Package 'mnirs'

September 13, 2026

Title Muscle Near-Infrared Spectroscopy Processing and Analysis

Version 0.8.0

Description Read, process, and analyse data from muscle near-infrared spectroscopy (mNIRS) devices. Import raw data from file and return time-series data and metadata. Standardised methods for cleaning, filtering, transforming, and analysing mNIRS data. Custom plot theme and colour palette. Intended for mNIRS researchers and practitioners in exercise physiology, sports science, and clinical practice.

License MIT + file LICENSE

URL https://jemarnold.github.io/mnirs/, https://github.com/jemarnold/mnirs

BugReports https://github.com/jemarnold/mnirs/issues

Depends R (>= 4.1)

Imports cli, data.table, lifecycle, readxl, rlang, stats, tibble, tidyselect, utils

Suggests dplyr, ggplot2, knitr, quarto, scales, signal, testthat (>= 3.0.0), zoo

VignetteBuilder quarto

Config/Needs/website quarto, tidyverse

Config/roxygen2/version 8.1.0

Config/testthat/edition 3

Encoding UTF-8

NeedsCompilation no

Author Jem Arnold [aut, cre, cph] (ORCID: <https://orcid.org/0000-0003-3908-9447>)

Maintainer Jem Arnold <jem.arnold@gmail.com>

Repository CRAN

Date/Publication 2026-09-13 00:00:02 UTC

Contents

analyse_kinetics *Analyse kinetics across mNIRS channels and intervals*

Description

Fit oxygenation kinetics (response time course) models with various parametric and non-parametric methods.

Usage

```
analyse_kinetics(
  data,
  nirs_channels = NULL,
  time_channel = NULL,
 method = c("response_time", "peak_slope", "monoexponential", "exponential_drift",
    "biexponential", "sigmoidal", "sigmoidal_drift"),
  start_time = NULL,
  direction = c("auto", "positive", "negative"),
  end_window = Inf,
  group_intervals = "ensemble",
  zero_time = FALSE,
  verbose = TRUE,
  ...,
  response_fraction = 0.5,
  width = NULL,
  span = NULL,
  align = c("centre", "left", "right"),
  partial = FALSE,
  na.rm = FALSE,
  use_TD = TRUE,
  shape = c("symmetric", "gompertz", "gompertz_left"),
  drift_fraction = NULL,
  fix = NULL
)

analyze_kinetics(
  data,
  nirs_channels = NULL,
  time_channel = NULL,
 method = c("response_time", "peak_slope", "monoexponential", "exponential_drift",
    "biexponential", "sigmoidal", "sigmoidal_drift"),
  start_time = NULL,
  direction = c("auto", "positive", "negative"),
  end_window = Inf,
  group_intervals = "ensemble",
  zero_time = FALSE,
  verbose = TRUE,
```

```
  ...
)
```

Arguments

`data`	A data frame, a list of data frames, or a grouped data frame of class *"mnirs"* containing time series data and metadata (see *Details*).
`nirs_channels`	A character vector giving the names of mNIRS columns to operate on. Must match column names in `data` exactly. • If `NULL` (default), the `nirs_channels` metadata attribute of `data` is used.
`time_channel`	A character string naming the time or sample column. Must match a column name in `data` exactly. • If `NULL` (default), the `time_channel` metadata attribute of `data` is used.
`method`	A character string specifying the kinetics analysis method. Additional arguments must be specified for each method. See *Details*. `"response_time"` Fractional (e.g. 50%, 63.2%, 90%) response time. Additional arguments: `response_fraction`. See `response_time()`. `"peak_slope"` Peak rolling linear regression slope. Additional arguments: `width` or `span`, `align`, `partial`, `na.rm`. See `peak_slope()`. `"monoexponential"` Monoexponential curve fit via `stats::nls()`. Additional arguments: `use_TD`, `fix`, `control`. See `monoexponential()`. `"exponential_drift"` Two-phase kinetics: monoexponential primary phase with a secondary linear drift, fit via `stats::nls()`. Additional arguments: `use_TD`, `drift_fraction`, `fix`, `control`. See `exponential_drift()`. `"biexponential"` Two-phase kinetics: overlapping fast primary and slow secondary exponential curves fit via `stats::nls()`. Additional arguments: `use_TD`, `fix`, `control`. See `biexponential()`. `"sigmoidal"` Logistic or Gompertz-family curve fit via `stats::nls()`. Additional arguments: `shape`, `fix`, `control`. See `logistic()`. `"sigmoidal_drift"` Two-phase kinetics: Logistic or Gompertz-family primary phase with a secondary linear drift, fit via `stats::nls()`. Additional arguments: `shape`, `drift_fraction`, `fix`, `control`. See `sigmoidal_drift()`.
`start_time`	A numeric value in units of `time_channel` specifying the response onset (effectively time = `0` of the fit). If `NULL` (*default*), retrieves `interval_times` from *"mnirs"* metadata, or falls back to `0` or the first positive time value (see *Details*).
`direction`	A character string specifying the response direction `"positive"`, or `"negative"`, or detect with `"auto"` (*default*). See *Details*.
`end_window`	A numeric value in units of `time_channel` specifying the window in which to look for the end of the kinetics fit; with no greater/ lesser values within `end_window` after the first extrema (min/max). `end_window = Inf` (*default*) returns the global extreme from the full data range (see *Details*). For *"biexponential"*, `end_window` bounds the fast-phase window only, and the default is `30` sec; the full model is then fit to the full data range.

`group_intervals`
: Either `"ensemble"` (*default*) to analyse all samples of each data frame together, or a `list()` of integer vectors of sample (row) numbers, each analysed as a separate interval, e.g. `list(trial1 = 1:10, trial2 = 11:20)`.

  List names become interval names (`interval_<n>` when unnamed) (see *Details*).

`zero_time`
: Logical. Default is `FALSE`. If `TRUE`, re-bases `time_channel` values to start from zero within each interval or `group_intervals` group.

`verbose`
: Logical. `TRUE` (*default*) will display, and `FALSE` will silence warnings and information messages helpful for troubleshooting. Global default can be set via `options(mnirs.verbose = FALSE)`.

`...`
: Additional arguments passed to the underlying method function. See *Details*. For the `stats::nls()` methods (**monoexponential, exponential_drift, biexponential, sigmoidal, sigmoidal_drift**), `control = list()` can be passed to `stats::nls.control()`, e.g. `control = list(maxiter = 200)`, applied globally to all channels and intervals.

`response_fraction`
: **response_time**: A numeric vector in the range `[0, 1]` specifying the fractional response amplitude(s) to detect. Defaults to `0.5` (50% response, i.e. half-response time). Multiple values (e.g. `c(0.5, 0.632)`) return one coefficient row per fraction.

`width`
: **peak_slope**: An integer defining the local window in number of samples around `idx` in which to calculate slopes. Only one of either `width` or `span` must be defined.

`span`
: **peak_slope**: A numeric value defining the local window time span in units of `time_channel` around `idx` in which to calculate slopes. Only one of either `width` or `span` must be defined.

`align`
: **peak_slope**: Window alignment as *"centre"/"center"* (the *default*), *"left"*, or *"right"*. Where *"left"* is forward looking, and *"right"* is backward looking from the current sample by the `width` or `span`.

`partial`
: **peak_slope**: Logical; default is `FALSE`, requires local windows to have complete number of samples specified by `width` or `span`. If `TRUE`, processes local windows with at minimum two available samples. See *Details*.

`na.rm`
: **peak_slope**: Logical; default is `FALSE`, propagates `NA`s to the returned vector and may return errors or warnings. If `TRUE`, ignores `NA`s and processes available valid samples within the local window. (see *Details*).

`use_TD`
: **monoexponential, exponential_drift, biexponential**: Logical; default is `TRUE`, attempts to fit the model with a "time-delay" parameter `TD` between `start_time` and the response onset. If `use_TD = FALSE` or the fit fails (with a warning), attempts to fall back to a reduced parameter model without `TD`.

`shape`
: **sigmoidal, sigmoidal_drift**: Character; the 4-parameter sigmoidal shape to fit. One of `"symmetric"` (*default*; inflection occurs at 50% amplitude), `"gompertz"` (early-inflection; 36.8% `1/e`), or `"gompertz_left"` (late-inflection; 63.2% `1 - 1/e`).

`drift_fraction` **exponential_drift, sigmoidal_drift**: A numeric fraction of the primary amplitude in `(0.5, 1)` at which the linear secondary drift begins, where the primary response reaches `A + drift_fraction * (B - A)`. *Default* is `0.95`. Specify per-channel as a list keyed by channel name, e.g. `drift_fraction = list(smo2 = 0.9)`. See *Details*.

`fix` **monoexponential, exponential_drift, biexponential, sigmoidal, sigmoidal_drift**: An *optional* named list of model parameters (coefficients) to hold constant during fitting, e.g. `fix = list(A = 0)` fixes the starting amplitude at `0`.

Fixed parameters are excluded from estimation and returned as constant. Specify per-channel as a list of lists keyed by channel name, e.g. `fix = list(smo2 = list(A = 0))`. See *Details*.

Details

Data input formats:

`analyse_kinetics()` accepts `data` in multiple formats:

- A **single *"mnirs"* data frame** is processed as a single interval.
- A **list of *"mnirs"* data frames**: each interval is processed separately.
- A **grouped *"mnirs"* data frame**, e.g. with `dplyr::group_by()`: the data frame is split by grouping levels and each group is processed as a separate interval.
- A special case for *recursive analysis*: The results from `analyse_kinetics()` can be fed into a second call to analyse the `results$coefficients` table, split into data frames by `nirs_channel` with one row per interval (see *Recursive analysis*).

Specified `nirs_channels` (or channels retrieved from *"mnirs"* metadata) will be analysed and results returned as a formatted table.

Response start_time and the baseline window:

`start_time` should be specified as the time point separating the pre-response baseline (`time_channel <= start_time`) from the start of the systematic response fit window (`time_channel > start_time`). This often corresponds to a stimulus or start/end of an intervention (e.g. start/end of an exercise interval).

For intervals extracted with `extract_intervals()`, `start_time` can be retrieved from *"mnirs"* metadata. Otherwise `start_time` defaults to `0` or the first positive `time_channel` value.

All methods are fitted on *time elapsed* from `start_time`, so returned time & duration coefficients are relative to response onset (e.g. `start_time = 0`).

- For *"response_time"*, the baseline window before `start_time` defines the mean starting amplitude `A` directly and anchors the start of the `response_time` parameter.
- For *"peak_slope"*, `start_time` anchors the start of the `peak_slope_time` parameter.
- For *"exponential"*- and *"sigmoidal"*-family, the baseline window before `start_time` anchors the starting fitted amplitude `A` and the start of `TD` and `MRT`, or `xmid` parameters. (see respective *method* sections below).

The time-delay models (*"exponential"*-family with `use_TD = TRUE`) are flat at `A` before `TD`, so the pre-onset baseline is included in the fit and anchors `A`. Their reduced forms (`use_TD = FALSE`, or a `TD` fit that failed and fell back) have no such flat region and are fitted only where `time_channel >= start_time`.

Response direction and the fit end_window:

`direction` is detected automatically by default as either *"positive"* (upward) or *"negative"* (downward) response, and can be overwritten manually. `end_window` is a time span in units of `time_channel` defining the end of the kinetics fitting window by locating the first extrema (peak/trough, depending on `direction`) with no greater/lesser values within the subsequent `end_window` time span. The curve fitting window extends to the end of `end_window` beyond the detected extrema.

For *"exponential"*- and *"sigmoidal"*-family methods, `direction` also constrains the sign of the fitted amplitude `B - A`, and the sigmoidal `slope`. For the *"biexponential"* method, `direction` constrains the sign of the fast-phase amplitude `B - A`. A fit that cannot satisfy the requested direction returns `NA` coefficients with a warning.

Grouping samples with group_intervals:

`group_intervals = "ensemble"` (the *default*) analyses every sample of each data frame together as one interval. A `list()` of sample (row) numbers instead splits each data frame into one interval per group, e.g. for a 20-row data frame:

```
analyse_kinetics(
    data,
    method = "monoexponential",
    group_intervals = list(trial1 = 1:10, trial2 = 11:20)
)
```

- List names become interval names; unnamed groups are `interval_<n>`.
- Interval names are suffixed `<group>_<df>` (e.g. `trial1_A`).
- For *"mnirs_kinetics"* results analysed recursively, the source `nirs_channel` is prefixed to the analysed coefficient names (e.g. `smo2_slope`)
- Samples in no group are excluded from analysis (with a message). Samples in more than one group are allowed (with a warning).
- Row-grouped intervals no longer correspond to their `extract_intervals()` `interval_times` metadata, which is dropped, so `start_time` falls back to the first non-negative `time_channel` value unless supplied explicitly (optionally per-interval, keyed by group name).
- `zero_time = TRUE` rebases each group's `time_channel` to its first sample, so `start_time` then defaults to `0`.
- Per-interval arguments key by the group names (see below).

Per-channel and per-interval arguments:

Arguments apply globally to all `nirs_channels` by default. Arguments can instead be uniquely supplied per-channel as a named `list()` with names matching `nirs_channels`. For multi-interval input (a list of data frames or a grouped data frame), a named `list()` can also be keyed by interval name (the list names, group keys, or `interval_<n>`) to supply values per-interval, and each per-interval value may itself be a per-channel `list()`, e.g.

```
analyse_kinetics(
    data,
    nirs_channels = c(o2hb, hhb),
    method = "peak_slope",
    span = list(10, o2hb = 20),
    direction = list(
        interval_1 = list(hhb = "negative", "auto"),
```

```
        interval_2 = "positive"
    )
)
```

The same rules apply at both levels:

- A non-list value applies to every interval and channel (the *default* behaviour).
- A `list()` named by interval or `nirs_channels` applies to those values per-interval or per-channel.
- A single unnamed value in the list is the fallback applied to any unlisted intervals or channels (e.g. `span = list(10, o2hb = 20)` gives `o2hb` 20 and every other channel 10). If no unnamed fallback value in the list, unlisted intervals or channels fall back to the argument's default (i.e. `NULL`, or may fail with a warning).
- `list()` names matching neither interval names nor `nirs_channels` are warned about and ignored.

`start_time`, `direction`, and `end_window` are per-channel and per-interval capable, along with the `method`-specific arguments except `control`, which is always global. `fix` is itself a named `list()` of model parameters, so a per-channel or per-interval `fix` is supplied as a `list()` of `list()`s keyed by channel or interval name. A plain parameter list applies everywhere:

```
## fix `A` at 0 for every channel
fix = list(A = 0)

## fix `A` per-channel, leaving unspecified channels free
fix = list(o2hb = list(A = 0), hhb = list(A = 5, B = 20))

## fix `A` per-interval, optionally nested per-channel
fix = list(interval_1 = list(A = 0))
fix = list(interval_1 = list(o2hb = list(A = 0)))
```

Triple nested `list()`s is janky, but it works for now!

Limitation: `method` itself currently only accepts a single value applied globally to all intervals and `nirs_channels`. So analysing channels or intervals with entirely different kinetics models must be done with independent `analyse_kinetics()` calls, or other iterative solutions (e.g. `lapply()` or `purrr::map()`).

method = "response_time":

Aliases: `method = c("response time", "half recovery time", "half time", "HRT")`.

A non-parametric approach (estimated directly from the observed data without assuming a specific mathematical shape) to estimate the response time at which a signal reaches a specified fraction of its total response amplitude relative to the baseline. e.g. *half-response time* (`response_fraction = 0.5`) is the time from response onset to attain 50% of the total amplitude change and approximates the inflection point (`xmid` of a symmetrical sigmoid function).

`response_fraction = 0.632` approximates the time constant (`tau`; τ) parameter from a mono-exponential function, or the inflection point (`xmid`) of an asymmetrical left-Gompertz function. `response_fraction = 0.368` approximates `xmid` of a right-Gompertz function. This is a good fallback estimation method if parametric methods are not successfully fit.

The target response value is: `fitted = A + (B - A) * response_fraction`

Where A is the mean baseline value (time_channel <= start_time) and B is the first local extreme (peak or trough) value with no greater extreme values within end_window. response_value is the first observed sample where the signal is equal to or greater/lesser than the target response_fitted value. response_time is the elapsed time from start_time to response_value. See response_time() for the full algorithm and coefficients.

method = "peak_slope":

Aliases: method = c("peak slope", "slope", "lm").

A semi-parametric approach to estimate the maximum positive or negative local linear slope of a signal using rolling least-squares regression. The steepest local rate of change in NIRS signals can be interpreted as the moment of greatest mismatch between oxygen delivery and extraction. peak_slope_time is the time from response onset start_time to this moment of greatest mismatch.

The local window is defined by either width (number of samples) or span (in units of time_channel). See peak_slope() for window mechanics, partial-window behaviour, and the returned vector-level list.

method = "monoexponential":

Aliases: method = c("monoexp", "exponential", "exp", "tau", "MRT").

A parametric approach fitting a self-starting monoexponential function to the response curve using stats::nls() with SSmonoexponential() for either a 4-parameter (A, B, tau, TD) or 3-parameter (A, B, tau) model.

Model equations:

- 3-parameter: A + (B - A) * (1 - exp(-t / tau))
- 4-parameter: A + (B - A) * (1 - exp(-pmax(t - TD, 0) / tau))

TD is the *time delay* from start_time to the onset of the exponential response curve. tau is the *time constant* of the response. The *rate constant* k is the reciprocal (k = 1 / tau). The *mean response time* is the time sum MRT = TD + tau. See monoexponential() for the model family and SSmonoexponential() for self-start initialisation.

Any parameter may be held constant with fix, e.g. fix = list(A = 0). This excludes them from the fit optimisation procedure, and effectively reduces the function to a lower-parameter model. TD can only be fixed when use_TD = TRUE and disables the 3-parameter fallback. It is recommended to specify use_TD = FALSE rather than fix TD = 0.

method = "exponential_drift":

Aliases: method = c("exp_drift", "exp_linear", "monoexp_drift").

A parametric approach fitting a self-starting two-phase curve using stats::nls() with SSexponential_drift(). A *fast* monoexponential() primary response plus a *slow* linear secondary drift beginning near the primary asymptote.

Model equation:

A + (B - A) * (1 - exp(-pmax(t - TD, 0) / tau)) + slope_B * pmax(t - TD + tau * log(1 - drift_fraction), 0)

A, B, tau, TD, and the derived k, MRT, and HRT are as for *"monoexponential"*. slope_B is the linear drift rate dx/dt. The drift onset is not a free estimate. drift_fraction specifies the fraction ((0.5, 1)) of the primary response amplitude where the drift begins; TD - tau * log(1 - drift_fraction) (*default* 0.95; TD + 3 * tau).

The excursion point texc is where the drift rate overtakes the decaying primary rate, TD + tau * log(|B - A| / (|slope_B
floored at the drift onset, elapsed from start_time (the same frame as TD and MRT).

The drift component is kept only when the data support it. The model will fall back to *"mono-exponential"* when the fit fails or if the total drift amplitude is below twice the fit RMSE, with a warning recorded in warnings. The model column in coefficients names the final method for each row. A hidden argument model_fallback = FALSE will override the fallback process and retain the more complex model, or return an error.

Parameters may be held constant with fix, e.g. fix = list(A = 0), as above.

method = "biexponential":

Aliases: method = c("biexp", "double exponential").

A parametric approach fitting a self-starting two-phase biexponential excursion-recovery function to the response curve using stats::nls() with SSbiexponential(). A fast *primary* component driving the initial excursion, and a slow *secondary* component recovering the response toward a stable plateau.

Model equations:

- 5-parameter: A + (B - A) * (1 - exp(-t / tau)) + (B2 - B) * (1 - exp(-t / tau2))
- 6-parameter, where ts = pmax(t - TD, 0): A + (B - A) * (1 - exp(-ts / tau)) + (B2 - B) * (1 - exp(-ts / tau2))

A is the starting value. B & tau are the asymptote and time constant of the fast response. B2 & tau2 are the asymptote and time constant of the slower response plateau (typically tau2 >> tau).

Set use_TD = TRUE (*default*) to specify the time-delay parameter TD. The fast-phase mean response time MRT = TD + tau is reported as for *"monoexponential"*. See biexponential() for the model family and SSbiexponential() for self-start initialisation.

The two phases are fit sequentially.

- Stage 1 fits the fast phase as a *"monoexponential"* on the supplied end_window window, giving A, tau, and TD (if selected).
- Stage 2 fits the full model to the whole response with A, tau, and TD held within a tight range of their stage-1 values, and B, B2, tau2 free.

Secondary tau2 is floored above the primary tau, so the phases stay separated. tau2 is arbitrarily capped at ten times the fit window timespan, functionally implying the true asymptote is linear not exponential. end_window should be set to isolate the fast phase; by default resolves to 30 sec instead of Inf (recorded in channel_args).

The biexponential fit is kept only when the data support both phases. The model will fall back to *"exponential_drift"* when the fit fails (e.g. phases not separable), the fitted response is monotonic (no estimable excursion point texc), tau2 exceeds twice the fitted time span (a slow phase the record cannot tell from a linear drift), or the slow-phase amplitude |B2 - B| is below twice the fit RMSE.

The exponential-drift fit is in turn subject to its own fallback to *"monoexponential"* (see above). Each fallback is warned about and recorded in warnings. The model column in coefficients names the final method for each row. A hidden argument model_fallback = FALSE will override the fallback process and retain the more complex model, or return an error.

Parameters may be held constant with fix, e.g. fix = list(A = 0), as above.

method = "sigmoidal":

Aliases: method = c("logistic", "gompertz", "xmid").

A parametric approach fitting a self-starting 4-parameter sigmoidal function to the response curve using `stats::nls()` in one of three shapes.

Model equations (all 4-parameter):

- `shape = "symmetric"` (`SSlogistic()`): `A + (B - A) / (1 + exp(-4 * slope * (t - xmid) / (B - A)))`
- `shape = "gompertz"` (`SSgompertz()`): `A + (B - A) * exp(-exp(-k * (t - xmid)))` with `k = slope * e / (B - A)`. Early-acceleration; inflection height fixed at `A + (B - A) / e`; 36.8% of the amplitude.
- `shape = "gompertz_left"` (`SSgompertz_left()`): `A + (B - A) * (1 - exp(-exp(k * (t - xmid))))` with `k = slope * e / (B - A)`. Late-acceleration; inflection height fixed at `A + (B - A) * (1 - 1/e)`; 63.2% of the amplitude.

`xmid` is the time from `start_time` to the *inflection point*; the steepest point of the response. `slope` is the response rate `dx/dt` at the inflection.

A *"symmetric"* shape is the default when no obvious asymmetry is expected. *"gompertz"* (right-inflection) growth is appropriate for fast-onset, slow-tail responses. *"gompertz_left"* for slow-onset, fast-tail responses. See `logistic()`, `gompertz()`, and `gompertz_left()` for the model families and `SSlogistic()`, `SSgompertz()`, and `SSgompertz_left()` for self-start initialisations.

Parameters may be held constant with `fix`, e.g. `fix = list(A = 0)`, as above.

method = "sigmoidal_drift":

Aliases: `method = c("sigmoid_drift", "sig_drift", "sig-lin", "logistic_drift", "gompertz_drift")`.

A parametric approach fitting a self-starting two-phase curve using `stats::nls()` with `SSsigmoidal_drift()`. A *fast "sigmoidal"* primary response of the given `shape` plus a *slow* linear secondary drift beginning near the primary ending asymptote.

Model equation:

`S(t) + slope_B * pmax(t - onset, 0)`

`S(t)` and `A`, `B`, `xmid`, and `slope` are as for *"sigmoidal"*. `slope_B` is the linear drift rate `dx/dt` at the asymptote `B`. The drift is not a free estimate. `drift_fraction` specifies the fraction (`(0.5, 1)`) of the primary response amplitude where the drift begins (*default* `0.95`).

The excursion point `texc` is where the drift rate overtakes the decaying primary rate, `|S'(t)| = |slope_B|`, floored at the drift onset, elapsed from `start_time` (the same frame as `xmid`).

The drift component is kept only when the data support it. The model will fall back to *"sigmoidal"* when the fit fails or if the total drift amplitude is below twice the fit RMSE, with a warning recorded in `warnings`. The `model` column in `coefficients` names the final method for each row. A hidden argument `model_fallback = FALSE` will override the fallback process and retain the more complex model, or return an error.

Parameters may be held constant with `fix`, e.g. `fix = list(A = 0)`, as above.

Recursive analysis:

An *"mnirs_kinetics"* result may be passed back as `data` to analyse how coefficients change across intervals, e.g. `analyse_kinetics(result, nirs_channels = tau, time_channel = start_time, method = "peak_slope")`. `nirs_channels` and `time_channel` must name coefficient columns explicitly; no metadata defaults are applied.

Time-point coefficients (`response_time`, `peak_slope_time`, `TD`, `MRT`, `HRT`, `texc`, `xmid`) are elapsed from each interval's `start_time`. When one of these is given as `time_channel`, `start_time` is

added row-wise so the analysis runs on absolute time. `start_time` itself and duration coefficients (e.g. `tau`) are unchanged.

Coefficient rows from separate trials can be analysed separately with `group_intervals`, e.g. 20 occlusion slopes from two trials:

```
analyse_kinetics(
    result,
    nirs_channels = slope,
    time_channel = peak_slope_time,
    method = "monoexponential",
    group_intervals = list(trial1 = 1:10, trial2 = 11:20)
)
```

Value

A formatted table of results, with individual elements accessible as a structured list of class *"mnirs_kinetics"* containing:

`method`	The method used, e.g. `"response_time"`.
`model`	A named list of model objects (per interval, per `nirs_channel`). For `"peak_slope"`; each element is an lm object. For parametric models; an nls object. For `"response_time"`; `NULL`. Models are fitted on *time elapsed* from `start_time`, so predict expects a `time_channel` column in `newdata` with adjusted units. The offset for each interval can be retrieved from `coefficients$start_time`.
`coefficients`	A data frame of coefficients with one row per `nirs_channel` per interval, containing `interval`, `nirs_channels`, the resolved `start_time` (the fit onset from which time coefficients are elapsed), and method-specific parameters. For methods with fallback options; `model` names the final method for each row.
`data`	A list of the original input data frames augmented with a `*_fitted` column of fitted values for each processed `nirs_channel` (e.g. `smo2_fitted`).
`interval_times`	A data frame with one row per interval and numeric column `start_times` – the resolved response onset used for fitting (the supplied `start_time`, else the `extract_intervals()` metadata, else 0 or the first positive time value) – and `end_times` when any interval carries an end time from the metadata.
`diagnostics`	A data frame of model diagnostics (`n_obs`, `n_params`, `r2`, `adj_r2`, `rmse`, `cv_rmse`, `snr`, `aic`, `aicc`, `bic`) with one row per `nirs_channel` per interval. `n_params` counts the free parameters estimated by the solver, excluding any held by `fix`, so a reduced-parameter fallback fit is distinguishable from a full one. `n_obs` and `n_params` need to be considered carefully when comparing fit diagnostics between models.
`channel_args`	A data frame of the resolved arguments used for each `nirs_channel` with one row per `nirs_channel` per interval.
`warnings`	A data frame of warning and error messages captured during fitting, with columns `interval`, `nirs_channels` (empty for interval-level warnings), `type` (`"warning"` or `"error"`), and `message`; zero rows when none occurred. Conditions are captured regardless of `verbose`, which controls console output only.
`call`	The matched call.

See Also

extract_intervals(), response_time(), peak_slope(), monoexponential(), exponential_drift(), biexponential(), logistic(), gompertz(), gompertz_left(), sigmoidal_drift()

Examples

```
result <- read_mnirs(
    file_path = example_mnirs("train.red"),
    nirs_channels = c(
        smo2_left = "SmO2 unfiltered",
        smo2_right = "SmO2 unfiltered"
    ),
    time_channel = c(time = "Timestamp (seconds passed)"),
    zero_time = TRUE,
    verbose = FALSE
) |>
    resample_mnirs(method = "linear", verbose = FALSE) |>
    extract_intervals(
        group_intervals = "distinct",
        start = by_time(368, 1084),
        span = c(-20, 90),
        zero_time = TRUE,
        verbose = FALSE
    ) |>
    analyse_kinetics(
        nirs_channels = c(smo2_left, smo2_right),
        method = "peak_slope",
        span = 10,          ## 10-second rolling window
        direction = "auto", ## auto-detect slope direction
        verbose = FALSE
    )

## formatted table of results
result

## coefficients are accessible from the result list
result$coefficients

## along with diagnostics and other returned objects
result$diagnostics

## plot results
plot(result)
```

---

artinis_intervals.xlsx

10 Hz Artinis Oxysoft export recorded with Oxymon MKIII

Description

Exported from Artinis Oxysoft, recorded on Oxymon MKIII at 50 Hz and exported at 10 Hz. Containing two 5-minute cycling work intervals and an ischaemic occlusion, placed on the vastus lateralis muscle site.

Format

.xlsx file with header metadata and five columns and 20919 rows:

Column 1 Sample index (divide by sample rate for seconds).

Column 2 O2Hb: oxyhaemoglobin concentration change (μM).

Column 3 HHb: deoxyhaemoglobin concentration change (μM).

Column 4 Event marker (character).

Column 5 Unmarked event label (character).

Channels are detected automatically from the file legend, or can be specified explicitly for `read_mnirs()`:

- `nirs_channels = c(O2Hb = 2, HHb = 3)`
- `time_channel = c(sample = 1)`
- `event_channel = c(event = 4)`
- `interval_times = list(  ## two intervals, post-exercise occlusion start = c(158, 999, 1750), end`

Source

Artinis Medical Systems. Oxymon MKIII, exported via Oxysoft desktop software (https://artinis.com/)

See Also

`read_mnirs()`, `example_mnirs()`

Examples

```
example_mnirs("artinis_intervals")
```

`biexponential` *Biexponential function*

Description

Calculate a two-phase curve: a *fast* monoexponential primary response toward `B` and a *slow* monoexponential secondary response from `B` toward a stable plateau at `B2`, both clocked from the response onset and summed. Model family fit by `analyse_kinetics()` with `method = "biexponential"`, and by `stats::nls()` via the self-starting wrapper `SSbiexponential()`.

Usage

```
biexponential(t, A, B, tau, B2, tau2, TD = NULL)
```

Arguments

`t`	A numeric vector of the predictor variable (time).
`A`	A numeric parameter for the starting value of the response variable (the `t = 0` intercept).
`B`	A numeric parameter for the asymptote of the *fast* component; the value the fast response alone would approach.
`tau`	A numeric parameter for the *fast* time constant (τ_1), in units of the predictor variable `t`. Dominates the initial steep response.
`B2`	A numeric parameter for the asymptote of the *slow* component; the stable plateau the response recovers toward as `t` approaches infinity.
`tau2`	A numeric parameter for the *slow* time constant (τ_2), in units of the predictor variable `t`. Typically `tau2 >> tau`.
`TD`	A numeric parameter for the *time delay* before the onset of the response, in units of the predictor variable `t`. If `NULL` (*default*), a 5-parameter model without time delay is used.

Details

Model equations:

- 5-parameter: `A + (B - A) * (1 - exp(-t / tau)) + (B2 - B) * (1 - exp(-t / tau2))`
- 6-parameter, where `ts = pmax(t - TD, 0)`: `A + (B - A) * (1 - exp(-ts / tau)) + (B2 - B) * (1 - exp(-ts / tau2))`

`A`, `B`, and `B2` are all values on the response scale. The fast component is a `monoexponential()` response from `A` toward `B` with amplitude `B - A`; the slow component runs concurrently from the same onset with amplitude `B2 - B`. The curve starts at `A`, approaches `B2` as `t` grows, and is smooth throughout. If `B = B2`, the curve reduces to a `monoexponential()` with time constant `tau` and asymptote `B2`.

Excursion point:

The expected response is a *fast* excursion toward a minimum or maximum short of `B`, followed by a *slow* recovery back to a stable plateau at `B2`. The excursion point `texc` occurs where the two phase rates cancel: `texc = TD + log(ratio) / (1 / tau - 1 / tau2)` with `ratio = -(B - A) * tau2 / ((B2 - B) * tau)`, which exists only when the amplitudes oppose in sign and the fast phase dominates at the onset (`ratio > 1`). If `B` is between `A` and `B2`, the response is monotonic but still two-phase.

Value

A numeric vector of predicted values the same length as the predictor variable `t`.

See Also

`analyse_kinetics()`, `SSbiexponential()`, `monoexponential()`, `exponential_drift()`

Examples

```
## create a biexponential excursion-recovery curve with random noise
set.seed(1)
t <- 0:120
x <- biexponential(t, A = 70, B = 40, tau = 5, B2 = 60, tau2 = 40) +
    rnorm(length(t), 0, 0.8)
data <- data.frame(t, x)

## 5-parameter fit with the self-starting wrapper
model <- nls(
    x ~ SSbiexponential(t, A, B, tau, B2, tau2),
    data = data,
    algorithm = "port",
    lower = c(-Inf, -Inf, 0, -Inf, 0),
    control = nls.control(warnOnly = TRUE)
)
summary(model)

y <- predict(model, data)


    if (requireNamespace("ggplot2", quietly = TRUE)) {
        ggplot2::ggplot(data, ggplot2::aes(t, x)) +
            theme_mnirs() +
            ggplot2::geom_point() +
            ggplot2::geom_line(ggplot2::aes(y = y))
    }
```

`breaks_timespan` *Breaks for time span data*

Description

Pretty time span breaks for plotting in units of 5, 15, 30, 60 sec, etc. Modified from `scales::breaks_timespan()`.

Usage

```
breaks_timespan(unit = c("secs", "mins", "hours", "days", "weeks"), n = 5)
```

Arguments

`unit`	The time unit used to interpret numeric data input (*defaults* to *"secs"*).
`n`	Desired number of breaks. You may get slightly more or fewer breaks than requested.

Value

Returns a function for generating breaks.

Examples

```
x <- 0:120
y <- sin(2 * pi * x / 15) + rnorm(length(x), 0, 0.2)

ggplot2::ggplot(data.frame(x, y), ggplot2::aes(x, y)) +
    theme_mnirs() +
    ggplot2::scale_x_continuous(breaks = breaks_timespan()) +
    ggplot2::geom_line()
```

---

by_time *Specify interval boundaries by time, label, lap, or sample*

Description

Helper functions to define interval start or end boundaries for `extract_intervals()`.

Usage

```
by_time(...)

by_label(..., ignore_case = FALSE, fixed = FALSE)

by_lap(...)

by_sample(...)
```

Arguments

`...`	Specify start or end boundaries. `by_time(...)` Numeric time values in units of `time_channel`. `by_label(...)` Character strings to match in `event_channel`. Matched as regular expressions by default; see `ignore_case` and `fixed`. All matching occurrences are returned. `by_lap(...)` Integer lap numbers to match in `event_channel`. For `start`, resolves to the first sample of each lap. For `end`, resolves to the last sample. `by_sample(...)` Integer sample indices (row numbers).
`ignore_case`	For `by_label()`. If `TRUE`, match case-insensitive labels. Default `FALSE`.
`fixed`	For `by_label()`. If `TRUE`, treat labels as fixed strings rather than regular expressions. Useful when labels contain regex metacharacters (`.`, `*`, `(`, etc.). Default `FALSE`.

Details

These helpers can be used explicitly for arguments `start`/`end`, or raw values can be passed directly:

- Numeric -> `by_time()`
- Character -> `by_label()`,
- Explicit integer (e.g. `2L`) -> `by_lap()`.
- Use `by_sample()` explicitly for sample indices.

Multiple specification types can be combined for a single boundary with `list()` (e.g. `list(by_time(30), by_label("go"))`). Resolved boundary times are concatenated in the order supplied. Combined specifications must use the `by_` helpers directly: raw values are ignored with a warning.

Value

An object of class `"mnirs_interval"` for use with the `start` and `end` arguments of `extract_intervals()`.

Examples

```
## read example data
data <- read_mnirs(
    example_mnirs("train.red"),
    nirs_channels = c(
        smo2_left = "SmO2 unfiltered",
        smo2_right = "SmO2 unfiltered"
    ),
    time_channel = c(time = "Timestamp (seconds passed)"),
    event_channel = c(lap = "Lap/Event"),
    zero_time = TRUE,
    verbose = FALSE
)

## start and end by time
extract_intervals(data, start = by_time(66), end = by_time(357))

## start by lap
extract_intervals(data, start = by_lap(2, 4), span = 0)

## combine multiple specification types
extract_intervals(
    data,
    start = list(by_lap(2), by_time(400)),
    end = by_sample(1500)
)

## simulate event_channel with character label match
data$event <- NA_character_
data$event[c(1000, 1001)] <- c("start", "lap.1")
data <- create_mnirs_data(data, event_channel = "event")

## case-insensitive label match
extract_intervals(data, start = by_label("START", ignore_case = TRUE))
```

```
## literal-string label match (regex metacharacters treated as text)
extract_intervals(data, start = by_label("lap.1", fixed = TRUE))
```

`correct_blood_volume` *Correct for blood volume changes*

Description

Normalises mNIRS channels for the effects of blood volume changes, following the sample-wise iterative method of *Beever & Tripp et al, 2020*.

Usage

```
correct_blood_volume(
  data,
  oxy_channel = NULL,
  deoxy_channel = NULL,
  total_channel = NULL,
  verbose = TRUE
)
```

Arguments

`data`	A data frame of class *"mnirs"* containing time series data and metadata, a list of data frames, or a grouped data frame (see *Details*).
`oxy_channel`	A character vector naming the `oxy[haem]` (oxygenated haemoglobin and myoglobin; *O2Hb*) column(s) in `data`. Must match exactly.
`deoxy_channel`	A character vector naming the `deoxy[haem]` (deoxygenated haemoglobin and myoglobin; *HHb*) column(s) in `data`. Must match exactly.
`total_channel`	A character vector naming the `total[haem]` (total haemoglobin and myoglobin; *THb*; proxy for blood volume) column(s) in `data`. Must match exactly.
`verbose`	Logical. `TRUE` (*default*) will display, and `FALSE` will silence warnings and information messages helpful for troubleshooting. Global default can be set via `options(mnirs.verbose = FALSE)`.

Details

Specify NIRS component channels:

At least two of `oxy_channel`, `deoxy_channel`, and `total_channel` must be specified to calculate the blood volume correction factor. Best practice is to specify all existing channels in `data`. Missing channels are derived from the specified pair before the correction is applied.

- `total = oxy + deoxy`
- `oxy = total - deoxy`
- `deoxy = total - oxy`

Multiple channel pairs can be corrected in one call by passing equal-length vectors, with each element number forming a pair (e.g. `oxy_channel = c(o2hb_1, o2hb_2), deoxy_channel = c(hhb_1, hhb_2)`).

NOTE: the returned data frame will *ONLY* include corrected values for the specified channels. Non-specified channels will remain uncorrected and will therefore no longer be comparable to corrected channels. Best practice is to specify all existing channels in `data`.

Compute blood volume correction:

If any NIRS channels have negative values, all specified channels will be ensemble-shifted by a common offset so that all channels contain only positive values. Relative scaling across channels is preserved. This is modified from the method in *Beever & Tripp et al, 2020* to properly calculate `total[haem]` and the blood volume correction factor `beta` when there are negative NIRS values.

The correction factor `beta` is effectively the single-channel fractional (%) oxygen saturation used to normalise `oxy[haem]` and `deoxy[haem]` relative to an adjusted invariant `total[haem]`. This is computed as the cumulative sum of adjusted incremental differences:

$$\Delta\text{O2Hb}_c = \Delta\text{O2Hb} - \beta \cdot \Delta\text{THb}$$

$$\Delta\text{HHb}_c = \Delta\text{HHb} - (1 - \beta) \cdot \Delta\text{THb}$$

After correction, `total[haem]` is zero (blood volume changes are normalised).

Value

A tibble of class *"mnirs"* with blood volume-corrected channels written back to the specified columns, and with metadata available with `attributes()`. For list or grouped data frame input, returns a named list of *"mnirs"* tibbles, one per interval.

Data input formats

mnirs processing functions accept `data` in multiple formats:

- A **single *"mnirs"* data frame** is processed and returned directly.
- A **list of *"mnirs"* data frames**: each interval is processed separately and returned as a named list.
- A **grouped *"mnirs"* data frame**, e.g. with `dplyr::group_by()`: the data frame is split by grouping levels and each group is processed as a separate interval, returned as a named list.

References

Beever AT, Tripp TR, Zhang J, MacInnis MJ (2020) Nirs-Derived Skeletal Muscle Oxidative Capacity Is Correlated with Aerobic Fitness and Independent of Sex. J Appl Physiol (1985). doi:10.1152/japplphysiol.00017.2020

Ryan TE, Erickson ML, Brizendine JT, et al. (2012) Noninvasive Evaluation of Skeletal Muscle Mitochondrial Capacity with near-Infrared Spectroscopy: Correcting for Blood Volume Changes. J Appl Physiol (1985). doi:10.1152/japplphysiol.00319.2012

Examples

```
data <- read_mnirs(
    file_path = example_mnirs("artinis"),
    nirs_channels = c(o2hb = 2, hhb = 3),
    time_channel = c(sample = 1),
    verbose = FALSE,
)

plot(data)

result <- correct_blood_volume(
    data,
    oxy_channel = "o2hb",
    deoxy_channel = "hhb", ## thb will be derived from o2hb + hhb
)

plot(result)
```

---

create_mnirs_data *Create an* mnirs *data frame with metadata*

Description

Manually add class `"mnirs"` and metadata to an existing data frame.

Usage

```
create_mnirs_data(data, ...)
```

Arguments

`data`	A data frame with existing metadata (accessed with `attributes(data)`).
`...`	Additional arguments with metadata to add to the data frame. Can be either seperate named arguments or a list of named values. • nirs_device • nirs_channels • time_channel • event_channel • sample_rate • start_timestamp • interval_times • interval_span `nirs_channels`, `time_channel`, and `event_channel` accept named character vectors in the same form as `read_mnirs()`; `c(renamed = "original_name")`. Existing column names can be renamed, and the new names specified as `*_channel` in metadata.

Details

Intended primarily for internal use, but can be used to inject *mnirs* metadata into any data frame.

Value

A tibble of class `"mnirs"`. Metadata are stored as attributes and can be accessed with `attributes(data)`.

Examples

```
data <- data.frame(
    A = 1:3,
    B = seq(10, 30, 10),
    C = seq(11, 33, 11)
)

attributes(data)

## inject metadata
nirs_data <- create_mnirs_data(
    data,
    nirs_channels = c("B", "C"),
    time_channel = "A",
    sample_rate = 1
)

attributes(nirs_data)

## rename channels and update metadata
create_mnirs_data(
    nirs_data,
    nirs_channels = c(smo2 = "B", thb = "C"),
    time_channel = c(time = "A")
)
```

---

example_mnirs *Get path to* mnirs *example files*

Description

Get path to *mnirs* example files

Usage

```
example_mnirs(file = NULL)
```

Arguments

`file`	Name of file as character string. If `NULL`, returns a vector of all available file names.

Value

A file path character string for selected example files stored in this package.

Examples

```
## lists all files
example_mnirs()

## partial matching will error if matches multiple
try(example_mnirs("moxy"))

example_mnirs("moxy_ramp")
```

`exponential_drift` *Exponential-drift function*

Description

Calculate a two-phase curve: a *fast* `monoexponential()` primary response plus a *slow* linear secondary drift beginning near the primary asymptote. Model family fit by `analyse_kinetics()` with `method = "exponential_drift"`, and by `stats::nls()` via the self-starting wrapper `SSexponential_drift()`.

Usage

```
exponential_drift(t, A, B, tau, slope_B, drift_fraction, TD = NULL)
```

Arguments

`t`	A numeric vector of the predictor variable (time).
`A`	A numeric parameter for the starting baseline of the response variable.
`B`	A numeric parameter for the ending asymptote of the response variable.
`tau`	A numeric parameter for the *time constant* (τ) of the exponential response, in units of the predictor variable `t`.
`slope_B`	A numeric parameter for the linear drift rate `dx/dt` of the secondary phase, in response units per unit of the predictor variable `t`.
`drift_fraction`	A numeric fraction of the primary amplitude `B - A` in `(0.5, 1)` at which the linear drift begins, where the primary response reaches `A + drift_fraction * (B - A)`.
`TD`	A numeric parameter for the *time delay* before the onset of the exponential response, in units of the predictor variable `t`. If `NULL` (*default*), a 3-parameter model without time delay is used.

Details

Model equations:

- 5-parameter: `A + (B - A) * (1 - exp(-t / tau)) + slope_B * pmax(t + tau * log(1 - drift_fraction), 0)`
- 6-parameter: `A + (B - A) * (1 - exp(-pmax(t - TD, 0) / tau)) + slope_B * pmax(t - TD + tau * log(1 - drift_fraction), 0)`

`A`, `B`, `tau`, and `TD` are as for `monoexponential()`. The drift onset is not a free estimate: the secondary drift is exactly zero before `TD - tau * log(1 - drift_fraction)` (`TD = 0` when absent), and `drift_fraction = 0.95` places the onset at `TD + 3 * tau`.

The excursion point `texc` is where the drift rate overtakes the decaying primary rate, `TD + tau * log(|B - A| / (|slope_B` floored at the drift onset.

Value

A numeric vector of predicted values the same length as the predictor variable `t`.

See Also

`analyse_kinetics()`, `SSexponential_drift()`, `monoexponential()`, `biexponential()`, `sigmoidal_drift()`

Examples

```
## create an exponential curve with late linear drift and random noise
set.seed(13)
t <- 1:180
x <- exponential_drift(
    t, A = 10, B = 100, tau = 12,
    slope_B = -0.5, drift_fraction = 0.95, TD = 15
) + rnorm(length(t), 0, 2)
data <- data.frame(t, x)

## the drift onset fraction is held constant in the formula
model <- nls(
    x ~ SSexponential_drift(
        t, A, B, tau, slope_B, drift_fraction = 0.95, TD
    ),
    data = data,
    algorithm = "port",
    lower = c(-Inf, -Inf, 0, -Inf, 0),
    control = nls.control(warnOnly = TRUE)
)
summary(model)

y <- predict(model, data)


    if (requireNamespace("ggplot2", quietly = TRUE)) {
        ggplot2::ggplot(data, ggplot2::aes(t, x)) +
            theme_mnirs() +
            ggplot2::geom_point() +
```

```
            ggplot2::geom_line(ggplot2::aes(y = y))
    }
```

`extract_intervals` *Extract intervals from* mnirs *data*

Description

Extract intervals from *"mnirs"* time series data, specifying interval start and end boundaries by time value, event label, lap number, or sample index.

Usage

```
extract_intervals(
  data,
  nirs_channels = NULL,
  time_channel = NULL,
  event_channel = NULL,
  sample_rate = NULL,
  group_intervals = c("distinct", "ensemble"),
  group_channels = NULL,
  start = NULL,
  end = NULL,
  span = list(c(-60, 60)),
  zero_time = FALSE,
  verbose = TRUE,
  event_groups = deprecated()
)
```

Arguments

`data`	A data frame of class *"mnirs"* containing time series data and metadata, or a `list()` of such data frames.
`nirs_channels`	A character vector of mNIRS channel names to operate on. Names must match column names in `data` exactly. • If `NULL` (default), channels are retrieved from *"mnirs"* metadata. • **[Deprecated]** Passing a `list()` for per-group channel selection is deprecated; use `group_channels`.
`time_channel`	A character string naming the time or sample column. Must match a column name in `data` exactly. • If `NULL` (default), the `time_channel` metadata attribute of `data` is used.
`event_channel`	An *optional* character string giving the name of an event/lap column. The column may contain character event labels or integer lap numbers.

- Required when using by_label() or by_lap() for start or end.
- Retrieved from metadata if not defined explicitly.

sample_rate — An *optional* numeric sample rate (Hz) used to bin time values for ensemble-averaging. If NULL, will be estimated from time_channel (see *Details*).

group_intervals

Either a character string or a non-empty list() of non-empty integer-valued numeric vectors specifying how to group intervals (see *Details*). Custom indices must be between 1 and the number of detected intervals.

"distinct" The default. Extract each interval as an independent data frame.

"ensemble" Ensemble-average each specified nirs_channel across all detected intervals, returning a single data frame.

list(c(1, 2), c(3, 4)) Ensemble-average each specified nirs_channel within each group and return one data frame per group.

group_channels — A character vector or a list() of character vectors selecting which nirs_channels are ensemble-averaged within each interval group (see *Details*).

- If NULL (default), all nirs_channels are used for every group.
- Only relevant when group_intervals contains *"ensemble"*-averaged intervals; with group_intervals = "distinct" no channel processing occurs.

start — Specifies where intervals begin. Either raw values – numeric for time values, character for event labels, explicit integer (e.g. 2L) for lap numbers – or created with by_time(), by_label(), by_lap(), or by_sample(). Multiple specifications can be combined with list() (e.g. list(by_time(30), by_label("go"))); see *Details*.

end — Specifies where intervals end. Either raw values – numeric for time values, character for event labels, explicit integer (e.g. 2L) for lap numbers – or created with by_time(), by_label(), by_lap(), or by_sample(). Multiple specifications can be combined with list() (e.g. list(by_time(30), by_label("go"))); see *Details*.

span — A one- or two-element numeric vector expanding the time bounds around c(start, end), in units of time_channel; or a list() of such vectors. (*default* span = c(-60, 60). Applied additively to interval boundaries:

- When both start and end are specified: span[1] shifts start times, span[2] shifts end times.
- When only start or only end is specified: both span[1] and span[2] apply as a window around the event).
- A single *positive* value is recycled to shift the end times (e.g. span = 60 -> c(0, 60)).
- A single *negative* value is recycled to shift the start times (e.g. span = -60 -> c(-60, 0)).

zero_time — Logical. Default is FALSE. If TRUE, re-calculates numeric time_channel values to start from zero within each interval data frame.

verbose — Logical. TRUE (*default*) will display, and FALSE will silence warnings and information messages helpful for troubleshooting. Global default can be set via options(mnirs.verbose = FALSE).

`event_groups` **[Deprecated]** Renamed to `group_intervals` for naming consistency across the package.

Details

Interval specification:

Interval `start` and `end` boundaries are specified using helper functions, or by passing raw values directly:

`by_time()` Time values in units of `time_channel`.

`by_label()` Strings to match in `event_channel`. All matching occurrences are returned.

`by_lap()` Lap numbers to match in `event_channel`. Resolves to the first sample of each lap for `start`, and the last lap sample for `end`

`by_sample()` Integer sample indices (row numbers).

Raw values supplied to `start`/`end` are auto-coerced:

- Numeric -> `by_time()`
- Character -> `by_label()`,
- Explicit integer (e.g. `2L`) -> `by_lap()`.
- Use `by_sample()` explicitly for sample indices.

`start` and `end` can use different specification types (e.g., start by label, end by time). When lengths differ, the shorter is recycled.

Multiple specification types can be combined for a single boundary with `list()` (e.g. `start = list(by_time(30), by_label("go"))`). Resolved boundary times are concatenated in the order supplied. Combined specifications must use the `by_` helpers directly: raw values are ignored with a warning.

Time span window:

`span` additively expands the time span window around interval boundaries.

- A two-value vector expands the `start` and `end`, respectively: `span = c(-60, 60)` expands the `start` earlier by `60`, and the `end` later by `60`. For example, `start = by_time(30), end = by_time(60), span = c(-5` returns an interval of `[25, 70]`.
- A single numeric value is recycled according to the sign: `span = -60` becomes `c(-60, 0)` to expand the `start` earlier. `span = 60` becomes `c(0, 60)` to expand the `end` later.
- If only `start` is specified alone, both span values expand the single boundary window: `start = by_time(30), span = c(-5, 60)` returns `[25, 90]`.

Per-group channel selection with group_channels:

When `group_intervals = "ensemble"` or a list of numeric grouped intervals, `group_channels` can be specified as a list of column names to override which channels are ensemble-averaged within each group. For example, to exclude a channel from one interval:

```
group_channels = list(
  c(A, B, C),
  c(A, C) ## channel "B" data are excluded from the second interval
)
```

If all grouped intervals include all `nirs_channels`, `group_channels` can be left as `NULL` (the default) and all channels are ensemble-averaged within every group.

Grouping intervals:

`group_intervals` controls whether extracted intervals are returned as distinct data frames or ensemble-averaged.

`"distinct"` The default. Extract each interval and return a list of independent data frames.

`"ensemble"` Ensemble-average each specified `nirs_channel` across all detected intervals and return a one-item list with a single data frame.

`list(c(1, 2), c(3, 4))` Ensemble-average each specified `nirs_channel` within each group and return a list with one data frame for each group. Any intervals detected but not specified in `group_intervals` are returned as distinct.

`group_intervals` lists can be named (e.g. `list(low = c(1, 2), high = c(3, 4))`) and will pass those names to the returned list of data frames.

When `group_intervals` is a list of numeric interval numbers, list items in `group_channels` and `span` are recycled to the number of groups. If lists are only partially specified, the final item is recycled forward as needed. Extra items are ignored.

Value

A named `list()` of tibbles of class *"mnirs"*, each with metadata available via `attributes()`. When `data` is a list of data frames, results are flattened into a single-layer list with interval names indicating nested number of data frame and interval as `interval_<df>.<interval>`. Other interval names (e.g. `"ensemble"` or custom group names) are suffixed as `<name>_<df>`.

Examples

```
## read example data
data <- read_mnirs(
    example_mnirs("train.red"),
    nirs_channels = c(
        smo2_left = "SmO2 unfiltered",
        smo2_right = "SmO2 unfiltered"
    ),
    time_channel = c(time = "Timestamp (seconds passed)"),
    zero_time = TRUE,
    verbose = FALSE
) |>
    ## avoid issues ensemble-averaging irregular samples
    resample_mnirs(method = "linear", verbose = FALSE)

## ensemble-average across multiple intervals
interval_list <- extract_intervals(
    data,                         ## channels recycled to all intervals by default
    nirs_channels = c(smo2_left, smo2_right),
    group_intervals = "ensemble", ## ensemble-average across two intervals
    start = by_time(368, 1084),   ## manually identified interval start times
    span = c(-20, 90),           ## include the last 180-sec of each interval (recycled)
    zero_time = TRUE              ## re-calculate common time to start from `0`
)

interval_list[[1L]]
```

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  plot(interval_list, time_labels = TRUE) +
    ggplot2::geom_vline(xintercept = 0, linetype = "dotted")
}
```

`filter_butterworth` *Apply a Butterworth digital filter*

Description

Apply a Butterworth digital filter to vector data with `signal::butter()` and `signal::filtfilt()` which handles 'edges' better at the start and end of the data.

Usage

```
filter_butterworth(
  x,
  order = 2L,
  W,
  type = c("low", "high", "stop", "pass"),
  edges = c("rev", "rep1", "none"),
  na.rm = FALSE,
  ...
)

filter_butter(
  x,
  order = 2L,
  W,
  type = c("low", "high", "stop", "pass"),
  edges = c("rev", "rep1", "none"),
  na.rm = FALSE,
  ...
)
```

Arguments

`x`	A numeric vector.
`order`	An integer defining the filter order (*default* `order = 2`).
`W`	A one- or two-element numeric vector within `[0, 1]` defining the filter cutoff frequency(ies) as a fraction of the Nyquist frequency (see *Details*).
`type`	A character string indicating the digital filter type (see *Details*).
	`"low"` For a *low-pass* filter (the *default*).

	`"high"` For a *high-pass* filter.
	`"stop"` For a *stop-band* (band-reject) filter.
	`"pass"` For a *pass-band* filter.
`edges`	A character string indicating edge detection padding for `x`.
	`"rev"` Will pad `x` with the preceding 5% data in reverse sequence (*the default*).
	`"rep1"` Will pad `x` by repeating the last preceding value.
	`"none"` Will return the unpadded `signal::filtfilt()` output.
`na.rm`	Logical; default is `FALSE`, propagates any `NA`s to the returned vector. If `TRUE`, ignores `NA`s and processes available valid samples within the local window. May return errors or warnings. (see *Details*).
`...`	Additional arguments passed to the underlying method function. See *Details*.

Details

Applies a centred (two-pass symmetrical) Butterworth digital filter from `signal::butter()` and `signal::filtfilt()`.

Filter `type` defines how the desired signal frequencies are either passed or rejected from the output signal. *Low-pass* and *high-pass* filters allow only frequencies *lower* or *higher* than the cutoff frequency `W` to be passed through as the output signal, respectively. *Stop-band* defines a critical range of frequencies which are rejected from the output signal. *Pass-band* defines a critical range of frequencies which are passed through as the output signal.

The filter order (number of passes) is defined by `order`, typically in the range `order = [1, 10]`. Higher filter order tends to capture more rapid changes in amplitude, but also causes more distortion around those change points in the signal. General advice is to use the lowest filter order which sufficiently captures the desired rapid responses in the data.

The critical (cutoff) frequency is defined by `W`, a numeric value for *low-pass* and *high-pass* filters, or a two-element vector `c(low, high)` defining the lower and upper bands for *stop-band* and *pass-band* filters. `W` represents the desired fractional cutoff frequency in the range `W = [0, 1]`, where `1` is the Nyquist frequency, i.e., half the sample rate of the data in Hz.

Missing values (`NA`) in `x` will cause an error unless `na.rm = TRUE`. Then `NA`s will be ignored and passed through to the returned vector.

Value

A numeric vector the same length as `x`.

See Also

`signal::filtfilt()`, `signal::butter()`

Examples

```
set.seed(13)
sin <- sin(2 * pi * 1:150 / 50) * 20 + 40
noise <- rnorm(150, mean = 0, sd = 6)
noisy_sin <- sin + noise
without_edge_detection <- filter_butterworth(
```

```
    x = noisy_sin,
    order = 2,
    W = 0.1,
    edges = "none"
)
with_edge_detection <- filter_butterworth(
    x = noisy_sin,
    order = 2,
    W = 0.1,
    edges = "rep1"
)


ggplot2::ggplot(data.frame(), ggplot2::aes(x = seq_along(noise))) +
    theme_mnirs() +
    scale_colour_mnirs(name = NULL) +
    ggplot2::geom_line(ggplot2::aes(y = noisy_sin)) +
    ggplot2::geom_line(
        ggplot2::aes(y = without_edge_detection, colour = "without")
    ) +
    ggplot2::geom_line(
        ggplot2::aes(y = with_edge_detection, colour = "with")
    )
```

`filter_mnirs` *Filter a data frame*

Description

Apply digital filtering/smoothing to numeric vector data within a data frame using either:

1. A cubic smoothing spline.
2. A Butterworth digital filter.
3. A simple moving average.

Note the `method`-specific arguments below.

Usage

```
filter_mnirs(
  data,
  nirs_channels = NULL,
  time_channel = NULL,
  method = c("smooth_spline", "butterworth", "moving_average"),
  na.rm = FALSE,
  verbose = TRUE,
  ...,
  spar = NULL,
```

```
  order = 2L,
  W = NULL,
  fc = NULL,
  sample_rate = NULL,
  type = c("low", "high", "stop", "pass"),
  edges = c("rev", "rep1", "none"),
  width = NULL,
  span = NULL,
  partial = FALSE
)
```

Arguments

`data`	A data frame of class *"mnirs"* containing time series data and metadata, a list of data frames, or a grouped data frame (see *Details*).
`nirs_channels`	A character vector giving the names of mNIRS columns to operate on. Must match column names in `data` exactly. • If `NULL` (default), the `nirs_channels` metadata attribute of `data` is used.
`time_channel`	A character string naming the time or sample column. Must match a column name in `data` exactly. • If `NULL` (default), the `time_channel` metadata attribute of `data` is used.
`method`	A character string indicating how to filter the data. Additional arguments must be specified for each method. See *Details*. `"smooth_spline"` Fits a cubic smoothing spline. Additional arguments: `spar`. `"butterworth"` Uses a centred Butterworth digital filter. Additional arguments: `order`, `W` or `fc`, `sample_rate`, `type`, `edges`. See `filter_butterworth()`. `"moving_average"` Uses a centred moving average filter. Additional arguments: `width` or `span`, `partial`. See `filter_moving_average()`.
`na.rm`	Logical; default is `FALSE`, propagates any `NA`s to the returned vector. If `TRUE`, ignores `NA`s and processes available valid samples within the local window. May return errors or warnings. (see *Details*).
`verbose`	Logical. `TRUE` (*default*) will display, and `FALSE` will silence warnings and information messages helpful for troubleshooting. Global default can be set via `options(mnirs.verbose = FALSE)`.
`...`	Additional arguments passed to the underlying method function. See *Details*.
`spar`	**smooth_spline**: A numeric smoothing parameter passed to `stats::smooth.spline()`. If `NULL` (*default*), automatically determined via penalised log likelihood.
`order`	**butterworth**: An integer defining the filter order (*default* `order = 2`).
`W`	**butterworth**: A one- or two-element numeric vector within `[0, 1]` defining the filter cutoff frequency(ies) as a fraction of the Nyquist frequency (see *Details*). One of either `W` or `fc` must be specified.
`fc`	**butterworth**: A one- or two-element numeric vector defining the filter absolute cutoff frequency in Hz. Used with `sample_rate` to compute `W`. One of either `W` or `fc` must be specified.

sample_rate	**butterworth**: A numeric sample rate in Hz. Will be taken from metadata or estimated from time_channel if not defined.
type	**butterworth**: A character string specifying filter type, one of: c("low", "high", "stop", "pass") ("low" is the *default*).
edges	**butterworth**: A character string specifying the edge padding, one of: c("rev", "rep1", "none") ("rev" is the *default*). See filter_butterworth().
width	**moving_average**: An integer number of samples within the local window. One of either width or span must be specified.
span	**moving_average**: A numeric time duration in units of time_channel within the local window. One of either width or span must be specified.
partial	**moving_average**: Logical; default is FALSE, only returns values where a full window of valid (non-NA) samples are available. If TRUE, ignores NA and processes available valid samples (see *Details*).

Details

method = "smooth_spline":

Aliases: method = c("smooth spline", "spline")

Applies a non-parametric cubic smoothing spline from stats::smooth.spline(). Smoothing is defined by the parameter spar, which can be left as NULL and automatically determined via penalised log likelihood. This usually works well for responses occurring on the order of minutes or longer. spar can be specified typically, but not necessarily, in the range spar = [0, 1].

method = "butterworth":

Aliases: method = c("butter")

Applies a centred (two-pass symmetrical) Butterworth digital filter from signal::butter() and signal::filtfilt().

Filter type defines how the desired signal frequencies are either passed or rejected from the output signal. *Low-pass* and *high-pass* filters allow only frequencies *lower* or *higher* than the cutoff frequency, respectively to be passed through to the output signal. *Stop-band* defines a critical range of frequencies which are rejected from the output signal. *Pass-band* defines a critical range of frequencies which are passed through as the output signal.

The filter order (number of passes) is defined by order, typically in the range order = [1, 10]. Higher filter order tends to capture more rapid changes in amplitude, but also causes more distortion around those change points in the signal. General advice is to use the lowest filter order which sufficiently captures the desired rapid responses in the data.

The critical (cutoff) frequency can be defined by W, a numeric value for *low-pass* and *high-pass* filters, or a two-element vector c(low, high) defining the lower and upper bands for *stop-band* and *pass-band* filters. W represents the desired fractional cutoff frequency in the range W = [0, 1], where 1 is the Nyquist frequency, i.e., half the sample_rate of the data in Hz.

Alternatively, the cutoff frequency can be defined by fc and sample_rate together. fc represents the desired cutoff frequency directly in Hz, and sample_rate is the sample rate of the recorded data in Hz. Where W = fc / (sample_rate / 2).

Only one of either W or fc should be defined. If both are defined, W will be preferred over fc.

method = "moving_average":

Aliases: `method = c("moving average", "ma")`

Applies a centred (symmetrical) moving average filter in a local window, defined by either `width` as the number of samples around `idx` between `[idx - floor(width/2), idx + floor(width/2)]`. Or by `span` as the timespan in units of `time_channel` between `[t - span/2, t + span/2]`.

Missing values:

Missing values (`NA`) in `nirs_channels` will cause an error for `method = "smooth_spline"` or `"butterworth"`, unless `na.rm = TRUE`. Then `NA`s will be ignored and passed through to the returned data.

For `method = "moving_average"`, `na.rm` controls whether `NA`s within each local window are either propagated to the returned vector when `na.rm = FALSE` (the default), or ignored before processing if `na.rm = TRUE`.

Value

A tibble of class *"mnirs"* with metadata available with `attributes()`. For list or grouped data frame input, returns a named list of *"mnirs"* tibbles, one per interval.

Data input formats

mnirs processing functions accept `data` in multiple formats:

- A **single *"mnirs"* data frame** is processed and returned directly.
- A **list of *"mnirs"* data frames**: each interval is processed separately and returned as a named list.
- A **grouped *"mnirs"* data frame**, e.g. with `dplyr::group_by()`: the data frame is split by grouping levels and each group is processed as a separate interval, returned as a named list.

Per-channel arguments

Arguments apply globally to all `nirs_channels` by default. Relevant arguments can instead be supplied uniquely per-channel as a named `list()`, with names matching `nirs_channels`, e.g.

```
replace_mnirs(
    data,
    nirs_channels = c(hhb, smo2),
    invalid_values = list(hhb = -1, smo2 = c(0, 100)),
    invalid_above = list(hhb = 10),
    span = list(3, hhb = 5)
)
```

- A non-list value applies to every channel (the *default* behaviour).
- A `list()` named by `nirs_channels` applies per-channel values.
- A single unnamed value in the list will be applied to unlisted channels (e.g. `span = list(3, hhb = 5)` gives `hhb` 5 and every other channel 3). If no unnamed fallback value in the list, channels not named in the list will be returned un-processed (e.g. `span = list(hhb = 5)` will only process `hhb`).
- `list()` names not matching `nirs_channels` are warned about and ignored.

Examples

```
## read example data and clean for outliers
data <- read_mnirs(
    file_path = example_mnirs("moxy_ramp"),
    nirs_channels = c(smo2 = "SmO2 Live"),
    time_channel = c(time = "hh:mm:ss"),
    verbose = FALSE
) |>
    replace_mnirs(
        invalid_values = c(0, 100),
        outlier_cutoff = 3,
        width = 7,
        verbose = FALSE
    )

data

data_filtered <- filter_mnirs(
    data,                   ## blank channels will be retrieved from metadata
    method = "butterworth", ## Butterworth digital filter is a common choice
    order = 2,              ## filter order number
    W = 0.02,               ## filter fractional critical frequency `[0, 1]`
    type = "low",           ## specify a "low-pass" filter
    na.rm = TRUE            ## explicitly ignore NAs
)

## note the smoothed `smo2` values
data_filtered


    if (requireNamespace("ggplot2", quietly = TRUE)) {
        ## plot filtered data on top of raw to compare
        plot(data_filtered, time_labels = TRUE) +
            ggplot2::geom_line(
                data = data,
                ggplot2::aes(y = smo2, colour = "smo2"), alpha = 0.4
            )
    }
```

`filter_moving_average` *Apply a moving average filter*

Description

Apply a simple moving average smoothing filter to vector data. `filter_ma()` is an alias of `filter_moving_average()`.

Usage

```
filter_moving_average(
  x,
  t = seq_along(x),
  width = NULL,
  span = NULL,
  partial = FALSE,
  na.rm = FALSE,
  verbose = TRUE,
  ...
)

filter_ma(
  x,
  t = seq_along(x),
  width = NULL,
  span = NULL,
  partial = FALSE,
  na.rm = FALSE,
  verbose = TRUE,
  ...
)
```

Arguments

x	A numeric vector of the response variable.
t	An *optional* numeric vector of the predictor variable (e.g. time). Default is `seq_along(x)`.
width	An integer defining the local window in number of samples centred on `idx`, between `[idx - floor(width/2), idx + floor(width/2)]`.
span	A numeric value defining the local window time span around `idx` in units of `time_channel` or `t`, between `[t - span/2, t + span/2]`.
partial	Logical; default is `FALSE`, only returns values where a full window of valid (non-`NA`) samples are available. If `TRUE`, ignores `NA` and processes available valid samples (see *Details*).
na.rm	Logical; default is `FALSE`, propagates any `NA`s to the returned vector. If `TRUE`, ignores `NA`s and processes available valid samples within the local window. May return errors or warnings. (see *Details*).
verbose	Logical. `TRUE` (*default*) will display, and `FALSE` will silence warnings and information messages helpful for troubleshooting. Global default can be set via `options(mnirs.verbose = FALSE)`.
...	Additional arguments.

Details

Rolling window:

Applies a centred (symmetrical) moving average filter in a local window, defined by either `width` as the number of samples around `idx` between `[idx - floor(width/2), idx + floor(width/2)]`. Or by `span` as the timespan in units of `time_channel` between `[t - span/2, t + span/2]`.

Partial windows:

The default `partial = FALSE` requires a complete number of samples specified by `width` or `span` (estimated from the sample rate of `t` when `span` is used). `NA` is returned if fewer samples are present in the local window.

Setting `partial = TRUE` allows computation with only a single valid sample, such as at edge conditions. But these values will be more sensitive to noise and should be used with caution.

Missing values:

`na.rm` controls whether missing values (`NA`s) within each local window are either propagated to the returned vector when `na.rm = FALSE` (the default), or ignored before processing if `na.rm = TRUE`.

Value

A numeric vector the same length as `x`.

Examples

```
x <- c(1, 3, 2, 5, 4, 6, 5, 7)
t <- c(0, 1, 2, 4, 5, 6, 7, 10)  ## irregular time with gaps

## width: centred window of 3 samples
filter_moving_average(x, width = 3)

## partial = TRUE fills edge values with a narrower window
filter_moving_average(x, width = 3, partial = TRUE)

## span: centred window of 2 time-units (accounts for irregular sampling)
filter_moving_average(x, t, span = 2)

## na.rm = FALSE (default): any NA in the window propagates to the result
x_na <- c(1, NA, 3, 4, 5, NA, 7, 8)
filter_moving_average(x_na, width = 3, na.rm = FALSE)

## na.rm = TRUE: skip NAs and return the local mean of local valid values
filter_moving_average(x_na, width = 3, partial = TRUE, na.rm = TRUE)
```

---

format_hmmss *Format time span data as h:mm:ss*

Description

Convert numeric time span data to `h:mm:ss` format for pretty plotting. Inspired by `ggplot2::scale_x_time()`.

Usage

```
format_hmmss(x)
```

Arguments

x	A numeric vector.

Details

If all values are less than 3600 (1 hour), then format is returned as `mm:ss`. If any value is greater than 3600, format is returned as `h:mm:ss` with leading zeroes.

Value

A character vector the same length as `x`.

Examples

```
x <- 0:120
y <- sin(2 * pi * x / 15) + rnorm(length(x), 0, 0.2)

ggplot2::ggplot(data.frame(x, y), ggplot2::aes(x, y)) +
    theme_mnirs() +
    ggplot2::scale_x_continuous(
        breaks = breaks_timespan(),
        labels = format_hmmss
    ) +
    ggplot2::geom_line()
```

---

gompertz *Gompertz growth functions*

Description

Calculate 4-parameter Gompertz (asymmetric sigmoidal) curves. Model families fit by `analyse_kinetics()` with `method = "sigmoidal"` and `shape = "gompertz"` or `"gompertz_left"`, and by `stats::nls()` via the self-starting wrappers `SSgompertz()` and `SSgompertz_left()`.

Usage

```
gompertz(t, A, B, xmid, slope)

gompertz_left(t, A, B, xmid, slope)
```

Arguments

t	A numeric vector of the predictor variable (time).
A	A numeric parameter for the starting asymptote of the response variable.
B	A numeric parameter for the ending asymptote of the response variable.
xmid	A numeric parameter for the time at the *inflection point* (the steepest point) of the curve, in units of the predictor variable t.
slope	A numeric parameter for the response rate dx/dt at the inflection xmid.

Details

gompertz() (right-Gompertz) is asymmetric with the inflection point xmid closer to the starting asymptote A: early acceleration away from A, and a slow approach to the ending asymptote B. Appropriate for fast-onset, slow-tail responses.

gompertz_left() (left-Gompertz) has the inflection point closer to the ending asymptote B: slow departure from A, and late acceleration toward B. Appropriate for slow-onset, fast-tail responses.

Model equations:

Both forms are re-parameterised so xmid is the time at inflection and slope is the response rate dx/dt at the inflection, with k = slope * e / (B - A).

- gompertz(): A + (B - A) * exp(-exp(-k * (t - xmid))). Inflection height fixed at A + (B - A) / e; 36.8% of the amplitude.
- gompertz_left(): A + (B - A) * (1 - exp(-exp(k * (t - xmid)))). Inflection height fixed at A + (B - A) * (1 - 1/e); 63.2% of the amplitude.

Value

A numeric vector of predicted values the same length as the predictor variable t.

See Also

analyse_kinetics(), SSgompertz(), SSgompertz_left(), logistic(), sigmoidal_drift()

Examples

```
## create a Gompertz curve with random noise
set.seed(15)
t <- 1:60
x <- gompertz(t, A = 10, B = 100, xmid = 30, slope = 4) +
    rnorm(length(t), 0, 2)
data <- data.frame(t, x)

## fit with the self-starting wrapper
model <- nls(x ~ SSgompertz(t, A, B, xmid, slope), data = data)
summary(model)

y <- predict(model, data)
```

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
    ggplot2::ggplot(data, ggplot2::aes(t, x)) +
        theme_mnirs() +
        ggplot2::geom_point() +
        ggplot2::geom_line(ggplot2::aes(y = y))
}
```

`logistic` *Generalised logistic function*

Description

Calculate a 4- or 5-parameter logistic (sigmoidal) curve. The 4-parameter symmetric form is fit by `analyse_kinetics()` with `method = "sigmoidal"` and `shape = "symmetric"` (*default*), and by `stats::nls()` via the self-starting wrapper `SSlogistic()`.

Usage

```
logistic(t, A, B, xmid, slope, asym = NULL)
```

Arguments

`t`	A numeric vector of the predictor variable (time).
`A`	A numeric parameter for the starting asymptote of the response variable.
`B`	A numeric parameter for the ending asymptote of the response variable.
`xmid`	A numeric parameter for the time at the *inflection point* (the steepest point) of the curve, in units of the predictor variable `t`.
`slope`	A numeric parameter for the response rate `dx/dt` at the inflection `xmid`.
`asym`	A numeric parameter for the asymmetry index of the curve; the fraction of the amplitude `(y(xmid) - A) / (B - A)` at which the inflection `xmid` occurs, in `(0, 1)`. `asym = 0.5` is symmetric and equivalent to the 4-parameter form. If `NULL` (*default*), a symmetric 4-parameter model is used.

Details

The 5-parameter Richards form is exported for advanced use directly with `stats::nls()` but is not used by `analyse_kinetics()` due to convergence instability. For asymmetric responses, prefer `gompertz()` / `gompertz_left()`, which are more stable.

Model equations:

Both forms are re-parameterised from the Richards generalised logistic model so `xmid` is the time at inflection and `slope` is the response rate `dx/dt` at the inflection.

- 4-parameter (symmetric): `A + (B - A) / (1 + exp(-4 * slope * (t - xmid) / (B - A)))`

- 5-parameter (asymmetric): `A + (B - A) / (1 + exp(-k * (t - xmid)))^(1 / v)` with `v = -log(2) / log(asym)` and `k = 2 * slope * v / ((B - A) * asym)`.

The inflection is at `t = xmid` with `dx/dt = slope` and `y(xmid) = A + (B - A) * asym` for any `asym` in `(0, 1)`:

- `asym = 0.5` (`v = 1`) collapses to the 4-parameter form.
- `asym -> 0` gives an early-acceleration curve (inflection near `A`).
- `asym -> 1` gives a late-acceleration curve (inflection near `B`).
- `asym = 0.368` (`1/e`) approximates a right-inflection `gompertz()` curve.
- `asym = 0.632` (`1 - 1/e`) approximates a left-inflection `gompertz_left()` curve.

Value

A numeric vector of predicted values the same length as the predictor variable `t`.

See Also

`analyse_kinetics()`, `SSlogistic()`, `gompertz()`, `gompertz_left()`, `sigmoidal_drift()`, `monoexponential()`

Examples

```
## create an asymmetric logistic curve with random noise
set.seed(15)
t <- 1:60
x <- logistic(t, A = 10, B = 100, xmid = 30, slope = 4, asym = 0.3) +
    rnorm(length(t), 0, 2)
data <- data.frame(t, x)

## 5-parameter fit with the self-starting wrapper
model <- nls(x ~ SSlogistic(t, A, B, xmid, slope, asym), data = data)
summary(model)

y <- predict(model, data)


    if (requireNamespace("ggplot2", quietly = TRUE)) {
        ggplot2::ggplot(data, ggplot2::aes(t, x)) +
            theme_mnirs() +
            ggplot2::geom_point() +
            ggplot2::geom_line(ggplot2::aes(y = y))
    }
```

`monoexponential` *Monoexponential function*

Description

Calculate a 3- or 4-parameter monoexponential curve. Model family fit by `analyse_kinetics()` with `method = "monoexponential"`, and by `stats::nls()` via the self-starting wrapper `SSmonoexponential()`.

Usage

```
monoexponential(t, A, B, tau, TD = NULL)
```

Arguments

`t`	A numeric vector of the predictor variable (time).
`A`	A numeric parameter for the starting baseline of the response variable.
`B`	A numeric parameter for the ending asymptote of the response variable.
`tau`	A numeric parameter for the *time constant* (τ) of the exponential response, in units of the predictor variable `t`.
`TD`	A numeric parameter for the *time delay* before the onset of the exponential response, in units of the predictor variable `t`. If `NULL` (*default*), a 3-parameter model without time delay is used.

Details

Model equations:

- 3-parameter: `A + (B - A) * (1 - exp(-t / tau))`
- 4-parameter: `A + (B - A) * (1 - exp(-pmax(t - TD, 0) / tau))`

Clamping the shifted time at zero holds the curve flat at the baseline `A` until the response onset at `t = TD`.

Derived quantities:

The *rate constant* `k` is the reciprocal of `tau` (`k = 1 / tau`) in reciprocal units of `t` (e.g. `sec^-1`). The *mean response time* is the time sum `MRT = TD + tau`, and the *half-response time* is `HRT = TD + tau * log(2)`.

Value

A numeric vector of predicted values the same length as the predictor variable `t`.

See Also

`analyse_kinetics()`, `SSmonoexponential()`, `exponential_drift()`, `biexponential()`, `response_time()`, `peak_slope()`

Examples

```
## create an exponential curve with random noise
set.seed(13)
t <- 1:60
x <- monoexponential(t, A = 10, B = 100, tau = 8, TD = 15) +
    rnorm(length(t), 0, 3)
data <- data.frame(t, x)

## 4-parameter fit with the self-starting wrapper
model <- nls(x ~ SSmonoexponential(t, A, B, tau, TD), data = data)
summary(model)

y <- predict(model, data)


    if (requireNamespace("ggplot2", quietly = TRUE)) {
        ggplot2::ggplot(data, ggplot2::aes(t, x)) +
            theme_mnirs() +
            ggplot2::geom_point() +
            ggplot2::geom_line(ggplot2::aes(y = y))
    }
```

`moxy_intervals.csv` *0.5 Hz Moxy onboard export*

Description

Exported from Moxy onboard recording at 0.5 Hz no smoothing. Containing four 4-minute cycling work intervals, placed on the vastus lateralis muscle site.

Format

.csv file with seven columns and 936 rows:

mm-dd Recording date (dd-MMM format).

hh:mm:ss Recording time of day (hh:mm:ss format).

SmO2 Live Muscle oxygen saturation, raw signal (%).

SmO2 Averaged Muscle oxygen saturation, rolling average (%).

THb Total haemoglobin (arbitrary units).

Lap Lap marker (integer). Not typically in use.

Session Ct Session count of recordings.

Channel mapping for `read_mnirs()`:

- `nirs_channels = c("SmO2 Live", "SmO2 Averaged", "THb")`
- `time_channel = c("hh:mm:ss")`
- `interval_times = list( start = c(124, 486, 848, 1210), end = c(364, 726, 1088, 1450))`

Source

Moxy Monitor (Fortiori Design LLC), exported via Moxy Portal App. (https://www.moxymonitor.com/)

See Also

`read_mnirs()`, `example_mnirs()`

Examples

```
example_mnirs("moxy_intervals")
```

`moxy_ramp.xlsx` *2 Hz PerfPro export of Moxy data*

Description

Exported from PerfPro Studio software, recorded at 0.5 Hz no smoothing and exported at 2 Hz. Containing a ramp incremental cycling protocol, placed on bilateral vastus lateralis muscle sites. Intentional data errors (outliers, invalid values, and missing `NA` values) have been introduced to demonstrate *mnirs* cleaning functions.

Format

.xlsx file with five columns and 2202 rows:

mm-dd Recording date (dd-MMM format).

hh:mm:ss Time of day (hh:mm:ss format).

SmO2 Live Muscle oxygen saturation, left leg (%). Contains simulated erroneous and missing samples.

SmO2 Live(2) Muscle oxygen saturation, right leg (%).

Lap Lap marker (integer).

Channel mapping for `read_mnirs()`:

- `nirs_channels = c("SmO2 Live", "SmO2 Live(2)")`
- `time_channel = c("hh:mm:ss")`
- `event_channel = c("Lap")`
- `interval_times = list(start = c(204, 868))` (start and end of exercise)

Source

Moxy Monitor (Fortiori Design LLC), exported via PerfPro Studio desktop software (https://perfprostudio.com/).

See Also

`read_mnirs()`, `example_mnirs()`

Examples

```
example_mnirs("moxy_ramp")
```

`palette_mnirs` *Custom* mnirs *colour palette*

Description

Custom *mnirs* colour palette

Usage

```
palette_mnirs(...)
```

Arguments

...	Either a single numeric specifying the number of colours to return, or character strings specifying colour names. If empty, all colours are returned.

Value

Named (when selecting by name) or unnamed character vector of hex colours.

See Also

`theme_mnirs()`, `scale_colour_mnirs()`

Examples

```
scales::show_col(palette_mnirs())
scales::show_col(palette_mnirs(2))
scales::show_col(palette_mnirs("red", "blue", "green"))
```

peak_slope *Peak linear slope*

Description

Identify the maximum positive or negative local linear slope of a numeric vector using rolling least-squares regression, and return the regression parameters of the peak window. Vector-level companion to `analyse_kinetics()` with `method = "peak_slope"`.

Usage

```
peak_slope(
  x,
  t = seq_along(x),
  width = NULL,
  span = NULL,
  align = c("centre", "left", "right"),
  direction = c("auto", "positive", "negative"),
  partial = FALSE,
  na.rm = FALSE,
  verbose = TRUE,
  ...
)
```

Arguments

`x`	A numeric vector of the response variable.
`t`	An *optional* numeric vector of the predictor variable (e.g. time). Default is `seq_along(x)`.
`width`	An integer defining the local window in number of samples around `idx` in which to perform the operation, according to `align`.
`span`	A numeric value defining the local window time span around `idx` in which to perform the operation, according to `align`. In units of `time_channel` or `t`.
`align`	Window alignment as *"centre"/"center"* (the *default*), *"left"*, or *"right"*. Where *"left"* is *forward looking*, and *"right"* is *backward looking* from the current sample.
`direction`	A character string specifying the response direction `"positive"`, or `"negative"`, or detect with `"auto"` (*default*). See *Details*.
`partial`	Logical; default is `FALSE`, only returns values where a full window of valid (non-`NA`) samples are available. If `TRUE`, ignores `NA` and processes available valid samples (see *Details*).
`na.rm`	Logical; default is `FALSE`, propagates any `NA`s to the returned vector. If `TRUE`, ignores `NA`s and processes available valid samples within the local window. May return errors or warnings. (see *Details*).

verbose	Logical. TRUE (*default*) will display, and FALSE will silence warnings and information messages helpful for troubleshooting. Global default can be set via options(mnirs.verbose = FALSE).
...	Additional arguments.

Details

A semi-parametric approach to estimate the steepest local rate of change of a signal. In NIRS signals this can be interpreted as the moment of greatest mismatch between oxygen delivery and extraction. Rolling slopes are computed by rolling_slope(), and the peak window is refit with stats::lm() to return the regression parameters.

Rolling window:

The local window is defined by either width (number of samples) or span (time span in units of t); one of either width or span must be specified.

- width with align = "centre" spans [idx - floor((width - 1) / 2), idx + floor(width / 2)]. Even width values bias alignment to *"left"*, placing the unequal sample forward of idx.
- span with align = "centre" spans [t - span / 2, t + span / 2].

Direction:

direction is detected automatically by default as either *"positive"* (upward) or *"negative"* (downward) response, from the dominant excursion of x above or below its initial baseline (the median of the earliest samples). When tied, the greater absolute rolling slope decides. The greatest local slope in that direction is returned, and direction can be overwritten manually.

Partial windows:

partial = FALSE (the *default*) requires the complete number of samples specified by width or span, and returns NA for any window with fewer samples. partial = TRUE allows computation with as few as 2 valid samples. These windows, such as at edge conditions, are more sensitive to noise and should be used with caution.

Missing values:

na.rm = FALSE (the *default*) propagates any NA in a window to the returned slope. na.rm = TRUE ignores NAs and computes the slope from the remaining valid samples.

Value

A named list containing:

slope	The peak slope value in units of x / t.
intercept	The y-intercept of the peak local regression line.
y	The predicted value of x at the peak slope index.
t	The value of t at the peak slope index.
idx	The integer index of the peak slope window.
fitted	A numeric vector of predicted values spanning the peak slope window.
window_idx	An integer vector of indices spanning the peak slope window.
model	The lm object fit to the peak slope window.

See Also

analyse_kinetics(), rolling_slope(), response_time(), monoexponential()

Examples

```
x <- c(1, 3, 2, 5, 8, 7, 9, 12, 11, 15, 14, 17, 18)

## peak positive slope over a 5-sample window
peak_slope(x, width = 5)

## peak negative slope of the reversed signal
peak_slope(rev(x), width = 5)
```

---

pionirs_occlusion.ftn *1 Hz PIONIRS NIRSBOX export*

Description

Exported from PIONIRS software at 1 Hz, one channel. Containing baseline, arterial occlusion, and recovery phases marked by event tags, from the thenar eminence (CH1).

Format

tab-separated .ftn file with 15 columns and 700 rows:

Iteration Sample index.

Time Elapsed time (seconds).

uA_L1, uA_L2 Absorption coefficient at wavelengths 1 (685 nm) and 2 (830 nm; cm^-1).

uS_L1, uS_L2 Reduced scattering coefficient at wavelengths 1 and 2 (cm^-1).

DPF_L1, DPF_L2 Differential pathlength factor at wavelengths 1 and 2 (μM).

O2Hb Oxyhaemoglobin concentration (μM).

HHb Deoxyhaemoglobin concentration (μM).

THb Total haemoglobin concentration (μM).

StO2 Tissue oxygen saturation (%).

DQI Data quality index (0-1).

Tag Event marker (integer). `0` - no tag; `1` - manual tag; `2` - automatic tag from external trigger; `3` - automatic protocol- specific tag from the measurement software

TagLabel Event label text.

Channels are detected automatically, or can be specified explicitly for read_mnirs():

- `nirs_channels = c("StO2", "O2Hb", "HHb", "THb")`
- `time_channel = c("Time")`
- `event_channel = c("TagLabel")`
- `interval_times = list(start = 91, end = 391)`

Source

PIONIRS S.r.l. (https://www.pionirs.com/)

See Also

`read_mnirs()`, `example_mnirs()`

Examples

```
example_mnirs("pionirs")
```

`plot.mnirs` *Plot* mnirs *objects*

Description

Create a base plot for data frames or lists of data frames with class *"mnirs"*.

Usage

```
## S3 method for class 'mnirs'
plot(x, points = FALSE, time_labels = FALSE, na.omit = FALSE, ...)
```

Arguments

`x`	Data frame or list of data frames of class *"mnirs"* (e.g. from `extract_intervals()`). List input produces a faceted plot with one panel per element.
`points`	Logical. Default is `FALSE`. If `TRUE` displays `ggplot2::geom_points()`. Otherwise displays `ggplot2::geom_lines()`.
`time_labels`	Logical. Default is `FALSE`. If `TRUE` displays x-axis time values formatted as *"h:mm:ss"* using `format_hmmss()`. Otherwise, x-axis values are displayed as numeric.
`na.omit`	Logical. Default is `FALSE`. If `TRUE` omits missing (`NA`) and non-finite `c(Inf, -Inf, NaN)` from display.
`...`	Additional arguments.

Details

When `x` is a named list of *"mnirs"* data frames, elements are bound into a single data frame and displayed as faceted panels via `ggplot2::facet_wrap()`.

Accepts some arguments in `...`, such as `nrow`, `ncol`, and `scales` passed to `ggplot2::facet_wrap()`. `n.breaks` overrides the default number of y-axis breaks. `breaks` overrides the x-axis breaks directly.

Value

A ggplot2 object.

Examples

```
data <- read_mnirs(
    example_mnirs("train.red"),
    nirs_channels = c(smo2 = "SmO2"),
    time_channel = c(time = "Timestamp (seconds passed)"),
    verbose = FALSE
)

## plot time labels as "h:mm:ss"
plot(data, time_labels = TRUE)

data_list <- extract_intervals(
    data,
    start = by_time(2452, 3168),
    span = c(-60, 120),
    verbose = FALSE
)

## plot a list of mnirs data frames as faceted panels
plot(data_list, time_labels = TRUE)
```

---

plot.mnirs_kinetics *Plot* mnirs *kinetics results*

Description

Create a default plot for an *"mnirs_kinetics"* object returned from `analyse_kinetics()`. Observed signals are drawn per `nirs_channel`, faceted by interval, with the fitted response overlaid and the key kinetics coefficient(s) annotated per panel.

Usage

```
## S3 method for class 'mnirs_kinetics'
plot(x, fitted = TRUE, markers = TRUE, labels = TRUE, ...)
```

Arguments

`x`	An *"mnirs_kinetics"* object from `analyse_kinetics()`.
`fitted`	Logical. Default is `TRUE`; overlays a dashed fitted curve for parametric methods (`"peak_slope"`, `"monoexponential"`, `"exponential_drift"`, `"biexponential"`, `"sigmoidal"`, `"sigmoidal_drift"`) in a darker shade of the channel colour. `"response_time"` has no fitted curve.

markers	Logical. Default is TRUE; draws a dotted vertical line at the response onset (start_time) and key coefficient points in a darker shade of the channel colour.
labels	Logical. Default is TRUE; annotates each panel with the key coefficient value(s) for the fitted method, in the right-hand corner the observed signal leaves clear.
...	Additional arguments.

Details

Accepts some arguments in ..., such as label_size passed to ggplot2::geom_text(). Also accepts args passed to plot.mnirs(), such as points, time_labels, nrow, ncol, or scales.

A method with no annotation spec in kinetics_annotations() plots the observed signal and fitted curve only, without markers or labels.

Value

A ggplot2 object.

See Also

analyse_kinetics(), plot.mnirs()

Examples

```
result <- read_mnirs(
    example_mnirs("train.red"),
    nirs_channels = c(smo2 = "SmO2"),
    time_channel = c(time = "Timestamp (seconds passed)"),
    zero_time = TRUE,
    verbose = FALSE
) |>
    resample_mnirs(method = "linear", verbose = FALSE) |>
    extract_intervals(
        group_intervals = "distinct",
        start = by_time(368, 1084),
        span = c(-20, 90),
        zero_time = TRUE,
        verbose = FALSE
    ) |>
    analyse_kinetics(
        method = "peak_slope",
        span = 10,
        verbose = FALSE
    )

plot(result)
```

`portamon_oxcap.xlsx` *10 Hz Artinis Oxysoft export recorded with Portamon*

Description

Exported from Artinis Oxysoft, recorded on Portamon at 10 Hz on the vastus lateralis muscle. Containing two trials of repeated occlusion oxidative capacity testing, each with 17 occlusions.

Format

.xlsx file with header metadata and six columns and 7943 rows:

Column 1 Sample index (divide by sample rate for seconds).

Column 2 tHb: total haemoglobin concentration change (μM).

Column 3 HHb: deoxyhaemoglobin concentration change (μM).

Column 4 O2Hb: oxyhaemoglobin concentration change (μM).

Column 5 Event marker (character).

Column 6 Unmarked event label (character).

Channels are detected automatically from the file legend (the unmarked label column is named `"labels"`), or can be specified explicitly for `read_mnirs()`:

- `nirs_channels = c(THb = 2, HHb = 3, O2Hb = 4)`
- `time_channel = c(sample = 1)`
- `event_channel = c(event = 5, label = "labels")`

Source

Artinis Medical Systems. Portamon, exported via Oxysoft desktop software (https://artinis.com/)

See Also

`read_mnirs()`, `example_mnirs()`

Examples

```
example_mnirs("portamon")
```

print.mnirs *Methods for mnirs objects*

Description

Generic methods for objects of class "mnirs".

Usage

```
## S3 method for class 'mnirs'
print(x, ...)
```

Arguments

x	Object of class "mnirs".
...	Additional arguments passed to print() methods of x or each list element, e.g. n rows for tibbles.

Value

print	Returns x without class attributes.

Examples

```
x <- read_mnirs(
    example_mnirs("train.red"),
    nirs_channels = c(smo2 = "SmO2"),
    time_channel = c(time = "Timestamp (seconds passed)"),
    verbose = FALSE
) |>
    resample_mnirs(method = "linear", verbose = FALSE) |>
    extract_intervals(
        start = by_time(2452, 3168),
        span = c(-60, 120),
        verbose = FALSE
    )

print(x)
```

print.mnirs_kinetics *Methods for mnirs_kinetics objects*

Description

Generic methods for objects returned from `analyse_kinetics()`.

Usage

```
## S3 method for class 'mnirs_kinetics'
print(x, ...)
```

Arguments

x	Object of class "mnirs_kinetics" returned from `analyse_kinetics()`.
...	Additional arguments.

Value

print	Returns a model summary

Examples

```
result <- read_mnirs(
    example_mnirs("train.red"),
    nirs_channels = c(smo2 = "SmO2"),
    time_channel = c(time = "Timestamp (seconds passed)"),
    zero_time = TRUE,
    verbose = FALSE
) |>
    resample_mnirs(method = "linear", verbose = FALSE) |>
    extract_intervals(
        group_intervals = "distinct", ## return each interval distinctly
        start = by_time(368, 1084),
        span = c(-20, 90),
        zero_time = TRUE,
        verbose = FALSE
    ) |>
    analyse_kinetics(
        method = "peak_slope",
        span = 10,
        verbose = FALSE
    )

print(result)
```

read_mnirs *Read* mnirs *data from file*

Description

Import time-series data exported from common muscle NIRS (mNIRS) devices and return a tibble (data frame) of class `"mnirs"` with the specified signal channels and metadata.

Usage

```
read_mnirs(
  file_path,
  nirs_channels = NULL,
  time_channel = NULL,
  event_channel = NULL,
  sample_rate = NULL,
  add_timestamp = FALSE,
  zero_time = FALSE,
  keep_all = FALSE,
  verbose = TRUE
)
```

Arguments

`file_path`	Path of the data file to import. Supported file extensions include `".xls(x)"`, `".csv"`, `".txt"`, and `".ftn(2)"`.
`nirs_channels`	A character vector of one or more column names containing mNIRS signals to import. Names must match the file contents exactly. • If `NULL` (*default*), `read_mnirs()` attempts to automatically detect known `nirs_channel` names from the file contents. • A named character vector is used to rename columns, in the form `c(renamed = "original_name")`.
`time_channel`	A single character vector for the time (or sample) column name. Must match the file contents exactly. • If `NULL` (*default*), `read_mnirs()` attempts to automatically detect a time-like column from known device defaults, and/or time-formatted values. • A named character vector is used to rename the column, e.g. `c(time = "original_name")`.
`event_channel`	An *optional* single character vector for the event or lap column name. Must match the file contents exactly. A named character vector is used to rename the column, e.g. `c(event = "original_name")`.
`sample_rate`	An *optional* numeric sample rate in Hz. If `NULL` (*default*), the sample rate is estimated from `time_channel` (see *Details*).

add_timestamp	Logical. Default is FALSE. If TRUE and the source data contain date-time (POSIXct) values, will add a "timestamp" column in addition to the specified time_channel as a numeric time column.
zero_time	Logical. Default is FALSE. If TRUE, re-calculates numeric time_channel values to start from zero.
keep_all	Logical. FALSE (*default*) will only keep the channels explicitly specified in channels. If TRUE, will keep all columns found in the file data table. • If no channels are specified and the NIRS device file format is recognised, then all columns in the file data table will be returned to allow exploration of the file.
verbose	Logical. TRUE (*default*) will display, and FALSE will silence warnings and information messages helpful for troubleshooting. Global default can be set via options(mnirs.verbose = FALSE).

Details

Header detection:

read_mnirs() searches the file for a header row containing the requested channel names. The header row does not need to be the first row in the file.

- If duplicate column names exist, they are made unique with a numbered suffix (e.g. *_1), and can be renamed accordingly: nirs_channels = c(smo2_left = "smo2", smo2_right = "smo2_1").
- Unnamed columns containing data in the source file will be renamed to col_n, where n is the ordered column number in the file (e.g. col_6). *Artinis Oxysoft* files are an exception to this renaming convention. See *Artinis Oxysoft exports* below).

Renaming channels:

All channels can be renamed with a named character vector in the form c(renamed = "original_name"). The "original_name" must match the file contents header row exactly.

Artinis Oxysoft exports:

Artinis Oxysoft files have numbered data columns, with a "Legend" metadata block with channel names. read_mnirs() can detect and rename these channels automatically:

- nirs_channels names become clean lower-case column names with underscores (e.g. "Rx1 - Tx1 O2Hb" becomes rx1_tx1_o2hb). Channels can still be renamed by any of column number, cleaned name, or legend trace name, e.g. nirs_channels = c(o2hb = 2), c(o2hb = "rx1_tx1_o2hb"), or c(o2hb = "Rx1 - Tx1 O2Hb").
- "(Sample number)" column is renamed sample, and a time column in seconds is automatically derived from the export sample rate.
- "(Event)" column is renamed event and set as event_channel.
- *Oxysoft* exports a trailing un-numbered column containing optional event label text. This is renamed labels and returned with keep_all = TRUE, or dropped when empty. It can be selected as the event column explicitly with event_channel = c(event = "labels").

Explicit nirs_channels, time_channel, and event_channel renaming (as above) overrides automatically detected names.

PIONIRS exports:

PIONIRS `.ftn(2)` files are detected with `"Time"` as `time_channel`, `"TagLabel"` as `event_channel`, and `StO2` channels as `nirs_channels`. The `"Iteration"` sample index and numeric `"Tag"` companion columns are returned beside `time_channel` and `event_channel` with `keep_all = TRUE`.

Time parsing:

If `time_channel` is left as `NULL`, it can be resolved from a known NIRS device default, or by detecting a time-like column name (e.g. `"time"`, `"hh:mm:ss"`), or by detecting a column with date-time formatted (POSIXct-like) values.

If `time_channel` is a date-time (POSIXct) format, it will be converted to numeric and re-based to start from `0`, regardless of `zero_time`.

Sample rate:

If `sample_rate` is not specified, it is estimated from differences in `time_channel`. When irregular time sampling is detected, the estimated median `sample_rate` will be approximated as common known recording rate (e.g. an estimated rate of `11` may be rounded to `10 Hz`).

If `time_channel` is specified as a sample index (e.g. *Artinis Oxysoft "sample"* or *PIONIRS "Iterations"*), `sample_rate` will be mis-estimated as `1 Hz`. `sample_rate` should be specified explicitly in this case.

Data cleaning:

Entirely empty rows and columns are removed. Invalid values (e.g. `c(NaN, Inf, "-")`) are standardized to `NA`. A warning is displayed (respecting `verbose`) when irregular sampling is detected (e.g. non-monotonic, repeated, or unequal `time_channel` values). In this case, it is recommended to use `resample_mnirs()` to standardise the time grid to the desired `sample_rate`.

Value

A tibble of class `"mnirs"`. Metadata are stored as attributes and can be accessed with `attributes(data)`.

Examples

```
read_mnirs(
    file_path = example_mnirs("moxy_ramp"), ## call an example data file
    nirs_channels = c(
        smo2_left = "SmO2 Live",            ## identify and rename channels
        smo2_right = "SmO2 Live(2)"
    ),
   time_channel = c(time = "hh:mm:ss"),    ## date-time format will be converted to numeric
    event_channel = NULL,                   ## leave blank if unused
   sample_rate = NULL,                    ## if blank, will be estimated from time_channel
    add_timestamp = FALSE,                  ## omit a date-time timestamp column
    zero_time = TRUE,                       ## recalculate time values from zero
    keep_all = FALSE,                       ## return only the specified data channels
    verbose = TRUE                          ## show warnings & messages
)
```

replace_mnirs *Replace outliers, invalid, and missing values in* mnirs *data*

Description

Detect and replace local outliers, specified invalid values, and missing NA values across nirs_channels within an *"mnirs"* data frame. replace_mnirs() operates on a data frame, a list of data frames, or a grouped data frame, extending the vectorised functions.

replace_invalid() detects specified invalid values or range cutoffs in a numeric vector and replace them with the local median value or NA.

replace_outliers() detects local outliers in a numeric vector using a Hampel filter and replaces with the local median value or NA.

replace_missing() detects missing (NA) values in a numeric vector and replaces via interpolation.

Usage

```
replace_mnirs(
  data,
  nirs_channels = NULL,
  time_channel = NULL,
  invalid_values = NULL,
  invalid_above = NULL,
  invalid_below = NULL,
  outlier_cutoff = NULL,
  width = NULL,
  span = NULL,
  method = c("linear", "median", "locf", "none"),
  verbose = TRUE
)

replace_invalid(
  x,
  t = seq_along(x),
  invalid_values = NULL,
  invalid_above = NULL,
  invalid_below = NULL,
  width = NULL,
  span = NULL,
  method = c("median", "none"),
  verbose = TRUE,
  ...
)

replace_outliers(
  x,
  t = seq_along(x),
```

```
  outlier_cutoff = 3,
  width = NULL,
  span = NULL,
  method = c("median", "none"),
  verbose = TRUE,
  ...
)

replace_missing(
  x,
  t = seq_along(x),
  width = NULL,
  span = NULL,
  method = c("linear", "median", "locf"),
  verbose = TRUE,
  ...
)
```

Arguments

`data`	A data frame of class *"mnirs"* containing time series data and metadata, a list of data frames, or a grouped data frame (see *Details*).
`nirs_channels`	A character vector giving the names of mNIRS columns to operate on. Must match column names in `data` exactly. • If `NULL` (default), the `nirs_channels` metadata attribute of `data` is used.
`time_channel`	A character string naming the time or sample column. Must match a column name in `data` exactly. • If `NULL` (default), the `time_channel` metadata attribute of `data` is used.
`invalid_values`	A numeric vector of invalid values to be replaced, e.g. `invalid_values = c(0, 100, 102.3)`. Default `NULL` will not replace invalid values.
`invalid_above, invalid_below`	Numeric values each specifying cutoff values, above or below which (respectively) will be replaced, *inclusive* of the specified cutoff values.
`outlier_cutoff`	A numeric value for the local outlier threshold, as the number of standard deviations from the local median. • Default `NULL` will not replace outliers. • Lower values are more sensitive and flag more outliers; higher values are more conservative. • `outlier_cutoff = 3` Pearson's 3 sigma edit rule. `outlier_cutoff = 2` approximates a Tukey-style 1.5*IQR rule. `outlier_cutoff = 0` Tukey's median filter.
`width`	An integer defining the local window in number of samples centred on `idx`, between `[idx - floor(width/2), idx + floor(width/2)]`.
`span`	A numeric value defining the local window time span around `idx` in units of `time_channel` or `t`, between `[t - span/2, t + span/2]`.

method	A character string indicating how to handle NA replacement (see *Details*): "linear" Replaces NAs via linear interpolation (the *default*) using stats::approx(). "median" Replaces NAs with the local median of valid values within a centred window defined by width or span. "locf" *"Last observation carried forward"*. Replaces NAs with the most recent valid value to the left for trailing NAs or to the right for leading NAs, using stats::approx(). "none" Returns NAs without replacement.
verbose	Logical. TRUE (*default*) will display, and FALSE will silence warnings and information messages helpful for troubleshooting. Global default can be set via options(mnirs.verbose = FALSE).
x	A numeric vector of the response variable.
t	An *optional* numeric vector of the predictor variable (e.g. time). Default is seq_along(x).
...	Additional arguments.

Details

Automatic channel detection:

nirs_channels and time_channel are retrieved automatically from *"mnirs"* metadata if not specified explicitly. Columns in data not listed in nirs_channels are passed through unprocessed.

The rolling window:

replace_outliers() and replace_missing() (when method = "median") operate over a local rolling window for outlier detection and median interpolation. The window is specified by either width as the number of samples, or span as the time span in units of time_channel. A partial window is calculated at the edges of the data.

Replace invalid values with with replace_invalid():

Specific invalid_values can be replaced, such as c(0, 100, 102.3). Data ranges can be replaced with cutoff values specified by invalid_above and invalid_below, where any values higher or lower than the specified cutoff values (respectively) will be replaced, *inclusive* of the cutoff values themselves.

Outlier detection with replace_outliers():

Rolling local medians are computed across x within a window defined by width (number of samples) or span (time span in units of t).

Outliers are detected with robust median absolute deviation (MAD), adapted from pracma::hampel(). Deviations equal to or less than the smallest absolute time series difference in x are excluded, to avoid flagging negligible differences where local data have minimal or zero variation.

Replacement behaviour:

Values of x outside the local bounds defined by outlier_cutoff are identified as outliers and either replaced with the local median (method = "median", the *default*) or set to NA (method = "none").

Existing NA values in x are *not* replaced. They are passed through to the returned vector. See replace_missing().

Choosing outlier_cutoff:

`outlier_cutoff` is the number of (MAD-normalised) standard deviations from the local median. Higher values are more conservative; lower values flag more outliers.

- `outlier_cutoff = 3` – Pearson's 3 sigma edit rule (default).
- `outlier_cutoff = 2` – approximately Tukey-style 1.5*IQR rule.
- `outlier_cutoff = 0` – Tukey's median filter (every point replaced by local median).

Interpolation with replace_missing():

`method = "linear"` and `method = "locf"` use `stats::approx()` with `rule = 2`, so leading `NA`s are filled by *"nocb"* (*"next observation carried backward"*) and trailing `NA`s by *"locf"*.

`method = "median"` calculates the local median of valid (non-`NA`) values to either side of `NA`s, within a window defined by `width` (number of samples) or `span` (time span in units of `t`). Sequential `NA`s are all replaced by the same median value.

Edge behaviour for method = "median":

If there are no valid values within `span` to one side of the `NA`, the median of the other side is used (i.e. for leading and trailing `NA`s). If there are no valid values within either side, the first valid sample on either side is used (equivalent to `replace_missing(x, width = 1)`).

Value

`replace_mnirs()` returns a tibble of class `"mnirs"` with metadata available via `attributes()`. For list or grouped data frame input, returns a named list of *"mnirs"* tibbles, one per interval.

`replace_invalid()` returns a numeric vector the same length as `x` with invalid values replaced.

`replace_outliers()` returns a numeric vector the same length as `x` with outliers replaced.

`replace_missing()` returns a numeric vector the same length as `x` with missing values replaced.

Per-channel arguments

Arguments apply globally to all `nirs_channels` by default. Relevant arguments can instead be supplied uniquely per-channel as a named `list()`, with names matching `nirs_channels`, e.g.

```
replace_mnirs(
    data,
    nirs_channels = c(hhb, smo2),
    invalid_values = list(hhb = -1, smo2 = c(0, 100)),
    invalid_above = list(hhb = 10),
    span = list(3, hhb = 5)
)
```

- A non-list value applies to every channel (the *default* behaviour).
- A `list()` named by `nirs_channels` applies per-channel values.
- A single unnamed value in the list will be applied to unlisted channels (e.g. `span = list(3, hhb = 5)` gives `hhb` 5 and every other channel 3). If no unnamed fallback value in the list, channels not named in the list will be returned un-processed (e.g. `span = list(hhb = 5)` will only process `hhb`).
- `list()` names not matching `nirs_channels` are warned about and ignored.

Data input formats

mnirs processing functions accept `data` in multiple formats:

- A **single *"mnirs"* data frame** is processed and returned directly.
- A **list of *"mnirs"* data frames**: each interval is processed separately and returned as a named list.
- A **grouped *"mnirs"* data frame**, e.g. with `dplyr::group_by()`: the data frame is split by grouping levels and each group is processed as a separate interval, returned as a named list.

Examples

```
## vectorised operations
x <- c(1, 999, 3, 4, 999, 6)
replace_invalid(x, invalid_values = 999, width = 3, method = "median")

(x_na <- replace_outliers(x, outlier_cutoff = 3, width = 3, method = "none"))

replace_missing(x_na, method = "linear")

## read example data
data <- read_mnirs(
    file_path = example_mnirs("moxy_ramp"),
    nirs_channels = c(smo2 = "SmO2 Live"),
    time_channel = c(time = "hh:mm:ss"),
    verbose = FALSE
)

## clean data
data_clean <- replace_mnirs(
    data,                  ## channels retrieved from metadata
    invalid_values = 0,    ## known invalid values in the data
    invalid_above = 90,    ## remove data spikes above 90
    outlier_cutoff = 3,    ## Pearson's 3 sigma edit rule
    width = 7,            ## window for outlier detection and interpolation
    method = "linear"      ## linear interpolation over NAs
)


  if (requireNamespace("ggplot2", quietly = TRUE)) {
    ## plot original and show where values have been replaced
    ## ignore warning about replacing the existing colour scale
    plot(data, time_labels = TRUE) +
      ggplot2::scale_colour_manual(
        name = NULL,
        breaks = c("smo2", "replaced"),
        values = palette_mnirs(2)
      ) +
      ggplot2::geom_point(
        data = data[data_clean$smo2 != data$smo2, ],
        ggplot2::aes(y = smo2, colour = "replaced"),
        na.rm = TRUE
```

```
    ) +
    ggplot2::geom_line(
      data = {
        data_clean[!is.na(data$smo2), "smo2"] <- NA
        data_clean
      },
      ggplot2::aes(y = smo2, colour = "replaced"),
      linewidth = 1, na.rm = TRUE
    )
  }
```

`resample_mnirs` *Re-sample an* mnirs *data frame*

Description

Up- or down-sample an *"mnirs"* data frame to a new sample rate, filling new samples via nearest-neighbour matching or interpolation.

Usage

```
resample_mnirs(
  data,
  time_channel = NULL,
  sample_rate = NULL,
  resample_rate = sample_rate,
  method = c("none", "linear", "locf"),
  verbose = TRUE
)
```

Arguments

`data`	A data frame of class *"mnirs"* containing time series data and metadata, a list of data frames, or a grouped data frame (see *Details*).
`time_channel`	A character string naming the time or sample column. Must match a column name in `data` exactly. • If `NULL` (default), the `time_channel` metadata attribute of `data` is used.
`sample_rate`	A numeric sample rate in Hz. • If `NULL` (default), the `sample_rate` metadata attribute of `data` will be used if detected, or the sample rate will be estimated from `time_channel`.
`resample_rate`	An *optional* sample rate (Hz) for the output data frame. If `NULL` (*default*) re-samples to the existing `sample_rate`, which regularises any irregular samples without changing the rate.
`method`	A character string specifying how new samples are filled. Default is *"none"*. Filling must be opted into explicitly (see *Details*):

"none" Matches each new sample to the nearest original time_channel value without any interpolation, to within tolerance of half a sample-interval. New samples are returned as NA.

"locf" (*"Last observation carried forward"*). Fills new and missing samples with the most recent valid non-NA value to the left, or the nearest valid value to the right for leading NAs. Safe for numeric, integer, and character columns.

"linear" Fills new and missing samples via linear interpolation using stats::approx(). Suitable for numeric columns only; non-numeric columns will fall back to "locf" behaviour.

verbose — Logical. TRUE (*default*) will display, and FALSE will silence warnings and information messages helpful for troubleshooting. Global default can be set via options(mnirs.verbose = FALSE).

Details

This function uses replace_missing() (based on stats::approx()) to interpolate across new samples in the resampled data range.

Sample rate and time channel:

time_channel and sample_rate are retrieved automatically from data of class *"mnirs"*, if not defined explicitly.

Otherwise, sample_rate will be estimated from the values in time_channel. However, this may return unexpected values, and it is safer to define sample_rate explicitly or retrieve it from *"mnirs"* metadata.

Default behaviour:

When resample_rate is omitted, the output has the same sample_rate as the input but with a regular, evenly-spaced time_channel. This is useful for regularising data that contains missing or repeated samples without changing the nominal rate.

Column handling:

Numeric columns are interpolated according to method (see ?replace_missing). Non-numeric columns (e.g. character event labels, integer lap numbers) are always filled by last-observation-carried-forward, regardless of method:

- For method = "none", existing rows are matched to the nearest original values of time_channel without interpolation or filling, meaning newly created samples and any NAs in the original data are returned as NA.
- When down-sampling, numeric columns use linear interpolation averaging. Non-numeric columns use the first valid value in each output bin.

Value

A tibble of class "mnirs". Metadata are stored as attributes and can be accessed with attributes(data). For list or grouped data frame input, returns a named list of *"mnirs"* tibbles, one per interval.

Data input formats

mnirs processing functions accept `data` in multiple formats:

- A **single *"mnirs"* data frame** is processed and returned directly.
- A **list of *"mnirs"* data frames**: each interval is processed separately and returned as a named list.
- A **grouped *"mnirs"* data frame**, e.g. with `dplyr::group_by()`: the data frame is split by grouping levels and each group is processed as a separate interval, returned as a named list.

Examples

```
## read example data
data <- read_mnirs(
    file_path = example_mnirs("moxy_ramp"),
    nirs_channels = c(smo2 = "SmO2 Live"),
    time_channel = c(time = "hh:mm:ss"),
    verbose = TRUE
)

## note warning about irregular sampling
data

data_resampled <- resample_mnirs(
    data,               ## blank channels will be retrieved from metadata
    resample_rate = 2,  ## blank by default will resample to `sample_rate`
    method = "linear",  ## linear interpolation across resampled indices
    verbose = TRUE
)

## note the altered `time` values resolving the above warning
data_resampled
```

---

rescale_mnirs *Rescale data range*

Description

Expand or reduce the range (min and max values) of data channels to a new amplitude/dynamic range, e.g. rescale the range of NIRS data to `c(0, 100)`.

Usage

```
rescale_mnirs(
  data,
  nirs_channels = NULL,
  group_channels = c("ensemble", "distinct"),
  range,
  verbose = TRUE
)
```

Arguments

`data`	A data frame of class *"mnirs"* containing time series data and metadata, a list of data frames, or a grouped data frame (see *Details*).
`nirs_channels`	A character vector giving the names of mNIRS columns to operate on. Must match column names in `data` exactly. • If `NULL` (default), the `nirs_channels` metadata attribute of `data` is used.
`group_channels`	Either a character string or a `list()` of channel-name vectors specifying how to group `nirs_channels` (see *Details*). `"ensemble"` The *default*. Operate on all channels together, preserving the relative scaling between channels. `"distinct"` Operate on each channel independently, losing the relative scaling between channels. `list(c("A", "B"), c("C", "D"))` Operate on channels `A` & `B` in one group, and `C` & `D` in another group. Groups can be named (e.g. `list(smo2 = c("A", "B"))`). Each group must be non-empty and resulting group names must be unique.
`range`	A numeric vector in the form `c(min, max)`, indicating the range of output values to which `nirs_channels` will be rescaled.
`verbose`	Logical. `TRUE` (*default*) will display, and `FALSE` will silence warnings and information messages helpful for troubleshooting. Global default can be set via `options(mnirs.verbose = FALSE)`.

Details

`group_channels` controls how data channels are grouped to preserve absolute or relative scaling.

- `group_channels = "ensemble"` (the *default*) rescales all `nirs_channels` to a common range, preserving relative scaling between channels.
- `group_channels = "distinct"` rescales each channel independently, losing relative scaling between channels.
- A `list()` of channel-name vectors (e.g. `list(c("A", "B"), c("C", "D"))`) rescales channels `A` & `B` together and `C` & `D` together, preserving relative scaling within, but not between groups. `nirs_channels` omitted from the list are rescaled independently.
- Channel groups can be named (e.g. `list(smo2 = c("A", "B"))`) and names used as keys for per-group `range` argument.
- Channels (columns) in `data` not in `nirs_channels` are passed through without processing to the output data frame.

`nirs_channels` can be retrieved automatically from `data` of class *"mnirs"* which has been processed with `{mnirs}`, if not defined explicitly.

Value

A tibble of class *"mnirs"* with metadata available with `attributes()`. For list or grouped data frame input, returns a named list of *"mnirs"* tibbles, one per interval.

Data input formats

mnirs processing functions accept `data` in multiple formats:

- A **single *"mnirs"* data frame** is processed and returned directly.
- A **list of *"mnirs"* data frames**: each interval is processed separately and returned as a named list.
- A **grouped *"mnirs"* data frame**, e.g. with `dplyr::group_by()`: the data frame is split by grouping levels and each group is processed as a separate interval, returned as a named list.

Per-channel arguments

Arguments apply globally to all `nirs_channels` by default. Relevant arguments can instead be supplied uniquely per-channel as a named `list()`, with names matching either `nirs_channels` or list names in `group_channels`, e.g.

```
shift_mnirs(
    data,
    nirs_channels = c(A, B, C),
    group_channels = list(smo2 = c(A, B), hhb = C),
    to = list(100, C = 0),
    width = list(smo2 = 3),
    span = list(hhb = 5),
    position = "first"
)
```

- A non-list value applies to every channel (the *default* behaviour).
- A `list()` named by `nirs_channels` or `group_channels` applies per-channel / per-group values.
- A single unnamed value in the list will be applied to unlisted channels (e.g. `span = list(3, hhb = 5)` gives `hhb` 5 and every other channel 3). If no unnamed fallback value in the list, channels not named in the list will be returned un-processed (e.g. `span = list(hhb = 5)` will only process `hhb`).
- `list()` names not matching `nirs_channels` or `group_channels` are warned about and ignored.

Examples

```
## read example data
data <- read_mnirs(
    file_path = example_mnirs("moxy_ramp"),
    nirs_channels = c(smo2_left = "SmO2 Live",
                      smo2_right = "SmO2 Live(2)"),
    time_channel = c(time = "hh:mm:ss"),
    verbose = FALSE
) |>
    rescale_mnirs(        ## un-grouped nirs channels to rescale separately
        nirs_channels = c(smo2_left, smo2_right),
        group_channels = "distinct",
```

```
        range = c(0, 100)  ## rescale to a 0-100% functional exercise range
    )

data


    if (requireNamespace("ggplot2", quietly = TRUE)) {
        plot(data, time_labels = TRUE) +
            ggplot2::geom_hline(yintercept = c(0, 100), linetype = "dotted")
    }
```

`response_time` *Fractional response time*

Description

Estimate the time at which a numeric vector reaches a specified fraction of its total response amplitude relative to a baseline, e.g. *half-response time* at `response_fraction = 0.5`. Vector-level companion to `analyse_kinetics()` with `method = "response_time"`.

Usage

```
response_time(
  x,
  t = seq_along(x),
  start_time = 0,
  response_fraction = 0.5,
  direction = c("auto", "positive", "negative"),
  verbose = TRUE,
  ...
)
```

Arguments

`x`	A numeric vector of the response variable.
`t`	An *optional* numeric vector of the predictor variable (e.g. time). Default is `seq_along(x)`.
`start_time`	A numeric value in units of `t` specifying the response onset. Samples where `t <= start_time` define the baseline window. *Default* is `0`.
`response_fraction`	A numeric vector in the range `[0, 1]` specifying the fractional response amplitude(s) to detect. Defaults to `0.5` (50% response, i.e. half-response time). Multiple values (e.g. `c(0.5, 0.632)`) return one element per fraction.
`direction`	A character string specifying the response direction `"positive"`, or `"negative"`, or detect with `"auto"` (*default*). See *Details*.

verbose	Logical. TRUE (*default*) will display, and FALSE will silence warnings and information messages helpful for troubleshooting. Global default can be set via options(mnirs.verbose = FALSE).
...	Additional arguments.

Details

A non-parametric approach (estimated directly from the observed data without assuming a specific mathematical shape). response_fraction = 0.5 approximates the inflection point (xmid) of a symmetric sigmoid function. response_fraction = 0.632 approximates the time constant (tau; τ) of a monoexponential function, or xmid of a left-Gompertz function. response_fraction = 0.368 approximates xmid of a right-Gompertz function. This is a good fallback estimation method if parametric methods are not successfully fit.

Method:

The target response value is: fitted = A + (B - A) * response_fraction

Where A is the mean baseline value (t <= start_time) and B is the extreme (peak or trough) value after start_time. response_value is the first observed sample equal to or greater/lesser than the target fitted value (above for *"positive"*, below for *"negative"* direction). response_time is the elapsed time from start_time to response_value.

analyse_kinetics() first trims x to end_window past the first extreme, so B there is the first local extreme with no greater/lesser values within end_window. Called directly, B is the global extreme of x after start_time.

Direction:

direction is detected automatically by default as either *"positive"* (upward) or *"negative"* (downward) response, from the dominant excursion of x above or below its initial baseline (the median of the earliest samples). When tied, the greater absolute extreme decides. B is the maximum for *"positive"* or the minimum for *"negative"*, and can be overwritten manually.

Baseline:

When no samples exist where t <= start_time, the first sample x[1] is used as the baseline A with a warning. start_time must be within the range of t.

Value

A named list containing:

A	The mean baseline value of x where t <= start_time.
B	The extreme (maximum or minimum) value of x after start_time.
response_time	The elapsed time from start_time to the fractional response, in units of t; one element per response_fraction.
response_value	The observed value of x at the response index; one element per response_fraction.
fitted	The target fractional response value A + (B - A) * response_fraction; one element per response_fraction.
baseline_idx	Integer indices where t <= start_time.
response_idx	Integer index at each response_value.
extreme_idx	Integer index at the extreme value B.

See Also

analyse_kinetics(), peak_slope(), monoexponential()

Examples

```
## create an exponential curve with random noise
set.seed(13)
t <- 0:60
x <- monoexponential(t, A = 20, B = 60, tau = 8, TD = 10) +
    rnorm(length(t), 0, 1)

## half-response time (0.5) and time constant approximation (0.632 ~= tau)
RT <- response_time(x, t, start_time = 10, response_fraction = c(0.5, 0.632))
RT$response_time

plot(t, x, type = "l", col = "grey60", xlab = "t", ylab = "x")
## mean baseline `A` across the baseline window
segments(
    t[min(RT$baseline_idx)], RT$A,
    t[max(RT$baseline_idx)], RT$A,
    col = "red", lwd = 2
)
## response values at 0.5 (red) and 0.632 (blue), and the extreme `B`
points(
    t[RT$response_idx],
    RT$response_value,
    col = c("red", "blue"),
    pch = 19
)
points(t[RT$extreme_idx], RT$B, col = "red", pch = 19)
```

---

scale_colour_mnirs *Scales for custom* mnirs *palette*

Description

Scales for custom *mnirs* palette

Usage

```
scale_colour_mnirs(..., aesthetics = "colour")

scale_color_mnirs(..., aesthetics = "colour")

scale_fill_mnirs(..., aesthetics = "fill")
```

Arguments

`...`	Arguments passed to `ggplot2::discrete_scale()`.
`aesthetics`	A character vector with aesthetic(s) passed to `ggplot2::discrete_scale()`. *Default* is `"colour"`.

Value

A ggplot2 scale object.

See Also

`theme_mnirs()`, `palette_mnirs()`

Examples

```
## plot example data
data <- read_mnirs(
    file_path = example_mnirs("moxy_ramp"),
    nirs_channels = c(smo2_left = "SmO2 Live",
                      smo2_right = "SmO2 Live(2)"),
    time_channel = c(time = "hh:mm:ss"),
    verbose = FALSE
)

ggplot2::ggplot(data, ggplot2::aes(x = time)) +
    theme_mnirs() +
    scale_colour_mnirs() +
    ggplot2::geom_line(ggplot2::aes(y = smo2_left, colour = "smo2_left")) +
    ggplot2::geom_line(ggplot2::aes(y = smo2_right, colour = "smo2_right"))
```

`shift_mnirs` *Shift data range*

Description

Move the range of data channels in a data frame up or down, while preserving the absolute amplitude/dynamic range of each channel, and the relative scaling across channels. e.g. shift the minimum data value to zero for all positive values, or shift the mean of the first time span in a recording to zero.

Usage

```
shift_mnirs(
  data,
  nirs_channels = NULL,
  time_channel = NULL,
  group_channels = c("ensemble", "distinct"),
```

```
  to = NULL,
  by = NULL,
  width = NULL,
  span = NULL,
  position = c("min", "max", "first"),
  verbose = TRUE
)
```

Arguments

`data`	A data frame of class *"mnirs"* containing time series data and metadata, a list of data frames, or a grouped data frame (see *Details*).
`nirs_channels`	A character vector giving the names of mNIRS columns to operate on. Must match column names in `data` exactly. • If `NULL` (default), the `nirs_channels` metadata attribute of `data` is used.
`time_channel`	A character string naming the time or sample column. Must match a column name in `data` exactly. • If `NULL` (default), the `time_channel` metadata attribute of `data` is used.
`group_channels`	Either a character string or a `list()` of channel-name vectors specifying how to group `nirs_channels` (see *Details*). `"ensemble"` The *default*. Operate on all channels together, preserving the relative scaling between channels. `"distinct"` Operate on each channel independently, losing the relative scaling between channels. `list(c("A", "B"), c("C", "D"))` Operate on channels `A` & `B` in one group, and `C` & `D` in another group. Groups can be named (e.g. `list(smo2 = c("A", "B"))`). Each group must be non-empty and resulting group names must be unique.
`to`	A numeric value in units of `nirs_channels` to which the data channels will be shifted, e.g. shift the minimum value to zero.
`by`	A numeric value in units of `nirs_channels` by which the data channels will be shifted, e.g. shift all values up by 10 units.
`width`	An integer defining the local window in number of samples centred on `idx`, between `[idx - floor(width/2), idx + floor(width/2)]`.
`span`	A numeric value defining the local window time span around `idx` in units of `time_channel` or `t`, between `[t - span/2, t + span/2]`.
`position`	Indicates where the reference values will be shifted from. `"min"` (The *default*) will shift the minimum value(s) `to` or `by` the specified value. `"max"` Will shift the maximum value(s) `to` or `by` the specified values. `"first"` Will shift first value(s) `to` or `by` the specified values.
`verbose`	Logical. `TRUE` (*default*) will display, and `FALSE` will silence warnings and information messages helpful for troubleshooting. Global default can be set via `options(mnirs.verbose = FALSE)`.

Details

`group_channels` controls how data channels are grouped to preserve absolute or relative scaling (see `rescale_mnirs()`).

- `group_channels = "ensemble"` (the *default*) shifts all `nirs_channels` to a common value, preserving relative scaling between channels.
- `group_channels = "distinct"` shifts each channel independently, losing relative scaling between channels.
- A `list()` of channel-name vectors (e.g. `list(c("A", "B"), c("C", "D"))`) shifts channels `A` & `B` together and `C` & `D` together, preserving relative scaling within, but not between groups. `nirs_channels` omitted from the list are rescaled independently.
- Channel groups can be named (e.g. `list(smo2 = c("A", "B"))`) and names used as keys for per-group arguments.

Only one of either `to` or `by` and one of either `width` or `span` should be defined for each `group_channels`. If both of either pairing are defined, `to` will be preferred over `by`, and `width` will be preferred over `span`.

- Channels (columns) in `data` not in `nirs_channels` are passed through without processing to the output data frame.

`nirs_channels` and `time_channel` can be retrieved automatically from `data` of class *"mnirs"* which has been processed with `{mnirs}`, if not defined explicitly.

When `position` is *"min"* or *"max"*, only full windows of `width` or `span` are considered, to avoid bias from noise at edge conditions with partial samples.

Value

A tibble of class *"mnirs"* with metadata available with `attributes()`. For list or grouped data frame input, returns a named list of *"mnirs"* tibbles, one per interval.

Per-channel arguments

Arguments apply globally to all `nirs_channels` by default. Relevant arguments can instead be supplied uniquely per-channel as a named `list()`, with names matching either `nirs_channels` or list names in `group_channels`, e.g.

```
shift_mnirs(
    data,
    nirs_channels = c(A, B, C),
    group_channels = list(smo2 = c(A, B), hhb = C),
    to = list(100, C = 0),
    width = list(smo2 = 3),
    span = list(hhb = 5),
    position = "first"
)
```

- A non-list value applies to every channel (the *default* behaviour).

- A `list()` named by `nirs_channels` or `group_channels` applies per-channel / per-group values.
- A single unnamed value in the list will be applied to unlisted channels (e.g. `span = list(3, hhb = 5)` gives `hhb` 5 and every other channel 3). If no unnamed fallback value in the list, channels not named in the list will be returned un-processed (e.g. `span = list(hhb = 5)` will only process `hhb`).
- `list()` names not matching `nirs_channels` or `group_channels` are warned about and ignored.

Data input formats

mnirs processing functions accept `data` in multiple formats:

- A **single *"mnirs"* data frame** is processed and returned directly.
- A **list of *"mnirs"* data frames**: each interval is processed separately and returned as a named list.
- A **grouped *"mnirs"* data frame**, e.g. with `dplyr::group_by()`: the data frame is split by grouping levels and each group is processed as a separate interval, returned as a named list.

Examples

```
## read example data
data <- read_mnirs(
    file_path = example_mnirs("moxy_ramp"),
    nirs_channels = c(smo2_left = "SmO2 Live",
                      smo2_right = "SmO2 Live(2)"),
    time_channel = c(time = "hh:mm:ss"),
    verbose = FALSE
) |>
    shift_mnirs(        ## un-grouped nirs channels to shift separately
        nirs_channels = c(smo2_left, smo2_right),
        group_channels = "distinct",
        to = 0,         ## NIRS values will be shifted to zero
        span = 120,     ## shift the *first* 120 sec of data to zero
        position = "first"
    )

data


    if (requireNamespace("ggplot2", quietly = TRUE)) {
        plot(data, time_labels = TRUE) +
            ggplot2::geom_hline(yintercept = 0, linetype = "dotted")
    }
```

`sigmoidal_drift` *Sigmoidal-drift function*

Description

Calculate a two-phase curve: a *fast* sigmoidal primary response of the given `shape` plus a *slow* linear secondary drift beginning near the ending asymptote. Model family fit by `analyse_kinetics()` with `method = "sigmoidal_drift"`, and by `stats::nls()` via the self-starting wrapper `SSsigmoidal_drift()`.

Usage

```
sigmoidal_drift(
  t,
  A,
  B,
  xmid,
  slope,
  slope_B,
  drift_fraction,
  shape = c("symmetric", "gompertz", "gompertz_left")
)
```

Arguments

`t`	A numeric vector of the predictor variable (time).
`A`	A numeric parameter for the starting asymptote of the response variable.
`B`	A numeric parameter for the ending asymptote of the response variable.
`xmid`	A numeric parameter for the time at the *inflection point* (the steepest point) of the curve, in units of the predictor variable `t`.
`slope`	A numeric parameter for the response rate `dx/dt` at the inflection `xmid`.
`slope_B`	A numeric parameter for the linear drift rate `dx/dt` of the secondary phase at the ending asymptote `B`, in response units per unit of the predictor variable `t`.
`drift_fraction`	A numeric fraction of the primary amplitude `B - A` in `(0.5, 1)` at which the linear drift begins, where the sigmoid reaches `A + drift_fraction * (B - A)`.
`shape`	Character; the 4-parameter sigmoidal shape. One of `"symmetric"` (*default*; `logistic()`), `"gompertz"` (`gompertz()`), or `"gompertz_left"` (`gompertz_left()`).

Details

Model equation:

`S(t) + slope_B * pmax(t - onset, 0)`

`S(t)` is the 4-parameter sigmoid of the given `shape` with asymptotes `A` and `B`, inflection `xmid`, and inflection rate `slope` (see `logistic()` and `gompertz()`). The drift is a hinge line anchored at zero at the onset, so it is exactly zero up to the onset.

The drift onset is not a free estimate: it is the analytic inverse of each shape at the `drift_fraction` fraction `f` of its amplitude, `onset = xmid + u / k`:

- shape = "symmetric": k = 4 * slope / (B - A); u = log(f / (1 - f)).
- shape = "gompertz": k = slope * e / (B - A); u = -log(-log(f)).
- shape = "gompertz_left": k = slope * e / (B - A); u = log(-log(1 - f)).

The "gompertz" form places its onset furthest past xmid (slow tail) and "gompertz_left" nearest (fast tail).

The excursion point texc is where the drift rate overtakes the decaying primary rate, |S'(t)| = |slope_B|, floored at the drift onset.

Value

A numeric vector of predicted values the same length as the predictor variable t.

See Also

analyse_kinetics(), SSsigmoidal_drift(), logistic(), gompertz(), gompertz_left(), exponential_drift()

Examples

```
## create a sigmoidal curve with late linear drift and random noise
set.seed(13)
t <- 1:120
x <- sigmoidal_drift(
    t, A = 10, B = 100, xmid = 40, slope = 4,
    slope_B = -0.4, drift_fraction = 0.95
) + rnorm(length(t), 0, 2)
data <- data.frame(t, x)

## the drift onset fraction is held constant in the formula
model <- nls(
    x ~ SSsigmoidal_drift(
        t, A, B, xmid, slope, slope_B, drift_fraction = 0.95
    ),
    data = data,
    algorithm = "port",
    control = nls.control(warnOnly = TRUE)
)
summary(model)

y <- predict(model, data)


    if (requireNamespace("ggplot2", quietly = TRUE)) {
        ggplot2::ggplot(data, ggplot2::aes(t, x)) +
            theme_mnirs() +
            ggplot2::geom_point() +
            ggplot2::geom_line(ggplot2::aes(y = y))
    }
```

`SSbiexponential` *Self-starting biexponential model*

Description

Creates initial coefficient estimates for a `selfStart` wrapper around `biexponential()`, for use with `stats::nls()`. Supports both the 5-parameter (A, B, tau, B2, tau2) and 6-parameter forms adding a time delay TD; arity is inferred from the formula passed to `stats::nls()`.

Usage

```
SSbiexponential(t, A, B, tau, B2, tau2, TD)
```

Arguments

`t`	A numeric vector of the predictor variable (time).
`A`	A numeric parameter for the starting value of the response variable (the `t = 0` intercept).
`B`	A numeric parameter for the asymptote of the *fast* component; the value the fast response alone would approach.
`tau`	A numeric parameter for the *fast* time constant (τ_1), in units of the predictor variable `t`. Dominates the initial steep response.
`B2`	A numeric parameter for the asymptote of the *slow* component; the stable plateau the response recovers toward as `t` approaches infinity.
`tau2`	A numeric parameter for the *slow* time constant (τ_2), in units of the predictor variable `t`. Typically `tau2 >> tau`.
`TD`	A numeric parameter for the *time delay* before the onset of the response, in units of the predictor variable `t`. If `NULL` (*default*), a 5-parameter model without time delay is used.

Details

Model formulas:

- 5-parameter: `x ~ SSbiexponential(t, A, B, tau, B2, tau2)`
- 6-parameter: `x ~ SSbiexponential(t, A, B, tau, B2, tau2, TD)`

The two phases are weakly identified when `tau` and `tau2` are close, so `algorithm = "port"` with the time constants bounded non-negative and `control = nls.control(warnOnly = TRUE)` is recommended. `analyse_kinetics()` instead fits the phases sequentially, holding the fast phase near a monoexponential estimate.

The 5-parameter form is recommended for small samples or when no obvious time delay is expected, as it converges more reliably. `stats::nls()` reads the free parameters from the formula right-hand side, so omitting `TD` incurs no degrees-of-freedom penalty.

Starting estimates are profiled on a coarse grid of `tau`, `tau2` (and `TD`) with the amplitudes solved by least squares at each grid point, keeping the residual-minimising start. Grid pairs with `tau / tau2 > 0.98` are dropped as near-collinear.

The model function returns the analytic gradient for the free parameters as a `"gradient"` attribute, so `stats::nls()` does not resort to `stats::numericDeriv()`. `stats::predict()` on a fitted model carries the attribute; drop it with `as.vector()`.

Fixing parameters:

Any parameter may be held constant by writing a value in place of its name in the formula, e.g. `x ~ SSbiexponential(t, A, B, tau = 5, B2, tau2)` holds the fast time constant at 5. Fixed parameters are excluded from estimation and are not returned by `stats::coef()`.

Value

A numeric vector of predicted values the same length as the predictor variable `t`.

See Also

`biexponential()`, `analyse_kinetics()`, `stats::nls()`, `stats::selfStart()`, `SSmonoexponential()`, `SSexponential_drift()`

Examples

```
## create a biexponential excursion-recovery curve with random noise
set.seed(13)
t <- 0:120
x <- biexponential(t, A = 70, B = 40, tau = 5, B2 = 60, tau2 = 40) +
    rnorm(length(t), 0, 0.8)
data <- data.frame(t, x)

## 5-parameter fit
model <- nls(
    x ~ SSbiexponential(t, A, B, tau, B2, tau2),
    data = data,
    algorithm = "port",
    lower = c(-Inf, -Inf, 0, -Inf, 0),
    control = nls.control(warnOnly = TRUE)
)
summary(model)

## fix the fast time constant `tau` at a known value
model_fixed <- nls(
    x ~ SSbiexponential(t, A, B, tau = 5, B2, tau2),
    data = data,
    algorithm = "port",
    lower = c(-Inf, -Inf, -Inf, 0),
    control = nls.control(warnOnly = TRUE)
)
summary(model_fixed)
```

`SSexponential_drift` *Self-starting exponential-drift model*

Description

Creates initial coefficient estimates for a `selfStart` wrapper around `exponential_drift()`, for use with `stats::nls()`. Supports both the 5-parameter (A, B, tau, slope_B, drift_fraction) and 6-parameter forms adding a time delay TD; arity is inferred from the formula passed to `stats::nls()`.

Usage

```
SSexponential_drift(t, A, B, tau, slope_B, drift_fraction, TD)
```

Arguments

`t`	A numeric vector of the predictor variable (time).
`A`	A numeric parameter for the starting baseline of the response variable.
`B`	A numeric parameter for the ending asymptote of the response variable.
`tau`	A numeric parameter for the *time constant* (τ) of the exponential response, in units of the predictor variable `t`.
`slope_B`	A numeric parameter for the linear drift rate `dx/dt` of the secondary phase, in response units per unit of the predictor variable `t`.
`drift_fraction`	A numeric fraction of the primary amplitude `B - A` in `(0.5, 1)` at which the linear drift begins, where the primary response reaches `A + drift_fraction * (B - A)`.
`TD`	A numeric parameter for the *time delay* before the onset of the exponential response, in units of the predictor variable `t`. If `NULL` (*default*), a 3-parameter model without time delay is used.

Details

Model formulas:

- 5-parameter: `x ~ SSexponential_drift(t, A, B, tau, slope_B, drift_fraction)`
- 6-parameter: `x ~ SSexponential_drift(t, A, B, tau, slope_B, drift_fraction, TD)`

The hinge at the drift onset `TD - tau * log(1 - drift_fraction)` is not differentiable, so `algorithm = "port"` with `tau` (and `TD`) bounded non-negative and `control = nls.control(warnOnly = TRUE)` is recommended.

Starting estimates are profiled on a coarse grid of `tau` (and `TD`) with `A`, `B`, and `slope_B` solved by least squares at each grid point, keeping the residual-minimising start.

The model function returns the analytic gradient (one-sided at the hinge) for the free parameters as a `"gradient"` attribute, so `stats::nls()` does not resort to `stats::numericDeriv()`. `stats::predict()` on a fitted model carries the attribute; drop it with `as.vector()`.

Fixing parameters:

Any parameter may be held constant by writing a value in place of its name in the formula, e.g. `x ~ SSexponential_drift(t, A, B, tau, slope_B, drift_fraction = 0.95)` holds the drift onset at 95% of the amplitude (`TD + 3 * tau`). Fixed parameters are excluded from estimation and are not returned by `stats::coef()`.

Value

A numeric vector of predicted values the same length as the predictor variable `t`.

See Also

`exponential_drift()`, `analyse_kinetics()`, `stats::nls()`, `stats::selfStart()`, `SSmonoexponential()`, `SSbiexponential()`

Examples

```
## create an exponential curve with late linear drift and random noise
set.seed(13)
t <- 1:180
x <- exponential_drift(
    t, A = 10, B = 100, tau = 12,
    slope_B = -0.5, drift_fraction = 0.98, TD = 15
) + rnorm(length(t), 0, 2)
data <- data.frame(t, x)

## 6-parameter fit with the drift onset held at 98% of the amplitude
model <- nls(
    x ~ SSexponential_drift(
        t, A, B, tau, slope_B, drift_fraction = 0.98, TD
    ),
    data = data,
    algorithm = "port",
    lower = c(-Inf, -Inf, 0, -Inf, 0),
    control = nls.control(warnOnly = TRUE)
)
summary(model)
```

SSgompertz *Self-starting Gompertz models*

Description

Creates initial coefficient estimates for `selfStart` wrappers around `gompertz()` and `gompertz_left()`, for use with `stats::nls()`. Both wrappers use the same 4-parameter (A, B, xmid, slope) interface.

Usage

```
SSgompertz(t, A, B, xmid, slope)

SSgompertz_left(t, A, B, xmid, slope)

SSgompertz_left(t, A, B, xmid, slope)
```

Arguments

t	A numeric vector of the predictor variable (time).
A	A numeric parameter for the starting asymptote of the response variable.
B	A numeric parameter for the ending asymptote of the response variable.
xmid	A numeric parameter for the time at the *inflection point* (the steepest point) of the curve, in units of the predictor variable t.
slope	A numeric parameter for the response rate dx/dt at the inflection xmid.

Details

Model formulas:

- Right-Gompertz: x ~ SSgompertz(t, A, B, xmid, slope)
- Left-Gompertz: x ~ SSgompertz_left(t, A, B, xmid, slope)

Used by analyse_kinetics() with method = "sigmoidal" and shape = "gompertz" or "gompertz_left". Starting estimates locate the inflection from a smoothed first derivative. SSgompertz() masks stats::SSgompertz().

Fixing parameters:

Any parameter may be held constant by writing a value in place of its name in the formula, e.g. x ~ SSgompertz(t, A = 0, B, xmid, slope) fixes the starting asymptote at A = 0. Fixed parameters are excluded from estimation and are not returned by stats::coef().

Value

A numeric vector of predicted values the same length as the predictor variable t.

See Also

gompertz(), gompertz_left(), analyse_kinetics(), SSlogistic(), stats::nls(), stats::selfStart(), stats::SSgompertz()

Examples

```
## create a Gompertz curve with random noise
set.seed(15)
t <- 1:60
x <- gompertz(t, A = 10, B = 100, xmid = 30, slope = 4) +
    rnorm(length(t), 0, 2)
data <- data.frame(t, x)
```

```
model <- nls(x ~ SSgompertz(t, A, B, xmid, slope), data = data)
summary(model)

## fix the starting asymptote `A` at a known value
model_fixed <- nls(x ~ SSgompertz(t, A = 10, B, xmid, slope), data = data)
summary(model_fixed)

## left-Gompertz
set.seed(16)
x2 <- gompertz_left(t, A = 10, B = 100, xmid = 30, slope = 4) +
    rnorm(length(t), 0, 2)
data2 <- data.frame(t, x = x2)

model_left <- nls(x ~ SSgompertz_left(t, A, B, xmid, slope), data = data2)
summary(model_left)
```

`SSlogistic` *Self-starting logistic model*

Description

Creates initial coefficient estimates for a `selfStart` wrapper around `logistic()`, for use with `stats::nls()`. Supports both the 4-parameter symmetric (A, B, xmid, slope) and 5-parameter asymmetric (A, B, xmid, slope, asym) forms; arity is inferred from the formula passed to `stats::nls()`.

Usage

```
SSlogistic(t, A, B, xmid, slope, asym)
```

Arguments

`t`	A numeric vector of the predictor variable (time).
`A`	A numeric parameter for the starting asymptote of the response variable.
`B`	A numeric parameter for the ending asymptote of the response variable.
`xmid`	A numeric parameter for the time at the *inflection point* (the steepest point) of the curve, in units of the predictor variable `t`.
`slope`	A numeric parameter for the response rate `dx/dt` at the inflection `xmid`.
`asym`	A numeric parameter for the asymmetry index of the curve; the fraction of the amplitude `(y(xmid) - A) / (B - A)` at which the inflection `xmid` occurs, in `(0, 1)`. `asym = 0.5` is symmetric and equivalent to the 4-parameter form. If `NULL` (*default*), a symmetric 4-parameter model is used.

Details

Model formulas:

- 4-parameter: `x ~ SSlogistic(t, A, B, xmid, slope)`
- 5-parameter: `x ~ SSlogistic(t, A, B, xmid, slope, asym)`

The 4-parameter form is used by `analyse_kinetics()` with `method = "sigmoidal"` and `shape = "symmetric"`. The 5-parameter asymmetric form is retained for advanced/experimental use only; `analyse_kinetics()` instead dispatches to `SSgompertz()` / `SSgompertz_left()` for asymmetric shapes, which are more stable. `stats::nls()` reads the free parameters from the formula right-hand side, so omitting `asym` incurs no degrees-of-freedom penalty.

Fixing parameters:

Any parameter may be held constant by writing a value in place of its name in the formula, e.g. `x ~ SSlogistic(t, A = 0, B, xmid, slope)` fixes the starting asymptote at `A = 0`. Fixed parameters are excluded from estimation and are not returned by `stats::coef()`.

Value

A numeric vector of predicted values the same length as the predictor variable `t`.

See Also

`logistic()`, `analyse_kinetics()`, `stats::nls()`, `stats::selfStart()`, `stats::SSfpl()`, `SSgompertz()`

Examples

```
## create an asymmetric logistic curve with random noise
set.seed(15)
t <- 1:60
x <- logistic(t, A = 10, B = 100, xmid = 30, slope = 4, asym = 0.3) +
    rnorm(length(t), 0, 2)
data <- data.frame(t, x)

## 4-parameter fit
model4 <- nls(x ~ SSlogistic(t, A, B, xmid, slope), data = data)
summary(model4)

## 5-parameter fit on the same data
model5 <- nls(x ~ SSlogistic(t, A, B, xmid, slope, asym), data = data)
summary(model5)

## fix the starting asymptote `A` at a known value
model_fixed <- nls(x ~ SSlogistic(t, A = 10, B, xmid, slope), data = data)
summary(model_fixed)

y4 <- predict(model4, data)
y5 <- predict(model5, data)


    if (requireNamespace("ggplot2", quietly = TRUE)) {
        ggplot2::ggplot(data, ggplot2::aes(t, x)) +
```

```
        theme_mnirs() +
        ggplot2::geom_point() +
        ggplot2::geom_line(ggplot2::aes(y = y5, colour = "5-param")) +
        ggplot2::geom_line(ggplot2::aes(y = y4, colour = "4-param"))
}
```

`SSmonoexponential` *Self-starting monoexponential model*

Description

Creates initial coefficient estimates for a `selfStart` wrapper around `monoexponential()`, for use with `stats::nls()`. Supports both the 3-parameter (A, B, tau) and 4-parameter (A, B, tau, TD) forms; arity is inferred from the formula passed to `stats::nls()`.

Usage

```
SSmonoexponential(t, A, B, tau, TD)
```

Arguments

`t`	A numeric vector of the predictor variable (time).
`A`	A numeric parameter for the starting baseline of the response variable.
`B`	A numeric parameter for the ending asymptote of the response variable.
`tau`	A numeric parameter for the *time constant* (τ) of the exponential response, in units of the predictor variable `t`.
`TD`	A numeric parameter for the *time delay* before the onset of the exponential response, in units of the predictor variable `t`. If `NULL` (*default*), a 3-parameter model without time delay is used.

Details

Model formulas:

- 3-parameter: `x ~ SSmonoexponential(t, A, B, tau)`
- 4-parameter: `x ~ SSmonoexponential(t, A, B, tau, TD)`

The 3-parameter form is recommended for small samples or when no obvious time delay is expected, as it converges more reliably. `stats::nls()` reads the free parameters from the formula right-hand side, so omitting `TD` incurs no degrees-of-freedom penalty.

Starting estimates are profiled on a coarse grid of `tau` (and `TD`) with the asymptotes solved by least squares at each grid point, keeping the residual-minimising start.

The model function returns the analytic gradient for the free parameters as a `"gradient"` attribute, so `stats::nls()` does not resort to `stats::numericDeriv()`. `stats::predict()` on a fitted model carries the attribute; drop it with `as.vector()`.

Fixing parameters:

Any parameter may be held constant by writing a value in place of its name in the formula, e.g. `x ~ SSmonoexponential(t, A = 0, B, tau)` fixes the baseline at `A = 0`. Fixed parameters are excluded from estimation and are not returned by `stats::coef()`.

Value

A numeric vector of predicted values the same length as the predictor variable `t`.

See Also

`monoexponential()`, `analyse_kinetics()`, `stats::nls()`, `stats::selfStart()`, `stats::SSasymp()`

Examples

```
## create an exponential curve with random noise
set.seed(13)
t <- 1:60
x <- monoexponential(t, A = 10, B = 100, tau = 8, TD = 15) +
    rnorm(length(t), 0, 3)
data <- data.frame(t, x)

## 4-parameter fit
model4 <- nls(x ~ SSmonoexponential(t, A, B, tau, TD), data = data)
summary(model4)

## 3-parameter fit on the same data
model3 <- nls(x ~ SSmonoexponential(t, A, B, tau), data = data)
summary(model3)

## fix the baseline `A` at a known value
model_fixed <- nls(x ~ SSmonoexponential(t, A = 10, B, tau, TD), data = data)
summary(model_fixed)

y4 <- predict(model4, data)
y3 <- predict(model3, data)


    if (requireNamespace("ggplot2", quietly = TRUE)) {
        ggplot2::ggplot(data, ggplot2::aes(t, x)) +
            theme_mnirs() +
            ggplot2::geom_point() +
            ggplot2::geom_line(ggplot2::aes(y = y4, colour = "4-param")) +
            ggplot2::geom_line(ggplot2::aes(y = y3, colour = "3-param"))
    }
```

`SSsigmoidal_drift` *Self-starting sigmoidal-drift model*

Description

Creates initial coefficient estimates for a `selfStart` wrapper around `sigmoidal_drift()`, for use with `stats::nls()`: a 4-parameter sigmoid (A, B, xmid, slope) with a linear drift `slope_B` at its ending asymptote from the onset fraction `drift_fraction`.

Usage

```
SSsigmoidal_drift(t, A, B, xmid, slope, slope_B, drift_fraction, shape)
```

Arguments

`t`	A numeric vector of the predictor variable (time).
`A`	A numeric parameter for the starting asymptote of the response variable.
`B`	A numeric parameter for the ending asymptote of the response variable.
`xmid`	A numeric parameter for the time at the *inflection point* (the steepest point) of the curve, in units of the predictor variable `t`.
`slope`	A numeric parameter for the response rate `dx/dt` at the inflection `xmid`.
`slope_B`	A numeric parameter for the linear drift rate `dx/dt` of the secondary phase at the ending asymptote `B`, in response units per unit of the predictor variable `t`.
`drift_fraction`	A numeric fraction of the primary amplitude `B - A` in `(0.5, 1)` at which the linear drift begins, where the sigmoid reaches `A + drift_fraction * (B - A)`.
`shape`	Character; the 4-parameter sigmoidal shape. One of `"symmetric"` (*default*; `logistic()`), `"gompertz"` (`gompertz()`), or `"gompertz_left"` (`gompertz_left()`).

Details

Model formula:

`x ~ SSsigmoidal_drift(t, A, B, xmid, slope, slope_B, drift_fraction = 0.95, shape = "gompertz")`

`drift_fraction` should be written as a constant, and `shape` is a string constant (`"symmetric"` when omitted); neither is estimated. The hinge at the drift onset is not differentiable, so `algorithm = "port"` with `control = nls.control(warnOnly = TRUE)` is recommended.

Starting estimates seed the sigmoid as for `SSgompertz()`, resolve the drift onset from that seed, and regress the residual past the onset on time to seed `slope_B` and correct the asymptote `B`.

Fixing parameters:

Any parameter may be held constant by writing a value in place of its name in the formula, e.g. `x ~ SSsigmoidal_drift(t, A = 0, B, xmid, slope, slope_B, drift_fraction = 0.95)` fixes the starting asymptote at `A = 0`. Fixed parameters are excluded from estimation and are not returned by `stats::coef()`.

Value

A numeric vector of predicted values the same length as the predictor variable `t`.

See Also

`sigmoidal_drift()`, `analyse_kinetics()`, `stats::nls()`, `stats::selfStart()`, `SSlogistic()`, `SSgompertz()`, `SSexponential_drift()`

Examples

```
## create a Gompertz curve with late linear drift and random noise
set.seed(13)
t <- 1:120
x <- sigmoidal_drift(
    t, A = 10, B = 100, xmid = 40, slope = 4,
    slope_B = -0.4, drift_fraction = 0.95, shape = "gompertz"
) + rnorm(length(t), 0, 2)
data <- data.frame(t, x)

## fit with the drift onset held at 95% of the amplitude
model <- nls(
    x ~ SSsigmoidal_drift(
        t, A, B, xmid, slope, slope_B,
        drift_fraction = 0.95, shape = "gompertz"
    ),
    data = data,
    algorithm = "port",
    control = nls.control(warnOnly = TRUE)
)
summary(model)
```

`theme_mnirs` *Custom* mnirs *ggplot2 theme*

Description

A [ggplot2][ggplot2::ggplot2-package] theme for display.

Usage

```
theme_mnirs(
  base_size = 14,
  base_family = "sans",
  border = c("partial", "full"),
  ink = "black",
  paper = "white",
  accent = "#0080ff",
  ...
)
```

Arguments

`base_size`	Base font size, given in pts.
`base_family`	Base font family.
`border`	Define either a *partial* or *full* border around plots.
`ink`	Colour for text and lines. *Default* is *"black"*.
`paper`	Background colour. *Default* is *"white"*.
`accent`	Accent colour for highlights. *Default* is *"#0080ff"*.
`...`	Additional arguments to add to `[ggplot2::theme()]`.

Details

- `axis.title = element_text(face = "bold")` by *default* Modify to *"plain"*.
- `panel.grid.major` & `panel.grid.major` set to blank. Modify to `= element_line()` for visible grid lines.
- `legend.position = "top"` by *default* Modify `"none"` to remove legend entirely.
- `border = "partial"` uses `panel.border = element_blank()` and `axis.line = element_line()`.
- `border = "full"` uses `panel.border = element_rect(colour = "black", linewidth = 1)` and `axis.line = element_line()`.
- `base_family = "sans"` by *default*.

Value

A ggplot2 theme object.

See Also

`palette_mnirs()`, `scale_colour_mnirs()`

Examples

```
## plot example data
read_mnirs(
    file_path = example_mnirs("moxy_ramp"),
    nirs_channels = c(smo2_left = "SmO2 Live",
                      smo2_right = "SmO2 Live(2)"),
    time_channel = c(time = "hh:mm:ss"),
    verbose = FALSE
) |>
    plot(time_labels = TRUE)
```

`train.red_intervals.csv` *10 Hz Train.Red App export*

Description

Exported from Train.Red app, recorded at 10 Hz. Containing two 5-minute cycling work intervals, placed on bilateral vastus lateralis muscle sites. Some data channels have been omitted to reduce file size.

Format

.csv file with header metadata and 10 columns and 11995 rows:

Timestamp (seconds passed) Elapsed time (s).

Lap/Event Lap number (numeric).

SmO2 Muscle oxygen saturation, filtered (%). Two channels have duplicated names. If both are called, the second will be renamed to `SmO2_1`.

SmO2 unfiltered Muscle oxygen saturation, raw signal (%). Two channels have duplicated names. If both are called, the second will be renamed to `SmO2 unfiltered_1`.

O2HB unfiltered Oxyhaemoglobin concentration, raw signal (arbitrary units). Two channels have duplicated names. If both are called, the second will be renamed to `O2HB unfiltered_1`.

HHB unfiltered Deoxyhaemoglobin concentration, raw signal (arbitrary units). Two channels have duplicated names. If both are called, the second will be renamed to `HHb unfiltered_1`.

Channel mapping for `read_mnirs()`:

- `nirs_channels = c( "SmO2", "SmO2 unfiltered", "O2HB unfiltered", "HHb unfiltered" )`
- `time_channel = c("Timestamp (seconds passed)")`
- `event_channel = c("Lap/Event")`
- `interval_times = list( start = c(2150.09, 2872.28), end = c(2452.26, 3167.98) )`
- `interval_times = list(  ## from zero_time start = c(65.94, 788.13), end = c(368.11, 1083.83) )`

Source

Train.Red (Train.Red B.V.), exported via Train.Red app (https://train.red/)

See Also

`read_mnirs()`, `example_mnirs()`

Examples

```
example_mnirs("train.red")
```

Index