</code> .
<code>keep_na</code>	Logical. If TRUE (default), positions that were NA in the input are NA in the output — gaps stay gaps. If FALSE, the values used to fill those gaps are kept, so the output has fewer NAs than the input and genuinely-missing stretches come back as interpolated estimates.
<code>...</code>	Additional arguments passed to <code>replace_na_with()</code> .

Details

A fixed low-pass filter forces one compromise on the whole recording. Set the cutoff low and slow passages come out clean but fast ones lag behind; set it high and fast passages track well but slow ones jitter. The One Euro filter (Casiez, Roussel & Vogel, 2012) removes the compromise by making the cutoff a function of the estimated speed:

$$f_c = f_{c_{min}} + \beta|\dot{x}|$$

where $\dot{x}$ is itself low-pass filtered, at `d_cutoff`, so that noise in the derivative does not drive the cutoff around.

Tuning, following the authors' advice, is two-stage:

1. Set `beta = 0` and lower `min_cutoff` until jitter is acceptable while the subject is still.

2. Raise `beta` until lag is acceptable while it moves quickly.

`min_cutoff` therefore governs the slow-movement end and `beta` the fast-movement end, and the two can be tuned almost independently.

The filter is recursive, so it needs a complete series: NAs are filled by `na_action` before filtering. With `keep_na = TRUE` (the default) they are restored afterwards, so the gaps are not silently invented.

Value

Filtered numeric vector, same length as `x`.

References

Casiez, G., Roussel, N., & Vogel, D. (2012). 1 € Filter: A Simple Speed-based Low-pass Filter for Noisy Input in Interactive Systems. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '12)*, 2527–2530. doi:10.1145/2207676.2208639

See Also

`filter_lowpass()` for a fixed-cutoff Butterworth filter.

Examples

```
t <- seq(0, 2, by = 1 / 60)
x <- ifelse(t < 1, 0, 10) + rnorm(length(t), 0, 0.1)

# beta = 0 is a plain low-pass: smooth, but slow to follow the step
filter_one_euro(x, sampling_rate = 60, min_cutoff = 0.5)

# raising beta keeps the still passages smooth but tracks the step
filter_one_euro(x, sampling_rate = 60, min_cutoff = 0.5, beta = 0.5)
```

<code>filter_rollmean</code>	<i>Apply Rolling Mean Filter</i>
------------------------------	----------------------------------

Description

Applies a rolling mean filter to a numeric vector using `data.table::frollmean()`.

Usage

```
filter_rollmean(
  x,
  window_width = 5,
  min_obs = 1,
  align = c("center", "right", "left"),
  keep_na = TRUE
)
```

Arguments

<code>x</code>	Numeric vector to filter.
<code>window_width</code>	Integer specifying window size for the rolling calculation.
<code>min_obs</code>	Minimum number of non-NA values required in the window. Positions with fewer non-NA values return NA. Defaults to 1.
<code>align</code>	Window alignment. One of "center" (default), "right", or "left". A centred window is symmetric about the point, so the filtered signal keeps its timing; "right" looks only backwards, which is what you want when the next sample does not exist yet, at the cost of lagging the signal by $(\text{window_width} - 1) / 2$ samples.
<code>keep_na</code>	Logical. If TRUE (default), positions that were NA in the input are NA in the output — gaps stay gaps. If FALSE, the values used to fill those gaps are kept, so the output has fewer NAs than the input and genuinely-missing stretches come back as interpolated estimates.

Details

A centred window is symmetric about the point it replaces, so the filtered signal keeps its timing. A right-aligned one looks only backwards, which is what you want when the next sample does not exist yet — real-time tracking, closed-loop experiments — and lags the signal by $(\text{window_width} - 1) / 2$ samples in exchange. Smoothing before `animetric::add_kinematics()` with a lagging filter moves every event later by that much.

The trade runs the other way at the edges: a centred window has no data beyond the ends of the series, so the first and last $(\text{window_width} - 1) / 2$ values are NA, where a right-aligned window fills them from a partial window.

`keep_na` and `min_obs` control different things and are usually worth setting together. `keep_na` governs positions that were NA in the *input*; `min_obs` governs positions that were observed but whose *window* is too sparse to trust. Neither substitutes for the other: with `min_obs = 1`, positions next to a gap still produce values drawn from very few observations, and no `min_obs` setting blanks the input gaps without also blanking their neighbours and the series edges.

For `align = "right"` or `"left"`, partial windows at the edges of the series are computed (so position 1 with a width-5 right-aligned window returns the value at position 1, not NA). For `align = "center"`, edges are not partial: the first and last $(\text{window_width} - 1) \%\% 2$ positions return NA. This is a limitation of the underlying `data.table::frollmean()` implementation.

Value

Filtered numeric vector, same length as `x`.

Examples

```
x <- c(1, 2, 100, 4, 5, 6, 7)
filter_rollmean(x, window_width = 3)

# Centring the window changes which samples are averaged, and leaves NA at
```

```
# both edges rather than one
filter_rollmean(x, window_width = 3, align = "center")
```

filter_rollmedian	<i>Apply Rolling Median Filter</i>
-------------------	------------------------------------

Description

Applies a rolling median filter to a numeric vector using `data.table::frollmedian()`.

Usage

```
filter_rollmedian(
  x,
  window_width = 5,
  min_obs = 1,
  align = c("center", "right", "left"),
  keep_na = TRUE
)
```

Arguments

<code>x</code>	Numeric vector to filter.
<code>window_width</code>	Integer specifying window size for the rolling calculation.
<code>min_obs</code>	Minimum number of non-NA values required in the window. Positions with fewer non-NA values return NA. Defaults to 1.
<code>align</code>	Window alignment. One of "center" (default), "right", or "left". A centred window is symmetric about the point, so the filtered signal keeps its timing; "right" looks only backwards, which is what you want when the next sample does not exist yet, at the cost of lagging the signal by $(\text{window_width} - 1) / 2$ samples.
<code>keep_na</code>	Logical. If TRUE (default), positions that were NA in the input are NA in the output — gaps stay gaps. If FALSE, the values used to fill those gaps are kept, so the output has fewer NAs than the input and genuinely-missing stretches come back as interpolated estimates.

Details

Edge handling matches `filter_rollmean()`: partial windows at the edges for `align = "right"/"left"`; NA at the edges for `align = "center"`.

Value

Filtered numeric vector, same length as `x`.

Examples

```
x <- c(1, 2, 100, 4, 5, 6, 7)
filter_rollmedian(x, window_width = 3)

# The median discards the spike; a rolling mean smears it across three
# neighbouring samples instead
filter_rollmean(x, window_width = 3)
```

filter_sgolay	<i>Apply Savitzky-Golay Filter to Movement Data</i>
---------------	---

Description

This function applies a Savitzky-Golay filter to smooth movement data while preserving higher moments (peaks, valleys) better than moving average filters. The implementation uses zero-phase filtering to prevent temporal shifts in the data.

Usage

```
filter_sgolay(
  x,
  sampling_rate,
  window_width = ceiling(sampling_rate/10) * 2 + 1,
  order = 3,
  na_action = "linear",
  keep_na = TRUE,
  ...
)
```

Arguments

x	Numeric vector containing the movement data to be filtered
sampling_rate	Sampling rate of the data in Hz. Must match your data collection rate (e.g., 60 for 60 FPS motion capture).
window_width	Window size in samples (must be odd). Controls the amount of smoothing. Larger windows give more smoothing but may over-attenuate genuine movement features. Default is automatically calculated as <code>sampling_rate/10</code> (rounded up to nearest odd number).
order	Polynomial order (default = 3). Controls how well the filter preserves higher-order moments in the data: • order=2: Preserves position, velocity (good for smooth movements) • order=3: Also preserves acceleration (good for most movement data) • order=4: Also preserves jerk (good for quick movements) • order=5: Maximum preservation (may retain too much noise)

<code>na_action</code>	Method used to fill NA values <i>before</i> filtering, so the filter sees a complete series. One of "linear" (default), "spline", "stine", "locf", "value", or "error" to abort when NAs are present. Filling is internal: whether the filled values reach the output is controlled by <code>keep_na</code> .
<code>keep_na</code>	Logical. If TRUE (default), positions that were NA in the input are NA in the output — gaps stay gaps. If FALSE, the values used to fill those gaps are kept, so the output has fewer NAs than the input and genuinely-missing stretches come back as interpolated estimates.
<code>...</code>	Additional arguments passed to <code>replace_na_with()</code>

Details

The Savitzky-Golay filter fits successive polynomials to sliding windows of the data. This approach preserves higher moments of the data better than simple moving averages or Butterworth filters, making it particularly suitable for movement data where preserving features like peaks and valleys is important.

Edges are handled by `signal::sgolayfilt()` using extrapolation from the nearest interior polynomial fit, which is the standard Savitzky-Golay edge convention.

Parameter Selection Guidelines:

- `window_width`:
 - For 60 FPS: 5-15 frames (83-250ms) for quick movements, 15-31 for slow movements
 - For 120 FPS: 7-21 frames (58-175ms) for quick movements, 21-51 for slow movements
 - For 500 FPS: 25-75 frames (50-150ms) for quick movements, 75-151 for slow movements The default `window_width = sampling_rate/10` works well for typical human movement.
- `order`:
 - `order=2`: Smooth movements, position analysis
 - `order=3`: Most movement analysis (default)
 - `order=4`: Quick movements, sports analysis
 - `order=5`: Very quick movements, impact analysis Note: order must be less than `window_width`

Common values by application:

- Gait analysis (60 FPS): `window_width=15`, `order=3`
- Sports biomechanics (120 FPS): `window_width=21`, `order=4`
- Impact analysis (500 FPS): `window_width=51`, `order=4`
- Posture analysis (60 FPS): `window_width=31`, `order=2`

Value

Numeric vector containing the filtered movement data

References

Savitzky, A., & Golay, M.J.E. (1964). Smoothing and Differentiation of Data by Simplified Least Squares Procedures. *Analytical Chemistry*, 36(8), 1627-1639. doi:10.1021/ac60214a047.

See Also

[filter_lowpass](#) for frequency-based filtering [replace_na_with\(\)](#) for details on NA handling methods

Examples

```
# Generate example movement data: smooth motion + noise
t <- seq(0, 5, by = 1/60) # 60 FPS data
x <- sin(2*pi*0.5*t) + rnorm(length(t), 0, 0.1)

# Basic filtering with default parameters (60 FPS)
filtered <- filter_sgolay(x, sampling_rate = 60)

# Adjusting parameters for quick movements
filtered_quick <- filter_sgolay(x, sampling_rate = 60,
                                window_width = 11, order = 4)

# High-speed camera data (500 FPS) with larger window
filtered_high <- filter_sgolay(x, sampling_rate = 500,
                               window_width = 51, order = 3)
```

filter_triangular	<i>Apply Triangular Filter</i>
-------------------	--------------------------------

Description

Applies a triangular smoothing filter — a rolling mean of width `window_width` applied twice. The composition of two boxcars is a triangular kernel, so the effective kernel width is $2 * \text{window_width} - 1$ with peak weight at the centre.

Usage

```
filter_triangular(
  x,
  window_width = 5,
  min_obs = 1,
  align = c("center", "right", "left"),
  keep_na = TRUE
)
```

Arguments

<code>x</code>	Numeric vector to filter.
<code>window_width</code>	Integer width of each rolling-mean pass. The effective triangular kernel has width $2 * \text{window_width} - 1$.
<code>min_obs</code>	Minimum number of non-NA values required per window per pass. Defaults to 1.
<code>align</code>	Window alignment, passed to <code>filter_rollmean()</code> . One of "center" (default), "right", or "left".
<code>keep_na</code>	Logical. If TRUE (default), positions that were NA in the input are NA in the output — gaps stay gaps. If FALSE, the values used to fill those gaps are kept, so the output has fewer NAs than the input and genuinely-missing stretches come back as interpolated estimates.

Details

For `align = "center"`, the underlying `filter_rollmean()` returns NA at the first and last $(\text{window_width} - 1) \%\% 2$ positions of each pass, so the output has roughly `window_width - 1` NA values at each edge.

`keep_na = TRUE` restores the positions that were NA in the input, but note that this filter fills gaps more aggressively than a single `filter_rollmean()` pass: because the second pass smooths the output of the first, values interpolated across a gap propagate outward. `keep_na` blanks the original gap positions; it does not undo that spread into neighbouring positions. Raise `min_obs` to suppress it.

Triangular smoothing is sometimes useful as a lightweight alternative to a Gaussian kernel when the kernel shape is less critical than the simplicity of the implementation.

Value

Filtered numeric vector, same length as `x`.

Examples

```
x <- c(1, 2, 3, 100, 5, 6, 7, 8, 9)
filter_triangular(x, window_width = 3)
```

<code>filter_with</code>	<i>Apply a filter by name</i>
--------------------------	-------------------------------

Description

Generic entry point to the `filter_*`() family: pick the method with an argument rather than by choosing a function. Useful when the method is itself a parameter — comparing filters, or driving one from a config.

Usage

```
filter_with(
  x,
  method = c("gaussian", "rollmean", "rollmedian", "triangular", "sgolay", "lowpass",
             "highpass", "lowpass_fft", "highpass_fft", "kalman", "kalman_irregular", "one_euro",
             "ccma", "deadband"),
  ...
)
```

Arguments

<code>x</code>	A numeric vector, or a data frame of numeric coordinate columns.
<code>method</code>	Filter to apply. One of "gaussian", "rollmean", "rollmedian", "triangular", "sgolay", "lowpass", "highpass", "lowpass_fft", "highpass_fft", "kalman", "kalman_irregular", "one_euro", "ccma" or "deadband".
<code>...</code>	Arguments passed to the underlying filter.

Details

Returns the same shape it is given. A numeric vector gives a numeric vector; a data frame of coordinate columns gives a data frame.

Most methods are univariate and are applied one column at a time when given a frame. "ccma" and "deadband" are multivariate — each output coordinate depends on all of them — so they require a frame and cannot be used with `dplyr::across()`.

Method-specific arguments are passed through `...`, so a required one still has to be supplied: `sampling_rate` for "sgolay", "lowpass", "highpass", the `_fft` variants and "kalman"; `times` for "kalman_irregular"; `cutoff_freq` for the frequency filters; `threshold` for "deadband".

Value

The same shape as `x`, filtered.

See Also

`filter_across()` to apply a filter across an aniframe's spatial columns.

Examples

```
x <- c(1, 2, 3, 100, 5, 6, 7, 8, 9)

filter_with(x, "gaussian", sigma = 1)
filter_with(x, "rollmean", window_width = 3)

# A frame is filtered column by column
filter_with(data.frame(x = x, y = rev(x)), "rollmean", window_width = 3)
```

`find_peaks`*Find Peaks in Time Series Data*

Description

Identifies peaks (local maxima) in a numeric time series, with options to filter peaks based on height and prominence. The function handles missing values (NA) appropriately and is compatible with dplyr's mutate. Includes flexible handling of plateaus and adjustable window size for peak detection.

Usage

```
find_peaks(  
  x,  
  min_height = -Inf,  
  min_prominence = 0,  
  plateau_handling = c("strict", "middle", "first", "last", "all"),  
  window_width = 3  
)
```

Arguments

<code>x</code>	Numeric vector containing the time series data
<code>min_height</code>	Minimum height threshold for peaks (default: -Inf)
<code>min_prominence</code>	Minimum prominence threshold for peaks (default: 0)
<code>plateau_handling</code>	String specifying how to handle plateaus. One of: • "strict" (default): No points in plateau are peaks • "middle": Middle point(s) of plateau are peaks • "first": First point of plateau is peak • "last": Last point of plateau is peak • "all": All points in plateau are peaks
<code>window_width</code>	Integer specifying the size of the window to use for peak detection (default: 3). Must be odd and ≥ 3 . Larger values detect peaks over wider ranges.

Details

The function uses a sliding window algorithm for peak detection (window size specified by `window_width` parameter), combined with a region-based prominence calculation method similar to that described in Palshikar (2009).

Value

A logical vector of the same length as the input where:

- TRUE indicates a confirmed peak
- FALSE indicates a non-peak
- NA indicates peak status could not be determined due to missing data

Peak Detection

A point is considered a peak if it is the highest point within its window (default window_width of 3 compares each point with its immediate neighbors). The first and last $(\text{window_width}-1)/2$ points in the series cannot be peaks and are marked as NA. Larger window sizes will identify peaks that dominate over a wider range, typically resulting in fewer peaks being detected.

Prominence

Prominence measures how much a peak stands out relative to its surrounding values. It is calculated as the height of the peak minus the height of the highest minimum between this peak and any higher peaks (or the end of the series if no higher peaks exist).

Plateau Handling

Plateaus (sequences of identical values) are handled according to the plateau_handling parameter:

- strict: No points in a plateau are considered peaks (traditional behavior)
- middle: For plateaus of odd length, the middle point is marked as a peak. For plateaus of even length, the two middle points are marked as peaks.
- first: The first point of each plateau is marked as a peak
- last: The last point of each plateau is marked as a peak
- all: Every point in the plateau is marked as a peak

Note that in all cases, the plateau must still qualify as a peak relative to its surrounding window (i.e., higher than all other points in the window).

Missing Values (NA) Handling

The function uses the following rules for handling NAs:

- If a point is NA, it cannot be a peak (returns NA)
- If any point in the window is NA, peak status cannot be determined (returns NA)
- For prominence calculations, stretches of NAs are handled appropriately
- A minimum of window_width points is required; shorter series return all NAs

Note

- The function is optimized for use with dplyr's mutate
- For noisy data, consider using a larger window_width or smoothing the series before peak detection
- Adjust min_height and min_prominence to filter out unwanted peaks
- Choose plateau_handling based on your specific needs
- Larger window_width values result in more stringent peak detection

References

Palshikar, G. (2009). Simple Algorithms for Peak Detection in Time-Series. Proc. 1st Int. Conf. Advanced Data Analysis, Business Analytics and Intelligence.

See Also

- [find_troughs](#) for finding local minima

Examples

```
# Basic usage with default window size (3)
x <- c(1, 3, 2, 6, 4, 5, 2)
find_peaks(x)

# With larger window size
find_peaks(x, window_width = 5) # More stringent peak detection

# With minimum height
find_peaks(x, min_height = 4, window_width = 3)

# With plateau handling
x <- c(1, 3, 3, 3, 2, 4, 4, 1)
find_peaks(x, plateau_handling = "middle", window_width = 3) # Middle of plateaus
find_peaks(x, plateau_handling = "all", window_width = 5)    # All plateau points

# With missing values
x <- c(1, 3, NA, 6, 4, NA, 2)
find_peaks(x)

# Usage with dplyr
library(dplyr)
data.frame(
  time = 1:10,
  value = c(1, 3, 7, 4, 2, 6, 5, 8, 4, 2)
) |>
  mutate(peaks = find_peaks(value, window_width = 3))
```

find_troughs

Find Troughs in Time Series Data

Description

Identifies troughs (local minima) in a numeric time series, with options to filter troughs based on height and prominence. The function handles missing values (NA) appropriately and is compatible with dplyr's mutate. Includes flexible handling of plateaus and adjustable window size for trough detection.

Usage

```
find_troughs(
  x,
  max_height = Inf,
  min_prominence = 0,
  plateau_handling = c("strict", "middle", "first", "last", "all"),
  window_width = 3
)
```

Arguments

x	Numeric vector containing the time series data
max_height	Maximum height threshold for troughs (default: Inf)
min_prominence	Minimum prominence threshold for troughs (default: 0)
plateau_handling	String specifying how to handle plateaus. One of: • "strict" (default): No points in plateau are troughs • "middle": Middle point(s) of plateau are troughs • "first": First point of plateau is trough • "last": Last point of plateau is trough • "all": All points in plateau are troughs
window_width	Integer specifying the size of the window to use for trough detection (default: 3). Must be odd and ≥ 3 . Larger values detect troughs over wider ranges.

Details

The function uses a sliding window algorithm for trough detection (window size specified by window_width parameter), combined with a region-based prominence calculation method similar to that described in Palshikar (2009).

Value

A logical vector of the same length as the input where:

- TRUE indicates a confirmed trough
- FALSE indicates a non-trough
- NA indicates trough status could not be determined due to missing data

Trough Detection

A point is considered a trough if it is the lowest point within its window (default window_width of 3 compares each point with its immediate neighbors). The first and last (window_width-1)/2 points in the series cannot be troughs and are marked as NA. Larger window sizes will identify troughs that dominate over a wider range, typically resulting in fewer troughs being detected.

Prominence

Prominence measures how much a trough stands out relative to its surrounding values. It is calculated as the height of the lowest maximum between this trough and any lower troughs (or the end of the series if no lower troughs exist) minus the height of the trough.

Plateau Handling

Plateaus (sequences of identical values) are handled according to the plateau_handling parameter:

- strict: No points in a plateau are considered troughs (traditional behavior)
- middle: For plateaus of odd length, the middle point is marked as a trough. For plateaus of even length, the two middle points are marked as troughs.
- first: The first point of each plateau is marked as a trough
- last: The last point of each plateau is marked as a trough
- all: Every point in the plateau is marked as a trough

Note that in all cases, the plateau must still qualify as a trough relative to its surrounding window (i.e., lower than all other points in the window).

Missing Values (NA) Handling

The function uses the following rules for handling NAs:

- If a point is NA, it cannot be a trough (returns NA)
- If any point in the window is NA, trough status cannot be determined (returns NA)
- For prominence calculations, stretches of NAs are handled appropriately
- A minimum of window_width points is required; shorter series return all NAs

Note

- The function is optimized for use with dplyr's mutate
- For noisy data, consider using a larger window_width or smoothing the series before trough detection
- Adjust max_height and min_prominence to filter out unwanted troughs
- Choose plateau_handling based on your specific needs
- Larger window_width values result in more stringent trough detection

References

Palshikar, G. (2009). Simple Algorithms for Peak Detection in Time-Series. Proc. 1st Int. Conf. Advanced Data Analysis, Business Analytics and Intelligence.

See Also

- [find_peaks](#) for finding local maxima

Examples

```
# Basic usage with default window size (3)
x <- c(5, 3, 4, 1, 4, 2, 5)
find_troughs(x)

# With larger window size
find_troughs(x, window_width = 5) # More stringent trough detection

# With maximum height
find_troughs(x, max_height = 3, window_width = 3)

# With plateau handling
x <- c(5, 2, 2, 2, 3, 1, 1, 4)
find_troughs(x, plateau_handling = "middle", window_width = 3) # Middle of plateaus
find_troughs(x, plateau_handling = "all", window_width = 5)    # All plateau points

# With missing values
x <- c(5, 3, NA, 1, 4, NA, 5)
find_troughs(x)

# Usage with dplyr
library(dplyr)
data.frame(
  time = 1:10,
  value = c(5, 3, 1, 4, 2, 1, 3, 0, 4, 5)
) |>
  mutate(troughs = find_troughs(value, window_width = 3))
```

mask_na_across	<i>Mask values to NA across an aniframe's spatial columns</i>
----------------	---

Description

The aniframe-level entry point to the `mask_na_*`() family. Applies a named criterion to the frame's declared position columns, within its existing grouping — so a criterion is never evaluated across a track boundary.

Usage

```
mask_na_across(
  data,
  method = c("range", "speed", "excursion", "hampel", "roi", "confidence"),
  variables = NULL,
  ...,
  on_deltas = FALSE
)
```

Arguments

<code>data</code>	An aniframe.
<code>method</code>	Criterion to apply. One of "range", "speed", "excursion", "hampel", "roi" or "confidence".
<code>variables</code>	Columns to mask, as a tidyselect expression. Defaults to the declared position columns.
<code>...</code>	Arguments passed to the underlying function. For "speed", <code>threshold</code> additionally accepts "pooled": "auto" estimates a threshold separately for each group, so every track is judged against its own noise; "pooled" estimates one threshold from all groups at once, which is steadier when tracks are short.
<code>on_deltas</code>	If TRUE, difference each column, mask the differences, and re-integrate from the original starting value — rejecting implausible single-window displacements rather than implausible positions. See <i>Masking on displacements</i> below. Only "range" accepts it. The others either already judge a sample against its neighbours, or are not about displacement at all, so differencing first would answer a different question than their name promises; they error rather than quietly compute it.

Details

Beyond looping over columns, it fills in what the frame already knows: "speed" takes its `time` from the index column (`anicore::get_index()`), and "confidence" takes its `confidence` from the column of that name. Either can be passed explicitly to override. For

"confidence", the warning about rows with no score is given once for the whole frame, not once per group; see `mask_na_confidence()` for missing.

Only "range" is univariate. The others decide per row using all the selected columns at once, and blank every one of them on a flagged row.

Where a `confidence` column is present, it is blanked on rows that this call masked — the cross-column effect that the vector-level functions cannot perform, since `confidence` is not a coordinate.

Value

An aniframe of the same shape, with failing values replaced by NA.

Masking on displacements

`on_deltas` matters more here than it does for smoothing. Where the coordinates are cumulative — trackball data, say, whose raw readings are per-window displacements that were integrated with `cumsum()` — masking a *position* blanks the sample you flagged but leaves the spurious jump baked into every subsequent position. Masking the *displacement* removes the bad step itself, and everything downstream shifts back into place.

A masked step counts as no movement, and is restored as NA at its own sample alone; the NA is not propagated forward. So the reading is "this step is not believable", not "the animal's position is unknown from here". The first sample of each group is never masked: it is the starting point, and there is no step into it to judge.

See Also

`mask_na_with()` for the vector-level generic.

Examples

```
tracking_data <- anicore::example_anipoint()

# time comes from the aniframe's metadata
mask_na_across(tracking_data, "speed", threshold = "auto")

mask_na_across(tracking_data, "range", min_value = -2, max_value = 2)

# trackball positions are integrated displacements: reject the step,
# not the position it left behind
trackball_data <- anicore::as_anipoint(data.frame(
  time = 1:8,
  x = cumsum(c(0, 1, 1, 1, 50, 1, 1, 1)),
  y = cumsum(c(0, 1, 1, 1, 1, 1, 1, 1))
))
mask_na_across(
  trackball_data,
  "range",
  min_value = -10,
  max_value = 10,
  on_deltas = TRUE
```

```
)
```

mask_na_confidence	Mask low-confidence values to NA
--------------------	----------------------------------

Description

This function replaces spatial coordinate values with **NA** if the confidence values are below a specified threshold. The **confidence** column is also masked.

Usage

```
mask_na_confidence(
  data,
  threshold = 0.6,
  confidence = NULL,
  missing = c("keep", "mask")
)
```

Arguments

data	A data frame of numeric coordinate columns — typically supplied by <code>dplyr::pick()</code> inside <code>dplyr::mutate()</code> . To mask a whole aniframe, use <code>mask_na_across()</code> .
threshold	A numeric value specifying the minimum confidence level to retain data. Must be a single value between 0 and 1. Default is 0.6.
confidence	Numeric vector of confidence values, one per row.
missing	What to do with rows whose confidence is NA . "keep" (the default) leaves them unmasked; "mask" treats a missing score as failing the threshold, so they are masked like any low-confidence row.

Details

A missing confidence means *not assessed*, not *assessed as poor*, so by default those rows are left unmasked. A human annotator has no natural number to enter for "I did not assess this", and tracker scores are not bounded at 1 — SLEAP can exceed it — so **NA** is the sensible thing to record rather than a sentinel value. Set **missing = "mask"** when you would rather drop what was never assessed.

When **missing** is left at its default, a warning reports how many rows were left unmasked without a score, since silently skipping them would hide that they were never checked. Rows whose coordinates are already all **NA** are not counted: masking could not change them. Supplying **missing**, either value, says you have decided, and silences the warning.

Value

data, with coordinates replaced by **NA** where confidence is below the threshold.

Input shape

Takes and returns a frame of coordinate columns, so it composes inside `dplyr::mutate()`:

```
data |> mutate(
  mask_na_confidence(pick(all_of(c("x", "y"))), confidence = confidence)
)
```

The decision uses all coordinates at once, so this cannot be used with `dplyr::across()`. `confidence` is not a coordinate and so is never modified here; `mask_na_across()` masks it as well.

Examples

```
coords <- data.frame(x = 1:5, y = 6:10)
mask_na_confidence(
  coords,
  threshold = 0.6,
  confidence = c(0.5, 0.7, 0.4, 0.8, 0.9)
)

# Row 3 was never scored: kept by default, masked on request
scores <- c(0.5, 0.7, NA, 0.8, 0.9)
mask_na_confidence(coords, confidence = scores, missing = "keep")
mask_na_confidence(coords, confidence = scores, missing = "mask")
```

<code>mask_na_excursion</code>	<i>Mask position excursions that return</i>
--------------------------------	---

Description

Flags multi-frame tracking errors as NA using the criterion from Todd, Kain & de Bivort (2017): a frame-to-frame jump of more than `outlier_sd` standard deviations starts an excursion, which is then rejected if (and only if) the trajectory eventually returns either close to the pre-excursion position or close to the overall median position.

Usage

```
mask_na_excursion(data, outlier_sd = 5, return_sd = 1, by_axis = TRUE)
```

Arguments

<code>data</code>	A data frame of numeric coordinate columns — typically supplied by <code>dplyr::pick()</code> inside <code>dplyr::mutate()</code> . To mask a whole aniframe, use <code>mask_na_across()</code> .
<code>outlier_sd</code>	Threshold (in standard deviations) for flagging frame-to-frame jumps and for the "return to pre-excursion position" acceptance check. Todd's default is 5.

<code>return_sd</code>	Threshold (in standard deviations) for the "return to overall median position" acceptance check. Todd's default is 1.
<code>by_axis</code>	Logical. If TRUE (the default), Todd's literal per-axis behaviour is used: each spatial column has its own , median, and excursion state machine, and a row is blanked if any axis flags it. If FALSE , a single state machine runs on the joint Euclidean displacement (consistent with <code>mask_na_speed()</code>).

Details

For each coordinate column, the algorithm:

1. Computes the standard deviation of the coordinate over the full series and the overall median `m`.
2. Walks the series. When a frame-to-frame change exceeds `outlier_sd * m`, an excursion starts: that frame is flagged and subsequent frames are flagged until the position is either within `outlier_sd * m` of its pre-excursion value, or within `return_sd * m` of the overall median. The first such frame is accepted (not flagged), and the state machine resets.

This distinguishes transient excursions (a tracking glitch where the position eventually comes back) from sustained shifts (the animal genuinely moved to a new region) — the latter never satisfy the return condition unless the new region happens to be near the median, in which case it is accepted via the second criterion.

Every coordinate column is set to NA at a flagged row. `confidence` is not a coordinate and so is never modified here; `mask_na_across()` blanks it too.

Value

`data`, with flagged rows blanked.

References

Todd, J. G., Kain, J. S., & de Bivort, B. L. (2017). Systematic exploration of unsupervised methods for mapping behavior. *Physical Biology*, 14(1), 015002. doi:10.1088/14783975/14/1/015002.

See Also

`mask_na_speed()` for single-frame outliers.

Examples

```
# A two-frame glitch: x jumps away at row 41 and returns at row 43
coords <- data.frame(
  x = c(rep(0:1, 20), 50, 50, rep(0:1, 19)),
  y = rep(0:1, 40)
)

# Default Todd thresholds, per-axis.
mask_na_excursion(coords)
```

```
# Joint Euclidean variant, looser thresholds.
mask_na_excursion(coords, outlier_sd = 4, by_axis = FALSE)
```

mask_na_hampel

Mask spikes against a rolling median

Description

[Experimental]

Masks spikes — a point that jumps away for a frame or two and comes back — by judging each point against its own neighbourhood, after Hampel. A point is set to NA when its distance from the rolling median of the window around it exceeds `k` robust deviations of that window. Because both the median and the spread are local, the threshold follows the movement: tight where the keypoint moves slowly, loose where it moves fast — which a single threshold on speed or on position cannot be.

Unlike the classic Hampel filter, the point is masked rather than replaced with the median, so it composes like the rest of the `mask_na_*`() family: mask, then fill with `replace_na_with()`.

Usage

```
mask_na_hampel(data, window_width = 5, k = 3, min_distance = 0)
```

Arguments

<code>data</code>	A data frame of numeric coordinate columns — typically supplied by <code>dplyr::pick()</code> inside <code>dplyr::mutate()</code> . To mask a whole aniframe, use <code>mask_na_across()</code> .
<code>window_width</code>	An odd whole number, at least 3 (default 5): the number of rows in the window, centred on the point being judged. Up to $(\text{window_width} - 1) / 2$ consecutive spike frames can be caught.
<code>k</code>	A single non-negative number (default 3): how many robust deviations a point may lie from the rolling median before it is masked.
<code>min_distance</code>	A single non-negative number (default 0), in the units of the coordinates: a floor on the threshold. A point that lies within <code>min_distance</code> of its rolling median is never masked, however small the spread of its window.

Details

Each row is judged against the window of `window_width` rows centred on it:

1. The window's median position is taken axis by axis, with `filter_rollmedian()`.
2. The row's deviation `d` is the Euclidean distance of its point from that median.
3. The window's robust spread `S` is 1.4826 times the median of the distances of all the window's points from the same median: the median absolute deviation (MAD) of the window, scaled.

4. The row is masked when $d > \max(k * S, \text{min_distance})$.

With one coordinate column this is the Hampel identifier of Pearson et al. (2016). With several, the distance is Euclidean, so a point is masked in all axes at once, and the axes should share a unit. The factor 1.4826 makes the MAD estimate the standard deviation of one-dimensional Gaussian noise; a Euclidean distance in two or three dimensions is not distributed that way, so there k is a multiple of the robust spread rather than a number of standard deviations, and more conservative.

`min_distance` is what keeps a keypoint that barely moves from losing its noise: there the MAD of a window can be tiny, or zero, and any deviation at all exceeds k of it.

The spread is that of the window's points about the window's own median, as in Pearson et al., so it includes the distance travelled within the window. On steady motion a spike therefore has to reach several times the per-frame step to be masked — about nine at the defaults — while on a slow stretch the same settings catch spikes far smaller than the fast stretch's steps. A spread taken instead over each point's distance to its *own* rolling median is scaled by the noise rather than the motion, and so is more sensitive on fast stretches; but it is exactly zero wherever every axis moves monotonically through the window, so it flags noise there unless `min_distance` is set.

Every row is judged against the original data in a single pass, so a spike does not shift the verdict on its neighbours: the median and the MAD both ignore up to $(\text{window_width} - 1) / 2$ aberrant points in the window. A run of spikes longer than that dominates the window and is not caught — that is what `mask_na_excursion()` is for.

Missing values and edges:

A row with any coordinate missing is neither judged nor used to judge its neighbours: it is not a point, and is left as it is. A row is judged only when a majority of its window — at least $(\text{window_width} + 1) / 2$ rows — holds complete points, which is also the fewest that leave a spike in the minority. Positions beyond either end of the series count as missing, so the first and last rows are judged against the part of the window that exists, provided it is complete.

The window counts rows, not time, so rows are taken to be in temporal order; on irregularly sampled data it spans a varying duration.

Value

`data`, with every coordinate set to NA on rows judged to be spikes.

Input shape

Takes and returns a frame of coordinate columns, so it composes inside `dplyr::mutate()`:

```
data |> mutate(mask_na_hampel(pick(all_of(c("x", "y")))))
```

The deviation depends on all coordinates jointly, so this cannot be used with `dplyr::across()`. Every row of `data` is treated as one continuous track; called via `mask_na_across()`, or with `dplyr::pick()` inside a grouped `dplyr::mutate()`, a window never spans a track boundary. `confidence` is not a coordinate and so is never modified here; `mask_na_across()` blanks it on masked rows.

References

Pearson, R. K., Neuvo, Y., Astola, J., & Gabbouj, M. (2016). Generalized Hampel filters. *EURASIP Journal on Advances in Signal Processing*, 2016, 87. doi:10.1186/s13634016-03836.

See Also

`mask_na_speed()` and `mask_na_excursion()`, which judge steps and excursions rather than deviations from a neighbourhood; `filter_rollmedian()` to smooth with the median instead.

Examples

```
# One unit per frame, then twenty, with a 15-unit spike at row 4
coords <- data.frame(
  x = c(0, 1, 2, 3, 4, 5, 6, 26, 46, 66, 86, 106),
  y = c(0, 0, 0, 15, 0, 0, 0, 0, 0, 0, 0, 0)
)
mask_na_hampel(coords)

# A speed threshold low enough to catch the spike also takes the fast
# stretch, whose steps are larger than the spike
mask_na_speed(coords, threshold = 10, time = 1:12)

# A keypoint that barely moves: without a floor, its noise is flagged
still <- data.frame(x = c(5, 5, 5.01, 5, 5, 5, 4.99, 5))
mask_na_hampel(still)
mask_na_hampel(still, min_distance = 0.05)
```

mask_na_range	<i>Mask values outside a range to NA</i>
---------------	--

Description

Replaces values in a numeric vector that fall outside the specified range with NA. Values already NA in the input remain NA.

Usage

```
mask_na_range(x, min_value = -Inf, max_value = Inf)
```

Arguments

<code>x</code>	A numeric vector to mask
<code>min_value</code>	Minimum value (inclusive). Values below this become NA. Default is -Inf (no lower bound).
<code>max_value</code>	Maximum value (inclusive). Values above this become NA. Default is Inf (no upper bound).

Value

A numeric vector the same length as `x` with out-of-range values replaced by NA

Examples

```
mask_na_range(c(1, 5, 10, 15), min_value = 3, max_value = 12)
# Returns: c(NA, 5, 10, NA)

mask_na_range(c(1, NA, 10), min_value = 5)
# Returns: c(NA, NA, 10)
```

mask_na_roi	<i>Mask coordinates outside a region of interest (ROI)</i>
-------------	--

Description

Masks coordinates that fall outside a specified region of interest by setting them to NA. The ROI can be either rectangular/cuboid (defined by min/max coordinates) or circular/spherical (defined by center and radius). Handles 2D or 3D data according to whether a `z` column is present.

Usage

```
mask_na_roi(
  data,
  x_min = NULL,
  x_max = NULL,
  y_min = NULL,
  y_max = NULL,
  z_min = NULL,
  z_max = NULL,
  x_center = NULL,
  y_center = NULL,
  z_center = NULL,
  radius = NULL
)
```

Arguments

<code>data</code>	A data frame with numeric <code>x</code> and <code>y</code> columns (and optionally <code>z</code>) — typically supplied by <code>dplyr::pick()</code> inside <code>dplyr::mutate()</code> . To mask a whole aniframe, use <code>mask_na_across()</code> .
<code>x_min, x_max</code>	Bounds for x-coordinate (rectangular/cuboid ROI).
<code>y_min, y_max</code>	Bounds for y-coordinate (rectangular/cuboid ROI).
<code>z_min, z_max</code>	Bounds for z-coordinate (cuboid ROI, 3D only).

x_center, y_center, z_center
Center coordinates for circular/spherical ROI. For 3D data, provide all three; for 2D data, only x and y.

radius
Radius of circular (2D) or spherical (3D) ROI.

Value

data, with coordinates outside the ROI set to NA.

Examples

```
coords <- data.frame(
  x = rep(c(25, 50, 75), 3),
  y = rep(c(25, 50, 75), each = 3)
)

# Rectangular ROI
mask_na_roi(coords, x_min = 20, x_max = 60, y_min = 20, y_max = 60)

# Circular ROI
mask_na_roi(coords, x_center = 50, y_center = 50, radius = 30)

# 3D cuboid ROI
coords_3d <- data.frame(
  x = rep(c(25, 75), 4),
  y = rep(c(25, 75), each = 2, times = 2),
  z = rep(c(25, 75), each = 4)
)

mask_na_roi(
  coords_3d,
  x_min = 20, x_max = 60,
  y_min = 20, y_max = 60,
  z_min = 20, z_max = 60
)
```

mask_na_segment_length

Mask points whose segments are far off their usual length

Description

[Experimental]

Finds the points of a structure that sit in the wrong place, from the lengths of the segments joining them, and replaces their coordinates with NA. A segment meant to be rigid — a long bone, the pelvis — stretches or shrinks when one of its ends is wrong. Each segment is compared with its usual length, and the point to blame is masked, not the segment: a wrong ankle makes both shin and foot off while the thigh is fine, so the ankle goes, and the knee and the toe stay.

This catches errors that confidence, speed and smoothing miss: a point that sits wrong for many frames with a confident score, such as a match in multi-camera 3D that several cameras happen to agree on.

Unlike the other `mask_na_*`() functions, it judges each point by its neighbours, so it needs all of an individual's points at once. It takes the whole anipoint rather than one track at a time, and is not a method of `mask_na_across()`.

Usage

```
mask_na_segment_length(
  data,
  structure = NULL,
  tolerance = 0.3,
  min_difference = 0,
  segments = NULL
)
```

Arguments

data	An anipoint in 2D or 3D Cartesian coordinates, with a structure attached by <code>anicore::set_structure()</code> .
structure	Name of the structure to use. May be omitted when only one structure has segments.
tolerance	A single non-negative number (default 0.3): how far a segment's length may differ from its reference, as a fraction of the reference, before it is off.
min_difference	A single non-negative number (default 0): how far a segment's length may differ from its reference, in the frame's spatial units, before it is off. A segment must exceed both this and tolerance , so this keeps a short segment from being off over a difference that is large only relative to its length.
segments	Names of the segments to judge, as a character vector, or <code>NULL</code> (the default) for every segment of the structure. Choose those meant to be rigid: long bones and the pelvis, not the width of the shoulders or the neck. A point's segments are counted among these alone, so a point with one of them is an end point.

Value

data, with every axis of each masked point set to `NA`, and its **confidence** where present. Its rows, columns and metadata are unchanged.

Reference length

Each segment is compared with a reference length:

- its **length** in the structure (see `anicore::anistructure()`), when the structure gives one, in the frame's spatial units;

- otherwise, the median of its non-missing lengths over the track: within each group of `anicore::as_anisegment()`, that is per segment and per combination of the keys other than the structure's variable — per individual, and per session or trial when `data` has them. Every time point counts towards it, including those where the segment turns out to be off; the median holds while they are fewer than half.

A segment with no `length` in the structure has no usable reference in a track where its median is missing (its two ends are never present together) or zero (they coincide in at least half the frames). It is not judged in that track.

When a segment is off

The lengths are those of `anicore::as_anisegment()`: the distance between the segment's two points in each frame. A segment is off in a frame when its length `L` differs from its reference `R` by more than both thresholds:

```
abs(L - R) > tolerance * R & abs(L - R) > min_difference
```

A difference equal to a threshold is not off. With the default `min_difference = 0`, `tolerance` alone decides.

A segment whose length is missing in a frame, because either end is `NA` or has no row there, is not measured in that frame.

Which point is to blame

Blame is decided frame by frame — at each time point of each individual, and of each session or trial — from the segments measured in that frame. Only the selected `segments` count.

- A point with two or more measured segments is masked when all of them are off.
- A point with one measured segment — an end point such as a toe or a nose, or a point whose other segments are missing in that frame — is masked when that segment is off and the point at its other end has another measured segment that is not off. That point is in place, so the fault is at this end. It is also what keeps a wrong ankle from taking the toe with it: the toe's one segment is the foot, and the ankle's other segment, the shin, is off as well.
- When neither end of an off segment has another measured segment, nothing tells which end is wrong, and neither is masked.
- A point with no measured segment is left alone, as is any level of the structure's variable that the selected segments do not reach.

It is one pass over the lengths as measured: masking a point does not re-judge its neighbours. A correct point whose neighbours are all wrong has every segment off, so it is masked along with them.

What is masked

A masked point has every axis set to `NA`, and its `confidence` too where the frame has that column, as `mask_na_across()` does. A point that is already `NA`, in any axis, has no measured segment, so it is not judged, and its neighbours are judged by their other segments.

See Also

`anicore::as_anisegment()` for the lengths, and `anicore::set_structure()` to attach a structure. `mask_na_across()` for the criteria judged one track at a time.

Examples

```
# A leg standing straight for five frames
leg <- anicore::anipoint(
  individual = 1,
  keypoint = rep(c("hip", "knee", "ankle", "toe"), each = 5),
  time = rep(1:5, times = 4),
  x = 0,
  y = rep(c(4, 2, 0, -1), each = 5),
  variables_what = c("individual", "keypoint"),
  index = "time",
  variables_where = c("x", "y")
)
leg <- anicore::set_structure(
  leg,
  anicore::anistructure(
    segments = data.frame(
      segment = c("thigh", "shin", "foot"),
      from = c("hip", "knee", "ankle"),
      to = c("knee", "ankle", "toe")
    )
  )
)

# In frame 3 the ankle is off to the side: shin and foot both stretch
leg$x[leg$keypoint == "ankle" & leg$time == 3] <- 2

# Only the ankle is masked, not the knee or the toe
mask_na_segment_length(leg) |>
  dplyr::filter(time == 3)

# Judge thigh and shin alone: the ankle is now an end point, and goes
# because the knee's other segment, the thigh, is fine
mask_na_segment_length(leg, segments = c("thigh", "shin")) |>
  dplyr::filter(time == 3)

# Expected lengths recorded in the structure are used instead of the median
leg <- anicore::set_structure(
  leg,
  anicore::anistructure(
    segments = data.frame(
      segment = c("thigh", "shin", "foot"),
      from = c("hip", "knee", "ankle"),
      to = c("knee", "ankle", "toe"),
      length = c(2, 2, 1)
    )
  )
)
```

```
mask_na_segment_length(leg, min_difference = 0.5) |>
  dplyr::filter(time == 3)
```

mask_na_speed	<i>Mask values that move faster than a speed threshold</i>
---------------	--

Description

Masks single-frame outliers based on movement speed. Spatial coordinates and confidence values at flagged rows are replaced with NA.

Usage

```
mask_na_speed(data, threshold = "auto", time = NULL)
```

Arguments

data	A data frame of numeric coordinate columns — typically supplied by <code>dplyr::pick()</code> inside <code>dplyr::mutate()</code> . To mask a whole aniframe, use <code>mask_na_across()</code> .
threshold	A numeric value specifying the speed threshold, or "auto". • If numeric: Rows whose speed exceeds this value have their spatial and confidence values replaced with NA. • If "auto": Sets threshold at mean speed + 3 standard deviations.
time	Numeric vector of time values, one per row.

Details

For each row, two step speeds are computed: the backward step (from the previous row to this one) and the forward step (from this row to the next), each as the magnitude of the position change divided by the time step. The row's speed is the **minimum** of the two — so a row is only flagged when both the step in *and* the step out are fast. This isolates single-frame outliers (a position that jumps away and comes back) from legitimate state changes (a sustained move to a new region), which only have one fast step.

Endpoints have only one neighbor; their speed falls back to the available one-sided step. NAs in inputs do not contaminate adjacent rows: a missing coordinate at row *i* only affects row *i*'s speed estimate.

Every row of **data** is treated as one continuous track: a step is formed between each consecutive pair. Called via `mask_na_across()` or with `dplyr::pick()` inside a grouped `dplyr::mutate()`, that means one group, so a step is never formed across a track boundary.

When using **threshold** = "auto", the threshold is the mean speed plus three standard deviations of the rows given. Called through `mask_na_across()` that means one threshold per group; pass **threshold** = "pooled" there to estimate a single threshold from every group at once instead.

Value

data, with coordinates replaced by NA where speed exceeds the threshold.

Input shape

Takes and returns a frame of coordinate columns, so it composes inside `dplyr::mutate()`:

```
data |> mutate(mask_na_speed(pick(all_of(c("x", "y"))), time = time))
```

Speed depends on all coordinates jointly, so this cannot be used with `dplyr::across()`. `confidence` is not a coordinate and so is never modified here; `mask_na_across()` blanks it on masked rows.

Examples

```
coords <- data.frame(x = c(1, 2, 4, 7, 11), y = c(1, 1, 2, 3, 5))

mask_na_speed(coords, threshold = 3, time = 1:5)
mask_na_speed(coords, threshold = "auto", time = 1:5)
```

mask_na_with	<i>Mask values to NA by a named criterion</i>
--------------	---

Description

Generic entry point to the `mask_na_*`() family: pick the criterion with an argument rather than by choosing a function. Every method replaces values that fail its criterion with NA; none of them fill or smooth.

Usage

```
mask_na_with(
  x,
  method = c("range", "speed", "excursion", "hampel", "roi", "confidence"),
  ...
)
```

Arguments

x	A numeric vector, or a data frame of numeric coordinate columns.
method	Criterion to apply. One of "range", "speed", "excursion", "hampel", "roi" or "confidence".
...	Arguments passed to the underlying function.

Details

Returns the same shape it is given.

Only "range" is univariate, and it is applied one column at a time when given a frame. The rest decide per row using all coordinates at once, so they require a frame of coordinate columns and cannot be used with `dplyr::across()`.

Method-specific arguments go through `...: min_value/max_value` for "range", `threshold` and `time` for "speed", `outlier_sd/return_sd` for "excursion", `window_width`, `k` and `min_distance` for "hampel", the ROI bounds for "roi", and `threshold`, `confidence` and `missing` for "confidence".

Value

The same shape as `x`, with failing values replaced by `NA`.

See Also

`filter_with()` for the smoothing and frequency filters, `replace_na_with()` for filling gaps.

Examples

```
mask_na_with(c(1, 5, 10, 15), "range", min_value = 3, max_value = 12)

coords <- data.frame(x = c(0, 1, 2, 50, 4), y = c(0, 0, 0, 0, 0))
mask_na_with(coords, "speed", threshold = 10, time = 1:5)
```

<code>replace_na_across</code>	<i>Fill missing values across an aniframe's spatial columns</i>
--------------------------------	---

Description

The aniframe-level entry point to the `replace_na_*`() family. Fills gaps in the frame's declared position columns, within its existing grouping — so a gap is never filled by interpolating between two different tracks.

Usage

```
replace_na_across(
  data,
  method = c("linear", "spline", "stine", "locf", "value"),
  variables = NULL,
  ...
)
```

Arguments

data	An aniframe.
method	Character string specifying the replacement method: • "linear": Linear interpolation (default) • "spline": Spline interpolation for smoother curves • "stine": Stineman interpolation preserving data shape • "locf": Last observation carried forward • "value": Replace with a constant value
variables	Columns to fill, as a tidyselect expression. Defaults to the declared position columns.
...	Arguments passed to <code>replace_na_with()</code> , such as <code>value</code> , <code>min_gap</code> and <code>max_gap</code> .

Value

An aniframe of the same shape, with gaps filled.

See Also

`replace_na_with()` for the vector-level generic.

Examples

```
tracking_data <- anicore::example_anipoint()
tracking_data$x[c(2, 3, 10)] <- NA

replace_na_across(tracking_data, "linear", max_gap = 5)
replace_na_across(tracking_data, "value", value = 0, variables = c(x, y))
```

<code>replace_na_linear</code>	<i>Replace Missing Values Using Linear Interpolation</i>
--------------------------------	--

Description

Replaces missing values using linear interpolation, with control over both minimum and maximum gap sizes to interpolate.

Usage

```
replace_na_linear(x, min_gap = 1, max_gap = Inf, times = NULL, ...)
```

Arguments

x	A vector containing numeric data with missing values (NAs)
min_gap	Integer specifying minimum gap size to interpolate. Gaps shorter than this will be left as NA. Default is 1 (interpolate all gaps).
max_gap	Integer or Inf specifying maximum gap size to interpolate. Gaps longer than this will be left as NA. Default is Inf (no upper limit).
times	Optional numeric vector of positions for the values in x , normally the frame's index. Interpolation is then over elapsed time rather than over row position, so an irregularly sampled gap is filled correctly. Defaults to row position.
...	Additional parameters passed to stats::approx

Details

The function applies both minimum and maximum gap criteria:

- Gaps shorter than `min_gap` are left as NA
- Gaps longer than `max_gap` are left as NA
- Only gaps that meet both criteria are interpolated. If both parameters are specified, `min_gap` must be less than or equal to `max_gap`.

Value

A numeric vector with NA values replaced by interpolated values where gap length criteria are met.

Examples

```
x <- c(1, NA, NA, 4, 5, NA, NA, NA, 9)
replace_na_linear(x) # interpolates all gaps
replace_na_linear(x, min_gap = 2) # only gaps >= 2
replace_na_linear(x, max_gap = 2) # only gaps <= 2
replace_na_linear(x, min_gap = 2, max_gap = 3) # gaps between 2 and 3
```

replace_na_locf
Replace Missing Values Using Last Observation Carried Forward

Description

Replaces missing values by carrying forward the last observed value, with control over both minimum and maximum gap sizes to fill.

Usage

```
replace_na_locf(x, min_gap = 1, max_gap = Inf)
```

Arguments

x	A vector containing numeric data with missing values (NAs)
min_gap	Integer specifying minimum gap size to fill. Gaps shorter than this will be left as NA. Default is 1 (fill all gaps).
max_gap	Integer or Inf specifying maximum gap size to fill. Gaps longer than this will be left as NA. Default is Inf (no upper limit).

Details

The function applies both minimum and maximum gap criteria:

- Gaps shorter than min_gap are left as NA
- Gaps longer than max_gap are left as NA
- Only gaps that meet both criteria are filled. If both parameters are specified, min_gap must be less than or equal to max_gap.

Value

A numeric vector with NA values replaced by the last observed value where gap length criteria are met.

Examples

```
x <- c(1, NA, NA, 4, 5, NA, NA, NA, 9)
replace_na_locf(x) # fills all gaps
replace_na_locf(x, min_gap = 2) # only gaps >= 2
replace_na_locf(x, max_gap = 2) # only gaps <= 2
replace_na_locf(x, min_gap = 2, max_gap = 3) # gaps between 2 and 3
```

replace_na_spline	<i>Replace Missing Values Using Spline Interpolation</i>
--------------------------	--

Description

Replaces missing values using spline interpolation, with control over both minimum and maximum gap sizes to interpolate.

Usage

```
replace_na_spline(x, min_gap = 1, max_gap = Inf, times = NULL, ...)
```

Arguments

x	A vector containing numeric data with missing values (NAs)
min_gap	Integer specifying minimum gap size to interpolate. Gaps shorter than this will be left as NA. Default is 1 (interpolate all gaps).
max_gap	Integer or Inf specifying maximum gap size to interpolate. Gaps longer than this will be left as NA. Default is Inf (no upper limit).
times	Optional numeric vector of positions for the values in x , normally the frame's index. Interpolation is then over elapsed time rather than over row position, so an irregularly sampled gap is filled correctly. Defaults to row position.
...	Additional parameters passed to <code>stats::spline</code>

Details

The function applies both minimum and maximum gap criteria:

- Gaps shorter than `min_gap` are left as NA
- Gaps longer than `max_gap` are left as NA
- Only gaps that meet both criteria are interpolated If both parameters are specified, `min_gap` must be less than or equal to `max_gap`.

Value

A numeric vector with NA values replaced by interpolated values where gap length criteria are met.

Examples

```
x <- c(1, NA, NA, 4, 5, NA, NA, NA, 9)
replace_na_spline(x) # interpolates all gaps
replace_na_spline(x, min_gap = 2) # only gaps >= 2
replace_na_spline(x, max_gap = 2) # only gaps <= 2
replace_na_spline(x, min_gap = 2, max_gap = 3) # gaps between 2 and 3
```

<code>replace_na_stine</code>	<i>Replace Missing Values Using Stineman Interpolation</i>
-------------------------------	--

Description

Replaces missing values using Stineman interpolation, with control over both minimum and maximum gap sizes to interpolate.

Usage

```
replace_na_stine(x, min_gap = 1, max_gap = Inf, times = NULL, ...)
```

Arguments

<code>x</code>	A vector containing numeric data with missing values (NAs)
<code>min_gap</code>	Integer specifying minimum gap size to interpolate. Gaps shorter than this will be left as NA. Default is 1 (interpolate all gaps).
<code>max_gap</code>	Integer or Inf specifying maximum gap size to interpolate. Gaps longer than this will be left as NA. Default is Inf (no upper limit).
<code>times</code>	Optional numeric vector of positions for the values in <code>x</code> , normally the frame's index. Interpolation is then over elapsed time rather than over row position, so an irregularly sampled gap is filled correctly. Defaults to row position.
<code>...</code>	Additional parameters passed to <code>stinepack::stinterp</code>

Details

The function applies both minimum and maximum gap criteria:

- Gaps shorter than `min_gap` are left as NA
- Gaps longer than `max_gap` are left as NA
- Only gaps that meet both criteria are interpolated. If both parameters are specified, `min_gap` must be less than or equal to `max_gap`.

Stineman interpolation is particularly good at preserving the shape of the data and avoiding overshooting.

Value

A numeric vector with NA values replaced by interpolated values where gap length criteria are met.

References

Stineman, R. W. (1980). A consistently well-behaved method of interpolation. *Creative Computing*, 6(7), 54-57.

Examples

```
x <- c(1, NA, NA, 4, 5, NA, NA, NA, 9)
replace_na_stine(x) # interpolates all gaps
replace_na_stine(x, min_gap = 2) # only gaps >= 2
replace_na_stine(x, max_gap = 2) # only gaps <= 2
replace_na_stine(x, min_gap = 2, max_gap = 3) # gaps between 2 and 3
```

replace_na_value	<i>Replace Missing Values with a Constant Value</i>
------------------	---

Description

Replaces missing values with a specified constant value, with control over both minimum and maximum gap sizes to fill.

Usage

```
replace_na_value(x, value, min_gap = 1, max_gap = Inf)
```

Arguments

x	A vector containing numeric data with missing values (NAs)
value	Numeric value to use for replacement
min_gap	Integer specifying minimum gap size to fill. Gaps shorter than this will be left as NA. Default is 1 (fill all gaps).
max_gap	Integer or Inf specifying maximum gap size to fill. Gaps longer than this will be left as NA. Default is Inf (no upper limit).

Details

The function applies both minimum and maximum gap criteria:

- Gaps shorter than min_gap are left as NA
- Gaps longer than max_gap are left as NA
- Only gaps that meet both criteria are filled If both parameters are specified, min_gap must be less than or equal to max_gap.

Value

A numeric vector with NA values replaced by the specified value where gap length criteria are met.

Examples

```
x <- c(1, NA, NA, 4, 5, NA, NA, NA, 9)
replace_na_value(x, value = 0) # fills all gaps with 0
replace_na_value(x, value = -1, min_gap = 2) # only gaps >= 2
replace_na_value(x, value = -999, max_gap = 2) # only gaps <= 2
replace_na_value(x, value = 0, min_gap = 2, max_gap = 3) # gaps between 2 and 3
```

replace_na_with	<i>Fill missing values by a named method</i>
-----------------	--

Description

Generic entry point to the `replace_na_*`() family: pick the method with an argument rather than by choosing a function.

Usage

```
replace_na_with(
  x,
  method = c("linear", "spline", "stine", "locf", "value"),
  value = NULL,
  min_gap = 1,
  max_gap = Inf,
  times = NULL,
  ...
)
```

Arguments

x	A numeric vector, or a data frame of numeric columns, with missing values to fill.
method	Character string specifying the replacement method: • "linear": Linear interpolation (default) • "spline": Spline interpolation for smoother curves • "stine": Stineman interpolation preserving data shape • "locf": Last observation carried forward • "value": Replace with a constant value
value	Numeric value for replacement when method = "value" .
min_gap	Integer specifying the minimum gap size to fill. Gaps shorter than this are left as NA . Default 1 (fill all gaps).
max_gap	Integer or Inf specifying the maximum gap size to fill. Gaps longer than this are left as NA . Default Inf (no limit).
times	Optional numeric vector of positions for the values in x , normally the frame's index. Interpolation is then over elapsed time rather than over row position, so an irregularly sampled gap is filled correctly. Defaults to row position, and is ignored by the methods that do not interpolate.
...	Additional parameters passed to the underlying function.

Details

Returns the same shape it is given. Every method is univariate, so a data frame is filled one column at a time — meaning this generic does work with `dplyr::across()` as well as `dplyr::pick()`.

Value

The same shape as `x`, with gaps filled where the gap-length criteria are met.

See Also

`filter_with()` for smoothing, `mask_na_with()` for masking.

Examples

```
x <- c(1, NA, NA, 4, 5, NA, NA, NA, 9)

replace_na_with(x, "linear")
replace_na_with(x, "value", value = 0)
replace_na_with(x, "linear", max_gap = 2)

# A frame is filled column by column
replace_na_with(data.frame(a = x, b = rev(x)), "locf")
```

Index

`anicores::anistructure()`, 49
`anicores::as_anisegment()`, 50, 51
`anicores::get_index()`, 3, 39
`anicores::set_structure()`, 49, 51

`data.table::frollmean()`, 25, 26
`data.table::frollmedian()`, 27
`dplyr::across()`, 6, 8, 32, 42, 45, 53, 54, 61
`dplyr::mutate()`, 3, 7, 41, 42, 44, 45, 47, 52, 53
`dplyr::pick()`, 6, 7, 41, 42, 44, 45, 47, 52, 61

`filter_across`, 2
`filter_across()`, 4, 5, 7, 8, 32
`filter_ccma`, 4
`filter_deadband`, 6
`filter_gaussian`, 9
`filter_gaussian()`, 3, 5
`filter_highpass`, 10, 14, 21
`filter_highpass_fft`, 12, 23
`filter_kalman`, 15
`filter_kalman()`, 17
`filter_kalman_irregular`, 17
`filter_lowpass`, 12, 19, 23, 30
`filter_lowpass()`, 7, 8, 25
`filter_lowpass_fft`, 14, 21
`filter_one_euro`, 23
`filter_one_euro()`, 8
`filter_rollmean`, 25
`filter_rollmean()`, 27, 31
`filter_rollmedian`, 27
`filter_rollmedian()`, 44, 46
`filter_sgolay`, 28
`filter_sgolay()`, 5
`filter_triangular`, 30
`filter_with`, 31
`filter_with()`, 3, 4, 54, 62
`find_peaks`, 33, 38

`find_troughs`, 35, 36

`mask_na_across`, 39
`mask_na_across()`, 41–45, 47, 49–53
`mask_na_confidence`, 41
`mask_na_confidence()`, 40
`mask_na_excursion`, 42
`mask_na_excursion()`, 45, 46
`mask_na_hampel`, 44
`mask_na_range`, 46
`mask_na_roi`, 47
`mask_na_segment_length`, 48
`mask_na_speed`, 52
`mask_na_speed()`, 43, 46
`mask_na_with`, 53
`mask_na_with()`, 40, 62

`replace_na_across`, 54
`replace_na_linear`, 55
`replace_na_locf`, 56
`replace_na_spline`, 57
`replace_na_stine`, 58
`replace_na_value`, 60
`replace_na_with`, 61
`replace_na_with()`, 5, 12, 14, 21, 23, 24, 30, 44, 54, 55

`signal::sgolayfilt()`, 29