Package: aniprocess (via r-universe)

June 6, 2026

Type Package

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

Version 0.2.0

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

License MIT + file LICENSE

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

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

Encoding UTF-8

LazyData true

Imports aniframe, cli, data.table (>= 1.18.0), dplyr, methods, rlang

Suggests covr, knitr, signal, stinepack, testthat (>= 3.0.0)

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

Config/testthat/edition 3

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

VignetteBuilder knitr

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

Date/Publication 2026-05-07 21:58:21 UTC

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

RemoteRef HEAD

RemoteSha dd8a6be17188c4da248dcffd732ec2d62d7b9343

Contents

`filter_aniframe` *Smooth Movement Data*

Description

[Experimental]

Applies smoothing filters to movement tracking data to reduce noise.

Usage

```
filter_aniframe(
  data,
  method = c("rollmedian", "rollmean", "triangular", "gaussian", "kalman", "sgolay",
    "lowpass", "highpass", "lowpass_fft", "highpass_fft"),
  use_derivatives = FALSE,
  ...
)
```

Arguments

`data`	An aniframe. Spatial columns to filter are taken from the metadata field `variables_where` (e.g. `c("x", "y")` or `c("x", "y", "z")`). Filtering is applied within the aniframe's existing grouping, which is driven by `variables_what` (e.g. `c("individual", "keypoint")`, `"track"`, or `character(0)` for a single trajectory). Single-track data without an `individual` column is supported.
`method`	Character string specifying the smoothing method. Options: • `"kalman"`: Kalman filter (see `filter_kalman()`) • `"sgolay"`: Savitzky-Golay filter (see `filter_sgolay()`) • `"lowpass"`: Low-pass filter (see `filter_lowpass()`) • `"highpass"`: High-pass filter (see `filter_highpass()`) • `"lowpass_fft"`: FFT-based low-pass filter (see `filter_lowpass_fft()`) • `"highpass_fft"`: FFT-based high-pass filter (see `filter_highpass_fft()`) • `"rollmean"`: Rolling mean filter (see `filter_rollmean()`) • `"rollmedian"`: Rolling median filter (see `filter_rollmedian()`) • `"triangular"`: Triangular filter (see `filter_triangular()`) • `"gaussian"`: Gaussian kernel smoother (see `filter_gaussian()`)
`use_derivatives`	Filter on the derivative values instead of coordinates (important for e.g. trackball or accelerometer data)
`...`	Additional arguments passed to the specific filter function

Details

This function is a wrapper that applies the chosen filter to every spatial coordinate listed in `variables_where`, respecting the aniframe's existing grouping. Each filtering method has its own specific parameters - see the documentation of the individual filter functions for details:

- `filter_kalman()`: Kalman filter parameters
- `filter_sgolay()`: Savitzky-Golay filter parameters
- `filter_lowpass()`: Low-pass filter parameters
- `filter_highpass()`: High-pass filter parameters
- `filter_lowpass_fft()`: FFT-based low-pass filter parameters
- `filter_highpass_fft()`: FFT-based high-pass filter parameters
- `filter_rollmean()`: Rolling mean parameters (window_width, min_obs)
- `filter_rollmedian()`: Rolling median parameters (window_width, min_obs)
- `filter_triangular()`: Triangular filter parameters (window_width, min_obs)
- `filter_gaussian()`: Gaussian kernel parameters (sigma, window_width)

Value

An aniframe with the same structure as the input, but with smoothed spatial coordinates.

See Also

- filter_kalman()
- filter_sgolay()
- filter_lowpass()
- filter_highpass()
- filter_lowpass_fft()
- filter_highpass_fft()
- filter_rollmean()
- filter_rollmedian()
- filter_triangular()
- filter_gaussian()

Examples

```
## Not run:
# Apply rolling median with window of 5
filter_aniframe(tracking_data, "rollmedian", window_width = 5, min_obs = 1)

## End(Not run)
```

`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 = FALSE,
  ...
)
```

Arguments

`data`	An aniframe in Cartesian coordinates with 2 or 3 spatial columns (set via the `variables_where` metadata field). The curvature math is Cartesian-specific (cross products, Euclidean norms, circumradius), so polar / cylindrical / spherical aniframes are rejected.
`window_width_ma`	Integer width of the moving-average kernel (must be odd; even values are rounded up). Larger = more smoothing. Default `11`.
`window_width_cc`	Integer width of the curvature-correction kernel (must be odd). Larger = smoother correction but uses curvature info from further away. Default `7`.
`kernel`	Kernel shape for both stages. One of `"hanning"` (default; raised cosine) or `"uniform"` (boxcar).
`boundary`	Edge-handling strategy. Currently only `"padding"` (repeat the first and last point so output length equals input length).
`cc_mode`	If `FALSE`, returns just the moving-average result without curvature correction. Useful for comparison.
`na_action`	How to fill `NA` values in the spatial columns before filtering. One of `"linear"` (default), `"spline"`, `"stine"`, `"locf"`, `"value"`, or `"error"` (abort if any `NA`s are present). See `replace_na()`.
`keep_na`	If `TRUE`, restore `NA`s at their original positions in the output. Defaults to `FALSE`.
`...`	Additional arguments passed to `replace_na()` (e.g. `value`, `min_gap`, `max_gap`).

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_aniframe()`). 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()`.

Smoothing is applied within the aniframe's existing grouping (driven by `variables_what`), so each individual / track / keypoint is smoothed as its own trajectory.

Value

An aniframe of the same shape as the input, with the spatial columns smoothed.

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)*.

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

Examples

```
## Not run:
filter_ccma(tracking_data, window_width_ma = 11, window_width_cc = 7)

## End(Not run)
```

---

filter_gaussian *Apply Gaussian Kernel Smoother*

Description

Convolves a numeric vector with a discrete Gaussian kernel.

Usage

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

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 * ceiling(3 * sigma) + 1`, which truncates the kernel at ± 3 .

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 `NA`s 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 = FALSE,
  ...
)
```

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 sampling_rate/2.
`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 to handle NA values before filtering. One of: - "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 - "error": Raise an error if NAs are present

`keep_na`	Logical indicating whether to restore NAs to their original positions after filtering (default = FALSE)
...	Additional arguments passed to replace_na(). Common options include: - value: Numeric value for replacement when na_action = "value" - min_gap: Minimum gap size to interpolate/fill - max_gap: Maximum gap size to interpolate/fill

Details

The Butterworth filter response falls off at -6*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() internally for handling missing values. See ?replace_na 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` 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)
```

---

filter_highpass_fft *Apply FFT-based Highpass Filter to Signal*

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 = FALSE,
  ...
)
```

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 sampling_rate/2.
`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 to handle NA values before filtering. One of: - "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 - "error": Raise an error if NAs are present
`keep_na`	Logical indicating whether to restore NAs to their original positions after filtering (default = FALSE)
`...`	Additional arguments passed to replace_na(). Common options include: - value: Numeric value for replacement when na_action = "value" - min_gap: Minimum gap size to interpolate/fill - max_gap: 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() internally for handling missing values. See ?replace_na 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` 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(
  measurements,
  sampling_rate,
  base_Q = NULL,
  R = NULL,
  initial_state = NULL,
  initial_P = NULL
)
```

Arguments

`measurements`	Numeric vector containing the 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.

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(measurements) / sampling_rate` (so the per-step process noise `Q = base_Q / 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(measurements)` 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 measurements 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
measurements <- c(1, 1.1, NA, 0.9, 1.2, NA, 0.8, 1.1)
filtered <- filter_kalman(measurements, sampling_rate = 60)

# Custom parameters for more aggressive filtering
filtered_custom <- filter_kalman(measurements,
                                 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(
  measurements,
  times,
  base_Q = NULL,
  R = NULL,
  initial_state = NULL,
  initial_P = NULL,
  resample = FALSE,
  resample_freq = NULL
)
```

Arguments

`measurements`	Numeric vector containing the measurements to be filtered.
`times`	Numeric vector of timestamps corresponding to measurements.
`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).

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 measurements, increase R

See Also

filter_kalman for regularly sampled data

Examples

```
# Example with irregular sampling
measurements <- 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(measurements, times)

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

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

---

filter_lowpass *Apply Butterworth Lowpass Filter to Signal*

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 = FALSE,
  ...
)
```

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 sampling_rate/2.
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 to handle NA values before filtering. One of: - "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 - "error": Raise an error if NAs are present

`keep_na` Logical indicating whether to restore NAs to their original positions after filtering (default = FALSE)

`...` Additional arguments passed to replace_na(). Common options include: - value: Numeric value for replacement when na_action = "value" - min_gap: Minimum gap size to interpolate/fill - max_gap: Maximum gap size to interpolate/fill

Details

The Butterworth filter response falls off at -6*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() internally for handling missing values. See ?replace_na 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 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 *Apply FFT-based Lowpass Filter to Signal*

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 = FALSE,
  ...
)
```

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 sampling_rate/2.
`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 to handle NA values before filtering. One of: - "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 - "error": Raise an error if NAs are present
`keep_na`	Logical indicating whether to restore NAs to their original positions after filtering (default = FALSE)
`...`	Additional arguments passed to replace_na(). Common options include: - value: Numeric value for replacement when na_action = "value" - min_gap: Minimum gap size to interpolate/fill - max_gap: 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() internally for handling missing values. See ?replace_na 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` 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_na_confidence` *Filter low-confidence values in a dataset*

Description

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

Usage

```
filter_na_confidence(data, threshold = 0.6)
```

Arguments

`data`	An aniframe containing a `confidence` column and spatial columns as defined in the metadata's `variables_where`.
`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.

Value

An aniframe with the same structure as the input, but where spatial and `confidence` values are replaced with `NA` if the confidence is below the threshold.

Examples

```
# 2D example
data <- aniframe::aniframe(
  time = 1:5,
  x = 1:5,
  y = 6:10,
  confidence = c(0.5, 0.7, 0.4, 0.8, 0.9)
)

filter_na_confidence(data, threshold = 0.6)

# With z column (3D)
data_3d <- aniframe::aniframe(
  time = 1:5,
  x = 1:5,
  y = 6:10,
  z = 11:15,
  confidence = c(0.5, 0.7, 0.4, 0.8, 0.9),
  variables_where = c("x", "y", "z")
)

filter_na_confidence(data_3d, threshold = 0.6)
```

---

filter_na_excursion *Filter Out Position Excursions That Return*

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

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

Arguments

`data`	An aniframe.
`outlier_sd`	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`.
`return_sd`	Threshold (in standard deviations) for the "return to overall median position" acceptance check. Todd's default is `1`.

by_axis 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 filter_na_speed()).

Details

For each spatial coordinate listed in the metadata field variables_where (and within each existing aniframe group), 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 * , an excursion starts: that frame is flagged and subsequent frames are flagged until the position is either within outlier_sd * of its pre-excursion value, or within return_sd * 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.

Spatial columns and confidence (if present) are set to NA at flagged rows, matching the convention used by filter_na_speed().

Value

An aniframe of the same shape, 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

filter_na_speed() for single-frame outliers.

Examples

```
## Not run:
# Default Todd thresholds, per-axis.
filter_na_excursion(tracking_data)

# Joint Euclidean variant, looser thresholds.
filter_na_excursion(tracking_data, outlier_sd = 4, by_axis = FALSE)

## End(Not run)
```

filter_na_range *Filter values outside a range to NA*

Description

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

Usage

```
filter_na_range(x, min = -Inf, max = Inf)
```

Arguments

x	A numeric vector to filter
min	Minimum value (inclusive). Values below this become NA. Default is -Inf (no lower bound).
max	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

```
filter_na_range(c(1, 5, 10, 15), min = 3, max = 12)
# Returns: c(NA, 5, 10, NA)

filter_na_range(c(1, NA, 10), min = 5)
# Returns: c(NA, NA, 10)
```

filter_na_roi *Filter coordinates outside a region of interest (ROI)*

Description

Filters out 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). Automatically handles 2D or 3D data based on the spatial variables in the aniframe metadata.

Usage

```
filter_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

`data`	An aniframe containing spatial coordinates.
`x_min`, `x_max`	Bounds for x-coordinate (rectangular/cuboid ROI).
`y_min`, `y_max`	Bounds for y-coordinate (rectangular/cuboid ROI).
`z_min`, `z_max`	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

An aniframe with coordinates outside ROI set to NA.

Examples

```
# Create sample 2D data
sample_data <- aniframe::aniframe(
  time = 1:9,
  x = rep(c(25, 50, 75), 3),
  y = rep(c(25, 50, 75), each = 3)
)

# Rectangular ROI example
sample_data |>
  filter_na_roi(x_min = 20, x_max = 60, y_min = 20, y_max = 60)

# Circular ROI example
sample_data |>
  filter_na_roi(x_center = 50, y_center = 50, radius = 30)

# 3D cuboid ROI example
sample_3d <- aniframe::aniframe(
```

```
  time = 1:8,
  x = rep(c(25, 75), 4),
  y = rep(c(25, 75), each = 2, times = 2),
  z = rep(c(25, 75), each = 4),
  variables_where = c("x", "y", "z")
)

sample_3d |>
  filter_na_roi(x_min = 20, x_max = 60, y_min = 20, y_max = 60, z_min = 20, z_max = 60)
```

`filter_na_speed` *Filter values by speed threshold*

Description

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

Usage

```
filter_na_speed(data, threshold = "auto")
```

Arguments

`data`	An aniframe containing spatial coordinates and a time column.
`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.

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.

When using `threshold = "auto"`, the threshold is set to the mean speed plus three standard deviations.

Value

An aniframe with the same structure as the input, but with spatial and confidence values replaced by NA where speed exceeds the threshold.

Examples

```
data <- aniframe::aniframe(
  time = 1:5,
  x = c(1, 2, 4, 7, 11),
  y = c(1, 1, 2, 3, 5),
  confidence = c(0.8, 0.9, 0.7, 0.85, 0.6)
)

# Filter data by a speed threshold of 3
filter_na_speed(data, threshold = 3)

# Use automatic threshold
filter_na_speed(data, threshold = "auto")
```

---

filter_rollmean *Apply Rolling Mean Filter*

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("right", "left", "center")
)
```

Arguments

`x`	Numeric vector to filter.
`window_width`	Integer specifying window size for the rolling calculation.
`min_obs`	Minimum number of non-NA values required in the window. Positions with fewer non-NA values return `NA`. Defaults to `1`.
`align`	Window alignment. One of `"right"` (default), `"left"`, or `"center"`.

Details

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 `(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`.

`filter_rollmedian` *Apply Rolling Median Filter*

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("right", "left", "center")
)
```

Arguments

`x`	Numeric vector to filter.
`window_width`	Integer specifying window size for the rolling calculation.
`min_obs`	Minimum number of non-NA values required in the window. Positions with fewer non-NA values return `NA`. Defaults to `1`.
`align`	Window alignment. One of `"right"` (default), `"left"`, or `"center"`.

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`.

`filter_sgolay` *Apply Savitzky-Golay Filter to Movement Data*

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_size = ceiling(sampling_rate/10) * 2 + 1,
  order = 3,
  na_action = "linear",
  keep_na = FALSE,
  ...
)
```

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_size`	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 sampling_rate/10 (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)
`na_action`	Method to handle NA values before filtering. One of: - "linear": Linear interpolation (default) - "spline": Spline interpolation for smoother curves - "locf": Last observation carried forward - "value": Replace with a constant value - "error": Raise an error if NAs are present
`keep_na`	Logical indicating whether to restore NAs to their original positions after filtering (default = FALSE)
`...`	Additional arguments passed to replace_na()

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_size:
  - 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_size = 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_size

Common values by application:

- Gait analysis (60 FPS): window_size=15, order=3
- Sports biomechanics (120 FPS): window_size=21, order=4
- Impact analysis (500 FPS): window_size=51, order=4
- Posture analysis (60 FPS): window_size=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.

See Also

`filter_lowpass` for frequency-based filtering `replace_na` 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_size = 11, order = 4)

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

`filter_triangular` *Apply Triangular Filter*

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 * window_width - 1` with peak weight at the centre.

Usage

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

Arguments

`x`	Numeric vector to filter.
`window_width`	Integer width of each rolling-mean pass. The effective triangular kernel has width `2 * window_width - 1`.
`min_obs`	Minimum number of non-NA values required per window per pass. Defaults to `1`.
`align`	Window alignment, passed to `filter_rollmean()`. One of `"center"` (default), `"right"`, or `"left"`.

Details

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

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)
```

---

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_size = 3
)
```

Arguments

`x`	Numeric vector containing the time series data
`min_height`	Minimum height threshold for peaks (default: -Inf)
`min_prominence`	Minimum prominence threshold for peaks (default: 0)
`plateau_handling`	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

`window_size` 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_size 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_size of 3 compares each point with its immediate neighbors). The first and last (window_size-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_size 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_size 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_size 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_size = 5)  # More stringent peak detection

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

# With plateau handling
x <- c(1, 3, 3, 3, 2, 4, 4, 1)
find_peaks(x, plateau_handling = "middle", window_size = 3)  # Middle of plateaus
find_peaks(x, plateau_handling = "all", window_size = 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_size = 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_size = 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_size`	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_size 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_size of 3 compares each point with its immediate neighbors). The first and last (window_size-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_size 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_size 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_size 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_size = 5)  # More stringent trough detection

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

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

`replace_na` *Replace Missing Values Using Various Methods*

Description

A wrapper function that replaces missing values using various interpolation or filling methods.

Usage

```
replace_na(x, method = "linear", value = NULL, min_gap = 1, max_gap = Inf, ...)
```

Arguments

`x`	A vector containing numeric data with missing values (NAs)
`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 minimum gap size to interpolate/fill. Gaps shorter than this will be left as NA. Default is 1 (handle all gaps).
`max_gap`	Integer or Inf specifying maximum gap size to interpolate/fill. Gaps longer than this will be left as NA. Default is Inf (no upper limit).
`...`	Additional parameters passed to the underlying interpolation functions

Value

A numeric vector with NA values replaced according to the specified method where gap length criteria are met.

See Also

- replace_na_linear() for linear interpolation details
- replace_na_spline() for spline interpolation details
- replace_na_stine() for Stineman interpolation details
- replace_na_locf() for last observation carried forward details
- replace_na_value() for constant value replacement details

Examples

```
## Not run:
x <- c(1, NA, NA, 4, 5, NA, NA, NA, 9)

# Different methods
replace_na(x, method = "linear")
replace_na(x, method = "spline")
replace_na(x, method = "stine")
replace_na(x, method = "locf")
replace_na(x, method = "value", value = 0)

# With gap constraints
replace_na(x, method = "linear", min_gap = 2)
replace_na(x, method = "spline", max_gap = 2)
replace_na(x, method = "linear", min_gap = 2, max_gap = 3)

## End(Not run)
```

---

replace_na_linear *Replace Missing Values Using Linear Interpolation*

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, ...)
```

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).
...	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

```
## Not run:
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

## End(Not run)
```

---

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

```
## Not run:
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

## End(Not run)
```

`replace_na_spline` *Replace Missing Values Using Spline Interpolation*

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, ...)
```

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).
`...`	Additional parameters passed to stats::spline

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

```
## Not run:
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

## End(Not run)
```

---

replace_na_stine *Replace Missing Values Using Stineman Interpolation*

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, ...)
```

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).
...	Additional parameters passed to stinepack::stinterp

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.

Examples

```
## Not run:
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

## End(Not run)
```

---

replace_na_value *Replace Missing Values with a Constant Value*

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

```
## Not run:
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

## End(Not run)
```

Index