Package ‘gcplyr’

July 22, 2025

Type Package

Title Wrangle and Analyze Growth Curve Data

Version 1.11.0

Date 2025-01-16

Description Easy wrangling and model-free analysis of
microbial growth curve data, as commonly output by plate readers.
Tools for reshaping common plate reader outputs into 'tidy' formats and
merging them with design information, making data easy to work with using
'gcplyr' and other packages. Also streamlines common growth curve
processing steps, like smoothing and calculating derivatives, and
facilitates model-free characterization and analysis of growth data.
See methods at <`https://mikeblazanin.github.io/gcplyr/`>.

License MIT + file LICENSE

URL `https://mikeblazanin.github.io/gcplyr/`,
`https://github.com/mikeblazanin/gcplyr/`

Depends R (>= 2.10)

Imports dplyr, rlang, stats, tidyr, tools, utils

Suggests caret, ggplot2, knitr, lubridate, mgcv, readxl, rmarkdown,
sf, testthat (>= 3.0.0), xlsx

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

LazyData true

RoxygenNote 7.3.2

NeedsCompilation no

Author Mike Blazanin [aut, cre] (ORCID:
<`https://orcid.org/0000-0003-4630-6235`>)

Maintainer Mike Blazanin <`mikeblazanin@gmail.com`>

Repository CRAN

Date/Publication 2025-01-17 00:00:02 UTC

Contents

`auc` *Calculate area under the curve*

Description

This function takes a vector of `x` and `y` values and returns a scalar for the area under the curve, calculated using the trapezoid rule

Usage

```
auc(
  x,
  y,
  xlim = NULL,
  blank = 0,
  subset = NULL,
  na.rm = TRUE,
  neg.rm = FALSE,
  warn_xlim_out_of_range = TRUE,
  warn_negative_y = TRUE
)
```

Arguments

`x`	Numeric vector of x values
`y`	Numeric vector of y values
`xlim`	Vector, of length 2, delimiting the x range over which the area under the curve should be calculated (where NA can be provided for the area to be calculated from the start or to the end of the data)
`blank`	Value to be subtracted from `y` values before calculating area under the curve
`subset`	A vector of logical values indicating which x and y values should be included (TRUE) or excluded (FALSE).
`na.rm`	a logical indicating whether missing values should be removed
`neg.rm`	a logical indicating whether `y` values below zero should be treated as zeros. If `FALSE`, area under the curve for negative `y` values will be calculated normally, effectively subtracting from the returned value.
`warn_xlim_out_of_range`	logical whether warning should be issued when xlim is lower than the lowest x value or higher than the highest x value.
`warn_negative_y`	logical whether warning should be issued when `neg.rm == FALSE` but some y values are below 0.

Value

A scalar for the total area under the curve

`block_tidydesign` *Turn tidydesign into block format*

Description

This function allows users to convert designs created with tidydesign into a block format for easy output to csv for inclusion in lab notebooks, etc in a human-readable format

Usage

```
block_tidydesign(
  tidydesign,
  collapse = NULL,
  wellnames_sep = "_",
  wellnames_colname = "Well"
)
```

Arguments

`tidydesign`	A tidydesign data.frame (e.g. as created by make_tidydesign)
`collapse`	NULL or a string to use for concatenating design elements together. If NULL each design column will be put into its own block. If a string, that string will be used to paste together all design elements and all design elements will be returned in a single block
`wellnames_sep`	A string used when concatenating rownames and column names to create well names
`wellnames_colname`	Header for newly-created column containing the well names

Value

A list of blockdesign data.frames (if `collapse` is not `NULL` the list is of length 1

`calc_deriv` *Calculate derivatives of vector of data*

Description

Provided a vector of y values, this function returns either the plain or per-capita difference or derivative between sequential values

Usage

```
calc_deriv(
  y,
  x = NULL,
  return = "derivative",
  percapita = FALSE,
  x_scale = 1,
  blank = NULL,
  subset_by = NULL,
  window_width = NULL,
  window_width_n = NULL,
  window_width_frac = NULL,
  window_width_n_frac = NULL,
  trans_y = "linear",
  na.rm = TRUE,
  warn_ungrouped = TRUE,
  warn_logtransform_warnings = TRUE,
  warn_logtransform_infinite = TRUE,
  warn_window_toosmall = TRUE
)
```

Arguments

`y`	Data to calculate difference or derivative of
`x`	Vector of x values provided as a simple numeric.
`return`	One of c("difference", "derivative") for whether the differences in `y` should be returned, or the derivative of `y` with respect to `x`
`percapita`	When percapita = TRUE, the per-capita difference or derivative is returned
`x_scale`	Numeric to scale x by in derivative calculation Set x_scale to the ratio of the units of x to the desired units. E.g. if x is in seconds, but the desired derivative is in units of /minute, set `x_scale = 60` (since there are 60 seconds in 1 minute).
`blank`	y-value associated with a "blank" where the density is 0. Is required when `percapita = TRUE`. If a vector of blank values is specified, blank values are assumed to be in the same order as unique(subset_by)
`subset_by`	An optional vector as long as `y`. `y` will be split by the unique values of this vector and the derivative for each group will be calculated independently of the others. This provides an internally-implemented approach similar to group_by and mutate
`window_width`, `window_width_n`, `window_width_frac`, `window_width_n_frac`	Set how many data points are used to determine the slope at each point. When all are `NULL`, `calc_deriv` calculates the difference or derivative of each point with the next point, appending `NA` at the end. When one or multiple are specified, a linear regression is fit to all points in the window to determine the slope.

	`window_width_n` specifies the width of the window in number of data points. `window_width` specifies the width of the window in units of `x`. `window_width_n_frac` specifies the width of the window as a fraction of the total number of data points. When using multiple window specifications at the same time, windows are conservative. Points included in each window will meet all of the `window_width`, `window_width_n`, and `window_width_n_frac`. A value of `window_width_n = 3` or `window_width_n = 5` is often a good default.
`trans_y`	One of `c("linear", "log")` specifying the transformation of y-values. `'log'` is only available when calculating per-capita derivatives using a fitting approach (when non-default values are specified for `window_width` or `window_width_n`). For per-capita growth expected to be exponential or nearly-exponential, `"log"` is recommended, since exponential growth is linear when log-transformed. However, log-transformations must be used with care, since y-values at or below 0 will become undefined and results will be more sensitive to incorrect values of `blank`.
`na.rm`	logical whether NA's should be removed before analyzing
`warn_ungrouped`	logical whether warning should be issued when `calc_deriv` is being called on ungrouped data and `subset_by = NULL`.
`warn_logtransform_warnings`	logical whether warning should be issued when log(y) produced warnings.
`warn_logtransform_infinite`	logical whether warning should be issued when log(y) produced infinite values that will be treated as `NA`.
`warn_window_toosmall`	logical whether warning should be issued when only one data point is in the window set by `window_width_n`, `window_width`, or `window_width_n_frac`, and so `NA` will be returned.

Details

For per-capita derivatives, `trans_y = 'linear'` and `trans_y = 'log'` approach the same value as time resolution increases.

For instance, let's assume exponential growth $N = e^rt$ with per-capita growth rate r.

With `trans_y = 'linear'`, note that $dN/dt = re^rt = rN$. So we can calculate per-capita growth rate as $r = dN/dt * 1/N$.

With `trans_y = 'log'`, note that $log(N) = log(e^rt) = rt$. So we can calculate per-capita growth rate as the slope of a linear fit of $log(N)$ against time, $r = log(N)/t$.

Value

A vector of values for the plain (if `percapita = FALSE`) or per-capita (if `percapita = TRUE`) difference (if `return = "difference"`) or derivative (if `return = "derivative"`) between `y` values. Vector will be the same length as `y`, with `NA` values at the ends

---

`CentroidFunctions` *Calculate centroid*

Description

This function takes a vector of `x` and `y` values and returns the x and/or y position of the centroid of mass of the area under the curve

Usage

```
centroid(
  x,
  y,
  return,
  xlim = NULL,
  blank = 0,
  subset = NULL,
  na.rm = TRUE,
  neg.rm = FALSE,
  warn_xlim_out_of_range = TRUE,
  warn_negative_y = TRUE
)

centroid_x(x, y, return = "x", ...)

centroid_y(x, y, return = "y", ...)

centroid_both(x, y, return = "both", ...)
```

Arguments

`x`	Numeric vector of x values
`y`	Numeric vector of y values
`return`	One of c("x", "y", "both"), determining whether the function will return the x value of the centroid, the y value of the centroid, or a vector containing x then y
`xlim`	Vector, of length 2, delimiting the x range over which the centroid should be calculated (where NA can be provided for the area to be calculated from the start or to the end of the data)
`blank`	Value to be subtracted from `y` values before calculating the centroid
`subset`	A vector of logical values indicating which x and y values should be included (TRUE) or excluded (FALSE).
`na.rm`	a logical indicating whether missing values should be removed
`neg.rm`	a logical indicating whether `y` values below zero should be treated as zeros. If `FALSE`, the centroid for negative `y` values will be calculated normally, effectively pulling the centroid towards the x axis.

`warn_xlim_out_of_range`	logical whether warning should be issued when xlim is lower than the lowest x value or higher than the highest x value.
`warn_negative_y`	logical whether warning should be issued when `neg.rm == FALSE` but some y values are below 0.
`...`	Other arguments to pass to `centroid`

Details

This function uses st_centroid to calculate the centroid of mass

Value

A scalar for the x value (if `return = 'x'`) or y value (if `return = 'y'`) of the centroid of the data

---

`doubling_time` *Calculate doubling time equivalent of per-capita growth rate*

Description

Provided a vector of per-capita growth rates, this function returns the vector of equivalent doubling times

Usage

```
doubling_time(y, x_scale = 1)
```

Arguments

`y`	Vector of per-capita derivative data to calculate the equivalent doubling time of
`x_scale`	Numeric to scale per-capita derivative values by Set x_scale to the ratio of the the units of y to the desired units. E.g. if y is in per-second, but the desired doubling time is in minutes, `x_scale = 60` (since there are 60 seconds in 1 minute).

Value

A vector of values for the doubling time equivalent to the per-capita growth rate supplied for `y`

`example_design_tidy` *Design for example growth curve data A tidy-shaped dataset with the experimental design (i.e. plate layout) for the example data included with* `gcplyr`.

Description

Wells A1...A8 through F1...F8 contain 48 different simulated bacterial strains growing alone. Wells G1...G8 through L1...L8 contain the same 48 bacterial strains in an identical layout, but this time growing in the presence of a phage

Usage

```
example_design_tidy
```

Format

A dataframe with 96 rows and 3 variables:

Well The well of the plate

Bacteria_strain The numbered bacterial strain growing in each well

Phage Whether or not the bacteria were simulated growing with phages

---

`example_widedata` *Example noisy growth curve data in wide format*

Description

A dataset containing example growth of 96 wells of simulated bacteria or bacteria and phages

Wells A1...A8 through F1...F8 contain 48 different simulated bacterial strains growing alone. Wells G1...G8 through L1...L8 contain the same 48 bacterial strains in an identical layout, but this time growing in the presence of a phage

Usage

```
example_widedata
```

Format

A dataframe with 97 rows and 97 variables:

time time, in seconds, since growth curve began

A1, A2...H11, H12 bacterial density in the given well

Details

Bacterial populations exhibit diauxic growth as they approach their carrying capacity, and they also evolve resistance in the face of selection from the phage population.

This data includes some simulated noise to approximate the noise generated during data collection by plate readers

---

`example_widedata_noiseless`

Example growth curve data in wide format

Description

A dataset containing example growth of 96 wells of simulated bacteria or bacteria and phages

Wells A1...A8 through F1...F8 contain 48 different simulated bacterial strains growing alone. Wells G1...G8 through L1...L8 contain the same 48 bacterial strains in an identical layout, but this time growing in the presence of a phage

Usage

```
example_widedata_noiseless
```

Format

A dataframe with 97 rows and 97 variables:

time time, in seconds, since growth curve began

A1, A2...H11, H12 bacterial density in the given well

Details

Bacterial populations exhibit diauxic growth as they approach their carrying capacity, and they also evolve resistance in the face of selection from the phage population.

This data does not include any simulated noise

`ExtremaFunctions` *Find local extrema of a numeric vector*

Description

These functions take a vector of `y` values and identify local extrema.

Usage

```
find_local_extrema(
  y,
  x = NULL,
  window_width = NULL,
  window_width_n = NULL,
  window_height = NULL,
  window_width_frac = NULL,
  window_width_n_frac = NULL,
  return = "index",
  return_maxima = TRUE,
  return_minima = TRUE,
  return_endpoints = TRUE,
  subset = NULL,
  na.rm = TRUE,
  width_limit = NULL,
  width_limit_n = NULL,
  height_limit = NULL
)

first_maxima(
  y,
  x = NULL,
  window_width = NULL,
  window_width_n = NULL,
  window_height = NULL,
  window_width_frac = NULL,
  window_width_n_frac = 0.2,
  return = "index",
  return_endpoints = TRUE,
  ...
)

first_minima(
  y,
  x = NULL,
  window_width = NULL,
  window_width_n = NULL,
  window_height = NULL,
```

```
  window_width_frac = NULL,
  window_width_n_frac = 0.2,
  return = "index",
  return_endpoints = TRUE,
  ...
)
```

Arguments

`y`	Numeric vector of y values in which to identify local extrema
`x`	Optional numeric vector of corresponding x values
`window_width`, `window_width_n`, `window_height`, `window_width_frac`, `window_width_n_frac`	Arguments that set the width/height of the window used to search for local extrema. `window_width` is in units of `x`. `window_width_n` is in units of number of data points. `window_height` is the maximum change in `y` a single extrema-search step is allowed to take. `window_width_n_frac` is as a fraction of the total number of data points. For example, the function will not pass a peak or valley more than `window_width_n` data points wide, nor a peak/valley taller or deeper than `window_height`. A narrower width will be more sensitive to narrow local maxima/minima, while a wider width will be less sensitive to local maxima/minima. A smaller height will be more sensitive to shallow local maxima/minima, while a larger height will be less sensitive to shallow maxima/minima.
`return`	One of c("index", "x", "y"), determining whether the function will return the index, x value, or y value associated with the identified extremas
`return_maxima`, `return_minima`	logical for which classes of local extrema to return
`return_endpoints`	Should the first and last values in `y` be included if they are in the returned vector of extrema?
`subset`	A vector of logical values indicating which x and y values should be included (TRUE) or excluded (FALSE). If `return = "index"`, index will be for the whole vector and not the subset of the vector
`na.rm`	logical whether NA's should be removed before analyzing
`width_limit`	Deprecated, use `window_width` instead
`width_limit_n`	Deprecated, use `window_width_n` instead
`height_limit`	Deprecated, use `window_height` instead
`...`	(for `first_maxima` and `first_minima`), other parameters to pass to `find_local_extrema`

Details

For `find_local_extrema`, one of `window_width`, `window_width_n`, `window_height`, or `window_width_n_frac` must be provided.

For `first_minima` or `first_maxima`, set `window_width_n_frac = NULL` to override default width behavior.

If multiple of `window_width`, `window_width_n`, `window_height`, or `window_width_n_frac` are provided, steps are limited conservatively (a single step must meet all criteria).

In the case of exact ties in `y` values within a window, only the first local extrema is returned.

Value

`find_local_extrema` returns a vector corresponding to all the found local extrema.

`first_maxima` returns only the first maxima, so is a shortcut for `find_local_extrema(return_maxima = TRUE, return_minima = FALSE)[1]`

`first_minima` returns only the first minima, so is a shortcut for `find_local_extrema(return_maxima = FALSE, return_maxima = FALSE)[1]`

If `return = "index"`, the returned value(s) are the indices corresponding to local extrema in the data

If `return = "x"`, the returned value(s) are the x value(s) corresponding to local extrema in the data

If `return = "y"`, the returned value(s) are the y value(s) corresponding to local extrema in the data

---

`extr_val` *Extract parts of an object*

Description

A wrapper for `[` with handling of NA's for use in `dplyr::summarize()`

Usage

```
extr_val(x, i, allNA_NA = TRUE, na.rm = TRUE)
```

Arguments

`x`	object from which to extract element(s)
`i`	index specifying element to extract.
`allNA_NA`	logical indicating whether `NA` should be returned when `all(is.na(i)) == TRUE`.
`na.rm`	a logical indicating whether missing index values should be removed.

Value

If `all_NA = FALSE` and `na.rm = FALSE`, identical to `x[i]`.

If `all_NA = FALSE` and `na.rm = TRUE`, identical to `x[i[!is.na(i)]]`.

If `all_NA = TRUE`, identical to `x[i]` unless `all(is.na(i)) == TRUE`, in which case returns `NA`

`first_peak` *Find the first local maxima of a numeric vector*

Description

This function has been deprecated in favor of the identical new function first_maxima

Usage

```
first_peak(
  y,
  x = NULL,
  window_width = NULL,
  window_width_n = NULL,
  window_height = NULL,
  return = "index",
  return_endpoints = TRUE,
  ...
)
```

Arguments

`y`	Numeric vector of y values in which to identify local extrema
`x`	Optional numeric vector of corresponding x values
`window_width`	Width of the window (in units of `x`) used to search for local extrema. A narrower width will be more sensitive to narrow local maxima/minima, while a wider width will be less sensitive to local maxima/minima.
`window_width_n`	The maximum number of data points a single extrema-search step is allowed to take. For example, when maxima-finding, the function will not pass a valley consisting of more than `window_width_n` data points. A smaller `window_width_n` will be more sensitive to narrow local maxima/minima, while a larger `window_width_n` will be less sensitive to narrow local maxima/minima. If not provided, defaults to ~0.2*length(y)
`window_height`	The maximum change in `y` a single extrema-search step is allowed to take. For example, when maxima-finding, the function will not pass a valley deeper than `window_height`. A smaller `window_height` will be more sensitive to shallow local maxima/minima, while a larger `window_height` will be less sensitive to shallow maxima/minima.
`return`	One of c("index", "x", "y"), determining whether the function will return the index, x value, or y value associated with the first maxima in y values
`return_endpoints`	Should the first or last value in `y` be allowed to be returned?
`...`	Other parameters to pass to find_local_extrema

Details

This function takes a vector of `y` values and returns the index (by default) of the first local maxima. It serves as a shortcut for `find_local_extrema(return_maxima = TRUE, return_minima = FALSE)[1]`

If none of `window_width`, `window_width_n`, or `window_height` are provided, default value of `window_width_n` will be used.

Value

If `return = "index"`, a vector of indices corresponding to local extrema in the data

If `return = "x"`, a vector of x values corresponding to local extrema in the data

If `return = "y"`, a vector of y values corresponding to local extrema in the data

See Also

[first_maxima()]

`from_excel` *A function that converts base-26 Excel-style letters to numbers*

Description

A function that converts base-26 Excel-style letters to numbers

Usage

```
from_excel(x)
```

Arguments

`x`	A vector of column names in Excel-style base-26 letter format (any values that are already in base-10 will be returned as-is)

Value

A vector of numbers in base-10

`gc_smooth.spline` *Fit a Smoothing Spline*

Description

This function is a wrapper for smooth.spline, which fits a cubic smoothing spline to the supplied data, but includes the option to remove `NA` values, and returns values in the original order.

Usage

```
gc_smooth.spline(x, y = NULL, ..., na.rm = TRUE)
```

Arguments

`x`	A vector giving the values of the predictor variable.
`y`	A vector giving the values of the response variable. If `y` is missing or `NULL`, the responses are assumed to be specified by `x`, with `x` the index vector.
`...`	Additional arguments passed to smooth.spline.
`na.rm`	logical whether NA's should be removed before analyzing. Required to be TRUE if any `x` or `y` values are NA.

Details

See smooth.spline

Value

Similar to smooth.spline, an object of class `"smooth.spline"` with many components. Differs in that x, y, and w have NA's at any indices where `x` or `y` were NA in the inputs, and x, y, and w are returned to match the input `x` in order and length

---

`import_blockdesigns` *Import blockdesigns*

Description

Function to import block-shaped designs from files and return tidy designs. This function acts as a wrapper that calls read_blocks, paste_blocks, trans_block_to_wide, trans_wide_to_tidy, and separate_tidy

Usage

```
import_blockdesigns(
  files,
  block_names = NULL,
  block_name_header = "block_name",
  join_as_cols = TRUE,
  sep = NULL,
  values_colname = "Designs",
  into = NULL,
  keep_blocknames = !join_as_cols,
  warn_joinrows_nointo = TRUE,
  join_designs = NULL,
  ...
)
```

Arguments

`files`	A vector of filepaths relative to the current working directory where each filepath is a single plate read to be read by read_blocks.
`block_names`	Vector of names corresponding to each design element (each block). Inferred from filenames, if not specified. When `keep_blocknames` = `TRUE`, the output will have a column containing these values, with the column name specified by `block_name_header`. When `join_as_cols` = `TRUE`, the `block_names` are also used as the output column names for each separated design column.
`block_name_header`	When `keep_blocknames` = `TRUE`, the column name of the column containing the `block_names`.
`join_as_cols`	logical indicating whether blocks (if there are multiple) should be joined as columns (i.e. describe the same plate) in the tidy output. If `FALSE`, blocks are joined as rows (i.e. describe different plates) in the tidy output.
`sep`	If designs have been pasted together, this specifies the string they should be split apart by via separate_tidy.
`values_colname`	When `join_as_cols` = `FALSE` and `sep` is not specified, all the design values will be in a column named by `values_colname`. For other cases, see the **Value** section.
`into`	When `sep` is specified, `into` sets the names of the columns after splitting (see **Value** section for behavior when `into` is not set).
`keep_blocknames`	logical indicating whether the column containing `block_names` (or those inferred from file names) should be retained in the output. By default, blocknames are retained only if `join_as_cols` = `FALSE`.
`warn_joinrows_nointo`	logical indicating whether warning should be raised when multiple blocks are joined as rows (`join_as_cols` = `FALSE`) and `sep` is specified, but `into` is not specified.

`join_designs`	Deprecated, use `join_as_cols` instead
`...`	Other arguments to pass to read_blocks, paste_blocks, trans_block_to_wide, trans_wide_to_tidy, or separate_tidy. See Details for more information

Details

Other common arguments that you may want to provide via `...` include:

`startrow`, `endrow`, `startcol`, `endcol`, `sheet` - specifying the location of design information inside `files` to read_blocks.

`wellnames_sep` - specifying what character (or "" for none) should be used when pasting together the rownames and column names. Note that this should be chosen to match the well names in your measures.

`into` - specifying the column names resulting from using separate_tidy on the values_colname column.

Note that `import_blockdesigns` cannot currently handle metadata specified via the `metadata` argument of read_blocks.

If you find yourself needing more control, you can run the steps manually, first reading with read_blocks, pasting as needed with paste_blocks, transforming to tidy with trans_block_to_wide and trans_wide_to_tidy, and separating as needed with separate_tidy.

Value

A tidy-shaped `data.frame` containing the design information from `files`. This always includes a "Well" column.

If `keep_blocknames = TRUE`, this includes a column with the column name specified by `block_name_header` and containing `block_names` (or block names inferred from file names).

The layout of the design values varies depending on the inputs:

If `join_as_cols = TRUE`, each block was joined as a column, with the columns named according to `block_names` (or block names inferred from file names). In this case, if `sep` was specified, each column was split by `sep` into columns named by splitting the corresponding block name by `sep` (post-split column names can alternatively be specified directly via `into`).

Otherwise, when `join_as_cols = FALSE`, each block was joined as rows, with the column containing all design values named by `values_colname`. In this case, if `sep` was specified, that single design column was split by `sep` into columns named by splitting `values_colname` (post-split column names can alternatively be specified directly via `into`).

`import_blockmeasures` *Import blockmeasures*

Description

Function to import blockmeasures from files and return widemeasures This function acts as a wrapper to call read_blocks, uninterleave, then trans_block_to_wide in one go

Usage

```
import_blockmeasures(
  files,
  num_plates = 1,
  plate_names = NULL,
  wellnames_sep = "",
  ...
)
```

Arguments

`files`	Vector of filenames (as strings), each of which is a block-shaped file containing measures data. File formats can be .csv, .xls, or .xlsx
`num_plates`	Number of plates. If multiple plates uninterleave will be used to separate blockmeasures into those plates accordingly
`plate_names`	(optional) Names to put onto the plates when output
`wellnames_sep`	String to use as separator for well names between rowname and column name
`...`	Other arguments to pass to read_blocks, uninterleave, or trans_block_to_wide

Details

Common arguments that you may want to provide via `...` include:

`startrow`, `endrow`, `startcol`, `endcol`, `sheet` - specifying the location of design information inside `files` to read_blocks

`metadata` - specifying metadata to read_blocks

See read_blocks for more details

If you find yourself needing more control, you can run the steps manually, first reading with read_blocks, separating plates as needed with uninterleave, then transforming to wide with trans_block_to_wide.

Value

If `num_plates = 1`, a wide-shaped `data.frame` containing the measures data.

if `num_plates` is greater than one, a list of `data.frame`'s, where each `data.frame` is wide-shaped.

`lag_time` *Calculate lag time*

Description

Lag time is calculated by projecting a tangent line at the point of maximum (per-capita) derivative backwards to find the time when it intersects with the minimum y-value

Usage

```
lag_time(
  x = NULL,
  y = NULL,
  deriv = NULL,
  blank = NULL,
  trans_y = "log",
  na.rm = TRUE,
  slope = NULL,
  x1 = NULL,
  y1 = NULL,
  y0 = NULL,
  warn_logtransform_warnings = TRUE,
  warn_logtransform_infinite = TRUE,
  warn_min_y_mismatch = TRUE,
  warn_multiple_maxderiv = TRUE,
  warn_one_lag = TRUE,
  warn_no_lag = TRUE,
  warn_blank_length = TRUE
)
```

Arguments

`x`	Vector of x values (typically time)
`y`	Vector of y values (typically density)
`deriv`	Vector of derivative values (typically per-capita derivative)
`blank`	y-value associated with a "blank" where the density is 0. Is required when `trans_y = TRUE`. A vector of blank values may be specified only when all of `slope`, `x1`, `y1`, and `y0` are provided as vectors
`trans_y`	One of `c("linear", "log")` specifying the transformation of y-values. `'log'` is the default, producing calculations of lag time assuming a transition to exponential growth `'linear'` is available for alternate uses
`na.rm`	a logical indicating whether missing values or values that become `NA` or infinite during log-transformation should be removed
`slope`	Slope to project from x1,y1 to y0 (typically per-capita growth rate). If not provided, will be calculated as `max(deriv)`
`x1`	x value (typically time) to project slope from. If not provided, will be calculated as `x[which.max(deriv)]`.
`y1`	y value (typically density) to project slope from. If not provided, will be calculated as `y[which.max(deriv)]`.
`y0`	y value (typically density) to find intersection of slope from `x1`, `y1` with. If not provided, will be calculated as `min(y)`

`warn_logtransform_warnings`	logical whether warning should be issued when log(y) produced warnings.
`warn_logtransform_infinite`	logical whether warning should be issued when log(y) produced infinite values that will be treated as `NA`.
`warn_min_y_mismatch`	logical whether warning should be issued when `min(y)` does not equal `min(y[!is.na(x)])`.
`warn_multiple_maxderiv`	logical whether warning should be issued when there are multiple points in `deriv` that are tied for the highest, and only the first will be used.
`warn_one_lag`	logical whether warning should be issued when some, but not all, inputs are vectorized, and only one lag time value will be returned.
`warn_no_lag`	logical whether warning should be issued when calculated lag time is less than the minimum value of x.
`warn_blank_length`	logical whether warning should be issued when an unexpected number of `blank` values was provided and only the first will be used

Details

For most typical uses, simply supply `x`, `y`, and `deriv` (using the per-capita derivative and `trans_y = 'log'`).

Advanced users may wish to use alternate values for the slope of the tangent line (`slope`), origination point of the tangent line (`x1`, `y1`), or minimum y-value `y0`. If specified, these values will override the default calculations. If and only if all of `slope`, `x1`, `y1`, and `y0` are provided, `lag_time` is vectorized on their inputs and will return a vector of lag time values.

Value

Typically a scalar of the lag time in units of `x`. See Details for cases when value will be a vector.

`makemethod_train_smooth_data`

Create method argument for train of growth curve smoothers

Description

This function generates a list which is compatible to be used as the `method` argument to train. This enables users to call train directly themselves with `smooth_data` smoothing functions.

Usage

```
makemethod_train_smooth_data(sm_method, tuneGrid = NULL)
```

Arguments

`sm_method`	Argument specifying which smoothing method should be used. Options include "moving-average", "moving-median", "loess", "gam", and "smooth.spline".
`tuneGrid`	A data frame with possible tuning value. The columns should be named the same as the tuning parameters. Note that, when using train, the tuneGrid must be passed both to this function as well as directly to train.

Value

A list that can be used as the method argument to train. Contains elements: `library`, `type`, `prob`, `fit`, `parameters`, `grid`, `fit`, and `predict`.

See documentation on using a custom model model in train for more details.

`make_design` *Make design data.frame(s)*

Description

This is a function to easily input experimental design elements for later merging with read data

Usage

```
make_design(
  nrows = NULL,
  ncols = NULL,
  block_row_names = NULL,
  block_col_names = NULL,
  block_name_header = "block_name",
  output_format = "tidy",
  wellnames_numeric = FALSE,
  wellnames_sep = "",
  wellnames_colname = "Well",
  colnames_first = FALSE,
  lookup_tbl_start = 1,
  pattern_split = "",
  ...
)
```

Arguments

`nrows, ncols`	Number of rows and columns in the plate data
`block_row_names, block_col_names`	Names of the rows, columns of the plate blockmeasures data
`block_name_header`	The name of the field containing the `block_names`

`output_format`	One of c("blocks", "blocks_pasted", "wide", "tidy") denoting the format of the resulting data.frame For easy merging with tidymeasures, leave as default of 'tidy'. For human-readability to confirm design is correct, choose 'blocks' or 'blocks_pasted'. For writing to block-shaped file(s), choose 'blocks' or 'blocks_pasted'.
`wellnames_numeric`	If `block_row_names` or `block_col_names` are not specified, then names will be generated automatically according to `wellnames_numeric`. If `wellnames_numeric` is TRUE, rows and columns will be numbered with "R" and "C" prefixes, respectively. If `wellnames_numeric` is FALSE, rows will be lettered A through Z, while columns will be numbered
`wellnames_sep`	A string used when concatenating rownames and column names to create well names, when `output_format = "wide"` or `output_format = "tidy"`
`wellnames_colname`	Header for newly-created column containing the well names, when `output_format = "tidy"`
`colnames_first`	When wellnames are created for `output_format = "wide"` or `output_format = "tidy"` by paste-ing the rownames and column names, should the column names come first.
`lookup_tbl_start`	Value in the lookup table for the split pattern values that corresponds to the first value in the vector. Lookup table by default is c(1,2,...,8,9,A,B,...Y,Z,a,b,...,y,z). If, for example, lookup_tbl_start = "A", then the lookup table will now be c(A,B,...Y,Z,a,b,...,y,z)
`pattern_split`	character to split pattern elements provided in `...` by, if they're not already a vector
`...`	Each `...` argument must be named, and must be a list with five elements: 1. a vector of the values 2. a vector of the rows the pattern should be applied to 3. a vector of the columns the pattern should be applied to 4. a string or vector denoting the pattern in which the values should be filled into the rows and columns specified. If it's a string, will be split by `pattern_split`. Pattern will be used as the indices of the values vector. 0's refer to NA. The pattern will be recycled as necessary to fill all the wells of the rows and columns specified. 5. a logical for whether this pattern should be filled byrow

Details

Note that either `nrows` or `block_row_names` must be provided and that either `ncols` or `block_col_names` must be provided

Value

Depends on `output_format`:

If `output_format = "blocks"`, a list of `data.frame`'s where each `data.frame` is block-shaped containing the information for a single design element

If `output_format = "blocks_pasted"`, a single `data.frame` containing the paste-ed information for all design elements

If `output_format = "wide"`, a wide-shaped `data.frame` containing all the design elements

If `output_format = "tidy"`, a tidy-shaped `data.frame` containing all the design elements

Examples

```
make_design(nrows = 8, ncols = 12,
            design_element_name = list(c("A", "B", "C"),
                                       2:7,
                                       2:11,
                                       "112301",
                                       TRUE))

## To be reminded what arguments are needed, use make_designpattern:
make_design(nrows = 8, ncols = 12,
            design_element_name = make_designpattern(
                 values = c("A", "B", "C"),
                 rows = 2:7,
                 cols = 2:11,
                 pattern = "112301",
                 byrow = TRUE))
```

`make_designpattern` *Make design pattern*

Description

A helper function for use with make_design

Usage

```
make_designpattern(
  values,
  rows,
  cols,
  pattern = 1:length(values),
  byrow = TRUE
)

mdp(values, rows, cols, pattern = 1:length(values), byrow = TRUE)
```

Arguments

`values`	Vector of values to use
`rows`	Vector of rows where pattern applies
`cols`	Vector of cols where pattern applies
`pattern`	Numeric pattern itself, where numbers refer to entries in `values`
`byrow`	logical for whether pattern should be created by row

Value

`list(values, rows, cols, pattern, byrow)`

See Also

[gcplyr::make_design()]

Examples

```
make_design(nrows = 8, ncols = 12,
            design_element_name = make_designpattern(
                 values = c("A", "B", "C"),
                 rows = 2:7,
                 cols = 2:11,
                 pattern = "112301",
                 byrow = TRUE))
```

`make_example` *Create R objects or files as seen in vignette examples*

Description

This function makes it easy to generate R objects or files that are created in the vignette examples. Note that this function should not be counted on to produce the same output across different versions of `gcplyr`, as it will be frequently changed to match the examples in the vignettes.

Usage

```
make_example(vignette, example, dir = ".")
```

Arguments

`vignette`	Number of the vignette the example object or file is created in.
`example`	Number of the example the object or file is created in.
`dir`	The directory files should be saved into.

Value

An R object, or the names of the files if files have been written

---

`make_tidydesign` *Make tidy design data.frames*

Description

This is a function to easily input experimental design elements for later merging with read data

Usage

```
make_tidydesign(
  nrows = NULL,
  ncols = NULL,
  block_row_names = NULL,
  block_col_names = NULL,
  wellnames_sep = "",
  wellnames_colname = "Well",
  wellnames_Excel = TRUE,
  lookup_tbl_start = 1,
  pattern_split = "",
  colnames_first = FALSE,
  ...
)
```

Arguments

`nrows, ncols`	Number of rows and columns in the plate data
`block_row_names, block_col_names`	Names of the rows, columns of the plate blockmeasures data
`wellnames_sep`	A string used when concatenating rownames and column names to create well names
`wellnames_colname`	Header for newly-created column containing the well names
`wellnames_Excel`	If `block_row_names` or `block_col_names` are not specified, should rows and columns be named using Excel-style base-26 lettering for rows and numbering for columns? If FALSE, rows and columns will be numbered with "R" and "C" prefix.
`lookup_tbl_start`	Value in the lookup table for the split pattern values that corresponds to the first value in the vector. Lookup table by default is c(1,2,...,8,9,A,B,...Y,Z,a,b,...,y,z). If, for example, lookup_tbl_start = "A", then the lookup table will now be c(A,B,...Y,Z,a,b,...,y,z)
`pattern_split`	character to split pattern elements provided in `...` by
`colnames_first`	In the wellnames created by `paste`-ing the rownames and column names, should the column names come first

`...`	Each `...` argument must be a list with five elements: 1. a vector of the values 2. a vector of the rows the pattern should be applied to 3. a vector of the columns the pattern should be applied to 4. a string of the pattern itself, where numbers refer to the indices in the values vector 0's refer to NA This pattern will be split using pattern_split, which defaults to every character 5. a logical for whether this pattern should be filled byrow

Details

Note that either `nrows` or `block_row_names` must be provided and that either `ncols` or `block_col_names` must be provided

Examples: my_example <- make_tidydesign(nrows = 8, ncols = 12, design_element_name = list(c("Value1", "Value2", "Value3"), rowstart:rowend, colstart:colend, "111222333000", TRUE) To make it easier to pass arguments, use make_designpattern: my_example <- make_tidydesign(nrows = 8, ncols = 12, design_element_name = make_designpattern(values = c("L", "G", "C"), rows = 2:7, cols = 2:11, pattern = "11223300", byrow = TRUE))

Value

a tidy-shaped `data.frame` containing all the design elements

`merge_dfs` *Collapse a list of dataframes, or merge two dataframes together*

Description

This function is essentially a wrapper for any of `dplyr`'s mutate-joins (by default, a full_join). The most typical use of this function is to merge designs with measures data, or to use the collapse functionality to merge a list of dataframes into a single dataframe. Merging is done by column names that match between `x` and `y`.

Usage

```
merge_dfs(
  x,
  y = NULL,
  by = NULL,
  drop = FALSE,
  collapse = FALSE,
  names_to = NA,
  join = "full",
  warn_morerows = TRUE,
  ...
)
```

Arguments

`x`	First data.frame, or list of data frames, to be joined
`y`	Second data.frame, or list of data frames, to be joined
`by`	A character vector of variables to join by, passed directly to the join function
`drop`	Should only `complete_cases` of the resulting data.frame be returned?
`collapse`	A logical indicating whether `x` or `y` is a list containing data frames that should be merged together before being merged with the other
`names_to`	Column name for where `names(x)` or `names(y)` will be entered in if `collapse = TRUE`. If a value of `NA` then `names(x)` or `names(y)` will not be put into a column in the returned data.frame
`join`	Type of join used to merge `x` and `y`. Options are 'full' (default), 'inner', 'left', and 'right'. • A `full` join keeps all observations in `x` and `y` • A `left` join keeps all observations in `x` • A `right` join keeps all observations in `y` • An `inner` join only keeps observations found in both `x` and `y` (inner joins are not appropriate in most cases because observations are frequently dropped). See full_join, left_join, right_join, or inner_join for more details
`warn_morerows`	logical, should a warning be passed when the output has more rows than `x` and more rows than `y`?
`...`	Other arguments to pass to the underlying join function. See full_join, left_join, right_join, or inner_join for options.

Value

Data.frame containing merged output of `x` and `y`

`MinMaxGC` *Maxima and Minima*

Description

Returns the maxima and minima of the input values.

Usage

```
max_gc(..., na.rm = TRUE, allmissing_NA = TRUE)

min_gc(..., na.rm = TRUE, allmissing_NA = TRUE)
```

Arguments

`...`	numeric or character arguments
`na.rm`	a logical indicating whether missing values should be removed.
`allmissing_NA`	a logical indicating whether `NA` should be returned when there are no non-missing arguments passed to `min` or `max` (often because `na.rm = TRUE` but all values are `NA`)

Details

These functions are wrappers for `min` and `max`, with the additional argument `allmissing_NA`.

Value

If `allmissing_NA = FALSE`, identical to `min` or `max`.

If `allmissing_NA = TRUE`, identical to `min` or `max` except that, in cases where `min` or `max` would return an infinite value and raise a warning because there are no non-missing arguments, `min_gc` and `max_gc` return `NA`

`MovingWindowFunctions` *Moving window smoothing*

Description

These functions use a moving window to smooth data

Usage

```
moving_average(
  formula = NULL,
  data = NULL,
  x = NULL,
  y = NULL,
  window_width_n = NULL,
  window_width = NULL,
  window_width_n_frac = NULL,
  window_width_frac = NULL,
  na.rm = TRUE,
  warn_nonnumeric_sort = TRUE
)

moving_median(
  formula = NULL,
  data = NULL,
  x = NULL,
  y = NULL,
  window_width_n = NULL,
```

```
    window_width = NULL,
    window_width_n_frac = NULL,
    window_width_frac = NULL,
    na.rm = TRUE,
    warn_nonnumeric_sort = TRUE
)
```

Arguments

`formula`	Formula specifying the numeric response (density) and numeric predictor (time).
`data`	Dataframe containing variables in `formula`
`x`	A vector of predictor values to smooth along (e.g. time)
`y`	A vector of response values to be smoothed (e.g. density).
`window_width_n`	Number of data points wide the moving window is (therefore, must be an odd number of points)
`window_width`	Width of the moving window (in units of `x`)
`window_width_n_frac`	Width of the window (as a fraction of the total number of data points).
`window_width_frac`	Width of the window (as a fraction of the range of `x`)
`na.rm`	logical whether NA's should be removed before analyzing
`warn_nonnumeric_sort`	logical whether warning should be issued when predictor variable that data is sorted by is non-numeric.

Details

Either `x` and `y` or `formula` and `data` must be provided.

Values of `NULL` or `NA` will be ignored for any of `window_width_n`, `window_width`, `window_width_n_frac`, or `window_width_frac`

Value

Vector of smoothed data, with NA's appended at both ends

`paste_blocks` *Paste a list of blocks into a single block*

Description

This function uses paste to concatenate the same-location entries of a list of data.frames together (i.e. all the first row-first column values are pasted together, all the second row-first column values are pasted together, etc.)

Usage

```
paste_blocks(blocks, sep = "_", nested_metadata = NULL)
```

Arguments

`blocks`	Blocks, either a single data.frame or a list of data.frames
`sep`	String to use as separator for output pasted values
`nested_metadata`	A logical indicating the existence of nested metadata in the `blockmeasures` list, e.g. as is typically output by read_blocks. If NULL, will attempt to infer existence of nested metadata

Value

If nested_metadata = TRUE (or is inferred to be TRUE), a list containing a list containing: 1. a `data.frame` with the pasted data values from `blocks`, and 2. a vector with the pasted metadata values from `blocks`

If nested_metadata = FALSE (or is inferred to be FALSE), a list containing `data.frame`'s with the pasted values from `blocks`

`predict_interpolation` *Predict data by linear interpolation from existing data*

Description

Predict data by linear interpolation from existing data

Usage

```
predict_interpolation(
  x,
  y,
  newdata,
  extrapolate_predictions = TRUE,
  na.rm = TRUE
)
```

Arguments

`x`	A vector of known predictor values.
`y`	A vector of known response values.
`newdata`	A vector of new predictor values for which the response value will be predicted
`extrapolate_predictions`	Boolean indicating whether values of `newdata` that are out of the domain of `x` should be predicted (by extrapolating the slope from the endpoints of `x`). If `FALSE`, such values will be returned as `NA`.
`na.rm`	logical whether NA's should be removed before making predictions

Value

A vector of response values for each predictor value in `newdata`

`print_df` *Nicely print the contents of a data.frame*

Description

This function uses `write.table` to print the input `data.frame` in a nicely-formatted manner that is easy to read

Usage

```
print_df(x, col.names = FALSE, row.names = FALSE)
```

Arguments

`x`	The `data.frame` to be printed
`col.names`	Boolean for whether column names should be printed
`row.names`	Boolean for whether row names should be printed

`read_blocks` *Read blocks*

Description

A function that reads blocks into the R environment

Usage

```
read_blocks(
  files,
  filetype = NULL,
  startrow = NULL,
  endrow = NULL,
  startcol = NULL,
  endcol = NULL,
  sheet = NULL,
  metadata = NULL,
  block_names = NULL,
  block_names_header = "block_name",
  block_names_dot = FALSE,
  block_names_path = TRUE,
  block_names_ext = FALSE,
  header = NA,
```

```
    sider = NA,
    wellnames_numeric = FALSE,
    na.strings = c("NA", ""),
    extension,
    block_name_header,
    ...
)
```

Arguments

`files`	A vector of filepaths relative to the current working directory where each filepath is a single plate read
`filetype`	(optional) the type(s) of the files. Options include: "csv", "xls", or "xlsx". "tbl" or "table" to use read.table to read the file, "csv2" to use read.csv2, "delim" to use read.delim, or "delim2" to use read.delim2. If none provided, `read_blocks` will infer filetype(s) from the extension(s) in `files`. When extension is not "csv", "xls", or "xlsx", will use "table".
`startrow`, `endrow`, `startcol`, `endcol`	(optional) the rows and columns where the measures data are located in `files`. Can be a vector or list the same length as `files`, or a single value that applies to all `files`. Values can be numeric or a string that will be automatically converted to numeric by from_excel. If not provided, data is presumed to begin on the first row and column of the file(s) and end on the last row and column of the file(s).
`sheet`	(optional) If data is in .xls or .xlsx files, which sheet it is located on. Defaults to the first sheet if not specified
`metadata`	(optional) non-spectrophotometric data that should be associated with each read blockmeasures. A named list where each item in the list is either: a vector of length 2, or a list containing two vectors. In the former case, each vector should provide the row and column where the metadata is located in all of the blockmeasures input files. In the latter case, the first vector should provide the rows where the metadata is located in each of the corresponding input files, and the second vector should provide the columns where the metadata is located in each of the corresponding input files. (This case is typically used when reading multiple blocks from a single file.)
`block_names`	(optional) vector of names corresponding to each plate in `files`. If not provided, block_names are inferred from the filenames
`block_names_header`	The name of the metadata field containing the `block_names`
`block_names_dot`	If block_names are inferred from filenames, should the leading ’./’ (if any) be retained
`block_names_path`	If block_names are inferred from filenames, should the path (if any) be retained

`block_names_ext`	If block_names are inferred from filenames, should the file extension (if any) be retained
`header`	`TRUE`, `FALSE`, or `NA`, or a vector of such values, indicating whether the file(s) contains the column names as its first line. If `header = NA` will attempt to infer the presence of column names. If `header = FALSE` or no column names are inferred when `header = NA`, column names will be generated automatically according to `wellnames_numeric`
`sider`	`TRUE`, `FALSE`, or `NA`, or a vector of such values, indicating whether the file(s) contains the row names as its first column. If `sider = NA` will attempt to infer the presence of row names. If `sider = FALSE` or no row names are inferred when `sider = NA`, row names will be generated automatically according to `wellnames_numeric`
`wellnames_numeric`	If row names and column names are not provided in the input dataframe as specified by `header` and `sider`, then names will be generated automatically according to `wellnames_numeric`. If `wellnames_numeric` is TRUE, rows and columns will be numbered with "R" and "C" prefixes, respectively. If `wellnames_numeric` is FALSE, rows will be lettered A through Z, while columns will be numbered
`na.strings`	A character vector of strings which are to be interpreted as `NA` values by read.csv, read_xls, read_xlsx, or read.table
`extension`	Deprecated, use `filetype` instead
`block_name_header`	Deprecated, use `block_names_header` instead
`...`	Other arguments passed to read.csv, read_xls, read_xlsx, or read.table

Details

For metadata, `read_blocks` can handle an arbitrary number of additional pieces of information to extract from each blockcurve file as metadata. These pieces of information are specified as a named list of vectors where each vector is the c(row, column) where the information is to be pulled from in the input files.

This metadata is returned as the second list element of each blockcurve, e.g.:

[[1]] [1] "data" #1 [2] "metadata" [2][1] name #1

[2][2] date-time #1

[2][3] temp #1

[[2]] [1] "data" #2 [2] "metadata" [2][1] name #2

[2][2] date-time #2

[2][3] temp #2

...

Calling uninterleave on the output of read_blocks works on block data and the associated metadata because uninterleave operates on the highest level entries of the list (the [[1]] [[2]] level items), leaving the meta-data associated with the block data trans_block_to_wide integrates this metadata into the wide-shaped dataframe it produces

Value

A list where each entry is a list containing the block data frame followed by the block_names (or filenames, if block_names is not provided) and any specified metadata.

`read_tidys` *Read tidy-shaped files*

Description

A function that imports tidy-shaped files into R. Largely acts as a wrapper for read.csv, read_xls, read_xls, or read_xlsx, but can handle multiple files at once and has additional options for taking subsets of rows/columns rather than the entire file and for adding filename or run names as an added column in the output.

Usage

```
read_tidys(
  files,
  filetype = NULL,
  startrow = NULL,
  endrow = NULL,
  startcol = NULL,
  endcol = NULL,
  sheet = NULL,
  run_names = NULL,
  run_names_header = NULL,
  run_names_dot = FALSE,
  run_names_path = TRUE,
  run_names_ext = FALSE,
  na.strings = c("NA", ""),
  extension,
  names_to_col,
  ...
)
```

Arguments

`files`	A vector of filepaths (relative to current working directory) where each one is a tidy-shaped data file
`filetype`	(optional) the type(s) of the files. Options include: "csv", "xls", or "xlsx". "tbl" or "table" to use read.table to read the file, "csv2" to use read.csv2, "delim" to use read.delim, or "delim2" to use read.delim2. If none provided, `read_tidys` will infer filetype(s) from the extension(s) in `files`. When extension is not "csv", "xls", or "xlsx", will use "table".

`startrow, endrow, startcol, endcol`	(optional) the rows and columns where the data are located in `files`. Can be a vector or list the same length as `files`, or a single value that applies to all `files`. Values can be numeric or a string that will be automatically converted to numeric by from_excel. If not provided, data is presumed to begin on the first row and column of the file(s) and end on the last row and column of the file(s).
`sheet`	The sheet of the input files where data is located (if input files are .xls or .xlsx). If not specified defaults to the first
`run_names`	Names to give the tidy files read in. By default uses the file names if not specified. These names may be added to the resulting data frame depending on the value of the `names_to_col` argument
`run_names_header`	Should the run names (provided in `run_names` or inferred from `files`) be added as a column to the output? If `run_names_header` is TRUE, they will be added with the column name "run_name" If `run_names_header` is FALSE, they will not be added. If `run_names_header` is a string, they will be added and the column name will be the string specified for `run_names_header`. If `run_names_header` is NULL, they only will be added if there are multiple tidy data.frames being read. In which case, the column name will be "run_name"
`run_names_dot`	If run_names are inferred from filenames, should the leading ’./’ (if any) be retained
`run_names_path`	If run_names are inferred from filenames, should the path (if any) be retained
`run_names_ext`	If run_names are inferred from filenames, should the file extension (if any) be retained
`na.strings`	A character vector of strings which are to be interpreted as `NA` values by read.csv, read_xls, read_xlsx, or read.table
`extension`	Deprecated, use `filetype` instead
`names_to_col`	Deprecated, use `run_names_header` instead
`...`	Other arguments passed to read.csv, read_xls, read_xlsx, or read.table sheet

Details

`startrow`, `endrow`, `startcol`, `endcol`, `sheet` and `filetype` can either be a single value that applies for all files or vectors or lists the same length as `files`

Note that the startrow is always assumed to be a header

Value

A dataframe containing a single tidy data.frame, or A list of tidy-shaped data.frames named by filename

---

`read_wides` *Read wides*

Description

A function that imports widemeasures in files into the R environment

Usage

```
read_wides(
  files,
  filetype = NULL,
  startrow = NULL,
  endrow = NULL,
  startcol = NULL,
  endcol = NULL,
  header = TRUE,
  sheet = NULL,
  run_names = NULL,
  run_names_header = "file",
  run_names_dot = FALSE,
  run_names_path = TRUE,
  run_names_ext = FALSE,
  metadata = NULL,
  na.strings = c("NA", ""),
  extension,
  names_to_col,
  ...
)
```

Arguments

`files`	A vector of filepaths (relative to current working directory) where each one is a widemeasures set of data
`filetype`	(optional) the type(s) of the files. Options include: "csv", "xls", or "xlsx". "tbl" or "table" to use read.table to read the file, "csv2" to use read.csv2, "delim" to use read.delim, or "delim2" to use read.delim2. If none provided, `read_wides` will infer filetype(s) from the extension(s) in `files`. When extension is not "csv", "xls", or "xlsx", will use "table".
`startrow`, `endrow`, `startcol`, `endcol`	(optional) the rows and columns where the data are located in `files`. Can be a vector or list the same length as `files`, or a single value that applies to all `files`. Values can be numeric or a string that will be automatically converted to numeric by from_excel. If not provided, data is presumed to begin on the first row and column of the file(s) and end on the last row and column of the file(s).

`header`	logical for whether there is a header in the data. If FALSE columns are simply numbered. If TRUE, the first row of the data (`startrow` if specified) is used as the column names
`sheet`	The sheet of the input files where data is located (if input files are .xls or .xlsx). If not specified defaults to the first sheet
`run_names`	Names to give the widemeasures read in. By default uses the file names if not specified
`run_names_header`	Should the run names (provided in `run_names` or inferred from `files`) be added as a column to the widemeasures? If `run_names_header` is NULL, they will not be. If `run_names_header` is a string, that string will be the column header for the column where the names will be stored
`run_names_dot`	If run_names are inferred from filenames, should the leading './' (if any) be retained
`run_names_path`	If run_names are inferred from filenames, should the path (if any) be retained
`run_names_ext`	If run_names are inferred from filenames, should the file extension (if any) be retained
`metadata`	(optional) non-spectrophotometric data that should be associated with each read widemeasures. A named list where each item in the list is either: a vector of length 2, or a list containing two vectors. In the former case, each vector should provide the row and column where the metadata is located in all of the blockmeasures input files. In the latter case, the first vector should provide the rows where the metadata is located in each of the corresponding input files, and the second vector should provide the columns where the metadata is located in each of the corresponding input files. (This case is typically used when reading multiple blocks from a single file.)
`na.strings`	A character vector of strings which are to be interpreted as `NA` values by read.csv, read_xls, read_xlsx, or read.table
`extension`	Deprecated, use `filetype` instead
`names_to_col`	Deprecated, use `run_names_header` instead
`...`	Other arguments passed to read.csv, read_xls, read_xlsx, or read.table

Details

startrow, endrow, startcol, endcol, timecol, sheet and filetype can either be a single value that applies for all files or vectors or lists the same length as `files`,

Value

A dataframe containing a single widemeasures, or A list of widemeasures named by filename

---

`separate_tidy` *Separate a column into multiple columns*

Description

This function is primarily a wrapper for separate, which turns a single character column into multiple columns

Usage

```
separate_tidy(
  data,
  col,
  into = NULL,
  sep = "_",
  coerce_NA = TRUE,
  na.strings = "NA",
  message_inferred_into = TRUE,
  ...
)
```

Arguments

`data`	A data frame
`col`	Column name or position
`into`	A character vector of the new column names. Use `NA` to omit the variable in the output. If NULL, `separate_tidy` will attempt to infer the new column names by splitting the column name of `col`
`sep`	Separator between columns passed to separate: If character, `sep` is interpreted as a regular expression. If numeric, `sep` is interpreted as character positions to split at. Positive values start at 1 at the far-left of the string; negative values start at -1 at the far-right of the string. The length of `sep` should be one less than `into`
`coerce_NA`	logical dictating if strings matching any of `na.strings` will be coerced into `NA` values after separating.
`na.strings`	A character vector of strings which are to be interpreted as `NA` values if `coerce_NA == TRUE`
`message_inferred_into`	logical whether column names for `into` should be printed in a message when inferred
`...`	Other arguments passed to separate

Value

A data frame containing new columns in the place of `col`

`smooth_data` *Smooth data*

Description

This function calls other functions to smooth growth curve data

Usage

```
smooth_data(
  ...,
  x = NULL,
  y = NULL,
  sm_method,
  subset_by = NULL,
  return_fitobject = FALSE,
  warn_ungrouped = TRUE,
  warn_gam_no_s = TRUE
)
```

Arguments

`...`	Arguments passed to loess, gam, moving_average, moving_median, or smooth.spline. Typically includes tuning parameter(s), which in some cases are required. See Details for more information.
`x`	An (often optional) vector of predictor values to smooth along (e.g. time)
`y`	A vector of response values to be smoothed (e.g. density). If NULL, `formula` and `data` *must* be provided via `...`
`sm_method`	Argument specifying which smoothing method should be used to smooth data. Options include "moving-average", "moving-median", "loess", "gam", and "smooth.spline".
`subset_by`	An optional vector as long as `y`. `y` will be split by the unique values of this vector and the smoothed data for each group will be calculated independently of the others. This provides an internally-implemented approach similar to group_by and mutate
`return_fitobject`	logical whether entire object returned by fitting function should be returned. If FALSE, just fitted values are returned.
`warn_ungrouped`	logical whether warning should be issued when `smooth_data` is being called on ungrouped data and `subset_by = NULL`.
`warn_gam_no_s`	logical whether warning should be issued when gam is used without `s()` in the formula.

Details

For moving_average and moving_median, passing `window_width` or `window_width_n` via `...` is required. `window_width` sets the width of the moving window in units of `x`, while `window_width_n` sets the width in units of number of data points. Larger values for either will produce more "smoothed" data.

For loess, the `span` argument sets the fraction of data points that should be included in each calculation. It's typically best to specify, since the default of 0.75 is often too large for growth curves data. Larger values of `span` will produce more more "smoothed" data

For `gam`, both arguments to gam and s can be provided via `...`. Most frequently, the `k` argument to s sets the number of "knots" the spline-fitting can use. Smaller values will be more "smoothed".

When using `sm_method = "gam"`, advanced users may also modify other parameters of `s()`, including the smoothing basis `bs`. These bases can be thin plate (`bs = "tp"`, the default), cubic regressions (`bs = "cr"`), or many other options (see s). I recommend leaving the default thin plate regressions, whose main drawback is that they are computationally intensive to calculate. For growth curves data, this is unlikely to be relevant.

As an alternative to passing `y`, for more advanced needs with loess or gam, `formula` and `data` can be passed to `smooth_data` via the `...` argument (in lieu of `y`).

In this case, the formula should specify the response (e.g. density) and predictors. For `gam` smoothing, the formula should typically be of the format: y ~ s(x), which uses s to smooth the data. The data argument should be a `data.frame` containing the variables in the formula. In such cases, `subset_by` can still be specified as a vector with length `nrow(data)`

Value

If `return_fitobject == FALSE`:

A vector, the same length as `y`, with the now-smoothed y values

If `return_fitobject == TRUE`:

A list the same length as unique(subset_by) where each element is an object of the same class as returned by the smoothing method (typically a named list-like object)

`solve_linear` *Return missing information about a line*

Description

Takes a set of inputs that is sufficient information to infer a line and then returns information not provided (either the slope, an x point on the line, or a y point on the line)

Usage

```
solve_linear(
  x1,
  y1,
  x2 = NULL,
```

```
  y2 = NULL,
  x3 = NULL,
  y3 = NULL,
  m = NULL,
  named = TRUE
)
```

Arguments

`x1, y1`	A point on the line
`x2, y2`	An additional point on the line
`x3, y3`	An additional point on the line
`m`	The slope of the line
`named`	logical indicating whether the returned value(s) should be named according to what they are (m, x2, y2, x3, or y3)

Details

Note that there is no requirement that `x1` < `x2` < `x3`: the points can be in any order along the line.

`solve_linear` works with vectors of all inputs to solve multiple lines at once, where the `i`th element of each argument corresponds to the `i`th output. Note that all lines must be missing the same information. Input vectors will be recycled as necessary.

Value

A named vector with the missing information from the line:

If `m` and `x2` are provided, `y2` will be returned

If `m` and `y2` are provided, `x2` will be returned

If `x2` and `y2` are provided, but neither `x3` nor `y3` are provided, `m` will be returned

If `x2` and `y2` are provided and one of `x3` or `y3` are provided, the other (`y3` or `x3`) will be returned

---

`ThresholdFunctions` *Find point(s) when a numeric vector crosses some threshold*

Description

These functions take a vector of `y` values and identify points where the `y` values cross some `threshold` `y` value.

Usage

```
find_threshold_crosses(
  y,
  x = NULL,
  threshold,
  return = "index",
  return_rising = TRUE,
  return_falling = TRUE,
  return_endpoints = TRUE,
  subset = NULL,
  na.rm = TRUE
)

first_below(
  y,
  x = NULL,
  threshold,
  return = "index",
  return_endpoints = TRUE,
  ...
)

first_above(
  y,
  x = NULL,
  threshold,
  return = "index",
  return_endpoints = TRUE,
  ...
)
```

Arguments

`y`	Numeric vector of y values in which to identify threshold crossing event(s)
`x`	Optional numeric vector of corresponding x values
`threshold`	Threshold y value of interest
`return`	One of `c("index", "x")`, determining whether the function will return the `index` or `x` value associated with the threshold-crossing event. If `index`, it will refer to the data point immediately after the crossing event. If `x`, it will use linear interpolation and the data points immediately before and after the threshold-crossing to return the exact `x` value when the threshold crossing occurred
`return_rising`	logical for whether crossing events where `y` rises above `threshold` should be returned
`return_falling`	logical for whether crossing events where `y` falls below `threshold` should be returned

`return_endpoints`	logical for whether startpoint should be returned when the startpoint is above `threshold` and `return_rising = TRUE`, or when the startpoint is below `threshold` and `return_falling = TRUE`
`subset`	A vector of logical values indicating which x and y values should be included (TRUE) or excluded (FALSE). If `return = "index"`, index will be for the whole vector and not the subset of the vector
`na.rm`	logical whether NA's should be removed before analyzing. If `return = 'index'`, indices will refer to the original `y` vector *including* `NA` values
`...`	(for `first_above` and `first_below`) other arguments to pass to `find_threshold_crosses`

Value

`find_threshold_crosses` returns a vector corresponding to all the threshold crossings.

`first_above` returns only the first time the `y` values rise above the threshold, so is a shortcut for `find_threshold_crosses(return_rising = TRUE, return_falling = FALSE)[1]`

`first_below` returns only the first time the `y` values fall below the threshold, so is a shortcut for `find_threshold_crosses(return_rising = FALSE, return_falling = TRUE)[1]`

If `return = "index"`, the returned value(s) are the indices immediately following threshold crossing(s)

If `return = "x"`, the returned value(s) are the x value(s) corresponding to threshold crossing(s)

If no threshold-crossings are detected that meet the criteria, will return `NA`

`to_excel` *A function that converts numbers into base-26 Excel-style letters*

Description

A function that converts numbers into base-26 Excel-style letters

Usage

```
to_excel(x)
```

Arguments

`x`	A vector of numbers in base-10

Value

A vector of letters in Excel-style base-26 format

`train_smooth_data` *Test efficacy of different smoothing parameters*

Description

This function is based on train, which runs models (in our case different smoothing algorithms) on data across different parameter values (in our case different smoothness parameters).

Usage

```
train_smooth_data(
  ...,
  x = NULL,
  y = NULL,
  sm_method,
  preProcess = NULL,
  weights = NULL,
  metric = ifelse(is.factor(y), "Accuracy", "RMSE"),
  maximize = ifelse(metric %in% c("RMSE", "logLoss", "MAE", "logLoss"), FALSE, TRUE),
  trControl = caret::trainControl(method = "cv"),
  tuneGrid = NULL,
  tuneLength = ifelse(trControl$method == "none", 1, 3),
  return_trainobject = FALSE
)
```

Arguments

`...`	Arguments passed to smooth_data. These arguments cannot overlap with any of those to be tuned.
`x`	A vector of predictor values to smooth along (e.g. time)
`y`	A vector of response values to be smoothed (e.g. density).
`sm_method`	Argument specifying which smoothing method should be used. Options include "moving-average", "moving-median", "loess", "gam", and "smooth.spline".
`preProcess`	A string vector that defines a pre-processing of the predictor data. The default is no pre-processing. See train for more details.
`weights`	A numeric vector of case weights. This argument currently does not affect any `train_smooth_data` models.
`metric`	A string that specifies what summary metric will be used to select the optimal model. By default, possible values are "RMSE" and "Rsquared" for regression. See train for more details.
`maximize`	A logical: should the metric be maximized or minimized?
`trControl`	A list of values that define how this function acts. See train and trainControl for more details.

`tuneGrid`	A data frame with possible tuning values, or a named list containing vectors with possible tuning values. If a data frame, the columns should be named the same as the tuning parameters. If a list, the elements of the list should be named the same as the tuning parameters. If a list, expand.grid will be used to make all possible combinations of tuning parameter values.
`tuneLength`	An integer denoting the amount of granularity in the tuning parameter grid. By default, this argument is the number of levels for each tuning parameter that should be generated. If `trControl` has the option `search = "random"`, this is the maximum number of tuning parameter combinations that will be generated by the random search. (NOTE: If given, this argument must be named.)
`return_trainobject`	A logical indicating whether the entire result of train should be returned, or only the `results` element.

Details

See train for more information.

The default method is k-fold cross-validation (`trControl = caret::trainControl(method = "cv")`).

For less variable, but more computationally costly, cross-validation, users may choose to increase the number of folds. This can be done by altering the `number` argument in trainControl, or by setting `method = "LOOCV"` for leave one out cross-validation where the number of folds is equal to the number of data points.

For less variable, but more computationally costly, cross-validation, users may alternatively choose `method = "repeatedcv"` for repeated k-fold cross-validation.

For more control, advanced users may wish to call train directly, using makemethod_train_smooth_data to specify the `method` argument.

Value

If `return_trainobject = FALSE` (the default), a data frame with the values of all tuning parameter combinations and the training error rate for each combination (i.e. the `results` element of the output of train).

If `return_trainobject = TRUE`, the output of train

`trans_block_to_wide` *Transform blocks to wides*

Description

Takes blocks and returns them in a wide format

Usage

```
trans_block_to_wide(
  blocks,
  wellnames_sep = "",
  nested_metadata = NULL,
  colnames_first = FALSE
)
```

Arguments

`blocks`	Blocks, either a single data.frame or a list of data.frames
`wellnames_sep`	String to use as separator for well names between rowname and column name (ordered according to `colnames_first`
`nested_metadata`	A logical indicating the existence of nested metadata in the `blockmeasures` list, e.g. as is typically output by read_blocks. If NULL, will attempt to infer existence of nested metadata
`colnames_first`	In the wellnames created by paste-ing the rownames and column names, should the column names come first

Value

A single widemeasures data.frame

`trans_wide_to_tidy` *Pivot widemeasures longer*

Description

Essentially a wrapper for tidyr::pivot_longer that works on both a single widemeasures as well as a list of widemeasures

Usage

```
trans_wide_to_tidy(
  wides,
  data_cols = NA,
  id_cols = NA,
  names_to = "Well",
  values_to = "Measurements",
  values_to_numeric = TRUE,
  ...
)
```

Arguments

`wides`	A single widemeasures data.frame, or a list of widemeasures data.frame's
`data_cols`, `id_cols`	Specifies which columns have data vs are ID's (in pivot_longer parlance). Each can be a single vector (which will be applied for all widemeasures) or a list of vectors, with each vector corresponding to the same-index widemeasure in `widemeasures` Entries that are NA in the list will not be used If neither data_cols nor id_cols are specified, user must provide arguments to tidyr::pivot_longer via `...` for at least the `cols` argument and these arguments provided via `...` will be used for all `widemeasures` data.frame's
`names_to`, `values_to`	Specifies the output column names created by tidyr::pivot_longer. Each can be provided as vectors the same length as `widemeasures` Note that if neither data_cols nor id_cols
`values_to_numeric`	logical indicating whether values will be coerced to numeric. See below for when this may be overridden by arguments passed in `...`
`...`	Other functions to be passed to pivot_longer Note that including values_transform here will override the behavior of values_to_numeric

Value

Pivoted longer data.frame (if `widemeasures` is a single data.frame) or list of pivoted longer data.frame's (if `widemeasures` is a list of data.frame's)

`uninterleave` *Uninterleave list*

Description

Takes a list that is actually interleaved elements from multiple sources and uninterleaves them into the separate sources. For instance, a list of blockmeasures that actually corresponds to two different plates can be split into two lists, each of the blockmeasures corresponding to a single plate. Uninterleave assumes that the desired sub-groups are perfectly interleaved in the input (e.g. items belong to sub-groups 1,2,3,1,2,3,...)

Usage

```
uninterleave(interleaved_list, n)
```

Arguments

`interleaved_list`	A list of R objects
`n`	How many output sub lists there should be (i.e. how many groups the interleaved list should be divided into)

Value

A list of lists of R objects

`WhichMinMaxGC` *Where is the Min() or Max() or first TRUE or FALSE?*

Description

Determines the location, i.e. index, of the (first) minimum or maximum of a numeric (or logical) vector.

Usage

```
which_min_gc(x, empty_NA = TRUE)

which_max_gc(x, empty_NA = TRUE)
```

Arguments

`x`	numeric (logical, integer, or double) vector or an `R` object for which the internal coercion to double works whose min or max is searched for.
`empty_NA`	logical, indicating if an empty value should be returned as `NA` (the default) or as `integer(0)` (the same as `which.min` and `which.max`).

Details

These functions are wrappers for `which.min` and `which.max`, with the additional argument `empty_NA`.

Value

If `empty_NA = FALSE`, identical to `which.min` or `which.max`

If `empty_NA = TRUE`, identical to `which.min` or `which.max` except that, in cases where `which.min` or `which.max` would return `integer(0)`, `which_min_gc` and `which_max_gc` return `NA`

`write_blocks` *Write block designs to csv*

Description

This function writes block-shaped lists (as created by read_blocks or make_design) to csv files, including both `data` and `metadata` in a variety of output formats

Usage

```
write_blocks(
  blocks,
  file,
  output_format = "multiple",
  block_name_location = NULL,
  block_name_header = "block_name",
  paste_sep = "_",
  filename_sep = "_",
  na = "",
  dir = NULL,
  ...
)
```

Arguments

`blocks`	list of block-shaped data to be written to file
`file`	`NULL`, a character string naming a file to write to, or a vector of character strings naming files to write to. A file name is required when `output_format = "single"` A file name can be specified when `output_format = "pasted"`, or `file` can be set to `NULL` as long as `block_name_location = "filename"` (where pasted `block_name` metadata will be used for the file name) File names can be specified when `output_format = "multiple"`, or `file` can be set to `NULL` as long as `block_name_location = "filename"` (where the `block_name` metadata will be used for the file names)
`output_format`	One of "single", "pasted", "multiple". "single" will write all blocks into a single csv file, with an empty row between successive blocks. "pasted" will paste all blocks together using a `paste_sep`, and then write that now-pasted block to a single csv file. "multiple" will write each block to its own csv file.
`block_name_location`	Either `NULL`, 'filename' or 'file'. If `NULL`, `block_name_location` will be automatically selected based on `output_format`. For `output_format = 'single'` and `output_format = 'pasted'`, `block_name_location`

	defaults to 'file'. For `output_format = 'multiple'`, `block_name_location` defaults to 'filename' If 'filename', the `block_name` metadata will be used as the output file name(s) when no file name(s) are provided, or appended to file name(s) when they have been provided. If 'file', the `block_name` metadata will be included as a row in the output file.
`block_name_header`	The name of the field containing the `block_names`
`paste_sep`	When `output_format = 'pasted'`, what character will be used to paste together blocks.
`filename_sep`	What character will be used to paste together filenames when block_name_location = 'filename'.
`na`	The string to use for missing values in the data.
`dir`	The directory that file(s) will be written into. When `dir = NULL`, writes to the current working directory. (Can only be used when `file = NULL`)
`...`	Other arguments passed to write.table

Value

Nothing, but R objects are written to files

Index